Deep Scattering End-to-End Speech Recognition

# 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 methods to form a hybrid system. This also requires that various components of the speech recognizer are 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 all the sub-components of speech recognition as a single process. In this 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 is traded for higher internal model complexity and more computational resources needed to train the end-to-end models.

This research focuses on taking advantage of 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 lightweight, wavelet-based architecture added any new features that are beneficial for low resource Speech recognition.

The research showed that it is possible to use this compact strategy and improve upon speech features required for speech pattern recognition by deploying deep scattering network features with higher dimensional vectors when compared to traditional speech features. 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.

# 

# Chapter 1: Introduction

Automatic Speech Recognition 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 \cite{yu2016automatic}. 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 7000 human languages in the world \citep{besacier2014automatic} 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 \citep{Rosenberg2017end,besacier2014introduction}. 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 \citep{goldman2011easyalign,stan2016alisa}. Research, therefore, geared towards alternative approaches towards developing is ASR systems that are reproducible across languages lacking the resources required to build robust systems is apt.

## 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 \citep{becchetti1998}. 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.

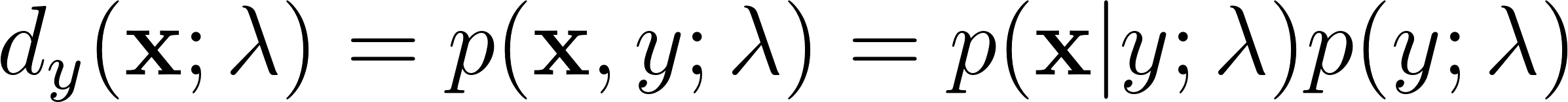
It has been acknowledged in the research community \citep{2015watanabe,deng2013machine} that work being done in Machine Learning has enhanced the research of automatic speech recognition. Similarly any progress made in ASR usually constitute contributions to enhancements made in the Machine Learning field. This also may be attributed to the fact that speech recognition in itself is a sequence pattern recognition problem subclass of machine learning. Therefore techniques within speech recognition could be applied generally to sequence pattern recognition problems at large.

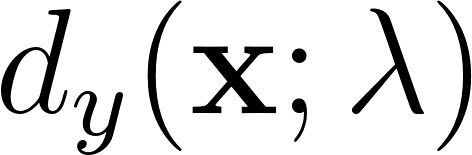
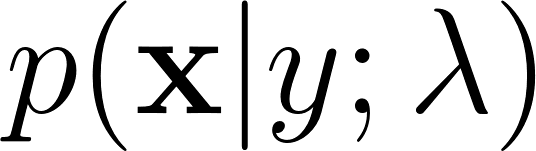
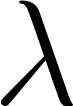
The two main approaches to Machine Learning problems historically involve two methods rooted in statistical science. These approaches are generative and discriminative models. From a computing science perspective, the generative approach is a brute-force approach while the discriminative model uses a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

## Generative-Discriminative Speech Models disambiguation

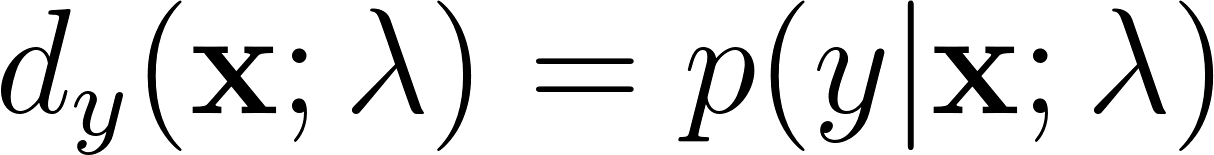
In chapter \ref{ch2litrev}, the Hidden Markov Model (HMM) is examined as a powerful and major driver behind generative modelling of sequential data like speech. Generative models are data-sensitive models because they are derived from the data by accumulating as many different features which can be seen 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 \citep{lasserre2006principled}. 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.

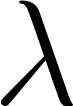
Ultimately, generative models for Machine Learning learning can be interpreted mathematically as a joint distribution that produces the highest likelihood of outputs and inputs based on a predefined decision function. The outputs for speech recognition being the sequence of words and the inputs for speech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

[](https://www.codecogs.com/eqnedit.php?latex=d_y(%5Cmathbf%7Bx%7D%3B%5Clambda)%3Dp(%5Cmathbf%7Bx%7D%2Cy%3B%5Clambda)%3Dp(%5Cmathbf%7Bx%7D%7Cy%3B%5Clambda)p(y%3B%5Clambda)%0)

where [](https://www.codecogs.com/eqnedit.php?latex=d_y(%5Cmathbf%7Bx%7D%3B%5Clambda)%0) is the decision function of [](https://www.codecogs.com/eqnedit.php?latex=y%0) for data labels [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bx%7D%0). This joint probability expression given as [](https://www.codecogs.com/eqnedit.php?latex=p(%5Cmathbf%7Bx%7D%7Cy%3B%5Clambda)%0) can also be expressed as the conditional probability product in equation (\ref{eqn1\_1}). In this equation, [](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%0) pre defines the nature of the distribution \citep{deng2013machine} referred to as model parameters.

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

[](https://www.codecogs.com/eqnedit.php?latex=d_y(%5Cmathbf%7Bx%7D%3B%5Clambda)%3Dp(y%7C%5Cmathbf%7Bx%7D%3B%5Clambda)%0)

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 ([](https://www.codecogs.com/eqnedit.php?latex=%5Clambda%0)) in equation (\ref{eqn1\_1}) and (\ref{eqn1\_2}) \citep{gales2012structured}.

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 corrupted 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 \ref{ASRMLP} to the next. The theme of pattern recognition through arbitrary layers of complexity is reinforced in the notion of deep learning defined in \cite{deng2014deep} 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 \citep{hannun2014deep} with modest results without a language model. However, applying a language model was observed to serve as a correction factor further improving recognition results \citep{hannun2014deep}.

## 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. \cite{saon2015ibm} recommends around 2000 hours of transcribed speech data for robust speech recognition system. As is covered in the next chapter, for new languages, which are low in training data such as transcribed speech, there are various strategies devised for low resource speech recognition. \cite{besacier2014automatic} 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 \cite{besacier2014automatic} 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 looked at 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 a 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.

## 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” \citep{wakirike}.

Although the population of the Wakirike community from a 1995 report \citep{ethnologue} 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) \citep{ethnologue}. 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 has 668,522 words.

## Research Aims and Objectives

In this research we develop speech processing models and language models which are of interest to new and low resource languages. In particular, we develop a character based language model based on Gated Recurrent Units (GRU) for the Wakirike language and an end-to-end bi-directional recurrent neural network (Bi-RNN) speech model for the English language. The aim of this research therefore is the development of end-to-end speech recognition systems ASR systems in a resource conservative manner.

The research objectives were as follows:

1. Discover fundamental tasks relating to Language learning in particular speech recognition.
2. Discover criteria for creating ASR sub-systems for new languages.
3. Build robust ASR systems using methods that integrate low-resource concerns.
4. Evaluate end-to-end ASR systems developed as resource-friendly ASR systems.

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

1. Reduction in the overall time taken to develop speech models
2. Optimisation of sub-tasks and training architecture within the ASR pipeline
3. Reduction of data required to train speech models
4. Make efficient use of training parallelism
5. Obtain comparable state of the art performance of ASR
6. Induce model simplicity thereby reducing training and development time without compromising performance

Furthermore, following the Interspeech 2015 Zero Resource Speech Challenge \citep{versteegh2015zero}, this research also fulfilled the objectives of modelling speech at subword, word and syntax level. The Zero Resource Speech Challenge is inspired from infants ability to construct acoustic and language models of speech in an end-to-end manner. At the word and syntax level this research develops a character-based language model that reinforces subword, word and syntax level speech model based on Character-Temporal-Classification CTC.

## Research Question

Considering the recent development of end-to-end systems facilitated by deep-learning and sequence modelling, can we combine the recent pattern recognition strides in deep scattering transform with an end-to-end sequential model that results in a simplified but robust speech recognition system which 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.

## Main Contributions to knowledge

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

## Limitation of the study

This study realises from the onset that it is not able to gather the data required to build ASR systems for the Wakirike language. 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. 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.

## 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 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 Units (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.

## Chapter 1 Summary

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

As this Chapter identifies, the ASR problem is traditionally a Machine Learning (ML) problem that models where 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 \cite{**besacier2014introduction**}, it becomes time-consuming and extremely laborious to develop new ASR models for speech recognition for these so-called ASR “low-resource” languages.

This Chapter also introduces the Wakirike language as a low resource language and the motivating language for this research. In addition, the various machine learning architectures used in this research for low resource speech recognition for the Wakirike and for English language are reviewed. In particular, Deep Neural Networks (DNNs) are highlighted as choice algorithms in speech recognition, and then, the Chapter goes on to describe the research novelty and the outline of this thesis report.

# 

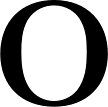
# Chapter 2: Previous Deep Scattering Research

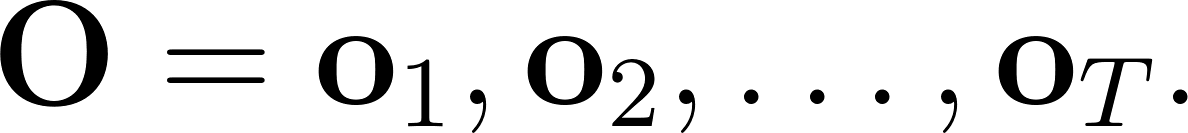
Current end-to-end ASR models based on CTC have been developed using standard MFCC features. The model developed in this work employs a deep scattering features which compared to MFCC possesses a greater number of features being of a higher dimension (152 compared to 39). These deep scattering vectors have been shown to perform well on music genre classification and TIMIT phone recognition \citep{[zeghidour2018end](https://arxiv.org/pdf/1806.07098)}.

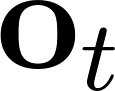
This chapter describes the transition from generative speech models to these discriminative end-to-end recurrent neural network models. Low speech recognition strategies are also discussed and the contribution to knowledge gained by using character-based discrimination as well as introducing deep scattering features to the bi-RNN speech model is brought to light.

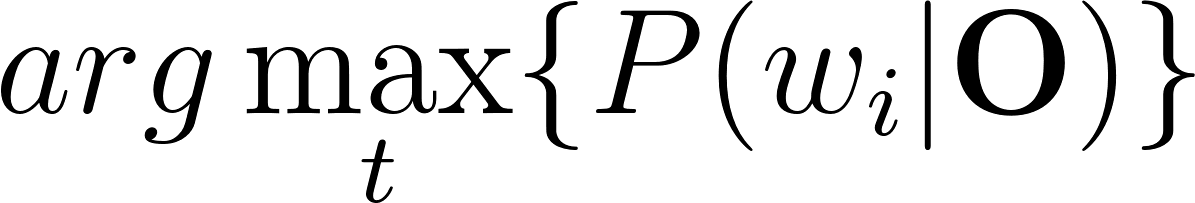
## Speech Recognition Overview

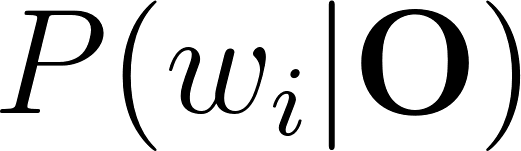
Computer speech recognition takes raw audio speech and converts it into a sequence of symbols. This can be considered as an analog to digital conversion as a continuous signal becomes discretised. The way this conversion is done is by breaking up the audio sequence into very small packets referred to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

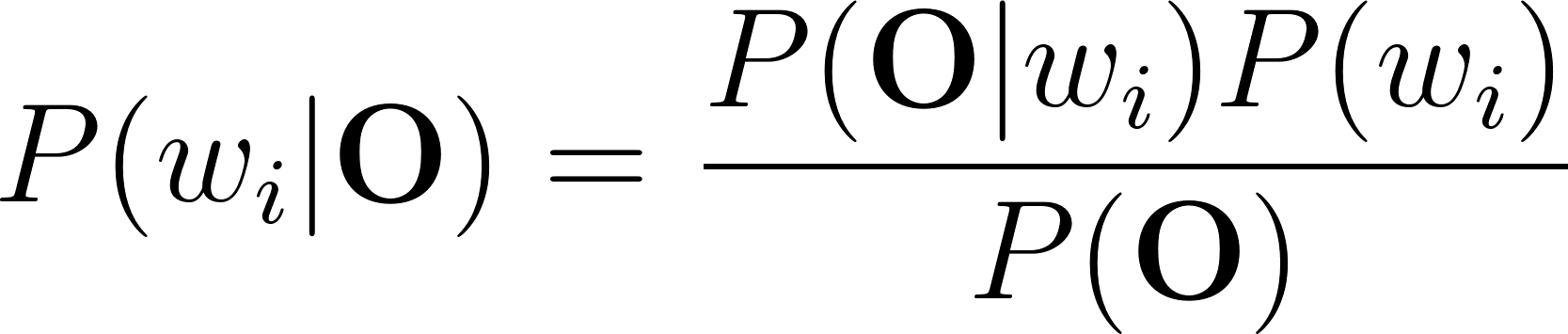
A statistical formulation \citep{young2002htk} for the speech recogniser follows given that each discretised output word in the audio speech signal is represented as a vector sequence of frame observations defined in the set [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BO%7D%0) such that

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BO%7D%3D%5Cmathbf%7Bo%7D_1%2C%5Cmathbf%7Bo%7D_2%2C%5Cdots%2C%5Cmathbf%7Bo%7D_T.%0) - - - (1.1)

Equation \ref{eqn\_1\_1\_sr\_inputs} says that, at each discrete time [](https://www.codecogs.com/eqnedit.php?latex=t%0), we have an observation [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bo%7D_t%0), which is, in itself is a vector in [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BR%7D%5ED%0). 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:

[](https://www.codecogs.com/eqnedit.php?latex=arg%5Cmax_t%5C%7BP(w_i%7C%5Cmathbf%7BO%7D)%5C%7D%0) - - - (1.2)

Section \ref{sec\_c2\_asr\_challenges} outline some challenges of speech recognition which makes analysis of [](https://www.codecogs.com/eqnedit.php?latex=P(w_i%7C%5Cmathbf%7BO%7D)%0) no trivial task. The divide and conquer (generative) strategy therefore employed uses Bayes formulation to simplify the problem. Accordingly, the argument that maximises the prior probability of an audio sequence given a particular word multiplied by the 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

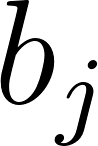
[](https://www.codecogs.com/eqnedit.php?latex=P(w_i%7C%5Cmathbf%7BO%7D)%3D%5Cfrac%7BP(%5Cmathbf%7BO%7D%7Cw_i)P(w_i)%7D%7BP(%5Cmathbf%7BO%7D)%7D%0) - - - (1.3)

That is, according to Bayes’ rule, the posterior probability is obtained by multiplying a certain likelihood probability by a prior probability. The likelihood in this case, [](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7BO%7D%7Cw_i)%0), 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.

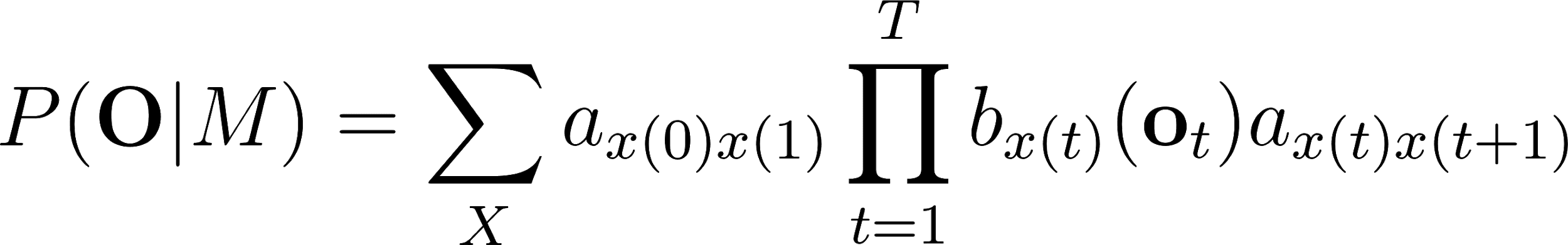
The second parameter in the speech model interpreted from Bayes' formula, is the prior probability of a given word. This aspect of the model is the language model is reviewed in section \ref{sec\_lrlm}.

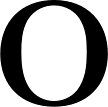
### HMM-based Generative speech model

A HMM represents a finite state machine where a process transits a sequence of states from a set of fixed states \citep{gales2008application, young2002htk}. 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 \ref{fig\_2\_1\_hmm} there are two important probabilities. The first is the state transition probability given by [](https://www.codecogs.com/eqnedit.php?latex=a_%7Bij%7D%0) this is the probability to move from state [](https://www.codecogs.com/eqnedit.php?latex=i%0) to state [](https://www.codecogs.com/eqnedit.php?latex=j%0). The second probability [](https://www.codecogs.com/eqnedit.php?latex=b_j%0) is the probability that an output observation is emitted when in a particular state.

Given that [](https://www.codecogs.com/eqnedit.php?latex=X%0) represents the sequence of states transitioned by a process, a HMM defines the joint probability of [](https://www.codecogs.com/eqnedit.php?latex=X%0) and the output probabilities given the HMM in the following representation:

[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7BO%7D%7CM)%3D%5Csum_Xa_%7Bx(0)x(1)%7D%5Cprod_%7Bt%3D1%7D%5ETb_%7Bx(t)%7D(%5Cmathbf%7Bo%7D_t)a_%7Bx(t)x(t%2B1)%7D%0)

Where [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BO%7D%0), are the output observations and [](https://www.codecogs.com/eqnedit.php?latex=M%0) is the HMM. Generally speaking, the HMM formulation presents 3 distinct challenges. The first is the likelihood of a sequence of observations given in equation \ref{eqn\_2\_4\_hmm} 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.

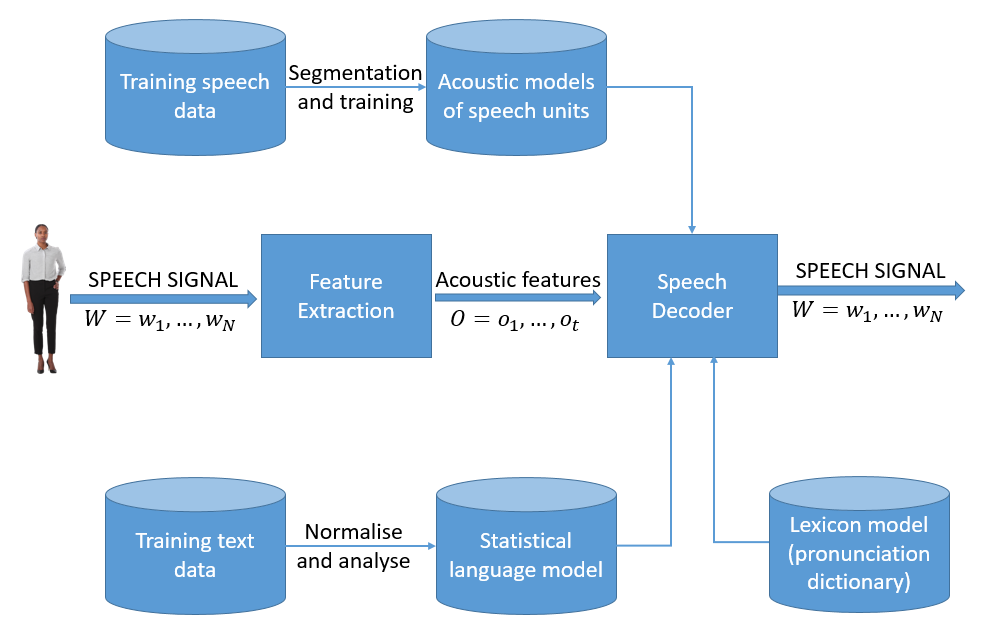
### Challenges of Speech Recognition

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 \citep{young2002htk}.

### 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 \ref{fig\_2\_2\_asr\_pipeline}, 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 \cite{besacier2014automatic} enumerate the challenges for developing low resource ASR systems to include the fact that phonologies (or language sound systems) differ across languages, word segmentation problems, fuzzy grammatical structures, unwritten languages, lack of native speakers having technical skills and the multidisciplinary nature of ASR constitute impedance to ASR system building.

## Low Resource Speech Recognition

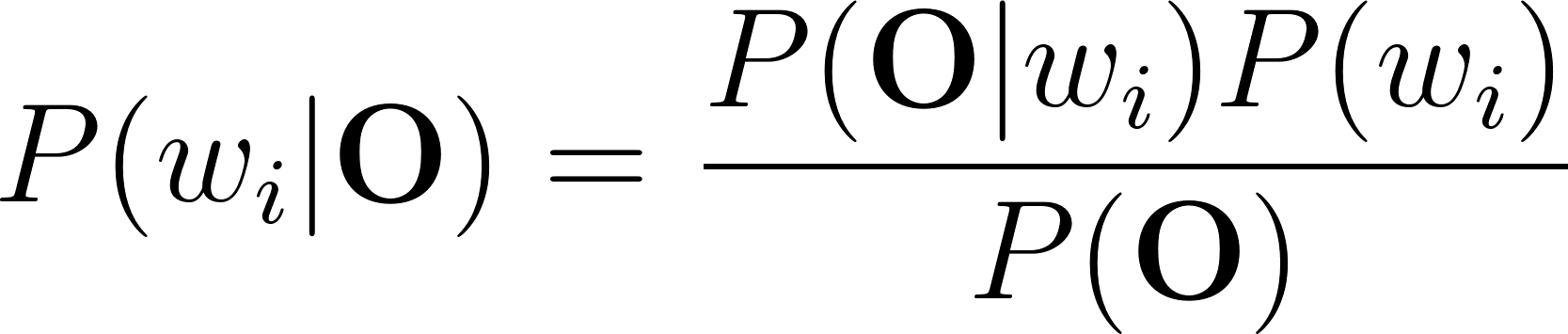
In this system building speech recognition research, the focus was on the development of a language model and an end-to-end speech model comparable in performance to state of the art speech recognition system consisting of an acoustic model and a language model. Low resource language and acoustic modelling 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\cite{Rosenberg2017end}.

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.

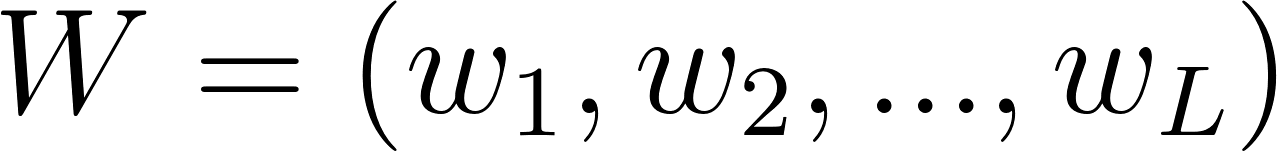
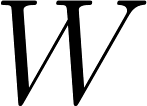
Transfer learning takes the inner representation of knowledge derived from a training algorithm used from one domain and applying this knowledge in a similar domain having different set of system parameters\citep{ramachandran2016unsupervised}. Early work of this nature for speech recognition is demonstrated in \citep{vu2013multilingual} 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 \citep{kunze2017transfer} is described.

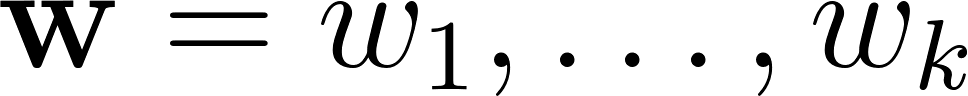
### Low Resource language modelling

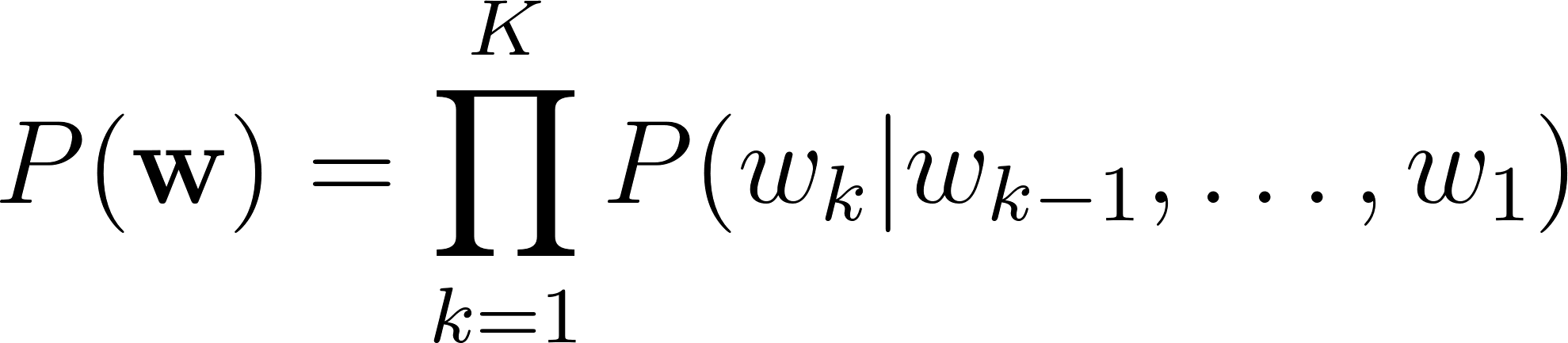
General language modelling is reviewed and then Low resource language modelling is discussed in this section. In section \ref{Ch\_2\_SROverview}, recall from equation \ref{eqn\_2\_3\_bayes\_sr}, the general speech model influenced by Bayes' theorem.

[](https://www.codecogs.com/eqnedit.php?latex=P(w_i%7C%5Cmathbf%7BO%7D)%3D%5Cfrac%7BP(%5Cmathbf%7BO%7D%7Cw_i)P(w_i)%7D%7BP(%5Cmathbf%7BO%7D)%7D%0)

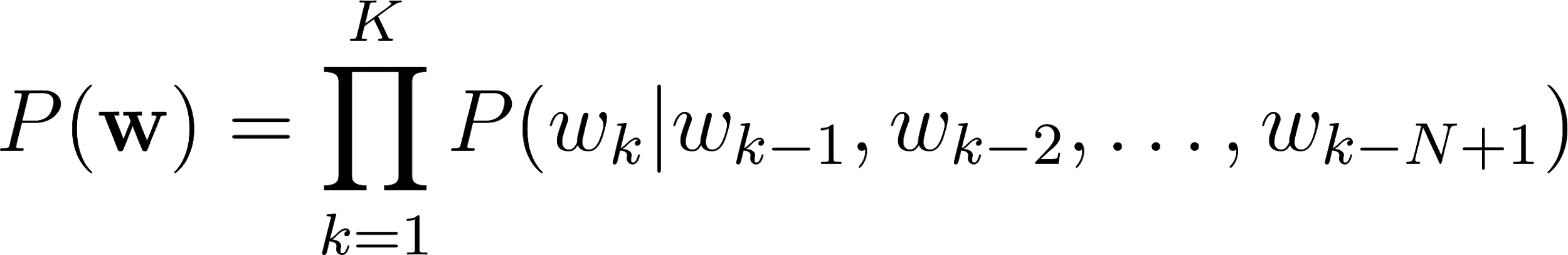
The speech recognition model is a product of an acoustic model (likelihood probability),[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7BO%7D%7Cw_i)%0) and the language model (prior probability),[](https://www.codecogs.com/eqnedit.php?latex=P(w_i)%0). The development of language models for speech recognition is discussed in \cite{juang2000automatic} and \cite{1996YoungA}.

Language modelling formulate rules that predict linguistic events and can be modelled in terms of discrete density [](https://www.codecogs.com/eqnedit.php?latex=P(W)%0), where [](https://www.codecogs.com/eqnedit.php?latex=W%3D(w_1%2C%20w_2%2C...%2C%20w_L)%0) is a word sequence. The density function [](https://www.codecogs.com/eqnedit.php?latex=P(W)%0) assigns a probability to a particular word sequence [](https://www.codecogs.com/eqnedit.php?latex=W%0). 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 modelling therefore, is an estimate for [](https://www.codecogs.com/eqnedit.php?latex=P(W)%0) from a given set of sentences, or corpus.

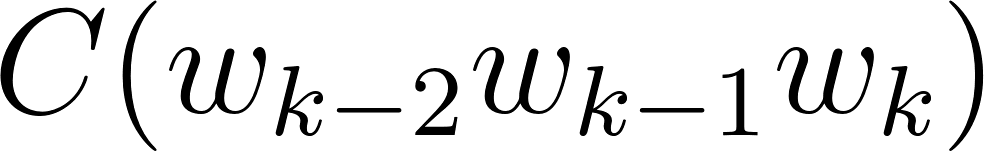
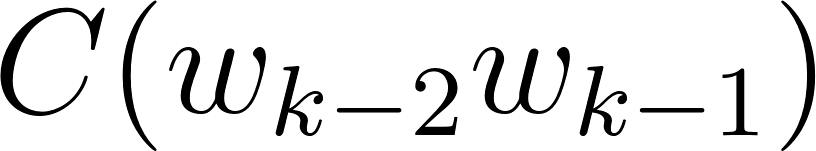
The prior probability of a word sequence [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bw%7D%3Dw_1%2C%5Cdots%2Cw_k%0) required in equation (2.2) is given by:

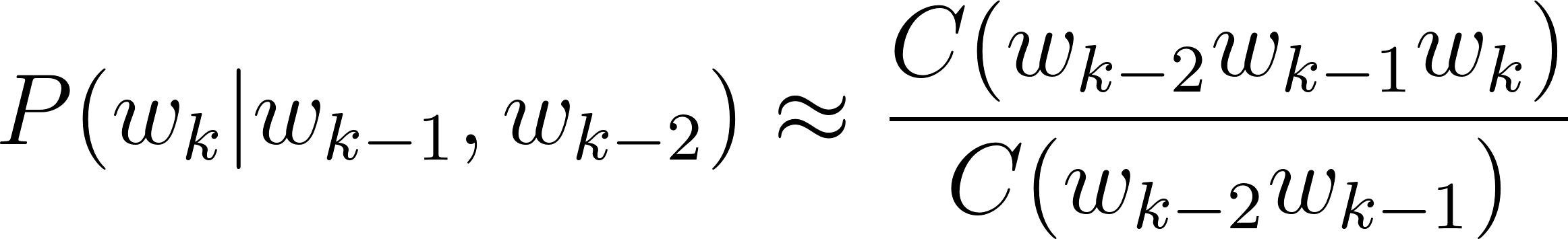
[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7Bw%7D)%3D%5Cprod_%7Bk%3D1%7D%5EKP(w_k%7Cw_%7Bk-1%7D%2C%5Cdots%2Cw_1)%0)

The N-gram model is formed by the conditioning of the word history in equation \ref{eqn\_c2\_lm01}. This therefore becomes:

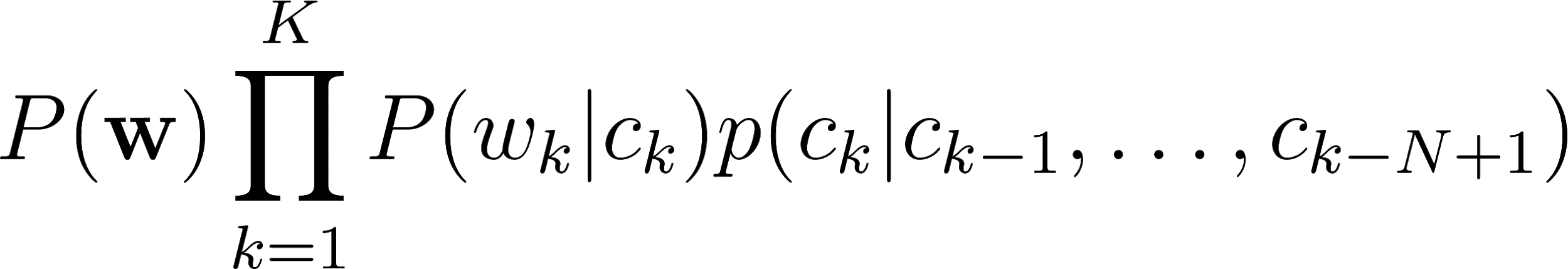
[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7Bw%7D)%3D%5Cprod_%7Bk%3D1%7D%5EKP(w_k%7Cw_%7Bk-1%7D%2Cw_%7Bk-2%7D%2C%5Cdots%2Cw_%7Bk-N%2B1%7D)%0)

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 [](https://www.codecogs.com/eqnedit.php?latex=C(w_%7Bk-2%7Dw_%7Bk-1%7Dw_k)%0) is the number of occurrences of the three words [](https://www.codecogs.com/eqnedit.php?latex=C(w_%7Bk-2%7Dw_%7Bk-1%7D)%0) is the count for [](https://www.codecogs.com/eqnedit.php?latex=w_%7Bk-2%7Dw_%7Bk-1%7Dw_k%0) then

[](https://www.codecogs.com/eqnedit.php?latex=P(w_k%7Cw_%7Bk-1%7D%2Cw_%7Bk-2%7D)%5Capprox%5Cfrac%7BC(w_%7Bk-2%7Dw_%7Bk-1%7Dw_k)%7D%7BC(w_%7Bk-2%7Dw_%7Bk-1%7D)%7D%0)

The major problem with maximum likelihood estimation scheme is data sparsity. This can be tackled by a combination of smoothing techniques involving discounting and backing-off. The alternative approach to robust language modelling is the so-called class based models \citep{Brown1992class,Kuhn1990cache} in which data sparsity is not so much an issue. Given that for every word [](https://www.codecogs.com/eqnedit.php?latex=w_k%0), there is a corresponding class [](https://www.codecogs.com/eqnedit.php?latex=c_k%0), then,

[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7Bw%7D)%5Cprod_%7Bk%3D1%7D%5EKP(w_k%7Cc_k)p(c_k%7Cc_%7Bk-1%7D%2C%5Cdots%2Cc_%7Bk-N%2B1%7D)%0)

In 2003, \cite{bengio2003neural} 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 \cite{mikolov2011empirical,sutskever2014sequence,luong2013better}, 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 \citep{pennington-etal-2014-glove}.

For the neural network implementations so far seen, a large amount of data is required due to the nature of words to have large vocabularies, even for medium-scale speech recognition applications. \cite{kim2016character} 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 \cite{xu2013cross}. 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 \cite{kim2016character}, however based on Gated Recurrent Units (GRU) RNNs \cite{}, 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.

### Low Resource Acoustic and speech modelling

Two transfer learning techniques for acoustic modelling investigated by \cite{povey2011subspace} and \cite{ghoshal2013multilingual} respectively include the sub-space Gaussian mixture models (SGMMs) 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 modeled using Gaussian mixture models or GMM are also called senones. \cite{povey2011subspace} 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 \cite{ghoshal2013multilingual} employed the use of deep neural networks, in particular deep belief networks \citep{bengio2007greedy}. Deep Belief Networks are pretrained, layer-wise stacked Restricted Boltzmann Machines (RBMs)\citep{smolensky1986information}. 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 \citep{grezl2008optimizing}.

### 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, to be quite data intensive\cite{hannun2014deep}. The next paragraphs introduce Deep-speech BiRNNs and the two strategies for tackling the data intensity drawback as related with low resource speech recognition.

### 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 \citep{woodland2000large} and the use of neural network precursors to initialise the HMM parameters \citep{mohamed2012acoustic}. 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, \cite{graves2014towards} 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 \cite{hannun2014first} follows an end-to-end Bi-directional Recurrent Neural Network (BiRNN) and CTC loss function \citep{graves2006connectionist}. 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.

### Speech Recognition on a low budget

In this section, a recent transfer learning speech model \citep{kunze2017transfer} that has some characteristics similar to the speech model developed in this thesis is reviewed. The end-to-end speech model described by \cite{kunze2017transfer} is based on that developed by \cite{collobert2016wav2letter} 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 \cite{kunze2017transfer} is the fact that the scattering network input is equivalent to the output of a light-weight convolutional neural network \cite{koushik2016understanding}. 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)\citep{@inproceedings{szegedy2015going}.

Introduction of the unique scattering network is discussed in the next section. It is worthy to note however that \cite{kunze2017transfer} uses a CNN network only while \citep{amodei2016deep} uses both RNN and CNN network. The speech model in this thesis uses a BiRNN model and combines an RNN model with the scattering layer which represents a light-weight low resource friendly pseudo enhanced CNN backing. What is meant by pseudo enhanced CNN backing is reserved for the next section, however, therefore, the proposed speech model in this thesis stands to gain from an enhanced but lightweight CNN combined with RNN learning.

### 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 \cite{ogunfunmi2015speech}.

In figure \ref{fig\_2\_2\_asr\_pipeline}, 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 (MFCCs) \citep{davis1980comparison} 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) \citep{hermansky1990perceptual} and RASTA (RelAtive SpecTrAl) \citep{hermansky1994rasta}.

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 \citep{anden2014deep}. 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 \citep{furui1986speaker} 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, \cite{anden2011multiscale} 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 \citep{mallat2016understanding}.

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 \citep{versteegh2015zero}. 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.

## Chapter 2 Summary

Chapter \ref{ch1\_intro} introduces the key terms Discriminative and Generative classification. In this Chapter, these two different classification mechanisms are compared and contrasted as they relate to speech recognition. The Hidden Markov Model (HMM) is considered as the major 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 and Models

This chapter describes the system building methodology \citep{nunamaker1990systems} as applied to deep recurrent architectures for speech recognition. As this approach involves theory building, system development, experimentation and observation, this chapter describes the procedures which were incorporated in order to achieve the aims and objectives of this research.

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

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

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

## Hypotheses and assumptions

This research makes the following assumptions:

1. The first hypothesis is that Software engineering systems are successfully developed using an incremental and iterative manner of increasing complexity.
2. 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.
3. End-to-end speech recognition has been made possible using recurrent neural networks (RNNs) and connectionist temporal classification (CTC) algorithm.
4. 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).
5. There is knowledge to be gained in the application of speech models to new languages.

## Speech processing software systems 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 codebases. The systems and platforms adapted for this research include the following:

1. CMUSphinx
2. Kaldi
3. Mozilla DeepSpeech
4. Scatternet toolbox
5. Matlab
6. Tensorflow
7. Choregraphe

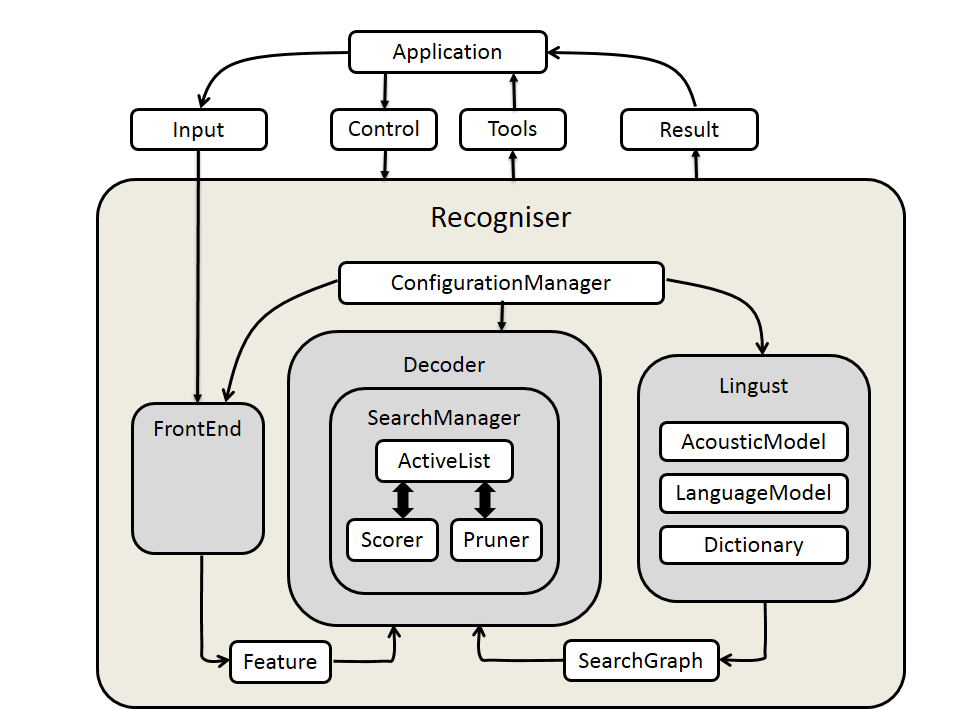
While the research sought to focus on speech models for the Wakirike language, several other sub systems were required for but development of the baseline models in addition to the final model the following system development steps were taken to arrive at the final output models:

1. Auto-correlation experiments
2. Experiments with Nao robot
3. CMUSphinx Digits speech recogniser
4. Digit speech recogniser using Kaldi
5. Python based speech alignment experiments
6. Sequence-to-sequence grapheme-to-phoneme (G2P) model
7. TensorFlow sequence-to-sequence character-to-diacritically-labelled-character model
8. GRU language model for Wakirike language based on TensorFlow
9. Bi-Directional LSTM-based end-to-end speech model

In the following paragraphs, 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.

### CMUSphinx

The CMU Sphinx recogniser system is illustrated in Figure \ref{fig\_c3\_sphinx}. 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.



*Figure \label{fig\_c3\_sphinx}: CMU Sphinx4 recogniser system*

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 Features 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 \citep{walker2004sphinx}.

#### FrontEnd Module

Being consistent with having a “pluggable” framework, CMU Sphinx4 has the ability that most of its components can be replaced and at runtime. This flexibility allows various implementations of the comprising components of the recogniser. Accordingly, the front end supports but is not limited to MFCC, PLP and LPC implementations. In addition, comprising modules within the various implementations include support for various signal processing utilities such as Hamming windows, discrete fourier transforms (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 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 (FSG), finite-state transducers and small N-gram language models. This module has three pluggable modules representing the **Dictionary**, **LanguageModel** and **AcousticModel**. The Dictionary comprises the pronunciation of all the words to be used in the **Decoder**. Sphinx-4 **Linguist** provides primary support for the CMU Pronouncing Dictionary (Carnegie Mellon University, 2016). The **SearchGraph** produced by the **Linguist** is capable of sharing parameters such as Gaussian mixtures, transition matrices and mixture weights and Sphinx-4 provides a single Acoustic model supporting acoustic models generated by the Sphinx-3 trainer. Depending on the memory architecture various implementations of the **Linguist** include the **FlatLinguist**, **DynamicFlatLinguist** and **LexTreeLinguist**. These will either create the **SearchGraph** entirely in memory or on demand. Finally, the **LanguageModel** supports a variety of formats such as **SimpleWorldListGramar** which as the name implies supports a simple word list. The **JSGFGramar** is a BNF-style platform-independent realisation of the Java Speech API Grammar format. **LMGrammar** produces a bigram model. **FSTGrammar** supports finite-state transducer ARPA FST grammar format. The **SimpleNGramModel** support N-gram model and the **LargeTriGramModel** is suited to optimise memory storage.

#### Decoder

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

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

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

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

#### Other modules

**ConfigurationManager** allows various module implementations to be combined in various ways. Finally, we illustrate how the **ConfigurationManager** creates Automatic Speech Recognition (ASR) experiments using the CMU-sphinx4 objects described above in the sample code from \citep{Lamere03thecmu}

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:/edu/cmu/sphinx/models/en-us/en-us");

configuration

.setDictionaryPath("resource:/edu/cmu/sphinx/models/en-us/cmudict-en-us.dict");

configuration

.setLanguageModelPath("resource:/edu/cmu/sphinx/models/en-us/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, \cite{walker2004sphinx} claims 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.

### Kaldi

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

#### Architecture

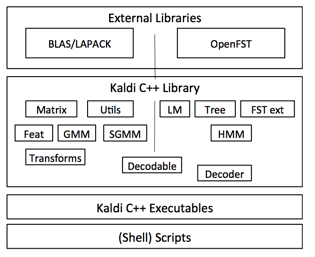
The component architecture of Kaldi is illustrated in the figure below. Modules can be divided into those that utilise the linear algebra libraries and those that use OpenFST. The decodable class forms the link between these two scopes. The rest of the modules lower down the hierarchy are based on modules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

Fig 1.

#### 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 Determinant Analysis (LDA), STC, Maximum Likelihood Linear Transformation (MLLT), Heterodastic Linear Discrimination Analysis (HLDA). These systems are made complete with various configuration parameters for fine tuning the individual models produced by them.

#### Acoustic Modelling

Full covariance as well as diagonal covariance GMM modelling is implemented in Kaldi. Efficient log-likelihoods are computed using simple dot products of mean times invariance and invariance covariance. The DiagGmm class is responsible for storing covariances 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 monophone model, tree-clustering algorithms automatically generate the decision tree.

#### Language Modelling and Decoding graphs

Using the 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.

### Mozilla DeepSpeech \label{sec\_c3\_moz}

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 \citep{hannun2014deep}. 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/linux 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 \citep{Heafield-estimate}.

#### 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 \ref{sec\_c3\_tf} 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 \ref{sec\_c3\_tf}.

#### Common Voice training data

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 percent 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 and the Wall Street Journal (WSJ)\citep{paul1992design}. 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:

1. Geometry - Defines the number of neurons in the hidden layers of the neural network
2. Cluster configuration - Parameters responsible for distributed training of the speech model across various nodes.
3. 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 overfitting where instead of learning inference features the data, the neural network memorizes the training data.
4. Adam optimiser - parameters for the Adam optimiser
5. Batching - set the number of batches during training.
6. Weight Initialisation - standard deviation coefficients for initialising weights
7. 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.
8. Exporting - Includes parameters for saving a saturated model for inference.
9. Reporting - Includes options for setting the log-level however reports are only sent to the standard console output.
10. 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.
11. 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.

### Matlab and ScatterNet toolbox

In this research, feature processing of audio files to obtain their deep scattering transforms was achieved using a MATLAB toolbox known as ScatNet \citep{anden2014scatnet}. The scatnet toolbox in general analyses time-series sampled analog signals and has been used successfully for music genre classification, texture and image classification \cite{anden2011multiscale, sifre2014rigid, sifre2013rotation}. In particular, the scattering transforms produced are signal processing layers of increasing width where each layer constitutes the convoution of a linear filter bank wavelet operator (**Wop**) with a non linear complex modulus.

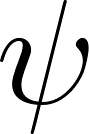
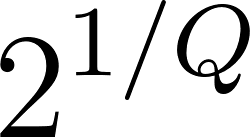
[](https://www.codecogs.com/eqnedit.php?latex=%7C%7C%5Ctext%7Bcomplex%20signal%7D%7C%5Cstar%20%5Cmathbf%7BWop%7D%7C%5Cstar(%5Ctext%7Blowpass%20filter%7D)%0) - - - (eqn)

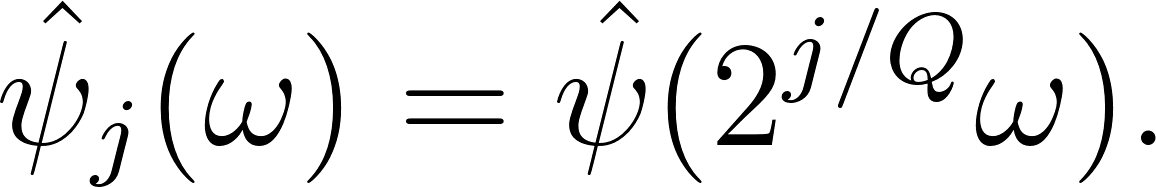
It is the scattering transforms of the audio files that were fed into the DeepSpeech model trainer discussed in Section \ref{sec\_c3\_moz}. The architecture of a scattering networks resembles a deep convolutional network in the sense that each subsequent layer is a mapping of all possible paths from the previous layer.

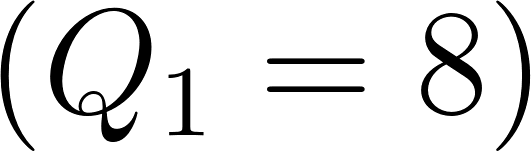
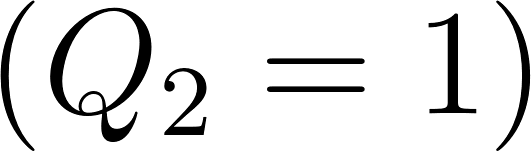
Scatnet provides default options for most of the parameters that require tuning in order to derive the scattering coefficients for an input signal. In particular, for audio signals, the most important hyperparameters set by the library is the number of scattering layers that captures the entire audio 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.

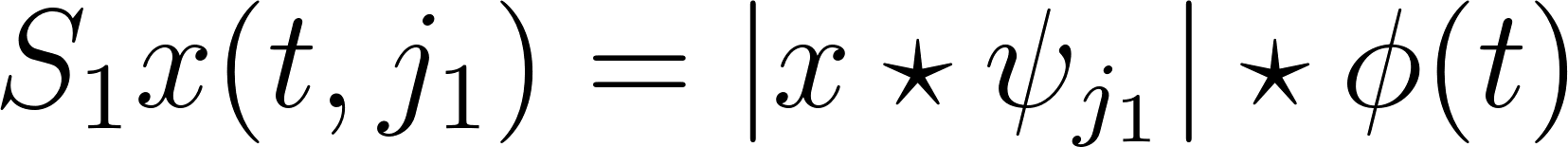
Further, the maximum number of wavelets J is automatically derived from the sampling from the sampling period T. The filter banks are formed by dilating the mother wavelet ([](https://www.codecogs.com/eqnedit.php?latex=%5Cpsi%0)) by the dyadic factor ([](https://www.codecogs.com/eqnedit.php?latex=2%5E%7B1%2FQ%7D%0)). In the Fourier domain this is expressed as

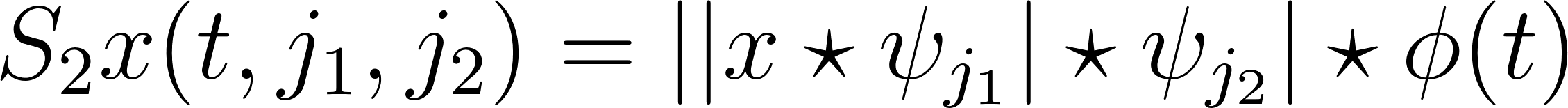
[](https://www.codecogs.com/eqnedit.php?latex=%20%5Chat%5Cpsi_j(%5Comega)%20%3D%20%5Chat%5Cpsi(2%5E%7Bj%2FQ%7D%5Comega).%0)

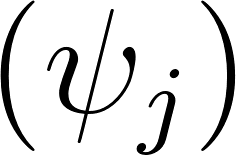
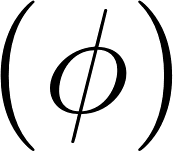
For audio application, to ensure optimized frequency coverage without frequency-redundancy or overlapping, the mother wavelet [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi%20%5C)%0) is chosen such that  [](https://www.codecogs.com/eqnedit.php?latex=(Q_1%20%3D%208)%0) and [](https://www.codecogs.com/eqnedit.php?latex=(Q_2%20%3D%201)%0) 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 (\ref{sec\_c5\_wproof} and \ref{sec\_c7\_wparams}) where it is shown that the first and second order scattering coefficients are respectively defined by the following forms

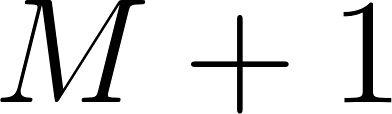
[](https://www.codecogs.com/eqnedit.php?latex=S_1x(t%2Cj_1)%20%3D%20%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cphi(t)%20%0) - - - (1)

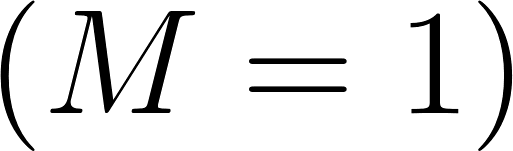
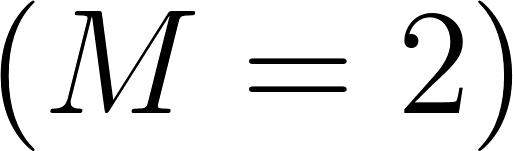
[](https://www.codecogs.com/eqnedit.php?latex=S_2x(t%2Cj_1%2Cj_2)%20%3D%20%7C%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cpsi_%7Bj_2%7D%7C%5Cstar%5Cphi(t)%20%0) - - - (2)

where [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi_%7Bj%7D%20%5C)%0) are band-pass filters and [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cphi%20%5C)%0) is a low-pass filter.

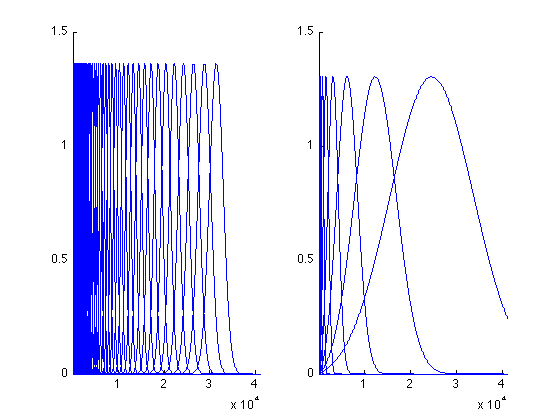
Furthermore, the wavelet transform operators(Wop) created by the wavelet\_factory\_1d function are only function handles and do not have any data in themselves. A second return value may be retrieved from the wavelet factory which contains the set of filters returned as a cell array by the wavelet factory.

|  |
| --- |
| [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 [](https://www.codecogs.com/eqnedit.php?latex=M%2B1%0) elements, where only [](https://www.codecogs.com/eqnedit.php?latex=M%0) 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 filters{m}.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 [](https://www.codecogs.com/eqnedit.php?latex=%5C(M%20%3D%201%5C)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5C(M%20%3D%202%5C)%0). The resulting plot is displayed in Figure \ref{fig\_c3\_wplot}.

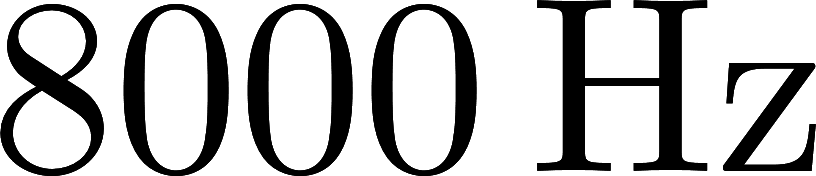
|  |
| --- |
| figure;  for m = 1:2  subplot(1,2,m);  hold on;  for k = 1:length(filters{m}.psi.filter)  plot(realize\_filter(filters{m}.psi.filter{k}, N));  end  hold off;  ylim([0 1.5]);  xlim([1 5\*N/8]);  end |



#### Using the Matlab API

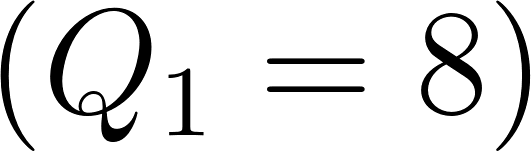
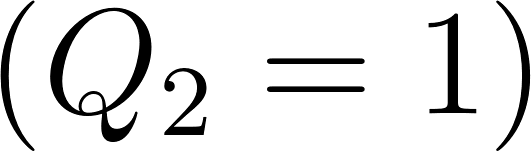
In the previous sections it was seen how the ScatNet toolbox calculates scatter coefficients based on wavelet theory. In this section, the scattering spectrum of an audio signal is implemented using only three command calls to ScatNet library.

The three steps taken in this section are as follows. First load the audio file and 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 using the “load handel” command.

Given that the sampling rate of the loaded clip is [](http://www.texrendr.com/?eqn=%5C(%208000%20%5Ctext%7B%5C%2CHz%7D%20%5C)%0), the parameters set are

1. N - the number of samples in the signal, and
2. T - the window size. Here, T is set to 4096 which is approximately 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 [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20Q_1%20%3D%208%20%5C)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20Q_2%20%3D%201%20%5C)%0). 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 \ref{sec\_c3\_sbatch}, 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); |

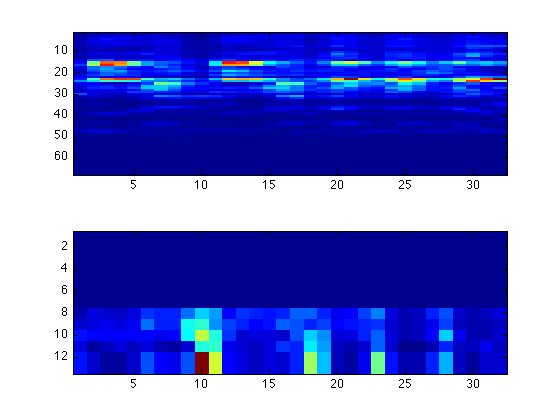
#### Scatter Feature Enhancements and Batch Processing\label{sec\_c3\_sbatch}

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); |

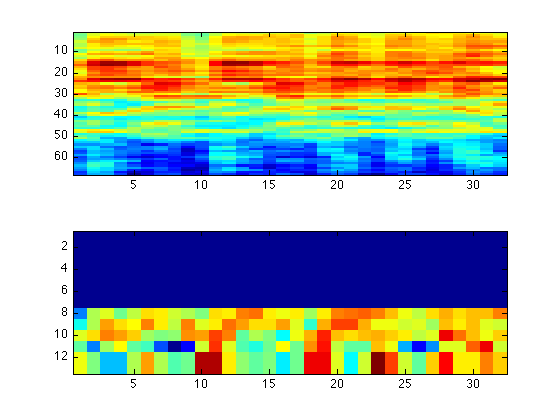
The resulting illustration in figure \ref{ref\_c3\_sgram00} is the unnormalised scattergram



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 \ref{fig\_c3\_sgram01}.



Finally, to utilise the scattering coefficients as features for classification tasks, we extract the vector form using format\_scat function.

|  |
| --- |
| [S\_table, meta] = format\_scat(S); |

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

For batch processing ScatNet provides a database feature which can accept a collection of input vectors rather than a single input signal. The following commands show ScatNet commands for performing batch processing on the [GTZAN dataset](https://github.com/AmbarZaidi/Audio-Genre-Classification) 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 \ref{sec\_c3\_scat00,sec\_c3\_scat01} and \ref{sec\_c3\_scat02} 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 subsampling each signal. The feature\_sampling option is used to specify subsampling.

|  |
| --- |
| database\_options.feature\_sampling = 8; |

Finally, a call is made to prepare\_database function to compute all the scattering coefficients 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 \ref{app\_scatcode}) thus bypassing the batch processing utility of ScatNet.

### 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 \cite{goldsborough2016tour,abadi2016tensorflow} and \cite{abadi2017computational}.

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).

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 from 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 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 to the next operation. The inputs to a labeled operation are therefore the labels which have edges directed towards the operation.

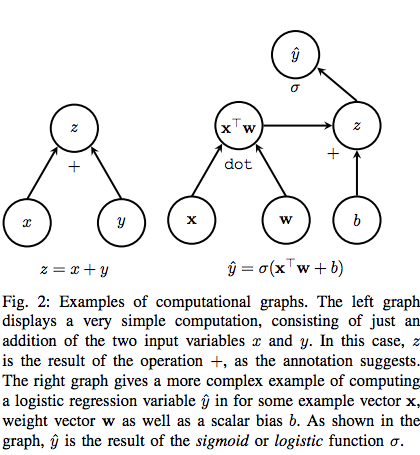
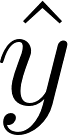
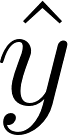


Figure 2: Sample Data flow Graphs

In Figure 2, The left graph displays a basic compute graph consisting of an addition operator having two input variables [](https://www.codecogs.com/eqnedit.php?latex=x%0) and [](https://www.codecogs.com/eqnedit.php?latex=y%0). The result, [](https://www.codecogs.com/eqnedit.php?latex=z%0) is the output of the [](https://www.codecogs.com/eqnedit.php?latex=%2B%0) operation. The right graph gives an example logistic regression function. [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7By%7D%0) is the final output of the function for some sample vector $\mathbf{x}$, weight vector [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bw%7D%0) and scalar bias [](https://www.codecogs.com/eqnedit.php?latex=b%0). As shown in the graph, [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7By%7D%0) is the output of the sigmoid or logistic function [](https://www.codecogs.com/eqnedit.php?latex=%5Csigma%0).

#### Backprop nodes

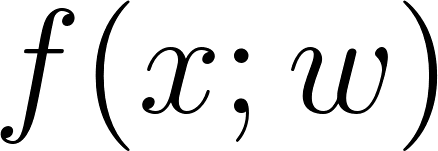
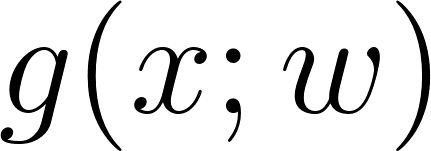
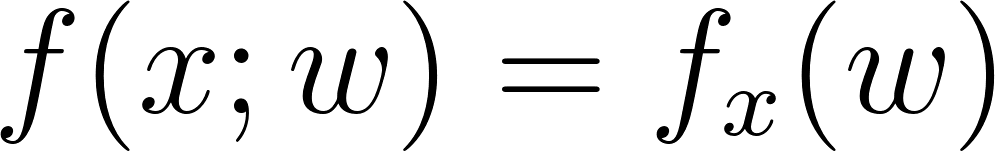
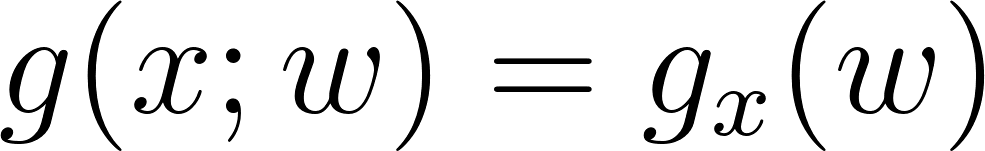
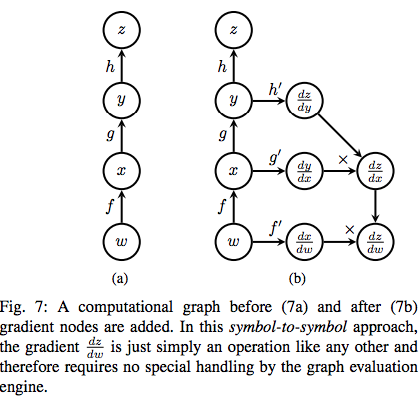
The Backprop algorithm \cite{Goodfellow-et-al-2016} is an efficient method to compute error values for weights within a multilayer or deep neural network. The algorithm is summarised as follows. Assuming a neural network with two hidden layers. The two layers within the network respectively have output functions [](https://www.codecogs.com/eqnedit.php?latex=f(x%3Bw)%0) and [](https://www.codecogs.com/eqnedit.php?latex=g(x%3Bw)%0) such that [](https://www.codecogs.com/eqnedit.php?latex=f(x%3Bw)%3Df_x(w)%0) and [](https://www.codecogs.com/eqnedit.php?latex=g(x%3Bw)%3Dg_x(w)%0). Where [](https://www.codecogs.com/eqnedit.php?latex=x%0) is the input from the previous layer or from the input layer and [](https://www.codecogs.com/eqnedit.php?latex=w%0) are the weights. The error function [](https://www.codecogs.com/eqnedit.php?latex=e%0), is an implicit function of all the previous layers. In the case of the 2-layer network [](https://www.codecogs.com/eqnedit.php?latex=e%3D(f_x%20%5Ccirc%20g_x)(w)%3Df_x(g_x(w))%0). 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 [](https://www.codecogs.com/eqnedit.php?latex=de_x%2Fdw%0). The backprop algorithm therefore uses the formula [](https://www.codecogs.com/eqnedit.php?latex=%5Bf_x(g_x(w))%5D'%3Df'_x(g_x(w))%5Ccdot%20g'x(w)%0) as it traverses the graph in reverse to compute the updates.

Figure \ref{fig\_c3\_bprop} 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.



This approach of adding gradient nodes to the compute graph is known as the symbol-to-symbol approach. This straightforward approach benefits from the fact that no error handling is required as the gradient node is just another operation within the compute graph.

#### 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 \ref{sec\_c3\_tfwf} for a walk through). The process of stepping back through a loop in reverse to compute gradients is known as back-propagation through time \citep{al2016theano}.

#### Checkpoints

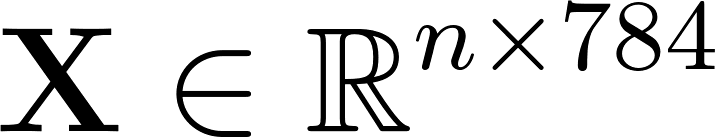
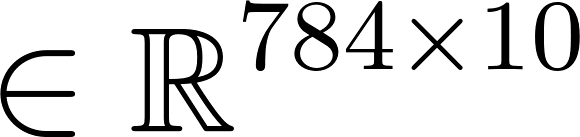
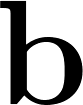
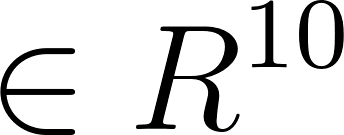
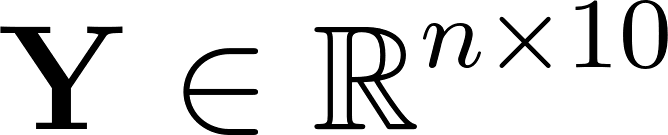
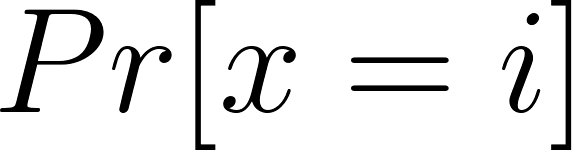
One can add Save a node to a compute graph, connecting them to variables whose tensors can then be serialized. At another instance one may connect the same variable to a Restore operation. This operation deserializes the stored tensor at another point within the execution graph. This is especially useful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

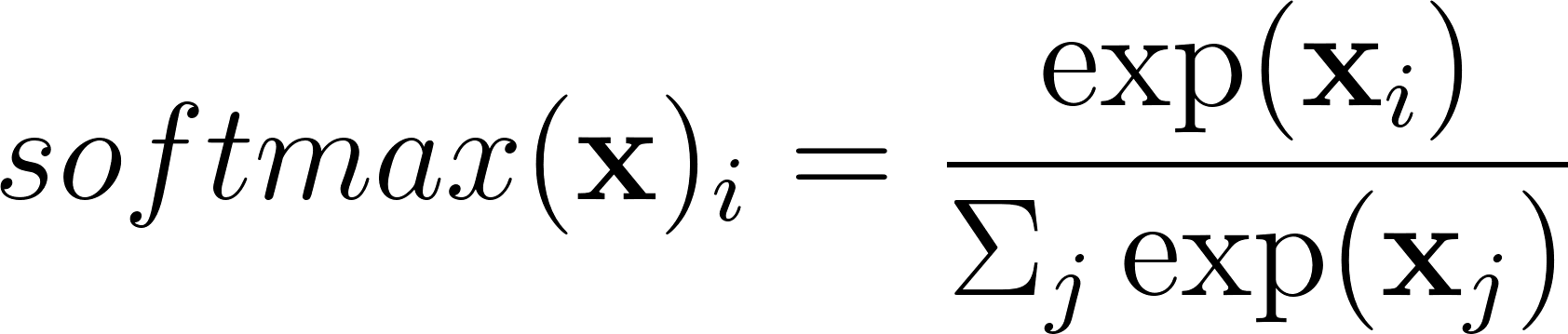
#### Programming Interface

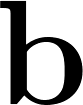
Tensorflow implementation provides two developer interfaces which include the Python interface and the C++ interface. While the python interface offers a rich feature set for creation and execution of computation graphs, the C++ interface is primarily a back end implementation with a much more limited API primarily used for executing graphs built with Python and serialised to Google’s protocol buffer.

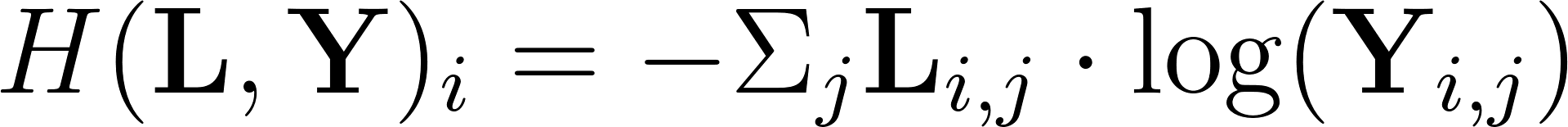
It is worth noting that unlike PyTorch \citep{ketkar2017introduction}, the Python API handshakes very well with NumPy\citep{numpy} numeric and scientific open source programming library. As such, TensorFlow tensors can be naturally substituted with NumPy ndarrays without any need for type-conversion seen in PyTorch tensors.

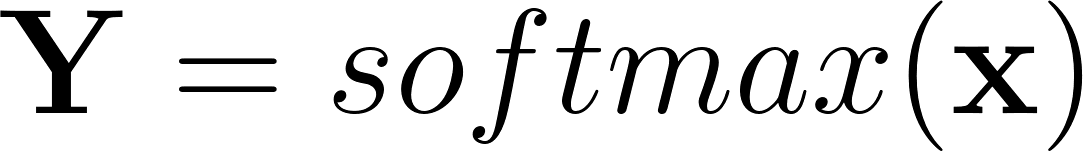
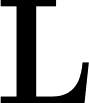
#### Tensorflow client model walk through \label{sec\_c3\_tfw}

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\citep{krizhevsky2012imagenet} dataset. In this dataset, the examples are small images 28 x 28 pixels depicting handwritten digits from 0 to 9. The examples form a matrix having the shape [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BX%7D%5Cin%5Cmathbb%7BR%7D%5E%7Bn%5Ctimes%20784%7D%0) where [](https://www.codecogs.com/eqnedit.php?latex=n%0) represents the number of images, and 784 represents the flattened 28 x 28 pixel image. The client code performs an affine transform operation, [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BX%5Ccdot%20W%2Bb%7D%0), where [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BW%7D%0) is the matrix of weights [](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20%5Cmathbb%7BR%7D%5E%7B784%20%5Ctimes%2010%7D%0), and [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bb%7D%0) is a vector of biases [](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20R%5E%7B10%7D%0). The result of the affine transform operator is the matrix [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BY%7D%5Cin%20%5Cmathbb%7BR%7D%5E%7Bn%5Ctimes%2010%7D%0). The resulting non-probabilistic logits gives an unnormalised distribution of digits.In order to obtain the valid probability distribution [](https://www.codecogs.com/eqnedit.php?latex=Pr%5Bx%3Di%5D%0) where [](https://www.codecogs.com/eqnedit.php?latex=x%0)-th example is classified as the digit [](https://www.codecogs.com/eqnedit.php?latex=i%0), the softmax operation is called.

[](https://www.codecogs.com/eqnedit.php?latex=softmax(%5Cmathbf%7Bx%7D)_i%3D%5Cfrac%7B%5Cexp(%5Cmathbf%7Bx%7D_i)%7D%7B%5CSigma_j%5Cexp(%5Cmathbf%7Bx%7D_j)%7D%0).

Error loss values are then computed using an objective function and the model’s current training parameters [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BW%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bb%7D%0). This is obtained from the cross entropy calculation given by

[](https://www.codecogs.com/eqnedit.php?latex=H(%5Cmathbf%7BL%2CY%7D)_i%3D-%5CSigma_j%5Cmathbf%7BL%7D_%7Bi%2Cj%7D%5Ccdot%5Clog(%5Cmathbf%7BY%7D_%7Bi%2Cj%7D)%0)

Where [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BY%7D%3Dsoftmax(%5Cmathbf%7Bx%7D)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BL%7D%0) are the correct one-hot-encoded labels. More precisely, the batch-mean loss over all inputs [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bx%7D%0).

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. The complete code listing is given in appendix \ref{app\_tfw}.

#### 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

### [Choregraphe](https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5326209)

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 \ref{pot2009choregraphe}. 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.

### Alisa \label{sec\_c3\_alisa}

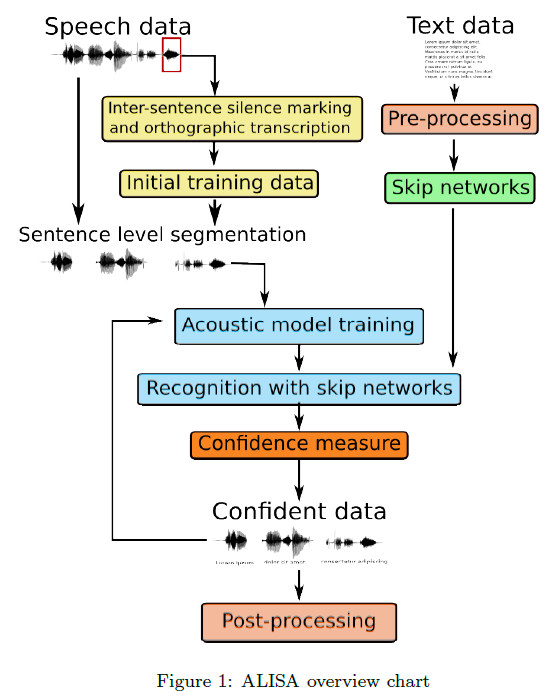
Alisa tool is a lightly supervised sentence segmentation tool based on Voice Activity Detection (VAD) algorithms. It is so called lightly supervised because it requires small amounts of training data. Generally the tool was asserted to be optimised for sentence segmentation and offered assistance in 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 3-5 Alisa Overview (Stan et al., 2016)*

Figure 3-5 \ref{fig\_c3\_alisa00} 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 \citep{schluter2001model} and methods described in \citep{novotney2009analysis}. It was observed that Alisa provided good alignment but was not fully featured. For instance it had no way of adding insertions and substitutions in the audio data not provided in the transcription. Finally, Alisa was found to be restricted to only languages that can utilise the English alphabet.

## Initial Experiments

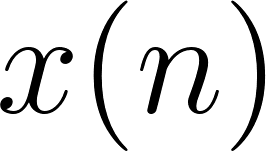
The experiments in the following sections describe initial experiments based on the initial study of a language learning companion before the research was narrowed down to a low resource speech recognition. These preliminary experiments in addition to a preliminary Language Learning Survey helped to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

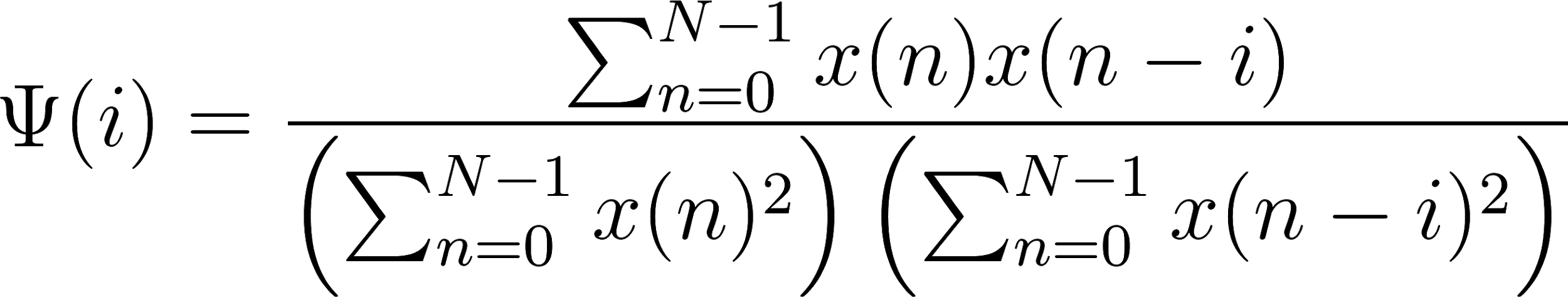
The following sections describe analysis of raw wave-forms using auto-correlation signal processing in Matlab and experiments made with the Nao robot speech processing engine and experiments with speech recognition toolkit and speech processing tasks. These tasks include digit recognition systems using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

### Autocorrelation 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 autocorrelation estimate. This presents an unnormalised posterior distribution measurement of phoneme segments over the entire signal.

The correlation theory is based on the idea that when a signals is superimposed on itself in a time-shifted manner, the convolution over itself is highest when the two signals have zero time lag that is, perfectly overlapped in sync and the better 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 autocorrelation value is obtained in \cite{picone1996fundamentals} from a signal [](https://www.codecogs.com/eqnedit.php?latex=x(n)%0) in the following equation:

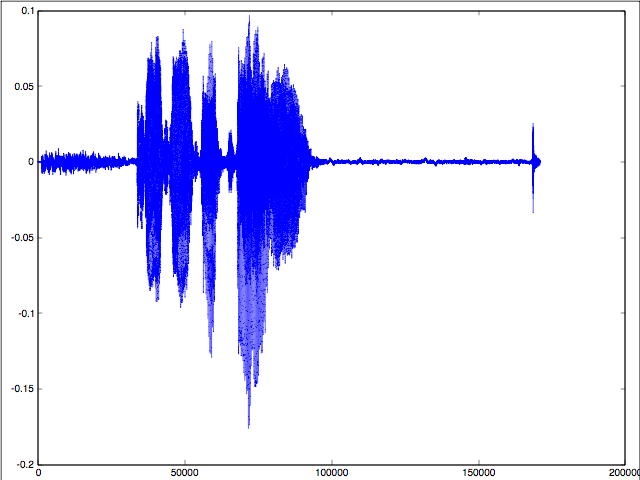
[](https://www.codecogs.com/eqnedit.php?latex=%5CPsi(i)%3D%5Cfrac%7B%5Csum_%7Bn%3D0%7D%5E%7BN-1%7Dx(n)x(n-i)%7D%7B%5Cleft(%5Csum_%7Bn%3D0%7D%5E%7BN-1%7Dx(n)%5E2%5Cright)%5Cleft(%5Csum_%7Bn%3D0%7D%5E%7BN-1%7Dx(n-i)%5E2%5Cright)%7D%0)*- - - 4-4*

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

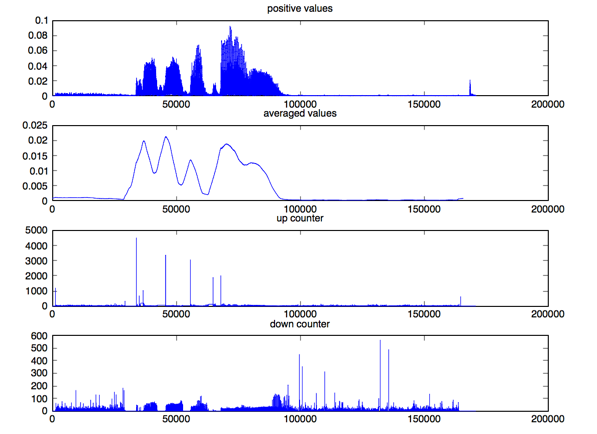
This two stage procedure performs segmentation of phonemes and then discovery of phoneme clusters using a statistical auto-correlation algorithm. The process is described in the following sections.

#### Segmentation

Figure 4-2 describes the various steps of the segmentation phase while Figure 4-1 shows the original audio file. At the segmentation phase, we first of all adjust the scale of the original raw audio file to have only positive values rather than having it centred on zero (Figure 4-2a). At the next step a smoothing filter based on experimentation is used to perform both smoothing as well as determining the peaks and trough (Figure 4-2b). Then a threshold is applied to segment the waveform based on discovered inflection points (Figure 4-2c).



*Figure 4-1 Original Signal for Autocorrelation Method*



*Figure 4-2 Autocorrelation Method - Segmentation Results*

#### Autocorrelation

At the autocorrelation 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 resampled to eliminate mismatching of contour representations of the individual phonemes.

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

### 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 this the pocketsphinx method, acoustic models trained high performance systems can then be deployed to Nao for fast decoding within the Nao.

### Digit Speech Recognition and Alignment Experiments \label{sec\_digitspeech}

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 \cite{stan2016alisa} tool which is a python based tool with calls made to the HMM toolkit \cite{young2002htk}. 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 datasets. 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.

## End-to-end models experiments \label{sec\_postalign}

A significant issue arises when using HMM-based toolkits such as Kaldi. This is the requirement for aligned speech. In more recent endeavours, there has been efforts towards automatic alignment of transcribed audio speech recordings through successive Baum-Welch estimation techniques \cite{gales2014speech,ragni2018automatic,ragni2014data

}. However, this technique is not particularly compatible with end-to-end goals adopted for this research as it would require preprocessing and successive pre-training of the data set.

The following section describes the post-alignment experiments and in Chapter \ref{ch6\_speech}, how these methods deal with the problem of automatic speech alignment in a fashion compatible with end-to-end speech processing. The end-to-end requirements were desirable for low-resource speech recognition as it introduces a simpler speech model design. The downside however to the end-to-end approach is the dependency on very deep recurrent neural network structures which require large volumes of data for successful training.

### Tensor flow sequence-to-sequence character-to-diacritically-labelled-character model\label{sec\_c2d}

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

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

### Sequence-to-sequence grapheme-to-phoneme (G2P) model

This follow up experiment to the previous experiment in section \ref{sec\_c2d}, attempts to automatically generate a phonetic dictionary from graphemes in a text corpus. Grapheme-to-phoneme experiments come in two flavours, the first being a continuation of the previous experiment, that is, using diacritically marked symbols, and the second flavour using non-marked graphemes as input. The experiments we performed used the latter non-marked graphemes as input. As this experiment was also a subtask in HMM speech model building, the results of these experiments were reserved for further study.

What follows in the next three sections are sequence-to-sequence experiments actively developed in this research and are detailed in chapters (\ref{ch6\_wlm,ch6\_speech,ch8\_future}). A brief summary of the experiments are highlighted in the following sections. Note that these models all utilise TensorFlow deep learning library including the Bi-directional speech model (section \ref{sec\_be2e}) which is built on top of Mozilla DeepSpeech with the exception of section \ref{c3sec\_espnet} which is based on PyTorch; a similar deep learning library.

### GRU language model for Wakirike language based on tensorflow\label{sec\_grulm}

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 \ref{ch3RNN} and \ref{ch6\_speech}, 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.

### Bi-Directional LSTM-based end-to-end speech model \label{sec\_be2e}

A similar LSTM sequence-to-sequence network based on Baidu Research’s original research design \cite{hannun2014deep} 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 (\ref{sec\_c3\_alisa},\ref{sec\_digitspeech},\ref{sec\_postalign}).

### EspNet Experiments\label{c3sec\_espnet}

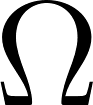
The ESPNet (End-to-end SPeech Network) toolkit \citep{watanabe2018espnet}, is a speech processing toolkit that was of interest to this research because it offered end-to-end capabilities not only in Automatic Speech Recognition (ASR) but also in Text-To-Speech (TTS) or speech synthesis and other speech-sequence-processing related tasks. In addition, the toolkit offers multi-modal training combining both Attention networks \cite{vaswani2017attention} with CTC Transformer networks as well as multi-channel feature representation that is, the fusing together of multiple feature representations of an audio signal. Only preliminary experiments were carried out using ESPNet and is discussed in Chapters \ref{ch6\_speech} and \ref{ch08furtherstudy} of this work.

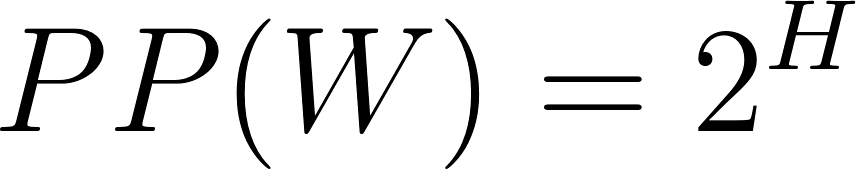
## Method of evaluation

System building methodology for speech recognition systems requires models to be evaluated against speech recognition Machine Learning metrics. For language models, perplexity metric was used for evaluation. Bleu has also been used as a metric for evaluating language models.

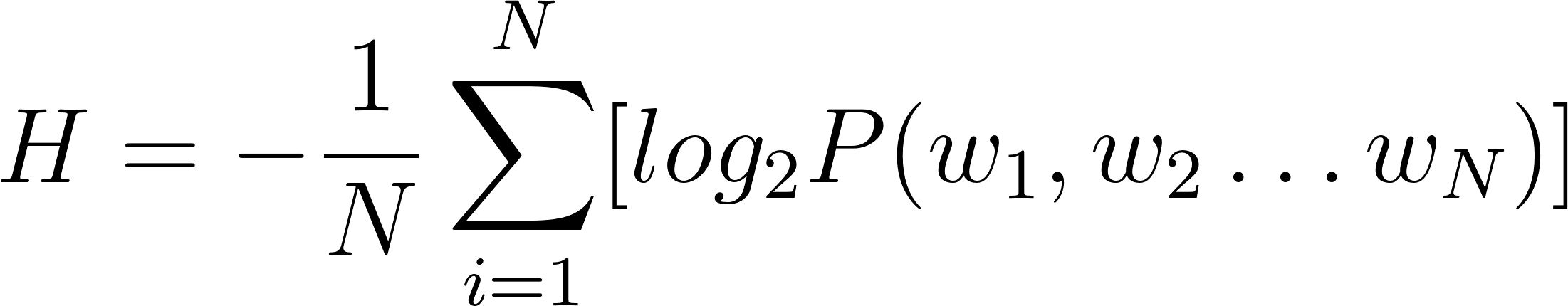
Perplexity measures the complexity of a language that the language model is designed to represent \citep{1976jelinekcontinuous}. In practice, the entropy of a language with an N-gram language model [](https://www.codecogs.com/eqnedit.php?latex=P_N(W)%0) is measured from a set of sentences and is defined as

[](https://www.codecogs.com/eqnedit.php?latex=H%3D%5Csum_%7B%5Cmathbf%7BW%7D%5Cin%5COmega%7DP_N(%5Cmathbf%7BW%7D)%0)

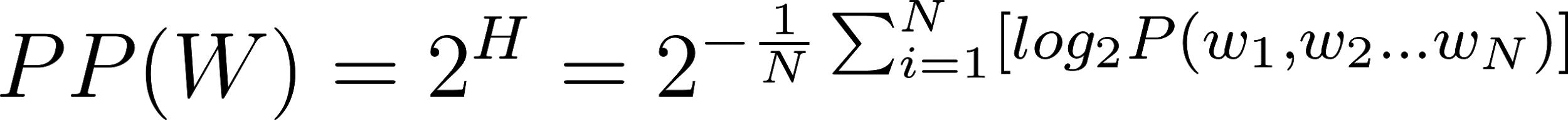
where [](https://www.codecogs.com/eqnedit.php?latex=%5COmega%0) is a set of sentences of the language. The perplexity, which is interpreted as the average word-branching factor, is defined as

[](https://www.codecogs.com/eqnedit.php?latex=PP(W)%3D2%5EH%0)

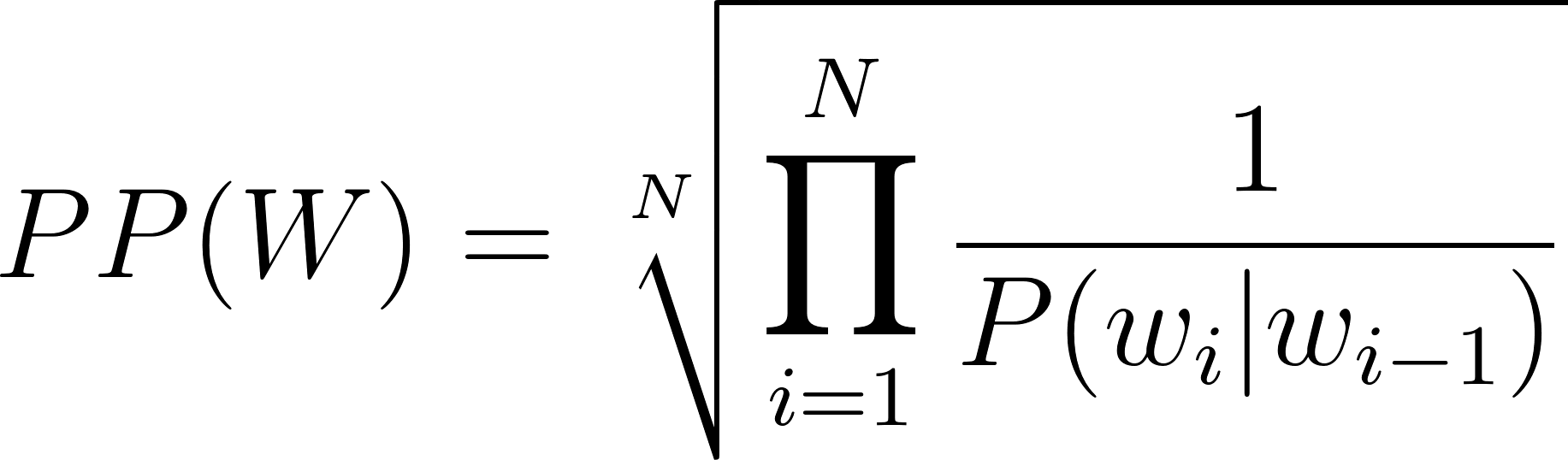
where H is the average entropy of the system or the average log probability defined as

[](https://www.codecogs.com/eqnedit.php?latex=H%3D-%5Cfrac%7B1%7D%7BN%7D%5Csum_%7Bi%3D1%7D%5EN%5Blog_2P(w_1%2Cw_2%5Cdots%20w_N)%5D%0)

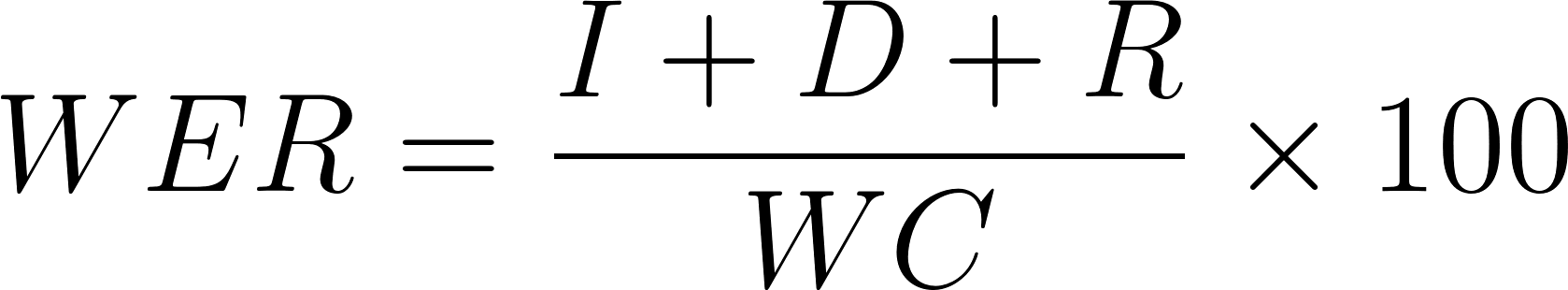
For a bi gram model therefore, equation (\ref{eqn\_c2\_lm07}) becomes

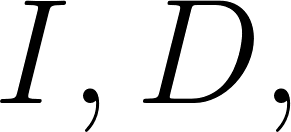
[](https://www.codecogs.com/eqnedit.php?latex=PP(W)%3D2%5EH%3D2%5E%7B-%5Cfrac%7B1%7D%7BN%7D%5Csum_%7Bi%3D1%7D%5EN%5Blog_2P(w_1%2Cw_2%5Cdots%20w_N)%5D%7D%0)

After simplifying we have

[](https://www.codecogs.com/eqnedit.php?latex=PP(W)%3D%5Csqrt%5BN%5D%7B%5Cprod_%7Bi%3D1%7D%5EN%5Cfrac%7B1%7D%7BP(w_i%7Cw_%7Bi-1%7D)%7D%7D%0)

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

[](https://www.codecogs.com/eqnedit.php?latex=WER%3D%5Cfrac%7BI%2BD%2BR%7D%7BWC%7D%5Ctimes%20100%0)

Here [](https://www.codecogs.com/eqnedit.php?latex=I%2CD%2C%0) and [](https://www.codecogs.com/eqnedit.php?latex=R%0) are wrong insertions, deletions and replacements respectively and [](https://www.codecogs.com/eqnedit.php?latex=WC%0) is the word count.

Metrics used for low speech recognition in the zero speech challenge \citep{versteegh2015zero} also 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).

## Chapter Summary

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

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

# Chapter 4: Recurrent Neural networks in Speech Recognition

The HMM model described in Chapter \ref{c02} uses a divide and conquer strategy which has also been described as a generative Machine Learning algorithm in which we use the smaller components’ representations as modelled by the HMM to learn the entire speech process. In previous chapters, this was referred to as the bottom-top strategy. The discriminative method however uses the opposite mechanism. Instead of using the building blocks of speech to determine speech parameters of a HMM, the discriminative strategy determines the posterior probability directly using the joint probability distribution of the parameters involved in the discriminative process. The discriminative approach, discussed in this chapter focuses in on Neural network architectures.

## \section{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 \cite{cowan1990discussion} as the Perceptron in Figure \ref{fig\_3\_1\_ptron}

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=7cm]{thesis/images/ptron2.png}\\

\caption{Perceptron} \label{fig\_3\_1\_ptron}

\end{figure}

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 functions. In Figure \ref{fig\_3\_2\_nn} 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 \ref{fig\_3\_2\_nn} :

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=7cm]{thesis/images/ptron3.png}\\

\caption{Neural network} \label{fig\_3\_2\_nn}

\end{figure}

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.

### \subsection{Multi-layer Perceptron (MLP)}

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

\begin{equation}

z\_j=h(b\_j)=\frac{1}{ 1+e^{-b\_j}}

\label{eqn\_c3\_nn\_01}

\end{equation}

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

\begin{equation}

b\_j=\sum\_{i=0}^Nw\_{ji}^{(1)}\qquad j=1,2,\dots,M

\label{eqn\_c3\_nn\_02}

\end{equation}

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.

### \subsection{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 $(\mathbf{x})$.

\begin{equation}

p(C\_1|\mathbf{x})=f(a)=f(\mathbf{w^\top x}+w\_0)

\label{eqn\_c3\_nn\_02}\end{equation}

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

\begin{equation}

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)}

\label{eqn\_c3\_nn\_03}\end{equation}

Dividing through by $p(\mathbf{x}|C\_1)p(C\_1)$ gives us

\begin{equation}

p(C\_1|\mathbf(x)=\frac{1}{1+\frac{p(\mathbf{x}|C\_1)p(C\_1)}{p(\mathbf{x}|C\_2)p(C\_2)}}

\label{eqn\_c3\_nn\_04}\end{equation}

If we define the ratio of the log posterior probabilities as

\begin{equation}

a=\ln\frac{p(\mathbf{x}|C\_1)p(C\_1)}{p(\mathbf{x}|C\_2)p(C\_2)}

\label{eqn\_c3\_nn\_05}\end{equation}

If we substitute back into (4) we have:

\begin{equation}p(C\_1|\mathbf{x})=f(a)=\frac{1}{1+e^{-a}}

\label{eqn\_c3\_nn\_06}

\end{equation}

Here $a=\mathbf{w^\top x}=w\_0$. 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:

\begin{equation}

y\_k=p(C\_k|\mathbf{x})=\frac{e^{a\_k}}{\Sigma\_{\ell=1}^K e^{a\_\ell}}

\label{eqn\_c3\_nn\_07}\end{equation}

\begin{equation}

a\_k=\sum\_{i=0}^dw\_{ki}x\_i

\label{eqn\_c3\_nn\_08}\end{equation}

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.

### \subsection{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 \citep{boden2002guide}.

### \subsection{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 $N$.

\begin{equation}

E^n=\frac{1}{2}\sum\_{k=1}^K(y\_k^n-t\_k^n)^2

\label{eqn\_c3\_nn\_09}\end{equation}

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.

\begin{equation}\nabla\_{\mathbf{W}}E=\left(\frac{\partial E}{\partial w\_{10}},\dots,\frac{\partial E}{\partial w\_{ki}},\dots,\frac{\partial E}{\partial w\_{Kd}}\right)

\label{eqn\_c3\_nn\_10}\end{equation}

The adjustment on each weight therefore on each iteration is:

\begin{equation}

w\_{kj}^{\tau+1}=w\_{kj}^{\tau}-\eta\frac{\partial E}{\partial w\_{kj}}

\label{eqn\_c3\_nn\_11}\end{equation}

Where $\tau$ is the iteration and $\eta$ 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.

## \section{RNN, LSTM and GRU Networks}

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

### \subsection{Deep Neural Networks (DNNs)}\label{dnn}

Deep neural networks have been accepted to be networks having multiple layers and capable of hierarchical knowledge representation \citep{yu2016automatic}.

This will therefore include multi-layer Perceptrons (MLPs) having more than one hidden layer \citep{dahl2012context} as well as deep belief networks (DBNs)\citep{mohamed2009deep,yu2010roles} having a similar structure. Therefore, following the MLP architecture, A DNN uses multiple hidden layers and generates distribution function, $p(c|x\_t)$ on the output layer when an input vector $\mathbf{x}\_t$ is applied. At the first hidden layer, activations are vectors evaluated using

\begin{equation}\mathbf{h}^{(1)}=\sigma(\mathbf{W}^{(1)T}\mathbf{x}\_t+\mathbf{b}^{(1)})

\label{eqn\_c3\_dnn01}\end{equation}

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

\begin{equation}\mathbf{h}^{(1)}=\sigma(\mathbf{W}^{(1)T}\mathbf{h}^{(i-1)}+\mathbf{b}^{(1)})

\label{eqn\_c3\_dnn02}\end{equation}

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,

\begin{equation}p(c=c\_k|x\_t)=\frac{exp(-(\mathbf{W}^{(s)T}\_kh^{(i-1)}+b\_k^{(1)}))}{\sum\_j exp(-(\mathbf{W}^{(s)T}\_kh^{(i-1)}+b\_k^{(1)}))}

\label{eqn\_c3\_dnn03}\end{equation}

$W\_k^{(s)}$ 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 \ref{c3\_ctc}, of this thesis, predicts per character conditional distributions.

Combining equations (\ref{eqn\_c3\_nn\_11}, \ref{eqn\_c3\_dnn01}, \ref{eqn\_c3\_dnn02} and \ref{eqn\_c3\_dnn03}) the following simplified algorithm ensues

\begin{algorithm}[H]

\SetAlgoLined

\KwResult{Optimal weights }

initialise weights randomly\;

\While{error is significant or epochs less than maximum}{

forward computation in equation (\ref{eqn\_c3\_dnn01} and \ref{eqn\_c3\_dnn02} )\;

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 (\ref{eqn\_c3\_nn\_11});

}

\caption{DNN training algorithm}

\end{algorithm}

### \subsection{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 \citep{yu2016automatic}.

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 \citep{hannun2014first, yu2016automatic}.

\begin{equation}h\_t^{(j)}=\sigma(\mathbf{W}^{(j)T}h\_t^{(i-1)}+\mathbf{W}^{(j)T}\_kh\_{t-1}^{(j)}+b^{(j)}))

\label{eqn\_c3\_rnn01}\end{equation}

It can be seen in equation (\ref{eqn\_c3\_rnn01}) above that given a selected RNN hidden layer $j$, a temporally recurrent weight matrix $W^{(f)}$ is computed for output activations $h^{(j)}\_{t-1}$ 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 (\ref{eqn\_c3\_rnn01}) 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) \citep{boden2002guide,jaeger2002tutorial} is derived in section \ref{c3\_BPTT} below.

### \subsection{Back propagation through time (BPTT) algorithm}\label{c3\_BPTT}

First we define an arbitrary but carefully chosen number of time steps $t=1,2,\dots,T$ such that at each time step the states of the neuron activations $j=1,2,\dots,J$ are captured.

Using the sum-squared error as the cost function

\begin{equation}

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 \label{eqn\_c3\_bptt01}\end{equation}

Where $c$ is a gradient descent convenience factor in equation (\ref{eqn\_c3\_bptt01}). $||\mathbf{l}\_t-\mathbf{y}\_t||$ 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 \cite{yu2016automatic} is involves the recursive computation of the cost function and updating of the network weights.

For each of these steps recall from equation (\ref{eqn\_c3\_rnn01}) 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

\begin{equation}

\delta^y\_T(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,\dots,L \label{eqn\_c3\_bptt04}\end{equation}

or

\begin{equation}

\mathbf{\delta}\_T^y=(\mathbf{l}\_T-\mathbf{y}\_T)\bullet g'(\mathbf{v}\_T) \label{eqn\_c3\_bptt05}\end{equation}

The error at the hidden layer is given as

\begin{equation}

\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 \label{eqn\_c3\_bptt06}

\end{equation}

or $\delta\_T^h=\mathbf{W}\_{hy}^T\mathbf{\delta}\_T^y\bullet f'(\mathbf{u}\_T)$

where $\bullet$ is element-wise multiplication.

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

\begin{equation}

\delta\_t^y(j)=(l\_t(j)-y\_t(j))g'(v\_t(j))\text{ for } j=1,2,\dots,L

\label{eqn\_c3\_bptt07}\end{equation}

or \begin{equation}

\mathbf{\delta}\_t^y = (\mathbf{l}\_t-\mathbf{y}\_t)\bullet g'(\mathbf{v}\_t) \label{eqn\_c3\_bptt08}\end{equation}

Therefore the output units are \begin{equation}\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,\dots,N \\ \text{ or } \delta\_t^h&=\left[\mathbf{W}\_{hh}^\top\mathbf{\delta}\_{t+1}^h+\mathbf{W}\_{hy}^\top\mathbf{\delta}\_t^y\right]\bullet f'(\mathbf{u}\_t)\end{aligned}\label{eqn\_c3\_bptt09}\end{equation}

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

#### \subsubsection{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{equation}

\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\mathbf{\delta}\_y^t\mathbf{h}\_t^\top\end{aligned} \label{eqn\_c3\_bptt10}\end{equation}

For the input weight matrices, we get \begin{equation}

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) \label{eqn\_c3\_bptt11}\end{equation}

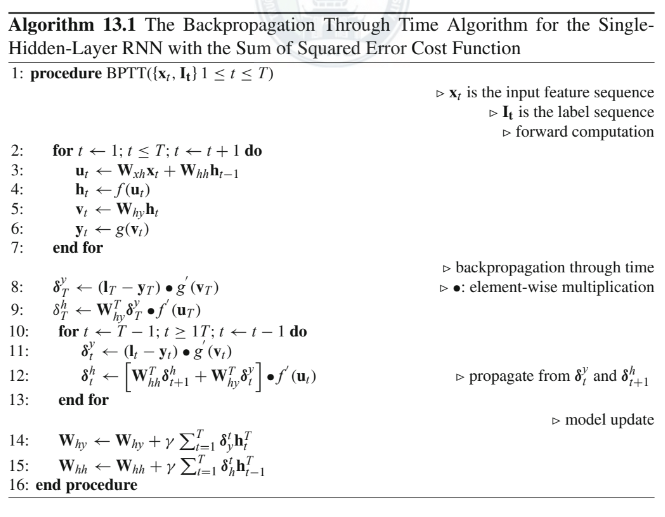
or \begin{equation}

\mathbf{W}\_{xh}^{new}=\mathbf{W}\_{xh}+\gamma\sum\_{t=1}^T\mathbf{\delta}\_h^t\mathbf{x}\_t^\top \label{eqn\_c3\_bptt\_13}\end{equation}

For the recurrent weight matrices we have

\begin{equation} \begin{split}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\mathbf{\delta}\_h^t(i)h\_{t-1}(j)\\ \text{ or }&=\mathbf{W}\_{hh}^{new}=\mathbf{W}\_{hh}+\gamma\sum\_{t=1}^T\mathbf{\delta}\_h^t\mathbf{h}\_{t-1}^\top \end{split} \label{eqn\_c3\_bptt14}\end{equation}

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



### \subsection{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 \cite{sak2014long}. 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:

\begin{equation}

\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)})

\label{eqn\_c3\_lstm01}

\end{equation}

\begin{equation}

\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)})

\label{eqn\_c3\_lstm02}

\end{equation}

\begin{equation}

\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)})\label{eqn\_c3\_lstm03}

\end{equation}

\begin{equation}

\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)})\label{eqn\_c3\_lstm04}\end{equation}

\begin{equation}

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

\label{eqn\_c3\_lstm05}

\end{equation}

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=7cm]{lstmcell}\\

\caption{An LSTM Cell \citep{graves2013hybrid}}\label{fig\_3\_3\_lstmcell}

\end{figure}

The gates in the above formula are illustrated in Figure \ref{fig\_3\_3\_lstmcell}. $\mathbf{i}\_t$ 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 \citep{cho2014learning}.

The architecture of a GRU is formulated as follows:

\begin{equation}

\mathbf{z}\_{(t)}=\sigma(\mathbf{W}\_{xz}^T\cdot\mathbf{x}\_{(t)}+\mathbf{W}\_{hz}^T\cdot\mathbf{x}\_{(t-1)})\label{eqn\_c3\_gru01}

\end{equation}

\begin{equation}

\mathbf{r}\_{(t)}=\sigma(\mathbf{W}\_{xr}^T\cdot\mathbf{x}\_{(t)}+\mathbf{W}\_{hr}^T\cdot\mathbf{x}\_{(t-1)})\label{eqn\_c3\_gru01}

\end{equation}\begin{equation}

\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)}))\label{eqn\_c3\_gru01}

\end{equation}\begin{equation}

\mathbf{h}\_{(t)}=(1-\mathbf{z}\_{(t)})\otimes(\mathbf{h}\_{(t-1)})+\mathbf{z}\_{(t)}\otimes\mathbf{g}\_{t}\label{eqn\_c3\_gru01}

\end{equation}

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.

## \section{Deep speech architecture}\label{deepspeech}

This work makes use of an enhanced RNN architecture called the Bi-directional Recurrent Neural Network (BiRNN). While \cite{hannun2014first} 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 $h^{(f)}\_t$ and $h^{(b)}\_t$ is given as.

\begin{equation}h\_t^{(f)}=\sigma(\mathbf{W}^{(j)T}h\_t^{(i-1)}+\mathbf{W}^{(f)T}\_kh\_{t-1}^{(j)}+b^{(j)}))

\label{eqn\_c3\_ds01}\end{equation}

\begin{equation}h\_t^{(b)}=\sigma(\mathbf{W}^{(j)T}h\_t^{(i-1)}+\mathbf{W}^{(b)T}\_kh\_{t+1}^{(b)}+b^{(j)}))

\label{eqn\_c3\_ds02}\end{equation}

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

\cite{hannun2014first} 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 $h^{(j)}\_t$ for the layer is now the superposition of the two RNN components,

\begin{equation}h\_t^{(j)}=h\_t^{(f)}+h\_t^{(b)}

\label{eqn\_c3\_ds03}\end{equation}

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

## CTC Loss Algorithm \label{c3\_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.

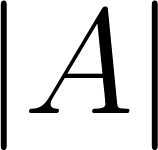
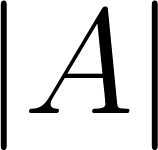
CTC works by making predictions at any point in the input sequence. For the case of speech modelling, CTC makes a character prediction for every time step of the raw audio input speech signal. Although this initially seems counter intuitive, this method models the many to one relationship seen in the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

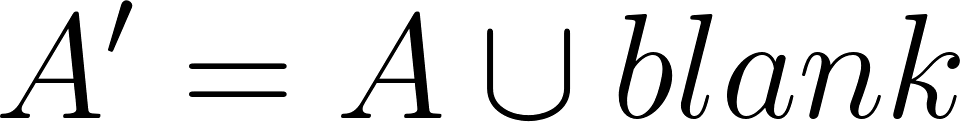
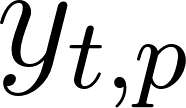
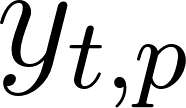
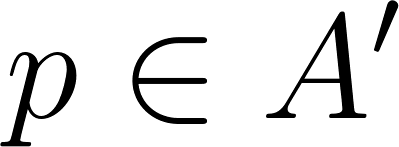
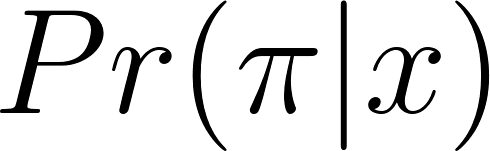
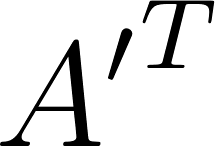
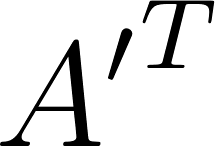
For hybrid DNN-HMM systems, speech or more accurately, acoustic models, require separate training of targets for every time-slice in the input sequence. Secondly, and as a consequence of this, it becomes necessary to segment the audio sequence, in order to provide targets for every time-slice. A third 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.

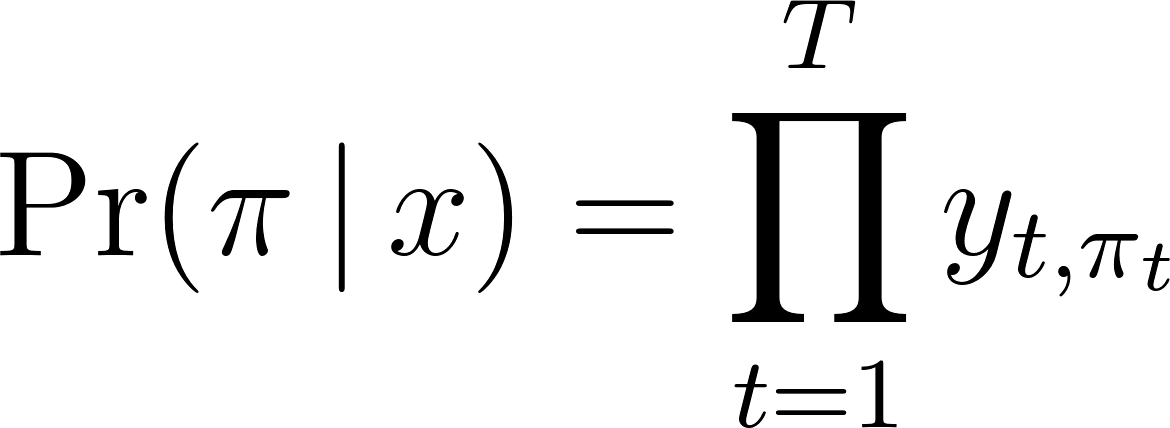
In the CTC case, so long as the overall sequence of labels is correct the network can be optimised to correct the temporal or fuzzy alignments. Since this many to one fuzzy alignment is simultaneously modelled in CTC, then there is no need for pre-segmented data. At the same time, CTC computes probabilities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

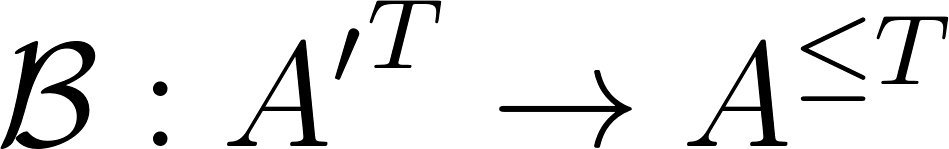
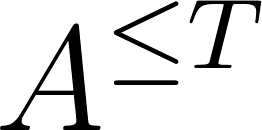
Similar to the HMM sequence model, the CTC algorithm is a sequence model that predicts the next label in a sequence as a cumulative of previous sequences. This section develops the CTC loss function borrowing concepts used in HMM models such as the forward backward algorithm as outlined in \citep{graves2006connectionist}. In the following paragraph we introduce terminology associated with the CTC loss function.

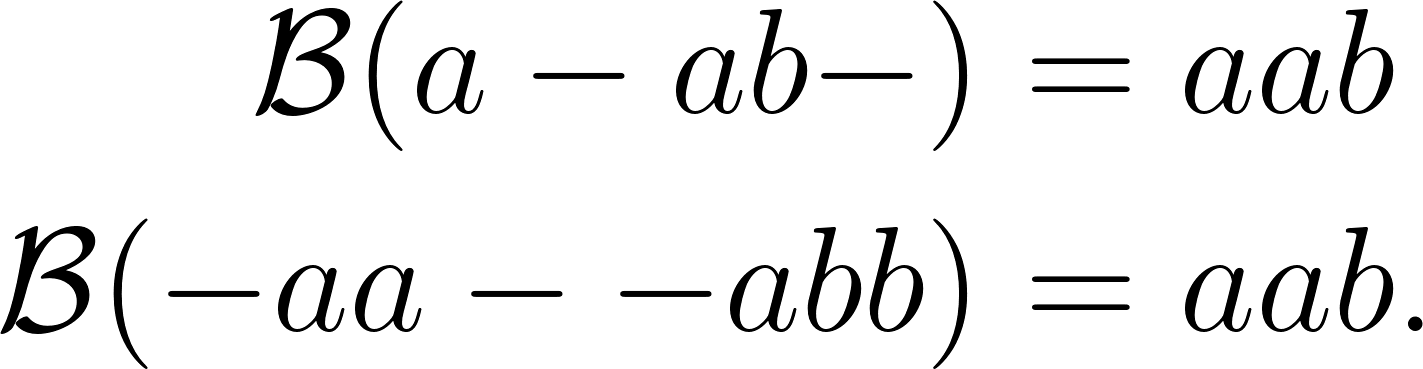
Given two symbols [](https://www.codecogs.com/eqnedit.php?latex=A%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) such that [](https://www.codecogs.com/eqnedit.php?latex=A%0) has a many to one relationship with [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0), signifying the temporal nature of the classification. The symbol [](https://www.codecogs.com/eqnedit.php?latex=A%0) 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).

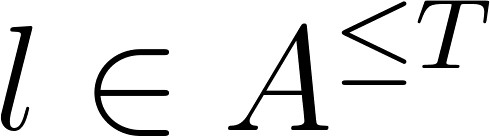
This output models the probability distribution of a complete sequence of arbitrary length [](https://www.codecogs.com/eqnedit.php?latex=%7CA%7C%0) over all possible labels in [](https://www.codecogs.com/eqnedit.php?latex=A%0) from activations within [](https://www.codecogs.com/eqnedit.php?latex=%7CA%7C%0). An extra activation is given to represent the probability of outputting a [](https://www.codecogs.com/eqnedit.php?latex=blank%0), 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.

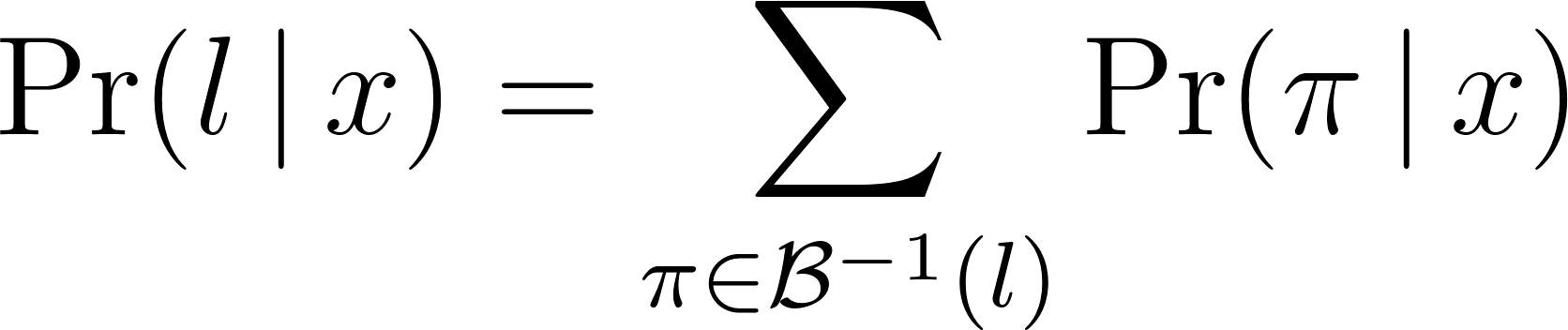
It is now possible to define the extended alphabet [](https://www.codecogs.com/eqnedit.php?latex=A'%20%3D%20A%20%5Ccup%20%7Bblank%7D%0), also, [](https://www.codecogs.com/eqnedit.php?latex=y_%7Bt%2Cp%7D%0) as the activation of network output [](https://www.codecogs.com/eqnedit.php?latex=p%0) at time [](https://www.codecogs.com/eqnedit.php?latex=t%0). Therefore [](https://www.codecogs.com/eqnedit.php?latex=y_%7Bt%2Cp%7D%0) is the probability that the network will output element [](https://www.codecogs.com/eqnedit.php?latex=p%20%5Cin%20A'%0) at time [](https://www.codecogs.com/eqnedit.php?latex=t%0) given that [](https://www.codecogs.com/eqnedit.php?latex=x%0) is the input sequence of length [](https://www.codecogs.com/eqnedit.php?latex=T%0). The distribution sought after [](https://www.codecogs.com/eqnedit.php?latex=Pr(%5Cpi%7Cx)%0), is the conditionally-independent distribution over the subset [](https://www.codecogs.com/eqnedit.php?latex=%7BA'%7D%5ET%0) where[](https://www.codecogs.com/eqnedit.php?latex=%7BA'%7D%5ET%0) is a set of length [](https://www.codecogs.com/eqnedit.php?latex=T%0) comprising a sequence of symbols each of which belong to the set [](https://www.codecogs.com/eqnedit.php?latex=%7BA'%7D%0) such that

[](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%20%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x%20)%20%3D%20%5Cprod_%7Bt%3D1%7D%5E%7BT%7D%20y_%7Bt%2C%5Cpi_t%7D%0) - - - (1)

From equation \ref{eqn\_ch6\_ctc02}, it is now possible to define the many-to-one mapping [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%20%3A%20%7BA'%7D%5ET%20%5Crightarrow%20A%5E%7B%5Cle%20T%7D%0). This is the mapping of a set [](https://www.codecogs.com/eqnedit.php?latex=A'%5ET%0), which indicates the paths, onto another set [](https://www.codecogs.com/eqnedit.php?latex=A%5E%7B%5Cle%20T%7D%0), that of possible labels in [](https://www.codecogs.com/eqnedit.php?latex=x%0). [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) then becomes a sequence of symbols with length less than or equal to [](https://www.codecogs.com/eqnedit.php?latex=T%0) over [](https://www.codecogs.com/eqnedit.php?latex=A%0). Note that [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) is a set containing sequential symbols belonging to the set [](https://www.codecogs.com/eqnedit.php?latex=A%0) and not. [](https://www.codecogs.com/eqnedit.php?latex=A'%0) because there is no blank symbol in [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0). This is achieved when first take out all repeated labels and then take out all the blanks from the sequence [](https://www.codecogs.com/eqnedit.php?latex=A'%5ET%0). For instance,

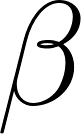
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Baligned%7D%5Cmathcal%7BB%7D(a%20-%20ab-)%20%26%3D%20aab%20%5C%5C%20%5Cmathcal%7BB%7D(-aa%20-%20-abb)%20%26%3D%20aab.%5Cend%7Baligned%7D%0) - - - (2)

The mapping obtained by [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) 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 [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) which is the labelling of [](https://www.codecogs.com/eqnedit.php?latex=l%20%5Cin%20A%5E%7B%5Cle%20T%7D%0) being a many to one of [](https://www.codecogs.com/eqnedit.php?latex=A'%5ET%0) is determined by summing over all the paths in [](https://www.codecogs.com/eqnedit.php?latex=A'%5ET%0) mapped onto it by [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0). Thus:

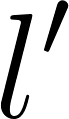
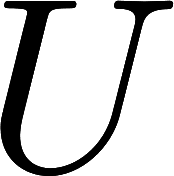
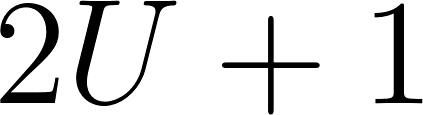
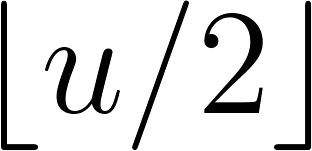
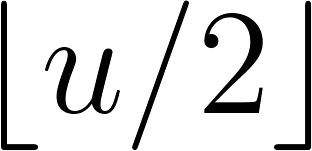
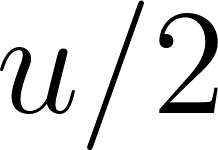
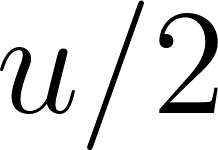
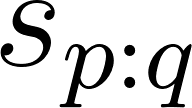
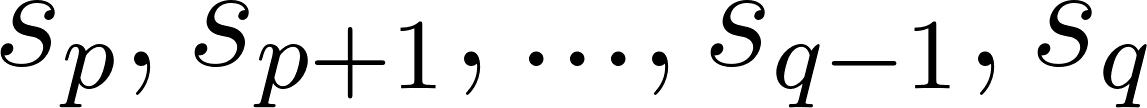
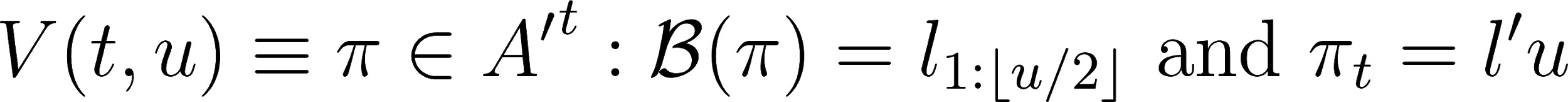
[](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%20l%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20%5Csum_%7B%5Cpi%20%5Cin%20%5Cmathcal%7BB%7D%5E%7B-1%7D(l)%7D%20%5CPr(%20%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (3)

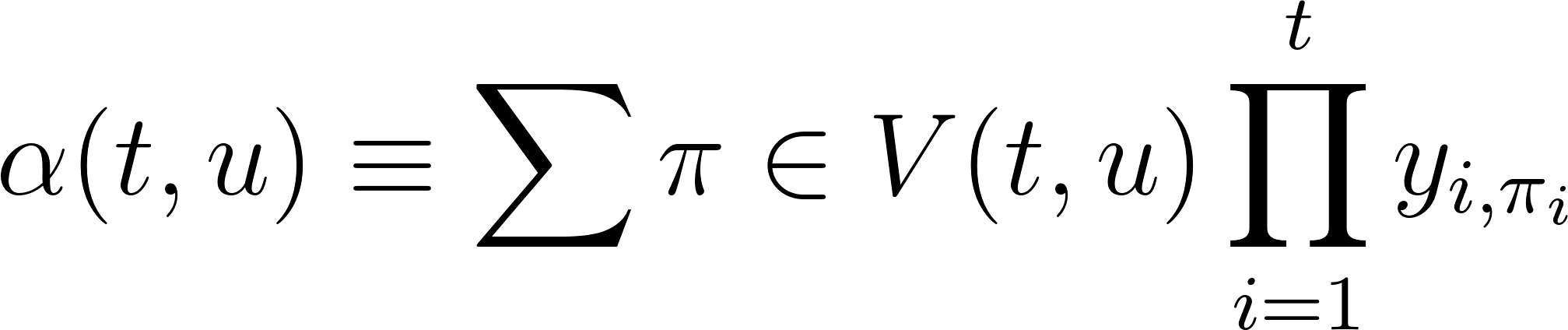
This mapping makes CTC robust to unsegmented data as it predicts all the labels where they occur and later the ‘collapsed’ sequence will be extended over the approximate period where the previous extended sequence occurred thus aligning labels to input sequences on-the-fly without knowing in advance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

### Forward-backward algorithm

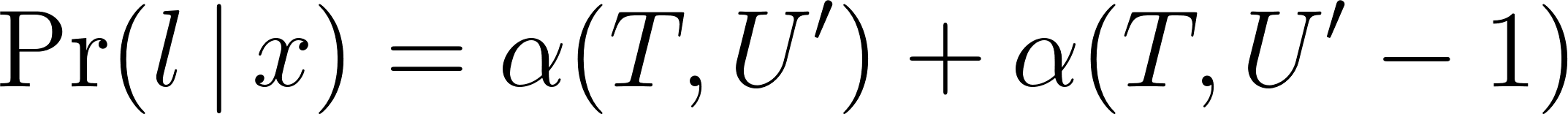
The forward-backward algorithm is used to estimate the probability of a point in the sequence as the product of all point leading up to that point from the initial state, the forward variable ([](https://www.codecogs.com/eqnedit.php?latex=%5Calpha%0)), multiplied by the probability of all the points from that state to the end of the sequence, the backward variable ([](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta%0)).

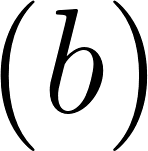
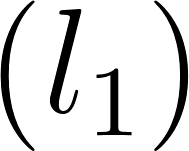
The difference between this estimation and that determined from equation (\ref{eqn\_c3\_ct03}) is the fact that the forward-backward algorithm converts equation (\ref{eqn\_c3\_ct03}) into a form that is both recursive as well as reduces the computational complexity from an otherwise intractable computation to one that is readily computable.

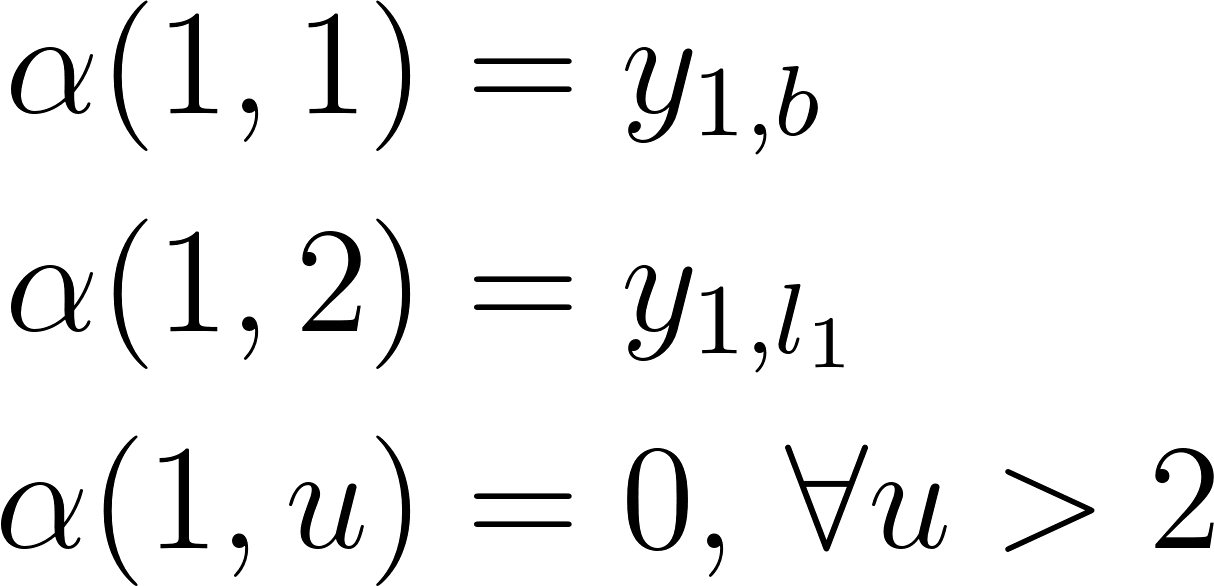
With CTC, consider a modified "label sequence" [](https://www.codecogs.com/eqnedit.php?latex=l'%0), that caters for blank characters in between regular ones [](https://www.codecogs.com/eqnedit.php?latex=l%0), as defined in [](https://www.codecogs.com/eqnedit.php?latex=A'%0). Thus, if [](http://www.texrendr.com/?eqn=U%0) is defined as the length of [](https://www.codecogs.com/eqnedit.php?latex=l%0). Then [](https://www.codecogs.com/eqnedit.php?latex=U'%0) is of length [](https://www.codecogs.com/eqnedit.php?latex=2U%20%2B%201%0). CTC therefore integrates probability distributions of transitions between blank and non-blank labels at the same time CTC calculates those transition occurring between pairs of distinct non-blank labels. The forward variable [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0) now becomes the summed probability of all length [](https://www.codecogs.com/eqnedit.php?latex=t%0) paths that are mapped by [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) onto the length [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft%20%5Clfloor%7Bu%2F2%7D%5Cright%20%5Crfloor%0) prefix of [](https://www.codecogs.com/eqnedit.php?latex=l%0). (Note, [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft%20%5Clfloor%7Bu%2F2%7D%5Cright%20%5Crfloor%0) is the floor of [](https://www.codecogs.com/eqnedit.php?latex=u%2F2%0), the greatest integer less than or equal to [](https://www.codecogs.com/eqnedit.php?latex=u%2F2%0).) For some sequence [](https://www.codecogs.com/eqnedit.php?latex=s%0), let [](https://www.codecogs.com/eqnedit.php?latex=s_%7Bp%3Aq%7D%0) denote the sub-sequence [](https://www.codecogs.com/eqnedit.php?latex=s_p%2C%20s_%7Bp%2B1%7D%2C...%2Cs_%7Bq-1%7D%2Cs_q%0), and define the set [](https://www.codecogs.com/eqnedit.php?latex=V(t%2Cu)%20%5Cequiv%20%7B%20%5Cpi%20%5Cin%20%7BA'%7D%5Et%20%3A%20%5Cmathcal%7BB%7D(%5Cpi)%20%3D%20l_%7B1%3A%5Cleft%20%5Clfloor%7Bu%2F2%7D%5Cright%20%5Crfloor%7D%20%5Ctext%7B%20and%20%7D%20%5Cpi_t%20%3D%20l'u%20%7D%0)*.* [**](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0) *then becomes*

[**](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%20%5Cequiv%20%5Csum%7B%5Cpi%20%5Cin%20V(t%2Cu)%7D%20%5Cprod_%7Bi%3D1%7D%5E%7Bt%7D%20y_%7Bi%2C%5Cpi_i%7D%0) - - - (4)

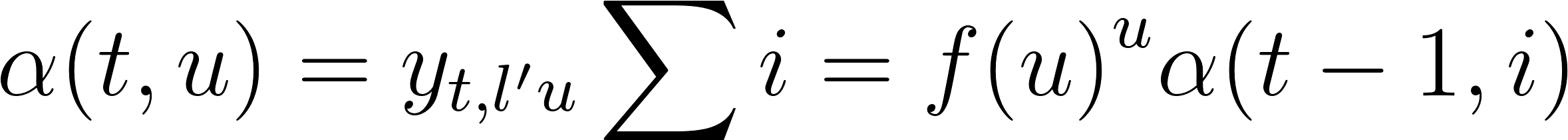
The forward variables at time [](https://www.codecogs.com/eqnedit.php?latex=t%0) is calculated recursively from the preceding values at time [](https://www.codecogs.com/eqnedit.php?latex=t%20-%201%0) and expressed as the sum of the forward variables with and without the final blank at time [](https://www.codecogs.com/eqnedit.php?latex=T%0).

[](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%20l%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20%5Calpha(T%2C%20U')%20%2B%20%5Calpha(T%2C%20U'%20-%201)%0) - - - (5)

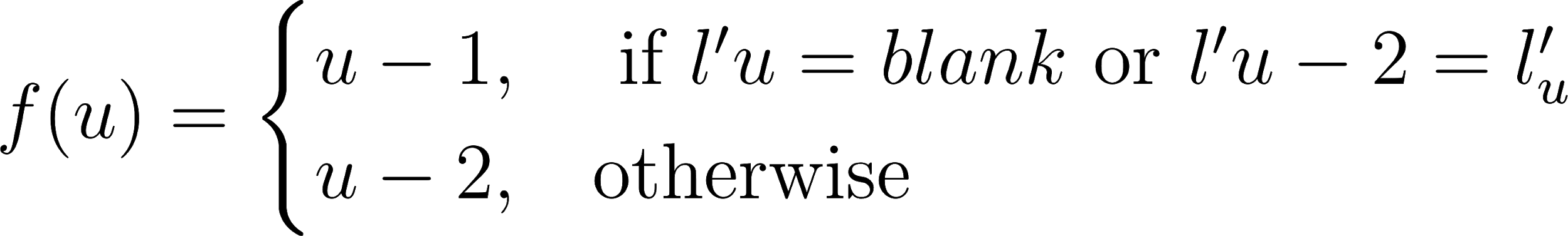
For the initial conditions, correct paths begin with a blank symbol [](https://www.codecogs.com/eqnedit.php?latex=(b)%0) and the first symbol [](https://www.codecogs.com/eqnedit.php?latex=(l_1)%0):

[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Baligned%7D%5Calpha(1%2C%201)%20%26%3D%20y_%7B1%2Cb%7D%20%5C%5C%20%5Calpha(1%2C%202)%20%26%3D%20y_%7B1%2Cl_1%7D%20%5C%5C%20%5Calpha(1%2C%20u)%20%26%3D%200%2C%20%5C%2C%20%5Cforall%20u%20%3E%202%20%5Cend%7Baligned%7D%0) - - - (6)

The forward variable then takes the following recursive form:

[](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%20%3D%20y_%7Bt%2C%20l'u%7D%20%5Csum%7Bi%20%3D%20f(u)%7D%5E%7Bu%7D%20%5Calpha(t-1%2C%20i)%0) - - - (7)

where

[](https://www.codecogs.com/eqnedit.php?latex=f(u)%20%3D%5Cbegin%7Bcases%7Du-1%2C%20%26%20%5Ctext%7B%20if%20%7D%20l'u%20%3D%20blank%20%5Ctext%7B%20or%20%7D%20l'%7Bu-2%7D%20%3D%20l'_%7Bu%7D%20%5C%5C%20u-2%2C%20%26%20%5Ctext%7Botherwise%7D%5Cend%7Bcases%7D%0) - - - (8)

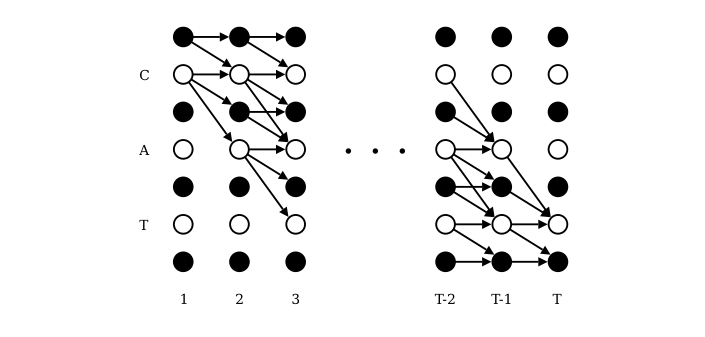
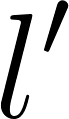
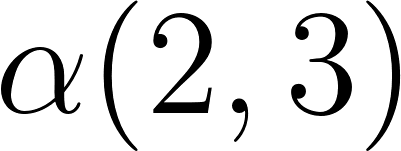
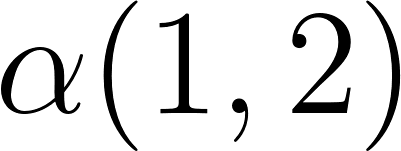
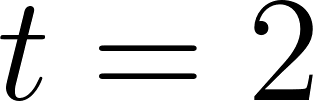
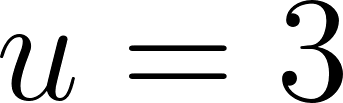
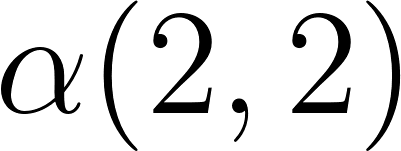
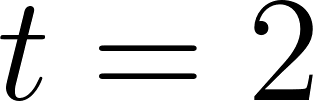
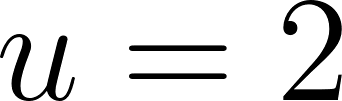
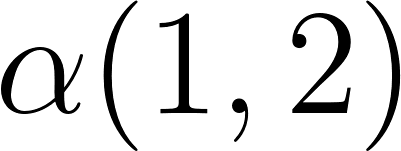
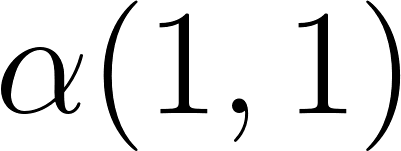
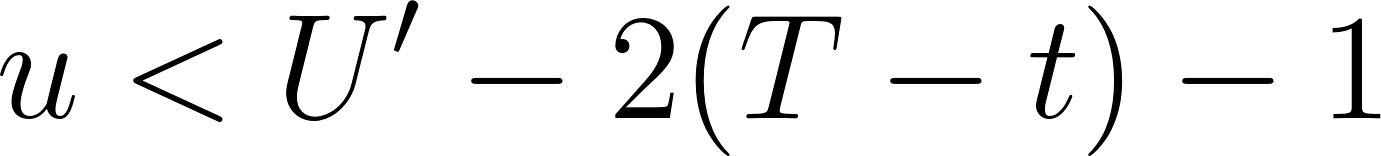
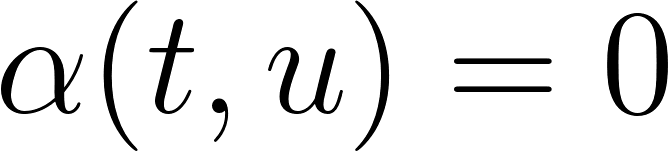
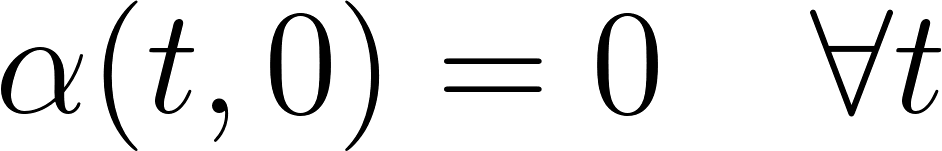
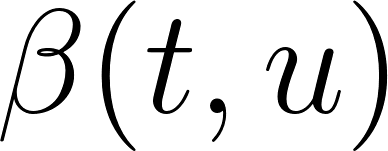
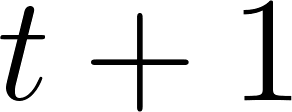
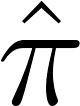


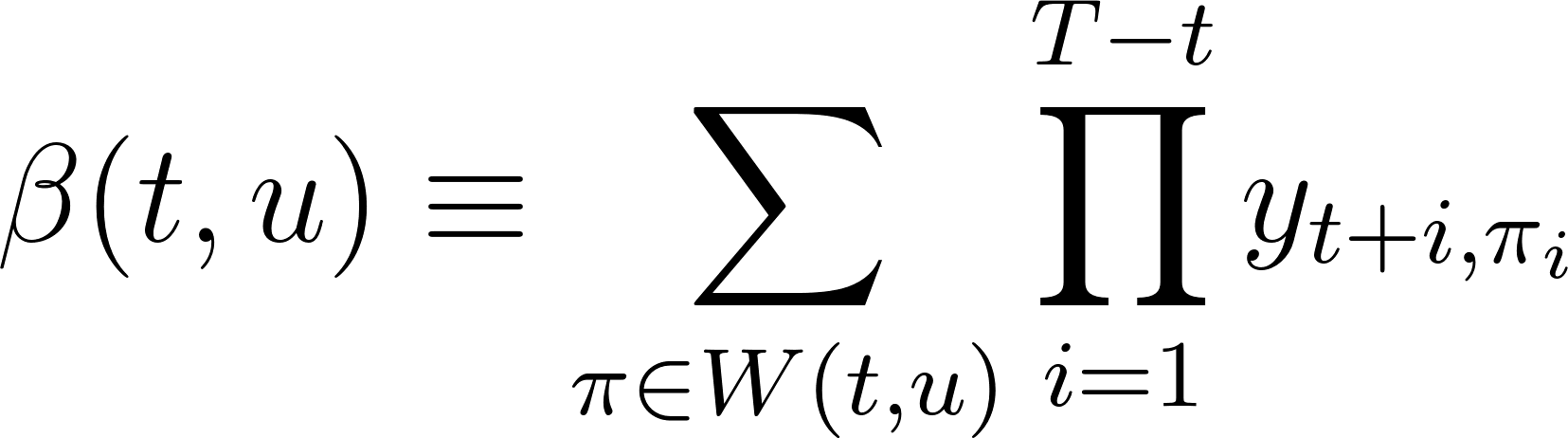
Figure 6.2: Forward-Backward Algorithm Lattice \citep{graves2006}

Figure \ref{fig\_c4\_lattice} expresses the recurrence relation for [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%0). While [](https://www.codecogs.com/eqnedit.php?latex=t%0) is expressed on the [](https://www.codecogs.com/eqnedit.php?latex=x%0) axis, [](https://www.codecogs.com/eqnedit.php?latex=u%0) is illustrated on the [](https://www.codecogs.com/eqnedit.php?latex=y%0) 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 [](https://www.codecogs.com/eqnedit.php?latex=l'%0). The sequential graph constructed from this 2-dimensional matrix show computational dependencies between sequential pairs of the recurrence relation for [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0). Therefore, the value [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(2%2C3)%0), formed from [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C2)%0), corresponds to the [](https://www.codecogs.com/eqnedit.php?latex=blank%0) symbol at [](https://www.codecogs.com/eqnedit.php?latex=t%3D2%0) and [](https://www.codecogs.com/eqnedit.php?latex=u%3D3%0), is . Also, [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(2%2C2)%0), equivalent to the symbol [](https://www.codecogs.com/eqnedit.php?latex=c%0) at [](https://www.codecogs.com/eqnedit.php?latex=t%3D2%0) and [](https://www.codecogs.com/eqnedit.php?latex=u%3D2%0), is gotten from [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C2)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C1)%0). Note that there are not enough time steps when [](https://www.codecogs.com/eqnedit.php?latex=u%20%3C%20U'-2(T-t)-1%0), therefore, [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%3D0%0).

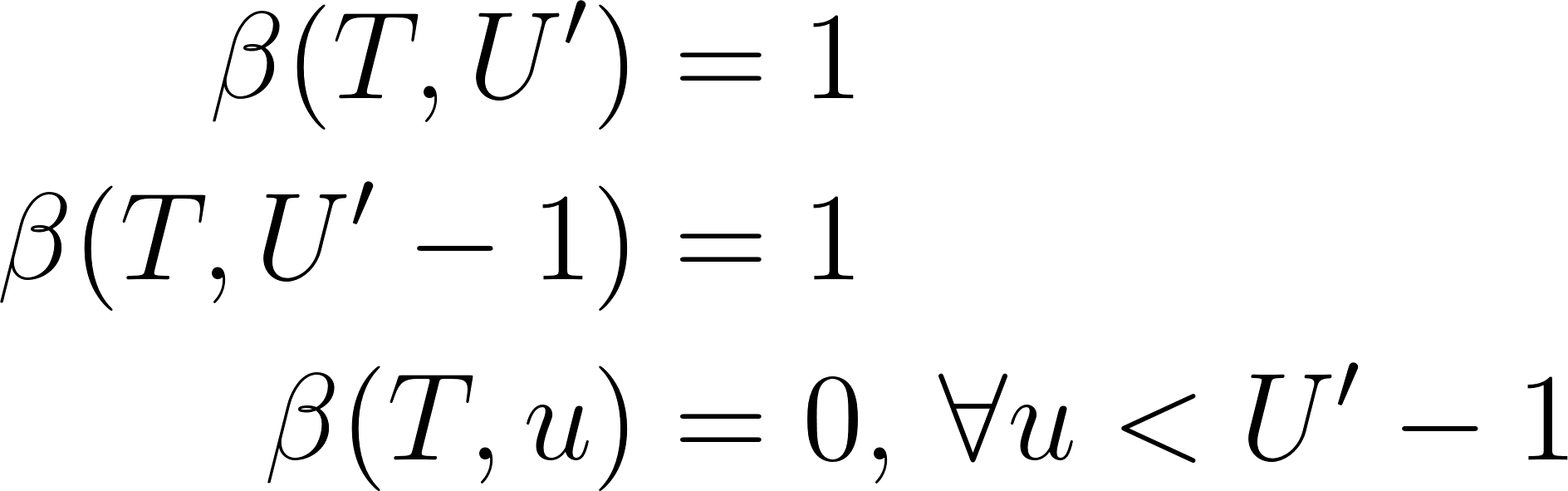
Also note the boundary condition

[](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%200)%20%3D%200%20%5Cquad%20%5Cforall%20t%0) - - - (9)

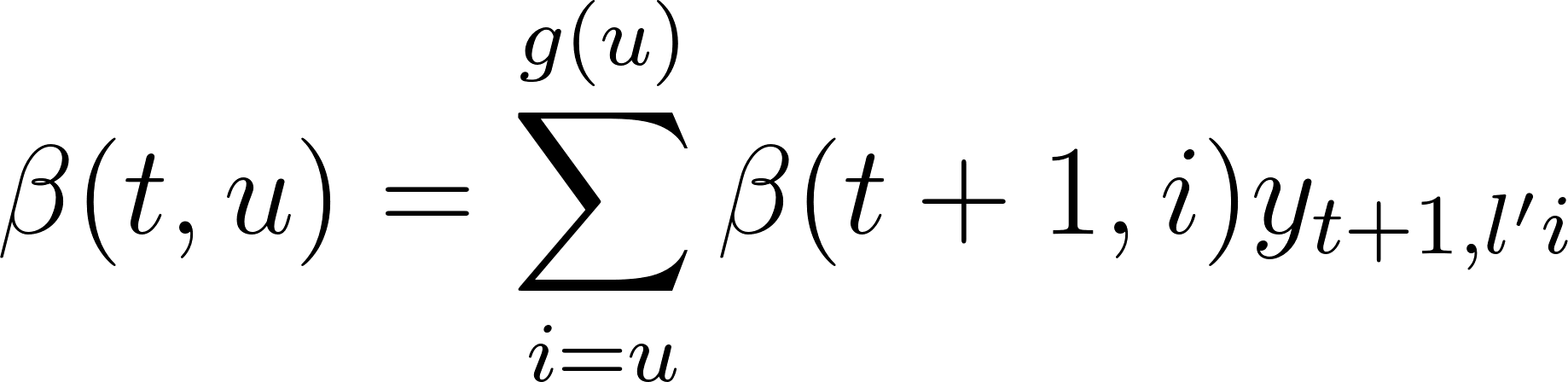
The backward variable [](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2Cu)%0) is built similarly to the forward variable. Rather than moving from the start of the sequence to [](https://www.codecogs.com/eqnedit.php?latex=t%0) we define the path starting at [](https://www.codecogs.com/eqnedit.php?latex=t%20%2B%201%0) that completes the sequence at [](https://www.codecogs.com/eqnedit.php?latex=T%0) when appended the path [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7B%5Cpi%7D%0) that generates [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0). The following is defined [](https://www.codecogs.com/eqnedit.php?latex=W(t%2Cu)%20%5Cequiv%20%7B%20%5Cpi%20%5Cin%20%7BA'%7D%5E%7BT-t%7D%20%3A%20%5Cmathcal%7BB%7D(%5Chat%7B%5Cpi%7D%20%2B%20%5Cpi)%20%3D%20l%20%5C%2C%20%5C%2C%20%5Cforall%20%5Chat%7B%5Cpi%7D%20%5Cin%20V(t%2Cu)%20%7D%0). Then

[](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2Cu)%20%5Cequiv%20%5Csum_%7B%5Cpi%20%5Cin%20W(t%2Cu)%7D%20%5Cprod_%7Bi%3D1%7D%5E%7BT%20-%20t%7D%20y_%7Bt%20%2B%20i%2C%5Cpi_i%7D%0) - - - (10)

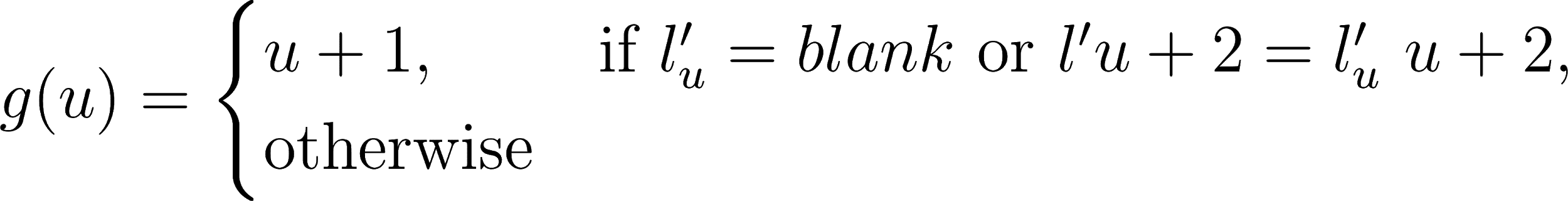
The backward variable is therefore equivalently initialised as thus

[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Baligned%7D%20%5Cbeta(T%2C%20U')%20%26%3D%201%20%5C%5C%20%5Cbeta(T%2C%20U'%20-%201)%20%26%3D%201%20%5C%5C%20%5Cbeta(T%2C%20u)%20%26%3D%200%2C%20%5C%2C%20%5Cforall%20u%20%3C%20U'%20-%201%20%5Cend%7Baligned%7D%0) - - - (11)

The recursion rule is defined as follows:

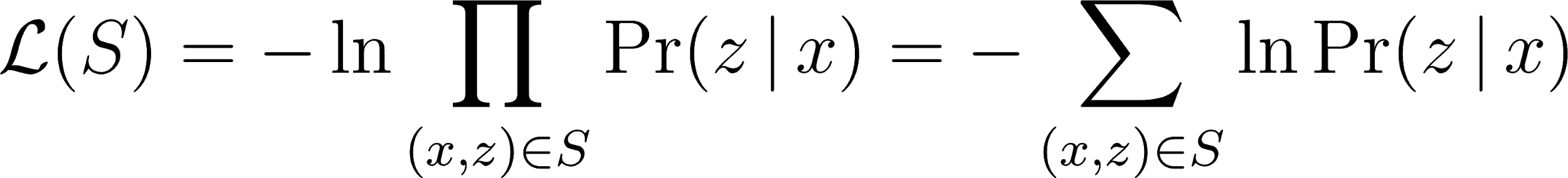
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2C%20u)%20%3D%20%5Csum_%7Bi%20%3D%20u%7D%5E%7Bg(u)%7D%20%5Cbeta(t%2B1%2C%20i)%20y_%7Bt%2B1%2C%20l'i%7D%0) - - - (12)

*similarly,*

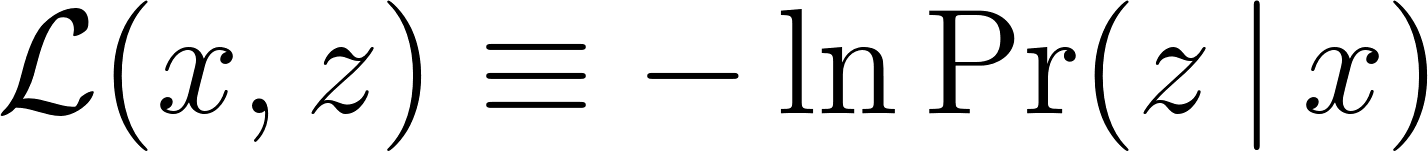
[**](https://www.codecogs.com/eqnedit.php?latex=g(u)%20%3D%20%5Cbegin%7Bcases%7D%20u%20%2B%201%2C%26%20%5Ctext%7Bif%20%7D%20l'_u%20%3D%20blank%20%5Ctext%7B%20or%20%7D%20l'%7Bu%2B2%7D%20%3D%20l'_%7Bu%7D%20%5C%20u%20%2B%202%2C%26%20%5Ctext%7Botherwise%7D%20%5Cend%7Bcases%7D%0) - - - (13)

### Loss Function

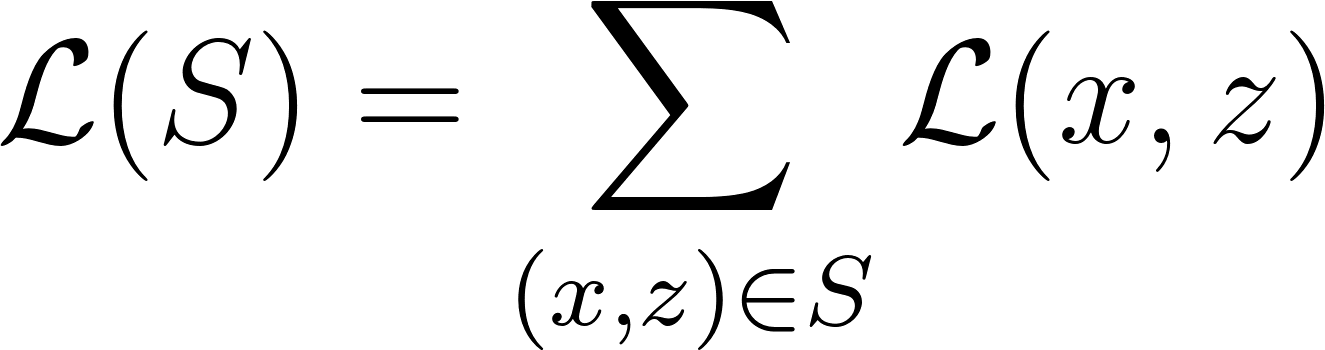
The cross entropy error is a loss function used to measure accuracy of probabilistic measures. It is calculated as the negative log probability of a likelihood measure. The CTC loss function [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%0) uses the cross entropy loss function of and is defined as the cross entropy error of correctly labeling all the training samples in some training set S:

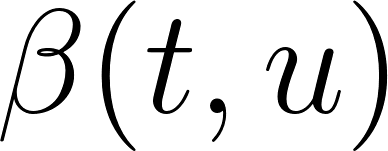
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%20%3D%20-%20%5Cln%20%5Cprod_%7B(x%2Cz)%20%5Cin%20S%7D%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20-%20%5Csum_%7B(x%2Cz)%20%5Cin%20S%7D%20%5Cln%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (14)

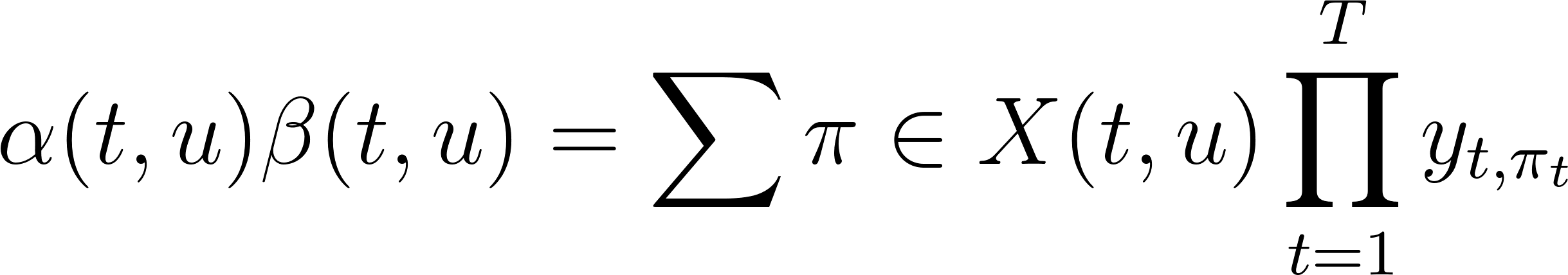
where [](https://www.codecogs.com/eqnedit.php?latex=z%0) is the output label and [](https://www.codecogs.com/eqnedit.php?latex=x%0) is the input sequence. Since [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%0) in equation \ref{eqn\_c3\_ctc11} is differentiable, this loss function can be back propagated to the softmax layer in the BiRNN configuration discussed in section \ref{deepspeech}.

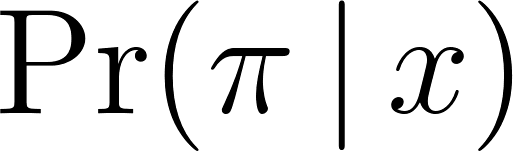
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(x%2Cz)%20%5Cequiv%20-%20%5Cln%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (15)

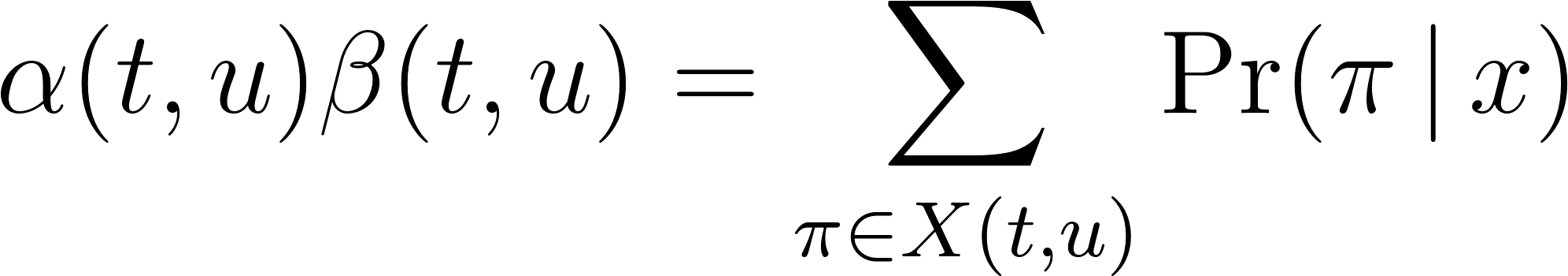
and therefore

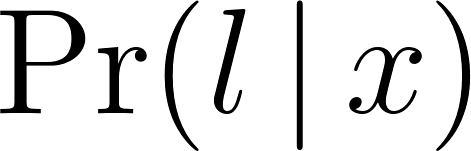
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%20%3D%20%5Csum_%7B(x%2Cz)%20%5Cin%20S%7D%20%5Cmathcal%7BL%7D(x%2Cz)%0) - - - (16)

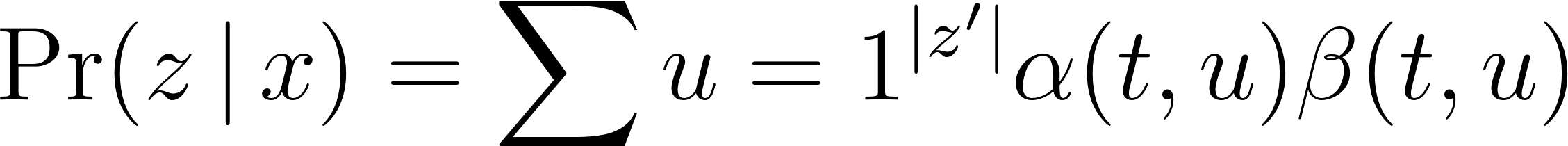
From the definition of the forward and backward variables [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2C%20u)%0), we also establish that [](https://www.codecogs.com/eqnedit.php?latex=X(t%2Cu)%20%5Cequiv%20%5C%7B%20%5Cpi%20%5Cin%20%7BA'%7D%5ET%20%3A%20%5Cmathcal%7BB%7D(%5Cpi)%20%3D%20z%2C%20%5Cpi_t%20%3D%20%7Bz'%7Du%20%5C%7D%0)*, such that*

[**](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%20%3D%20%5Csum%7B%5Cpi%20%5Cin%20X(t%2Cu)%7D%20%5Cprod_%7Bt%20%3D%201%7D%5E%7BT%7D%20y_%7Bt%2C%20%5Cpi_t%7D%0) - - - (17)

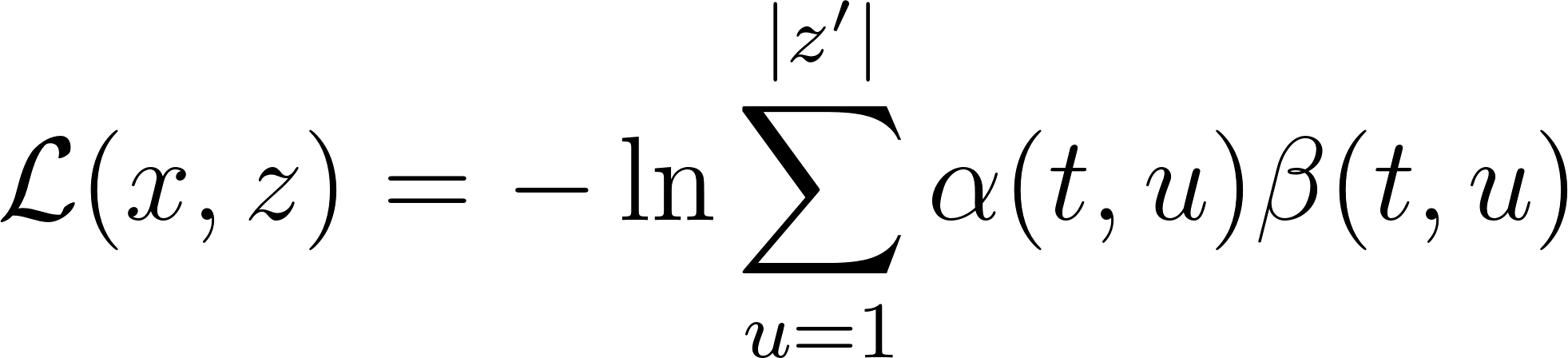
then substituting [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x)%0) from the expression in equation \ref{eqn\_c3\_ctc01}, we have

[](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%20%3D%20%5Csum_%7B%5Cpi%20%5Cin%20X(t%2Cu)%7D%20%5CPr(%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (18)

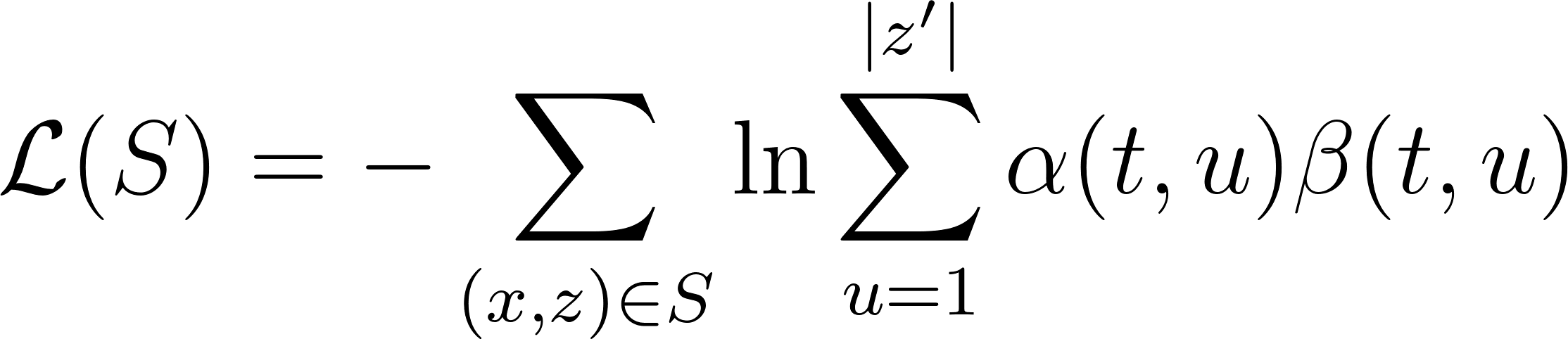
Also observe that [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(l%20%5C%2C%20%7C%20%5C%2C%20x)%0) is equivalent to the total probability [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0). Paths going through [](https://www.codecogs.com/eqnedit.php?latex=z'u%0) *at time* [**](https://www.codecogs.com/eqnedit.php?latex=t%0) *can be obtained as summed over all* [**](https://www.codecogs.com/eqnedit.php?latex=u%0) *to get*

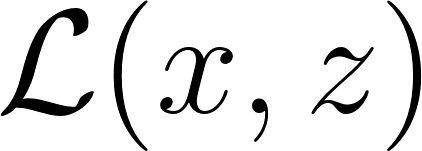
[**](https://www.codecogs.com/eqnedit.php?latex=%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20%5Csum%7Bu%20%3D%201%7D%5E%7B%7Cz'%7C%7D%20%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%0) *- - -* (19)

Thus a sample loss is determined by

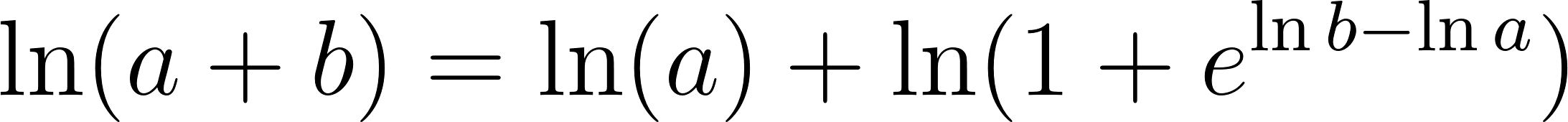
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(x%2C%20z)%20%3D%20-%20%5Cln%20%5Csum_%7Bu%20%3D%201%7D%5E%7B%7Cz'%7C%7D%20%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%0) - - - (20)

and therefore the overall loss is given by

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%20%3D%20-%5Csum_%7B(x%2Cz)%20%5Cin%20S%7D%20%5Cln%20%5Csum_%7Bu%20%3D%201%7D%5E%7B%7Cz'%7C%7D%20%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%0) - - - (21)

In the model described in this work, the gradient [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(x%2C%20z)%0) 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

\begin{equation}

[](https://www.codecogs.com/eqnedit.php?latex=%5Cln(a%20%2B%20b)%20%3D%20%5Cln(a)%20%2B%20%5Cln(1%20%2B%20e%5E%7B%5Cln%20b%20-%20%5Cln%20a%7D)%0) - - - (22)

# 

## Chapter Summary

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

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

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

A special algorithm, the Connectionist Temporal Classification (CTC) algorithm, also employed by the BiRNN is described in this Chapter. The CTC algorithm overcomes the challenge of a character-based speech model when considering misaligned nature of audio data between specific points in the audio that correspond to equivalent characters in the transcription. In addition, it is also discussed how the CTC algorithm utilises the forward-backward algorithm to perform classification. In a later Chapter \ref{ch6\_speech}, 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: The Scattering Transform layer

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 \cite{mallat2016understanding}. 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 \citep{peddinti2014deep,zeghidour2016deep,anden2011multiscale,sainath2014deep} 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 \cite{sainath2014deep} 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.

\section{Fourier transform}\label{c4\_fourier}

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 \ref{fig\_4\_1\_fourier\_eqn}).

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=7cm]{thesis/images/fourier.png}\\

\caption{Fourier Equation} \label{fig\_4\_1\_fourier\_eqn}

\end{figure}

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

\begin{equation}

a\_k=\frac{1}{T}\int\_{-T/2}^{T/2}x(t)e^{\left(-j\frac{2\pi kt}{T}\right)}

\label{eqn\_c4\_fourier01}

\end{equation}

Then, the corresponding synthesis equation is given by

\begin{equation}

x(t)=\sum\_{k=-\infty}^{\infty}a\_ke^{\left(j\frac{2\pi kt}{T}\right)}

\label{eqn\_c4\_fourier02}

\end{equation}

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 \ref{fig\_4\_2\_spectral}), 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 \ref{fig\_4\_2\_spectral}, the energies are concentrated at frequencies of about 100, 150 and 400 hertz.

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=7cm]{thesis/images/spectral.png}\\

\caption{Sample Spectrogram} \cite{cwt\_lecture}\label{fig\_4\_2\_spectral}

\end{figure}

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 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.

\section{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 \ref{fig\_4\_2\_tftile}

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=14cm]{thesis/images/tftile}\\

\caption{Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) (c) Wavelet transform}\label{fig\_4\_2\_tftile}

\end{figure}

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 fundamental representation of the continuous wavelet function is:

\begin{equation}

C(a,b)=\int f(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt\label{eqn\_c4\_wavelet01}

\end{equation}

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.

\section{Discrete and Fast wavelet transform}

Synthesis and analysis equations (\ref{eqn\_c4\_fourier02} and \ref{eqn\_c4\_fourier01}) 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 (\ref{eqn\_c4\_fourier02} and \ref{eqn\_c4\_fourier01}) now become

\begin{equation}

x(t)=\sum\_{k}a\_k\phi\_k

\label{eqn\_c4\_dwt02}

\end{equation}

\begin{equation}

a\_k=\int x(t)\tilde{\phi}\_k(t)

\label{eqn\_c4\_dwt01}

\end{equation}

With respect to scaling and shifting variables of continuous wavelet transforms in equation (\ref{eqn\_c4\_wavelet01}), a similar linear combination transformation can be applied by constructing orthonormal bases parameters, referred to as scaling ($\phi$) and translating ($\psi$) functions. For example, a simple Haar mother wavelet transform associated with a delta function, it is seen that:

\begin{equation}

\phi\_{j,k}(t)=2^{j/2}\phi(2^jt-k)

\label{eqn\_c4\_dwt03}

\end{equation}

\begin{equation}

\psi\_{j,k}(t)=2^{j/2}\psi(2^jt-k)

\label{eqn\_c4\_dwt04}

\end{equation}

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

\begin{equation}

\phi(t)=h\_\phi[n]\sqrt{2}\phi(2t-n)

\label{eqn\_c4\_dwt05}

\end{equation}

\begin{equation}

\psi(t)=h\_\phi[n]\sqrt{2}\psi(2t-n)

\label{eqn\_c4\_dwt06}

\end{equation}

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

\begin{equation}

f[n]=\frac{1}{\sqrt{M}}\sum\_kW\_\phi[j\_0,k]\phi\_{j\_0,k}[n]+\frac{1}{\sqrt{M}}\sum\_{j=j\_0}^\infty\sum\_kW\_\psi[j,k]\psi\_{j,k}[n]

\label{eqn\_c4\_dwt07}

\end{equation}

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 $\{\psi\_{(j,k)\in\mathbf{Z}^2,j\ge j\_0}\}$ are orthogonal to each other. We can simply take the inner product to obtain the wavelet coefficients.

\begin{equation}

W\_\phi[j\_0,k]=\frac{1}{\sqrt{M}}\sum\_nf[n]\phi\_{j\_0,k}[n]

\label{eqn\_c4\_dwt08}

\end{equation}

\begin{equation}

W\_\psi[j,k]=\frac{1}{\sqrt{M}}\sum\_nf[n]\psi\_{j,k}[n] \quad j\ge j\_0

\label{eqn\_c4\_dwt09}

\end{equation}

Equation (\ref{eqn\_c4\_dwt08}) is called approximation coefficient while (\ref{eqn\_c4\_dwt09}) 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 \citep{mallat1989theory}.

\begin{equation}

W\_\phi[j,k]=h\_\phi[-n]\ast W\_\phi[j+1,n]|\_{n=2k, k\ge 0}

\label{eqn\_c4\_dwt10}

\end{equation}

\begin{equation}

W\_\psi[j\_0,k]=h\_\psi[-n]\ast W\_\phi[j+1,n]|\_{n=2k, k\ge 0}

\label{eqn\_c4\_dwt11}

\end{equation}

For analysis of the Haar wavelet and the derivation of equations (\ref{eqn\_c4\_dwt10} and \ref{eqn\_c4\_dwt11}) see appendix \ref{app01}.

## Mel Filter banks

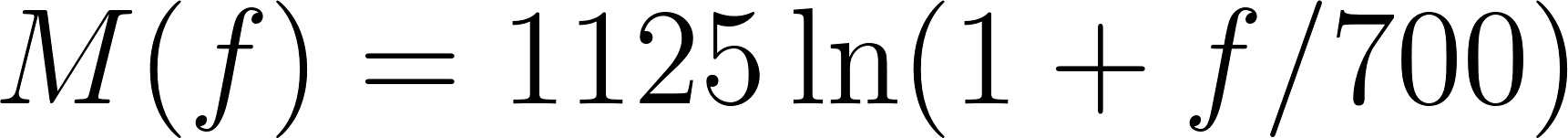
The Fourier and wavelet transform are general means of extracting information from continuous signals using the frequency domain and in the case of the Wavelet transform using both time and frequency domain. The objective in machine learning, however, is to extract patterns from the derived information. 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 lossless encoding of speech signals relevant for speech recognition.

The Mel filter banks form the basis of the Mel Frequency Cepstral Coefficients (MFCCs) described by \citep{davis1980comparison}. 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), \citep{mcloughlin2009applied, dines2010measuring}. The following paragraphs describe how the mel filters are derived.

The Mel scale as described by \cite{stevens1937scale} 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 nonlinear as the human ear processes sound non-linearly both in frequency as well as amplitude.

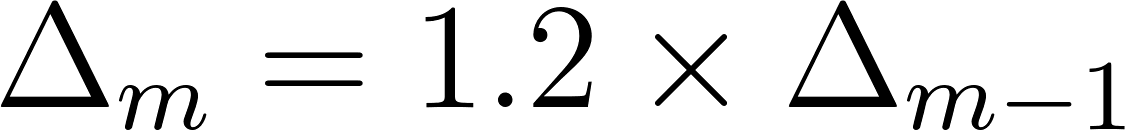
For the case of frequency, the human ear can discriminate lower frequencies more accurately than the higher frequencies. The Mel scale model this behaviour by utilising frequency bins. The frequency bin ranges are narrow at low frequencies and become wide in higher frequencies. In the case of the speech 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 \citep{becchetti1999behaviour}.

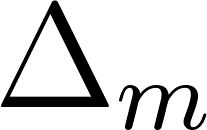
The minimum frequency number of bins used for the Mel scale is 26 bins. In order to determine the frequency ranges we use the following formula to convert between the Mel scale and the regular frequency scale:

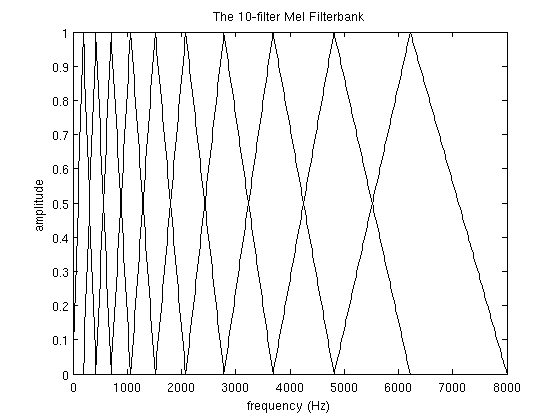
[](https://www.codecogs.com/eqnedit.php?latex=M(f)%3D1125%5Cln(1%2Bf%2F700)%0) - - - (1)

[](https://www.codecogs.com/eqnedit.php?latex=M%5E%7B-1%7D(m)%3D700%5Cexp(m%2F1125)-1%0) - - - (2)

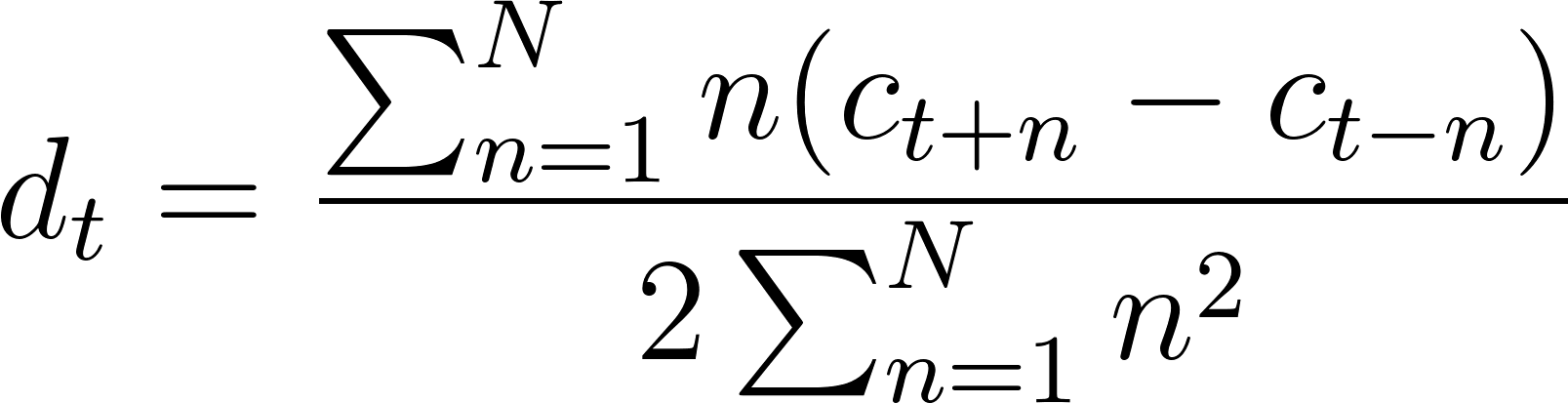
A simple approximation for the Mel scale is obtained by applying linear scale for the first ten filters and for the first 1kHz of the speech frequency range then applying the following formula for the rest \citep{becchetti1999behaviour}:

[](https://www.codecogs.com/eqnedit.php?latex=%5CDelta_m%3D1.2%5Ctimes%20%5CDelta_%7Bm-1%7D%0) - - - (3)

where m is the frequency bin index and [](https://www.codecogs.com/eqnedit.php?latex=%5CDelta_m%0) is the frequency range between the start and end frequencies for the m-th bin. The resulting filters are overlapping filters shown in the figure below.



For speech recognition, we compute a statistical value or coefficient for each Mel frequency bin from the Inverse Discrete Fourier Transform (IDFT) of the Mel filters. The coefficients are also concatenated with their delta and delta-delta values. The delta and delta-delta values are determined from the following equation:

[](https://www.codecogs.com/eqnedit.php?latex=d_t%3D%5Cfrac%7B%5Csum_%7Bn%3D1%7D%5ENn(c_%7Bt%2Bn%7D-c_%7Bt-n%7D)%7D%7B2%5Csum_%7Bn%3D1%7D%5ENn%5E2%7D%0) - - - (4)

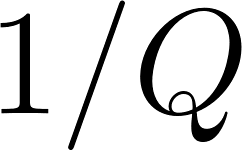
where [](https://www.codecogs.com/eqnedit.php?latex=c_x%0) is the x-th coefficient and 2n is the delta range which is usually 2-4. The delta values are first order derived coefficients obtained from the original Mel filter coefficients while the delta-delta values are second-order derived coefficients 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, decorelated signals optimise algorithm computation efficiency involving matrix operations such that rather than using full covariance 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\citep{gales2008application}.

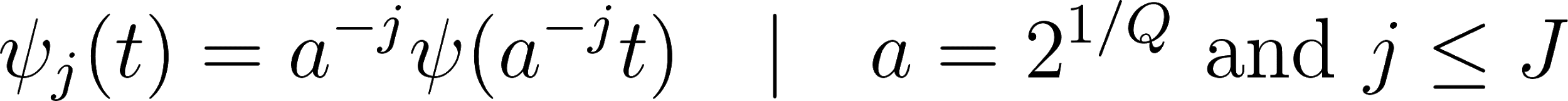
As an attempt for MFCCs to incorporate dynamic frequency changes of the signal, the deltas and the delta-deltas are obtained from the coefficient computation in equation (4). However, it is worthy to note that only the first 13 of the coefficients and the resulting dynamic coefficients are used as speech features as it is observed that higher frequency dynamic coefficients rather degrade ASR performance \citep{gales2008application}.

## Deep Scattering Spectrum

Scattering wavelets are interpreted from \acrfull{mfsc} which are the Mel filters without applying the \acrshort{idft} or the \acrshort{dct}.In this section reference is made to \citep{anden2011multiscale, anden2014deep, zeghidour2016deep} for the scattering wavelet derivation.

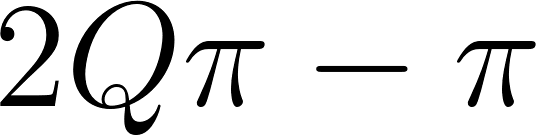
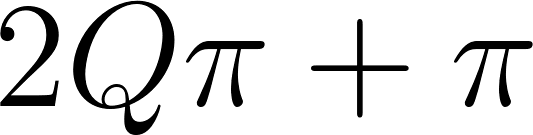
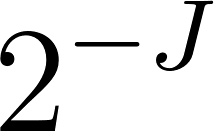
The Mel scale can be interpreted as dilations of a wavelet. For the case of the MFSC, it lacks the ability to capture non-stationary structures when outside the frame window. Such MFSC filters are constructed by dilating the wavelet by an octave bandwidth of [](https://www.codecogs.com/eqnedit.php?latex=1%2FQ%0) as follows

\begin{equation}

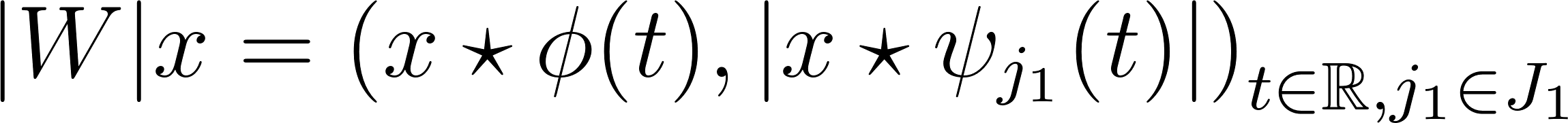
[](https://www.codecogs.com/eqnedit.php?latex=%5Cpsi_j(t)%3Da%5E%7B-j%7D%5Cpsi(a%5E%7B-j%7Dt)%5Cquad%7C%5Cquad%20a%3D2%5E%7B1%2FQ%7D%5Ctext%7B%20and%20%7Dj%5Cle%20J%0)

\label{eqn\_c4\_deep00}

\end{equation}

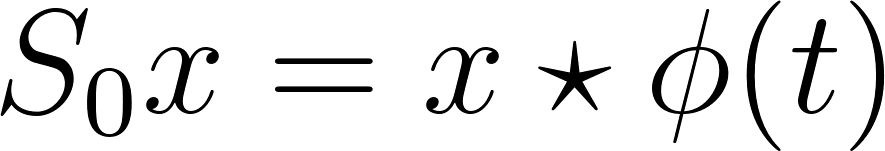
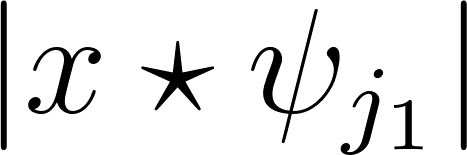
The transfer function of the constructed filter approximately ranges between [](https://www.codecogs.com/eqnedit.php?latex=2Q%5Cpi-%5Cpi%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5C2Q%5Cpi%2B%5Cpi%0). For low frequencies below [](https://www.codecogs.com/eqnedit.php?latex=2%5CpiQa%5E%7B-J%7D%0) 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 [](https://www.codecogs.com/eqnedit.php?latex=x%0) we define the zeroth order Scattering transform as follows:

\begin{equation}

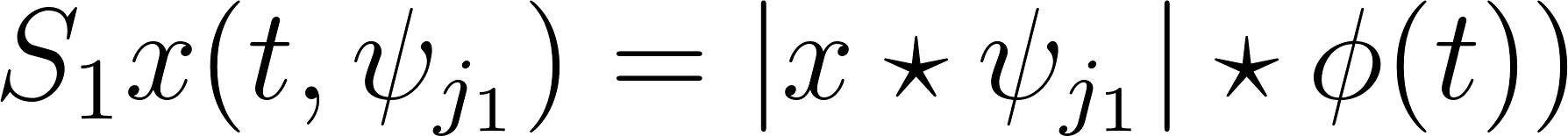
[](https://www.codecogs.com/eqnedit.php?latex=%7CW%7Cx%3D%5Cleft(x%5Cstar%5Cphi(t)%2C%7Cx%5Cstar%5Cpsi_%7Bj_1%7D(t)%7C%5Cright)_%7Bt%5Cin%5Cmathbb%7BR%7D%2Cj_1%5Cin%20J_1%7D%0)

\label{eqn\_c4\_dss01}

\end{equation}

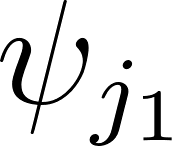
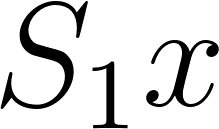
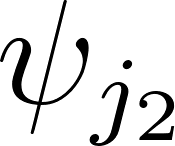
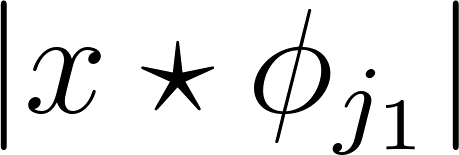
However, the difference between the \acrshort{mfsc} and the \acrfull{dsn} is that the \acrshort{mfsc} is a shallow architecture while the \acrshort{dsn} is a deep architecture. Therefore, it is observed that time-averaging of the low-pass filter loses information contained in the high frequencies. Nevertheless, since the wavelet transform is invertible, the \acrshort{DSN} is able to go deeper by applying the next level scattering operation. The transform is therefore reapplied over [](https://www.codecogs.com/eqnedit.php?latex=S_0x%3Dx%5Cstar%5Cphi(t)%0) on the residue wavelet coefficients [](https://www.codecogs.com/eqnedit.php?latex=%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%0) and the process of time averaging to obtain invariant contracted envelopes reapplied. The resulting first order scattering coefficient were obtained from

\begin{equation}

[](https://www.codecogs.com/eqnedit.php?latex=S_1x(t%2C%5Cpsi_%7Bj_1%7D)%3D%7Cx%20%5Cstar%5Cpsi_%7Bj_1%7D%7C%20%5Cstar%5Cphi(t)%5Clabel%7Beqn_c4_dss03%7D)%0)

\label{eqn\_c4\_dss02}

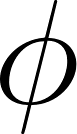
\end{equation}

It is shown in \cite{anden2014deep} that if the wavelets [](https://www.codecogs.com/eqnedit.php?latex=%5Cpsi_%7Bj_1%7D%0) have the same frequency resolution as the standard Mel-filters, then the [](https://www.codecogs.com/eqnedit.php?latex=S_1x%0) 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 \acrshort{DSN}. This can be observed when first order wavelets [](https://www.codecogs.com/eqnedit.php?latex=%5Cpsi_%7Bj_2%7D%0) are applied to \acrshort{DSN} coefficients [](https://www.codecogs.com/eqnedit.php?latex=%7Cx%5Cstar%5Cphi_%7Bj_1%7D%7C%0):

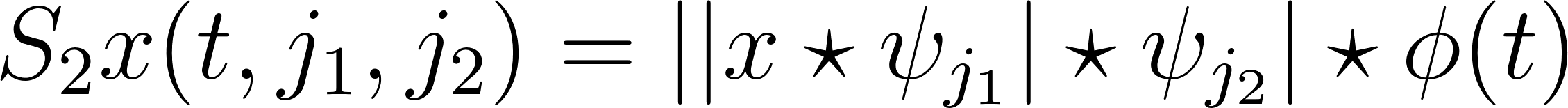
\begin{equation}

[](https://www.codecogs.com/eqnedit.php?latex=%7CW_2%7C%7Cx%5Cstar%5Cphi_%7Bj_1%7D%7C%3D%5Cleft(%7Cx%5Cstar%5Cpsi_j_1%7D%7C%5Cstar%5Cphi%2C%7C%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cpsi_%7Bj_2%7D%7C%5Cright)_%7Bj_2%5Cin%20J_2%7D%0)

\label{eqn\_c4\_dss04})\end{equation}

The second order \acrshort{DSN} coefficients also requires time averaging to derive local invariance stability, hence the resulting coefficients are averaged again with a low-pass filter [](https://www.codecogs.com/eqnedit.php?latex=%5Cphi%0) and the final second order \acrshort{DSN} scattering parameters are obtained.

\begin{equation}

[](https://www.codecogs.com/eqnedit.php?latex=S_2x(t%2Cj_1%2Cj_2)%3D%7C%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cpsi_%7Bj_2%7D%7C%5Cstar%5Cphi(t)%0)

\label{eqn\_c4\_dss05})\end{equation}

Figure \ref{fig\_c4\_scatter} 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. \cite{anden2014deep} shows that speech signals can be analysed using the first two \acrshort{dsn} layers and the coefficients obtained are generally stable to deformation and translation invariant while possessing better discriminating features than MFCCs.

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=14cm]{thesis/images/scatter}\\

\caption{Scattering network - 2 layers deep} \cite{anden2011multiscale}\label{fig\_c4\_scatter}

\end{figure}

# 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 \ref{ch3RNN}.

## 6.1 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 on-line 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, \citep{geron2019hands}, the dataset 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 overfitting 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 hyperparameters. 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 hyperparameter 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.

## 6.2 GRU RNN Architecture

The modified LSTM RNN known as the Gated Recurrent Unit (GRU) discussed in Chapter \ref{ch3RNN} 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 \citep{cho2014learning}.

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 \citep{goodfellow2013multi}, it was shown that increasing the number of nodes in a neural network will lead to overfitting, 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 hyperparameter and the number of recurrent connections where each connection represents a sequenced time step.

Another hyperparameter 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 overfitting using the dropout \citep{srivastava2014dropout} method which has been shown to be effective for regularising deep neural networks. The hyperparameter 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.

## 6.3 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 \cite{Heafield-estimate} estimates as a baseline model. Character-based perplexity measurements were used to compare the character-based models and a conversion factor based on \citep{} 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 \ref{tab6\_1:LMX}.

Table \label{tab6\_1:LMX}

|  |  |  |  |
| --- | --- | --- | --- |
| Language Model | Training time | Perplexity | Epochs |
| GRU RNN 3-layer model (CPU training) | ~ 2 days | 30.920 | 75 |
| 5-gram with Keysner Smoothing and interpolation | ~ 5 minutes | 238.72 | NA |
| GRU RNN single-layer model (GPU training) | ~ 5 hours | 2.35x1013 | 120 |
| GRU RNN 3-layer model (GPU cloud-training) | ~ 2 hours | 30.920 | 75 |

## 6.4 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 \ref{fig\_ch7\_00losses}. 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 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.

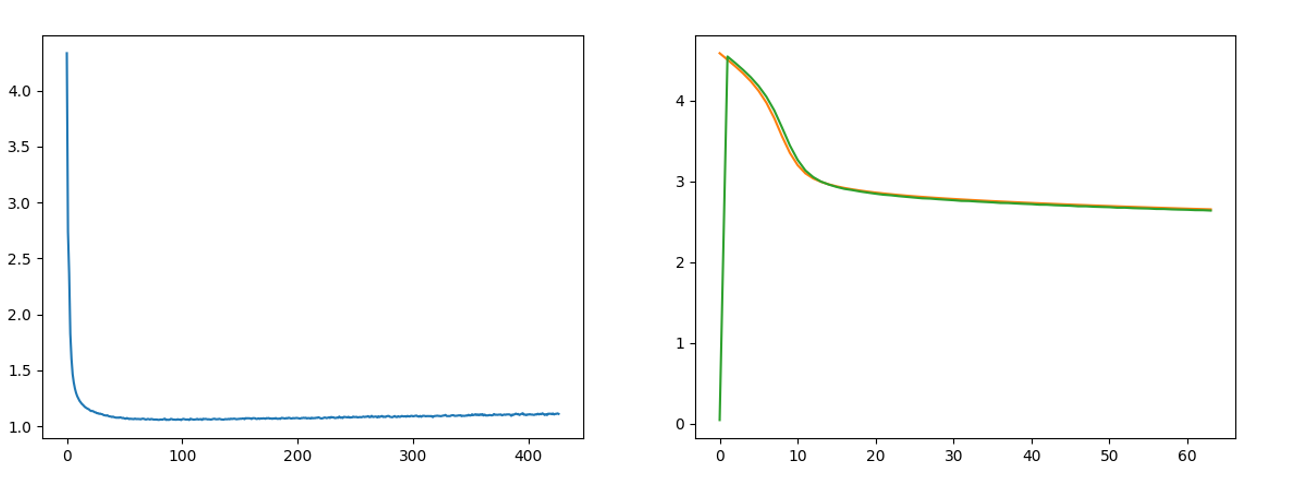


Figure 6.1 Loss curves for a. 3-Layer GRU and b) Single-Layer RNN

# 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 dataset and measures how probable the test dataset is. Perplexity is a relative measure given by the formula:

# - - - (6)

# - - - (7)

# Where 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.

## - - - (8)

## However, our base-line language model is a 5-gram word-based language model. The way to compare word-based perplexity to character based perplexity is by applying a conversion factor as discussed by \cite{hwang2017character}

## 

## Table 1 shows the Results of the Perplexity model of the GRU Wakirike Language model and an equivalent 5-gram Language model with interpolation and Keysner smoothing [(Chen & Goodman, 1999)](https://paperpile.com/c/GaEMMA/Y8KI) for various lengths of the held-out data.

## 6.5 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 not however learn anything having a very high perplexity heading towards infinity. The paragraphs below demonstrate output generations from the single layer and 3-layer GRU RNN language models based on the sampling procedure described in section\{sec64}

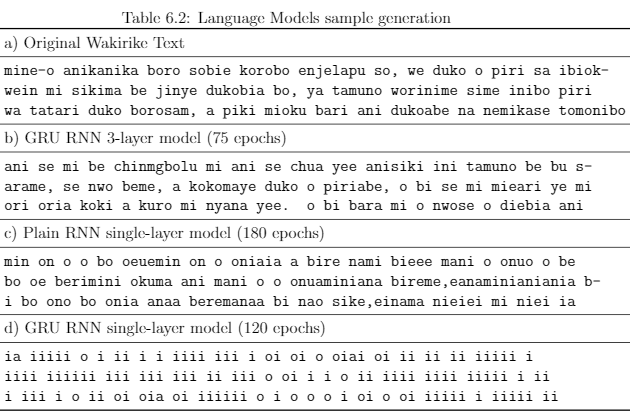


Table \ref{tab6\_2:Gen}(a) to (d) demonstrates how the language generated by various language models resembles the original training data. The original Wakirike language is given in part (a) then, the other language model generations (following the procedure described in Section \ref{sec6\_4}) are shown 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.

## 6.6 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.

# 

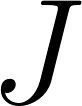
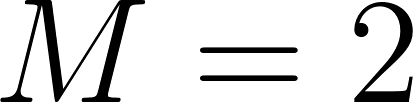
# Chapter 7: Empirical Analysis 2: Deep Recurrent Speech Recognition models

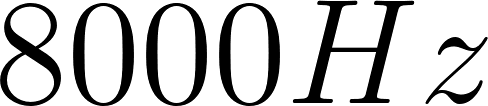
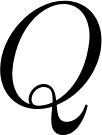
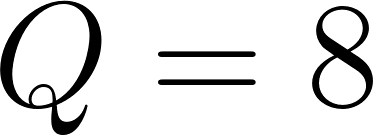
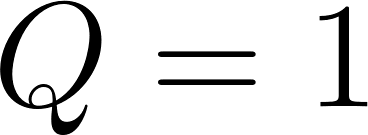
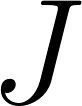
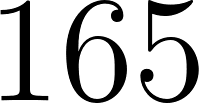
Throughout the development of this thesis, the establishment of deep learning as a strategy where computers learn through representation of patterns at varying degrees of complexity has been an underlying theme. It was also emphasised how this is achieved by internal layer-wise encapsulations. Structures discussed in Chapter \ref{ch2litrev}, such as layer-wise stacking of neural network type architectures such as the \acrlong{rbm} (\acrshort{rbm}) \acrlong{dbn} (\acrshort{dbn}) were used to implement such representations.

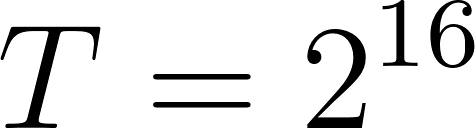
In this chapter, the end-to-end Bi-directional Recurrent Neural Network model is described. \acrshort{birnn} for speech recognition tasks is employed here as opposed to regular \acrshort{rnn}s or \acrshort{dbm}s mentioned above in the preceding paragraph. Use of \acrshort{birnn}s is attributed to the contextual nature of speech. Furthermore, we saw in the preceding Chapter how deep stacking of \acrshort{gru}s outperform single-layer \acrshort{rnn}s for extended sequences. That is to say, words in a sentence or paragraph are contextual to the sentence/paragraph over particularly long sequences are captured better by the GRU architecture. In addition, \acrshort{birnn}'s have a forward and backward \acrshort{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 \citep{graves2006connectionist}.

This chapter, however, first discusses speech features developed by making use of the deep scattering convolution networks \acrshort{dsn}. The \acrshort{dsn}s are used as inputs to the end-to-end model. Two end-to-end networks are then described. The core \acrshort{birnn} network and a second \acrshort{birnn} 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.

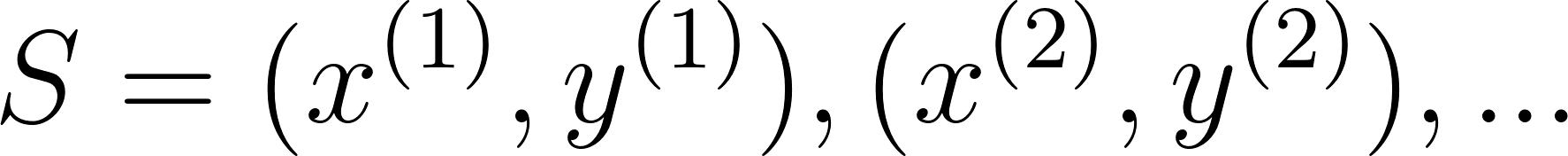
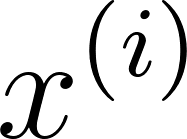
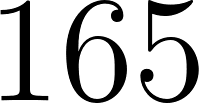
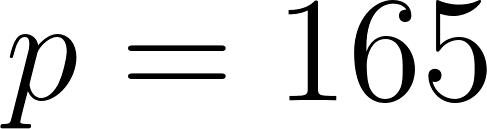
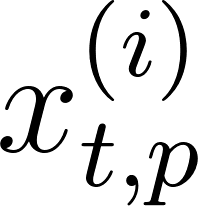
## Deep Scattering Features

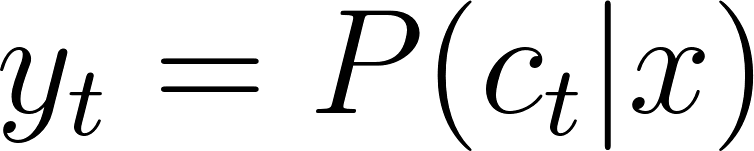
In Chapter 4, we derived a fast wavelet transform from a low pass filter and a high pass filter. The speech features used for the BiRNN is obtained from successive wavelet-modulus operations of a deep scattering network [](https://www.codecogs.com/eqnedit.php?latex=2%0) layers deep. This 2-layer DSN 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, [](https://www.codecogs.com/eqnedit.php?latex=T%0); The Q-band value for the band pass filter and the number of wavelets [](https://www.codecogs.com/eqnedit.php?latex=J%0) at each scattering layer for the total number of layers, [](https://www.codecogs.com/eqnedit.php?latex=M%3D2%0).

The matlab scatnet toolbox \citep{anden2014scatnet}, 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 [](https://www.codecogs.com/eqnedit.php?latex=T%0)= [](https://www.codecogs.com/eqnedit.php?latex=512%0) samples per window. This corresponds a window of 50 milliseconds for the audio signals sampled at [](https://www.codecogs.com/eqnedit.php?latex=8000%20Hz%0). For the first scattering layer the [](https://www.codecogs.com/eqnedit.php?latex=Q%0)-band parameter was [](https://www.codecogs.com/eqnedit.php?latex=Q%3D8%0) and the second scattering layer took the value [](https://www.codecogs.com/eqnedit.php?latex=Q%3D1%0). Finally [](https://www.codecogs.com/eqnedit.php?latex=J%0) is pre-calculated based on the value of [](https://www.codecogs.com/eqnedit.php?latex=T%0). These after Scat-Net processing produce a feature-vector having [](https://www.codecogs.com/eqnedit.php?latex=165%0) dimensions. These feature vectors in turn are used as inputs to the bi-direction neural network model whose architecture is described in succeeding sections.

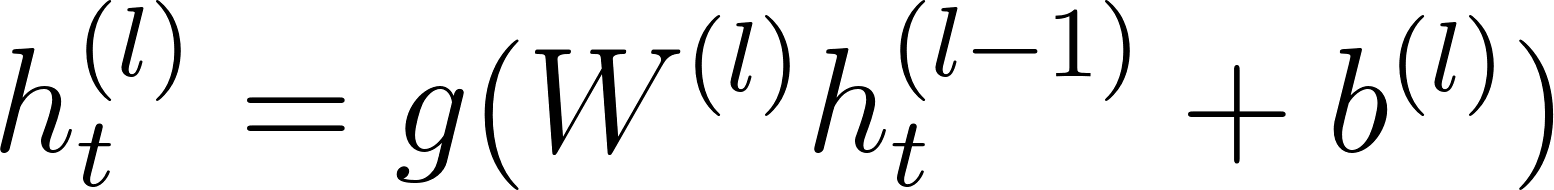
For the second end-to-end architecture involving a transducer with attention mechanism, a period of [](https://www.codecogs.com/eqnedit.php?latex=T%3D2%5E%7B16%7D#0) is used to capture a window of 4 seconds for audio signals sampled at 16000Hz. The same Q-band parameters are used.

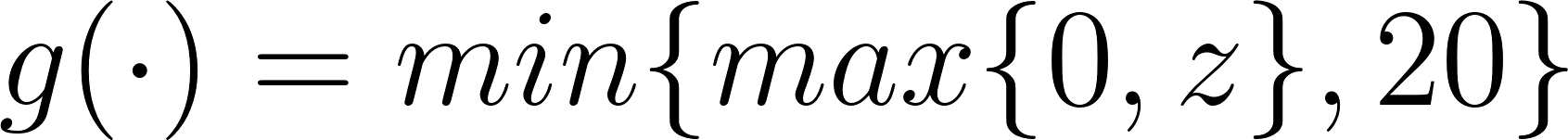
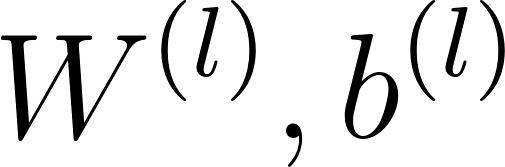
## CTC-BiRNN Architecture

The core of the system is a bidirectional recurrent neural network (BiRNN) trained to ingest scatter coefficients described in the previous section, in order to generate English text transcriptions. An end-to-end system therefore specifies that utterances [](https://www.codecogs.com/eqnedit.php?latex=x%0) and the corresponding label [](https://www.codecogs.com/eqnedit.php?latex=y%0) be sampled from a training set such that the sample [](https://www.codecogs.com/eqnedit.php?latex=S%20%3D%20%7B(x%5E%7B(1)%7D%2C%20y%5E%7B(1)%7D)%2C%20(x%5E%7B(2)%7D%2C%20y%5E%7B(2)%7D)%2C%20.%20.%20.%7D%0) In our end-to-end model, each utterance, [](https://www.codecogs.com/eqnedit.php?latex=x%5E%7B(i)%7D%0) is a processed feature vector consisting of [](https://www.codecogs.com/eqnedit.php?latex=165%0) dimensions. Recall, every window passes through a scattering transform to yield an input of vector of [](https://www.codecogs.com/eqnedit.php?latex=p%3D165%0) features; consequently, [](https://www.codecogs.com/eqnedit.php?latex=x%5E%7B(i)%7D_%7Bt%2Cp%7D%0) denotes the p-th feature in a scatter transform at time [](https://www.codecogs.com/eqnedit.php?latex=t%0).

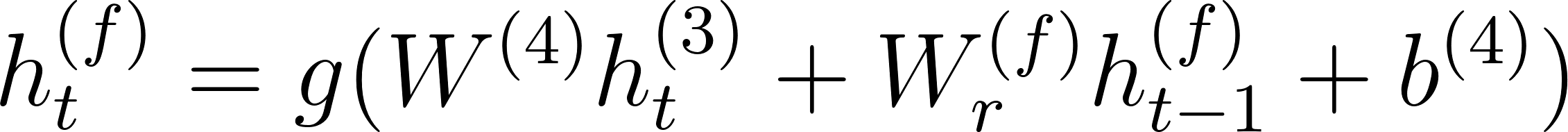
GPU training of the speech model architecture developed above was conducted using Mozilla deepspeech \cite{mozilla/deepspeech\_2019} CTC bi-directional RNN implementation along with the accompanying Mozilla Common voice dataset \cite{common voice by mozilla\_2019}. The Common Voice Dataset project consists of voice samples in short recordings approximately 4 seconds each. The complete dataset is about 250 hours of recording divided into training, test and development subsets. The BiRNN, given the input sequence, [](https://www.codecogs.com/eqnedit.php?latex=x%0), outputs a sequence of probabilities [](https://www.codecogs.com/eqnedit.php?latex=%5Cyat%7By%7D_t%3D%5CmathBB%7BP%7D(c_t%7Cx)%0), where [](https://www.codecogs.com/eqnedit.php?latex=c_t%20%5Cin%20%7Ba%2Cb%2Cc%2C%20.%20.%20.%20%2C%20z%2C%20space%2C%20apostrophe%2C%20blank%7D%0).

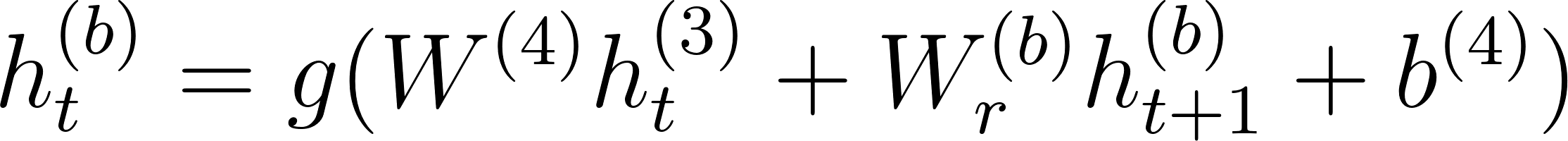
The actual architecture of our core Bi-RNN is similar to the deepspeech system described in \cite{hannun2014deep}. This structure consists of a 5 hidden layers and one output layer. The first three layers are regular DNNs followed by a bi-directional recurrent layer. As such, the output of the first three layers are computed by:

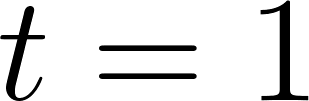
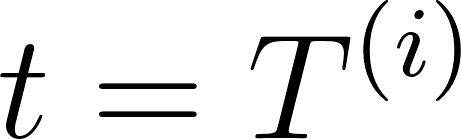
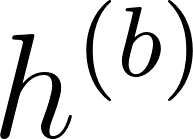
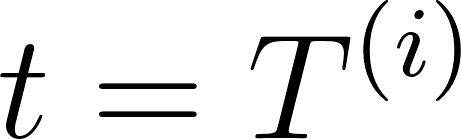
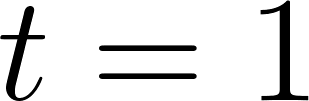
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(l)%7D_t%20%3D%20g(W%5E%7B(l)%7D%20h%5E%7B(l-1)%7D_t%20%2B%20b%5E%7B(l)%7D)%0)

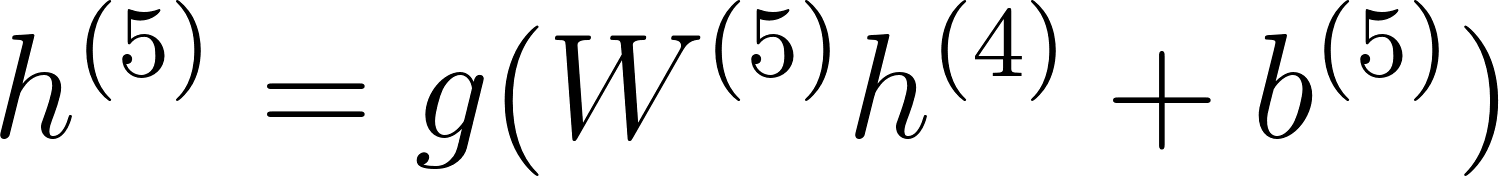
[](https://www.codecogs.com/eqnedit.php?latex=g(%5Ccdot)%20%3D%20min%5C%7Bmax%5C%7B0%2Cz%5C%7D%2C20%5C%7D%0) is the clipped rectified linear unit and [](https://www.codecogs.com/eqnedit.php?latex=W%5E%7B(l)%7D%2Cb%5E%7B(l)%7D%0) are weight matrix and bias parameters for layer [](https://www.codecogs.com/eqnedit.php?latex=l%0) as described in sections \ref{dnn} and \ref{deepspeech} respectively.

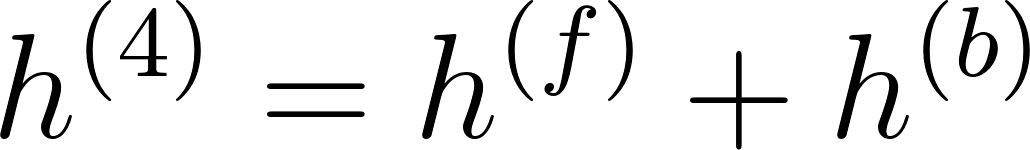
It was shown in chapter \ref{ch3RNN} the recurrent layer comprise a forward and backward RNNs whose equations are repeated here for reference

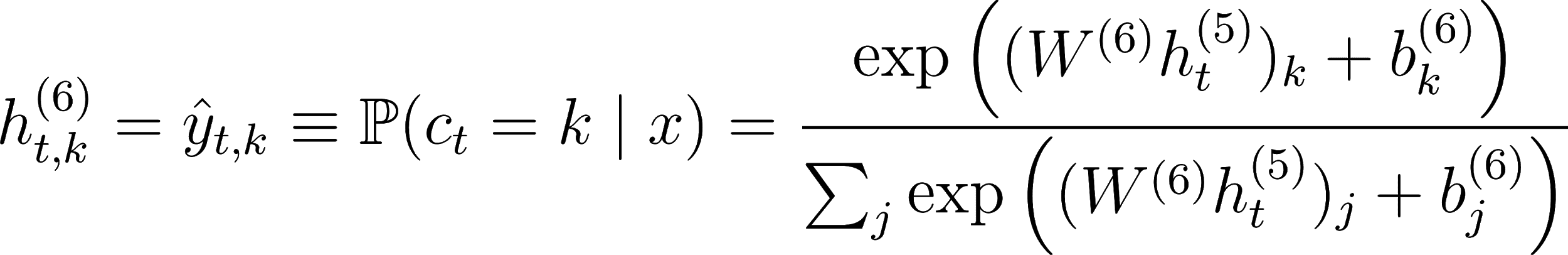
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(f)%7D_t%20%3D%20g(W%5E%7B(4)%7D%20h%5E%7B(3)%7D_t%20%2B%20W%5E%7B(f)%7D_r%20h%5E%7B(f)%7D_%7Bt-1%7D%20%2B%20b%5E%7B(4)%7D)%0)

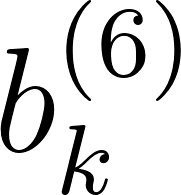
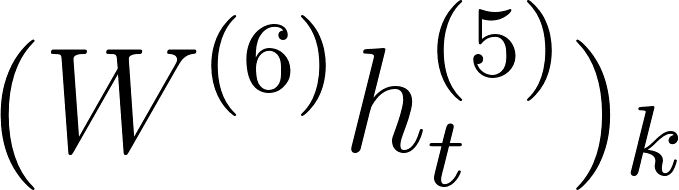
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(b)%7D_t%20%3D%20g(W%5E%7B(4)%7D%20h%5E%7B(3)%7D_t%20%2B%20W%5E%7B(b)%7D_r%20h%5E%7B(b)%7D_%7Bt%2B1%7D%20%2B%20b%5E%7B(4)%7D)%0)

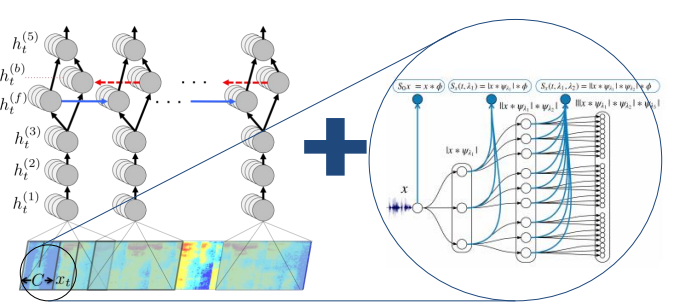
Consequently, [](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(f)%7D%0) is the sequential computation from [](https://www.codecogs.com/eqnedit.php?latex=t%3D1%0) to [](https://www.codecogs.com/eqnedit.php?latex=t%3DT%5E%7B(i)%7D%0) for the [](https://www.codecogs.com/eqnedit.php?latex=i%0)-th utterance and [](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(b)%7D%0) is the reverse computation from [](https://www.codecogs.com/eqnedit.php?latex=t%3DT%5E%7B(i)%7D%0) to [](https://www.codecogs.com/eqnedit.php?latex=t%3D1%0). In addition the output from layer five is summarily given as the combined outputs from the recurrent layer:

[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(5)%7D%20%3D%20g(W%5E%7B(5)%7D%20h%5E%7B(4)%7D%20%2B%20b%5E%7B(5)%7D)%0)

where [](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(4)%7D%20%3D%20h%5E%7B(f)%7D%20%2B%20h%5E%7B(b)%7D%0). The output of the Bi-RNN on layer 6 is a standard soft-max function that outputs a predicted character over probabilities for each time slice [](https://www.codecogs.com/eqnedit.php?latex=t%0) and character [](https://www.codecogs.com/eqnedit.php?latex=k%0) in the alphabet:

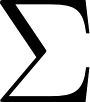
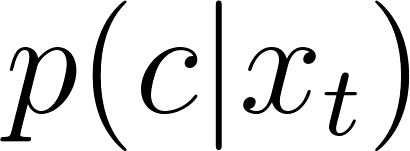
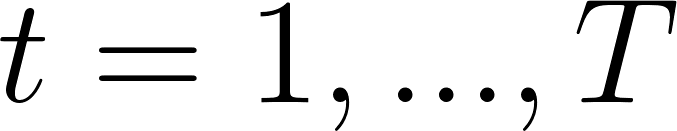
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(6)%7D_%7Bt%2Ck%7D%20%3D%20%5Chat%7By%7D_%7Bt%2Ck%7D%20%5Cequiv%20%5Cmathbb%7BP%7D(c_t%20%3D%20k%20%5Cmid%20x)%20%3D%20%5Cfrac%7B%5Cexp%7B%20%5Cleft(%20(W%5E%7B(6)%7D%20h%5E%7B(5)%7D_t)_k%20%2B%20b%5E%7B(6)%7D_k%20%5Cright)%7D%7D%7B%5Csum_j%20%5Cexp%7B%5Cleft(%20(W%5E%7B(6)%7D%20h%5E%7B(5)%7D_t)_j%20%2B%20b%5E%7B(6)%7D_j%20%5Cright)%7D%7D%0)

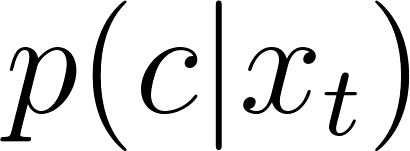
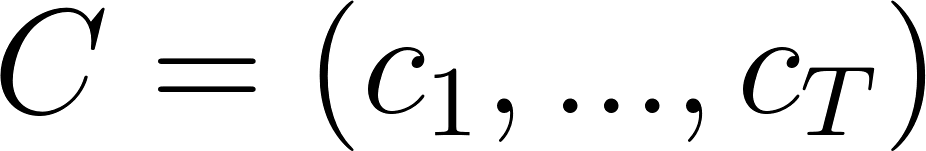
[](https://www.codecogs.com/eqnedit.php?latex=b%5E%7B(6)%7D_k%0) takes on the [](https://www.codecogs.com/eqnedit.php?latex=k%0)-th bias and [](https://www.codecogs.com/eqnedit.php?latex=(W%5E%7B(6)%7D%20h%5E%7B(5)%7D_t)_k%0) is the matrix product of the [](https://www.codecogs.com/eqnedit.php?latex=k%0)-th element. The error of the outputs are then computed using the CTC loss function \cite{graves\_2014} described in chapter \ref{ch3DNN}. A summary of our model is illustrated in figure \ref{ch06\_01\_ctc\_scatter}.

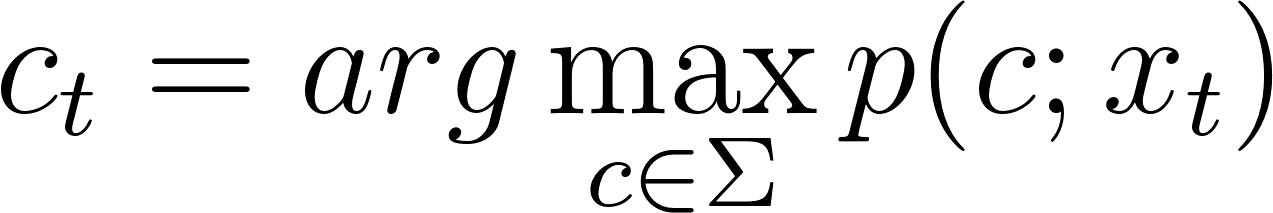


### CTC decoding

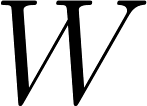
In chapter three the CTC loss function algorithm was established as being able to maximise the probability of two cases. The first case of transiting to a blank and the second case of transiting to a non blank. In this section, this concept is used to enable decoding of the network output from posterior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

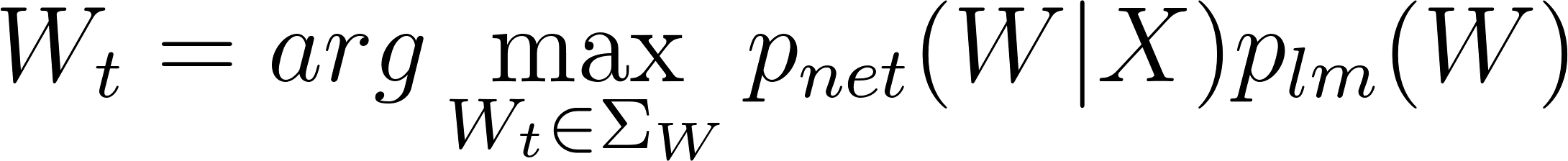
Recall, all the output symbols are in the alphabet [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma%0) and augmented with the blank symbol. The posterior output of the CTC network is the probability of the symbol given the speech feature input [](https://www.codecogs.com/eqnedit.php?latex=p(c%7Cx_t)%0) at time [](https://www.codecogs.com/eqnedit.php?latex=t%0) for [](https://www.codecogs.com/eqnedit.php?latex=t%3D1%2C...%2C%20T%0) and [](https://www.codecogs.com/eqnedit.php?latex=T%0) 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 [](https://www.codecogs.com/eqnedit.php?latex=p_b%0) and that of a non blank character [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bnb%7D%0).

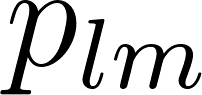
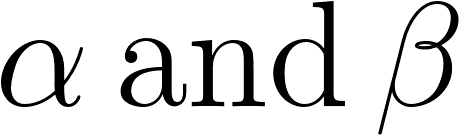
Several strategies have been employed to obtain a translation string from the output of the deep neural network. The prefix beam search employed by the CTC decoder of this research is derived from an initial greedy approximation, where at each time step it determines the argument that maximises the probability [](https://www.codecogs.com/eqnedit.php?latex=p(c%7Cx_t)%0) at each time step. Let [](https://www.codecogs.com/eqnedit.php?latex=C%3D(c_1%2C%20...%2C%20c_T)%0) be the character string then, the greedy approach has

[](https://www.codecogs.com/eqnedit.php?latex=c_t%3Darg%5Cmax_%7Bc%5Cin%5CSigma%7Dp(c%3Bx_t)%0) - - - 23b

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.

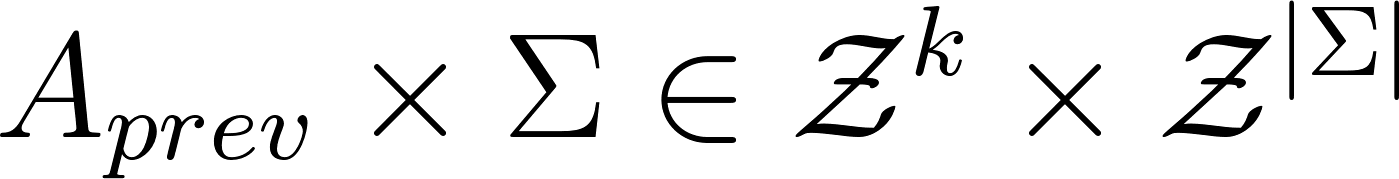
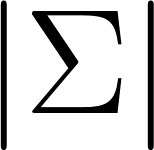
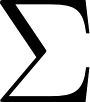
The prefix beam search algorithm \cite{hannun2014first} adopted in this work incorporates a language model derived from a lexicon in addition to keeping track of the various likelihoods used for decoding. For the language model constraint, the transcription [](https://www.codecogs.com/eqnedit.php?latex=W%0) is recovered from acoustic input [](https://www.codecogs.com/eqnedit.php?latex=X%0) at time [](https://www.codecogs.com/eqnedit.php?latex=t%0) by choosing the word which maximising the posterior probability:

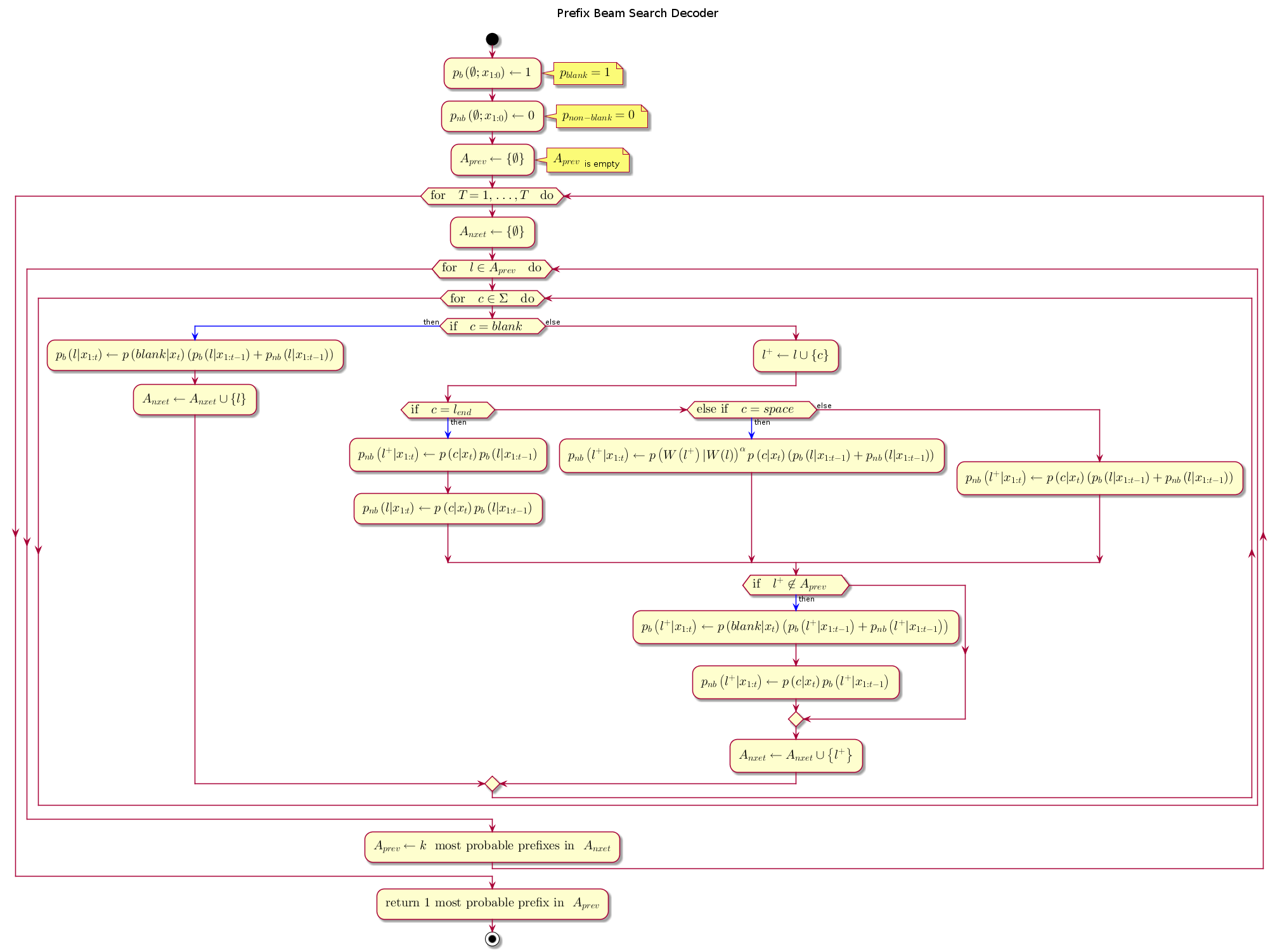
[](https://www.codecogs.com/eqnedit.php?latex=W_t%3Darg%5Cmax_%7BW_t%20%5Cin%20%5CSigma_W%7D%20p_%7Bnet%7D(W%7CX)p_%7Blm%7D(W)%0) - - - (24)

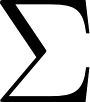
In equation \ref{eqn\_c6\_decoder01}, the Bayes product of language model prior [](https://www.codecogs.com/eqnedit.php?latex=p_%7Blm%7D%0) and the network output [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bnet%7D%0) are utilised to maximise the probability of a particular character-word sequence in the lexicon given by [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma_W%0). The overall calculation used to derive the final posterior distribution includes word insertion factors ([](https://www.codecogs.com/eqnedit.php?latex=%5Calpha%20%5C%2C%20%5Ctext%7Band%7D%20%5C%2C%20%5Cbeta%0)) used to balance the highly constrained n-gram language-model.

[](https://www.codecogs.com/eqnedit.php?latex=W_t%3Darg%5Cmax_%7BW_t%20%5Cin%20%5CSigma_W%7D%20p_%7Bnet%7D(W%7CX)p_%7Blm%7D(W)%5E%5Calpha%7CW%7C%5E%5Cbeta%0) - - - (25)

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 [](https://www.codecogs.com/eqnedit.php?latex=k%0) 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.

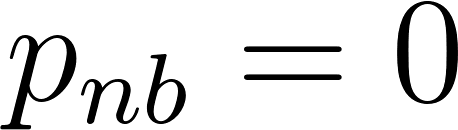
The recursive prefix beam search algorithm illustrated in figure \ref{fig\_c6\_decoder01} attempts to find the string formulated in equation \ref{eqn\_c6\_decoder02}. Two sets prefixes [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnxet%7D%0) are initialised, such that at [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnxet%7D%0) maintains the prefixes in the current timestep while [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) maintains only [](https://www.codecogs.com/eqnedit.php?latex=k%0)-prefixes from the previous timestep. Note that at the end of each time step [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) is updated with only [](https://www.codecogs.com/eqnedit.php?latex=k%0)-most probable prefixes from [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnxet%7D%0). Therefore while, [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnxet%7D%0) contains all the possible new paths from based on [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) as a cartesian product of [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%20%5Ctimes%20%5CSigma%20%5Cin%20%5Cmathcal%7BZ%7D%5Ek%20%5Ctimes%20%5Cmathcal%7BZ%7D%5E%7B%7C%5CSigma%7C%7D%0) where [](https://www.codecogs.com/eqnedit.php?latex=%7C%5CSigma%7C%0) is the length of [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma%0). 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



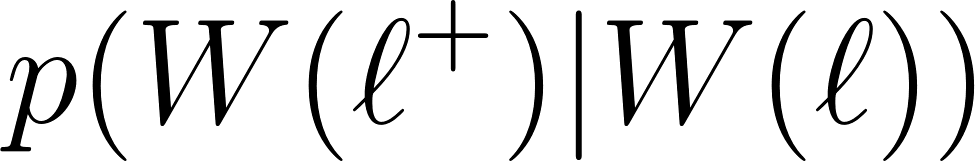
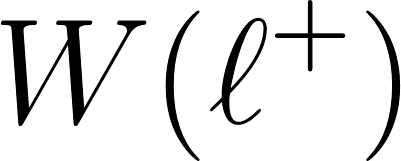
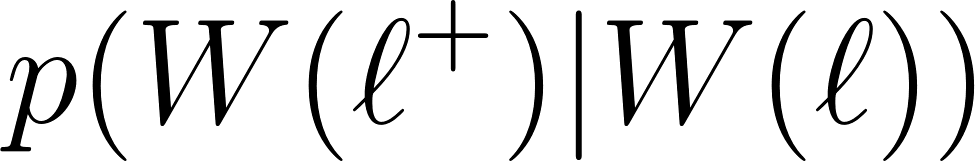
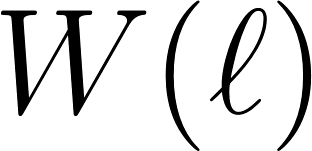
At every time step and for every prefix [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0) currently in [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0), a symbol from the alphabet [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma%0) is presented to the prefix. The prefix is only extended only when the presented symbol is not a blank or a space. [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnext%7D%0) 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.

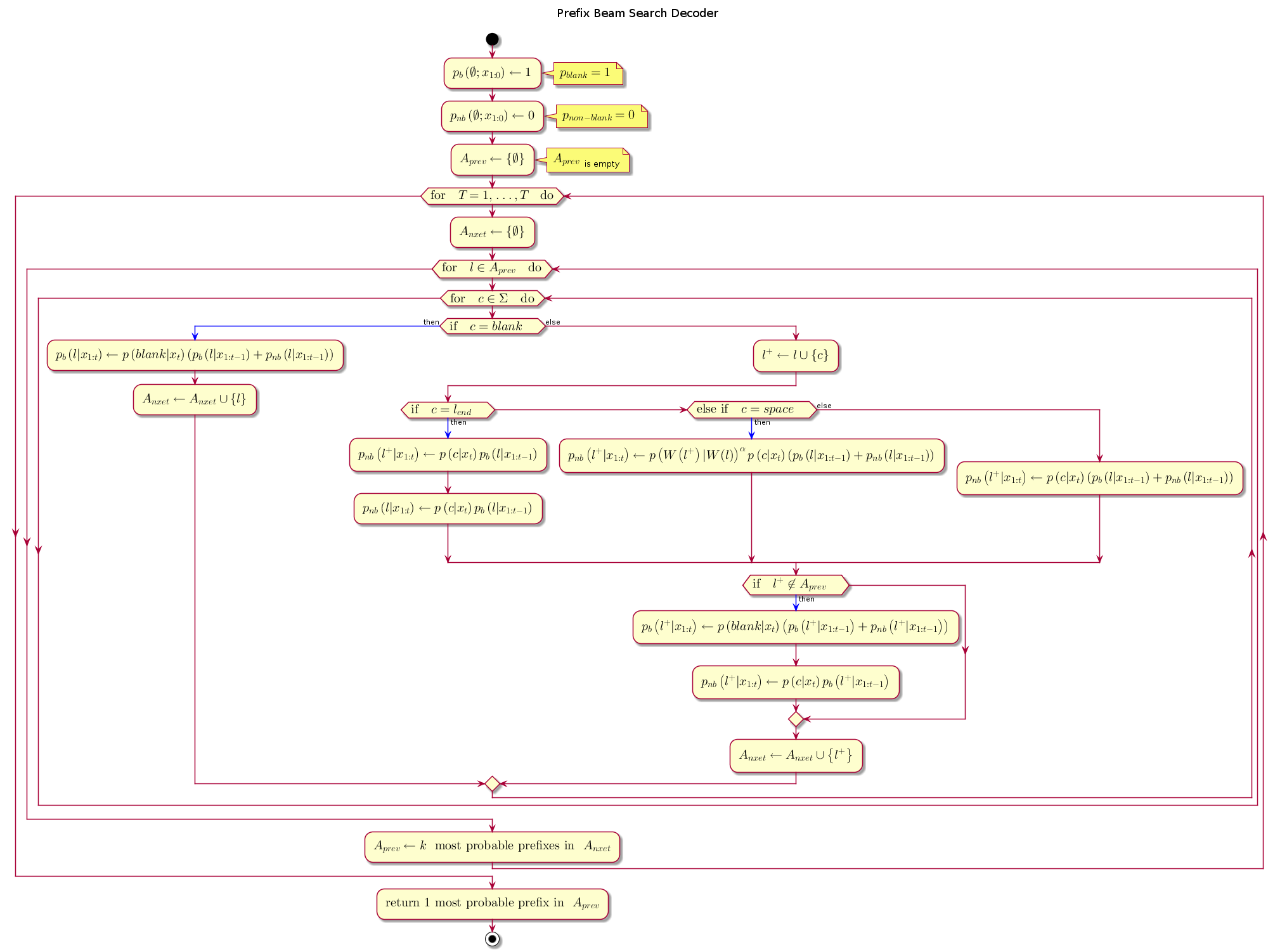
[](https://www.codecogs.com/eqnedit.php?latex=p(%5Cell%7Cx_%7B1%3At%7D)%3D(p_%7Bnb%7D(%5Cell%7Cx_%7B1%3At%7D)%2Bp_b(%5Cell%7Cx_%7B1%3At%7D))%7CW(%5Cell)%7C%5E%5Cbeta%0) - - - (26)

[](https://www.codecogs.com/eqnedit.php?latex=W(%5Ccdot)%0) 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.

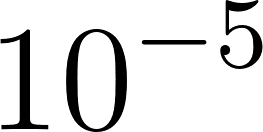
[](https://www.codecogs.com/eqnedit.php?latex=%5Cell_%7Bend%7D%0) is the variable representing the last symbol in the [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0). If the symbol presented is the same as [](https://www.codecogs.com/eqnedit.php?latex=%5Cell_%7Bend%7D%0) then the probability of a non-blank symbol, [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bnb%7D%3D0%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

[](https://www.codecogs.com/eqnedit.php?latex=p(%5Cell%5E%2B%7Cx_%7B1%3At%7D)%3Dp(W(%5Cell%5E%2B)%7CW(%5Cell))%5E%5Calpha(p_%7Bnb%7D(%5Cell%7Cx_%7B1%3At%7D)%2Bp_b(%5Cell%7Cx_%7B1%3At%7D))%7CW(%5Cell)%7C%5E%5Cbeta%0) - - - (27)

Note that [](https://www.codecogs.com/eqnedit.php?latex=p(W(%5Cell%5E%2B)%7CW(%5Cell))%0) is set to [](https://www.codecogs.com/eqnedit.php?latex=0%0) if the current word [](https://www.codecogs.com/eqnedit.php?latex=W(%5Cell%5E%2B)%0) is not in the lexicon. This becomes a constraint to enforce all character strings to consist only of words in the lexicon. Furthermore, [](https://www.codecogs.com/eqnedit.php?latex=p(W(%5Cell%5E%2B)%7CW(%5Cell))%0) is extended to include all the character sequences representing number of words considered by the n-gram language model by constituting the last [](https://www.codecogs.com/eqnedit.php?latex=n-1%0) words in character sequence [](https://www.codecogs.com/eqnedit.php?latex=W(%5Cell)%0).

[](http://www.plantuml.com/plantuml/img/hLPTRvim57tthxZINa1jil3QD5KtxGTCQgKzp6jam1IhndZ6h9ae_tt6XCyIGjVoG8XzVSwznuThRZDDbCuJZf1ccbFuhcZ43l2LaWJkA5513Dzea8PK0HO8sN24hguJekCRkVSTJ1Ef_sPKm_QmBxohJUbYJYCJfz9dyAu_syWj4gcceDXZh14qgmkV4_5KxfeeH4K8LM17K0Y_FCEncUSGgLWTUJQdUBxi2wdedxA7YekM5PSp3CtIOmYm3EowZkCvPaPOfmx7cXvq4QMgnBzp4iBzphjShzUNzr3_h-V3jC5p4K2Nf3XGVJv9W3bAp04p0Mt99sddOI2mCFYEFIO4vb2gYqL361OrA3kmctD4Q1Q2Zga0fxft802hdvzydjDVgnleZNPcCLxXBzOYUc0HwLWICwLTft9j4xNoIlVYwBB-cDiJSqGpjGtjM91BNF31NjGNvPbHihhtwsb1-SD53uzNE4LGJYGnBoNV5yPsPItc6Is7UeucXk-VoMvQrS0gEXPqCkrNcEz6hCGSYs5bxYkIIHBGPURw245-L5FkY_bntGTCk8n9MzH_6--qUzwuKU_9G8GiMkR3oikcyOrxpCUeFrVa-1rlumQb3aONUdM1_mUm2oLyIqyneavqbHRIFDbkZNgFfm-tfsEhXYJDD4YL-iGtdLxQmvzcOFQkRFXRqZ5k3Q6epfK0RnAg0UgVa5lp4P5AX6uDdldm-0S0)  
[PNG](http://www.plantuml.com/plantuml/img/hLPTRvim57tthxZINa1jil3QD5KtxGTCQgKzp6jam1IhndZ6h9ae_tt6XCyIGjVoG8XzVSwznuThRZDDbCuJZf1ccbFuhcZ43l2LaWJkA5513Dzea8PK0HO8sN24hguJekCRkVSTJ1Ef_sPKm_QmBxohJUbYJYCJfz9dyAu_syWj4gcceDXZh14qgmkV4_5KxfeeH4K8LM17K0Y_FCEncUSGgLWTUJQdUBxi2wdedxA7YekM5PSp3CtIOmYm3EowZkCvPaPOfmx7cXvq4QMgnBzp4iBzphjShzUNzr3_h-V3jC5p4K2Nf3XGVJv9W3bAp04p0Mt99sddOI2mCFYEFIO4vb2gYqL361OrA3kmctD4Q1Q2Zga0fxft802hdvzydjDVgnleZNPcCLxXBzOYUc0HwLWICwLTft9j4xNoIlVYwBB-cDiJSqGpjGtjM91BNF31NjGNvPbHihhtwsb1-SD53uzNE4LGJYGnBoNV5yPsPItc6Is7UeucXk-VoMvQrS0gEXPqCkrNcEz6hCGSYs5bxYkIIHBGPURw245-L5FkY_bntGTCk8n9MzH_6--qUzwuKU_9G8GiMkR3oikcyOrxpCUeFrVa-1rlumQb3aONUdM1_mUm2oLyIqyneavqbHRIFDbkZNgFfm-tfsEhXYJDD4YL-iGtdLxQmvzcOFQkRFXRqZ5k3Q6epfK0RnAg0UgVa5lp4P5AX6uDdldm-0S0) | [SVG](http://www.plantuml.com/plantuml/svg/hLPTRvim57tthxZINa1jil3QD5KtxGTCQgKzp6jam1IhndZ6h9ae_tt6XCyIGjVoG8XzVSwznuThRZDDbCuJZf1ccbFuhcZ43l2LaWJkA5513Dzea8PK0HO8sN24hguJekCRkVSTJ1Ef_sPKm_QmBxohJUbYJYCJfz9dyAu_syWj4gcceDXZh14qgmkV4_5KxfeeH4K8LM17K0Y_FCEncUSGgLWTUJQdUBxi2wdedxA7YekM5PSp3CtIOmYm3EowZkCvPaPOfmx7cXvq4QMgnBzp4iBzphjShzUNzr3_h-V3jC5p4K2Nf3XGVJv9W3bAp04p0Mt99sddOI2mCFYEFIO4vb2gYqL361OrA3kmctD4Q1Q2Zga0fxft802hdvzydjDVgnleZNPcCLxXBzOYUc0HwLWICwLTft9j4xNoIlVYwBB-cDiJSqGpjGtjM91BNF31NjGNvPbHihhtwsb1-SD53uzNE4LGJYGnBoNV5yPsPItc6Is7UeucXk-VoMvQrS0gEXPqCkrNcEz6hCGSYs5bxYkIIHBGPURw245-L5FkY_bntGTCk8n9MzH_6--qUzwuKU_9G8GiMkR3oikcyOrxpCUeFrVa-1rlumQb3aONUdM1_mUm2oLyIqyneavqbHRIFDbkZNgFfm-tfsEhXYJDD4YL-iGtdLxQmvzcOFQkRFXRqZ5k3Q6epfK0RnAg0UgVa5lp4P5AX6uDdldm-0S0) | [TXT](http://www.plantuml.com/plantuml/txt/hLPTRvim57tthxZINa1jil3QD5KtxGTCQgKzp6jam1IhndZ6h9ae_tt6XCyIGjVoG8XzVSwznuThRZDDbCuJZf1ccbFuhcZ43l2LaWJkA5513Dzea8PK0HO8sN24hguJekCRkVSTJ1Ef_sPKm_QmBxohJUbYJYCJfz9dyAu_syWj4gcceDXZh14qgmkV4_5KxfeeH4K8LM17K0Y_FCEncUSGgLWTUJQdUBxi2wdedxA7YekM5PSp3CtIOmYm3EowZkCvPaPOfmx7cXvq4QMgnBzp4iBzphjShzUNzr3_h-V3jC5p4K2Nf3XGVJv9W3bAp04p0Mt99sddOI2mCFYEFIO4vb2gYqL361OrA3kmctD4Q1Q2Zga0fxft802hdvzydjDVgnleZNPcCLxXBzOYUc0HwLWICwLTft9j4xNoIlVYwBB-cDiJSqGpjGtjM91BNF31NjGNvPbHihhtwsb1-SD53uzNE4LGJYGnBoNV5yPsPItc6Is7UeucXk-VoMvQrS0gEXPqCkrNcEz6hCGSYs5bxYkIIHBGPURw245-L5FkY_bntGTCk8n9MzH_6--qUzwuKU_9G8GiMkR3oikcyOrxpCUeFrVa-1rlumQb3aONUdM1_mUm2oLyIqyneavqbHRIFDbkZNgFfm-tfsEhXYJDD4YL-iGtdLxQmvzcOFQkRFXRqZ5k3Q6epfK0RnAg0UgVa5lp4P5AX6uDdldm-0S0) | [Edit](http://www.planttext.com/planttext?text=hLPTRvim57tthxZINa1jil3QD5KtxGTCQgKzp6jam1IhndZ6h9ae_tt6XCyIGjVoG8XzVSwznuThRZDDbCuJZf1ccbFuhcZ43l2LaWJkA5513Dzea8PK0HO8sN24hguJekCRkVSTJ1Ef_sPKm_QmBxohJUbYJYCJfz9dyAu_syWj4gcceDXZh14qgmkV4_5KxfeeH4K8LM17K0Y_FCEncUSGgLWTUJQdUBxi2wdedxA7YekM5PSp3CtIOmYm3EowZkCvPaPOfmx7cXvq4QMgnBzp4iBzphjShzUNzr3_h-V3jC5p4K2Nf3XGVJv9W3bAp04p0Mt99sddOI2mCFYEFIO4vb2gYqL361OrA3kmctD4Q1Q2Zga0fxft802hdvzydjDVgnleZNPcCLxXBzOYUc0HwLWICwLTft9j4xNoIlVYwBB-cDiJSqGpjGtjM91BNF31NjGNvPbHihhtwsb1-SD53uzNE4LGJYGnBoNV5yPsPItc6Is7UeucXk-VoMvQrS0gEXPqCkrNcEz6hCGSYs5bxYkIIHBGPURw245-L5FkY_bntGTCk8n9MzH_6--qUzwuKU_9G8GiMkR3oikcyOrxpCUeFrVa-1rlumQb3aONUdM1_mUm2oLyIqyneavqbHRIFDbkZNgFfm-tfsEhXYJDD4YL-iGtdLxQmvzcOFQkRFXRqZ5k3Q6epfK0RnAg0UgVa5lp4P5AX6uDdldm-0S0)

### 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 \citep{kingma2014adam} was used with initial learning rate of [](https://www.codecogs.com/eqnedit.php?latex=10%5E%7B-5%7D%0), 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 \ref{tab\_c6\_01\_training}. For decoding with prefix search we use a beam size of 200 and cross-validated with a held-out set to find a good setting of the parameters [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha%0) and β. Figure \ref{fig\_6\_3\_wer} shows word error rates for various GPU configurations and audio dataset sizes.

## BiRNN with Transducer+Attention end-to-end Architecture

The objective of attention-based networks highlighted by \cite{vaswani2017attention} 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 \citep{kaiser2016can,kalchbrenner2016neural, gehring2017convolutional}. \cite{vaswani2017attention} 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.

The end-to-end architecture at the core of ESPNet is the CTC-Transformer+Attention Transducer model. Together these two architectures achieve joint multi-objective speech training and decoding. The CTC-Transformer model is based on a Bi-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 \cite{chorowski2015attention}. Moreover, the multi-objective training was performed with equal weights on both the CTC-transformer and the Attention-Transducer. Finally the system was trained for 20 epochs only.

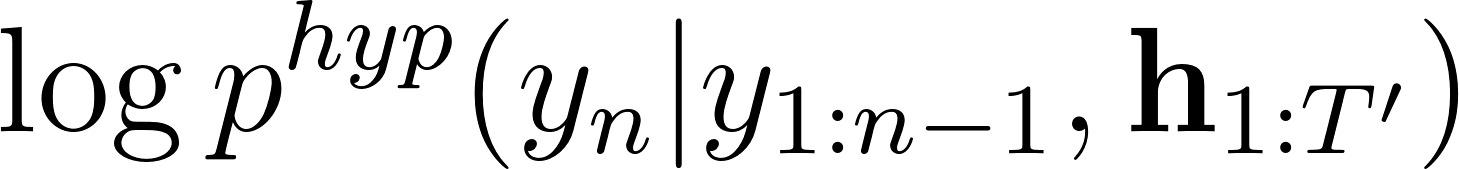
With this minimal default setting, the test set had a final recognition score of 9.5% character error rate (CER). The next Chapter discusses how the baseline can be scaled and remodelled for integrating scattering features.

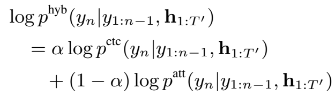
The inclusion of Attention-based models and the warp-CTC decoder implemented in ESPNet ensures faster convergence and ultimately faster time to train, in addition to the end-to-end architecture. Using a weighting function [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha%0) one can control how much 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.

**

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

[](https://www.codecogs.com/eqnedit.php?latex=%5Clog%20p(y_n%7Cy_%7B1%3An-1%7D%2C%5Cmathbf%7Bh%7D_%7B1%3AT'%7D)%3D%5Clog%20p%5E%7Bhyp%7D(y_n%7Cy_%7B1%3An-1%7D%2C%5Cmathbf%7Bh%7D_%7B1%3AT'%7D)%2B%5Cbeta%5Clog%20p%5E%7Blm%7D(y_n%7Cy_%7B1%3An-1%7D)%0)

Where joint decoding [](https://www.codecogs.com/eqnedit.php?latex=%5Clog%20p%5E%7Bhyp%7D(y_n%7Cy_%7B1%3An-1%7D%2C%5Cmathbf%7Bh%7D_%7B1%3AT'%7D)%0) is given by

**

Furthermore, multi-channel training allows noise robust and far-field speech recognition tasks which incorporates joint 83-dimension feature-training in addition to speech enhancement features such as beam forming and STFT masking \cite{ochiai2017multichannel}.

## Model baseline

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 \citep{panayotov2015librispeech, mozilla/deepspeech\_2019}. Study by \cite{hannun2014first} 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.

## 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 BiRNN-only model. The first GPU configuration for the BiRNN-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.

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

For the BiRNN-only end-to-end experiments, GPU configuration experiments were carried out on varying-size subsets of the common voice corpus. The various GPU configurations along with the training times are shown in Table \ref{tab\_c6\_01\_training}.

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment | Hours of speech | Training time | Training status |
| 1. 2xGPU 10GB RAM | 1 | 7 days | complete |
| 2. 2xGPU 10GB RAM | 10 | 40 days | incomplete |
| 3. 5xGPU 15GB RAM | 2 | 2 days | complete |
| 4. 5xGPU 15GB RAM | 40 | 40 days | incomplete |
| 5. 1xCPU 16GB RAM | 20 | 20 days | Incomplete |
| 6. 1xGPU 2GB RAM | 20 | 20 days | Incomplete |

It can be seen in Table \ref{tab\_c6\_01\_training}, 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 \ref{tab\_c6\_01\_training} shows the details for the \acrfull{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 \ref{fig\_6\_3\_wer}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Experiment | Hours of speech | Corpus | No of epochs | WER |
| 2xGPU 10GB (scatter feat.) | 1 | CV LVCSR | 40 | 100+ |
| 2xGPU 10GB (scatter feat.) | 10 | CV LVCSR | 25 | 100+ |
| 5xGPU 15GB (scatter feat.) | 2 | CV LVCSR | 40 | 100 |
| 5xGPU 15GB (scatter feat.) | 40 | CV LVCSR | 40 | 87 |

Table 2: BiRNN-only end-to-end model Experiments

### BiRNN with Transducer and Attention mechanism Experiments

The \acrfull{espnet} \citep{watanabe2018espnet} provides building blocks for \acrshort{blstm} transducer with attention mechanism described in Section \ref{sec\_7\_5\_blstm\_t}. Two experiments involving a much smaller audio corpus guaranteed to converge quickly at training and a larger Italian language speech corpus \citep{foxvorge2019} used for these experiments. The AN4 (alphanumeric) corpus by Carnegie Mellon University \citep{acero1990acoustical}, 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 recording made with near-field microphone quality. The compressed tar file comes with a variety of audio formats including raw wav format, the NIST sphere format and those already encoded as Mel cepstral coefficients.

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 \acrshort{blstm} and is similar to what is obtainable in the DeepSpeech model. There are up to 11 variants of Attention networks implemented in \acrshort{espnet}, however, the results of the \acrshort{espnet} experiment performed was determined from the model described in \cite{chorowski2015attention}. Moreover, the multi-objective training was performed with equal weights on both the CTC-transformer and the Attention-Transducer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Experiment | Hours of speech | Total Training Time | Number of epochs | Training status |
| 1xGPU 4GB (log mel.) | 1 | 15 mins | 20 | completed |
| 1xCPU 16GB (scatter feat) | 1 | 3 Days | 100 | completed |
| 1xGPU 4GB (log mel) | ~10 | 11 hours | 200 | completed |
| 1xGPU 4gb (scatter feat) | ~10 | 38 hours | 200 | completed |

Table 3: BiRNN-only end-to-end model Experiments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1xGPU 4GB (log mel) | 1 | AN4 SVCSR | WER(%) | 12.9 |
| 1xGPU 4GB (scatter feat.) | 1 | AN4 SVSCR | WER(%) | 26.8 |
| 1xGPU 4GB (scatter feat) | ~10 | Voxforge-italian (LVSCR) | WER(%) | 76.7 |
| 1xGPU 4GB (log mel) | ~10 | Voxforge-italian (LVSCR) | WER(%) | 72.4 |

Table 4: BiRNN with transducer and attention end-to-end model Experiments

## Interpretation of Model Results

Experiments carried out to train the end-to-end \acrshort{asr} were performed on the following system configurations

1. GPU (GTX1050) with 2GB RAM
2. GPU (GTX1050) with 4GB RAM
3. GPU (GTX1060) with 6GB RAM
4. GPU (GTX1070) with 3GB RAM
5. CPU with 16GB RAM

### 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 \acrshort{birnn}-only experiments, regardless of the batchsize allocations, only 2GPU configurations (1 and 3 in Table \ref{tab\_c6\_01\_training}) completed training for the given amounts of epochs and training data.

Table \ref{tab\_c6\_02\_training} 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 \ref{fig\_6\_3\_wer}. 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.

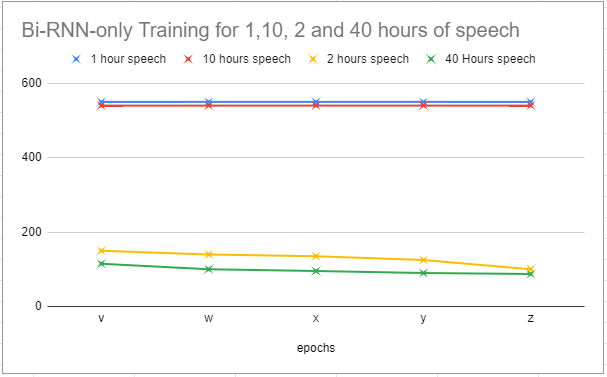


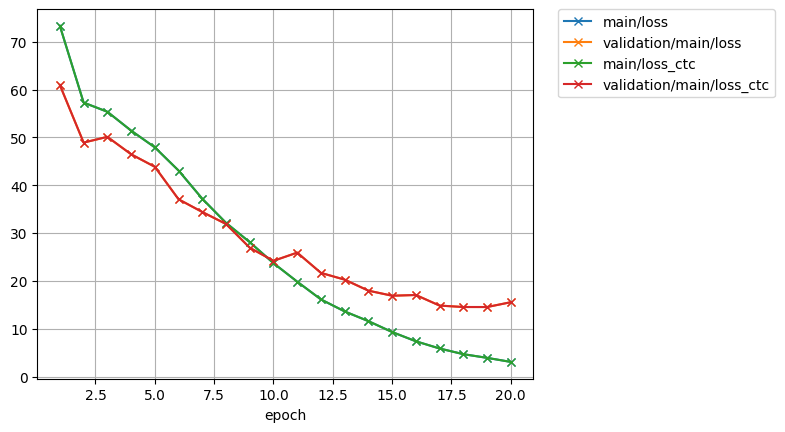
Figure 7.1 Accuracy curves for Bi-RNN-only experiments

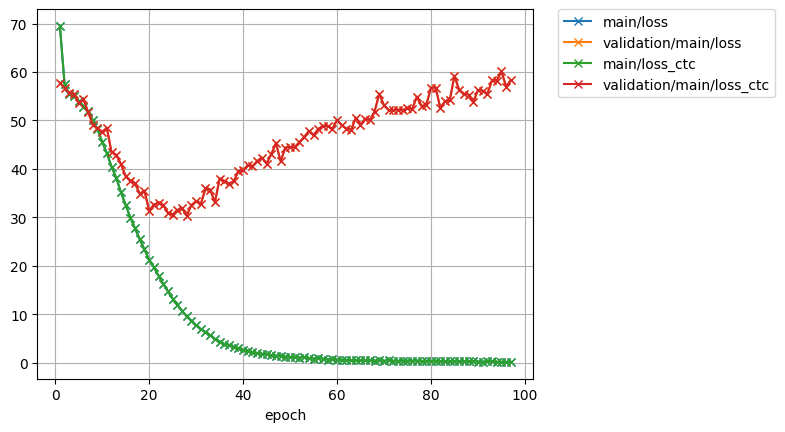
### Bi-RNN with Transducer and attention mechanism experiment discussion

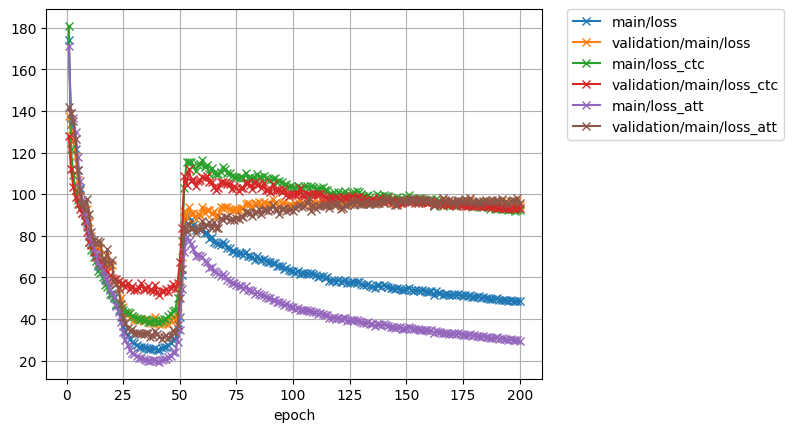
Results from Bi-RNN with Transducer and attention experiments had shown greater promise in terms of completion of training within the time constraints set. Results shown in table \ref{tab\_c6\_03\_training} show that both CPU and GPU training completed training for less than 20 hours of training and total number of epochs.

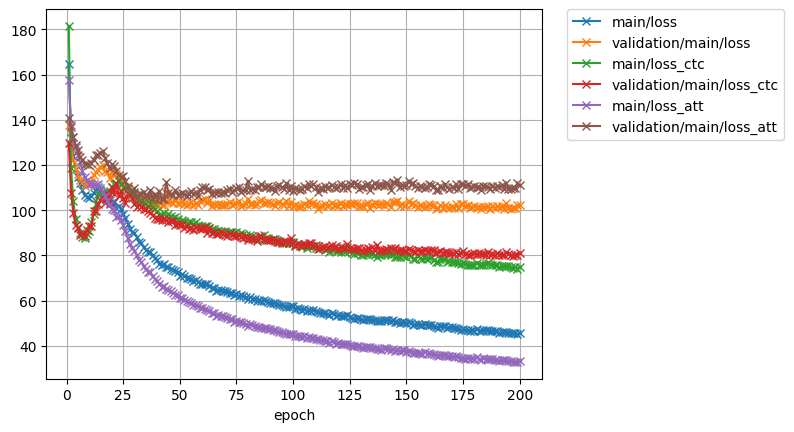
We used a \acrfull{svcsr} corpus of English language and a \acrfull{lvcsr} of Italian and achieved a decent score of 26.8\% for the \acrshort{svcsr} corpus and a high error score of 76.7\% for the \acrshort{lvcsr} corpus. The high error score for the \ref{lvcsr} from the observed results was attributed 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 \ref{fig\_6\_2\_loss} and \ref{fig\_6\_3\_wer} 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.









## 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 250-dimension 2nd-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 \acrshort{bilstm} 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 \ref{ch8\_future} we address how these results may be improved.

# 

# Chapter 8: Conclusion and Future study

textcolor{blue}{This research has been inspired by the notion of a \acrfull{llc}. One of the artificial intelligence tasks for \acrshort{llc} is \acrfull{asr}. This research, therefore, explored the current advancements that have been made in the field of \acrshort{asr} and discovered one of the major challenges in field of \acrshort{asr} is that of new and low resourced languages. In particular, considering \acrshort{asr} as a machine learning pipeline, this research favours a discriminative approach to \acrshort{asr} over generative approaches. Although hybrid \acrshort{hmm}-\acrshort{dnn}, which is considered a generative model, does deliver state-of-the art results, the \acrshort{asr} pipeline involves training of different aspects of the \acrshort{asr} pipeline in separate processes. The end-to-end approach, however, attempts to simplify this by offering a solution that involves training of a single discriminative "end-to-end" model. }

As \acrshort{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 and the intrinsic integration of a character-based language model help to train low-resource languages for ASR in a resource efficient manner.

This Chapter summarises 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 Attention-based models.

## Discussion of Output Research Models

Two well established key aspects of \acrshort{asr} from the \acrshort{asr} formulation in Equation \ref{eqn\_2\_3\_bayes\_sr} include an \acrfull{am} and a \acrfull{lm}. The objective of this research was to understand these key aspects \acrshort{asr} systems and develop \acrshort{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 (\acrshort{birnn}) end-to-end system, developed in this work, in addition, is augmented by features derived from a deep scattering network as opposed to the standard Mel Frequency Cepstral Coefficients(MFCC) features used in state of the art acoustic models. These specialised deep scattering features, consumed by the \acrshort{birnn} 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 speech models developed in this research.

### \subsection{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 Deep Scattering features derived from wavelet filter operations on audio data 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\% \acrshort{wer} (14\% points from the baseline) for \acrshort{svcsr} and 76.7\% for \acrshort{lvcsr} which was 4\% points from the baseline.

### \subsection{Summary of goals Achieved in this work}

The Zero-resource speech recognition challenge \citep{versteegh2015zero}, that mirrors the \acrlong{llc} motivation for this work, seeks \acrshort{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 signal using convolution (Section \ref{sec\_c3\_corr}) as a precursor to the scattering transform. Also, at the pilot stage, various sequence-to-sequence models were developed for sub-word \acrshort{asr} system modeling. A grapheme-to-phoneme model was trained in Section \ref{sec\_c3\_g2p}, from which a phonetic dictionary for the Wakirike language was developed. An \acrshort{asr} system post-processor was also developed based on a sub-word sequence-to-sequence \acrshort{rnn} model (Section \ref{sec\_c2d}). This post-processor was to insert diacritic, tonal symbols to \acrshort{asr} output text. Although the sub-word systems built in this work brought insight to sequence modelling using \acrfull{rnn}s, in practice only the diacritic post-processing tool had a direct application to the final research output. The \acrfull{g2p} tool not have a direct application to the research output which eliminates the need of using \acrshort{g2p} mechanisms. The tool would however be valuable for hybrid \acrshort{gmm}-\acrshort{hmm}/\acrshort{dnn} \acrshort{asr} systems.

Some aspects of \acrshort{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 \ref{sec\_digitspeech}). These are pre-input-feature processing tasks which occurs 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 to end-to-end CTC-based systems implemented in our work for new languages is the time saved from the need for accurate 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 \acrshort{dsn} features, in Chapter \ref{ch6\_wlm}, a word-level language model was developed using a character-based, \acrshort{gru}-\acrshort{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 \acrshort{gru}-\acrshort{rnn} language character based model developed by this research was integrated into some of the final speech models built in this work.

## 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. While a handful of \acrshort{asr} sequence models were developed for the Wakirike language, the Wakirike language, 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 \acrlong{lvcsr}. The training of \acrshort{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 are available, a number of improvements to the present models are possible. One includes further investigation of first and second order scatter network features. While first order features have been recommended, experiments used in this research utilised 2nd-order features.

## 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 \acrshort{dsn} speech features include the addition of \acrfull{vtln} and noise-robust speech enhancement. These feature engineering could immediately improve robustness of the \acrshort{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 of this work.

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

## Attention-based Models

The objective of attention-based networks highlighted by \cite{vaswani2017attention} 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 \citep{kaiser2016can,kalchbrenner2016neural, gehring2017convolutional}. \cite{vaswani2017attention} 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. \cite{vaswani2017attention} 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 \acrshort{dss} features, considerable resource-saving may be gained by employing self-attention-based networks. This has been explored by \cite{salazar2019self} with satisfactory results.

### Model pretraining

The major setback this work suffered was from a reasonable time for training. This work therefore recommends that speech models or indeed artificial intelligence models should be trainable within a maximum of 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 \citep{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.

\cite{hendrycks2019using} 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 \citep{ramachandran2016unsupervised,} uses a knowledge transfer mechanism where hidden layers in an already existing related network is re-trained with new extended layers to complete training in the new domain. The effectiveness of such a transfer method will be measured of the how the two domains correlate with one another. It therefore, would be logical to conclude that the more the domains correlate the faster the pretraining model is likely to converge faster.

Finally, \cite{wang2019bridging} 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 lower 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.

## Conclusion

The outcome of this research is an \acrshor{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 \acrshort{asr} system was the intrinsic integration of a character-based language model enabling it to satisfy both low-resource challenge criteria of top level word and sentence modelling through the character-based language model and the sub-word and acoustic modelling of input features.

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 \acrlong{vtln} was not done. This could be added as a precursor to the scattering transform.

This research demonstrates sequence-to-sequence and end-to-end models as flexible and still relatively untapped effective tools in many aspects of \acrshort{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 rich-resource English language.

# References

Lyons, J. (2012). Mel Frequency Cepstral Coefficient ([MFCC](http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/)) tutorial. Retrieved from <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>

# 