Deep Scattering End-to-End Speech Recognition

# Abstract

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

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

# 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. 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.

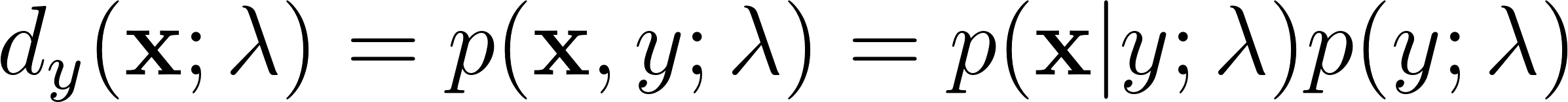
It has been acknowledged in the research community \citep{2015watanabe,deng2013machine} that work being done in Machine Learning has enhanced the research of automatic speech recognition. Similarly any progress made in ASR usually constitute contributions to enhancements made in the Machine Learning 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.

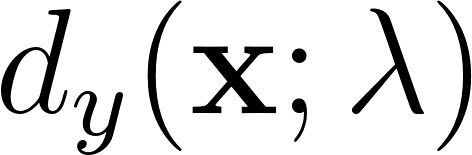
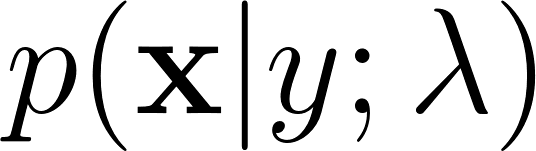
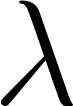
The two main approaches to Machine Learning problems historically involve two methods rooted in statistical science. These approaches are generative and discriminative models. From a computing science perspective, the generative approach is a brute-force approach while the discriminative model uses a 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

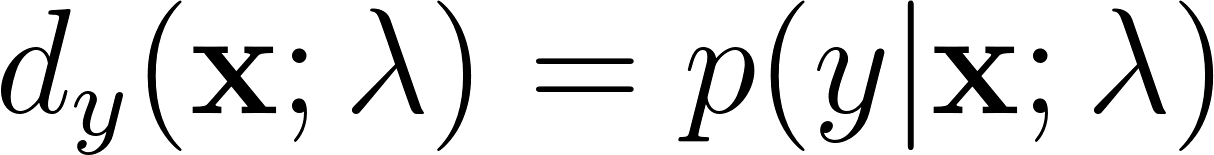
In the next chapter, the Hidden Markov Model (HMM) is examined as a powerful and major driver behind generative modelling of sequential data like speech. Generative models are data-sensitive models because they are derived from the data by accumulating as many different features which can be seen and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

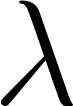
Ultimately, generative models for Machine Learning learning can be interpreted mathematically as a joint distribution that produces the highest likelihood of outputs and inputs based on a predefined decision function. The outputs for speech recognition being the sequence of words and the inputs for speech 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) predefines 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, information is lost in training data due to the physiologic transformations of the intended speech message as it moves from one speech 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 deep-speech model \citep{hannun2014first} 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. In theory this deep learning technique is sufficient in itself without a language model. However, applying a language model was found 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 \citep{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, English language was substituted and used as a control variable to study low resource effects of a language when exposed to speech models developed in this work.

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

## Research Aims and Objectives

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

The research objectives were as follows:

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

Within this framework, 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 of time to train speech models
2. Optimisation of sub-tasks and training architecture within the ASR pipeline
3. Reduction of data to train speech models
4. Make efficient use of training parallelism
5. Obtain better or state of the art performance
6. Induce model simplicity thereby reducing training time without compromising performance

## 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 pristine and state of the art audio signal processing mechanism involving a hierarchical scattering network to engineer features to compete with current acoustic and deep architectures for speech recognition. This was shown to be heading towards model saturation.

## Thesis outline

The outline of this report follows the development of an end-to-end speech recogniser and develops the theory based on the building blocks of the final system. 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.

Various Low speech recognition methods are reviewed and the relevance of this study is also highlighted. Chapter three 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 Four 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 five and six give descriptions of the models developed by this thesis and details the experimental setup along with the results obtained. Chapters seven 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.

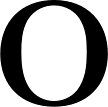
This Chapter also introduces the Wakirike language as a low resource language and the motivating language for this research. In addition, the various machine learning architectures used in this research for low resource speech recognition for the Wakirike and for English language are reviewed. In particular,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

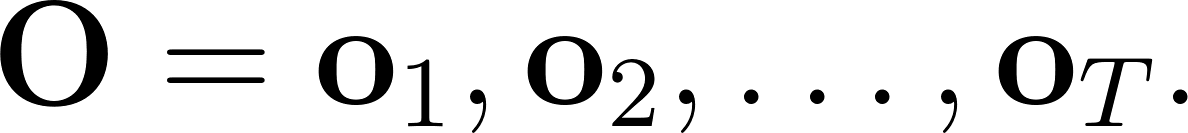
# Previous Deep Scattering Research

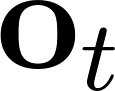
The speech recogniser developed in this thesis is based on an end-to-end discriminative deep recurrent neural network. Two models were developed. The first model, a Gated-Recurrent-Unit Recurrent Neural network (GRU-RNN), was used to develop a character-based Language Model (LM). The second model is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

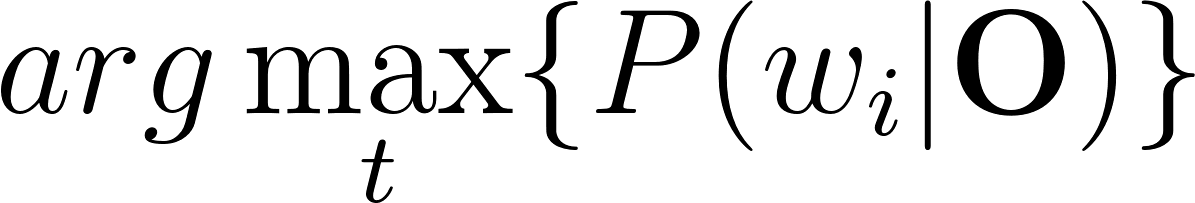
## Speech Recognition Overview

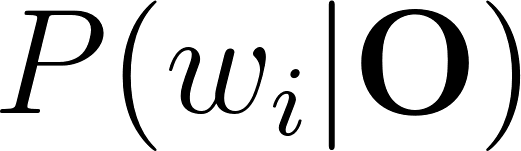
Computer speech recognition takes raw audio speech and converts it into a sequence of symbols. This can be considered as an analog to digital conversion as a continuous signal becomes discretised. The way this conversion is done is by breaking up the audio sequence into very small packets referred to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

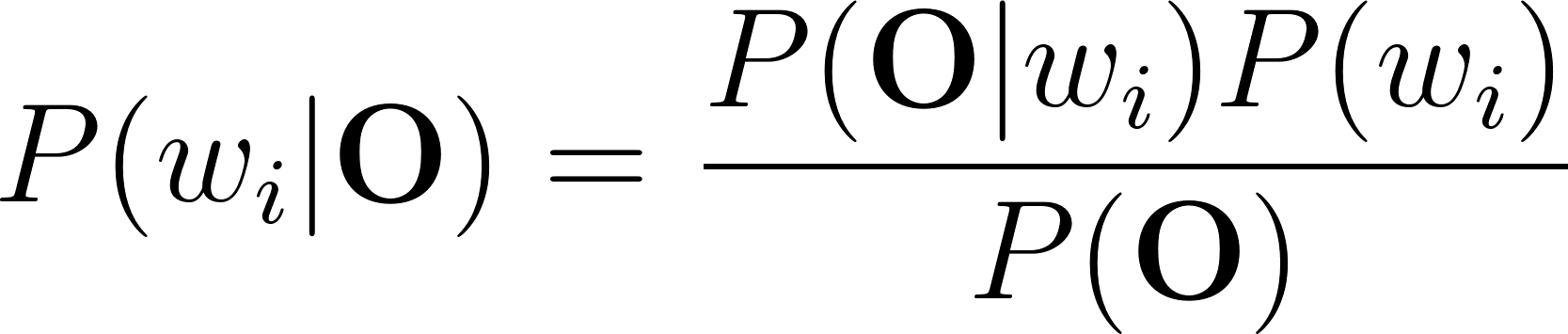
A statistical formulation \citep{young2002htk} for the speech recogniser follows given that each discretised output word in the audio speech signal is represented as a vector sequence of frame observations defined in the set [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BO%7D%0) such that

[](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)

As we describe in the next section on speech recognition challenges, there is no straightforward analysis of [](https://www.codecogs.com/eqnedit.php?latex=P(w_i%7C%5Cmathbf%7BO%7D)%0). 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

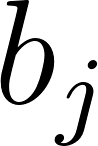
[](https://www.codecogs.com/eqnedit.php?latex=P(w_i%7C%5Cmathbf%7BO%7D)%3D%5Cfrac%7BP(%5Cmathbf%7BO%7D%7Cw_i)P(w_i)%7D%7BP(%5Cmathbf%7BO%7D)%7D%0) - - - (1.3)

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.

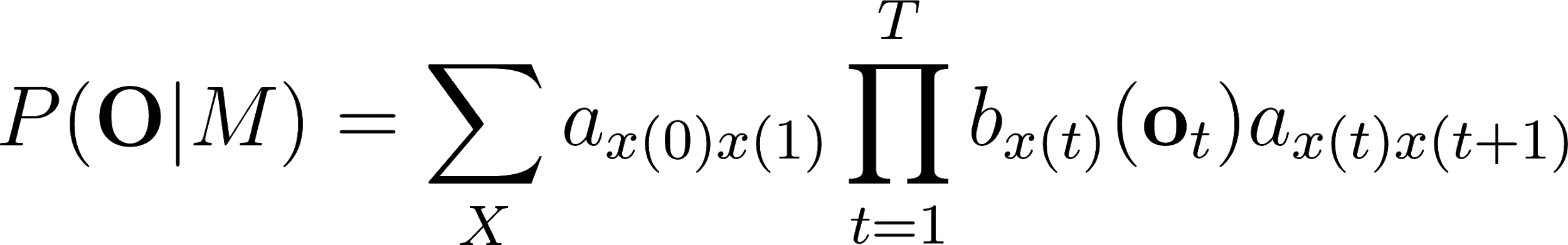
The second parameter in the speech model interpreted from Bayes' formula, is the prior probability of a given word. This aspect of the model is the language model is reviewed in section \ref{sec\_lrlm}.

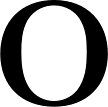
### HMM-based Generative speech model

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

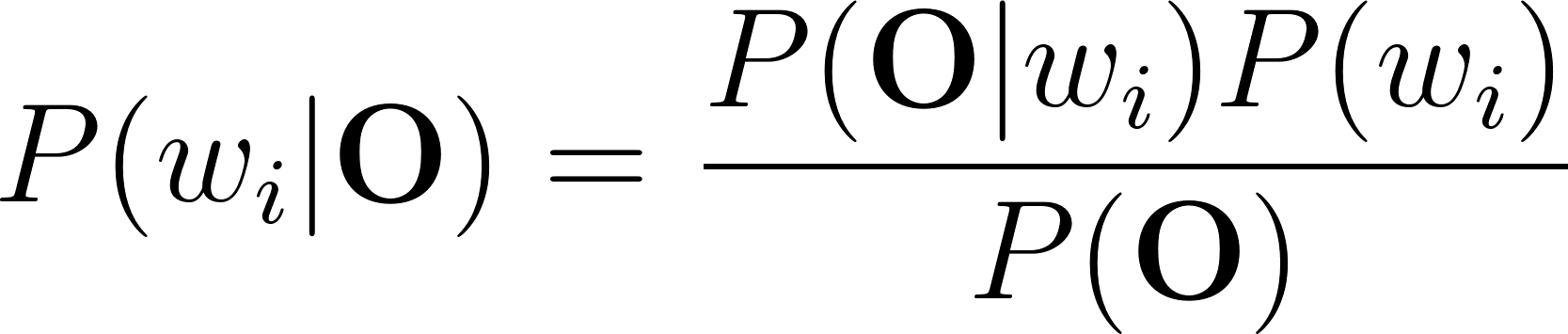
In this system building speech recognition research, the focus was on the development of a language model and an end-to-end speech model comparable in performance to state of the art speech recognition system consisting of an acoustic model and a language model. Low resource language and acoustic modelling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

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.

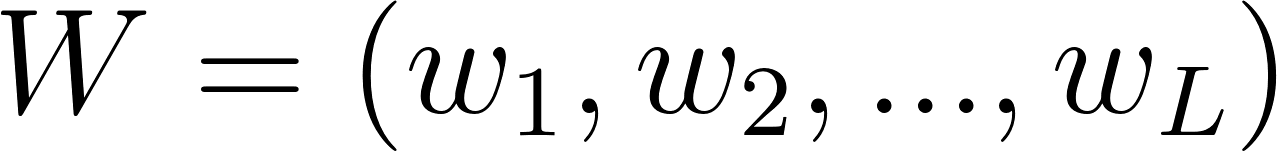
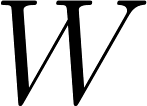
Transfer learning takes the inner representation of knowledge derived from a training algorithm used from one domain and applying this knowledge in a similar domain having different set of system parameters. Early work of this nature for speech recognition is demonstrated in \citep{vu2013multilingual}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

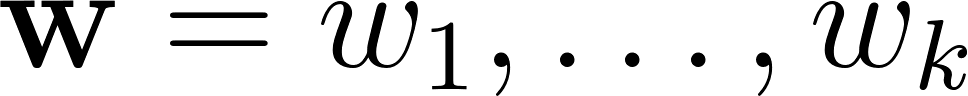
### Low Resource language modelling

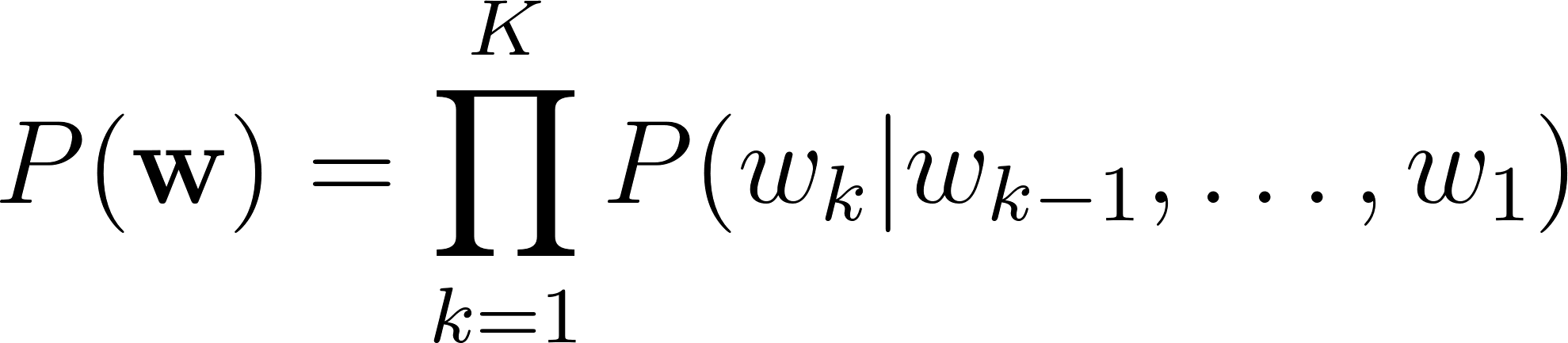
General language modelling is reviewed and then Low resource language modelling is discussed in this section. In section \ref{Ch\_2\_SROverview}, recall from equation \ref{eqn\_2\_3\_bayes\_sr}, the general speech model influenced by Bayes' theorem.

[](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)

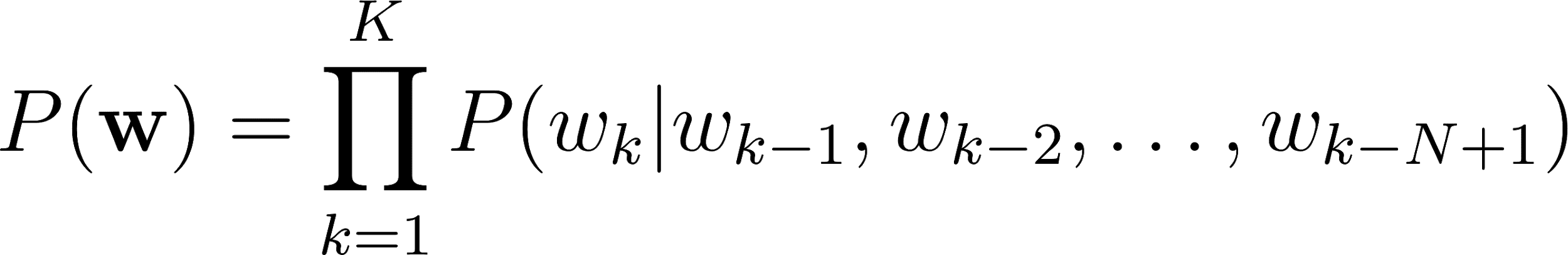
The speech recognition model is a product of an acoustic model (likelihood probability),[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7BO%7D%7Cw_i)%0) and the language model (prior probability),[](https://www.codecogs.com/eqnedit.php?latex=P(w_i)%0). The development of language models for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

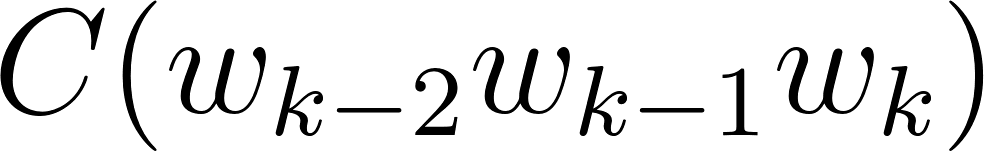
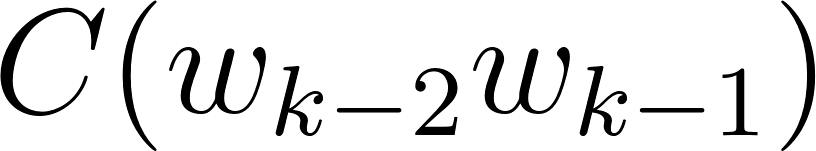
The prior probability of a word sequence [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7Bw%7D%3Dw_1%2C%5Cdots%2Cw_k%0) required in equation (2.2) is given by:

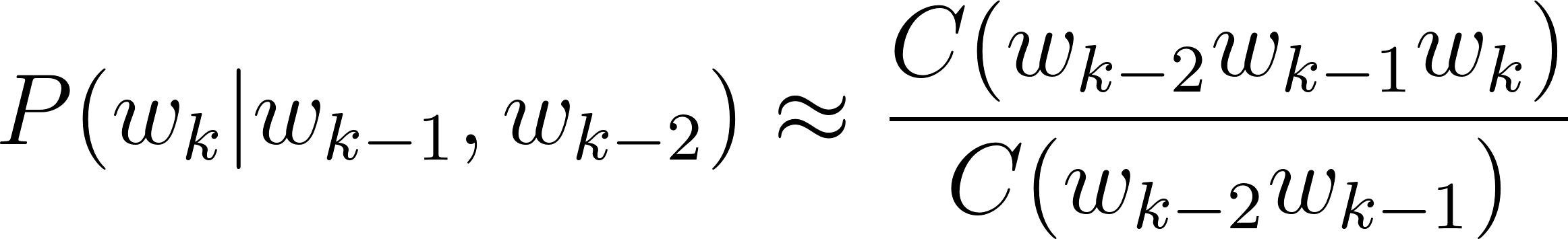
[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7Bw%7D)%3D%5Cprod_%7Bk%3D1%7D%5EKP(w_k%7Cw_%7Bk-1%7D%2C%5Cdots%2Cw_1)%0)

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

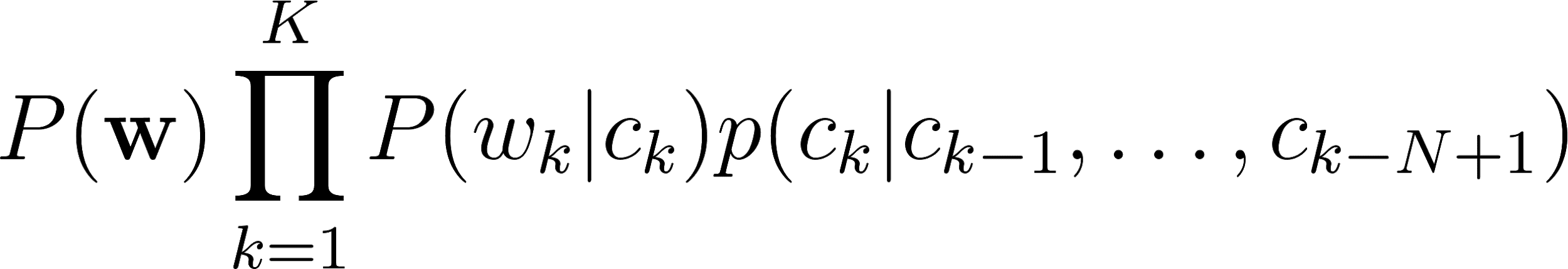
[](https://www.codecogs.com/eqnedit.php?latex=P(%5Cmathbf%7Bw%7D)%3D%5Cprod_%7Bk%3D1%7D%5EKP(w_k%7Cw_%7Bk-1%7D%2Cw_%7Bk-2%7D%2C%5Cdots%2Cw_%7Bk-N%2B1%7D)%0)

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 Okrika language which is a low resource language, bearing in mind that the character level network will reduce the number of parameters required for training, just enough to develop a working language model for the purpose of speech recognition.

### 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 swap-hat method.

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

For phonetic systems, based on the premise that sounds are produced by approximate movements and positions of articulators comprising the human speech sound system is common for 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 bottom-to-top baggage that comes as a byproduct of the generative process sponsored by the HMM-based speech models so that we can gain from 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 possible input-output sequence alignments thereby maximising 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{hannun2014first}. 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).

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.

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 \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 behaviour lost by MFSC coefficients is captured by the scattering transform multi scale co-occurrence coefficients and the scattering representation includes MFSC-like measurements. Together with higher-order co-occurrence coefficients, deep scattering spectrum coefficients represents audio signals similar to models based on cascades of constant-Q filter banks and rectifiers. In particular, second-order co-occurrence coefficients carry important 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 operators 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 \cite{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.

# Methodology

This chapter describes the system building methodology \citep{nunamaker1990systems} as applied to this research. 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 core language technology assistive features whereby speech recognition for low resource languages was chosen as the area this research focuses on.

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 means 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 and claims.

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.

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.

## 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. Choreographe

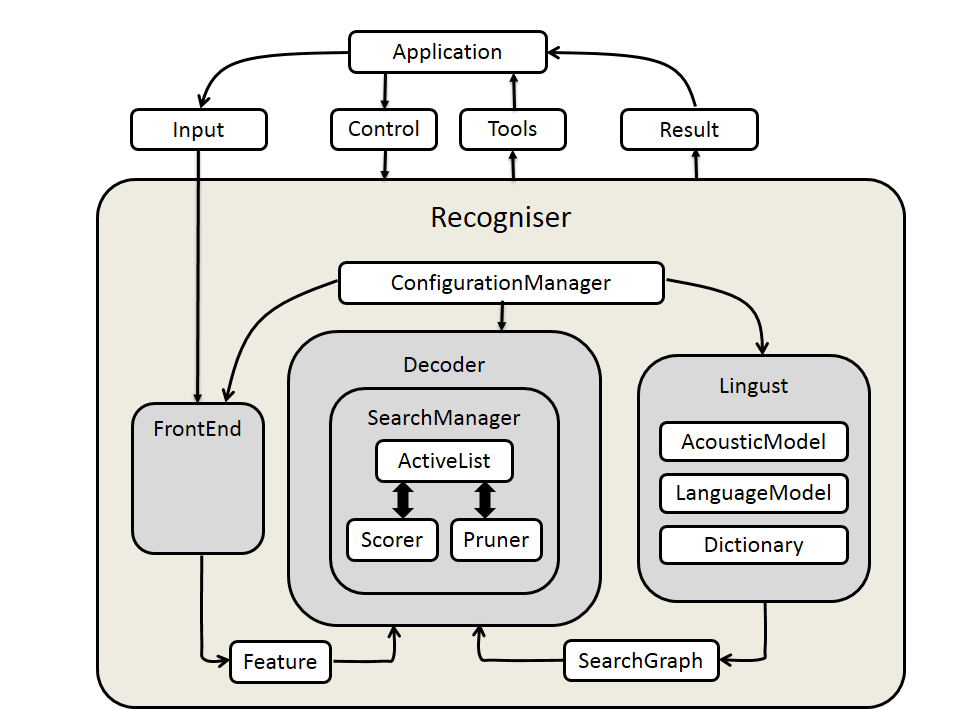
While the research sought to focus on speech models for the Wakirike language, several other sub systems were required for but development of the baseline models in addition to the final model the following system development steps were taken to arrive at the final output models:

1. Autocorrelation 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. Tensor flow 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 5 below. 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 figure it is observed how the components of feature extraction, acoustic modelling, language modelling and decoding are mapped onto the CMU Sphinx system. Note that for identification and clarity classes/modules are capitalised.



*Figure 5: CMU Sphinx4 recogniser system*

In the CMU Sphinx realisation, we can easily map the swimlane components. The three major components of the Recogniser to be mapped to the four swimlane components include the **FrontEnd**, the **Linguist** and the **Decoder**. The **FrontEnd** module implements feature extraction swimlane. The **Linguist** module implements the acoustic model simlane and the language model swimlane. Finally, the **Decoder** module implements the decoder swimlane from the UML diagram in Figure 2. The **ConfigurationManager** class is used to determine the behaviour of the recogniser by specifying the parameters of the other modules.

In the actual 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 decoderare **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 summarises the submodules (Walker et al., 2004).

#### FrontEnd Module

Being consistent with having a “pluggable” framework, CMU Sphinx4 has the ability of most of its components being 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 is 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 two swimlanes in our UML diagram in Section II; The acoustic model swimlane and the language model swimlane. 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 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 (Carnegie Mellon University (CMU) Sphinx., 2015)

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, (Walker et al., 2004) 4 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

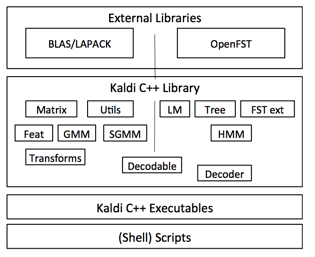
The component architecture of Kaldi is illustrated in the figure below. Modules can be divided into those that utilise the linear algebra libraries and those that use OpenFST. The decodable class forms the link between these two scopes. The rest of the modules lower down the hierarchy are based on modules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

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.

#### 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 XXX 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 XXX.

#### Common Voice training data

The speech corpora 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 not having 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). 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{}. 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{}. 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.

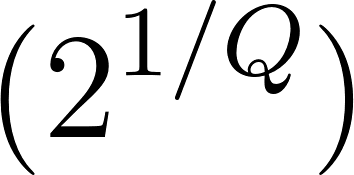
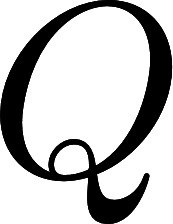
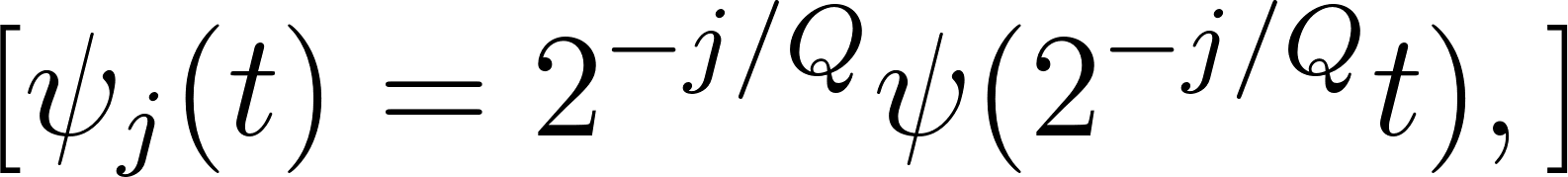
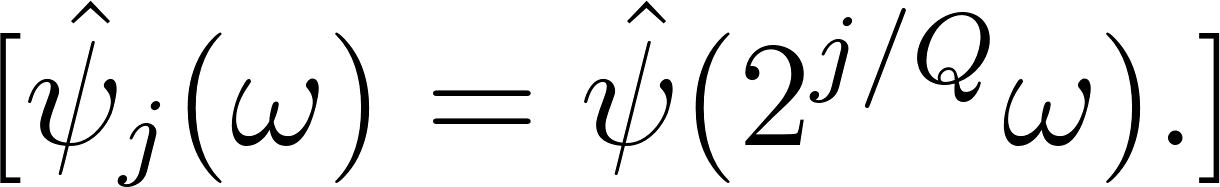
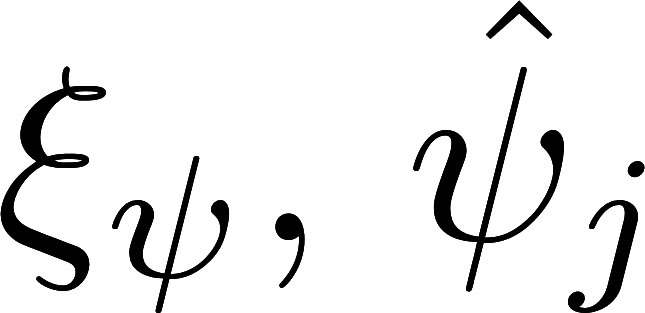
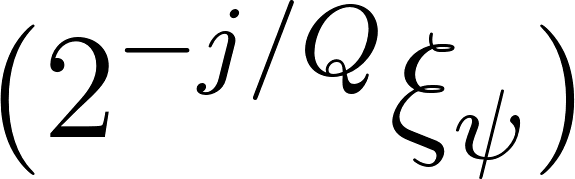
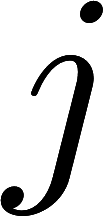
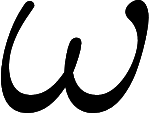
[](https://www.codecogs.com/eqnedit.php?latex=%7C%7C%5Ctext%7Bcomplex%20signal%7D%7C%5Cstar%20%5Cmathbf%7BWop%7D%7C%5Cstar(%5Ctext%7Blowpass%20filter%7D)%0) - - - (eqn)

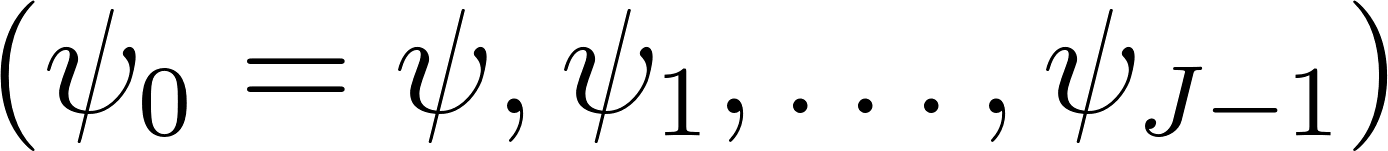
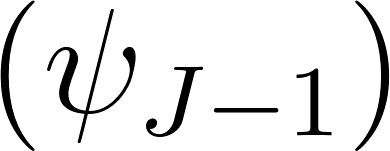
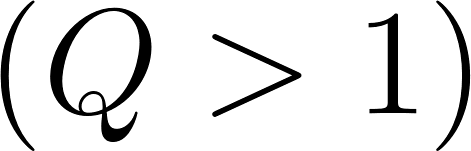
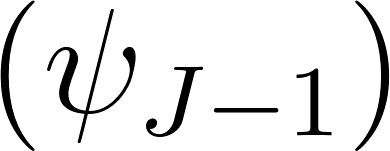
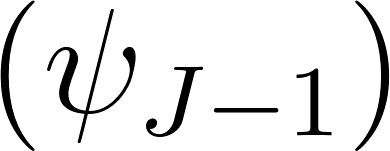
It is the scattering transforms of the audio files that were fed into the DeepSpeech model trainer discussed in section xxx. The architecture of a scattering networks resembles a deep convolutional network in the sense that each subsequent layer is a mapping of all possible paths from the previous layer.

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 one line of code through built-in “factories”, which perform wavelet analysis tasks.

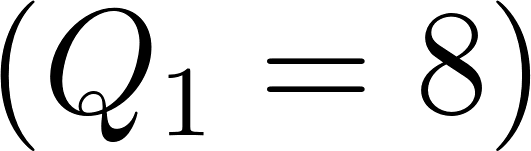
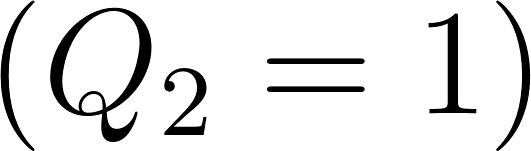
Wavelets are built by dilating a mother wavelet [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi%20%5C)%0) by a factor [](https://www.codecogs.com/eqnedit.php?latex=%5C(%202%5E%7B1%2FQ%7D%20%5C)%0), for some quality factor [](http://www.texrendr.com/?eqn=%5C(%20Q%20%5C)%0), so as to obtain the filter bank [](https://www.codecogs.com/eqnedit.php?latex=%5C%5B%20%5Cpsi_j(t)%20%3D%202%5E%7B-j%2FQ%7D%5Cpsi(2%5E%7B-j%2FQ%7Dt)%2C%5C%5D%0) or, in terms of Fourier transforms, [](https://www.codecogs.com/eqnedit.php?latex=%5C%5B%20%5Chat%5Cpsi_j(%5Comega)%20%3D%20%5Chat%5Cpsi(2%5E%7Bj%2FQ%7D%5Comega).%5C%5D%0) If we suppose that [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Chat%5Cpsi%20%5C)%0) has central frequency [](http://www.texrendr.com/?eqn=%5C(%20%5Cxi_%5Cpsi%20%5C)%2C%20%5C(%5Chat%20%5Cpsi_j%20%5C)%0) will have central frequency [](https://www.codecogs.com/eqnedit.php?latex=%5C(%202%5E%7B-j%2FQ%7D%20%5Cxi_%5Cpsi%5C)%0). The scale index [](http://www.texrendr.com/?eqn=%5C(%20j%20%5C)%0), then is a logarithmic measure of frequency, with a linear increase in [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20j%20%5C)%0) resulting an exponential decrease of [](http://www.texrendr.com/?eqn=%5C(%20%5Comega%20%5C)%0).

In fact, the mother wavelet [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi%20%5C)%0) may only be dilated up to the maximum scale maximum scale [](http://www.texrendr.com/?eqn=%5C(%20J-1%20%5C)%0), resulting in [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20J%20%5C)%0) wavelets [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi_0%20%3D%20%5Cpsi%2C%20%5Cpsi_1%2C%20%5Cldots%2C%20%5Cpsi_%7BJ-1%7D%20%5C)%0). The filter [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi_%7BJ-1%7D%20%5C)%0) has the smallest possible bandwidth in frequency and the largest possible support in time. If [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20Q%20%3E%201%20%5C)%0), this is not sufficient to cover the frequency spectrum. To address this issue, [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi_%7BJ-1%7D%20%5C)%0) is translated in frequency to capture the remaining frequencies. That way, the temporal bandwidth will not exceed that of [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi_%7BJ-1%7D%20%5C)%0). In summary, [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20J%20%5C)%0) determines the maximum wavelet time support [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20T%20%5C)%0), which is equal to the averaging length of the scattering transform, and is chosen by the user.

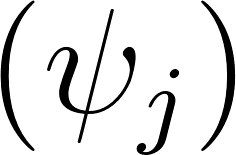
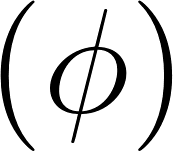
So as to get [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20J%20%5C)%0) in a practical way, ScatNet provides the utility function T\_to\_J, which takes the integer T and the structure filt\_opt as arguments. Thus, a customized set of options for wavelet\_factory\_1d may be built in the following way :

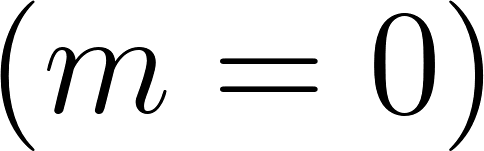
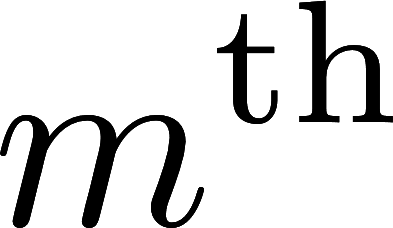
|  |
| --- |
| filt\_opt.Q = [8 8];  filt\_opt.J = T\_to\_J(T, filt\_opt); |

Note that the original prototype T\_to\_J(T,Q) is no longer supported in the new version of ScatNet.

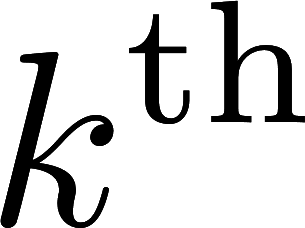
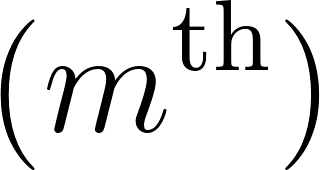
To ensure coverage of the frequency domain without becoming redundant, the mother wavelet [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20%5Cpsi%20%5C)%0) is chosen such that adjacent wavelets barely overlap in frequency. In filters, one has ch that adjacent wavelets barely overlap in frequency. In filters, one has filters, one has ers, one has [](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, which means that the first order has higher resolution in frequency compared to the second. This can be confirmed by looking at the Fourier transforms of the filters above.

#### Exploration of a network by scale paths

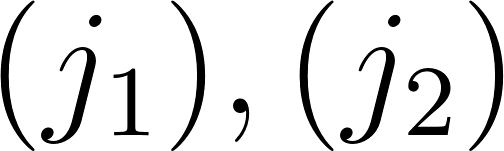
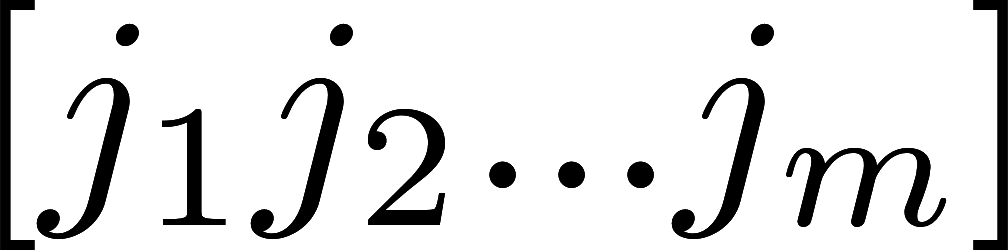
You may recall that scattering coefficients are defined by [](https://www.codecogs.com/eqnedit.php?latex=%5C%5B%20S_1x(t%2Cj_1)%20%3D%20%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cphi(t)%20%5C%5D%20%5C%5B%20S_2x(t%2Cj_1%2Cj_2)%20%3D%20%7C%7Cx%5Cstar%5Cpsi_%7Bj_1%7D%7C%5Cstar%5Cpsi_%7Bj_2%7D%7C%5Cstar%5Cphi(t)%20%5C%5D%0) and so on, 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) a low-pass filter.

In ScatNet, the scattering representation S is a cell array, whose elements correspond to respective layers in the scattering transform. Theoretically, the root of the network has the index [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20m%3D0%20%5C)%0) ; however, since MATLAB uses 1-based indexing, an offset of 1 is required in the cell array : the [](http://www.texrendr.com/?eqn=%5C(m%5E%7B%5Ctext%7Bth%7D%7D%5C)%0) layer is stored as S{1+m}. Each layer has two fields :

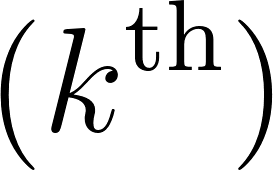
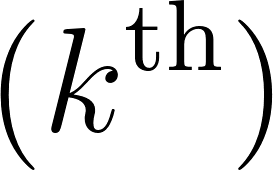
* signal: the cell array of all nodes in the layer, and
* meta: a struct containing the meta-information of the layer, such as arrays of scales and resolutions of each node.

The signal field is a cell array of P elements, each entry corresponding to a node in the scattering layer. The [](http://www.texrendr.com/?eqn=%5C(k%5E%7B%5Ctext%7Bth%7D%7D%5C)%0) node of the [](https://www.codecogs.com/eqnedit.php?latex=%5C(m%5E%7B%5Ctext%7Bth%7D%7D%5C)%0) layer is expressed as :

|  |
| --- |
| S{1+m}.signal{k}; |

An alternative way to identify scattering nodes is by their paths along the scattering network. For translation-invariant scattering representations, these paths are expressed with the wavelet filter scales [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20j_1%20%5C)%2C%20%5C(%20j_2%20%5C)%0), etc. at each layer. To make the correspondence between the cell array S{m+1}.signal of coefficients and these scales, the matrix S{m+1}.meta.j provides the paths of scales of each coefficient in the layer. Namely, S{m+1}.meta.j(:,p) provides the scales [](http://www.texrendr.com/?eqn=%5Bj_1%20j_2%20...%20j_m%5D%0) associated with S{m+1}.signal{p}.

Similarly, the meta field in each layer contains other properties of the scattering coefficients, such as resolution, bandwidth, and so forth. Each field is arranged so that the

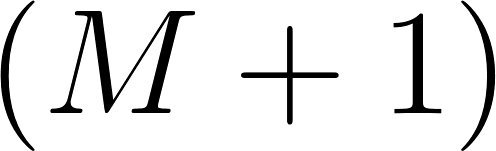
 column is associated with the [](https://www.codecogs.com/eqnedit.php?latex=%5C(k%5E%7B%5Ctext%7Bth%7D%7D%5C)%0) signal in S{m+1}.signal. To get the node corresponding to a specific path, one may use the MATLAB find function ; but be aware that, by default, only paths of increasing scale are computed, so find may return an empty array if the required path does not belong to the network.

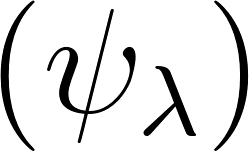
Note that convolutions are implemented with symmetric boundary conditions. This can be changed with the filt\_opt.boundary parameter. For more information, see the documentation of the filter\_bank function.(default 'morlet\_1d'). [J is the number of wavelets].

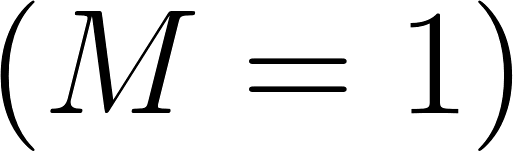
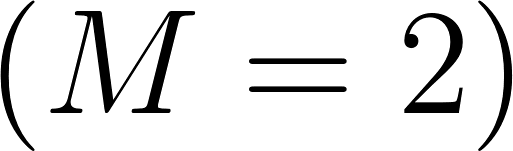
#### Filter banks

The wavelet transforms Wop provided by wavelet\_factory\_1d are merely function handles, with no actual data in them. Instead, they rely on a cell array named filters, which can be extracted from the factory as an optional second output argument:

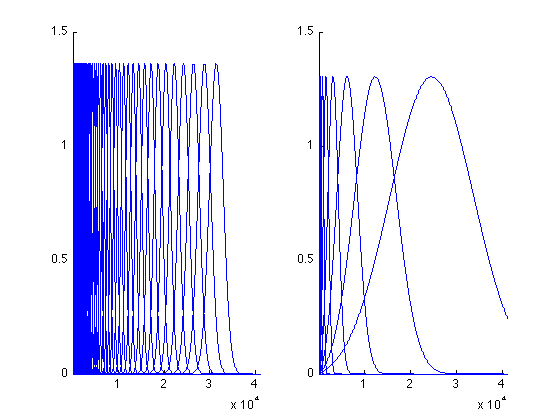
|  |
| --- |
| [Wop,filters] = wavelet\_factory\_1d(N, filt\_opt); |

Like a scattering network, each element in the cell array corresponds to a layer ; however, while S has [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20M%2B1%20%5C)%0) elements, filters has only [](https://www.codecogs.com/eqnedit.php?latex=%5C(%20M%20%5C)%0) elements, as there is no filter bank at the root.

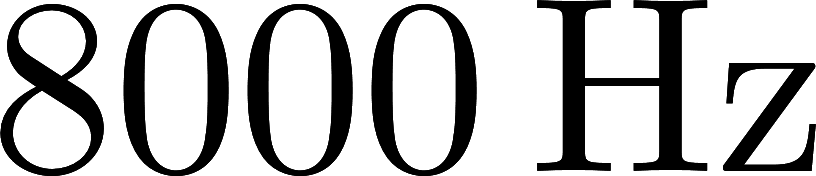
The psi.filter field of filters{m} contains the band-pass filters [](https://www.codecogs.com/eqnedit.php?latex=%5C(%5Cpsi_%5Clambda%5C)%0). These filters are expressed in the Fourier domain, and only their nonzero coefficients are stored. For visualization purposes, one may use realize\_filter to convert them to arrays of length N by padding them with zeros.

Here is a short script to display the filter banks at both 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).

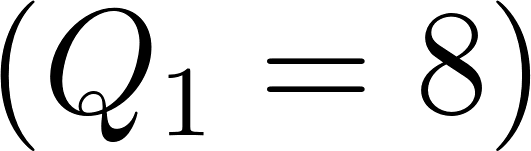
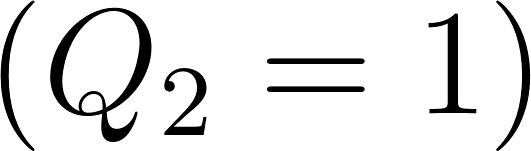
|  |
| --- |
| figure;  for m = 1:2  subplot(1,2,m);  hold on;  for k = 1:length(filters{m}.psi.filter)  plot(realize\_filter(filters{m}.psi.filter{k}, N));  end  hold off;  ylim([0 1.5]);  xlim([1 5\*N/8]);  end |

This should result in the following figure.

This section describes how to compute the scattering spectrum of an audio signal with just three instructions to ScatNet.

The sound file we propose to use for this tutorial is an excerpt of Handel's Messiah, available in all versions of MATLAB. Once loaded the handel.mat file, we choose a length T for the averaging window. Since the signal y is encoded at the sampling rate of [](http://www.texrendr.com/?eqn=%5C(%208000%20%5Ctext%7B%5C%2CHz%7D%20%5C)%0) setting T to 4096 corresponds to about half a second.

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

Before computing the scattering transform of y, we need to define the linear operators that will make up the layers of the scattering networks. In audio processing, these linear operators are constant-Q filter banks. In most experiments, two layers are sufficient, as they capture almost the whole energy in the signal, along an averaging window of up to one second. By default, the quality factors of these 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). By calling default\_filter\_options with the label 'audio', these parameters are automatically integrated into the structure filt\_opt, which contains the options related to the construction of the filter banks.

|  |
| --- |
| filt\_opt = default\_filter\_options('audio', T); |

Scattering networks rely on a set of built-in "wavelet factories", which are suited to specific classes of signals. For translation-invariant representations in one dimension, one may call wavelet\_factory\_1d in the following way :

|  |
| --- |
| Wop = wavelet\_factory\_1d(N, filt\_opt); |

The previous instruction may take a few seconds on a personal computer, as it requires to build many filters at once.

Note : with the new version of ScatNet, the length N of the signal y no longer needs to be a multiple of the length T of the averaging window.

Finally, the scattering transform of y is computed with the generic function scat, which is at the core of the ScatNet toolbox.

|  |
| --- |
| S = scat(y, Wop); |

Most often, scat requires less computer time than wavelet\_factory\_1d. Once the factory has been run, scattering transforms may be calculated iteratively with the same operators Wop, provided the input signals have the same length. More information about the vectorisation of several scattering transforms of one-dimensional signals may be found in the [Batch processing](https://www.di.ens.fr/data/software/scatnet/quickstart-audio/batch-processing) section of this tutorial.

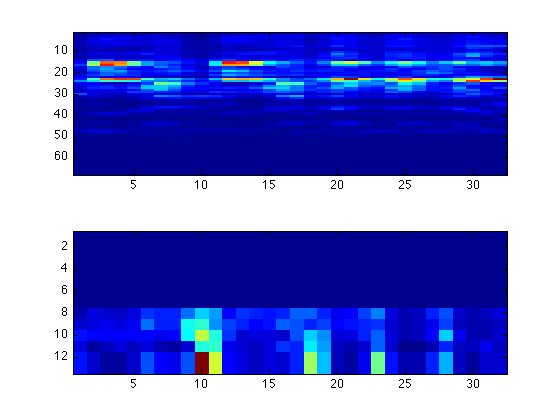
#### Display and Formatting

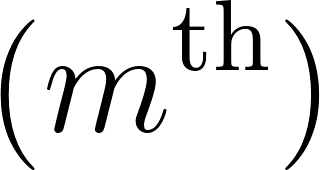
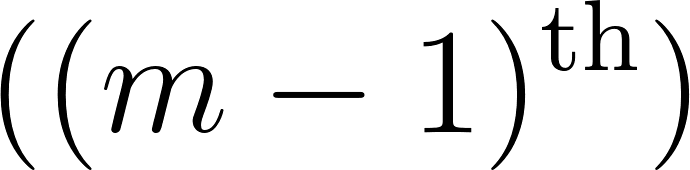
The scattergram built-in function in ScatNet provides a spectrogram-like visualization of the second-order, translation-invariant scattering transform of one-dimensional signals. The top image is made of first-order coefficients, organized in a time-scale matrix. Very low frequencies, i.e. large scales, appear at the bottom of the image. For its part, the second-order coefficients , for a fixed scale j1, are shown in the bottom image, again in a time-scale matrix.

In the following example, one displays the first-order coefficients and second-order coefficients for an arbitrary j1 = 23. Note that the wildcard [] means that all paths are gathered at the first order.

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

This should give the figure below.

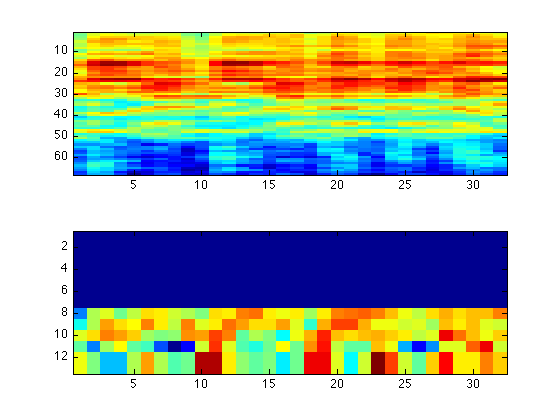


To improve visualization or classification performance, ScatNet offers utility functions to manipulate a scattering representation, such as renormalisation (renorm\_scat) and logarithmic transformation (log\_scat). The former renormalizes second-order coefficients by dividing them by the corresponding first-order coefficients to decorrelate the two : [](https://www.codecogs.com/eqnedit.php?latex=%5C(m%5E%7B%5Ctext%7Bth%7D%7D%5C)%0)-order coefficients are similarly renormalized by dividing by [](https://www.codecogs.com/eqnedit.php?latex=%5C((m-1)%5E%7B%5Ctext%7Bth%7D%7D%5C)%0)-order coefficients. The latter computes the logarithm of the coefficients at each point. Both of these functions use the format described above for the scattering transform for both input and output.

Let's renormalize the coefficients and compute the logarithm.

|  |
| --- |
| S = renorm\_scat(S);  S = log\_scat(S);    scattergram(S{2},[],S{3},j1); |

This results in a figure where much more details are noticeable.



#### Batch Processing

In order to use the scattering coefficients for classification or other tasks, we need to assemble them together in vector form. To do this, we use the function format\_scat.

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

Here, S\_table will be a P-by-N table, where P is the total number of scattering coefficients (all orders combined) and N is the number of time points. Now, S\_table can be fed into a classifier for training or testing.

The ScatNet toolbox also contains a pipeline for classification using an affine space classifier or a support vector machine (SVM). To use it, we need to create a database of feature vectors by calculating the scattering coefficients of a collection of audio files. Here, we shall consider the[GTZAN dataset](http://marsyasweb.appspot.com/download/data_sets/) for musical genre classification.

First, we create the list of files accompanied by their class labels using the gtzan\_src function.

|  |
| --- |
| src = gtzan\_src('/path/to/gtzan/dataset'); |

Second, we need to specify the feature transform to use. This is done, as before, using wavelet\_factory\_1d and the scattering transform functions.

|  |
| --- |
| 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))))); |

Note that we must specify N = 5\*2^17, corresponding to 29.72 s, because the files in src have been truncated to this length. Finally, we can to specify additional options for generating the features. Here, to speed up training and testing, we decide to only keep every eighth frame of the scattering transform.

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

Now we use the prepare\_database function to calculate the scattering coefficients of the files in src.

|  |
| --- |
| database = prepare\_database(src, feature\_fun, database\_options); |

Depending on the speed of the machine, this can take some time. If you have access to the distributed computing toolbox, invoking matlabpool open before calling prepare\_database allows the function to use multiple cores simultaneously.

### Tensorflow

Deep learning is a branch of artificial intelligence employing deep neural network architectures that has significantly advanced the state-of-the-art in computer vision, speech recognition, natural language processing and other domains. In November 2015, Google released TensorFlow, an open source deep learning software library for defining, training and deploying

machine learning models.

TensorFlow relies on dataflow graphs with mutable state. Tensorflow was initially written in TLA[14]. A dataflow graph consists of nodes and edges, where each node represents an instantiation of an operation, and values flow along the edges. The operations are implemented by kernels that can be run on particular types of devices (for instance, CPUs or GPUs).

Computation graphs intuitively represent complicated functional architectures. Having nodes and vertices, much as in general graphs in computer science, in tensorflow, each graph node represents an operation applied to some input and can generate output passed to other nodes. Another analogy is an assembly line where each machine (node) gets or creates its raw material (input), processes it and then passes output to other machines in an orderly fashion, producing subcomponents and eventually the final product when the assembly process comes to an end.

The main values of interest are tensors: arrays of arbitrary dimensionality where the underlying element type is specified or inferred at graph-construction time. Accordingly, many of the operations are mathematical functions such as matrix multiplication.

In addition, some of the operations may read or update state. In TensorFlow, a variable is a special kind of operation that returns a handle to a tensor. For example, the tensor may contain the weights of a layer in a neural network, which are updated during the training process.

Moreover, in addition to edges for communicating tensors, a graph may include control edges that constrain the order of execution. This order can affect the observable semantics in the presence of mutable state, and it can also affect performance.

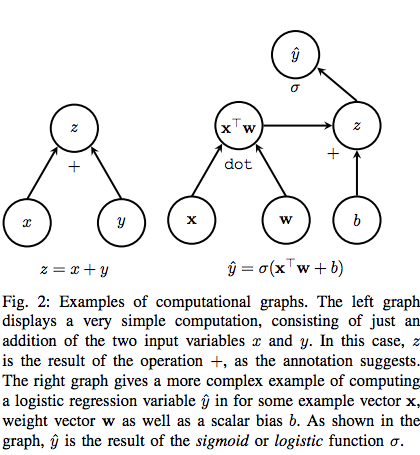
A client typically constructs a graph using a front-end language such as Python. Then the client can make a call to run the graph, specifying which inputs to “feed” and which outputs to “fetch”. TensorFlow propagates the input values, repeatedly applying the operations prescribed by the graph, until no more nodes can fire. The order in which nodes fire is constrained by data dependencies and control edges, but is not necessarily unique. The execution ends with values on the graph’s output edges. In these respects, TensorFlow graphs are similar to expressions in programming languages.

Often, a graph is executed multiple times. Most tensors do not survive past a single execution of the graph. However, mutable state does persist across executions. In a typical application, a graph represents a step of training for a machine-learning model, the parameters of the model are stored in tensors held in variables, and they are updated as part of running the graph.

#### Programming Model

1. Computation Graph architecture

A computational or dataflow graph is a form of directed graph where vertices or nodes describe operations, while edges represent data flowing between these operations. If an output variable z is the result of applying a binary operation to two inputs x and y, then we draw directed edges from x and y to an output node representing z and annotate the vertex with a label describing the performed computation. Examples for computational graphs are given in Figure 2.



* 1. Operations

The major benefit of representing an algorithm in form of a graph is not only the intuitive (visual) expression of dependencies between units of a computational model, but also the fact that the definition of a node within the graph can be kept very general. In TensorFlow, nodes represent operations, which in turn express the combination or transformation of data flowing through the graph [8]. An operation can have zero or more inputs and produce zero or more outputs. As such, an operation may represent a mathematical equation, a variable or constant, a control flow directive, a file I/O operation or even a network communication port. It may seem unintuitive that an operation, which the reader may associate with a function in the mathematical sense, can represent a constant or variable. However, a constant may be thought of as an operation that takes no inputs and always produces the same output corresponding to the constant it represents. Analogously, a variable is really just an operation taking no input and producing the current state or value of that variable. Table ?? gives an overview of different kinds of operations that may be declared in a TensorFlow graph.

Any operation must be backed by an associated implementation. In [8] such an implementation is referred to as the operation’s kernel. A particular kernel is always specifically built for execution on a certain kind of device, such as a CPU, GPU or other hardware unit.

* 1. Tensors

In TensorFlow, edges represent data flowing from one operation to another and are referred to as tensors. A tensor is a multi-dimensional collection of homogeneous values with a fixed, static type. The number of dimensions of a tensor is termed its rank. A tensor’s shape is the tuple describing its size, i.e. the number of components, in each dimension. In the mathematical sense, a tensor is the generalization of two-dimensional matrices, one-dimensional Category Examples Element-wise operations Add, Mul, Exp Matrix operations MatMul, MatrixInverse Value-producing operations Constant, Variable Neural network units SoftMax, ReLU, Conv2D Checkpoint operations Save, Restore TABLE I: Examples for TensorFlow operations [8]. vectors and also scalars, which are simply tensors of rank zero.

In terms of the computational graph, a tensor can be seen as a symbolic handle to one of the outputs of an operation. A tensor itself does not hold or store values in memory, but provides only an interface for retrieving the value referenced by the tensor. When creating an operation in the TensorFlow programming environment, such as for the expression x + y, a tensor object is returned. This tensor may then be supplied as input to other computations, thereby connecting the source and destination operations with an edge. By these means, data flows through a TensorFlow graph.

Next to regular tensors, TensorFlow also provides a SparseTensor data structure, allowing for a more space efficient dictionary-like representation of sparse tensors with only few non-zeros entries.

* 1. Variables

In a typical situation, such as when performing stochastic gradient descent (SGD), the graph of a machine learning model is executed from start to end multiple times for a single experiment. Between two such invocations, the majority of tensors in the graph are destroyed and do not persist. However, it is often necessary to maintain state across evaluations of the graph, such as for the weights and parameters of a neural network. For this purpose, there exist variables in TensorFlow, which are simply special operations that can be added to the computational graph. In detail, variables can be described as persistent, mutable handles to in-memory buffers storing tensors. As such, variables are characterized by a certain shape and a fixed type. To manipulate and update variables, TensorFlow provides the assign family of graph operations. When creating a variable node for a TensorFlow graph, it is necessary to supply a tensor with which the variable is initialized upon graph execution. The shape and data type of the variable is then deduced from this initializer. Interestingly, the variable itself does not store this initial tensor. Rather, constructing a variable results in the addition of three distinct nodes to the graph: 1) The actual variable node, holding the mutable state. 2) An operation producing the initial value, often a constant. 3) An initializer operation, that assigns the initial value to the variable tensor upon evaluation of the graph.

* 1. Sessions

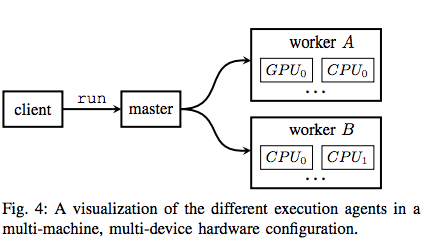
In TensorFlow, the execution of operations and evaluation of tensors may only be performed in a special environment referred to as session. One of the responsibilities of a session is to encapsulate the allocation and management of resources such as variable buffers. Moreover, the Session interface of the TensorFlow library provides a run routine, which is the primary entry point for executing parts or the entirety of a computational graph. This method takes as input the nodes in the graph whose tensors should be computed and returned. Moreover, an optional mapping from arbitrary nodes in the graph to respective replacement values — referred to as feed nodes — may be supplied to run as well [8].

Upon invocation of run, TensorFlow will start at the requested output nodes and work backwards, examining the graph dependencies and computing the full transitive closure of all nodes that must be executed. These nodes may then be assigned to one or many physical execution units (CPUs, GPUs etc.) on one or many machines. The rules by which this assignment takes place are determined by TensorFlow’s placement algorithm, discussed in detail in Subsection ??. Furthermore, as there exists the possibility to specify explicit orderings of node evaluations, called control dependencies, the execution algorithm will ensure that these dependencies are maintained.

1. Execution Model

To execute computational graphs composed of the various elements just discussed, TensorFlow divides the tasks for its implementation among four distinct groups: the client, the master, a set of workers and lastly a number of devices. When the client requests evaluation of a TensorFlow graph via a Session’s run routine, this query is sent to the master process, which in turn delegates the task to one or more worker processes and coordinates their execution. Each worker is subsequently responsible for overseeing one or more devices, which are the physical processing units for which the kernels of an operation are implemented.

Within this model, there are two degrees of scalability. The first degree pertains to scaling the number of machines on which a graph is executed. The second degree refers to the fact that on each machine, there may then be more than one device, such as, for example, five independent GPUs and/or three CPUs. For this reason, there exist two “versions” of TensorFlow, one for local execution on a single machine (but possibly many devices), and one supporting a distributed implementation across many machines and many devices

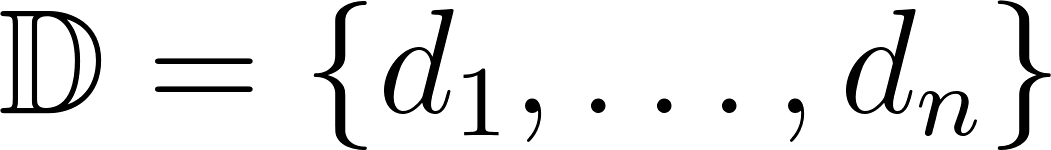
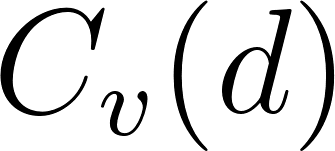


* 1. Devices

Devices are the smallest, most basic entities in the TensorFlow execution model. All nodes in the graph, that is, the kernel of each operation, must eventually be mapped to an available device to be executed. In practice, a device will most often be either a CPU or a GPU. However, TensorFlow supports registration of further kinds of physical execution units by the user. For example, in May 2016, Google announced its Tensor Processing Unit (TPU), a custom built ASIC (application-specific-integrated-circuit) optimized specifically for fast tensor computations [17]. It is thus understandably easy to integrate new device classes as novel hardware emerges.

To oversee the evaluation of nodes on a device, a worker process is spawned by the master. As a worker process may manage one or many devices on a single machine, a device is identified not only by a name, but also an index for its worker group. For example, the first CPU in a particular group may be identified by the string “/cpu:0”.

* 1. Placement algorithms

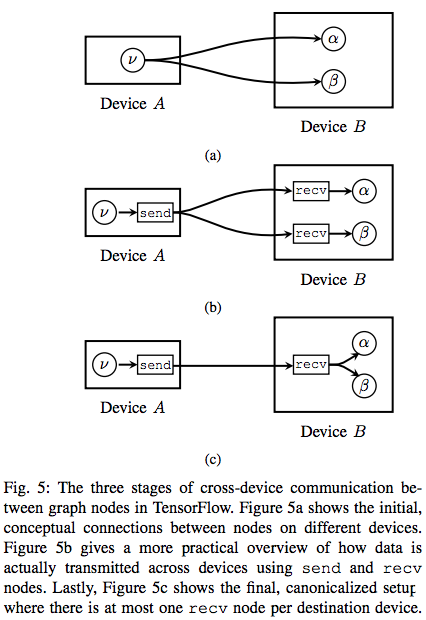
The placement algorithm simulates the execution of the computational graph and traverses its nodes from input tensors to output tensors. To decide on which of the available devices [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BD%7D%3D%5C%7Bd_1%2C%5Cdots%2Cd_n%5C%7D%0) to place a given node [](https://www.codecogs.com/eqnedit.php?latex=%5Cnu%0), encountered during this traversal, the algorithm consults a cost model [](https://www.codecogs.com/eqnedit.php?latex=C_v(d)%0). This cost model takes into account four pieces of information to determine the optimal device [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7Bd%7D%3D%5Cargmin_%7Bd%5Cin%5Cmathbb%7BD%7D%7DC_v(d)%0) on which to place the node during execution.

1. Whether or not there exists an implementation (kernel) for a node on the given device at all. For example, if there is no GPU kernel for a particular operation, any GPU device would automatically incur an infinite cost.
2. Estimates of the size (in bytes) for a node’s input and output tensors.
3. The expected execution time for the kernel on the device.
4. A heuristic for the cost of cross-device (and possibly cross-machine) transmission of the input tensors to the operation, in the case that the input tensors have been placed on nodes different from the one currently under consideration.
   1. Cross-device execution

If the hardware configuration of the user’s system provides more than one device, the placement algorithm will often distribute a graph’s nodes among these devices. This can be seen as partitioning the set of nodes into classes, one per device. As a consequence, there may be cross-device dependencies between nodes that must be handled via a number of additional steps. Let us consider for this two devices A and B with particular focus on a node ν on device A. If ν’s output tensor forms the input to some other operations α, β on device B, there initially exist cross-device edges ν → α and ν → β from device A to device B. This is visualized in Figure 5a.

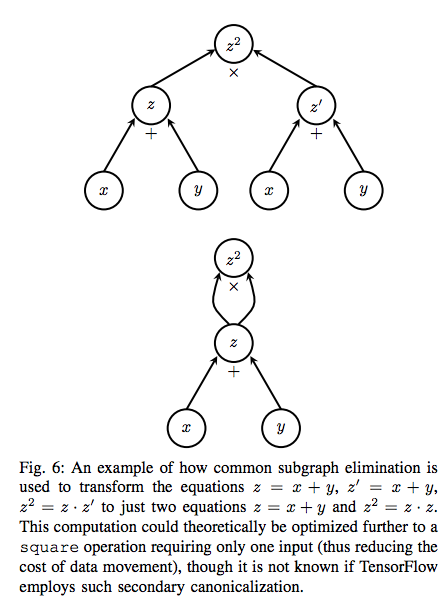
In practice, there must be some means of transmitting ν’s output tensor from A, say a GPU device, to B — maybe a CPU device. For this reason, TensorFlow initially replaces the two edges ν → α and ν → β by three new nodes. On device A, a send node is placed and connected to ν. In tandem, on device B, two recv nodes are instantiated and attached to α and β, respectively. The send and recv nodes are then connected by two additional edges. This step is shown in Figure 5b. During execution of the graph, cross-device communication of data occurs exclusively via these special nodes. When the devices are located on separate machines, transmission between the worker processes on these machines may involve remote communication protocols such as TCP or RDMA.

Finally, an important optimization made by TensorFlow at this step is “canonicalization” of (send, receive) pairs. In the setup displayed in Figure 5b, the existence of each recv node on device B would imply allocation and management of a separate buffer to store ν’s output tensor, so that it may then be fed to nodes α and β, respectively. However, an equivalent and more efficient transformation places only one recv node on device B, streams all output from ν to this single node, and then to the two dependent nodes α and β. This last and final evolution is given in Figure 5c



1. Optimisations
   1. Common subgraph elimination

An optimization performed by many modern compilers is common subexpression elimination, whereby a compiler may possibly replace the computation of an identical value two or more times by a single instance of that computation. The result is then stored in a temporary variable and reused where it was previously recalculated. Similarly, in a TensorFlow graph, it may occur that the same operation is performed on identical inputs more than once. This can be inefficient if the computation happens to be an expensive one. Moreover, it may incur a large memory overhead given that the result of that operation must be held in memory multiple times. Therefore, TensorFlow also employs a common subexpression, or, more aptly put, common subgraph elimination pass prior to execution. For this, the computational graph is traversed and every time two or more operations of the same type (e.g. MatMul) receiving the same input tensors are encountered, they are canonicalized to only one such subgraph. The output tensor of that single operation is then redirected to all dependent nodes. Figure 6 gives an example of common subgraph elimination.



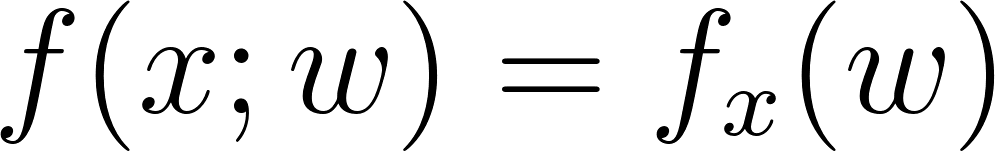
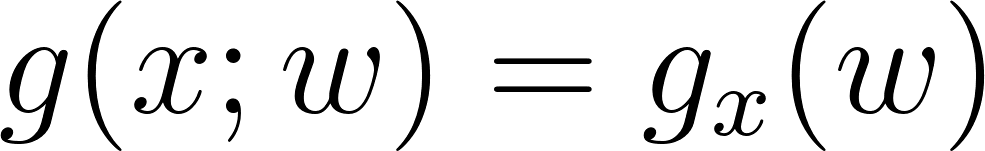
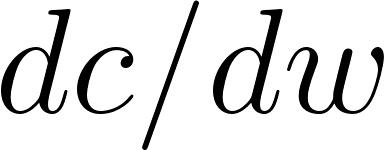
* 1. Scheduling

A simple yet powerful optimization is to schedule node execution as late as possible. Ensuring that the results of operations remain in memory only for the minimum required amount of time reduces peak memory consumption and can thus greatly improve the overall performance of the system. The authors of [8] note that this is especially vital on devices such as GPUs, where memory resources are scarce. Furthermore, careful scheduling also pertains to the activation of send and recv nodes, where not only memory but also network resources are contested.

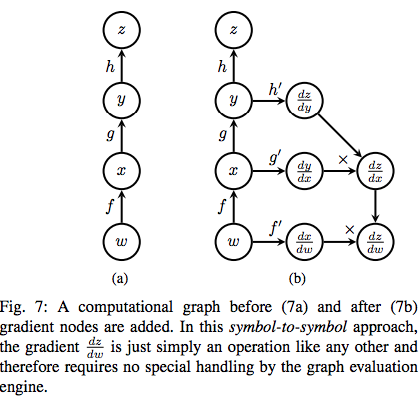
* 1. Lossy compression

One of the primary goals of many machine learning algorithms used for classification, recognition or other tasks is to build robust models. With robust we mean that an optimally trained model should ideally not change its response if it is first fed a signal and then a noisy variation of that signal. As such, these machine learning algorithms typically do not require high precision arithmetic as provided by standard IEEE 754 32-bit floating point values. Rather, 16 bits of precision in the mantissa would do just as well. For this reason, another optimization performed by TensorFlow is the internal addition of conversion nodes to the computational graph, which convert such high-precision 32-bit floating-point values to truncated 16-bit representations when communicating across devices and across machines. On the receiving end, the truncated representation is converted back to 32 bits simply by filling in zeros, rather than rounding [8].

1. Extensions
   1. Backprop nodes

If a neural network consists of two hidden layers represented by functions [](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) with internal weights [](https://www.codecogs.com/eqnedit.php?latex=w%0), we can express the cost for that example as [](https://www.codecogs.com/eqnedit.php?latex=c%3D(f_x%20%5Ccirc%20g_x)(w)%3Df_x(g_x(w))%0). We would then typically calculate the gradient [](https://www.codecogs.com/eqnedit.php?latex=dc%2Fdw%0) of the cost with respect to the weights and use it to update [](https://www.codecogs.com/eqnedit.php?latex=w%0). The backprop algorithm traverses the graph in reverse to compute the cost using the chain rule [](https://www.codecogs.com/eqnedit.php?latex=%5Bf_x(g_x(w))%5D'%3Df'_x(g_x(w))%5Ccdot%20g'x(w)%0).

When TensorFlow needs to compute the gradient of a particular node ν with respect to some other tensor α, it traverses the graph in reverse order from ν to α. Each operation o encountered during this traversal represents a function depending on α and is one of the “links” in the chain (ν ◦ . . . ◦ o ◦ . . .)(α) producing the output tensor of the graph. Therefore, TensorFlow adds a gradient node for each such operation o that takes the gradient of the previous link (the outer function) and multiplies it with its own gradient. At the end of the traversal, there will be a node providing a symbolic handle to the overall target derivative dν/dα , which implicitly implements the back-propagation algorithm. It should now be clear that back-propagation in this symbol-to-symbol approach is just another operation, requiring no exceptional handling. Figure 7 shows how a computational graph may look before and after gradient nodes are added.



In [8] it is noted that symbol-to-symbol derivatives may incur a considerable performance cost and especially result in increased memory overhead. To see why, it is important to understand that there exist two equivalent formulations of the chain rule. The first reuses previous computations and therefore requires them to be stored longer than strictly necessary for forward-propagation. For arbitrary functions f,g and h it is given in Equation 1:



The second possibility for computing the chain rule was already shown, where each function recomputes all of its arguments and invokes every function it depends on. It is given in Equation 2 for reference:



According to [8], TensorFlow currently employs the first approach. Given that the inner-most functions must be recomputed for almost every link of the chain if this approach is not employed, and taking into consideration that this chain may consist of many hundreds or thousands of operations, this choice seems sensible. However, on the flip side, keeping tensors in memory for long periods of time is also not optimal, especially on devices like GPUs where memory resources are scarce. For Equation 2, memory held by tensors could in theory be freed as soon as it has been processed by its graph dependencies. For this reason, in [8] the development team of TensorFlow states that recomputing certain tensors rather than keeping them in memory may be a possible performance improvement for the future.

* 1. Control flow

Some machine learning algorithms may benefit from being able to control the flow of their execution, performing certain steps only under a particular condition or repeating some computation a fixed or variable number of times. For this, TensorFlow provides a set of control flow primitives including if-conditionals and while-loops. The possibility of loops is the reason why a TensorFlow computational graph is not necessarily acyclic. If the number of iterations for of a loop would be fixed and known at graph compile-time, its body could be unrolled into an acyclic sequence of computations, one per loop iteration [5]. However, to support a variable amount of iterations, TensorFlow is forced to jump through an additional set of hoops, as described in [8].

One aspect that must be especially cared for when introducing control flow is back-propagation. In the case of a conditional, where an if-operation returns either one or the other tensor, it must be known which branch was taken by the node during forward-propagation so that gradient nodes are added only to this branch. Moreover, when a loop body (which may be a small graph) was executed a certain number of times, the gradient computation does not only need to know the number of iterations performed, but also requires access to each intermediary value produced. This technique of stepping through a loop in reverse to compute the gradients is referred to as back-propagation through time in [5].

* 1. Checkpoints

It is possible to add Save nodes to the computational graph and connect them to variables whose tensors you wish to serialize. Furthermore, a variable may be connected to a Restore operation, which deserializes a stored tensor at a later point. This is especially useful when training a model over a long period of time to keep track of the model’s performance while reducing the risk of losing any progress made. Also, checkpoints are a vital element to ensuring fault tolerance in a distributed environment [8].

#### Programming Interface

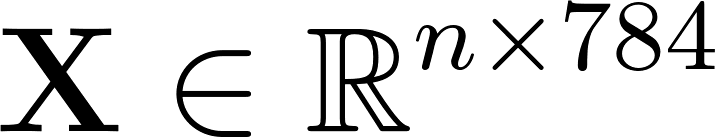
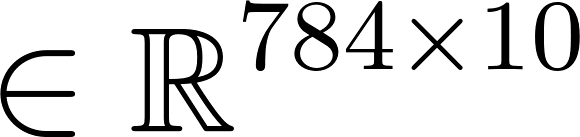
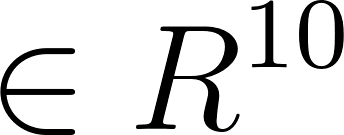
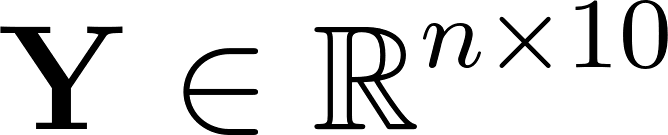
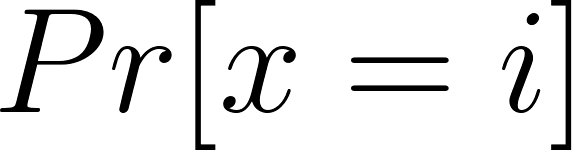
1. Interface

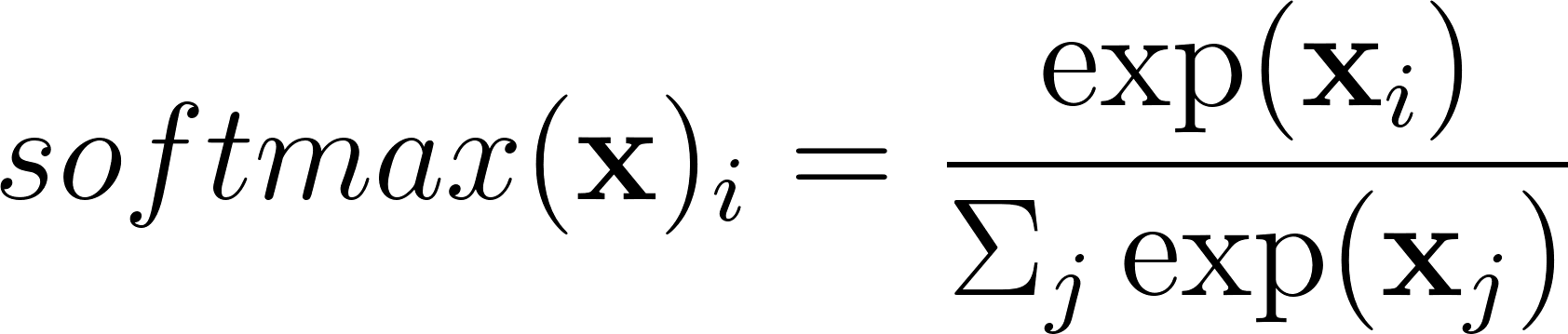
Tensorflow implementation provides two developer interfaces which include the Python interface and the C++ interface. While the python interface offers a rich feature set for creation and execution of computation graphs, the C++ interface is primarily a back end implementation with a much more limited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

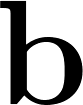
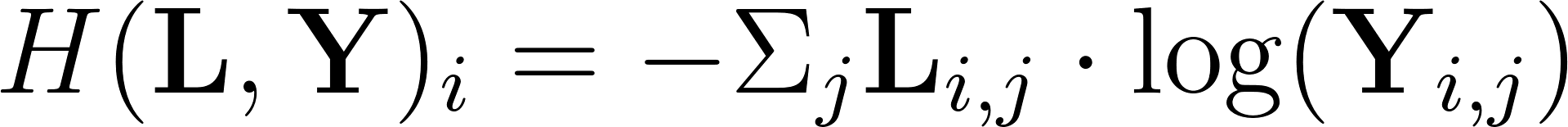
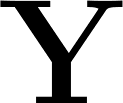
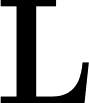
It is noteworthy that the Python API integrates very well with NumPy[21], a popular open source Python numeric and scientific programming library. As such, TensorFlow tensors may be interchanged with NumPy ndarrays in many places.

1. Neural network Basic setup

We train a simple multi-layer perceptron (MLP) with one input and one output layer to classify hand-writtin digits in the MNIST[22] dataset. In this dataset, the examples are small images 28 x 28 pixels depicting handwritten digits from 0 to 9.

Given an example matrix [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BX%7D%5Cin%5Cmathbb%7BR%7D%5E%7Bn%5Ctimes%20784%7D%0) containing [](https://www.codecogs.com/eqnedit.php?latex=n%0) images, the learning task then applies an affine transformation [](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 a weight matrix [](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20%5Cmathbb%7BR%7D%5E%7B784%20%5Ctimes%2010%7D%0), and a bias vector [](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20R%5E%7B10%7D%0). This yields a new matrix [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BY%7D%5Cin%20%5Cmathbb%7BR%7D%5E%7Bn%5Ctimes%2010%7D%0), containing the scores or logits of our model for each example and each possible digit. These scores are more or less arbitrary. To transform the ligits to valid probability distribution, giving the likelihood [](https://www.codecogs.com/eqnedit.php?latex=Pr%5Bx%3Di%5D%0) that the [](https://www.codecogs.com/eqnedit.php?latex=x%0)-th example represents the digit [](https://www.codecogs.com/eqnedit.php?latex=i%0), we make use of a softmax function in the equation below

[](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).

The objective function is pcomputed to give the error or loss of the model given its current trainable 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 by calculating the cross entropy [ between the probability](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) distributions of our estimates [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BY%7D%0) and the one-hot-encoded labels [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbf%7BL%7D%0). More precisely, the mean cross entropy over all examples as the loss.

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.

1. Abstractions

There are high-level Tensorflow-based libraries that offer alternative ways of creating deep learning models to tensorflow while relying on the Tensorflow architecture behind the scenes. These alternative libraries provide top level classes that encapsulate the more complex models and provide black box interfaces for the deep learning developer and can be used to quickly simulate complex neural network architectures. Some of these high level libraries include PrettyTensor[23], TFLearn[24] and Keras[25].

* 1. PrettyTensor
  2. TFLearn

1. Visualisation
   1. Features

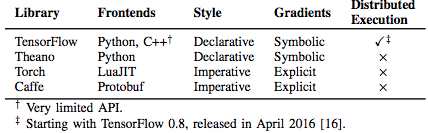
The core feature of TensorBoard is the lucid visualization of computational graphs, exemplified in Figure 8a. Graphs with complex topologies and many layers can be displayed in a clear and organized manner, allowing the user to understand exactly how data flows through it. Especially useful is TensorBoard’s notion of name scopes, whereby nodes or entire subgraphs may be grouped into one visual block, such as a single neural network layer. Such name scopes can then be expanded interactively to show the grouped units in more detail. Figure 8b shows the expansion of one the name scopes of Figure 8a.

Furthermore, TensorBoard allows the user to track the development of individual tensor values over time. For this, you can attach two kinds of summary operations to nodes of the computational graph: scalar summaries and histogram summaries. Scalar summaries show the progression of a scalar tensor value, which can be sampled at certain iteration counts. In this way, you could, for example, observe the accuracy or loss of your model with time. Histogram summary nodes allow the user to track value distributions, such as those of neural network weights or the final softmax estimates. Figures 8c and 8d give examples of scalar and histogram summaries, respectively. Lastly, TensorBoard also allows visualization of images. This can be useful to show the images sampled for each mini-batch of an image classification task, or to visualize the kernel filters of a convolutional neural network [8].

We note especially how interactive the TensorBoard web interface is. Once your computational graph is uploaded, you can pan and zoom the model as well as expand or contract individual name scopes

#### Benchmarks

1. Qualitative



### [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.

### 

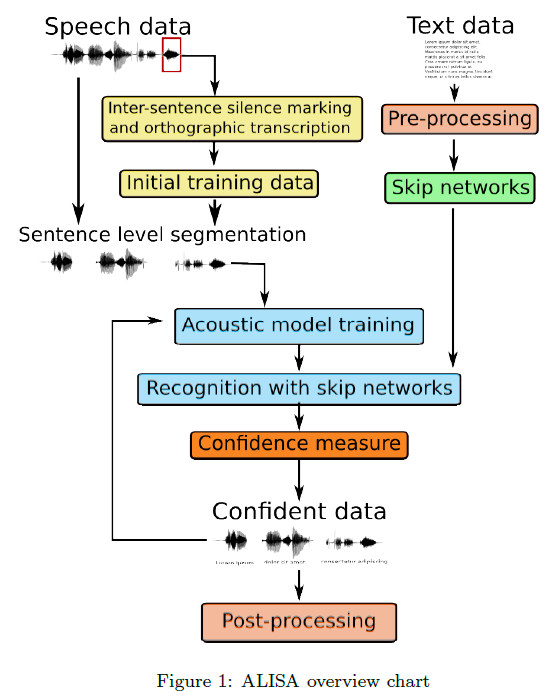
### Alisa

A number of enhancements was observed to be introduced when comparing Alisa (Stan et al., 2016) to the EasyAlign tool above. The researchers refer to Alisa as being lightly supervised, because it requires small amounts of training data. Generally the tool is shown to be highly speech resource optimised offering creation of new speech corpora in a language-independent fashion. The researchers assert that using a two-step method for aligning speech, up to 70% imperfect transcripts often found in online resources can be aligned with a word error rate of less than 0.5%. Ultimately, with this method, Alisa researchers also assert that multilingual transcription can be achieved.

Alisa Enhancements

The motivation behind Alisa was to reduce the time and effort used to gather a large amount of 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.

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 above 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.

We have earlier mentioned that the downside of using grapheme based acoustic model is the increase in the error margin or trade of accuracy. Several steps were introduced in Alisa to minimise this 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 methods used drop down the error margin include the use of discriminative training with the Maximum Mutual Information (MMI) criterion (Schluter & Ney, 2001; Bouchard & Triggs, 2004) and methods described in (Novotney & Schwartz, 2009; Lamel, Gauvain & Adda, 2002). Finally, 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. Secondly, Alisa is restricted to languages that can utilise the english alphabets.

## Initial Experiments

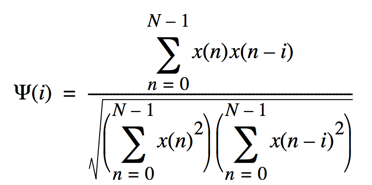
The experiments in the following sections describe initial experiments based on the initial study of a language learning companion before the research was narrowed down to a low resource speech recognition. These paragraphs describe analysis of raw waveforms using autocorrelation signal processing in matlab and experiments made with the nao robot speech processing engine.

### 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 autocorrelation algorithm to attempt to discover a phoneme alphabets in a particular dataset in a semi-supervised fashion.

This method had desirable goals when compared with other segmentation techniques outlined in the previous chapter. The chief being the ability to simulate a posterior distribution statistic from autocorrelation estimate. This presents an unnormalised posterior distribution measurement of every segmented phonemes 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 autocorrelation value is obtained in Picone (1996) from a signal x(n) in the following equation:

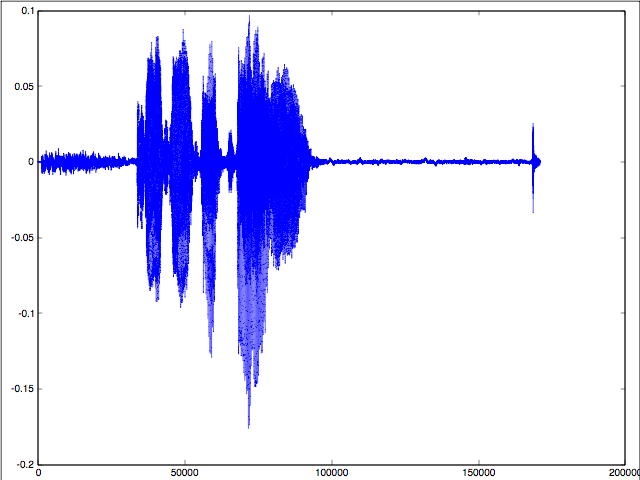
*- - - 4-4*

Based on experimental procedure, estimated locations of similar waveforms representing segmented phonemes are calculated. Although the procedure is subject degrade in the face of most of the difficulties associated with speech recognition, it further emphasises the need for accurate speech production models and pre-processing highlighted in the previous chapter.

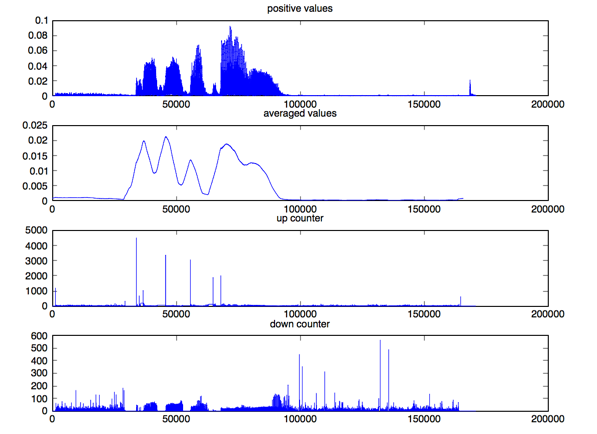
This two stage procedure performs segmentation of phonemes and then discovery of phoneme clusters using a statistical autocorrelation algorithm. The process is described in the following sections.

#### 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 (Section 3.6, Kemper, Jansen & Goldwater (2016)) which is most related to this method also improved on these weaknesses was able to improve on these weakness by using the standard ASR feature extraction and using a combination of acoustic embedding DTW for clustering rather than autocorrelation. In effect using the extracted features for clustering is in theory a better speech estimate with less noise for classification than using an only smoothed audio data.

### Experiments with Nao robot

Nao is a robot used in educational settings. 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 it did not possess the processing power to perform CPU intensive training of acoustic models. However, using pocketsphinx system by CMU, it is possible to perform slower training acoustic models on a more powerful CPU using the deployed acoustic models perform the faster decoding within Nao. The next set of experiments will be based on this method.

## Digit Speech Recognition and Alignment Experiments

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

## Post-Alignment Research Experiments

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

### 

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

### GRU language model for Wakirike language based on tensorflow

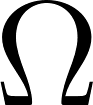
### Bi-Directional LSTM-based end-to-end speech model

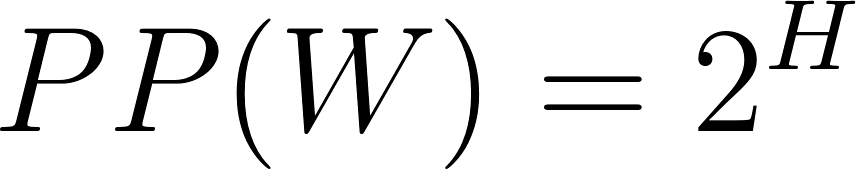
## Method of evaluation

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.

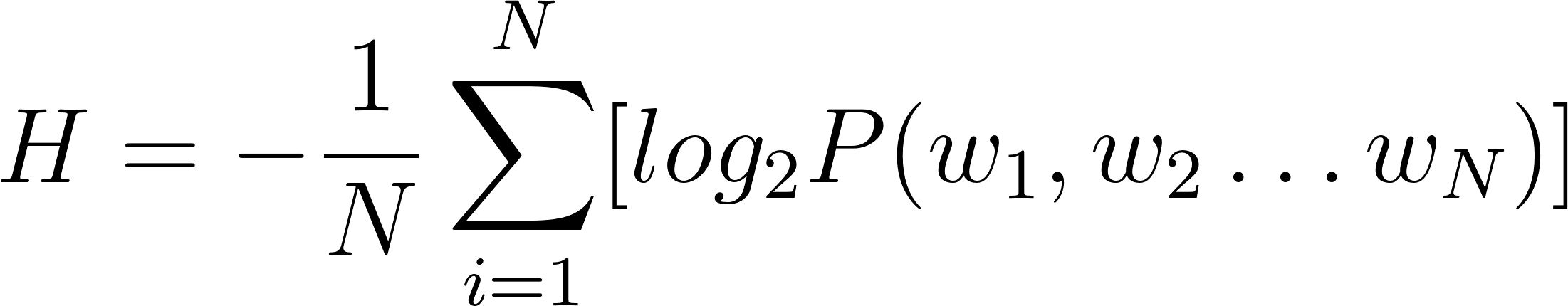
Perplexity measures the complexity of a language that the language model is designed to represent \citep{1976jelinekcontinuous}. In practice, the entropy of a language with an N-gram language model [](https://www.codecogs.com/eqnedit.php?latex=P_N(W)%0) is measured from a set of sentences and is defined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

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

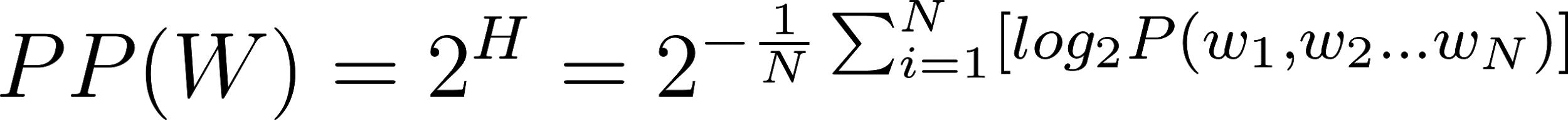
where [](https://www.codecogs.com/eqnedit.php?latex=%5COmega%0) is a set of sentences of the language. The perplexity, which is interpreted as the average word-branching factor, is defined as

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

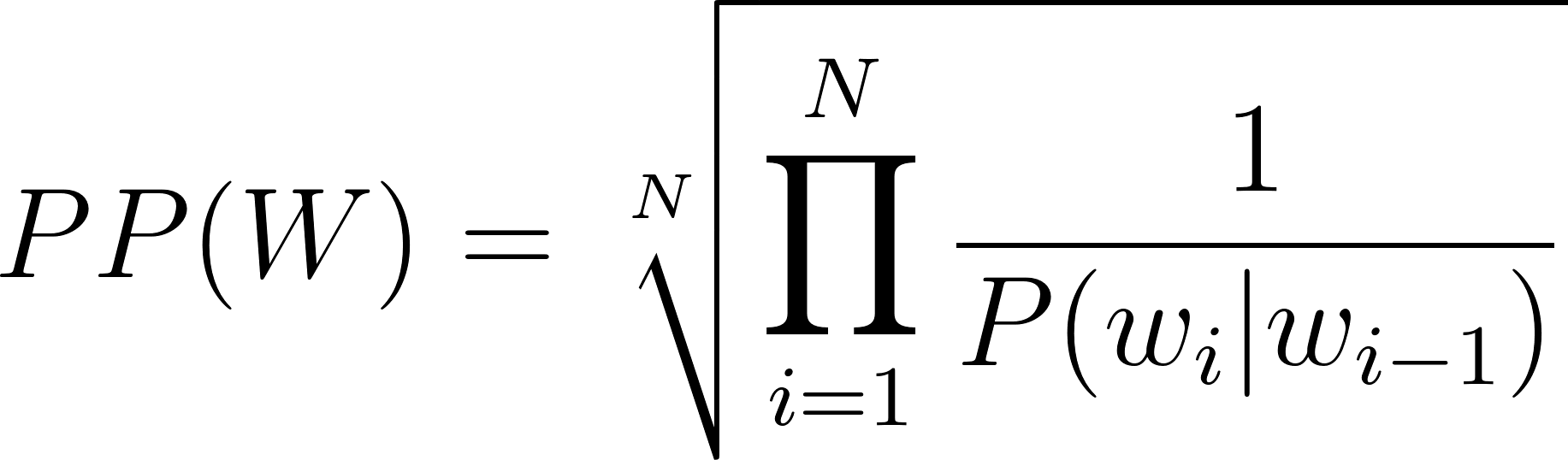
where H is the average entropy of the system or the average log probability defined as

[](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)

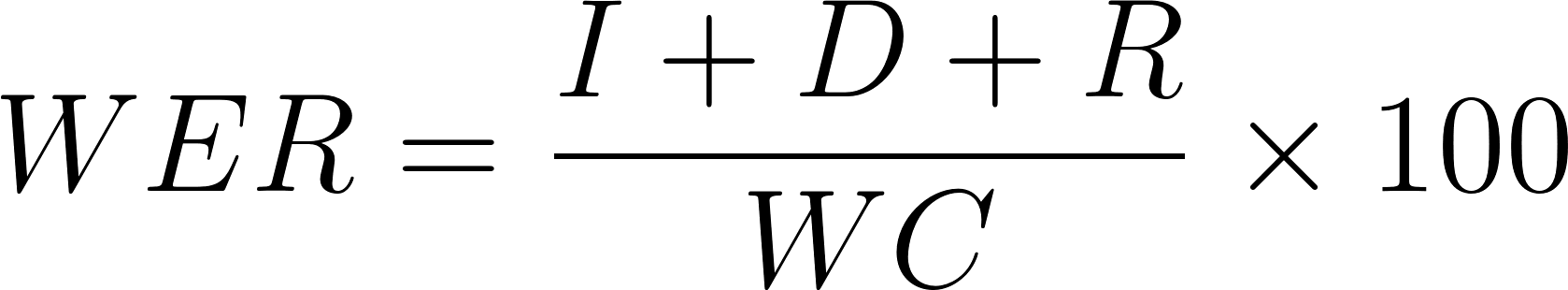
For a bi gram model therefore, equation (\ref{eqn\_c2\_lm07}) becomes

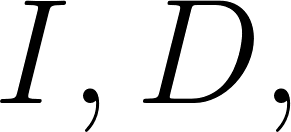
[](https://www.codecogs.com/eqnedit.php?latex=PP(W)%3D2%5EH%3D2%5E%7B-%5Cfrac%7B1%7D%7BN%7D%5Csum_%7Bi%3D1%7D%5EN%5Blog_2P(w_1%2Cw_2%5Cdots%20w_N)%5D%7D%0)

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).

# 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, 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:

\begin{algorithm}[H]

\SetAlgoLined

\KwResult{Optimal weights }

initialise weights randomly\;

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

forward computation \;

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\;

}

\caption{RNN training algorithm}

\end{algorithm}

\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.

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

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

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

\begin{equation}

\Pr( \pi \, | \, x ) = \prod\_{t=1}^{T} y\_{t,\pi\_t}

\label{eqn\_c3\_ctc01}\end{equation}

From the above, it is now possible to define the many-to-one mapping $\mathcal{B} : A'^T \rightarrow A^{\le T}$, from the set of paths onto the set $A^{\le T}$ of possible labellings of $x$ (i.e. the set of sequences of length less than or equal to $T$ over $A$). We do this by removing first the repeated labels and then the blanks from the paths. For example,

\begin{equation}\begin{aligned}\mathcal{B}(a - ab-) &= aab \\ \mathcal{B}(-aa - -abb) &= aab.\end{aligned} \label{eqn\_c3\_ctc02}

\end{equation}

Intuitively, this corresponds to outputting a new label when the network either switches from predicting no label to predicting a label, or from predicting one label to another. As $\mathcal{B}$ is many-to-one, the probability of some labelling $l \in A^{\le T}$ can be calculated by summing the probabilities of all the paths mapped onto it by $\mathcal{B}$:

\begin{equation}

\Pr( l \, | \, x) = \sum\_{\pi \in \mathcal{B}^{-1}(l)} \Pr( \pi \, | \, x)

\label{eqn\_c3\_ct03}\end{equation}

This 'collapsing together' of different paths onto the same labelling is what makes it possible for CTC to use unsegmented data, because it allows the network to predict the labels without knowing in advance where they occur. In theory, it also makes CTC networks unsuitable for tasks where the location of the labels must be determined. However in practice CTC tends to output labels close to where they occur in the input sequence.

\subsection{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 ($\alpha$), multiplied by the probability of all the points from that state to the end of the sequence, the backward variable ($\beta$).

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" $l'$, that caters for blank characters in between regular ones $l$, as defined in $A$. Thus, if $U$ is defined as the length of $l$. Then $U'$ is of length $2U + 1$. CTC therefore integrates probability distributions of transitions between blank and non-blank labels at the same time CTC calculates those transition occurring between pairs of distinct non-blank labels. The forward variable, $\alpha(t,u)$ now becomes the summed probability of all length $t$ paths that are mapped by $\mathcal{B}$ onto the length $\left \lfloor{u/2}\right \rfloor$ prefix of $l$. (Note, $\left \lfloor{u/2}\right \rfloor$ is the floor of $u/2$, the greatest integer less than or equal to $u/2$.) For some sequence $s$, let $s\_{p:q}$ denote the sub-sequence $s\_p, s\_{p+1},\dots,s\_{q-1},s\_q$, and define the set $V(t,u) \equiv \{ \pi \in A'^t : \mathcal{B}(\pi) = l\_{1:\left \lfloor{u/2}\right \rfloor} \text{ and } \pi\_t = l'\_u \}$. $\alpha(t,u)$ then becomes

\begin{equation}

\alpha(t,u) \equiv \sum\_{\pi \in V(t,u)} \prod\_{i=1}^{t} y\_{i,\pi\_i}

\label{eqn\_c3\_ctc04}

\end{equation}

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

\begin{equation}

\Pr( l \, | \, x) = \alpha(T, U') + \alpha(T, U' - 1)

\label{eqn\_c3\_ctc05}

\end{equation}

All correct paths must start with either a blank $(b)$ or the first symbol in $l$ $(l\_1)$, yielding the following initial conditions:

\begin{equation}

\begin{aligned}\alpha(1, 1) &= y\_{1,b} \\ \alpha(1, 2) &= y\_{1,l\_1} \\ \alpha(1, u) &= 0, \, \forall u > 2 \end{aligned}

\label{eqn\_c3\_ctc05}\end{equation}

Thereafter the variables can be calculated recursively:

\begin{equation}

\alpha(t,u) = y\_{t, l'\_u} \sum\_{i = f(u)}^{u} \alpha(t-1, i)

\label{eqn\_c3\_ctc06}\end{equation}

where

\begin{equation}

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

\label{eqn\_c3\_ctc07}\end{equation}

Graphically we can express the recurrence relation for $\alpha(t, u)$ as follows.

\begin{figure}

\centering

% Requires \usepackage{graphicx}

\includegraphics[width=9cm]{thesis/images/Lattice.png}

\caption{Beam Search Lattice Structure \citep{graves2006connectionist}}

\label{fig\_3\_a\_lattice}

\end{figure}

where $t$ runs along the $x$ axis and $u$ runs along the $y$ axis. The black circles of the diagram represent $blank$ elements of $l'$ while the white circles represent non-$blank$ elements of $l'$. The arrows represent computational dependencies derived from our recursion relation for $\alpha(t,u)$. So, for example, the value of $\alpha(2,3)$, corresponding to the $blank$ at $t=2$ and $u=3$, is derived from $\alpha(1,2)$. Similarly, the value of $\alpha(2,2)$, corresponding to the letter $c$ at $t=2$ and $u=2$, is derived from $\alpha(1,2)$ and $\alpha(1,1)$.

\begin{equation}

\alpha(t,u)=0 \quad \forall u < U'-2(T-t)-1

\label{eqn\_c3\_ctc07}\end{equation}

because these variables correspond to states for which there are not enough time-steps left to complete the sequence. We also impose the boundary condition

\begin{equation}

\alpha(t, 0) = 0 \quad \forall t

\label{eqn\_c3\_ctc08}

\end{equation}

The backward variables $\beta(t,u)$ are defined as the summed probabilities of all paths starting at $t + 1$ that "complete" $l$ when appended to any path $\hat{\pi}$ contributing to $\alpha(t,u)$. Define $W(t,u) \equiv \{ \pi \in A'^{T-t} : \mathcal{B}(\hat{\pi} + \pi) = l \, \, \forall \hat{\pi} \in V(t,u) \}$. Then

\begin{equation}

\beta(t,u) \equiv \sum\_{\pi \in W(t,u)} \prod\_{i=1}^{T - t} y\_{t + i,\pi\_i} \label{eqn\_c3\_ctc08}\end{equation}

The rules for initialisation of the backward variables are as follows

\begin{equation} \begin{aligned}

\beta(T, U') &= 1 \\

\beta(T, U' - 1) &= 1 \\

\beta(T, u) &= 0, \, \forall u < U' - 1

\end{aligned}\label{eqn\_c3\_ctc09}\end{equation}

The rules for recursion are as follows:

\begin{equation}

\beta(t, u) = \sum\_{i = u}^{g(u)} \beta(t+1, i) y\_{t+1, l'\_i}\label{eqn\_c3\_ctc10}\end{equation}

where

\begin{equation}

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

\end{equation}

\subsection{CTC Loss function}

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

\begin{equation}

\mathcal{L}(S) = - \ln \prod\_{(x,z) \in S} \Pr(z \, | \, x) = - \sum\_{(x,z) \in S} \ln \Pr(z \, | \, x)

\label{eqn\_c3\_ctc11}\end{equation}

where $z$ is the output label and $x$ is the input sequence. Since $\mathcal{L}(S)$ 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}.

\begin{equation}

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

\label{eqn\_c3\_ctc12}\end{equation}

and therefore

\begin{equation}

\mathcal{L}(S) = \sum\_{(x,z) \in S} \mathcal{L}(x,z)

\label{eqn\_c3\_ctc12}\end{equation}

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

\begin{equation}

\alpha(t, u) \beta(t, u) = \sum\_{\pi \in X(t,u)} \prod\_{t = 1}^{T} y\_{t, \pi\_t}\label{eqn\_c3\_ctc13}\end{equation}

then substituting $\Pr(\pi \, | \, x)$ from the expression in equation \ref{eqn\_c3\_ctc01}, we have

\begin{equation}

\alpha(t, u) \beta(t, u) = \sum\_{\pi \in X(t,u)} \Pr(\pi \, | \, x)

\label{eqn\_c3\_ctc14}\end{equation}

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

\begin{equation}

\Pr(z \, | \, x) = \sum\_{u = 1}^{|z'|} \alpha(t, u) \beta(t, u)

\label{eqn\_c3\_ctc15}\end{equation}

Thus a sample loss is determined by

\begin{equation}

\mathcal{L}(x, z) = - \ln \sum\_{u = 1}^{|z'|} \alpha(t, u) \beta(t, u)\label{eqn\_c3\_ctc16}

\end{equation}

and therefore the overall loss is given by

\begin{equation}\mathcal{L}(S) = -\sum\_{(x,z) \in S} \ln \sum\_{u = 1}^{|z'|} \alpha(t, u) \beta(t, u)

\label{eqn\_c3\_ctc17}\end{equation}

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

\begin{equation}

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

\label{eqn\_c3\_ctc18}\end{equation}

## Chapter 3 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.

# 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

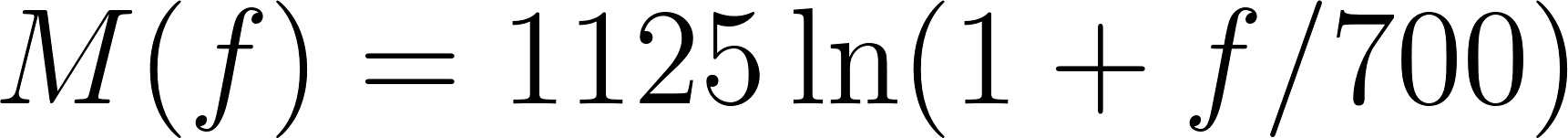
The Fourier and wavelet transform are general means of extracting information from continuous signals using the frequency domain and in the case of the Wavelet transform using both time and frequency domain. The objective in Machine Learning, however, is to extract patterns from the derived information. 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.

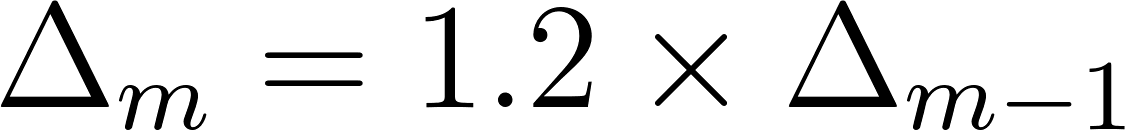
For the case of frequency, the human ear can discriminate lower frequencies more accurately than the higher frequencies. The Mel scale model this behaviour by utilising frequency bins. The frequency bin ranges are narrow at low frequencies and become wide in higher frequencies. In the case of the speech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

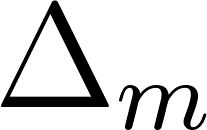
The minimum frequency number of bins used for the Mel scale is 26 bins. In order to determine the frequency ranges we use the following formula to convert between the Mel scale and the regular frequency scale:

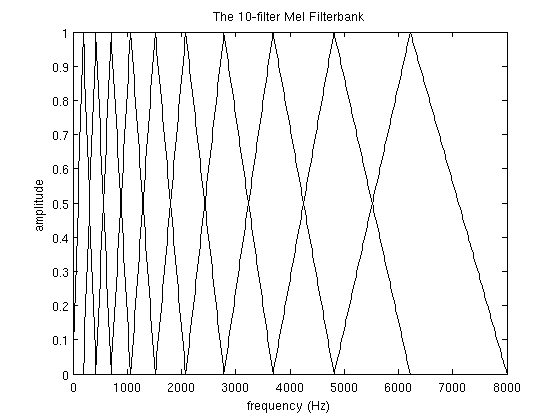
[](https://www.codecogs.com/eqnedit.php?latex=M(f)%3D1125%5Cln(1%2Bf%2F700)%0) - - - (1)

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

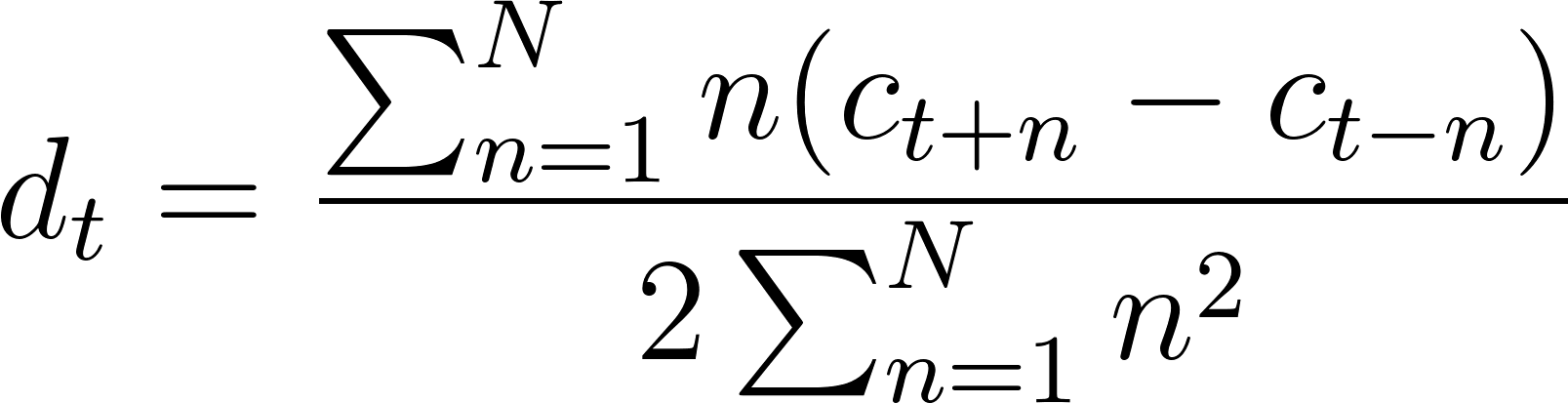
A simple approximation for the Mel scale is obtained by applying linear scale for the first ten filters and for the first 1kHz of the speech frequency range then applying the following formula for the rest \citep{becchetti1999behaviour}:

[](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

In this section reference is made to \citep{anden2011multiscale, anden2014deep, zeghidour2016deep}. For a signal $x$ we define the following transform $W\_x$ as a convolution with a low-pass filter $\phi$ and higher frequency complex analytic wavelets $\psi\_{\lambda\_1}$:

\begin{equation}

Wx=(x\star\phi(t),x\star\psi\_{\lambda\_1}(t))\_{t\in\mathbb{R},\lambda\_1\in\Lambda\_1} \label{eqn\_c4\_dss01}

\end{equation}

We apply a modulus operator to the wavelet coefficients to remove complex phase and extract envelopes at different resolutions

\begin{equation}

|W|x=\left(x\star\phi(t),|x\star\psi\_{\lambda\_1}(t)|\right)\_{t\in\mathbb{R},\lambda\_1\in\Lambda\_1} \label{eqn\_c4\_dss02}

\end{equation}

$S\_0x=x\star\phi(t)$ is locally invariant to translation thanks to the time averaging $\phi$. This time-averaging loses the high frequency information, which is retrieved in the wavelet modulus coefficients $|x\star\psi\_{\lambda\_1}|$. However, these wavelet modulus coefficients are not invariant to translation, and as for $S\_0$, a local translation invariance is obtained by a time averaging which defines the first layer of scattering coefficients

\begin{equation}

S\_1x(t,\psi\_{\lambda\_1})=|x \star\psi\_{\lambda\_1}| \star\phi(t)\label{eqn\_c4\_dss03})

\end{equation}

It is shown in \cite{anden2014deep} that if the wavelets $\psi\_{\lambda\_1}$ have the same frequency resolution as the standard Mel-filters, then the $S\_1x$ coefficients approximate the Mel-filter coefficients. Unlike the Mel-filter banks however, there is a strategy to recover the lost information, by passing the wavelet modulus coefficients $|x\star\phi\_{\lambda\_1}|$ through a bank of higher frequency wavelets $\psi\_{\lambda\_2}$:

\begin{equation}

|W\_2||x\star\phi\_{\lambda\_1}|=\left(|x\star\psi\_{\lambda\_1}|\star\phi,||x\star\psi\_{\lambda\_1}|\star\psi\_{\lambda\_2}|\right)\_{\lambda\_2\in\Lambda\_2} \label{eqn\_c4\_dss04})\end{equation}

This second layer of wavelet modulus coefficients is still not invariant to translation, hence we average these coefficients with a low-pass filter $\phi$ to derive a second layer of of scattering coefficients.

\begin{equation}

|W\_2||x\star\phi\_{\lambda\_1}|=\left(|x\star\psi\_{\lambda\_1}|\star\phi,||x\star\psi\_{\lambda\_1}|\star\psi\_{\lambda\_2}|\right)\_{\lambda\_2\in\Lambda\_2}\label{eqn\_c4\_dss04})\end{equation}

Repeating these successive steps of computing invariant features and retrieving lost information leads to the scattering spectrum, as seen in Fig. 1, however speech signals are almost entirely characterized by the first two layers of the spectrum, that is why a two layers spectrum is typically used for speech representation. It is shown in [6] that this representation is invariant to translations and stable to deformations, while keeping more information than the Mel-filter banks coefficients

\begin{figure}

\centering

% Requires \usepackage{graphicx}

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

\caption{Scattering network - 2 layers deep} \cite{zeghidour2016deep}\label{fig\_4\_3\_scatter}

\end{figure}

## Chapter 4 Summary

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

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

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

# 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.

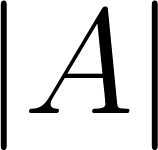
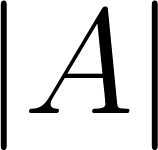
CTC works by making predictions at any point in the input sequence. For the case of speech modelling, CTC makes a character prediction for every time step of the raw audio input speech signal. Although this initially seems counter intuitive, this method models the many to one relationship seen in the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

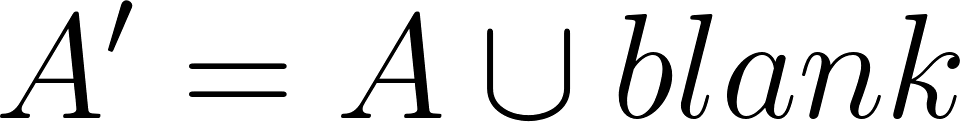
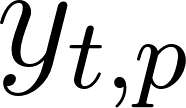
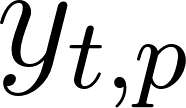
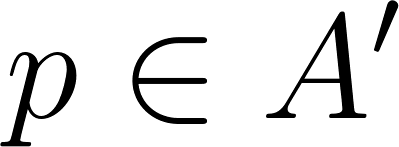
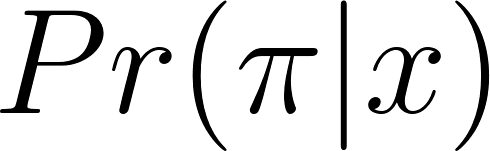
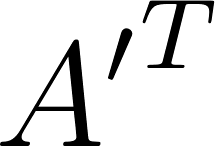
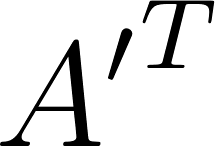
For hybrid DNN-HMM systems, speech or more accurately, acoustic models, require separate training of targets for every time-slice in the input sequence. Secondly, and as a consequence of this, it becomes necessary to segment the audio sequence, in order to provide targets for every time-slice. A third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

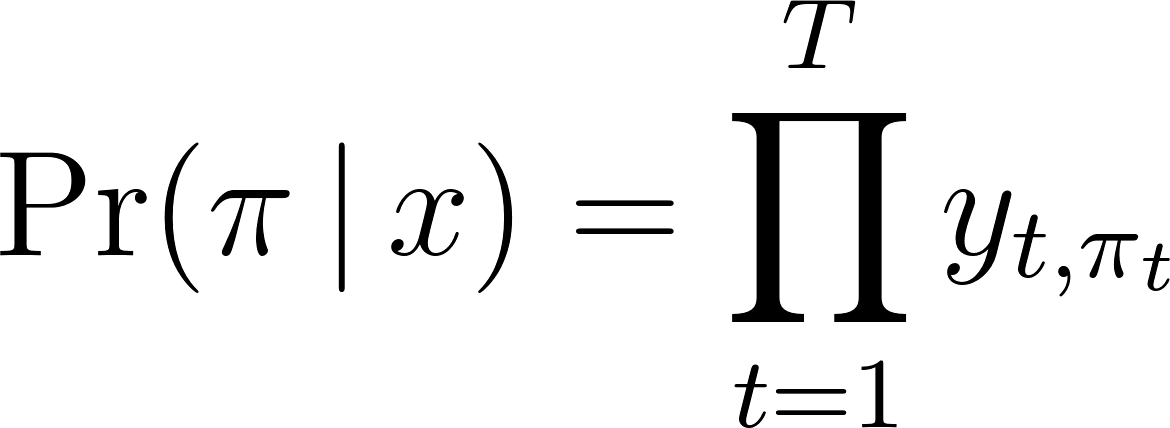
In the CTC case, so long as the overall sequence of labels is correct the network can be optimised to correct the temporal or fuzzy alignments. Since this many to one fuzzy alignment is simultaneously modelled in CTC, then there is no need for pre-segmented data. At the same time, CTC computes probabilities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

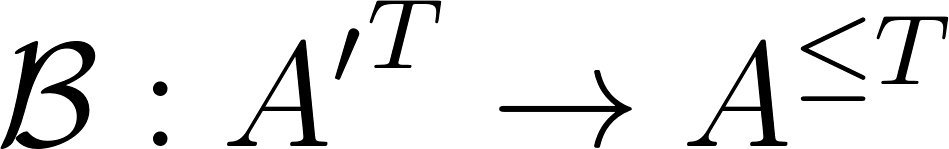
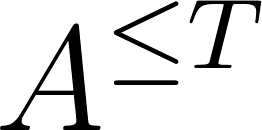
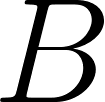
Similar to the HMM sequence model, the CTC algorithm is a sequence model that predicts the next label in a sequence as a cumulative of previous sequences. This section develops the CTC loss function borrowing concepts used in HMM models such as the forward backward algorithm as outlined in \citep{graves2006connectionist}.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

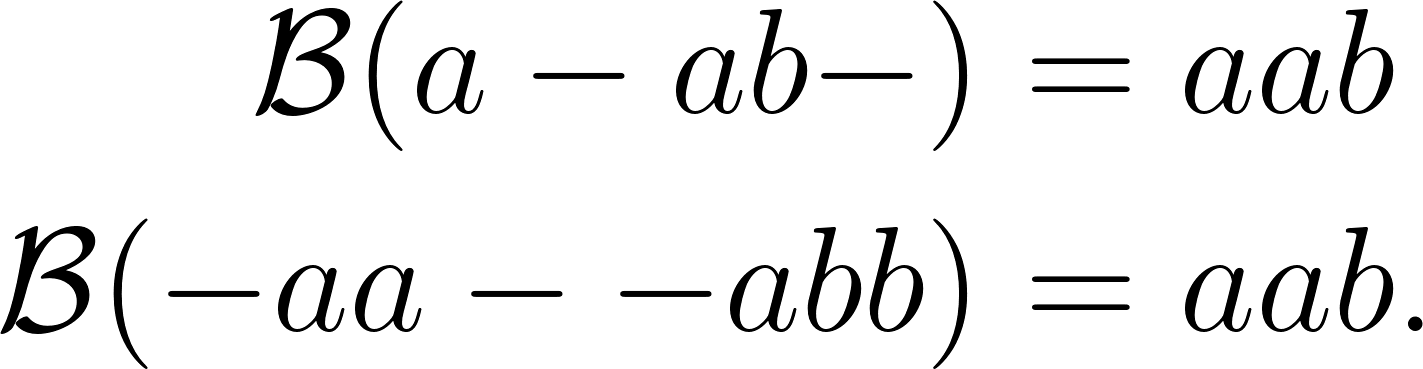
Given two symbols [](https://www.codecogs.com/eqnedit.php?latex=A%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0) such that [](https://www.codecogs.com/eqnedit.php?latex=A%0) has a many to one relationship with [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BB%7D%0),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

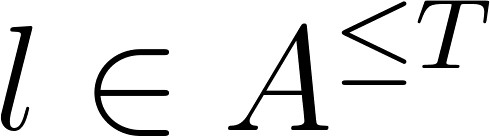
This output models the probability distribution of a complete sequence of arbitrary length [](https://www.codecogs.com/eqnedit.php?latex=%7CA%7C%0) over all possible labels in [](https://www.codecogs.com/eqnedit.php?latex=A%0) from activations within [](https://www.codecogs.com/eqnedit.php?latex=%7CA%7C%0).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

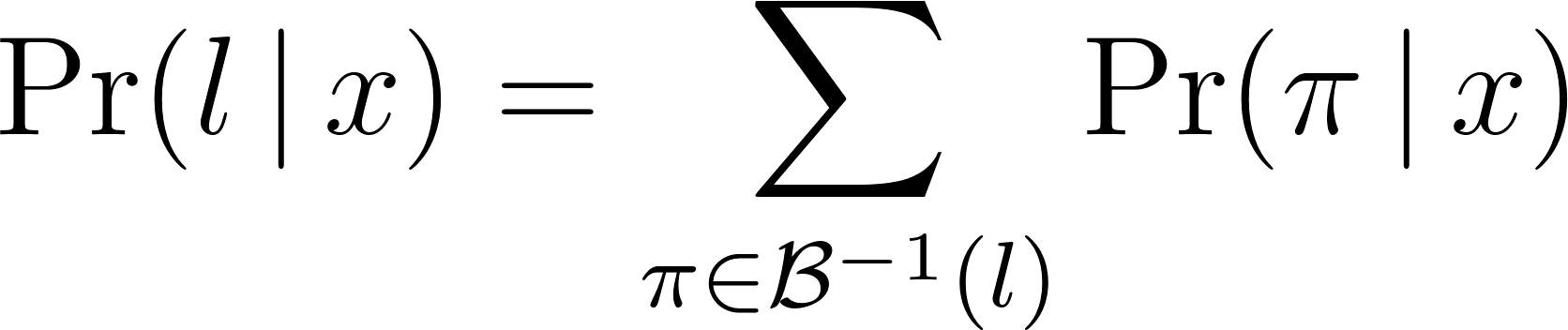
It is now possible to define the extended alphabet [](https://www.codecogs.com/eqnedit.php?latex=A'%20%3D%20A%20%5Ccup%20%7Bblank%7D%0), also, [](https://www.codecogs.com/eqnedit.php?latex=y_%7Bt%2Cp%7D%0) as the activation of network output [](https://www.codecogs.com/eqnedit.php?latex=p%0) at time [](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), from the set of paths [](https://www.codecogs.com/eqnedit.php?latex=A'%5ET%0) onto the set [](https://www.codecogs.com/eqnedit.php?latex=A%5E%7B%5Cle%20T%7D%0) of possible labellings of [](https://www.codecogs.com/eqnedit.php?latex=x%0). [](https://www.codecogs.com/eqnedit.php?latex=%5Cmatcal%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 example,

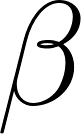
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Baligned%7D%5Cmathcal%7BB%7D(a%20-%20ab-)%20%26%3D%20aab%20%5C%5C%20%5Cmathcal%7BB%7D(-aa%20-%20-abb)%20%26%3D%20aab.%5Cend%7Baligned%7D%0) - - - (2)

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:

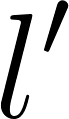
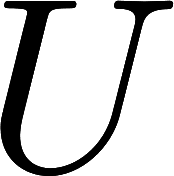
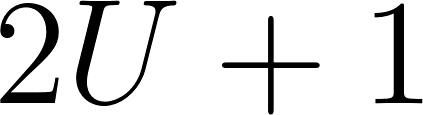
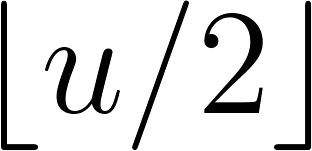
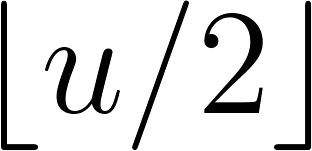
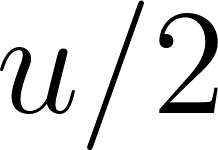
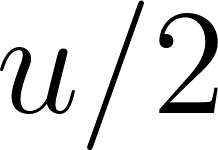
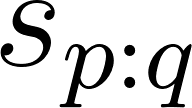
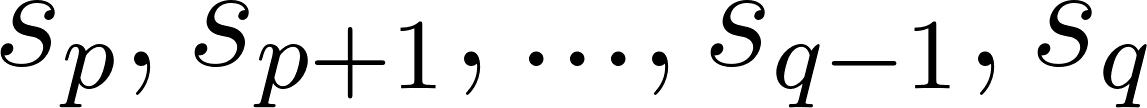
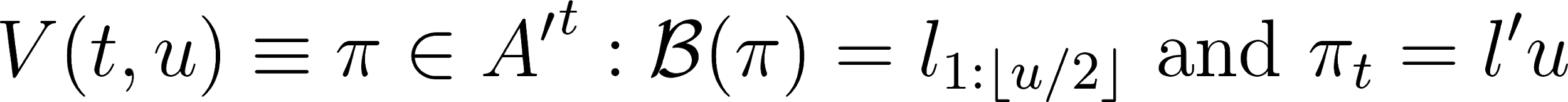
[](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%20l%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20%5Csum_%7B%5Cpi%20%5Cin%20%5Cmathcal%7BB%7D%5E%7B-1%7D(l)%7D%20%5CPr(%20%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (3)

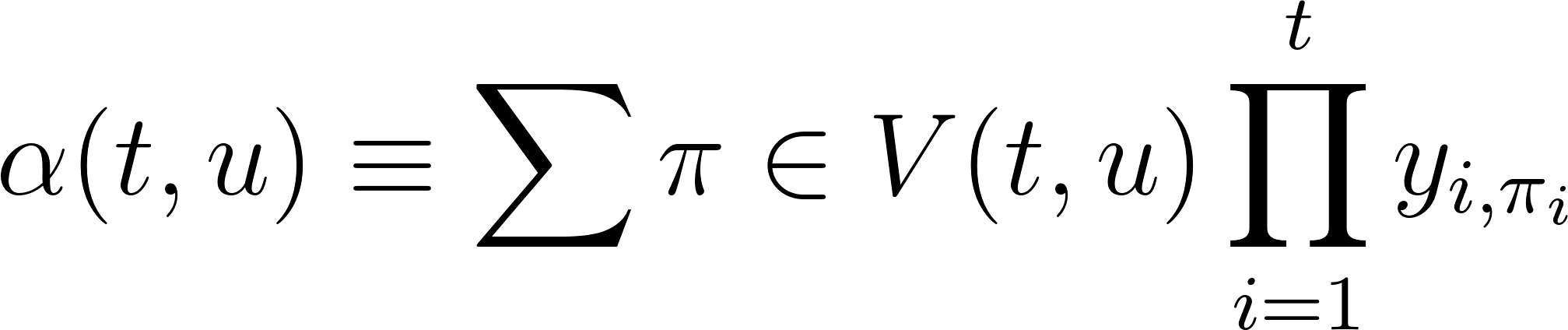
This mapping makes CTC robust to unsegmented data as it predicts all the labels where they occur and later the ‘collapsed’ sequence will be extended over the approximate period where the previous extended sequence occurred thus aligning labels to input sequences on-the-fly without knowing in advance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

## Forward-backward algorithm

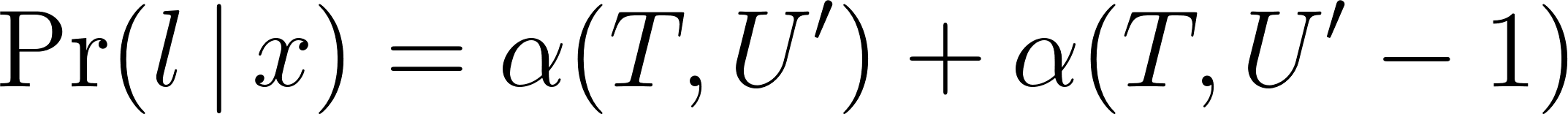
The forward-backward algorithm is used to estimate the probability of a point in the sequence as the product of all point leading up to that point from the initial state, the forward variable ([](https://www.codecogs.com/eqnedit.php?latex=%5Calpha%0)), multiplied by the probability of all the points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

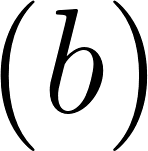
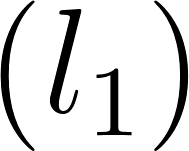
The difference between this estimation and that determined from equation (\ref{eqn\_c3\_ct03}) is the fact that the forward-backward algorithm converts equation (\ref{eqn\_c3\_ct03}) into a form that is both recursive as well as reduces the computational complexity from an otherwise intractable computation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

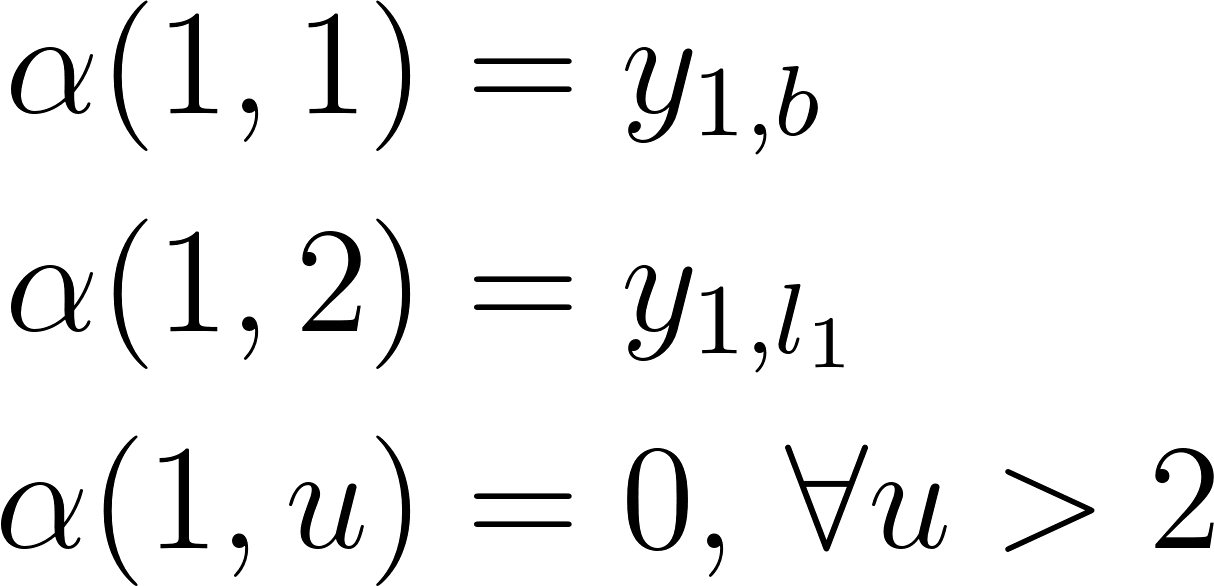
With CTC, consider a modified "label sequence" [](https://www.codecogs.com/eqnedit.php?latex=l'%0), that caters for blank characters in between regular ones [](https://www.codecogs.com/eqnedit.php?latex=l%0), as defined in [](https://www.codecogs.com/eqnedit.php?latex=A'%0). Thus, if [](http://www.texrendr.com/?eqn=U%0) 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)

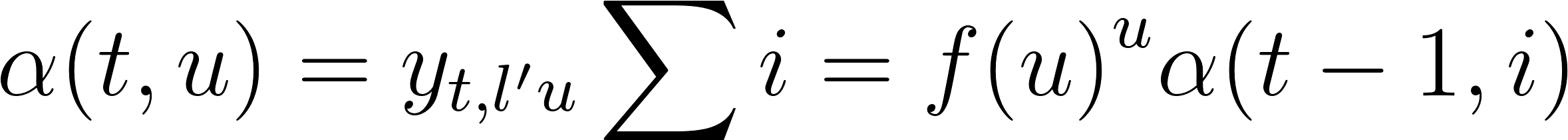
The forward variables at time [](https://www.codecogs.com/eqnedit.php?latex=t%0) can be calculated recursively from those at time [](https://www.codecogs.com/eqnedit.php?latex=t%20-%201%0) and expressed as the sum of the forward variables with and without the final blank at time [](https://www.codecogs.com/eqnedit.php?latex=T%0).

[](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)

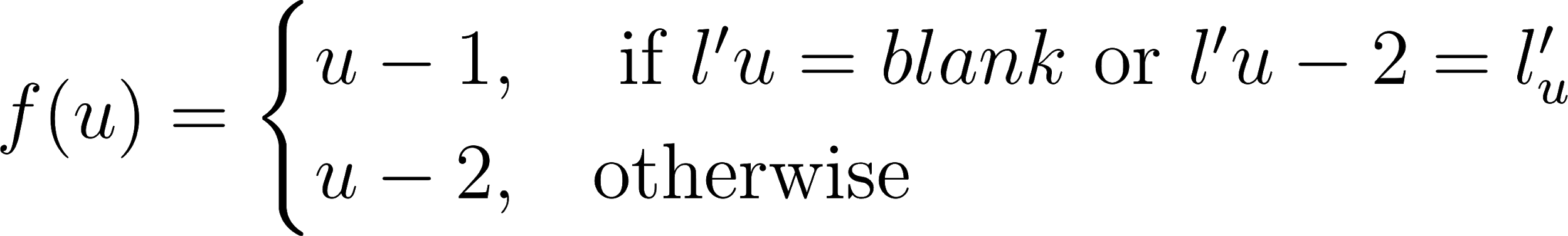
For the initial conditions, correct paths begin with a blank symbol [](https://www.codecogs.com/eqnedit.php?latex=(b)%0) and the first symbol [](https://www.codecogs.com/eqnedit.php?latex=(l_1)%0):

[](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)

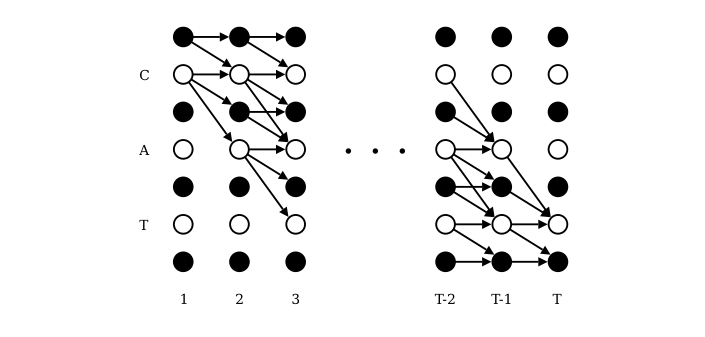
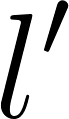
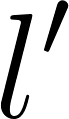
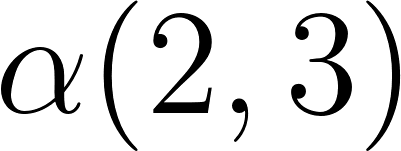
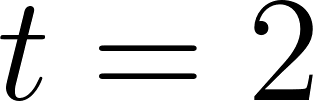
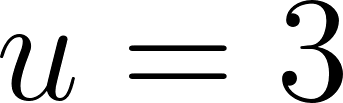
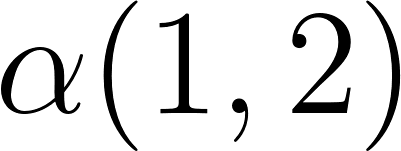
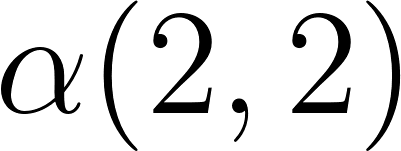
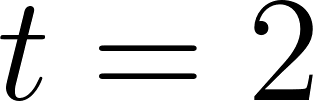
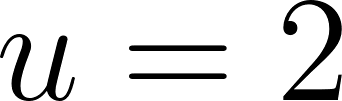
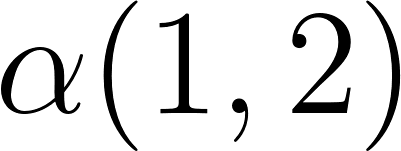
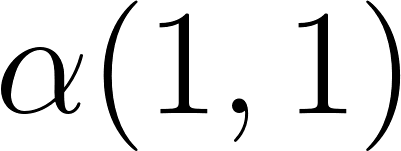
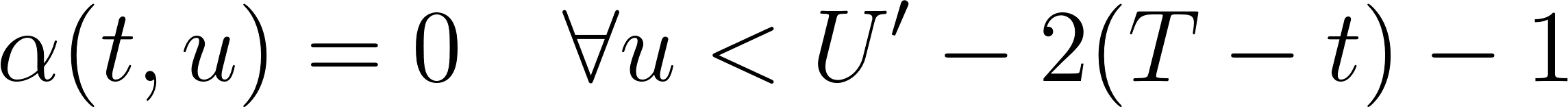
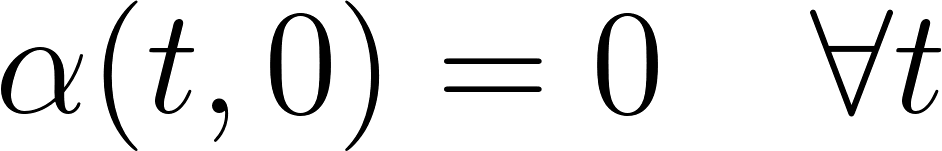


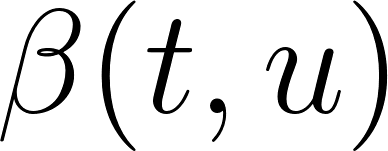
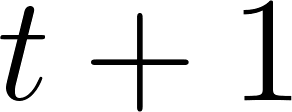
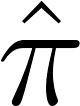
Figure 6.2: Forward-Backward Algorithm Lattice \citep{graves2006}

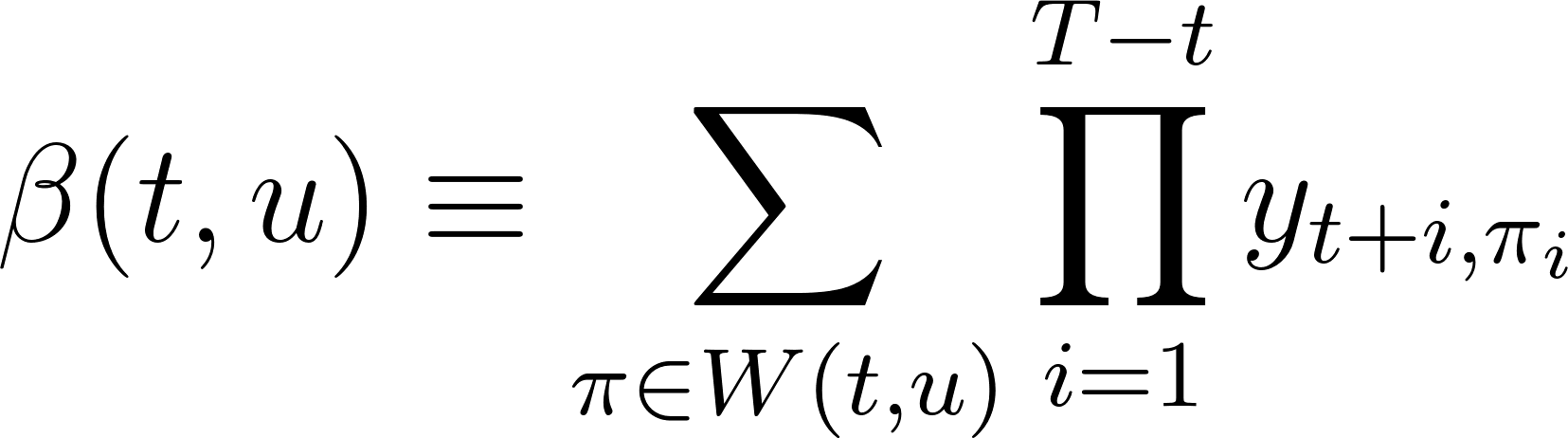
Figure \ref{fig\_6\_2\_lattice} expresses the recurrence relation for [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%0). While [](https://www.codecogs.com/eqnedit.php?latex=t%0) is expressed on the [](https://www.codecogs.com/eqnedit.php?latex=x%0) axis, [](https://www.codecogs.com/eqnedit.php?latex=u%0) is illustrated on the [](https://www.codecogs.com/eqnedit.php?latex=y%0) axis. The black circles of the diagram represent [](https://www.codecogs.com/eqnedit.php?latex=blank%0) elements of [](https://www.codecogs.com/eqnedit.php?latex=l'%0) while the white circles represent non-[](https://www.codecogs.com/eqnedit.php?latex=blank%0) elements of [](https://www.codecogs.com/eqnedit.php?latex=l'%0). The arrows represent computational dependencies derived from our recursion relation for [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0). So, for example, the value of [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(2%2C3)%0), corresponding to the [](https://www.codecogs.com/eqnedit.php?latex=blank%0) at [](https://www.codecogs.com/eqnedit.php?latex=t%3D2%0) and [](https://www.codecogs.com/eqnedit.php?latex=u%3D3%0), is derived from [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C2)%0). Similarly, the value of [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(2%2C2)%0), corresponding to the letter [](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 derived from [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C2)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(1%2C1)%0).

[](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%3D0%20%5Cquad%20%5Cforall%20u%20%3C%20U'-2(T-t)-1%0) - - - (9)

because these variables correspond to states for which there are not enough time-steps left to complete the sequence. We also impose the boundary condition

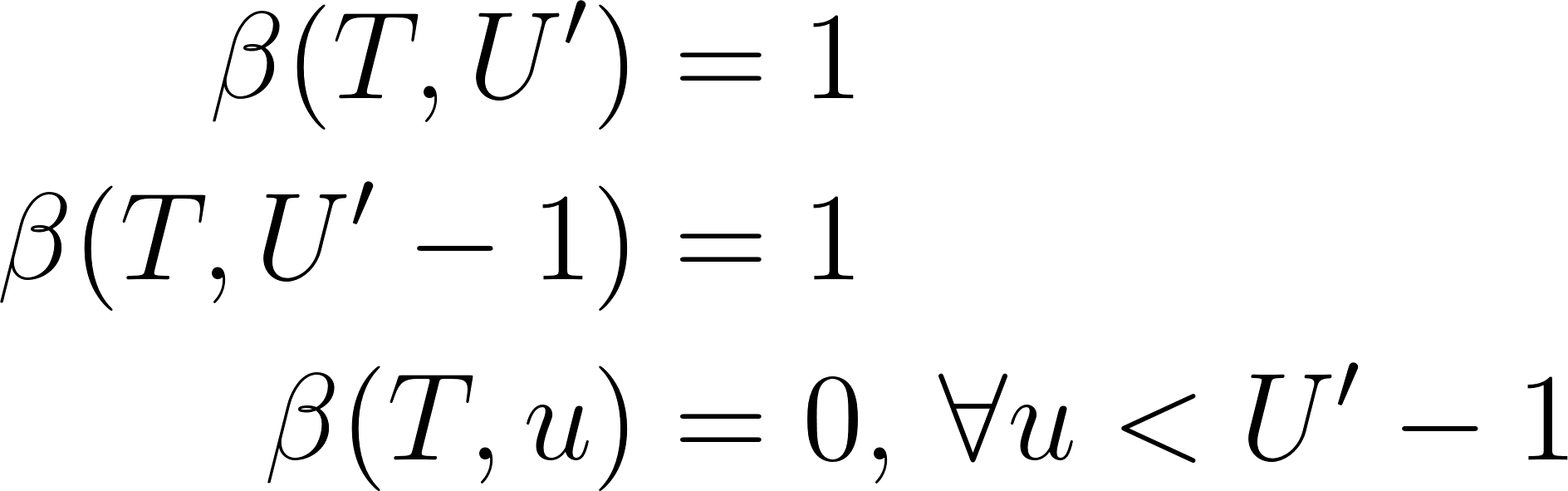
[](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%200)%20%3D%200%20%5Cquad%20%5Cforall%20t%0) - - - (9a)

The backward variables [](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2Cu)%0) are defined as the summed probabilities of all paths starting at [](https://www.codecogs.com/eqnedit.php?latex=t%20%2B%201%0) that "complete" [](https://www.codecogs.com/eqnedit.php?latex=l%0) when appended to any path [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7B%5Cpi%7D%0) contributing to [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2Cu)%0). Define [](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

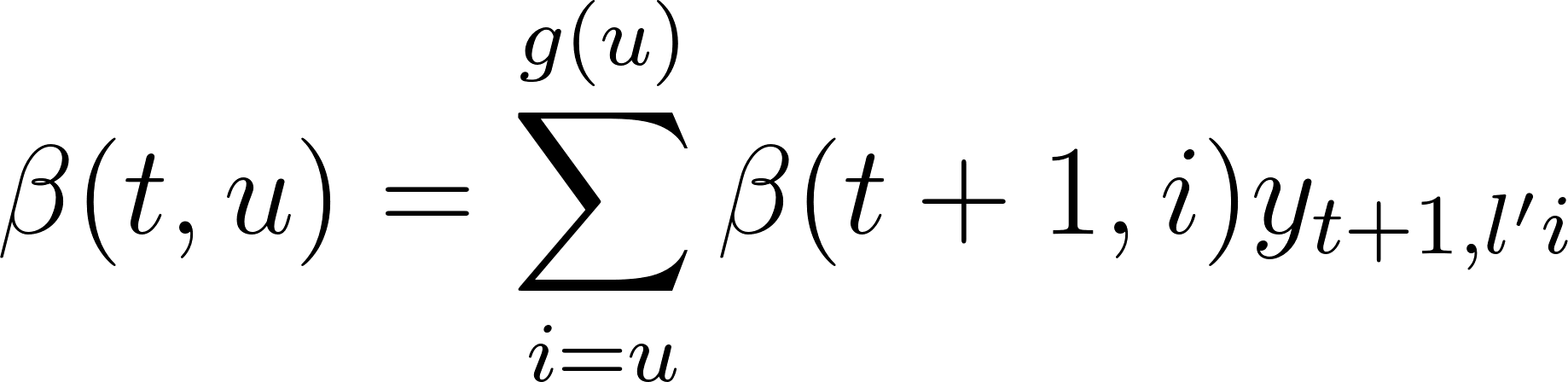
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2Cu)%20%5Cequiv%20%5Csum_%7B%5Cpi%20%5Cin%20W(t%2Cu)%7D%20%5Cprod_%7Bi%3D1%7D%5E%7BT%20-%20t%7D%20y_%7Bt%20%2B%20i%2C%5Cpi_i%7D%0) - - - (10)

The rules for initialisation of the backward variables are as follows

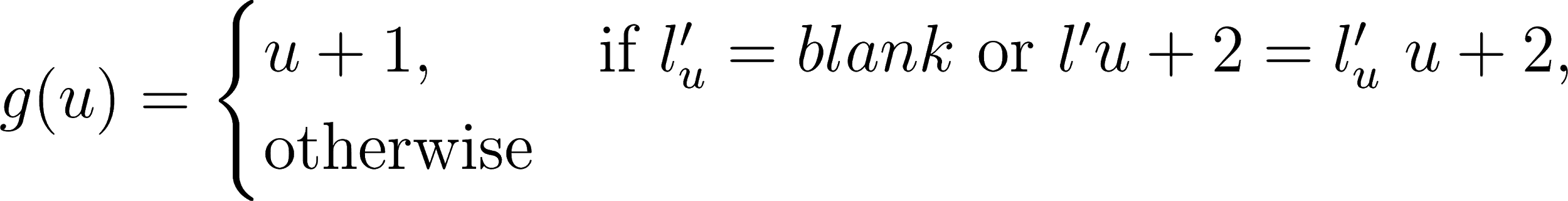
\begin{equation}

[](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 rules for recursion are as follows:

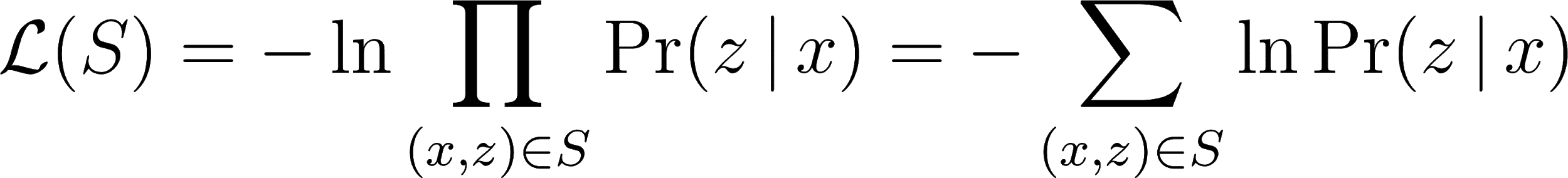
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2C%20u)%20%3D%20%5Csum_%7Bi%20%3D%20u%7D%5E%7Bg(u)%7D%20%5Cbeta(t%2B1%2C%20i)%20y_%7Bt%2B1%2C%20l'i%7D%0) - - - (12)

*where*

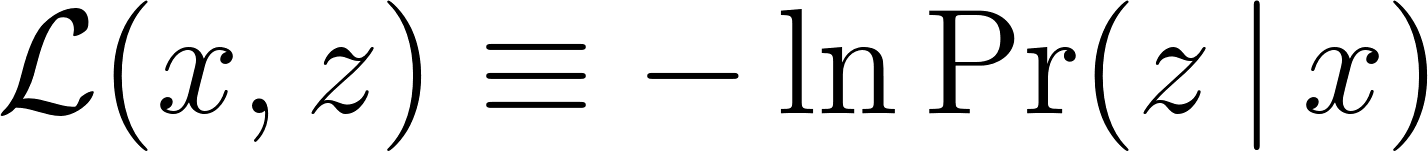
[**](https://www.codecogs.com/eqnedit.php?latex=g(u)%20%3D%20%5Cbegin%7Bcases%7D%20u%20%2B%201%2C%26%20%5Ctext%7Bif%20%7D%20l'_u%20%3D%20blank%20%5Ctext%7B%20or%20%7D%20l'%7Bu%2B2%7D%20%3D%20l'_%7Bu%7D%20%5C%20u%20%2B%202%2C%26%20%5Ctext%7Botherwise%7D%20%5Cend%7Bcases%7D%0) - - - (13)

## Loss Function

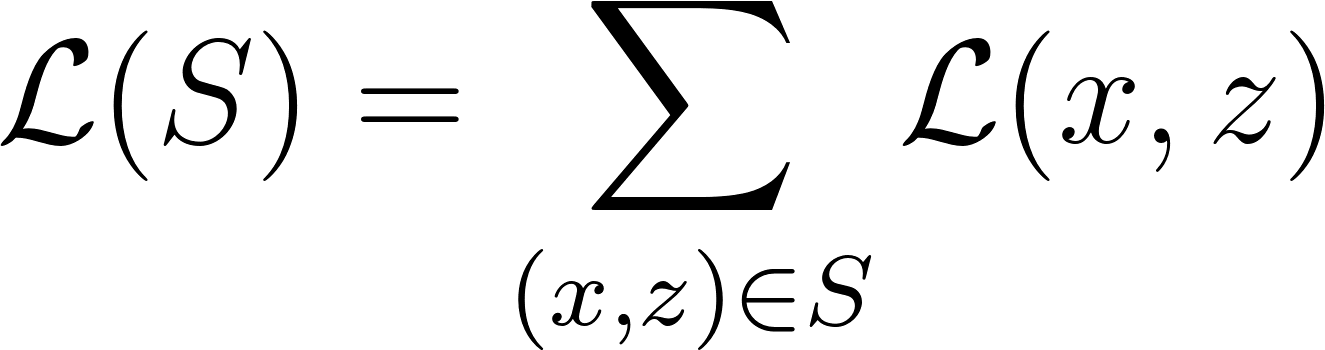
The cross entropy error is a loss function used to measure accuracy of probabilistic measures. It is calculated as the negative log probability of a likelihood measure. The CTC loss function [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%0) uses the cross entropy loss function of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

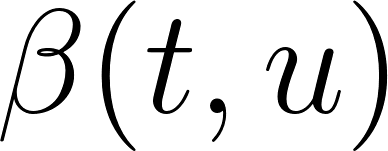
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%20%3D%20-%20%5Cln%20%5Cprod_%7B(x%2Cz)%20%5Cin%20S%7D%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20-%20%5Csum_%7B(x%2Cz)%20%5Cin%20S%7D%20%5Cln%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (14)

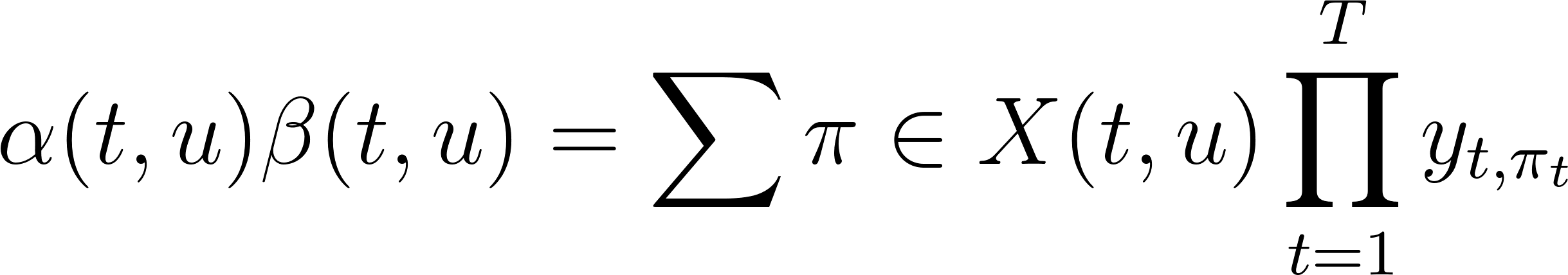
where [](https://www.codecogs.com/eqnedit.php?latex=z%0) is the output label and [](https://www.codecogs.com/eqnedit.php?latex=x%0) is the input sequence. Since [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%0) in equation \ref{eqn\_c3\_ctc11} is differentiable, this loss function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

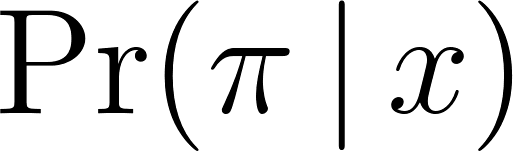
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(x%2Cz)%20%5Cequiv%20-%20%5Cln%20%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0) - - - (15)

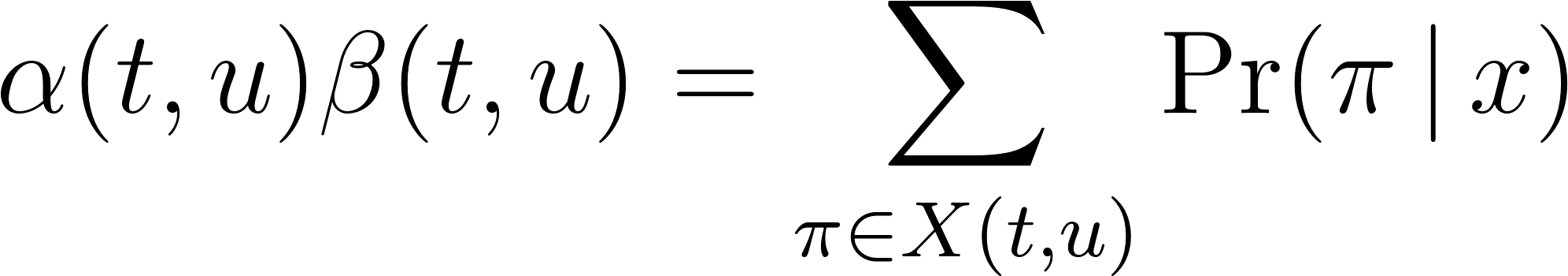
and therefore

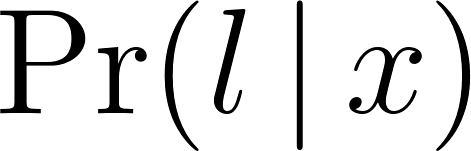
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(S)%20%3D%20%5Csum_%7B(x%2Cz)%20%5Cin%20S%7D%20%5Cmathcal%7BL%7D(x%2Cz)%0) - - - (16)

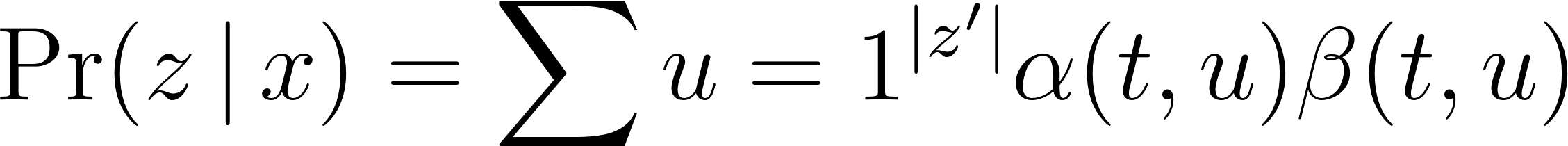
From the definition of the forward and backward variables [](https://www.codecogs.com/eqnedit.php?latex=%5Calpha(t%2C%20u)%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cbeta(t%2C%20u)%0), we also establish that [](https://www.codecogs.com/eqnedit.php?latex=X(t%2Cu)%20%5Cequiv%20%5C%7B%20%5Cpi%20%5Cin%20%7BA'%7D%5ET%20%3A%20%5Cmathcal%7BB%7D(%5Cpi)%20%3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

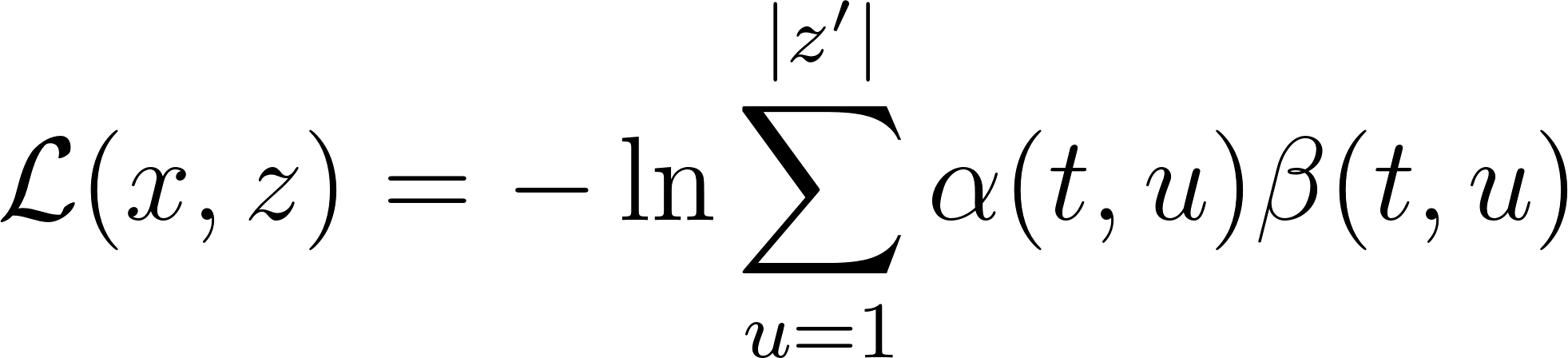
then substituting [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(%5Cpi%20%5C%2C%20%7C%20%5C%2C%20x)%0) from the expression in equation \ref{eqn\_c3\_ctc01}, we have

[](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)

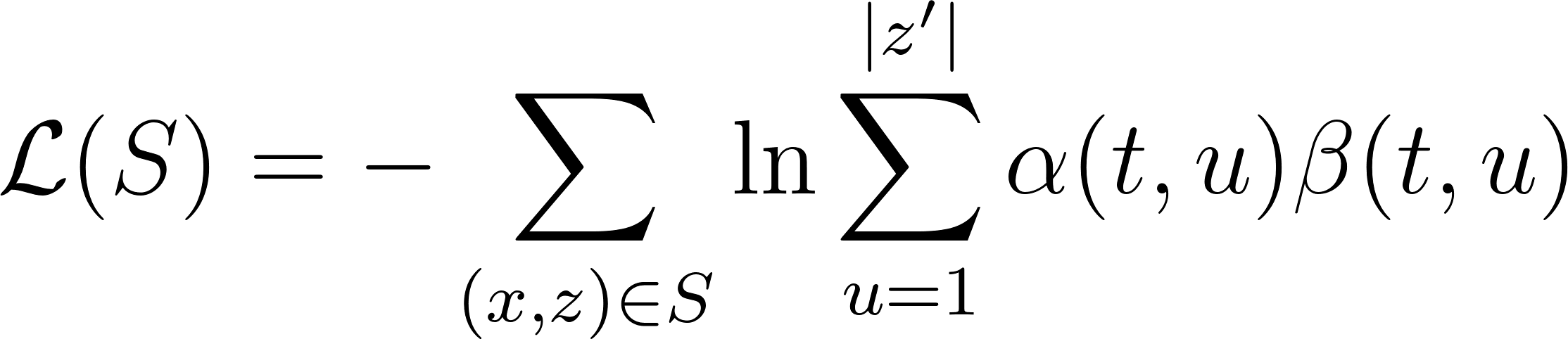
Also observe that [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(l%20%5C%2C%20%7C%20%5C%2C%20x)%0) is equivalent to the total probability [](https://www.codecogs.com/eqnedit.php?latex=%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%0). Paths going through [](https://www.codecogs.com/eqnedit.php?latex=z'u%0) *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*

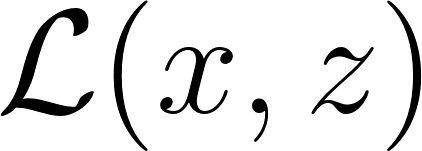
[**](https://www.codecogs.com/eqnedit.php?latex=%5CPr(z%20%5C%2C%20%7C%20%5C%2C%20x)%20%3D%20%5Csum%7Bu%20%3D%201%7D%5E%7B%7Cz'%7C%7D%20%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%0) *- - -* (19)

Thus a sample loss is determined by

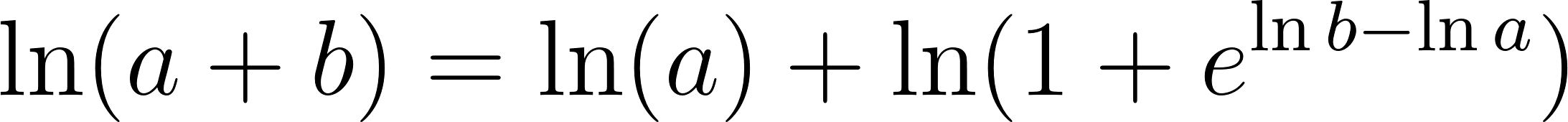
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathcal%7BL%7D(x%2C%20z)%20%3D%20-%20%5Cln%20%5Csum_%7Bu%20%3D%201%7D%5E%7B%7Cz'%7C%7D%20%5Calpha(t%2C%20u)%20%5Cbeta(t%2C%20u)%0) - - - (20)

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

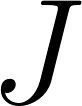
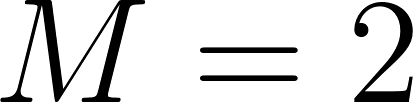
\begin{equation}

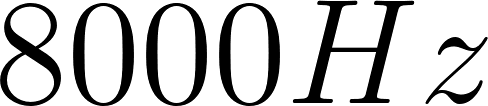
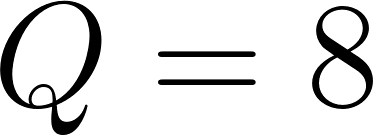
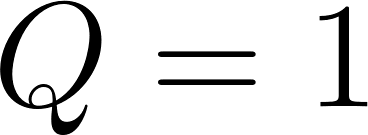
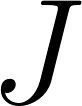
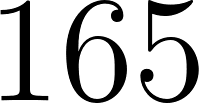
[](https://www.codecogs.com/eqnedit.php?latex=%5Cln(a%20%2B%20b)%20%3D%20%5Cln(a)%20%2B%20%5Cln(1%20%2B%20e%5E%7B%5Cln%20b%20-%20%5Cln%20a%7D)%0) - - - (22)

# The CTCC Speech Model

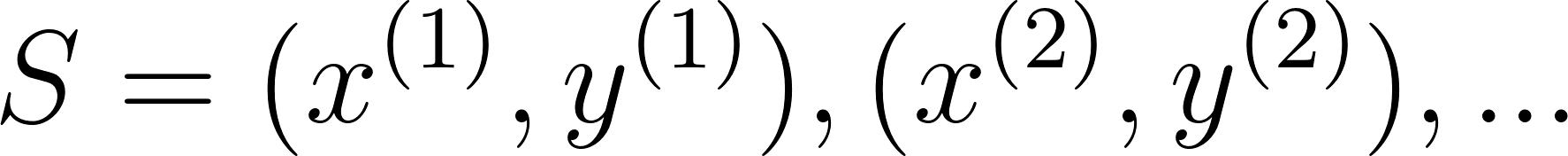
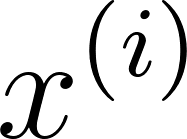
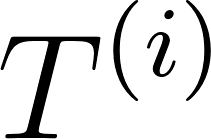
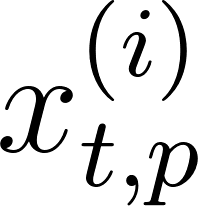
Earlier in chapter one, deep learning was defined as a type of representational learning whereby different levels of complexity are captured in internal layer-wise encapsulations. It has also been noted that layer-wise stacking of neural and neural network type architectures such as the Restricted Boltzmann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

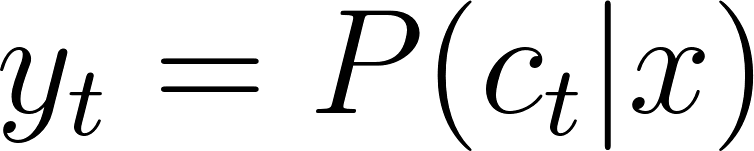
## Deep Scattering Features

The fast wavelet transformed is derived in Chapter 4 from a low pass filter and a high pass filter. The speech features used in this research using a deep scattering network 2 layers deep was created using the wavelet modulus operator comprising a low pass filter and a band pass filter. Hyper parameters of the system included the window period for each sampled sub section, T; The Q-band value for the band pass filter and the number of wavelets [](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).

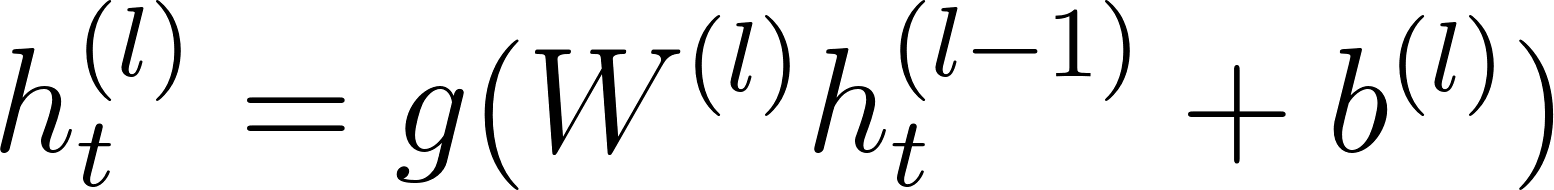
The matlab scatnet toolbox \citep{anden2014scatnet}, used to determine the scatter coefficient features for this research, provides optimal values for hyper parameters for audio signal processing into scatter features. In this regime the value for the hyper parameter [](https://www.codecogs.com/eqnedit.php?latex=T%0), the number of samples per window, = [](https://www.codecogs.com/eqnedit.php?latex=512%0) samples per window. This approximates 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 parameter, [](https://www.codecogs.com/eqnedit.php?latex=Q%3D8%0) and for the second scattering layer, the [](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 scatnet processing, eventually produce a feature-vector [](https://www.codecogs.com/eqnedit.php?latex=165%0) coefficients long. These feature vectors in turn are used as inputs to the bi-direction neural network model whose architecture is explained in the succeeding sections.

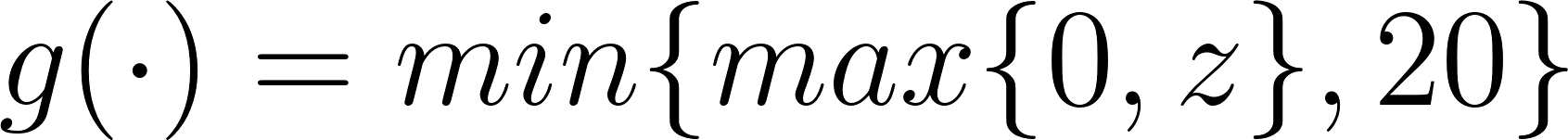
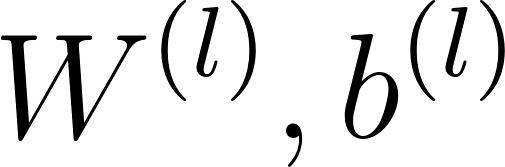
## CTCC-BiRNN Architecture

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 time-window consisting of [](https://www.codecogs.com/eqnedit.php?latex=T%5E%7B(i)%7D%0) samples. Each window passes through a scattering transform to yield an input of vector of p features; so [](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).

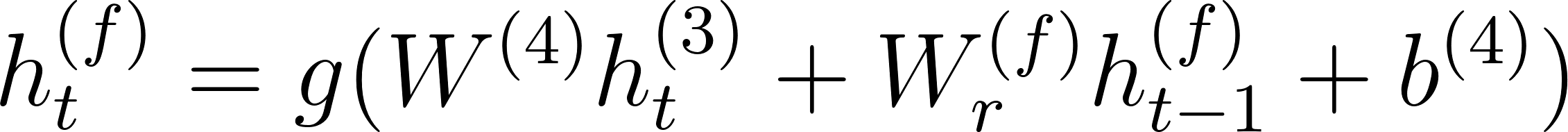
GPU training of the speech model architecture developed above was done using Mozilla deepspeech \cite{mozilla/deepspeech\_2019} CTC bi-directional RNN implementation along with the accompanying Mozilla Common voice dataset \cite{common voice by mozilla\_2019}. The Common Voice Dataset project consists 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).

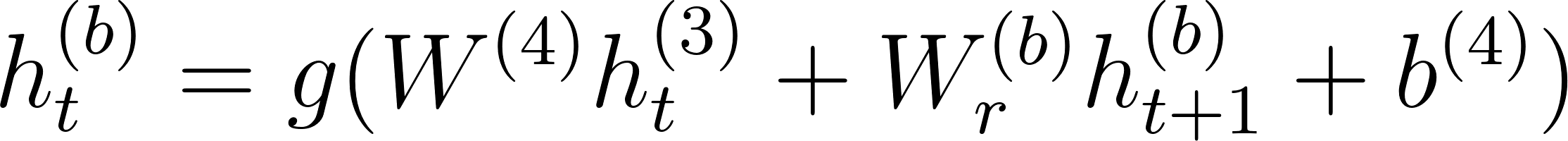
The actual architecture of our core Bi-RNN is similar to the deepspeech system described in \cite{hannun2014deep}. This structure consists of a 5 hidden layers and one output layer. The first three layers are regular DNNs followed by a bi-directional recurrent layer. As such, the output of the first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

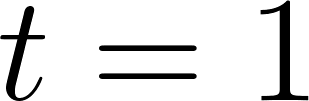
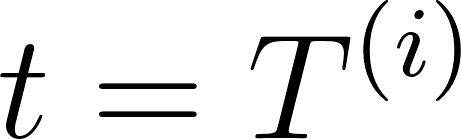
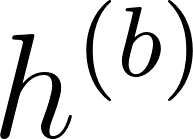
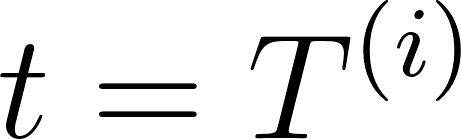
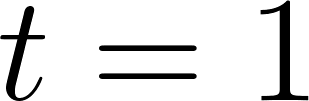
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(l)%7D_t%20%3D%20g(W%5E%7B(l)%7D%20h%5E%7B(l-1)%7D_t%20%2B%20b%5E%7B(l)%7D)%0)

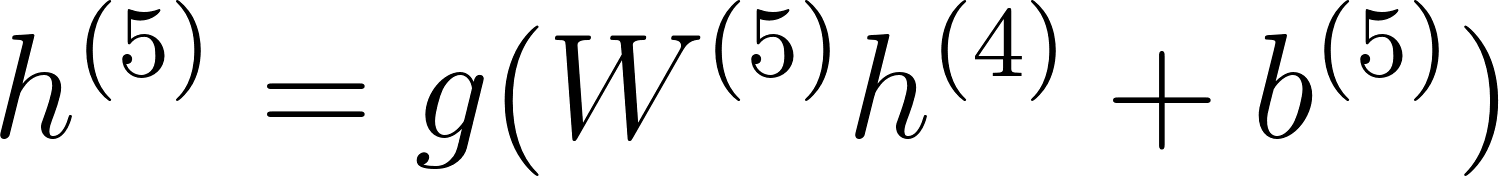
[](https://www.codecogs.com/eqnedit.php?latex=g(%5Ccdot)%20%3D%20min%5C%7Bmax%5C%7B0%2Cz%5C%7D%2C20%5C%7D%0) is the clipped rectified linear unit and [](https://www.codecogs.com/eqnedit.php?latex=W%5E%7B(l)%7D%2Cb%5E%7B(l)%7D%0) are weight matrix and bias parameters for layer [](https://www.codecogs.com/eqnedit.php?latex=l%0)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

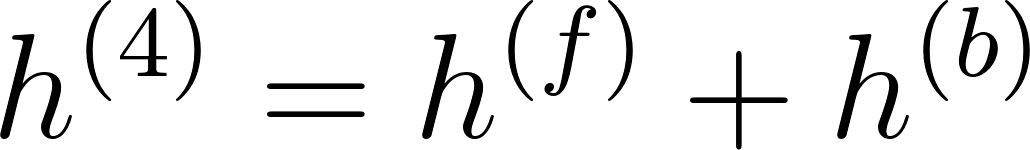
It was shown in chapter \ref{ch3RNN} the recurrent layer comprise a forward and backward RNNs whose equations are repeated here for reference

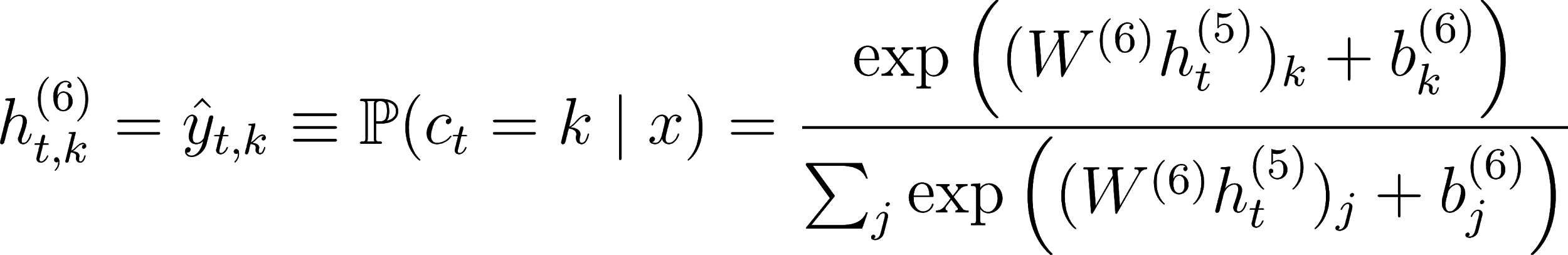
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(f)%7D_t%20%3D%20g(W%5E%7B(4)%7D%20h%5E%7B(3)%7D_t%20%2B%20W%5E%7B(f)%7D_r%20h%5E%7B(f)%7D_%7Bt-1%7D%20%2B%20b%5E%7B(4)%7D)%0)

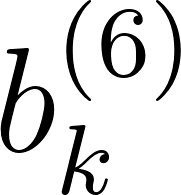
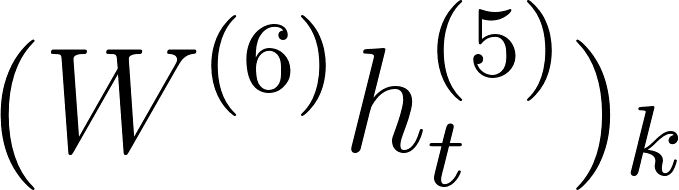
[](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(b)%7D_t%20%3D%20g(W%5E%7B(4)%7D%20h%5E%7B(3)%7D_t%20%2B%20W%5E%7B(b)%7D_r%20h%5E%7B(b)%7D_%7Bt%2B1%7D%20%2B%20b%5E%7B(4)%7D)%0)

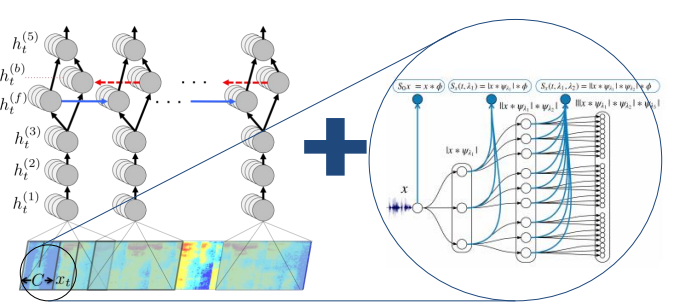
Consequently, [](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(f)%7D%0) is the sequential computation from [](https://www.codecogs.com/eqnedit.php?latex=t%3D1%0) to [](https://www.codecogs.com/eqnedit.php?latex=t%3DT%5E%7B(i)%7D%0) for the [](https://www.codecogs.com/eqnedit.php?latex=i%0)-th utterance 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)

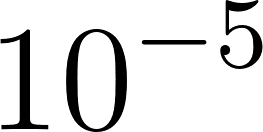
where [](https://www.codecogs.com/eqnedit.php?latex=h%5E%7B(4)%7D%20%3D%20h%5E%7B(f)%7D%20%2B%20h%5E%7B(b)%7D%0). The output of the Bi-RNN on layer 6 is a standard soft-max function that outputs a predicted character over probabilities for each time slice [](https://www.codecogs.com/eqnedit.php?latex=t%0)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

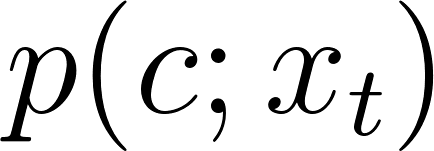
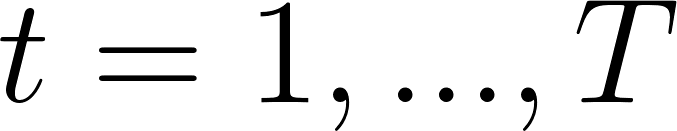
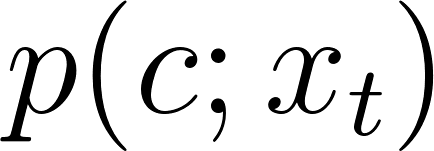
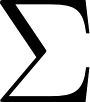
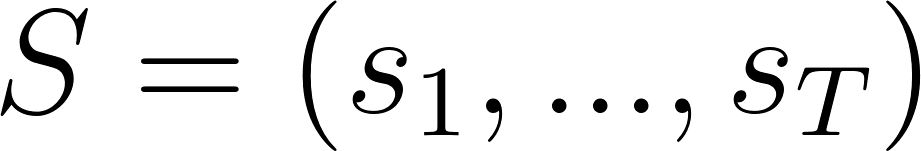
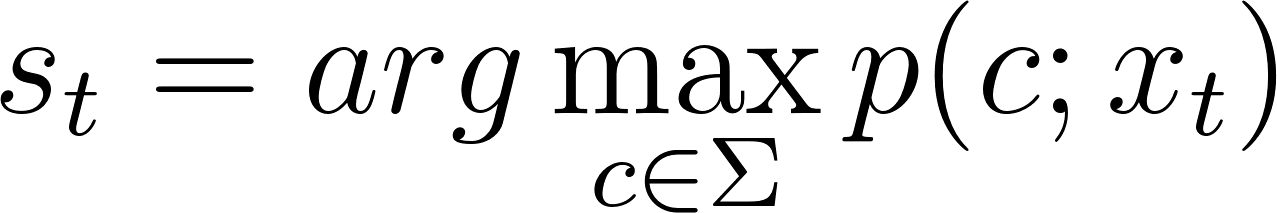


## Model Hyper parameters

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. 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.

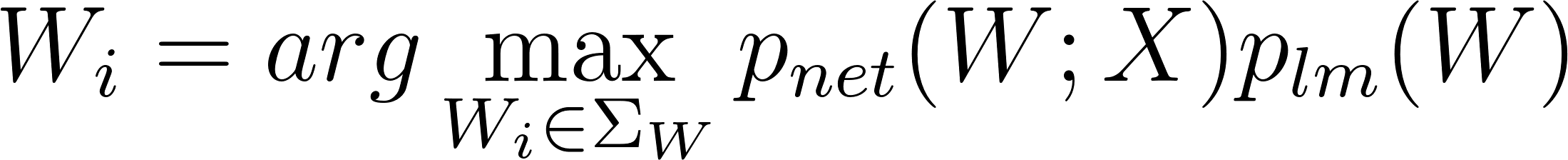
The network for network was trained for a total of five to fifty epochs over the training set for experiments conducted. The training time for Python GPU implementation is shown in Table \ref{ch06\_tab\_01}. For decoding with prefix search we use a beam size of 200 and cross-validated with a held-out set to find a good setting of the parameters α and β. Table 1 shows word error rates for various GPU configurations and audio dataset sizes.

## CTC decoding

Assuming an input of length T, the output of the neural network will be [](https://www.codecogs.com/eqnedit.php?latex=p(c%3Bx_t)%0) for [](https://www.codecogs.com/eqnedit.php?latex=t%3D1%2C...%2C%20T%0). Again [](https://www.codecogs.com/eqnedit.php?latex=p(c%3Bx_t)%0) is a distribution over possible characters in alphabet [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma%0), which includes the blank symbol, given audio input [](https://www.codecogs.com/eqnedit.php?latex=x_t%0). In order to recover a character string from the output of the neural network, as a first approximation, we take the argmax at each time step. Let [](https://www.codecogs.com/eqnedit.php?latex=S%3D(s_1%2C%20...%2C%20s_T)%0) be the character sequence where [](https://www.codecogs.com/eqnedit.php?latex=s_t%3Darg%5Cmax_%7Bc%5Cin%5CSigma%7Dp(c%3Bx_t)%0). The sequence S is mapped to a transcription by collapsing repeat characters and removing blanks. This gives a sequence which can be scored against reference transcription using both CER and WER.

The first approximation lacks the ability to include the constraint of either a lexicon or language model. We propose a generic algorithm which is capable of incorporating such constraints. Taking X to be acoustic input of time T, we seek a transcription W which maximises the probability.

\begin{equation}

[](https://www.codecogs.com/eqnedit.php?latex=W_i%3Darg%5Cmax_%7BW_i%20%5Cin%20%5CSigma_W%7D%20p_%7Bnet%7D(W%3BX)p_%7Blm%7D(W)%0)

\label{eqn\_c6\_brnn06}

\end{equation}

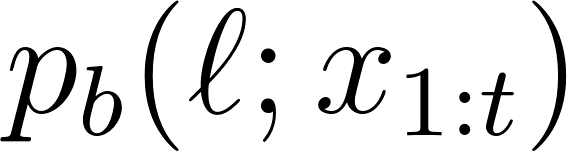
Here the overall probability of the transcription is modeled as the product of two factors: pnet given by the network and plm given by the language model prior. In practice the prior plm(W), when given by an n-gram language model, is too constraining and thus we down-weight it and include a word insertion penalty or bonus as

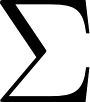
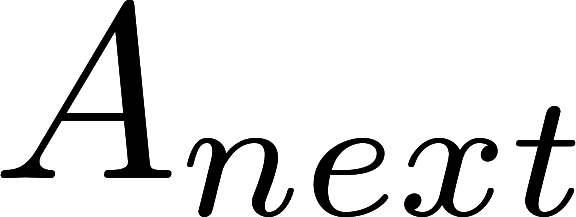
\begin{equation}

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

\label{eqn\_c6\_brnn07}

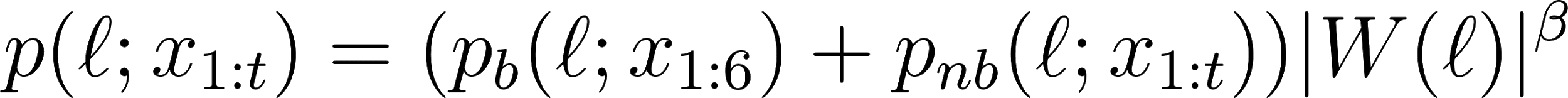
\end{equation}

Algorithm 1 attempts to find a word string W which maximizes equation 8. The algorithm maintains two separate probabilities for each prefix, [](https://www.codecogs.com/eqnedit.php?latex=p_b(%5Cell%3Bx_%7B1%3At%7D)%0) and [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bnb%7D(%5Cell%3Bx_%7B1%3At%7D)%0). Respectively, these are the probability of the prefix ending in blank or not ending in blank given the first [](https://www.codecogs.com/eqnedit.php?latex=t%0) time steps of the audio input [](https://www.codecogs.com/eqnedit.php?latex=X%0).

Algorithm 1 Prefix Beam Search: The algorithm initializes the previous set of prefixes [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) to the empty string. For each time step and every prefix [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0) currently in [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0), we propose adding a character from the alphabet [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma%0) to the prefix. If the character is a blank, we do not extend the prefix. If the character is a space, we incorporate the language model constraint. Otherwise we extend the prefix and incorporate the output of the network. All new active prefixes are added to [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnext%7D%0). We then set [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) to include only the [](https://www.codecogs.com/eqnedit.php?latex=k%0) most probable prefixes of [](http://www.texrendr.com/?eqn=A_%7Bnext%7D%0). The output is the 1 most probable transcript, although this can easily be extended to return an n-best list.

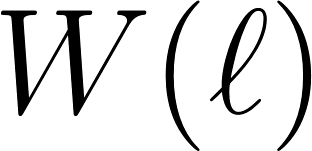
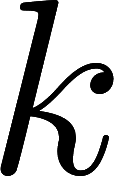
The sets [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnext%7D%0) maintain a list of active prefixes at the previous time step and a proposed prefixes at the next time step respectively. Note that the size of [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bprev%7D%0) is never larger than the beam width [](https://www.codecogs.com/eqnedit.php?latex=k%0). The overall probability of a prefix is the product of a word insertion term and the sum of the blank and non-black ending probabilities.

\begin{equation}

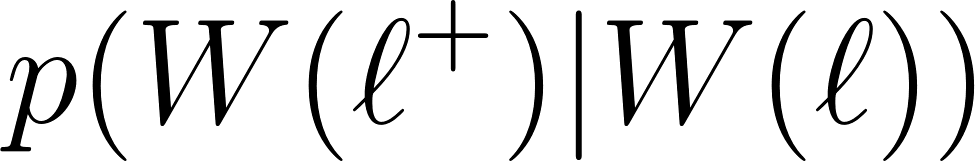
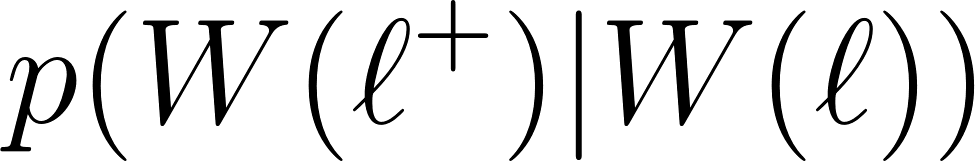
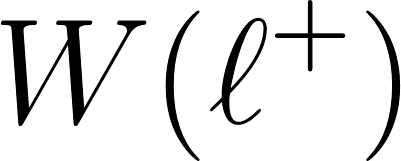
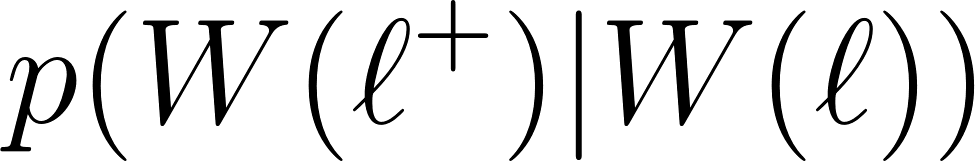
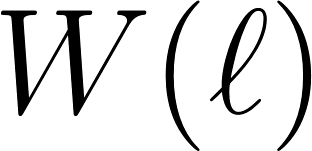
[](https://www.codecogs.com/eqnedit.php?latex=p(%5Cell%3Bx_%7B1%3At%7D)%3D(p_b(%5Cell%3Bx_%7B1%3A6%7D)%2Bp_%7Bnb%7D(%5Cell%3Bx_%7B1%3At%7D))%7CW(%5Cell)%7C%5E%5Cbeta%0)

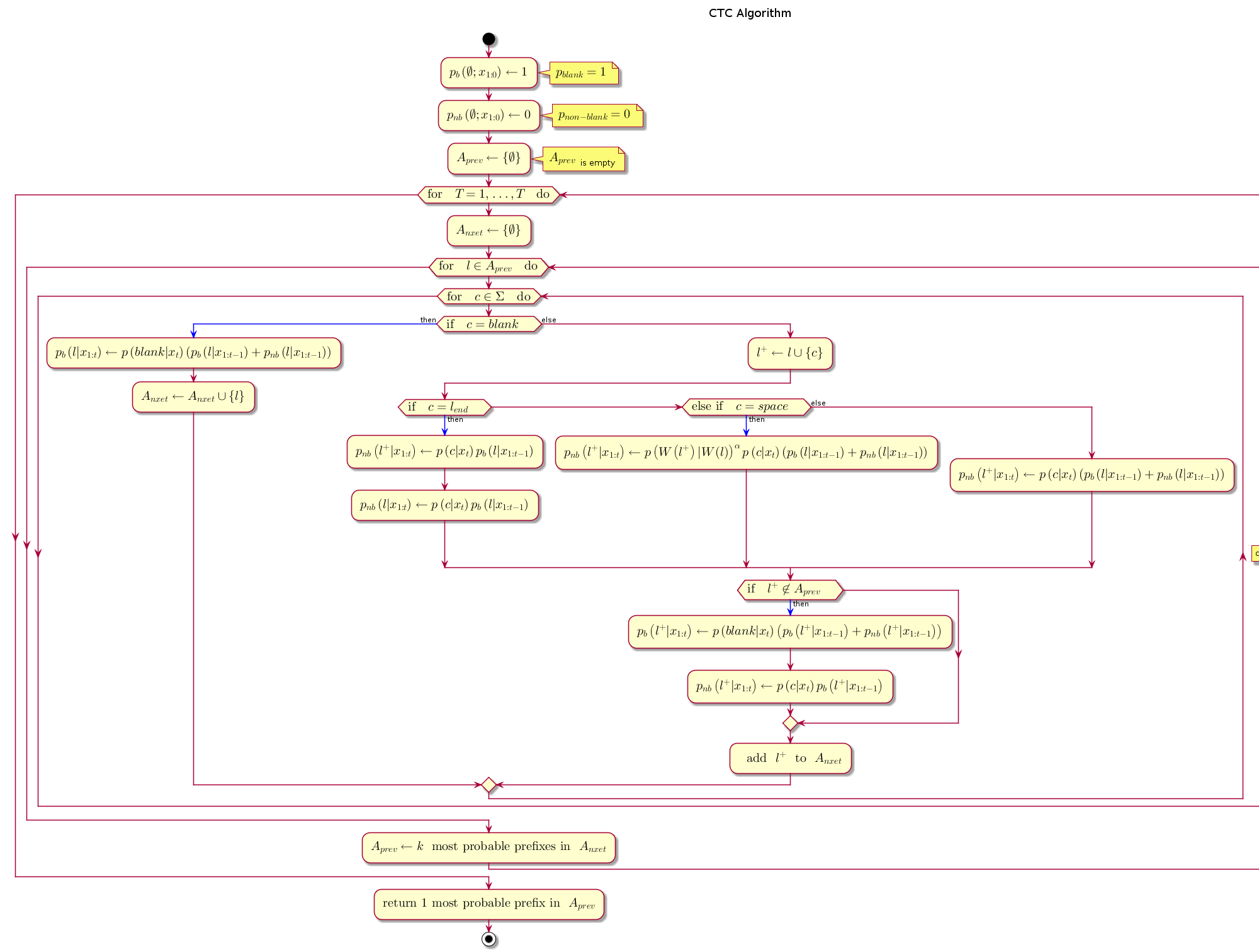
\label{eqn\_c6\_brnn08}

\end{equation}

Where [](https://www.codecogs.com/eqnedit.php?latex=W(%5Cell)%0) is a set of words in the sequence . When taking the [](http://www.texrendr.com/?eqn=k%0) most probable prefixes of [](https://www.codecogs.com/eqnedit.php?latex=A_%7Bnext%7D%0), we sort each prefix using the probability given in equation (\ref{eqn\_c6\_brnn08}).

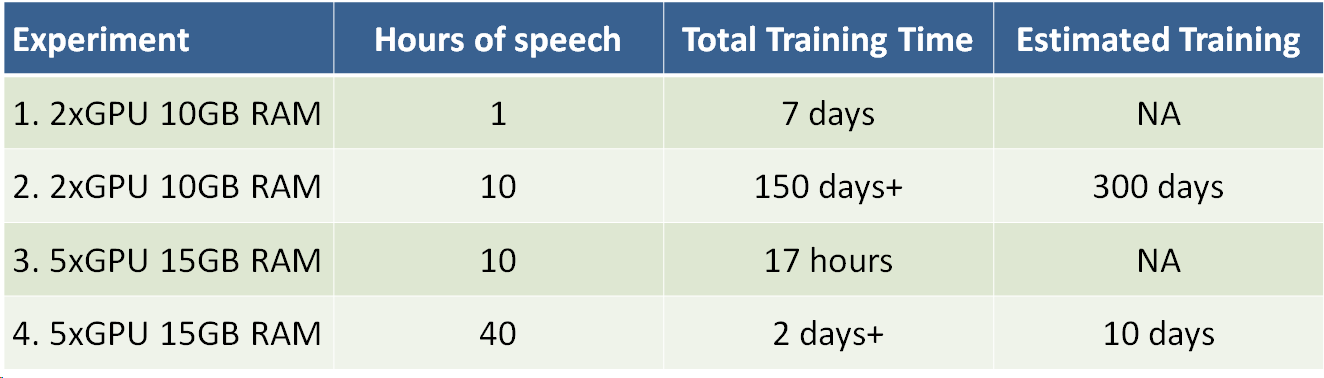
The variable [](https://www.codecogs.com/eqnedit.php?latex=%5Cell_%7Bend%7D%0) is the last character in the label sequence [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0). The function [](https://www.codecogs.com/eqnedit.php?latex=W(%5Ccdot)%0), which converts [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0) into a string of words, segments the sequence [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0) at each space character and truncates any characters trailing the last space.

We incorporate a lexicon or language model constraint by including the probability [](https://www.codecogs.com/eqnedit.php?latex=p(W(%5Cell%5E%2B)%7CW(%5Cell))%0) whenever the algorithm proposes appending a space character to [](https://www.codecogs.com/eqnedit.php?latex=%5Cell%0). By setting [](https://www.codecogs.com/eqnedit.php?latex=p(W(%5Cell%5E%2B)%7CW(%5Cell))%0) to 1 if the last word of [](https://www.codecogs.com/eqnedit.php?latex=W(%5Cell%5E%2B)%0) is in the lexicon and 0 otherwise, the probability acts a a constraint forcing all character strings to consists of words only in the lexicon. Furthermore, [](https://www.codecogs.com/eqnedit.php?latex=p(W(%5Cell%5E%2B)%7CW(%5Cell))%0) can represent an n-gram language model by considering only the last n-1 words in [](https://www.codecogs.com/eqnedit.php?latex=W(%5Cell)%0).

[](http://www.plantuml.com/plantuml/img/hLRBRkCW5Dtp5LUQJQmsQR9jcgfLFw6HPZ5C8spZ61M3nyPjAfT_7_0hs4cSpBHU-06NS-uz7C0FkIAPAXAEa6AAKtXQFy4ZtygCgJW1B12g8X2wlKk8Yk_JZJ_1D4dLHquLB7URSd4xrnxcD39ncNo7nTrD5Rb4GYeA6Tl62a4xklGv4QzwrKOXAaAmWNk2KlZw3CTyd4D8CMruvgTu7ZTbcj4txQ3YicF5UeIX7Tg40CkX6hVdU8-PqN9IXsD5TwgCPARnduA4i5ujhcUpsVKQwk-wFvGjdjUZJ6H68PUHIiWEc6XIs8CSr0Umhr3ig3fV8S1OlfW3DhoTNYTo1XY7WQ22mSzicn0OGm5mws-kmDR-9WFY5vna7qsxKtHziQXFpwAMVGML8nhcbX1KJ0LCxDthG02cltxulA2_f_VWjEuTQWpAFojVgfulqqb5OhgKDx5BfOsQBhHtrLZRRVESIHZG79k-hYqeKbpotXmsYl3Sp82zk_LqeFpboi7Z5gSCz94angNacz98hsinpsZPrtFGqL_q9hlZgWQLeaD1ZwPzWFbbH2lcK8nAPbUHF2K1lSnSto78JzlbVPg7vxrWmjEOT4Nzj_5Ek-SV9-eh6OYGHPVvq7hPBFxXtlGzweyLELpZNLolr5xhXLwzmFyzs0ibx7Xg1O648MYhNxC3WvBckzrS3lUK3j2yLNiySbvFdwUlBNmYSmLf9dtYc_F1X4LiHtExe-i3qY5k3P5HLMG25aUXMY3tK5wQ_nQP8lHWyCplpLy0)  
[PNG](http://www.plantuml.com/plantuml/img/hLRBRkCW5Dtp5LUQJQmsQR9jcgfLFw6HPZ5C8spZ61M3nyPjAfT_7_0hs4cSpBHU-06NS-uz7C0FkIAPAXAEa6AAKtXQFy4ZtygCgJW1B12g8X2wlKk8Yk_JZJ_1D4dLHquLB7URSd4xrnxcD39ncNo7nTrD5Rb4GYeA6Tl62a4xklGv4QzwrKOXAaAmWNk2KlZw3CTyd4D8CMruvgTu7ZTbcj4txQ3YicF5UeIX7Tg40CkX6hVdU8-PqN9IXsD5TwgCPARnduA4i5ujhcUpsVKQwk-wFvGjdjUZJ6H68PUHIiWEc6XIs8CSr0Umhr3ig3fV8S1OlfW3DhoTNYTo1XY7WQ22mSzicn0OGm5mws-kmDR-9WFY5vna7qsxKtHziQXFpwAMVGML8nhcbX1KJ0LCxDthG02cltxulA2_f_VWjEuTQWpAFojVgfulqqb5OhgKDx5BfOsQBhHtrLZRRVESIHZG79k-hYqeKbpotXmsYl3Sp82zk_LqeFpboi7Z5gSCz94angNacz98hsinpsZPrtFGqL_q9hlZgWQLeaD1ZwPzWFbbH2lcK8nAPbUHF2K1lSnSto78JzlbVPg7vxrWmjEOT4Nzj_5Ek-SV9-eh6OYGHPVvq7hPBFxXtlGzweyLELpZNLolr5xhXLwzmFyzs0ibx7Xg1O648MYhNxC3WvBckzrS3lUK3j2yLNiySbvFdwUlBNmYSmLf9dtYc_F1X4LiHtExe-i3qY5k3P5HLMG25aUXMY3tK5wQ_nQP8lHWyCplpLy0) | [SVG](http://www.plantuml.com/plantuml/svg/hLRBRkCW5Dtp5LUQJQmsQR9jcgfLFw6HPZ5C8spZ61M3nyPjAfT_7_0hs4cSpBHU-06NS-uz7C0FkIAPAXAEa6AAKtXQFy4ZtygCgJW1B12g8X2wlKk8Yk_JZJ_1D4dLHquLB7URSd4xrnxcD39ncNo7nTrD5Rb4GYeA6Tl62a4xklGv4QzwrKOXAaAmWNk2KlZw3CTyd4D8CMruvgTu7ZTbcj4txQ3YicF5UeIX7Tg40CkX6hVdU8-PqN9IXsD5TwgCPARnduA4i5ujhcUpsVKQwk-wFvGjdjUZJ6H68PUHIiWEc6XIs8CSr0Umhr3ig3fV8S1OlfW3DhoTNYTo1XY7WQ22mSzicn0OGm5mws-kmDR-9WFY5vna7qsxKtHziQXFpwAMVGML8nhcbX1KJ0LCxDthG02cltxulA2_f_VWjEuTQWpAFojVgfulqqb5OhgKDx5BfOsQBhHtrLZRRVESIHZG79k-hYqeKbpotXmsYl3Sp82zk_LqeFpboi7Z5gSCz94angNacz98hsinpsZPrtFGqL_q9hlZgWQLeaD1ZwPzWFbbH2lcK8nAPbUHF2K1lSnSto78JzlbVPg7vxrWmjEOT4Nzj_5Ek-SV9-eh6OYGHPVvq7hPBFxXtlGzweyLELpZNLolr5xhXLwzmFyzs0ibx7Xg1O648MYhNxC3WvBckzrS3lUK3j2yLNiySbvFdwUlBNmYSmLf9dtYc_F1X4LiHtExe-i3qY5k3P5HLMG25aUXMY3tK5wQ_nQP8lHWyCplpLy0) | [TXT](http://www.plantuml.com/plantuml/txt/hLRBRkCW5Dtp5LUQJQmsQR9jcgfLFw6HPZ5C8spZ61M3nyPjAfT_7_0hs4cSpBHU-06NS-uz7C0FkIAPAXAEa6AAKtXQFy4ZtygCgJW1B12g8X2wlKk8Yk_JZJ_1D4dLHquLB7URSd4xrnxcD39ncNo7nTrD5Rb4GYeA6Tl62a4xklGv4QzwrKOXAaAmWNk2KlZw3CTyd4D8CMruvgTu7ZTbcj4txQ3YicF5UeIX7Tg40CkX6hVdU8-PqN9IXsD5TwgCPARnduA4i5ujhcUpsVKQwk-wFvGjdjUZJ6H68PUHIiWEc6XIs8CSr0Umhr3ig3fV8S1OlfW3DhoTNYTo1XY7WQ22mSzicn0OGm5mws-kmDR-9WFY5vna7qsxKtHziQXFpwAMVGML8nhcbX1KJ0LCxDthG02cltxulA2_f_VWjEuTQWpAFojVgfulqqb5OhgKDx5BfOsQBhHtrLZRRVESIHZG79k-hYqeKbpotXmsYl3Sp82zk_LqeFpboi7Z5gSCz94angNacz98hsinpsZPrtFGqL_q9hlZgWQLeaD1ZwPzWFbbH2lcK8nAPbUHF2K1lSnSto78JzlbVPg7vxrWmjEOT4Nzj_5Ek-SV9-eh6OYGHPVvq7hPBFxXtlGzweyLELpZNLolr5xhXLwzmFyzs0ibx7Xg1O648MYhNxC3WvBckzrS3lUK3j2yLNiySbvFdwUlBNmYSmLf9dtYc_F1X4LiHtExe-i3qY5k3P5HLMG25aUXMY3tK5wQ_nQP8lHWyCplpLy0) | [Edit](http://www.planttext.com/planttext?text=hLRBRkCW5Dtp5LUQJQmsQR9jcgfLFw6HPZ5C8spZ61M3nyPjAfT_7_0hs4cSpBHU-06NS-uz7C0FkIAPAXAEa6AAKtXQFy4ZtygCgJW1B12g8X2wlKk8Yk_JZJ_1D4dLHquLB7URSd4xrnxcD39ncNo7nTrD5Rb4GYeA6Tl62a4xklGv4QzwrKOXAaAmWNk2KlZw3CTyd4D8CMruvgTu7ZTbcj4txQ3YicF5UeIX7Tg40CkX6hVdU8-PqN9IXsD5TwgCPARnduA4i5ujhcUpsVKQwk-wFvGjdjUZJ6H68PUHIiWEc6XIs8CSr0Umhr3ig3fV8S1OlfW3DhoTNYTo1XY7WQ22mSzicn0OGm5mws-kmDR-9WFY5vna7qsxKtHziQXFpwAMVGML8nhcbX1KJ0LCxDthG02cltxulA2_f_VWjEuTQWpAFojVgfulqqb5OhgKDx5BfOsQBhHtrLZRRVESIHZG79k-hYqeKbpotXmsYl3Sp82zk_LqeFpboi7Z5gSCz94angNacz98hsinpsZPrtFGqL_q9hlZgWQLeaD1ZwPzWFbbH2lcK8nAPbUHF2K1lSnSto78JzlbVPg7vxrWmjEOT4Nzj_5Ek-SV9-eh6OYGHPVvq7hPBFxXtlGzweyLELpZNLolr5xhXLwzmFyzs0ibx7Xg1O648MYhNxC3WvBckzrS3lUK3j2yLNiySbvFdwUlBNmYSmLf9dtYc_F1X4LiHtExe-i3qY5k3P5HLMG25aUXMY3tK5wQ_nQP8lHWyCplpLy0)

# Results

A total of four experiments were carried out on two different GPU configurations. A set of experiments was performed a GPU configuration consisting of 2 GPUs having a total of 10 gigabytes of memory. The second set of experiments was carried out on a GPU configuration comprising 5 GPUs having a total of 15 gigabytes of memory. For each configuration two experiments were carried out on a small subset of the dataset then on a larger subset of the common voice dataset being used. The various GPU configurations along with the training times is shown in Table 1.



The output of the training produced mostly gibberish when trained in both configurations using only just one hour of training data. Training loss reduced significantly once the data was increased to ten hours of training. However word error rates (WER) only showed improvement on the 40 hours dataset.

# Discussion

The results showed that the training of the model was heading towards a very slow convergence as indicated by the slow decrements in training loss. However, we perceive that given the complete dataset to train the model will not only converge but show improvements in word error rates.

Though this work is an on-going research, the authors would love to collaborate with other speech research groups and speech technology researchers in order to gain access to different hardware in order to speed up the training rates. As could be seen from the training times the research was limited by adequate training hardware.

# Conclusion

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

We show in this work that Deep Scattering features derived from wavelet filter operations on audio data produce viable candidates for end-to-end training of Automatic speech recognition models.

References

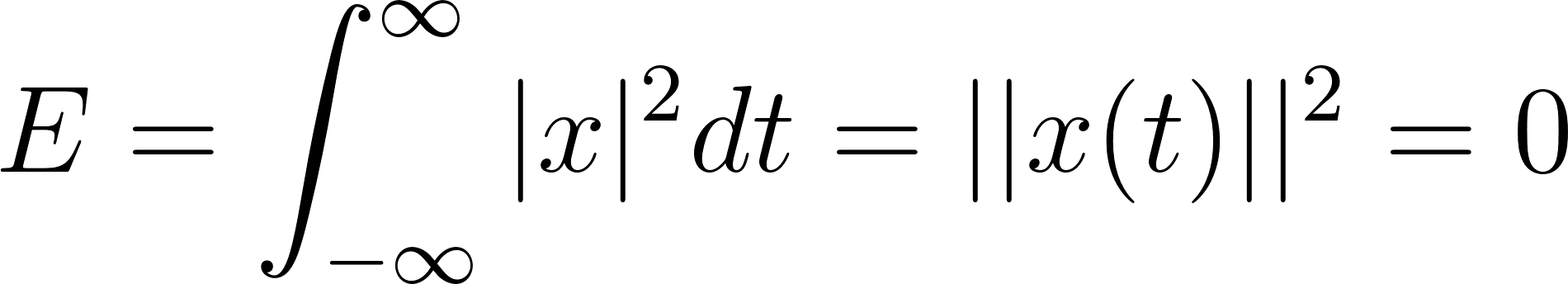
Lyons, J. (2012). Mel Frequency Cepstral Coefficient (MFCC) tutorial. Retrieved from <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>

# Appendices

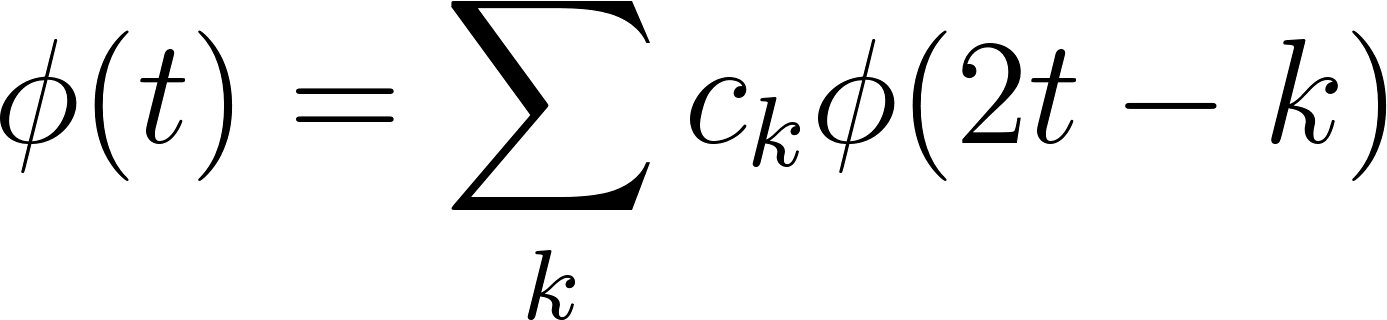
## Appendix 1 - Haar Wavelet

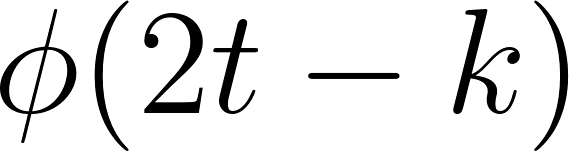
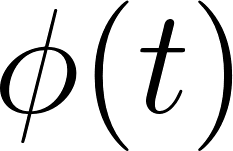
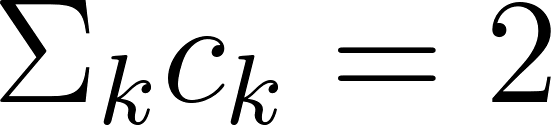
A fundamental purpose of analysing functions such as the Fourier and wavelet functions are the reconstruction of signals from it’s decomposition. Certain criteria or properties are therefore required for analysis functions.

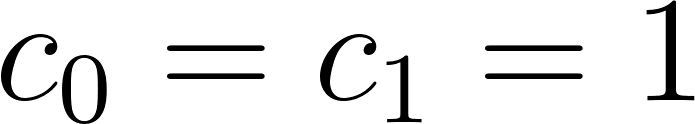
In Chapter \ref{c4\_fourier}, the orthogonal properties of the Fourier transform equations was introduced. In the case of wavelets the following properties ensue. In addition to orthogonal properties, wavelets are required to perform localised analysis of a function. Hence, unlike their Fourier counterparts, they need to be bounded in time. It is also seen that when the energy contained within the wavelet bases sum to zero (sometimes normalised to 1)i.e.

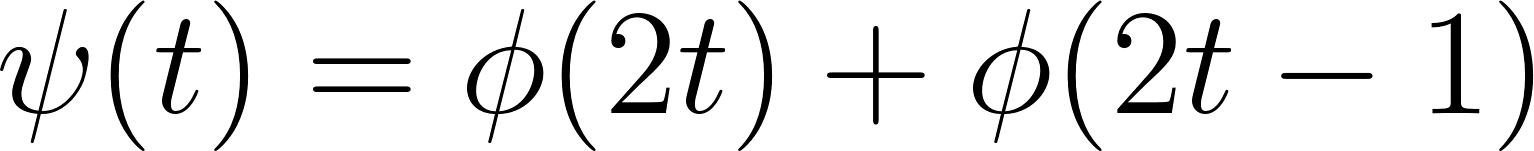
[](https://www.codecogs.com/eqnedit.php?latex=E%3D%5Cint_%7B-%5Cinfty%7D%5E%5Cinfty%20%7Cx%7C%5E2dt%3D%7C%7Cx(t)%7C%7C%5E2%3D0%0) - - - (1)

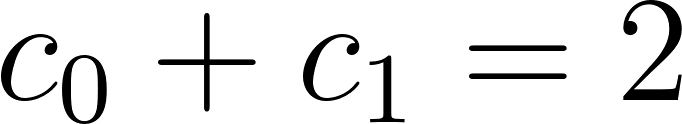
Then, such wavelet bases are orthonormal and the fundamental or scaling equation forms a recurrence relation which is a solution to the dilation equation as follows

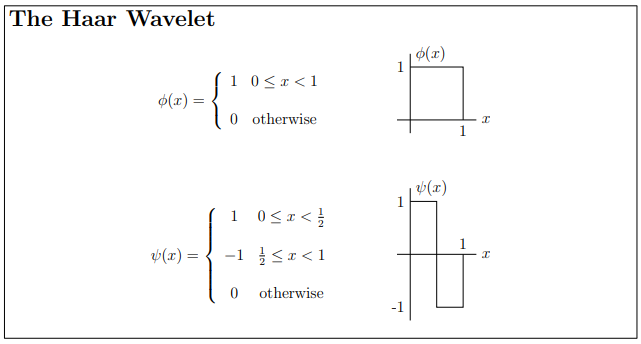
[](https://www.codecogs.com/eqnedit.php?latex=%5Cphi(t)%3D%5Csum_kc_k%5Cphi(2t-k)%0) - - - (2)

Where [](https://www.codecogs.com/eqnedit.php?latex=%5Cphi(2t-k)%0) is a contracted version of [](https://www.codecogs.com/eqnedit.php?latex=%5Cphi(t)%0) shifted along the time axis by an integer step k and factored by an associated coefficient [](https://www.codecogs.com/eqnedit.php?latex=c_k%0). At the same time it is also observed that it is possible to setup this recurrence relation to becoming dyadic such that the sum of coefficients, [](https://www.codecogs.com/eqnedit.php?latex=c_k%0) equals 2, i.e. [](https://www.codecogs.com/eqnedit.php?latex=%5CSigma_kc_k%3D2%0). Haar, wavelets constitute the simplest of this family of wavelets.

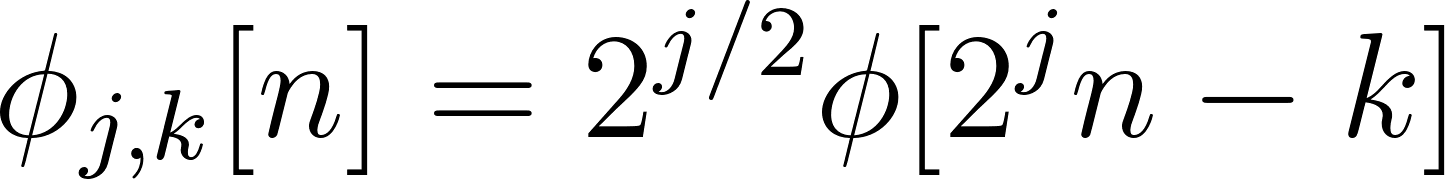
The mother wavelet of the Haar wavelet has only two coefficients [](https://www.codecogs.com/eqnedit.php?latex=c_0%3Dc_1%3D1%0) and is given by

[](https://www.codecogs.com/eqnedit.php?latex=%5Cpsi(t)%3D%5Cphi(2t)%2B%5Cphi(2t-1)%0) - - - (3)

Note that [](https://www.codecogs.com/eqnedit.php?latex=c_0%2Bc_1%3D2%0). The solution to this recurrence equation and the resulting plot is given in Figure \ref{haar}.



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

[](https://www.codecogs.com/eqnedit.php?latex=%5Cphi_%7Bj%2Ck%7D%5Bn%5D%3D2%5E%7Bj%2F2%7D%5Cphi%5B2%5Ejn-k%5D%0)

The parameter j, controls the resolution of the signal reconstruction and the following wavelets and function representation are given in Figure \ref{multires}

## Appendix 2 - Scatter-transform Wavelet Filters

### Gabor Wavelet filter

The 1d-gabor wavelet is defined spatially by



The fourier-transform is



It’s value bounded at 0 is, , so that we have

 - - - (1)

The wavelets are therefore computed as



Where a is the scale factor. If we call τ the value of the two gabor where there plots intersect, we have, as we can see on figure 1





 - - - (2)

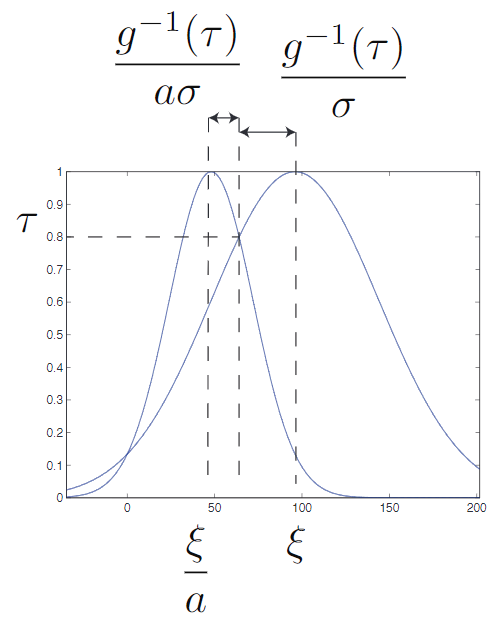


Figure 1: fourier transform of adjacent scale gabor wavelet.τhas been set to 0.8 here

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



Equation 1 and 2 show that the choice ofσis a trade off between two antagonist requirement on the wavelet :

1. a zero - mean :σshould be large.

2. a good littlewood paley sum for the filter bank and for thatτshould be large soσshoud be small

If we use 1 and 2 we can see that



so we can see that these tow requirement are more and more compatible as we take more and more band per octave.

In the implementation, the only parameter about the wavelet that the user can set is  ’as’ stands for adjacent scales. The implementation of parameters is the following:

1. The value of ξ is set to 3π/4.

2. The user choses values for τ, a and J

3. The value of σ is computed with



There is another parameter called τlc. ’lc’ stands for low-coarse. It controls the value at crossing between the low pass filters φJ (a gaussian) and the coarsest scale wavelet, ψJ−1 and this parameters determines the value of the bandwidth of the low pass filter in the same spirit