#+Title: Deep Neural Network Techniques in Low Resource Speech Recognition

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 gives evidence that this gap is closing, there is still 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 ASR. The level of ASR accuracy that have been so far achieved have been 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. In addition, due to the fact that collection of sufficient amounts of linguistic resources required to create accurate models for ASR are particularly laborious and time consuming to collect sometimes extending to decades, it is therefore wise to consider alternative approaches towards developing ASR systems for languages lacking the resources required to build such systems using existing mechanisms.

ASR as a machine learning problem \label{ASRMLP}

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. The process of automatic speech recognition attempts to reverse engineer the 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 constitutes a contribution to enhances made in the machine learning field. This also is an attribution to the fact that speech recogntion is a sequence pattern recogntion problem. Therefore techniques within speech recognition could be applied generally to sequence pattern recognition problems.

The two main approaches to machine learning problems historically involve two methods rooted in statistical science. These approaches are the generative and discriminitative models. From a computing science perspective, the generative approach is a bruteforce approach while the discriminative model uses a rather heuristic approach to machine learning. This chapter derives the basic definitions of these two approaches in order to establish the motivation for the proposed models used in this research for low resource speech recognition as well as introduces the Wakirike language as the motivating language case study.

Generative versus Discriminative speech model disambiguation

In the next chapter, the Hidden Markov Model (HMM) is examined as a powerful and major driver behind generative modeling 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 the features seen. The discriminative model, on the other hand, has a heuristic approach to make classification. Rather than using features of the data directly, the discriminative method attempts to characterise the data into features. It is possible to conclude 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 discover the fundamental structures.

Ultimately the 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 wave form or equivalent speech sequence.

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

\label{eqn1_1}

where $d_y(\mathbf{x};\lambda)$ is the decision function of y for data labels $\mathbf{x}$. This joint probability expression given as $p(\mathbf{x}|y;\lambda)$ can also be expressed as the conditional probability product in equation (\ref{eqn_1_1}). In this above equation, λ predefines the nature of the distribution \cite{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:
\begin{equation}

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

\label{eqn1_2}

\end{equation}

It is clearly seen that the discriminative paradigm is a much simpler and a more straight forward paradigm and indeed is the chosen paradigm for this study. However, what the discriminative model gains in discriminative simplicity it loses in model parameter estimation(λ) complexity in equation (\ref{1_1}) and (\ref{1_2}). As this research investigates, the although the generative process is able to generate arbitrary outputs from learned inputs, its major draw back 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, not directly dependent on the training data, gains over the generative approach being able to better compensate the latent variables. In the case of speech data information is lost in training data due to the physiologic transformations mentioned in section \ref{ASRMLP}. This rationale is reinforced from 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 are typically implemented using deep neural networks, this work applies deep recurrent neural networks as an end-to-end discriminative classifier, to speech recognition. This is a so called “an end-to-end model” because it adopts the top-to-bottom machine learning approach. 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 amount of resources required for speech recognition. From a low resource language perspective this is an attractive feature 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

A second challenge observed in complex machine learning models for both generative as well as discriminative learning models is the data intensive nature required for robust classification models. \cite{saon2015ibm} recommends around 2000 hours of transcribed speech data for a robust speech recognition system. As we cover in the next chapter, for new languages for 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 doesn’t affect the 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 was 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 modeling 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. Wakirike migrants settled at the Okrika mainland between AD860 at the earliest AD1515. Earliest settlers migrated from Central and Western Niger Delta. When the second set of settlers met the first set of 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 much less than that. 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 was about 9,000 words.

Due to limitations in transcribed speech for the Wakirike language, English was substituted and used for the final speech model. The English language was used as a control variable to measure accuracy of speech recognition for differing spans of speech data being validated against on algorithms developed in this research.

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 beginning from multi-layer perceptrons and probabilistic sequence models. Specialised recurrent neural networks, long short-term memory (LSTM) networks and the Gated Recurrent Units (GRU) used to develop the language model for the Wakirike language are detailed.

Chapter Four explains the wavelet theorem as well as the deep scattering spectrum. The chapter develops the theory from Fourier transform and details the the significance of using the scattering transform as a feature selection mechanism for low resource recognition.

Chapters five and six is a description of the models developed by this thesis and details the experiment setup along with the results obtained. Chapters seven is a discussion of the result and chapter 8 are the recommendations for further study.

Low Resource Speech Models, End-to-end models and the

scattering network

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 is a Gated-Recurrent-Unit Recurrent Neural network (GRU-RNN) was used to develop a character-based language model, while the second recurrent neural network is a Bi-Directional Recurrent neural Network (BiRNN) used as 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 biRNN 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 $\mathbf{O}$ such that
\begin{equation}$\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T.$
\label{eqn_1_1_sr_inputs}\end{equation}

At each discrete time t, we have an observation $\mathbf{o}_t$, which is, in itself is a vector in $\mathbb{R}^D$. From the conditional probability, it can be formulated that certain word sequences from a finite dictionary are most probable given a sequence of observations. That is:
\begin{equation}$arg\ \max_t\{P(w_i|\mathbf{O})\}$
\label{eqn_2_2_srgen}
\end{equation}

As we describe in the next section on speech recognition challenges, there is no straightforward analysis of of $P(w_i|\mathbf{O})$. The divide and conquer strategy therefore employed uses Bayes formulation to simplify the problem. Accordingly, the argument that maximises the probability of an audio sequence given a particular word multiplied by the probability of that word is equivalent to the original posterior probability required to solve the isolated word recognition problem. This is summarised by the following equation
\begin{equation}$P(w_i|\mathbf{O}) = \frac{P(\mathbf{O}|w_i)P(w_i)}{P(\mathbf{O})}$
\label{eqn_2_3_bayes_sr}
\end{equation}

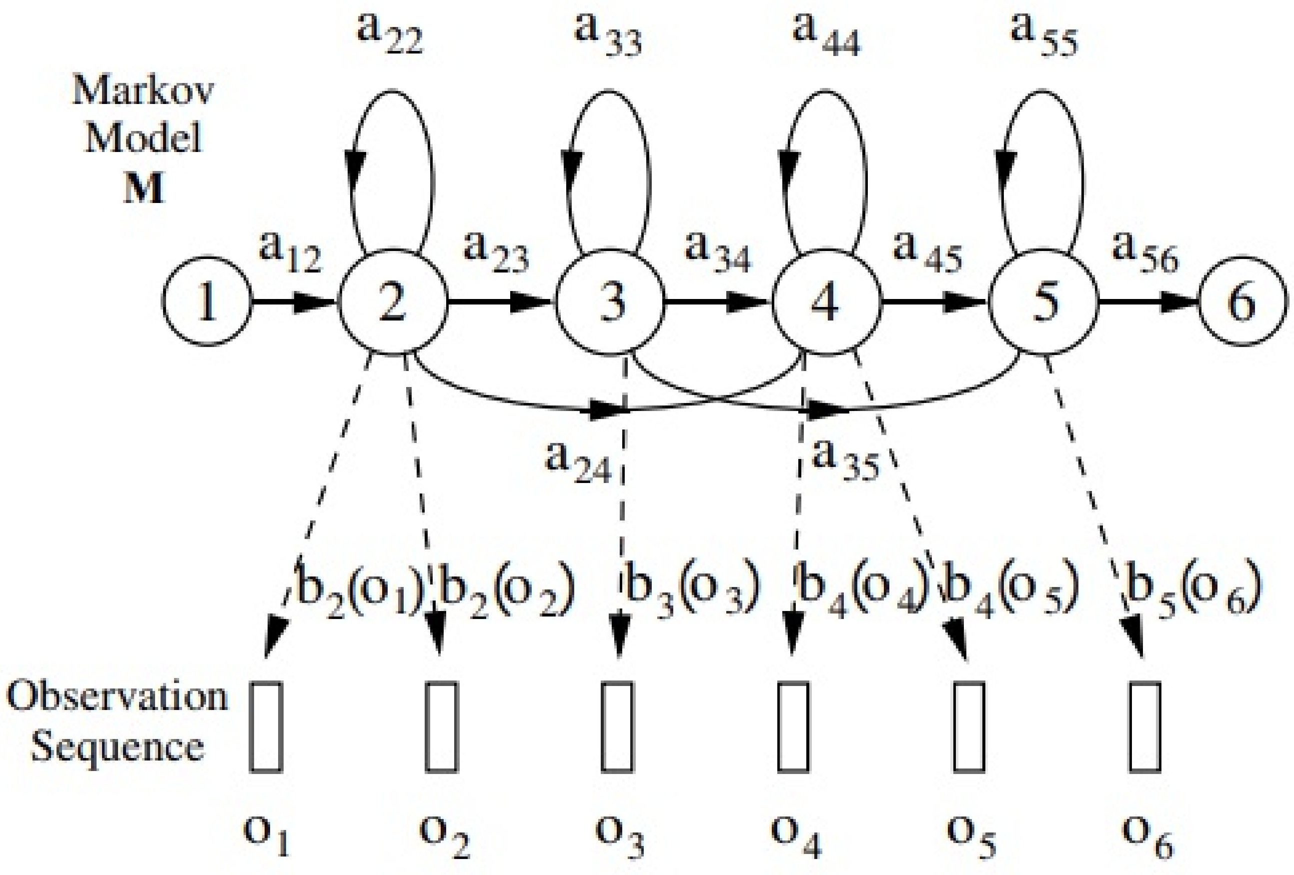
That is, according to Bayes' rule, the posterior probability is obtained by multiplying a certain likelihood probability by a prior probability. The likelihood in this case, $P(\mathbf{O}|w_i)$, is obtained from a Hidden Markov

Model (HMM) parametric model such that rather than estimating the observation densities in the likelihood probability, these are obtained by estimating the parameters of the HMM model. The HMM model explained in the next section gives a statistical representation of the latent variables of speech.

The second parameter in the speech model interpreted from Bayes' formula is prior is the probability a given word. This aspect of the model is the language model which we review 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. 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. For each state transition emits an output observation that represents the current internal state of the system.



```
\begin{figure}
\centering
% Requires \usepackage{graphicx}
\includegraphics[width=7cm]{thesis/images/hmm}\
\caption{HMM Generative
Model}\cite{young2002htk}}\label{fig_2_1_hmm}
\end{figure}
```

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 a_{ij} this is the probability to move from state i to state j. The second probability b_j is the probability that an output probability when a state emits an observation.

Given that X represents the sequence of states transitioned by a process a HMM the joint probability of X and the output probabilities given the HMM is given as the following representation:

\begin{equation}$P(\mathbf{O}|M) \ = \ \sum_X \ a_{x(0)x(1)} \ \prod_{t=1}^{T} \ b_{x(t)}(\mathbf{o}_t)a_{x(t)x(t+1)}$

\label{eqn_2_4_hmm}

\end{equation}

Generally speaking, the HMM formulation presents 3 distinct challenges. The first is that likelihood of a sequence of observations given in equation \ref{eqn_2_4_hmm} above. The next two which we describe 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

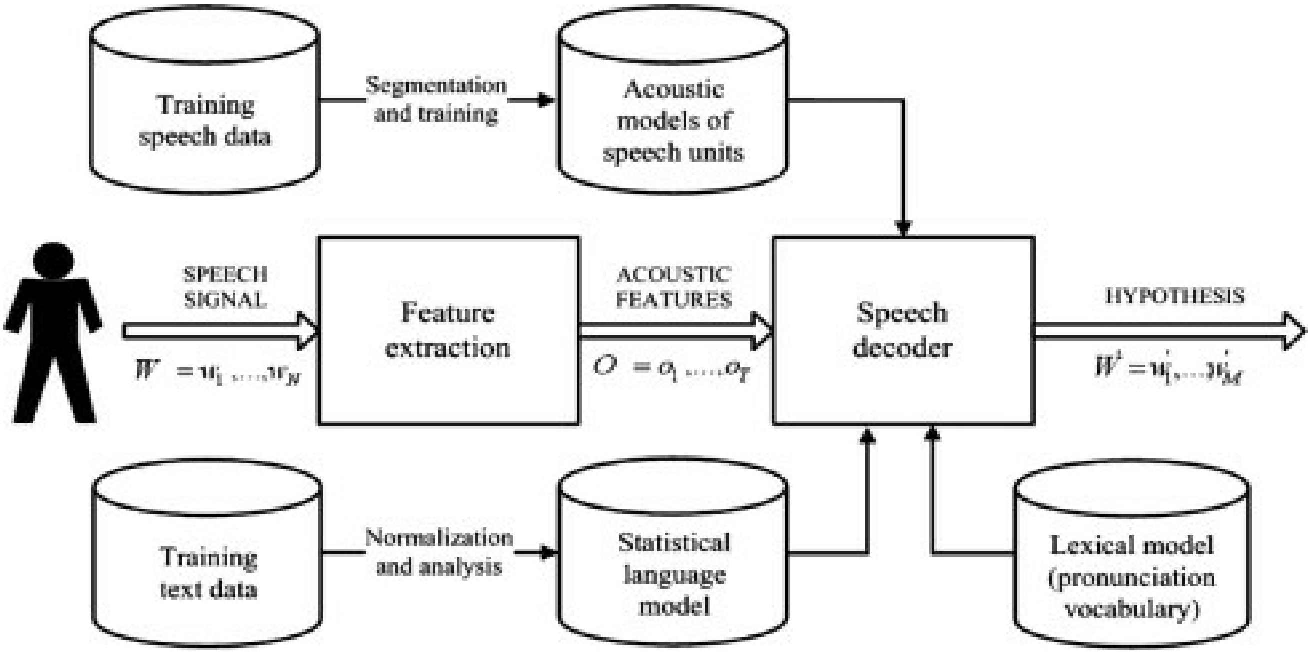
The realised symbol is assumed to have a one to one mapping with the segmented raw audio speech. However, the difficulty in computer speech recognition is the fact that there is significant amount of variation in speech that would make it practically intractable to establish a direct mapping from segmented raw speech audio to a sequence of static symbols. The phenomena known as coarticulation 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 speech transducing medium, the room acoustics. Et cetera.

Another challenge faced by automated speech recognisers is the fact that the boundaries of the words is 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 in this case would be to create assumption that somehow word boundaries can be determined. This first step in speech recognition is referred to as the isolated word recognition case.

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 required 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 of 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} enumerates 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.


Training speech data
Segmentation and training
Acoustic models of speech units
SPEECH SIGNAL
$W = w_1, \ldots, w_N$
Feature extraction
ACOUSTIC FEATURES
$O = o_1, \ldots, o_T$
Speech decoder
HYPOTHESIS
$W' = w'_1, \ldots, w'_M$
Training text data
Normalization and analysis
Statistical language model
Lexical model (pronunciation vocabulary)

```
\begin{figure}
\centering
% Requires \usepackage{graphicx}
\includegraphics[width=7cm]{thesis/images/asr_pipeline}\
\caption{Automatic Speech Recognition Pipeline}
\cite{besacier2014automatic}}\label{fig_2_2_asr_pipeline}
\end{figure}
```

Low Resource Speech Recognition

In this system building speech recognition research, the focus was on the development of a language model and 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 is 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 modeling 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 an algorithm used from one domain and applying this knowledge in a similar domain having different set of system parameters. Early work of this nature was 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 titled speech recognition on a budget, a transfer learning mechanism involving deep neural networks from \citep{kunze2017transfer} is described.

Low Resource language modelling \label{sec_lrlm}

General language modelling is reviewed and then Low resource language modelling is discussed in this section. Recall from the general speech model influenced by Bayes' theorem. The speech recognition model is a product of an acoustic model (likelihood probability) and the language model (prior probability). 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 modeled in terms discrete density $P(W)$, where $W = (w_1, w_2, ..., w_L)$ is a word sequence. The density function $P(W)$ assigns a probability to a particular word sequence W. This value is 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 convention in a probabilistic way. Statistical language modeling therefore, is an estimate for $P(W)$ from a given set of sentences, or corpus.

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

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

\label{eqn_c2_lm01}

\end{equation}

The N-gram model is formed by conditioning of the word history in equation \ref{eqn_c2_lm01}. This therefore becomes
\begin{equation}$P(\mathbf{w}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, w_{k-2}, \ldots, w_{k-N+1})$
\label{eqn_c2_lm02}
\end{equation}

N is typically in the range of 2-4.

N-gram probabilities are estimated from training corpus by counting N-gram occurrences. This is plugged into maximum likelihood (ML) parameter estimate. For example, Given that N=3 then the probability that three words occurred is assuming $C(w_{k-2}w_{k-1}w_k)$ is the number of occurrences of the three words $C(w_{k-2}w_{k-1})$ is the count for $w_{k-2}w_{k-1}w_k$ then
\begin{equation}
$P(w_k|w_{k-1}, w_{k-2}) \approx \frac{C(w_{k-2}\,w_{k-1}\,w_k)}{C(w_{k-2}w_{k-1})}$
\label{eqn_c2_lm03}
\end{equation}

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 w_k, there is a corresponding class c_k, then,

\begin{equation}

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

\label{eqn_c2_lm04}

\end{equation}

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 result of the MLP-based language models was found to be, in cases for models with large parameters, performing 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 are characterised by complex structures that render the architecture beyond the basic feed forward nature where data flows from input to output hence in the RNN architecture we have some feedback neurons as well. 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.

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 rather 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 particularly of interest to this article and forms the basis of the implementation described in this article due to the low resource constraints imposed 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 method employed in this report uses a character-based Neural network language model that employs an LSTM network similar to that of \cite{kim2016character} on 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 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 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 which 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 are modeled using Gaussian mixture models or GMM are also known as senones. \cite{povey2011subspace} postulated a method to factorize these Gaussian mixtures into a globally shared set of parameters that are not 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 bye-product 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 as other deep learning algorithms, as being very 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 around 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 find 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 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 are 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 in this work 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 speech model 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 is enhanced at the same time non-distinguishing features constituting noise is 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 pipe line. It has be shown that a common technique using Mel-frequency cepstral coefficients (MFCCs) \citep{davis1990comparison} 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 reprepresentation 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 isn’t 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} shows that non-stationary behavior 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 additio,n 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 with using deep scattering network and the second that of spoken term discovery criteria being satisfied with the end-to-end speech model supplemented with a language model.

Methodology

System building methodology \citep{nunamaker1990systems} for speech recognition systems require 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 $P_N(W)$ is measured from a set of sentences and is defined as

\begin{equation}$H = \sum_{\mathbf{W}\in\Omega} P_N(\mathbf{W})$

\label{eqn_c2_lm05}

\end{equation}

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

\begin{equation}$PP(W) = 2^H$

\label{eqn_c2_lm06}

\end{equation}

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

\begin{equation}

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

\label{eqn_c2_lm07}

\end{equation}

For a bigram model therefore, equation (\ref{eqn_c2_lm07}) becomes

\begin{equation}

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

\label{eqn_c2_lm08}

\end{equation}

After simplifying we have

\begin{equation}

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

\label{eqn_c2_lm09}

\end{equation}

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

\begin{equation}\label{eqn_2_3_wer}

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

\end{equation}

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

Metrics used for low speech recognition in the zero speech challenge \citep{versteegh2015zero} includes 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

\cite{deng2015**}

First we discuss the problem of classification, that is, predicting a single discrete class variable y given a vector of features $x = (x_1, x_2, \ldots, x_K)$. One simple way to accomplish this is to assume that once the class label is known, all the features are independent. The resulting classifier is called the naive Bayes classifier. It is based on a joint probability model of the form

\begin{equation}

$p(y, \mathbf{x}) = p(y) \prod_{k=1}^{K} p(x_k|y)$

\label{eqn_c3_seqmod00a}

\end{equation}

This model can be described by the directed model shown in Figure 2.2 (left). We can also write this model as a factor graph, by defining a factor $\Psi(y) = p(y)$, and a factor $\Psi_k(y, x_k) = p(x_k|y)$ for each feature . This factor graph is shown in Figure 2.2 (right).

Another well-known classier that is naturally represented as a graphical model is logistic regression (sometimes known as the maximum entropy classifier in the NLP community). In statistics, this classifier is motivated by the assumption that the log probability, $logp(y|x)$, of each class is a linear function of x, plus a normalization constant. This leads to the conditional distribution:

\begin{equation}

$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} exp\{\theta_y + \sum_{j=1}^{K} \theta_{y,j} x_j)\}$

\label{eqn_c3_seqmod00b}

\end{equation}

where $Z(x) = \Sigma_y exp\{\theta_y + \Sigma_{j=1}^{K} \theta_{y,j} x_j\}$ is a normalizing constant, and is a bias weight that acts like $logp(y)$ in naive Bayes. Rather than using one weight vector per class, as in equation (\ref{eqn_c3_seqmod00b}), we can use a different notation in which a single set of weights is shared across all the classes. The trick is to define a set of feature functions that are nonzero only for a single class. To do this, the feature functions can be defined as $f_{y',j}(y, \mathbf{x}) = 1_{\{y'=y\}} x_j$ for the feature weights and $f_{y'}(y, \mathbf{x}) = 1_{\{y'=y\}}$ for the bias weights. Now we can use f_k to index each feature function $f_{y';j}$ and θ_k to index corresponding weight $\theta_{y',j}$. Using this notational trick, the logistic regression model becomes:

\begin{equation}

$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} exp\{\sum_{j=1}^{K} \theta_k f_k(y, \mathbf{x})\}$

\label{eqn_c3_seqmod00c}

\end{equation}

We introduce this notation because it mirrors the notation for conditional random fields that we will present later.

Sequential models

Classifiers predict only a single class variable, but the true power of graphical models lies in their ability to model many variables that are interdependent. In this section, we discuss perhaps the simplest form of dependency, in which the output variables are arranged in a sequence. To motivate this kind of model, we discuss an application from natural language processing, the task of named-entity recognition (NER). NER is the problem of identifying and classifying proper names in text, including locations, such as China; people, such as George Bush; and organizations, such as the United Nations. The named-entity recognition task is, given a sentence, to segment which words are part of entities, and to classify each entity by type (person, organization, location, and so on). The challenge of this problem is that many named entities are too rare to appear even in a large training set, and therefore the system must identify them based only on context.

One approach to NER is to classify each word independently as one of either Person, Location, Organization, or Other (meaning not an entity). The problem with this approach is that it assumes that given the input, all of the named-entity labels are independent. In fact, the named-entity labels of neighboring words are dependent; for example, while New York is a location, New York Times is an organization. One way to relax this independence assumption is to arrange the output variables in a linear chain. This is the approach taken by the hidden Markov model (HMM) [96]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^{\top}$ by assuming that there is an underlying sequence of states $Y = \{y\}_{t=1}^{\top}$ drawn from a finite state set S. In the named-entity example, each observation x_t is the identity of the word at position t, and each state y_t is the named-entity label, that is, one of the entity types Person, Location, Organization, and Other.

To model the joint distribution $p(\mathbf{y}, \mathbf{x})$ tractably, an HMM makes two independence assumptions. First, it assumes that each state depends only on its immediate predecessor, that is, each state y_t is independent of all its ancestors ParseError: KaTeX parse error: Expected '}', got '−' at position 15: y_1; y_2, y_{t−̲2} given the preceding state y_{t-1}. Second, it also assumes that each observation variable x_t depends only on the current state y_t. With these assumptions, we can specify an HMM using three probability distributions: first, the distribution $p(y_1)$ over initial states; second, the transition distribution ParseError: KaTeX parse error: Expected '}', got '−' at position 11: p(y_t|y_{t−̲1}); and finally, the observation distribution $p(x_t|y_t)$. That is, the joint probability of a state sequence $\mathbf{y}$ and an observation sequence $\mathbf{x}$ factorizes as

\begin{equation}

$p(\mathbf{y}, \mathbf{x}) \ = \ \prod_{t=1}^{T} p(y_t|y_{t-1})$

\label{eqn_c3_seqmod01}

\end{equation}

where, to simplify notation, we write the initial state distribution $p(y_1)$ as $p(y_1|y_0)$. In natural language processing, HMMs have been used for sequence labeling tasks such as part-of-speech tagging, named-entity recognition, and information extraction.

Comparison

Of the models described in this section, two are generative (the naive Bayes and hidden Markov models) and one is discriminative (the logistic regression model). In a general, generative models are models of the joint distribution $p(y, \mathbf{x})$, and like naive Bayes have the form $p(y)p(x|y)$. In other words, they describe how the output is probabilistically generated as a function of the input. Discriminative models, on the other hand, focus solely on the conditional distribution $p(y|x)$. In this section, we discuss the differences between generative and discriminative modeling, and the potential advantages of discriminative modeling. For concreteness, we focus on the examples of naive Bayes and logistic regression, but the discussion in this section applies equally as well to the differences between arbitrarily structured generative models and conditional random fields.

The main difference is that a conditional distribution $p(y|x)$ does not include a model of $p(x)$, which is not needed for classification anyway. The difficulty in modeling $p(x)$ is that it often contains many highly dependent features that are difficult to model. For example, in named-entity recognition, an HMM relies on only one feature, the word's identity. But many words, especially proper names, will not have occurred in the training set, so the word-identity feature is uninformative. To label unseen words, we would like to exploit other features of a word, such as its capitalization, its neighboring words, its prefixes and suffixes, its membership in predetermined lists of people and locations, and so on.

The principal advantage of discriminative modeling is that it is better suited to including rich, overlapping features. To understand this, consider the family of naive Bayes distributions equation (\ref{eqn_c3_seqmod00a}). This is a family of joint distributions whose conditionals all take the "logistic regression form" equation

(\ref{eqn_c3_seqmod00c}). But there are many other joint models, some with complex dependencies among x, whose conditional distributions also have the form equation (\ref{eqn_c3_seqmod00c}). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(x)$. CRFs make independence assumptions among y, and assumptions about how the y can depend on x, but not among x. This point can also be understood graphically: Suppose that we have a factor graph representation for the joint distribution $p(y, x)$. If we then construct a graph for the conditional distribution $p(y|x)$, any factors that depend only on x vanish from the graphical structure for the conditional distribution. They are irrelevant to the conditional because they are constant with respect to y.

To include interdependent features in a generative model, we have two choices: enhance the model to represent dependencies among the inputs, or make simplifying independence assumptions, such as the naive Bayes assumption. The first approach, enhancing the model, is often difficult to do while retaining tractability. For example, it is hard to imagine how to model the dependence between the capitalization of a word and its suxes, nor do we particularly wish to do so, since we always observe the test sentences anyway. The second approach is to include a large number of dependent features in a generative model, but to include independence assumptions among them|is possible, and in some domains can work well. But it can also be problematic because the independence assumptions can hurt performance. For example, although the naive Bayes classifier performs well in document classification, it performs worse on average across a range of applications than logistic regression [16].

Furthermore, naive Bayes can produce poor probability estimates. As an illustrative example, imagine training naive Bayes on a data set in which all the features are repeated, that is, $x = (x_1, x_1, x_2, x_2, \ldots, x_K, x_K)$. This will increase the confidence of the naive Bayes probability estimates, even though no new information has been added to the data. Assumptions like naive Bayes can be especially problematic when we generalize to sequence models, because inference essentially combines evidence from different parts of the model. If probability estimates of the label at each sequence position are overconfident, it might be difficult to combine them sensibly.

The difference between naive Bayes and logistic regression is due only to the fact that the first is generative and the second discriminative; the two classifiers are, for discrete input, identical in all other respects. Naive Bayes and logistic regression consider the same hypothesis space, in the sense that any logistic regression classifier can be converted into a naive Bayes classifier with the same decision boundary, and vice versa. Another way of saying this is that the naive Bayes model equation (\ref{eqn_c3_seqmod00a}) denes the same family of distributions as the logistic regression model equation (\ref{eqn_c3_seqmod00c}), if we interpret it generatively as

\begin{equation}

$$p(y, \mathbf{x}) = \frac{exp\{\sum_k \theta_k f_k(y,\mathbf{x})\}}{\sum_{\tilde{y},\tilde{x}} exp\{\sum_k \theta_k f_k(y,\mathbf{x})\}}$$

\label{eqn_c3_seqmod02}

\end{equation}

This means that if the naive Bayes model equation

(\ref{eqn_c3_seqmod00a}) is trained to maximize the conditional likelihood, we recover the same classier as from logistic regression. Conversely, if the logistic regression model is interpreted generatively, as in equation (\ref{eqn_c3_seqmod02}), and is trained to maximize the joint likelihood $p(y; x)$, then we recover the same classier as from naive Bayes. In the terminology of Ng and Jordan [85], naive Bayes and logistic regression form a generative-discriminative pair. For a recent theoretical perspective on generative and discriminative models, see Liang and Jordan
[61].

One perspective for gaining insight into the difference between generative and discriminative modeling is due to Minka [80]. Suppose we have a generative model pg with parameters . By definition, this takes the form
\begin{equation}
$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta)p_g(\mathbf{x}|\mathbf{y}; \theta)$
\label{eqn_c3_seqmod03}
\end{equation}
\begin{equation}
$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta)p_g(\mathbf{y}|\mathbf{x}; \theta)$
\label{eqn_c3_seqmod04}
\end{equation}
where $p_g(x; \theta)$ and $p_g(y|x; \theta)$ are computed by inference, i.e., $p_g(x; \theta) = \Sigma_y p_g(\mathbf{y}; \mathbf{x}; \theta)$ and $p_g(y|x; \theta) = p_g(\mathbf{y}|\mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta)/p_g(\mathbf{x}; \theta)$.

Now, compare this generative model to a discriminative model over the same family of joint distributions. To do this, we need a prior $p(\mathbf{x})$ over inputs, such that $p(\mathbf{x})$ could have arisen from pg with some parameter setting. That is, ParseError: KaTeX parse error: Expected 'EOF', got '‘' at position 29: …athbf{x};\theta‘) =\Sigma_y p_g…. We combine this with a conditional distribution $p_c(\mathbf{y}|\mathbf{x};\,\theta)$ that could also have arisen from pg, that is, $p_c(\mathbf{y}|\mathbf{x};\,\theta) \;=\; p_g(\mathbf{y},\,\mathbf{x};\,\theta)/p_g(\mathbf{x};\,\theta)$. Then the resulting distribution is

\begin{equation}

$p_c(\mathbf{y},\,\mathbf{x}) \;=\; p_c g(\mathbf{x};\,\theta')p_c(\mathbf{y}|\mathbf{x};\,\theta)$

\label{eqn_c3_seqmod05}

\end{equation}

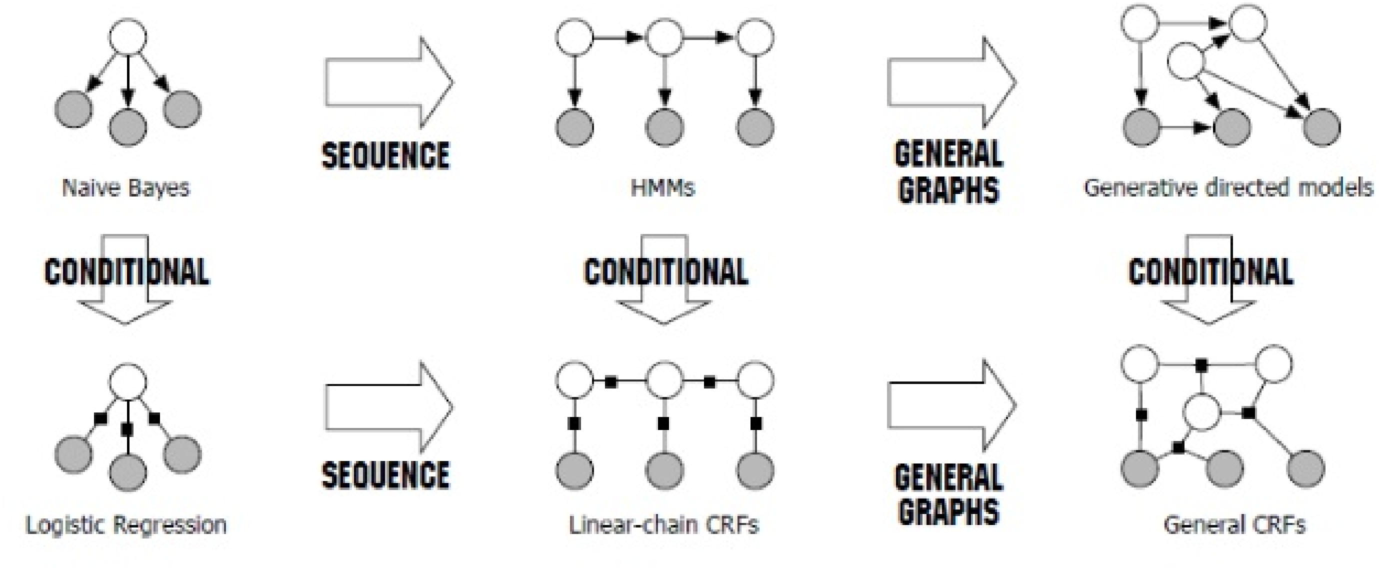
By comparing equation (\ref{eqn_c3_seqmod04}) with equation (\ref{eqn_c3_seqmod05}), it can be seen that the conditional approach has more freedom to fit the data, because it does not require that = 0. Intuitively, because the parameters in equation (\ref{eqn_c3_seqmod04}) are used in both the input distribution and the conditional, a good set of parameters must represent both well, potentially at the cost of trading off accuracy on $p(\mathbf{y}|\mathbf{x})$, the distribution we care about, for accuracy on $p(\mathbf{x})$, which we care less about. On the other hand, this added freedom brings about an increased risk of overfitting the training data, and generalizing worse on unseen data.

To be fair, however, generative models have several advantages of their own. First, generative models can be more natural for handling latent variables, partially-labeled data, and unlabelled data. In the most extreme case, when the data is entirely unlabeled, generative models can be applied in an unsupervised fashion, whereas unsupervised learning in discriminative models is less natural and is still an active area of research.

Second, on some data a generative model can perform better than a discriminative model, intuitively because the input model $p(x)$ may have a smoothing effect on the conditional. Ng and Jordan [85] argue that this effect is especially pronounced when the data set is small. For any particular data set, it is impossible to predict in advance whether a generative or a discriminative model will perform better. Finally, sometimes either the problem suggests a natural generative model, or the application requires the ability to predict both future inputs and future outputs, making a generative model preferable.

Because a generative model takes the form $p(\mathbf{y}; \mathbf{x}) \ = \ p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$, it is often natural to represent a generative model by a directed graph in which in outputs y topologically precede the inputs. Similarly, we will see that it is often natural to represent a discriminative model by a undirected graph, although this need not always be the case.

The relationship between naive Bayes and logistic regression mirrors the relationship between HMMs and linear-chain CRFs. Just as naive Bayes and logistic regression are a generative-discriminative pair, there is a discriminative analogue to the hidden Markov model, and this analogue is a particular special case of conditional random eld, as we explain in the next section. This analogy between naive Bayes, logistic regression, generative models, and conditional random fields is depicted in figure \ref{fig_3_1_crf} below.



\begin{figure}

\centering

% Requires \usepackage{graphicx}

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

\caption{Developemnt of Sequentiial Models}

\cite{xxx}}\label{fig_3_1_crf}

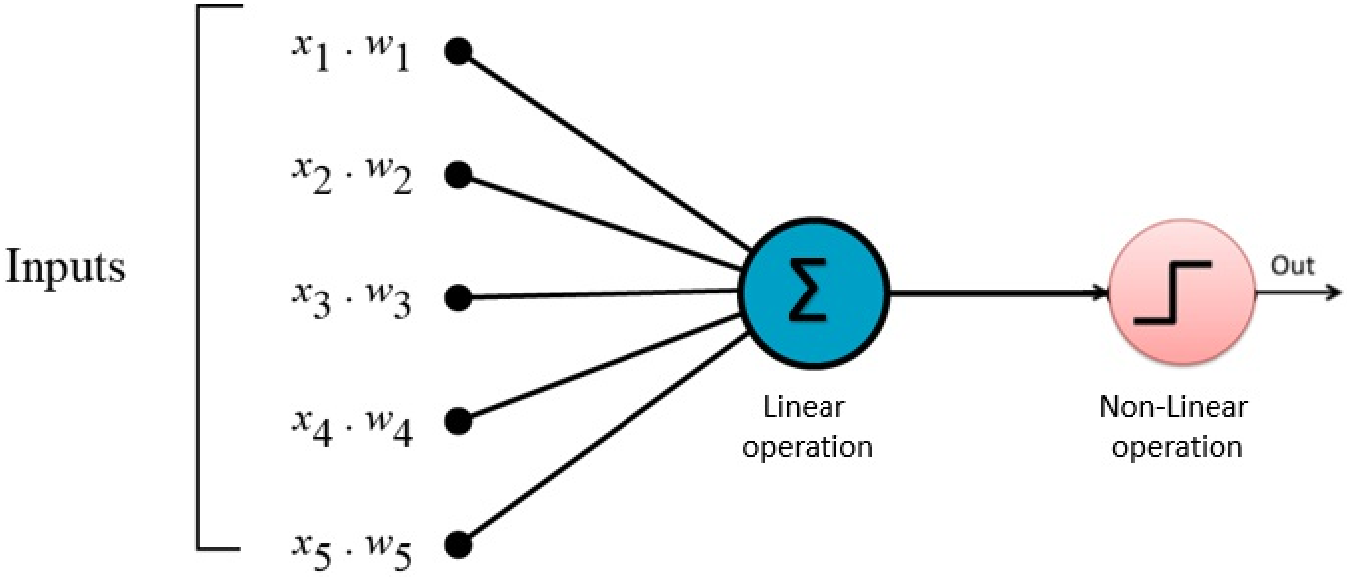
\end{figure}

Neural networks

The HMM method mentioned in the previous section is based on the divide and conquer strategy which has been defined as a generative method in which we use the smaller components represented by the HMM to learn the entire speech process. As earlier mentioned this can also be referred to as the bottom-up strategy. The discriminative method however uses the opposite mechanism. Rather than using the building blocks of speech to determine speech parameters of a HMM, the discriminative strategy rather determines the posterior probability directly using the joint probability distribution of the parameters involved in the discriminative process. The discriminative parameters are discussed in this section where the Neural network discriminative approach is described beginning with the architecture.

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 structure is described by McCulloch and Pitts (1942) (ref) as the perceptron in figure \ref{fig_3_2_ptron} below



\begin{figure}

\centering

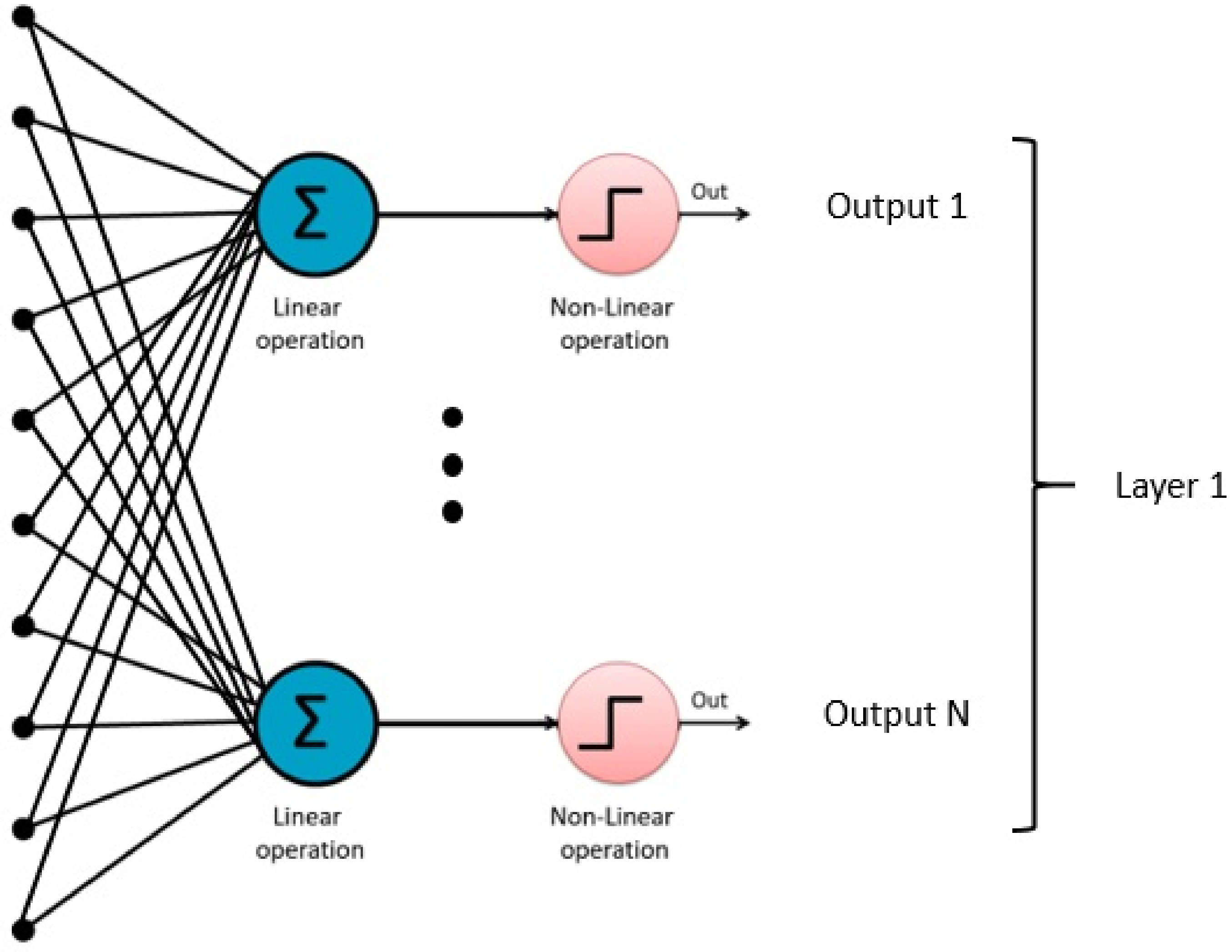
% Requires \usepackage{graphicx}

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

\caption{Perceptron} \cite{xxx}}\label{fig_3_2_ptron}

\end{figure}

The linear operation is the sum of the inputs multiplied by a weight vector. The non linear operation is the given by a any one of a selection of nonlinear functions. In the figure above this is given by 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} below :



\begin{figure}

\centering

% Requires \usepackage{graphicx}

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

\caption{Perceptron} \cite{xxx}}\label{fig_3_2_nn}

\end{figure}

In this regime 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.

Multilayer 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

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

\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.

Sigmoid and softmax 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 essence the step function is a squashing function that converts the inputs into a Naive Bayes function asking what is the probability that the output belongs to one of the input classes (C_y) given the data $(\mathbf{x})$.
\begin{equation}
$p(C_1|\mathbf{x}) \ = \ f(a) \ = \ f(\mathbf{w}^\top\mathbf{x} \ + \ w_0)$
\label{eqn_c3_nn_02}\end{equation}
In a two class problem with classes and , then we can express the posterior probability of C_1 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}
$p(C_1|(x) \ = \ \frac{1}{1+\frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)}}$
\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{eqution}$p(C_1|\mathbf{x}) = f(a) = \frac{1}{1+e^{-a}}$ - - - (7)

\label{eqn_c3_nn_06}\end{equation}

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

Rather than using the sigmoid function for multi-class classification a similar softmax 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}

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

\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, 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.

Back propagation algorithm

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 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 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 acceptable value. This is often referred to in machine learning as convergence (ref).

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. Therefore 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_01}\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}$\nabla_{\mathbf{w}} E \;=\; \left(\frac{\partial E}{\partial w_{10}},\ \ldots\ ,\ \frac{\partial E}{\partial w_{ki}},\ \ldots\ ,\ \frac{\partial E}{\partial w_{Kd}} \right)$

\label{eqn_c3_nn_01}\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_01}\end{equation}

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

LSTM network

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

A DNN computes a distribution function $p(c|x_t)$ using a series of hidden layers followed by an output layer. Given an input vector xt the first hidden layer activations are a vector computer \begin{equation}$h^{(1)} \ = \ \sigma(\mathbf{W}^{(1)T} x_t \ + \ b^{(1)})$ \label{eqn_c3_dnn01}\end{equation}

The matrix $W^{(1)}$ and vector $b^{(1)}$ are the weight matrix and bias vector for the layer. The function $\sigma(\cdot)$ is a pointwise nonlinearity. We use rectifier nonlinearities and thus choose $\sigma(z) \ = \ max(z, 0)$.

DNNs have arbitrarily many hidden layers. After the first hidden layer, the hidden activations $h^{(i)}$ for the layer i are computed as \begin{equation}$h^{(1)} \ = \ \sigma(\mathbf{W}^{(1)T} h^{(i-1)} \ + \ b^{(1)})$ \label{eqn_c3_dnn02}\end{equation}

To obtain a proper distribution over the set of possible characters c the final layer of the network is a softmax output layer of the form, \begin{equation}$p(c \ = \ c_k|x_t) \ = \ \frac{exp(-(\mathbf{W}_k^{(s)T} h^{(i-1)} + b_k^{(1)}))}{\sum_j \ exp(-(\mathbf{W}_k^{(s)T} h^{(i-1)} + b_k^{(1)}))}$ \label{eqn_c3_dnn02}\end{equation}
where $W_k^{(s)}$ is the k-th column of the output weight matrix $W^{(s)}$ and $b_k^{(k)}$ is the scalar bias term. We can compute the subgradient for all parameters of the DNN given a training example and thus utilise gradient-based optimisation techniques. This same formulation is used in DNN-HMM models to predict distribution over senones instead of characters.

Recurrent Neural Network

A transcription W has many temporal dependencies which a DNN may not sufficiently capture. At each timestep t the DNN computes its output using only the input features x_t, ignoring previous hidden representations and output distributions. To enable better modeling of the temporal dependencies present in a problem, we use a RDNN. In a RDNN we select one hidden layer j to have a temporally recurrent weight matrix $W^{(f)}$ and compute the layer's hidden activations as \begin{equation}$h_t^{(j)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(j)T} h_{t-1}^{(j)} + b^{(j)}))$ \label{eqn_c3_rnn01}\end{equation}
Note that we now make the distinction $h_t^{(j)}$ for the hidden activation vector of layer j at timestep t since it now depends upon the activation vector of layer j at time ParseError: KaTeX parse error: Expected 'EOF', got '−' at position 3: t −̲ 1.

When working with RDNNs, we found it important to use a modified version of the rectifier nonlinearity. This modified function selects $\sigma(z) = min(max(z, 0), 20)$ which clips large activations to prevent divergence during network training. Setting the maximum allowed activation to 20 results in the clipped rectifier acting as a normal rectifier function in all but the most extreme cases.

Aside from these changes, computations for a RDNN are the same as those in a DNN as described in DNN section above. Like the DNN, we can compute a subgradient for a RDNN using a method sometimes called backpropagation through time. In our experiments we always compute the gradient completely through time rather than truncating to obtain an approximate subgradient.

Gated Recurrent Units

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 give 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}

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

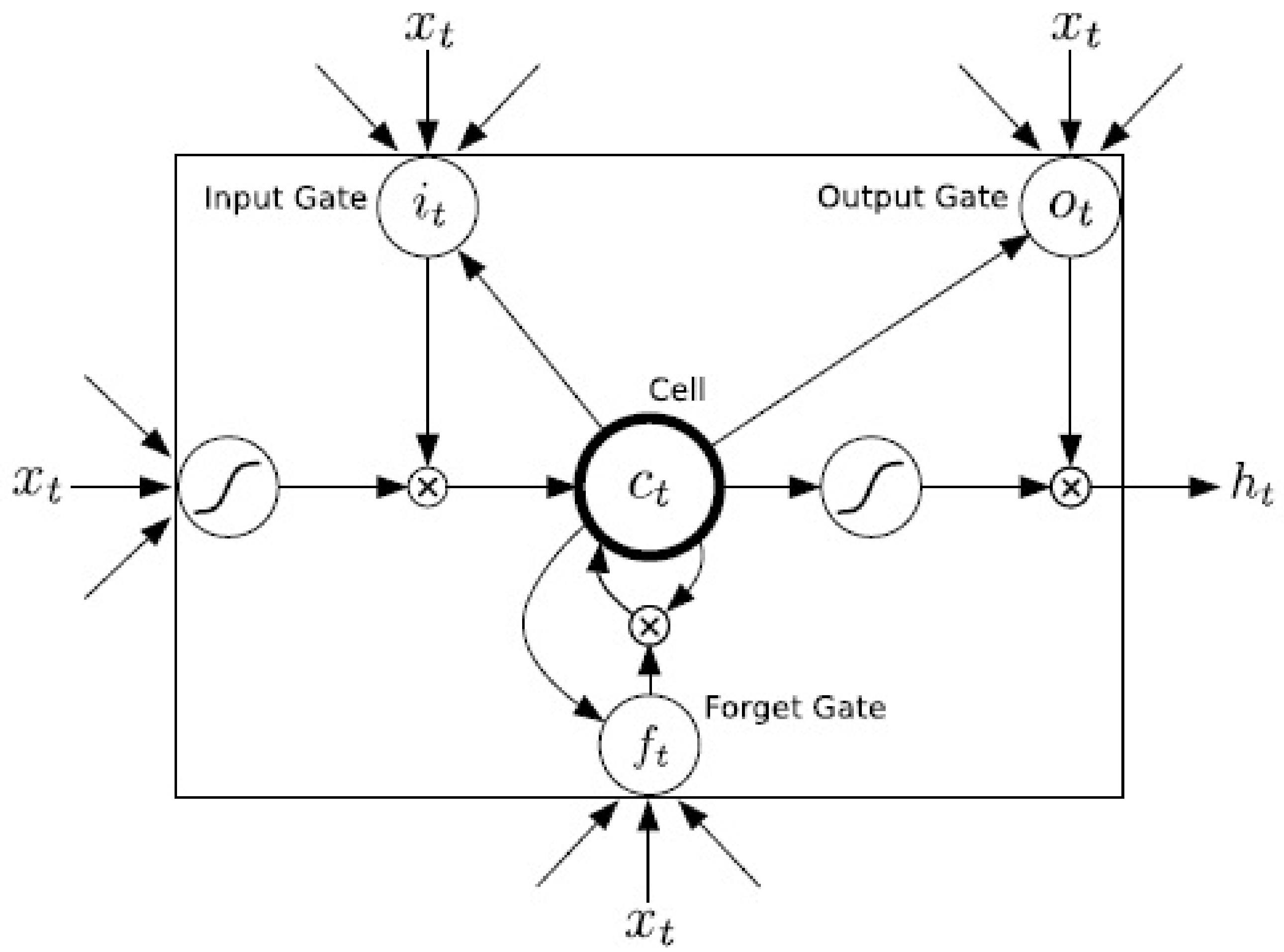
\label{eqn_c3_lstm04}\end{equation}

\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
\cite{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 not 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 $\mathbf{c}_t$.

Deep speech architecture

While forward recurrent connections reflect the temporal nature of the audio input, a perhaps more powerful sequence transduction model is a BRDNN, which maintains state both forwards and backwards in time. Such a model can integrate information from the entire temporal extent of the input features when making each prediction. We extend the RDNN to form a BRDNN by again choosing a temporally recurrent layer j. The BRDNN creates both a forward and backward intermediate hidden representation which we call $h_t^{(f)}$ and $h_t^{(b)}$ respectively. We use the temporal weight matrices $W^{(f)}$ and $W^{(b)}$ to propagate $h_t^{(f)}$ forward in time and $h_t^{(b)}$ backward in time respectively. We update the forward and backward components via the equations,
\begin{equation}$h_t^{(f)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(f)T} h_{t-1}^{(j)} + b^{(j)}))$
\label{eqn_c3_ds01}\end{equation}
\begin{equation}$h_t^{(b)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(b)T} h_{t+1}^{(b)} + b^{(j)}))$
\label{eqn_c3_ds02}\end{equation}
Note that the recurrent forward and backward hidden representations are computed entirely independently of each other. As with the RDNN we use the modified nonlinearity function $\sigma(z) = min(max(z, 0), 20)$.

To obtain the final representation $h_t^{(j)}$ for the layer we sum the two temporally recurrent components,
\begin{equation}$h_t^{(j)} = h_t^{(f)} + h_t^{(b)}$
\label{eqn_c3_ds03}\end{equation}
Aside from this change to the recurrent layer the BRDNN computes its output using the same equations as the RDNN. As for other models, we can compute a subgradient for the BRDNN directly to perform gradient-based optimization.

CTC Loss Algorithm

In accord with Deep Speech: Scaling up end-to-end speech recognition, the loss function used by our network should be the CTC loss function[2]. Unfortunately, as of this writing, the CTC loss function[2] is not implemented within TensorFlow[5]. Thus we will have to implement it ourselves. The next few sections are dedicated to this implementation.

The CTC algorithm was specifically designed for temporal classification tasks; that is, for sequence labelling problems where the alignment between the inputs and the target labels is unknown. Unlike hybrid approaches combining HMM and DNN, CTC models all aspects of the sequence with a single neural network, and does not require the network to be combined with a HMM. It also does not require pre-segmented training data, or external post-processing to extract the label sequence from the network outputs.

Generally, neural networks require separate training targets for every timeslice in the input sequence. This has two important consequences.

First, it means that the training data must be pre-segmented to provide targets for every timeslice. Second, as the network only outputs local classifications, global aspects of the sequence, such as the likelihood of two labels appearing consecutively, must be modelled externally. Indeed, without some form of post-processing the final label sequence cannot reliably be inferred at all.

CTC avoids this problem by allowing the network to make label predictions at any point in the input sequence, so long as the overall sequence of labels is correct. This removes the need for pre-segmented data, since the alignment of the labels with the input is no longer important. Moreover, CTC directly outputs the probabilities of the complete label sequences, which means that no external post-processing is required to use the network as a temporal classifier.

For a sequence labelling task where the labels are drawn from an alphabet A, CTC consists of a softmax output layer, our `layer_6`, with one more unit than there are labels in `A`. The activations of the first `|A|` units are the probabilities of outputting the corresponding labels at particular times, given the input sequence and the network weights. The activation of the extra unit gives the probability of outputting a $blank$, or no label. The complete sequence of network outputs is then used to define a distribution over all possible label sequences of length up to that of the input sequence.

Defining the extended alphabet $A' = A \cup \{blank\}$, the activation $y_{t,p}$ of network output p at time t is interpreted as the probability that the network will output element p of A' at time t, given the length T input sequence x. Let A'^T denote the set of length T sequences over A'. Then, if we assume the output probabilities at each timestep to be independent of those at other timesteps (or rather, conditionally independent given x), we get the following conditional distribution over $\pi \in A'^T$:

\begin{equation}

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

\label{eqn_c3_ctc01}\end{equation}

From now on we refer to the sequences π over A' as paths, to distinguish them from the label sequences or labellings l over A. The next step is to define a many-to-one function $\mathcal{B} : A'^T \to A^{\leq T}$, from the set of paths onto the set $A^{\leq T}$ 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 $l \in A^{\leq T}$ can be calculated by summing the probabilities of all the paths mapped onto it by $\mathcal{B}$:

\begin{equation}

$\Pr(l \mid x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} \Pr(\pi \mid x)$

\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.

In the original formulation of CTC there were no blank labels, and $\mathcal{B}(\pi)$ was simply π with repeated labels removed. This led to two problems. First, the same label could not appear twice in a row, since transitions only occurred when π passed between different labels. Second, the network was required to continue predicting one label until the next began, which is a burden in tasks where the input segments corresponding to consecutive labels are widely separated by unlabelled data (for example, in speech recognition there are often pauses or non-speech noises between the words in an utterance).

Forward-backward algorithm

So far we have defined the conditional probabilities $\Pr(l \mid x)$ of the possible label sequences. Now we need an efficient way of calculating them. At first sight, the previous equation suggests this will be problematic. The sum is over all paths corresponding to a given labelling. The number of these paths grows exponentially with the length of the input sequence. More precisely, for a length T input sequence and a length U labelling there are $2^{T-U^2+U(T-3)}3^{(U-1)(T-U)-2}$

paths.

Fortunately the problem can be solved with a dynamic-programming algorithm similar to the forward-backward algorithm for HMM's[6]. The key idea is that the sum over paths corresponding to a labelling l can be broken down into an iterative sum over paths corresponding to prefixes of that labelling.

To allow for blanks in the output paths, we consider a modified "label sequence" l', with blanks added to the beginning and the end of l, and inserted between every pair of consecutive labels. If the length of l is U, the length of l' is $U' = 2U + 1$. In calculating the probabilities of prefixes of l' we allow all transitions between blank and non-blank labels, and also those between any pair of distinct non-blank labels.

For a labelling l, the forward variable $\alpha(t, u)$ is defined as the summed probability of all length t paths that are mapped by $\mathcal{B}$ onto the length $\lfloor u/2 \rfloor$ prefix of l. (Note, ParseError: KaTeX parse error: $ within math mode 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 subsequence $s_p, s_{p+1}, \ldots, s_{q-1}, s_q$, and define the set $V(t, u) \equiv \{\pi \in A'^t : \mathcal{B}(\pi) = l_{1:\lfloor u/2 \rfloor}$ and $\pi_t = l'_u\}$. We can then define $\alpha(t, u)$ as
\begin{equation}
$\alpha(t, u) \equiv \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y_{i,\pi_i}$
\label{eqn_c3_ctc04}
As we will see, the forward variables at time t can be calculated recursively from those at time ParseError: KaTeX parse error: Expected

'EOF', got '−' at position 3: t −̲ 1.

Given the above formulation, the probability of l can be 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{eequation}

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}

$$\alpha(1,\, 1) \;=\; y_{1,b}$$

$$\alpha(1,\, 2) \;=\; y_{1,l_1}$$

$$\alpha(1,\, u) \;=\; 0,\; \forall u \;>\; 2$$

\label{eqn_c3_ctc05}\end{equation}

Thereafter the variables can be calculated recursively:

\begin{equation}

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

\label{eqn_c3_ctc06}\end{equation}

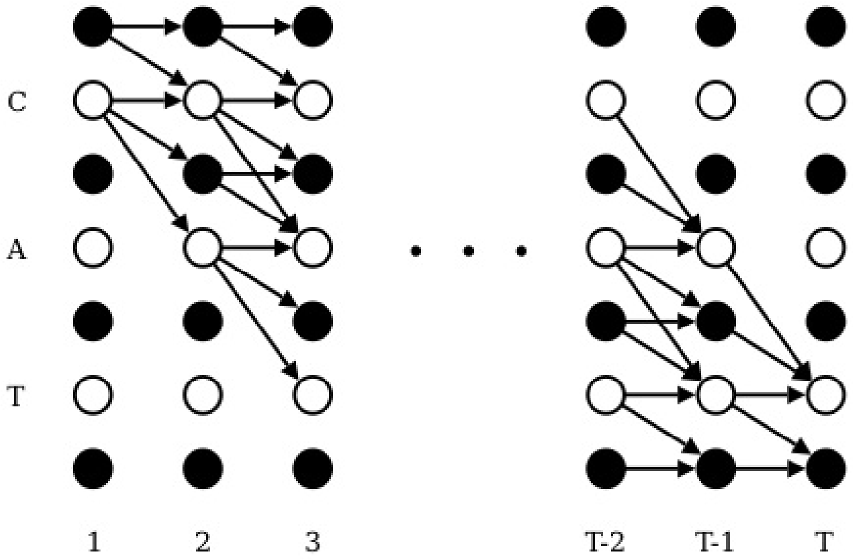
where

\begin{equation}

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

\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=7cm]{thesis/images/Lattice.png}\

\caption{Beam Search Lattice Structure}

\cite{xxx}}\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 timesteps 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

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

In practice, the above recursions will soon lead to underflows on any digital computer. A good way to avoid this is to work in the log scale, and only exponentiate to find the true probabilities at the end of the calculation. A useful equation in this context is

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

Loss Function

The CTC loss function $\mathcal{L}(S)$ is defined as the negative log probability of correctly labelling all the training examples 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}

Because the function is differentiable, its derivatives with respect to the network weights can be calculated with backpropagation through time, and the network can then be trained with any gradient-based non-linear optimisation algorithm.

\begin{equation}

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

\label{eqn_c3_ctc12}\end{equation}

Obviously

\begin{equation}

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

\label{eqn_c3_ctc12}\end{equation}

Now if we identify l and z and define $X(t,\, u) \;\equiv\; \{\pi \;\in\; A'^{T} \;:\; \mathcal{B}(\pi) \;=\; z,\; \pi_t \;=\; z'_u\}$, then our definition of $\alpha(t,\, u)$ and $\beta(t,\, u)$ imply
\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}

thus substituting our previous expression for $\Pr(\pi \,|\, x)$
\begin{equation}
$\alpha(t,\, u)\beta(t,\, u) \;=\; \sum_{\pi \in X(t,u)} \;\Pr(\pi \,|\, x)$
\label{eqn_c3_ctc14}\end{equation}
From our expression for $\Pr(l \,|\, x)$ we can see that this is the portion of the total probability of $\Pr(z \,|\, x)$ due to those paths going through z'_u at time t. For any t, we can therefore sum 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 the example loss is given by
\begin{quation}$\mathcal{L}(x,\, z) \;=\; -\,\ln\, \sum_{u=1}^{|z'|} \;\alpha(t,\, u)\beta(t,\, u)$
\label{eqn_c3_ctc16}\end{equation}
as
\begin{equation}$\mathcal{L}(S) \;=\; \sum_{(x,z)\in S} \mathcal{L}(x,\, z)$
\label{eqn_c3_ctc17}\end{equation}

the gradient of $\mathcal{L}(S)$ can be computed by computing the gradient of $\mathcal{L}(x,\, z)$. This gradient can be computed using the formulas above and

TensorFlow’s automatic differentiation.

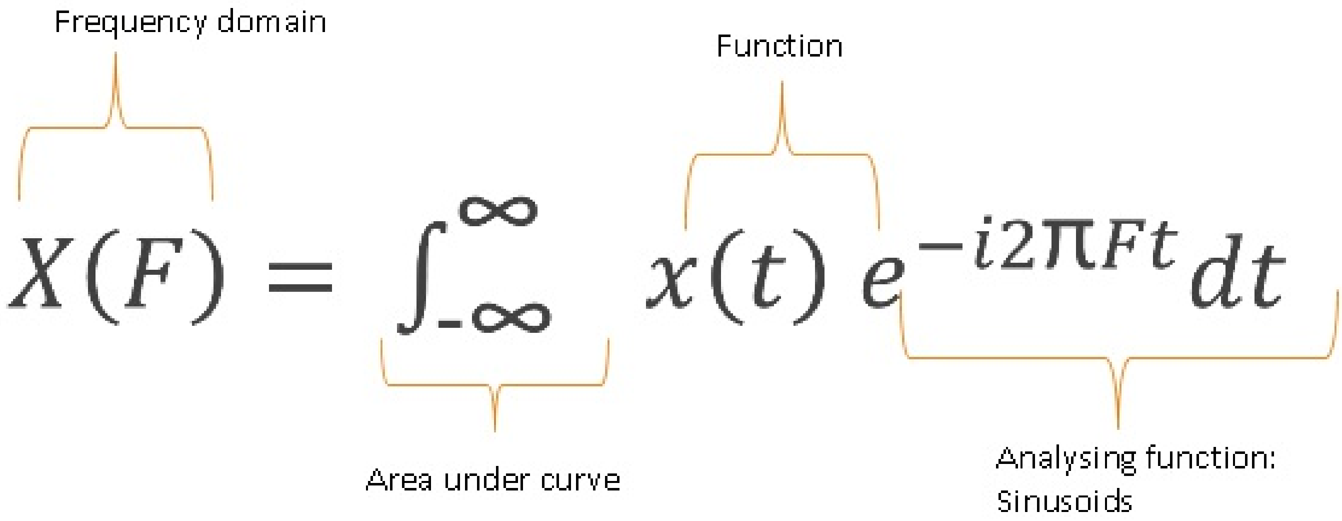
Deep Scattering Network

Curve fitting is a very common theme in pattern recognition. The concept of invariant functions are mapping functions that approximate a discriminating function when it is reduced from a high dimensional space to a low dimensional space \cite{mallat2016understanding}. In this chapter we build an invariance function called a scattering transform which enables invariance of groups of deformations that could possibly distort speech signals yet invariant to the higher level characterisations useful for classifying speech sounds. Works done by \citep{peddinti2014deep,zeghidour2016deep,anden2011multiscale,sainat 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 frequences cepstal coefficients (MFCCs) for 50 and 430 hours of English Broadcast News speech corpus.

This chapter iterates the use of the Fourier transform as the starting analysis function for buildng 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.

Fourier transform

The Fourier transform often referred to as the power spectrum, allows us to discover frequencies contained within a function. 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} \cite{xxx}}\label{fig_4_1_fourier_eqn}

\end{figure}

From the orthorgonal property of complex exponential function, two functions are orthogonal if $\int f(x)g(x) = 0$ where f(x) and g(x) are complimentary 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

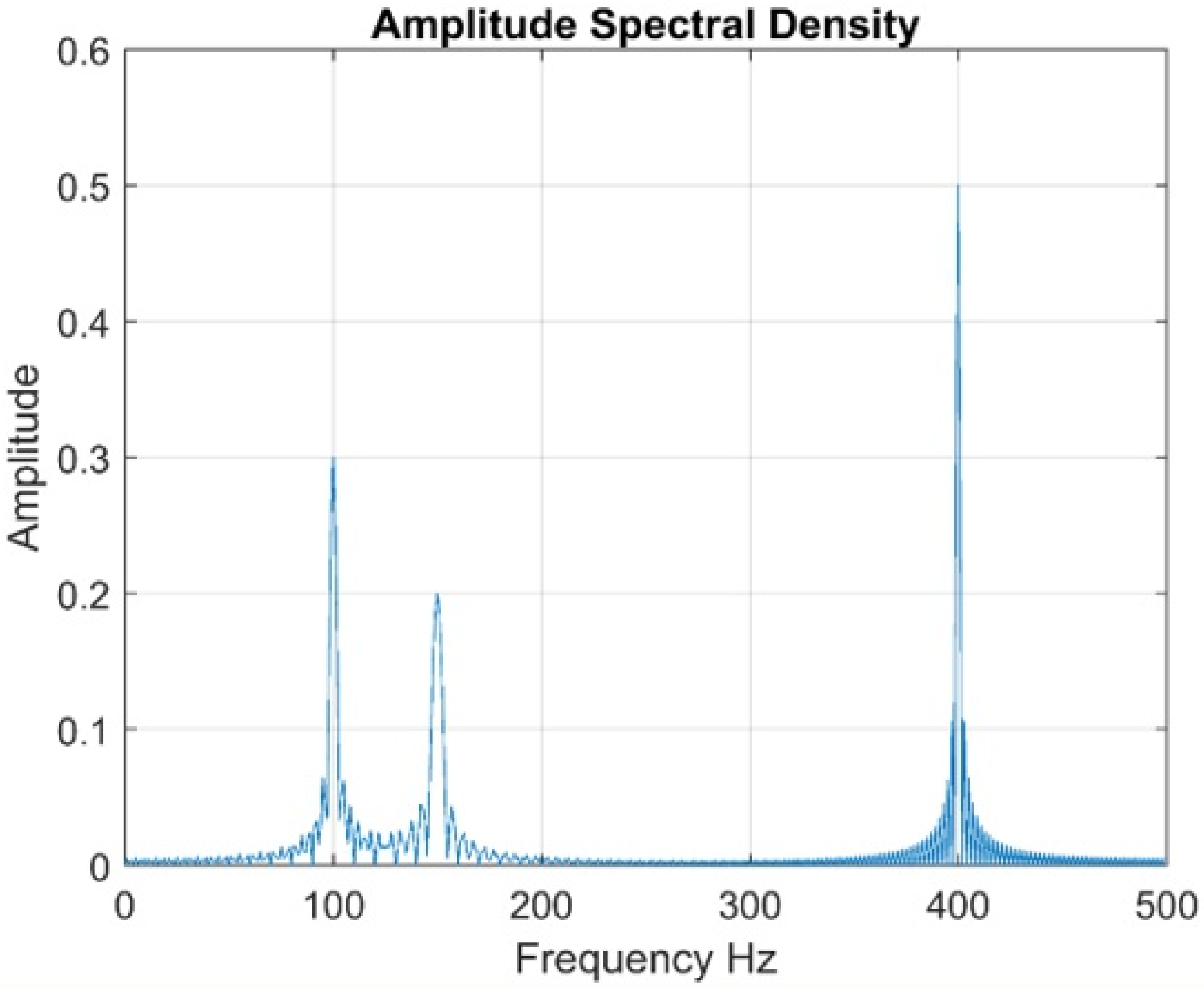
\begin{equation}

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

\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.

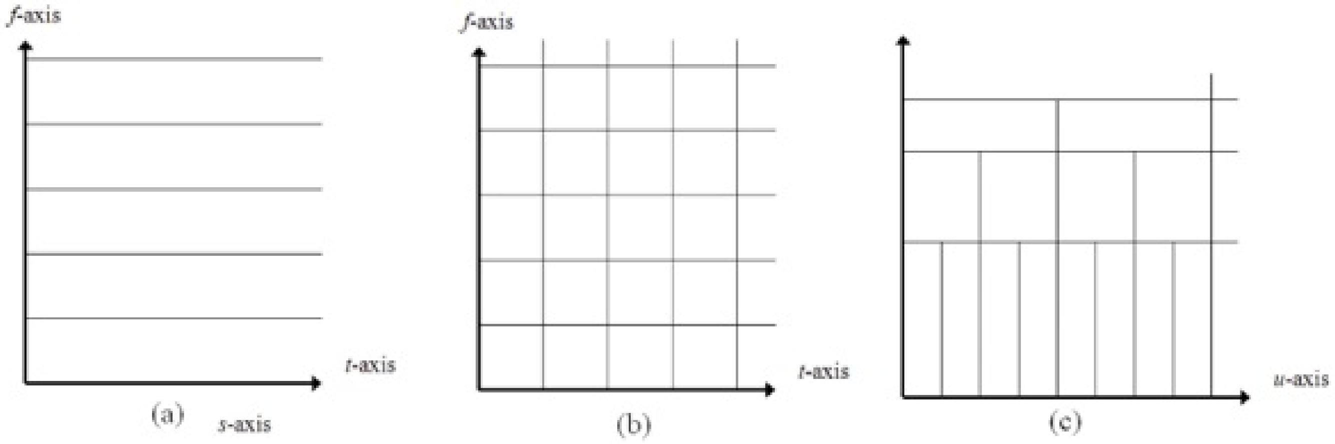


```
\begin{figure}
\centering
% Requires \usepackage{graphicx}
\includegraphics[width=7cm]{thesis/images/spectral.png}\
\caption{Sample Spectrogram} \cite{xxx}}\label{fig_4_2_spectral}
\end{figure}
```

The Fourier transform discussed in the previous section 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 freqeuncy accuracy and vice versa. In the next section on the continuous wavelet transform, how the wavelet transform improves on the weakneses of the Fourer Transform and the STFT is reviewed.

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_3_tftile}



\begin{figure}

\centering

% Requires \usepackage{graphicx}

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

\caption{Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) © Wavelet transform}

\cite{xxx}}\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 representation of the continuous wavelet function is given as:

\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}

There are a few mother wavelet functions discussed later in this chapter. Generally a wavelet can be identified as being an energy spike in an infinite signal of zero energy whose total energy within the spiking regions also sum to zero.

Discrete and Fast wavelet transform

Suppose the scaling function and wavelet function and bases of the following wavelets (Haar, Daubechies) are known. We can approximate discrete signal $l^2(\mathbb{Z})^1$ by

\begin{equation}

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

\label{eqn_c4_wavelet02}

\end{equation}

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

\begin{equation}

$\frac{1}{\sqrt{M}} \sum_k W_\phi[j_0, k]\phi_{j_0,k}[n]$

\label{eqn_c4_wavelet03}

\end{equation}

\begin{equation}

$\frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_\psi[j, k]\psi_{j,k}[n] \quad j \geq j_0$

\label{eqn_c4_wavelet04}

\end{equation}

Equation (\ref{eqn_c4_wavelet03}) are called approximation coefficient while (\ref{eqn_c4_wavelet04}) are called detailed coefficients.

Wavelet types

Daubechies

Based on these equations, Daubechies [9], designed a type of wavelet for a given vanishing moment p and found the minimum size discrete filter. The conclusion is that if we want the wavelet function with p vanishing moments, the minimum filter is 2p. The derivation starts from (5.17), rewrite as

-
  -
    - (5.19)

      The absolute-square of this function is

- - - (5.20)

      The last step makes $P\ \left(sin^2\frac{\omega}{2}\right)\ =\ |R(e^{j\omega})|^2$.

      Recall (5.6), we can determine the form of P(x).

      Let $y\ =\ sin^2\frac{\omega}{2}$, we have
- - - (5.21)

      A theorem in algebra, called Bezout theorem, can solve this equation. The unique solution is
- - (5.22)

  The polynomial P(y) is the minimum degree polynomial satisfying equation (5.21). Once we have P(y), the polynomial $R(e^{j\omega})$ can be derived. First we decompose $R(e^{j\omega})$ according to its roots
- - - (5.23)

      Let , the relation between P and R is
- - - (5.24)

      By solving the roots of ParseError: KaTeX parse error: Expected '}', got '−' at position 15: P\left(\frac{2−̲z−z^{−1}}{4}\ri…, we have the roots of R, $\{a_k\,,\ 1/a_k\}_{k=0,1,\ldots,m}$ and $r_0\ =\ 2^{p-1}$ Usually, we choose a_k lies in the unit circle to have minimum phase filter.

Taking p=2 for an example, the obtained polynomial P(y) is

- - - (5.25)
- - - (5.26)

The roots are $2 + \sqrt{3}$ and ParseError: KaTeX parse error: Expected 'EOF', got '−' at position 2: 2−̲\sqrt{3}. After factorisation, we have the lowpass filter to be

(5.27)

The discrete-time domain representation is

(5.28)

The result is the minimum size filter with 2 vanishing moments and the corresponding filter size is 4. Recall the conclusion mentioned above, the filter size is two times the vanishing moment. Higher order Daubechies wavelets are derived at similar way.

Symlets

Take a look at the discrete filters and the scaling/wavelet functions of Daubechies wavelets. These functions are far from symmetry. That’s because Daubechies wavelets select the minimum phase square root such that the energy concentrates near the starting point of their support. Symmlets select other set of roots to have closer symmetry but with linear complex phase.

Coiflets

For an application in numerical analysis, Coifman asked Daubechies [9] to construct a family of waveletsψthat have p vanishing moments, minimum-size support and

-
  -
    - (5.29)
-
  -
    - (5.30)

      The equation above can be taken as some requirement about vanishing moments of the scaling function. The resulting coiflets has a support of size3 p−1.

Mel filter banks

The first step in any automatic speech recognition system is to extract features i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc.

The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

Mel Frequency Cepstral Coefficents (MFCCs) are a feature widely used in automatic speech and speaker recognition. They were introduced by Davis and Mermelstein in the 1980’s, and have been state-of-the-art ever since. Prior to the introduction of MFCCs, Linear Prediction Coefficients (LPCs) and Linear Prediction Cepstral Coefficients (LPCCs) and were the main feature type for automatic speech recognition (ASR), especially with HMM classifiers.

Steps at a Glance

We will give a high level intro to the implementation steps, then go in depth why we do the things we do. Towards the end we will go into a more detailed description of how to calculate MFCCs.

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the DCT of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.

An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn’t change much (when we say it doesn’t change, we mean statistically i.e. statistically stationary, obviously the samples are constantly changing on even short time scales). This is why we frame the signal into 20-40ms frames. If the frame is much shorter we don’t have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame.

The next step is to calculate the power spectrum of each frame. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates (which wobbles small hairs), different nerves fire informing the brain that certain frequencies are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.

The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition (ASR). In particular the cochlea can not discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filterbank: the first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The Mel scale tells us exactly how to space our filterbanks and how wide to make them.

Once we have the filterbank energies, we take the logarithm of them. This is also motivated by human hearing: we don’t hear loudness on a linear scale. Generally to double the percieved volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes our features match more closely what humans actually hear. Why the logarithm and not a cube root? The logarithm allows us to use cepstral mean subtraction, which is a channel normalisation technique.

The final step is to compute the DCT of the log filterbank energies. There are 2 main reasons this is performed. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade ASR performance, so we get a small improvement by dropping them.

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 ϕ and higher frequency complex analytic wavelets ψ_{λ_1}:
$Wx \;=\; (x \,\star\, \phi(t),\, x \,\star\, \psi_{\lambda_1}(t))_{t \in \mathbb{R}, \lambda_1 \in \Lambda_1}$ - - - (1)

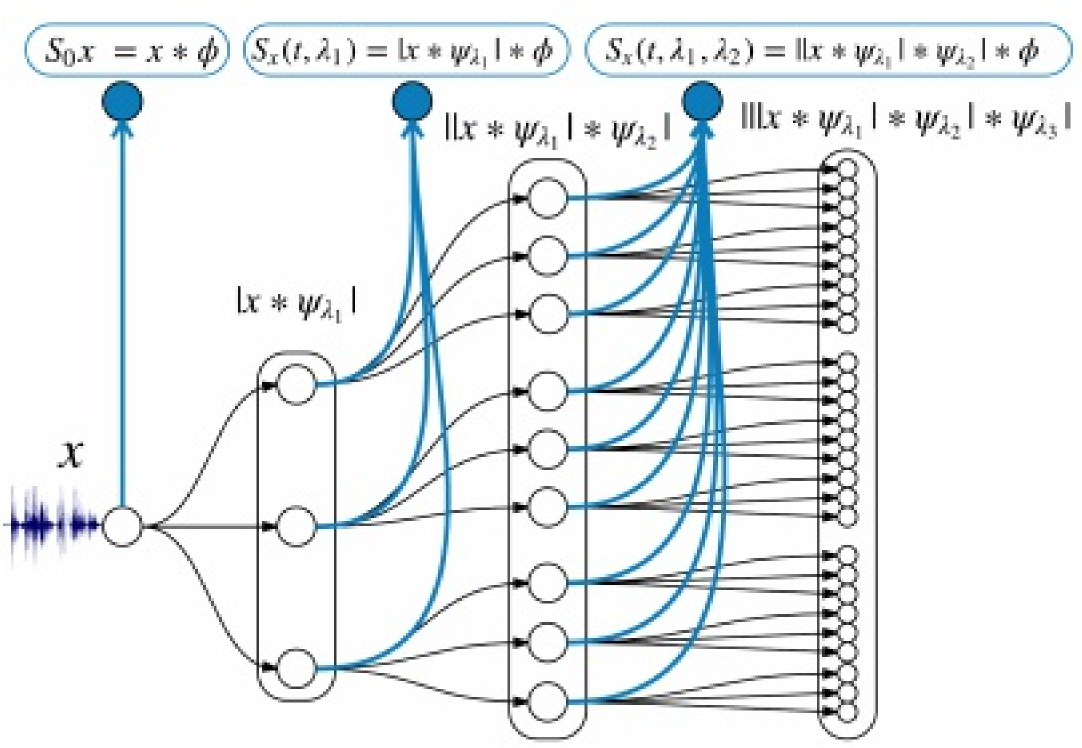
We apply a modulus operator to the wavelet coefficients to remove complex phase and extract envelopes at different resolutions
$|W|x \;=\; (x \,\star\, \phi(t),\, |x \,\star\, \psi_{\lambda_1}(t)|)_{t \in \mathbb{R}, \lambda_1 \in \Lambda_1}$ - - - (2)
$S_0 x \;=\; x \,\star\, \phi(t)$ is locally invariant to translation thanks to the time averaging ϕ. 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 ParseError: KaTeX parse error: Expected '}', got '\labmda' at position 39: …)=|x\star\psi_{\̲l̲a̲b̲m̲d̲a̲_1}|\star \phi(… - - - (3)

It is shown in [6] that if the wavelets ψ_{λ_1} have the same frequency resolution as the standard mel-filters, then the S1x 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 ψ_{λ_2}:
$|W_2||x \star \phi_{\lambda_1}| = (|x \star \psi_{\lambda_1}| \star \phi, ||x \star \psi \lambda 1| \star \psi \lambda 2|)_{\lambda_2 \in \Lambda_2}$ - - - (4)

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

-
  -
    - (5)
      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


$S_0x = x * \phi$
$S_x(t, \lambda_1) = |x * \psi_{\lambda_1}| * \phi$
$S_x(t, \lambda_1, \lambda_2) = ||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| * \phi$
$||x * \psi_{\lambda_1}| * \psi_{\lambda_2}|$
$|||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| * \psi_{\lambda_3}|$
$|x * \psi_{\lambda_1}|$
x

```
\begin{figure}
\centering
% Requires \usepackage{graphicx}
\includegraphics[width=7cm]{thesis/images/spectral.png}\
\caption{Scattering network - 2 layers deep}
\cite{zeghidour2016deep}}\label{fig_4_3_scatter}
\end{figure}
```

Wakirike Language Models

This work draws upon the premise that the grammar of a language is expressed in the character sequence pattern which is ultimately expressed in words and therefore the abstract grammar rules can be extracted and learned by a character-based RNN neural network.

Dataset Preparation

The Wakirike new testament bible served as the source of data for the deep neural network training. As there wasn't a soft or on-line copy of the Wakirike new testament bible readily available for use, the four gospels of the Wakirike new testament bible were quickly typed and duplicated once giving a complete corpus word size of about 165,180 words. This gracefully yielded a character count of about 1,113,820 characters void of punctuation characters. The dataset was then divided 1 in 10 parts for testing and the remaining 9 parts were used for training.

During data preparation, the dataset was first striped off all punctuation marks such that only characters and spaces are selected. Next, each character in the dataset was substituted with its equivalent Unicode numeric representation. Finally the numeric values were one-hot encoded deriving a sparse array of values having unique indexes set to one for each unique Unicode value and zero every where else. One-hot encoded array therefore, for each character input.

\subsection{LSTM Training}

In order to optimise performance of the network a modified LSTM cell known as the Gated Recurrent Unit (GRU) replaced the LSTM in the neural network model. These GRUs have been shown to give similar performance to regular LSTMs with a lighter system resource consumption foot print \cite{cho2014learning}. The internal network size of the GRU was 256 nodes and the number of GRUs representing each time step in the recurrent input sequence was 30 GRUs; one GRU per time step. In addition, each unrolled sequence was layered 3 times. Therefore the unrolled 30-GRU-sequence long network was also 3-layers deep. Due to the multi-layered high-dimensional depth of this neural network, there was a tendency for the network to over fit the data, hence, a small learning rate of 0.001 was used. To further reduce the risk of over fitting the popular and effective dropout method for regularising deep neural networks kept at 80% of activations while deactivating the rest.

\subsection{LSTM Output Language Generation}
Once training of the neural network as described above is completed after several epochs or training cycles to an acceptable margin of error. It is possible to seed the network with an input character and the model samples from the top-N most likely candidates. We thus have instructed the language model developed to immanently construct its own sentences. The output language should therefore be intelligible similar to the training data.

In this work, the output language generated was used to generate a corpus that measured the perplexity and compared the result with the perplexity based on an n-gram model applied to the original training data. The results discussed below showed that the LSTM model generated a superior model compared to the n-gram model that better matched the training data.

Grapheme to phoneme model

Deep Learning Speech Models

Earlier in chapter one, deep learning was defined as a type of representational learning whereby different levels of complexity are captured in internal layer-wise encapsulations. It has also been noted that layer-wise stacking of neural and neural network type architectures such as the Restricted Boltzmann Machine (RBM) deep belief networks (DBMs) 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 explained.

Deep speech model

The core of the system is a bidirectional recurrent neural network (BRNN) trained to ingest speech spectrograms and generate English text transcriptions.

Let a single utterance $x$ and label $y$ be sampled from a training set $S = {(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), . . .}$. Each utterance, $x^{(i)}$ is a time-series of length $T^{(i)}$ where every time-slice is a vector of audio features, $x^{(i)}t$ where $t=1,\ldots,T^{(i)}$. We use spectrograms as our features; so $x^{(i)}{t,p}$ denotes the power of the $p$-th frequency bin in the audio frame at time $t$. The goal of our BRNN is to convert an input sequence $x$ into a sequence of character probabilities for the transcription $y$, with $\hat{y}_t

=\mathbb{P}(c_t \mid x)$, where $c_t \in {a,b,c, . . . , z, space, apostrophe, blank}$. (The significance of $blank$ will be explained below.)

Deep Scattering Layer

Log Mel filter banks with 23 frequency bins

Context window of +/- 10 frames concatenated to form a final vector size of 483

No speaker adaptation

Output classes 26 letters, 3 punctuation marks and blank ‘_’.

Model Architecture

Our BRNN model is composed of 5 layers of hidden units. For an input x, the hidden units at layer l are denoted $h^{(l)}$ with the convention that $h^{(0)}$ is the input. The first three layers are not recurrent. For the first layer, at each time t, the output depends on the spectrogram frame x_t along with a context of C frames on each side. (We typically use $C \in {5, 7, 9}$ for our experiments.) The remaining non-recurrent layers operate on independent data for each time step. Thus, for each time t, the first 3 layers are computed by:

$$h_t^{(l)} \; = \; g(W^{(l)} h_t^{(l-1)} \; + \; b^{(l)})$$

where $g(z) = \min{\max{0, z}, 20}$ is the clipped rectified-linear (ReLu) activation function and $W^{(l)}$, $b^{(l)}$ are the weight matrix and bias parameters for layer l. The fourth layer is a bidirectional recurrent layer[1]. This layer includes two sets of hidden units: a set with forward recurrence, $h^{(f)}$, and a set with backward recurrence $h^{(b)}$:

$$
\begin{aligned}
h_t^{(f)} &= g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)}) \\
h_t^{(b)} &= g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})
\end{aligned}
$$

Note that $h^{(f)}$ must be computed sequentially from $t = 1$ to $t = T^{(i)}$ for the i-th utterance, while the units $h^{(b)}$ must be computed sequentially in reverse from $t = T^{(i)}$ to $t = 1$.

The fifth (non-recurrent) layer takes both the forward and backward units as inputs

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

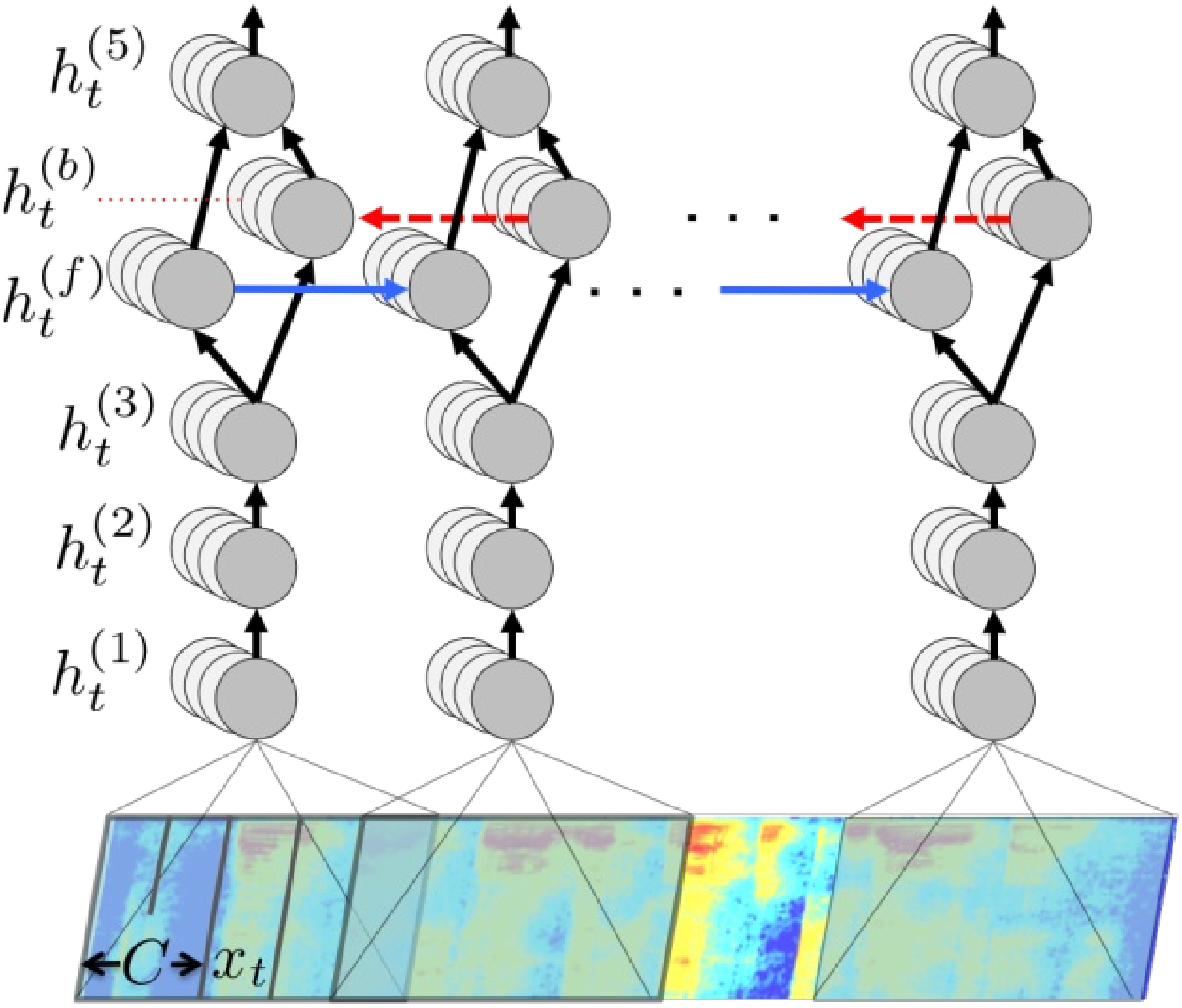
where $h^{(4)} = h^{(f)} + h^{(b)}$. The output layer is a standard softmax function that yields the predicted character probabilities for each time slice t and character k in the alphabet:

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

Here $b^{(6)}_k$ denotes the k-th bias and $(W^{(6)} h^{(5)}_t)_k$ the k-th element of the matrix product.

Once we have computed a prediction for $\mathbb{P}(c_t = k \mid x)$, we compute the CTC loss[2] $\cal{L}(\hat{y}, y)$ to measure the error in prediction. During training, we can evaluate the gradient $\nabla \cal{L}(\hat{y}, y)$ with respect to the network outputs given the ground-truth character sequence y. From this point, computing the gradient with respect to all of the model parameters may be done via back-propagation through the rest of the network. We use the Adam method for training[3].

The complete BRNN model is illustrated in the figure below.



5 hidden layers 1824 hidden units, 20.9M free parameters

Weights are initialised from a uniform random distribution scaled by the weight matrix input and output layer size (Glorot et al 2011)

Nesterov accelerated gradient optimisation algorithm as described in Sutskever et al. (2013) with initial learning rate 10-5, and maximum momentum 0.95.

After each full pass through the training set we divide the learning rate by 1.2 to ensure the overall learning rate decreases over time. We train the network for a total of 20 passes over the training set, which takes about 96 hours using our Python GPU implementation. For decoding with prefix search we use a beam size of 200 and cross-validate with a held-out set to find a good setting of the parameters α and β. Table 1 shows word and character error rates for multiple approaches to decoding with this trained BRDNN

CTC decoder

Assuming an input of length T, the output of the neural network will be $p(c; x_t)$ for $t = 1, \ldots, T$. Again $p(c; x_t)$ is a distribution over possible characters in alphabet Σ, which includes the blank symbol, given audio input x_t. 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 $S = (s_1, \ldots, s_T)$ be the character sequence where $s_t = arg\ \max_{c \in \Sigma} p(c; x_t)$. 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.
$p_{net}(W; X)p_{lm}(W)$ - - - (7)

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

• ◦ ▪ (8)

Algorithm 1 attempts to find a word string W which maximises equation 8. The algorithm maintains two separate probabilities for each prefix, $p_b(\ell; x_{1:t})$ and $p_{nb}(\ell; x_{1:t})$. Respectively, these are the probability of the prefix ending in blank or not ending in blank given the first t time steps of the audio input X.

Algorithm 1 Prefix Beam Search: The algorithm initializes the previous set of prefixes A_{prev} to the empty string. For each time step and every prefix ℓ currently in A_{prev}, we propose adding a character from the alphabet Σ 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 A_{next}. We

then set A_{prev} to include only the k most probable prefixes of A_{next}. The output is the 1 most probable transcript, although the this can easily be extended to return an n-best list.

The sets A_prev and A_next 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 A_prev is never larger than the beam width k. 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.

-
  -
    - (9)

      Where $W(\ell)$ is a set of words in the sequence . When taking the k most probable prefixes of A_next, we sort each prefix using the probability given in equation (9).

The variable ℓ_{end} is the last character in the label sequence ℓ. The function $W(\cdot)$, which converts ℓ into a string of words, segments the sequence ℓ at each space character and truncates any characters trailing the last space.

We incorporate a lexicon or language model constraint by including the probability $p(W(\ell^+)|W(\ell))$ whenever the algorithm proposes appending a space character to ℓ. By setting $p(W(\ell^+)|W(\ell))$ to 1 if the last word of $W(\ell^+)$ 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, $p(W(\ell^+)|W(\ell))$ can represent an n-gram language model by considering only the last n-1 words in $W(\ell)$.

Conclusion and Discussion of Results

Future Direction

Pidgin English models

OCR exploration

GAN exploration

Appendices

References

references:bib.bib

<!–[Highland-ScratchPad-Start]

Highland Scratchpad

Thesis notes

Chapter 1

- BiRNN - done
- GRU ref
- Scatnet references - done
- OOV ref?
- enhances - enhancement or advances
- emission, glossary beginning
- larger image - p18
- CNN are restricted DSS (Mallat, 2016)
- Deng2003 Language models
- Sample HMM-DNN system pipeline to show simplicity
- Deepspeech refs

Chapter 2

- discriminative AM models – done!
- Low resource Language models - attention models
- Low resource AM - RNN speech models
- check young 2008 for OOV term
- HMM problems expanded -> ch3 possibly.

Low resource speech recognition

- explanation of bottleneck features
- HMM recognition weakness

Low resource AM

- MLLR/MAP adaptation of SGMM models

Contribution to knowledge

1. BRNN simplifies processing
2. Scattering network increases feature representation for discrimination
3. Light weight CNN reduces training time

Methodology

1. Bleu ref

Chapter 3

Chapter 4

- Invariance introduction - done
- Fourier analysis - done
- Continuous wavelets - done
- wavelet types - done
- Multiresolution analysis - not done
- Fast wavelet transform - not done

Chapter 5

Chapter 6

- hannun2014deep training data
- hannun2014deep results

Chapter 7 conclusion/discussion

- hannun2014deep results/conclusion
- Various strategies for speech recog

  ** Kernel based methods
- h

Chapter 8 Future

- Various strategies for speech recog
- Kernel based methods

Questions

Chap 1

1. What is the aims and objectives
2. What is the contribution to knowledge

Chap 2

1. Why is Bleu not explained or used?
2. Why is ABX not explained

Chap 4

1. How does this chapter relate to your work?

2. Where is the Fast Scattering algorithm? Is it in a later chapter?

Other Questions

1. What about chapter summaries
2. Should equation references be big or small caps
3. To read and summarise \cite{maas2017building}
4. Publication rejection notes to be considered
5. Fast Fourier transform?
6. Chapter summaries

[Highland-ScratchPad-End]–>

<!–[Highland-Bin-Start]

[Highland-Bin-End]–>