SPOKEN LANGUAGE IDENTIFICATION SYSTEM IN MISMATCH DOMAIN

A Graduate Project Report submitted to Manipal Academy of Higher Education in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
In
Electronics and Communication Engineering

Submitted by

Shantanu Kapoor
Reg. No.: 160907430

Under the guidance of

Dr. Dileep A.D.
Associate Professor
School of Computing and Electrical Engg.
IIT Mandi

&

Dr. Aparna U.
Assistant Professor
Dept. of E&C.
MIT - Manipal

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

MANIPAL-56104, KARNATAKA, INDIA

MARCH/APRIL 2020

ACKNOWLEDGEMENTS

There is always a sense of gratitude which one expresses towards others for their help and supervision in achieving the goals. This formal piece of acknowledgement is an attempt to express the feeling of gratitude towards people who helped me in effectively completing my training.

It was a privilege to have a wonderful internship at IIT Mandi. I would like to express my heartiest gratitude to Dr Dileep A.D. and Mr Muralikrishna H, my training coordinators for their constant cooperation. They were always there for their competent guidance and valuable suggestions throughout the pursuance of this research project.

I would also like to thank the Director of MIT, Manipal, Dr D Srikanth Rao and HOD Professor Dr M Satish Kumar for providing the platform and approval of the project. Furthermore, I would like to thank my internal guide, Dr Aparna U for her guidance and inputs throughout this project

Above all, I would like to express my feelings to my parents and friends who supported me during the project. I am also thankful to all the respondents whose cooperation and support has helped me a lot in collecting necessary information.

ABSTRACT

Various state-of-the-art Spoken Language Identification Systems (LID) face problems on unseen test dataset whose domain specifications - usually channel and speaker properties - differ from the training dataset that has a limited channel and speaker diversification. This problem is considered as a domain-mismatch condition and is a widely researched topic in the Natural Language Processing (NLP) community.

In our project we aim to create a robust LID system that can produce domain invariant features, using a training database that contains extremely clean speech samples recorded from a small number of speakers. We propose to use two data-augmentation schemes followed by adversarial multi-task learning (AMTL) based training strategy to address the domain-mismatch condition. We use a new channel perturbation and state-of-the-art speed perturbation methods of data augmentation to increase the intra-class variance. Followed by this we use AMTL on the augmented dataset for encouraging the network to learn channel and speaker invariant representation of the input. The IIT-Madras-Indic-TTS dataset contains clean speech samples for training and IIT-Mandi dataset contains spontaneous speech from Youtube for testing.

The results obtained from our system demonstrates that combined channel and speech augmentation provided better generalization for the LID-system than individual data-augmented techniques. The AMTL based training also helped in inducing better domain invariance in IIT Mandi dataset while performing poorly on the validation set which is explained later.

Though results obtained are positive, they are far below the ideal condition. This portrays that hard challenges are to be faced in building robust LID systems and more data augmentation and training methods are to be explored for wide-scale use in real-time scenarios.

The following project is developed using python3 on the PyTorch framework. MATLAB software and the sox library were used for data-augmentation. RTX - 2080 and GTX 1080 GPUs were used for the training of the network.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Explanation
AMTL	Adversarial Multi-Task Learning
ANN	Artificial Neural Network
BPF	Bandpass Filters
BNF	Bottleneck Features
BPTT	Backpropagation Through Time
BLSTM	Bi-direction LSTM Layer
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DCT	Discrete Cosine Transform
DA	Domain Adaptation
FSL	Few Short Learning
FT	Fourier Transform
FCN	Fully Connected Network
GAN	Generative Adversarial Network
GRL	Gradient Reversal Layer
HMM	Hidden Markov Model
LID	Language Identification System
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
SIT	Speaker Invariant Training
TTS	Text-to-Speech

Contents

CHAPTER 1
INTRODUCTION

1.1 Introduction

Multiple companies are trying to develop multilingual smart assistants that can be used profusely at bilingual homes. The speech uttered during the interaction with such an assistant goes through a Language Identification System (LID). In this project, we propose a LID system that is robust to unseen test samples for low resource languages.

The ideal functioning of the LID system is usually challenged by the domain-mismatch conditions where the speaker, channel, background noise present in real-time or test samples differ from the training samples. This problem is usually addressed by training examples that traverse all probable environments. However, various state-of-the-art systems suffer from low-resource languages. In our project, we make use of a training dataset that contains only clean speech samples recorded for text-to-speech synthesis. This dataset contains few training examples and contains samples from only two native speakers (one male and one female). Due to limited channel and speaker diversity, the traditional state-of-the-art models struggle to give the same performance in an unseen real-world target domain. This project aims to investigate methods that address above-mentioned issues.

1.2 *Motivation*

Adversarial Multi-Task Learning (AMTL) is a Deep Neural Network (DNN) based system that motivates DNNs to learn domain-invariant features. It has flourished in accomplishing domain invariance in speech-related applications like speaker emotion recognition[1], speaker recognition/verification[2], speech recognition[3], etc. The AMTL method usually consists of 3 main components: a front-end feature extractor, a primary classifier and an auxiliary domain classifier for training samples. It uses training samples from multiple domains with corresponding domain labels which compels the network to learn features that are discriminative for primary tasks but invariant across domains. For example in [2] the domain classifier is trained with labels “target” and “source” to obtain invariant features across these domains. The performance of AMTL is highly dependent upon the contrast present in the training dataset.

Along with AMTL, Data-Augmentation is also a well-known strategy that generalises and increases the number of training samples by adding synthetic dataset obtained from modifying actual dataset. It is a regulariser that is implemented by applying some label-preserved transformations on original training dataset[4] or by augmenting unlabelled samples from unknown domains using semi-supervised methods[5]. The label preserved

transformations can be directly applied on the raw speech (such as speed perturbation[6]) or in the feature space of the training model while, for semi-supervised methods, first the unlabelled samples are passed through an existing classifier and then high-confidence samples used for data-augmentation. The efficiency of the data augmentation is best determined by how much its feature space simulates the test-dataset.

Motivated by the benefits provided by AMTL and data-augmentation, we aim to develop a training strategy that combines both for the robustness of a LID system. We first apply two data-augmentation techniques. Firstly, a novel channel perturbation for channel diversity and secondly, a state-of-the-art speed perturbation [6] for intra-class variability. Further, we apply the ATML technique on the augmented dataset to learn speaker and channel invariant features of the input dataset. Our proposed method demonstrates its robustness to unseen target domains as compared to methods applied individually.

The highlight of our work are as follows

- Using channel and speed perturbation to increase the diversity of the original dataset.
- Using AMTL to learn domain-invariant features.
- Experimentation with the proposed approach and comparison with the rest of state-of-the-art available models on low resource dataset.

Table 1.1: Project Work Schedule

July 2020	● Literature Review ● Data extraction and cleaning ● Extraction of Acoustic features ● Training and testing of Baseline LID system
August 2020	● Training and testing of Baseline LID system with dropout ● Data augmentation using channel perturbation, speed perturbation and SpecAugmentation and acoustic feature extraction ● Training and Testing of LID system using above augmentation methods (3 fold data) and combined channel and speed perturbation method (9 fold data)
September 2020	● Speaker Adversarial Training and testing of LID system on the original dataset ● Speaker and channel Adversarial Training and testing of LID system on the 3 fold channel perturbed dataset and on the 9 fold channel and speed perturbed dataset
October 2020	● Performance analysis of the above-mentioned methods ● Documentation

1.3 Organisation of the Report

In the next few chapters, literature review of DNNs, dropouts, LID systems using RNN, unsupervised domain adaptation, data-augmentation and state-of-the-art systems are discussed. Thereafter, working and methodology of our proposed method along with experimentation and analysis are discussed. It is followed by a conclusion and future scope of the work.

CHAPTER 2
Background Theory and Literature Review

In this chapter, we will discuss the methods, algorithms and features that have been used in the development of the LID systems. A brief introduction to NN, LSTM architecture and dropout mechanism. BNF feature extraction for basic phoneme unit representation and senone features used for classification are discussed. An auxiliary network for adversarial Speaker Invariant Training using gradient reversal layer is discussed for domain-independent features. Augmentation techniques like channel perturbation, speed perturbation and spec augmentation are discussed for the generalisation of the domain subspace. Later this chapter is concluded on the inferences made from the above methods

2.1 Artificial Neural Networks (ANN)

Deep learning is a grade of the machine learning algorithm, based on the Artificial Neural Network (ANN). These ANNs hold the resemblance to the function and structure of the biological neural network, wherein after a threshold criterion, an electrical impulse is passed from one neuron to another through synapses.

To imitate these neurons, the ANN has a perceptron, one-to-one or one-to-many mapping function, that maps the input to the output as shown in [7]. These maps are cyclic or acyclic. The ANNs consist of an input layer, a hidden layer and an output layer. Each layer is made up of a series of perceptrons. When the information is passed through these layers, it affects the structure of the ANNs by updating the connection so-called weights present between each perceptron to approximately map the given input and output. This updating of weights is called learning, which helps ANN memorize the data or important features required for mapping. These weights before training are sampled from a random distribution and updated after each training cycle through the Backpropagation. This technique adjusts the distribution of the weight by using the error present between input and output and optimizing them using gradient descent or stochastic gradient descent.

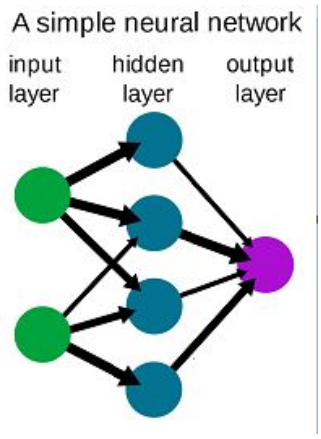


Fig2.1: Feedforward ANN

Perceptrons that comprise each ANN layer, take in vectors of real values, multiply it by a weight vector to calculate the linear combination. This linear combination approximates to 1 or -1 if it passes the threshold or not respectively. Relu, sigmoid,tanh, leaky-relu are such examples of the perceptron or non-linear activation functions. These nonlinear functions are preferred over simple linear combinations to effectively produce complex computational methods.

Let the input to the perceptron be a n-dimensional vector $X = [x_1,x_2,\ldots,x_n]$ and the weight be n-dimensional vector $W = [w_1,w_2,\ldots,w_n]$, with activation function tanh, the perceptron can be represented as $Y = \tanh(W^TX)$ where $W^TX = w_1x_1 + w_2x_2 + \ldots. w_nx_n$

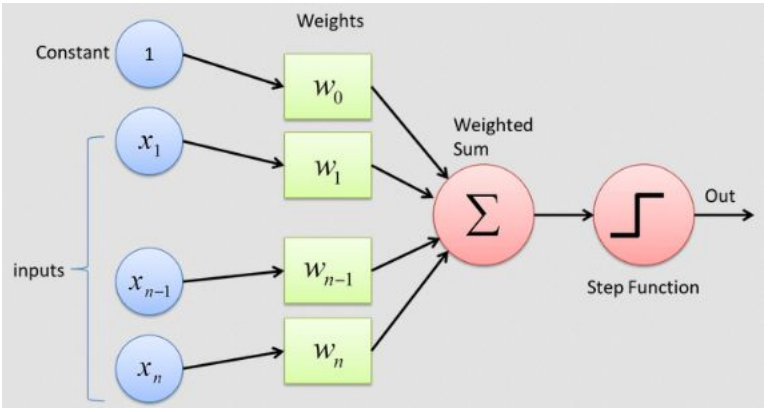


Fig2.2: Perceptron adapted from [7]

We will now discuss a special class of ANN namely Recurrent Neural Network that has been used in the model.

2.2 Recurrent Neural Networks(RNN)

RNN are a family of networks that are responsible for processing sequential data. It can scale to larger sequences than your ANN, this helps in the prediction of future states from the past experiences, but it also has a limit to its memory leading to prediction errors or bias.
RNN have variable length input and output sequences. It has a property of parameter sharing, which makes it possible to apply the model to examples of different length. In RNN same information can occur at multiple positions/index for eg:- "I was born in 1999"/"In 1999 I was born", here information of year is important. ANN would have learned the occurrence of the grammar at each position which would have led to extensive training and poor prediction.

2.2.1 Working and Structure of RNN

Activation in feedforward networks usually flows in one direction, RNN is similar to ANN with some connections pointing backwards. A simple RNN is a neuron which, on receiving input, produces an output that is sent back to the input as shown in [8]. This results in the sharing of parameters, which are understood using deep computational graphs. Acyclic computational graphs are used to represent this parameter sharing. For T time steps the graph will unfold t-1 times so for t =3 , $s^3 = f(s^2;\theta)=f(f(s^1);\theta);\theta)$

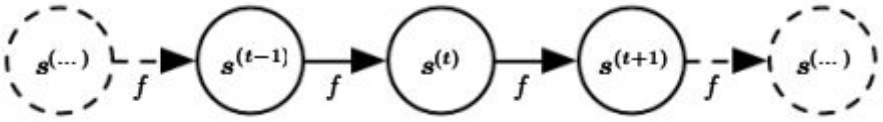


Fig2.3: Unfolding computational graphs. Source:Adapted from [8]

The units in RNN at time step t are usually described by state $h^t= f(h^{t-1},x^t;\theta)$, yielding output y^t. Training RNN results in h^t representing lossy information of up to t previous states by mapping x values of those states to a fixed-length vector h^t, thus calling them hidden states.

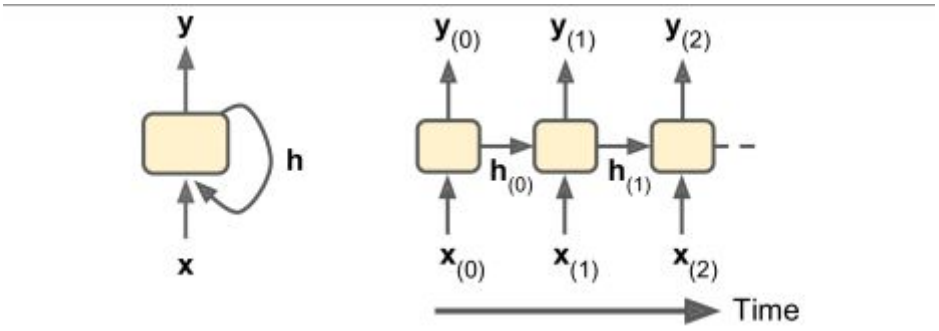


Fig2.4:Hidden RNN Source :adapted from [8]

Let us look into feedforward equations of RNN that produce an output ‘o’ of the given input x at each time step. The input to hidden connections are parametrized by weight matrix U and hidden recurrent to recurrent are parametrized by a Weight Matrix W and hidden to weight matrix by V. It begins by initializing the hidden state h^0. Then, for each time step from t = 0 to t = T, we apply the following equations:

$$a^t = b + Wh^{t-1} + Ux^t \text{ - (1)}$$

$$h^t=\tanh(a^t) \text{ - (2)}$$

$$y`=\text{softmax}(c + Vh^t) \text{ - (3)}$$

where b and c are bias vectors. Then loss usually cross-entropy is computed between the ground truth y and y`. The network is unrolled as mentioned above and moves in a backward direction. Gradients are computed by backpropagation through previous time-steps as shown in [8] and the weights are updated. This method is called backpropagation through time (BPTT) and takes O(T) time. Computing the gradient of the loss function is an expensive operation and cannot be reduced by parallelization as the hidden units are sequential.

2.2.2 Long-Term Dependencies

Despite being so effective in sequential data, RNN has major issues of vanishing or exploding gradients as gradients are propagated over time. Problems with long-term dependencies occur from the exponentially smaller weights given to long-term interactions. As weight matrices go through multiple multiplications while BPTT, after certain time steps these exponentially smaller weights approximate to 0, and when propagated furthermore the dependency on the past input is lost. Let’s take an example of sentimental analysis: a long review of the food bloggers which start with “ I like the food, the service and ambience but “ later explaining what else can be improved. The model might take the experience as negative if the dependency of the first few words is lost. Similarly exploding gradients can occur if the weights assigned are exponentially large. These problems are usually addressed using other recurrent models like LSTM and GRU.

2.3 Long Short Term Memory(LSTM) cell

One solution to long-term dependency is to have a model that can operate at both fine time steps for finer details and coarser time steps for the effective flow of distant information to present. LSTM cells[9] work in such a manner and are examples of Gated RNN . Gated RNNs create paths through a time where gradients don't vanish and explode. Gated RNNs like leaky units update their states through self linear-loop, wherein an assigned parameter decides to accumulate the previous state or a new state form the current input to amass information. Once the information is used, it is usually forgotten manually, which is learnt and automated by Gated RNNs. This is plausible by dynamically allocating weights to gated self-loops at each time step by conditioning weights on context.

2.3.1 LSTM Architecture

The LSTM cells have a 'conveyor belt' architecture where information is flowed from past states and is retained, merged or forgotten [8]. It consists of two major vectors h^t and c^t. The h^t is similar to a basic RNN cell and is generally referred to as a short-term state while c^t is called long-term state or 'cell' state. At each time step, while traversing through the network, firstly c^t is passed through the forget gate to drop some information and later via addition operation adds selected information by the input gate. Secondly, this long state information is passed through tanh and the output gate to filter the information, which later passed to the next state as h^t or current output of cell as y^t.

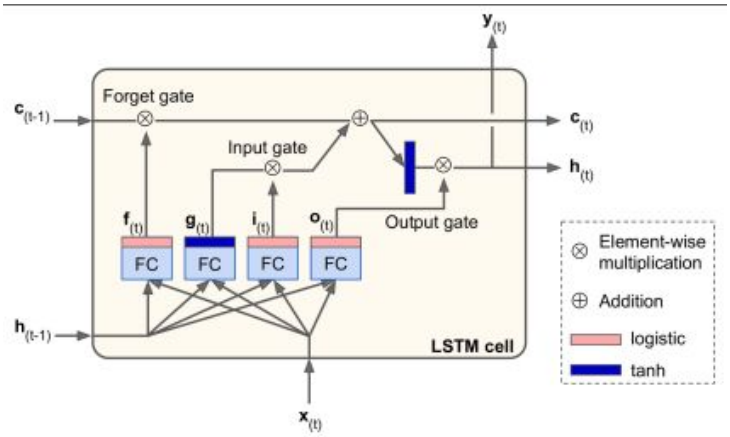


Fig 2.5: LSTM Cell. Source: Adapted from 8

Now let us look at the working of the gates. The input vector x^t along with previously hidden state h^t is fed to 4 distinct fully connected layer cells.

- The forget gate f_t controls which data from the cell state should be erased. Both input vectors x^t and h^{t-1} are multiplied by their respective weight matrices and biases are added. The sigmoid function is applied to the weighted matrix sum. The amount of information retained is highly likely if the output is 1, else it is forgotten. This vector is then multiplied to the cell state.
- Input gate i_t adds new information to the cell state. g_t analyses the input vector x_t and previous state h_{t-1} by passing the weighted matrix through tanh operation. This helps in squishing the values between -1 to 1 that are to be added to cell state. It is then multiplied with the input vector gate i_t - a sigmoid operation performed on two weighted input vectors, which determines what is to be kept from the tanh output. Their product is then added to cell state c^t via addition operation.
- Finally, the output gate o_t decides what part of the cell state c^t should be propagated to the output cell y_t or next hidden state h_t. The sigmoid operation of the weighted two input is multiplied with the tanh operation on the current cell state c_t to determine the output state.

The forward propagation equation in LSTM cells are as follows:

$$i_t = \sigma(W_{xi}^T.x_t + W_{hi}^T.h_{t-1} + b_i) \text{ -(4)}$$

$$f_t = \sigma(W_{xf}^T.x_t + W_{hf}^T.h_{t-1} + b_f) \text{ -(5)}$$

$$o_t = \sigma(W_{xo}^T.x_t + W_{ho}^T.h_{t-1} + b_o) \text{ -(6)}$$

$$g_t = \tanh(W_{xg}^T.x_t + W_{hg}^T.h_{t-1} + b_g) \text{ -(7)}$$

$$c_t = f_t * c_{t-1} + i_t * g_t \text{ - (8)}$$

$$y_t = h_t = o_t * \tanh(c_t) \text{ -(9)}$$

Where W_{xi}, W_{xo}, W_{xf}, W_{xg} are weight matrices for the input vector x.W_{hi}, W_{hf}, W_{ho}, W_{hg} are weight matrices for the previous state h_{t-1} and b_i, b_o, b_f, b_g are biases for the following layers.

2.4 Dropout

During backpropagation of typical NN architecture, a hidden state tailors to the shortcomings of the previous hidden states by adjusting accordingly to the derivatives of the previous hidden states for minimizing the loss function. This makes these hidden states highly co-dependent which leads to the overfitting of the data. To make these hidden units robust, few hidden units are temporarily masked for each training sample or time instant and are made to adjust with other random hidden units, forcing them to extract useful features without being codependent on other hidden units.

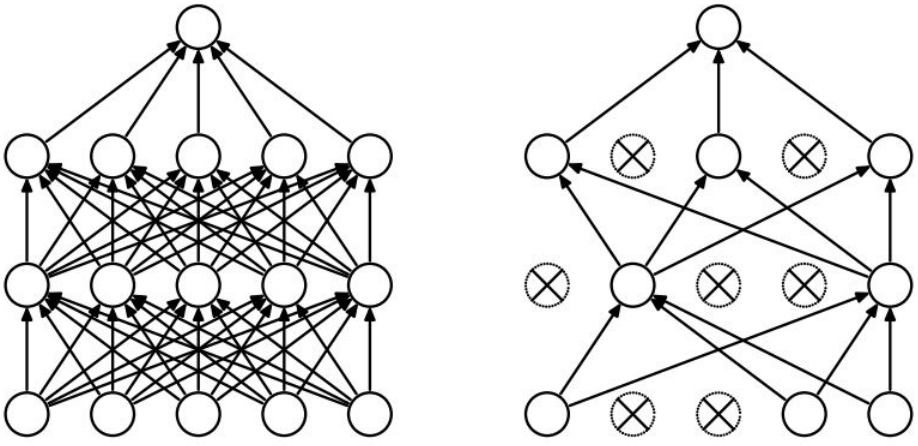


Fig 2.6: Dropout a) Standard Neural net with no dropout b) Standard Neural net with dropout Adapted from [10]

In other words, dropout by Srivastava et al [10], is a regularisation technique that masks random units temporarily in the NN architecture to avoid overfitting of data wherein a new dropout mask is sampled for every time step of RNN i.e input x_0 at t=0 receives a different binary dropout mask than input x_1 fed to the same RNN unit at t=1.

2.4.1 Dropout in a single hidden layer NN.

In this section, we explain the functioning of the dropout in a single hidden layer NN architecture that can be easily generalised to multiple hidden layers as explained in [10].Let input x be a K-dimensional vector to the NN that has an M dimensional hidden layer and output y an N-dimensional vector. W_1 (K X M), W_2 (M X N) are weight matrices connecting the input to hidden and hidden to output layer respectively. Let b be an M-dimensional bias vector for shifting the first linear transformation. The output vector $\hat{y} = \sigma(xW_1 + b)W_2$ where σ denotes element-wise non-linear transformation function.

Let t_1,t_2 be two binary vectors of dimension K and M respectively that are sampled to apply dropout. Each element in these binary vectors take values from the Bernoulli distribution with parameter $p_i \in [0,1]_{i=1,2}$, such that $t_{1,k}$~Bernoulli(p_1) for k = 1,..,K and $t_{2,m}$~Bernoulli(p_2) for m = 1,..,M. For input vector x, 1-p_1 amount of elements would be reduced to zero such that input to the first hidden layer is x o t_1 where o denotes Hamdard product. After this input passes through the hidden layer and t_2 proportions of elements are dropped out before going through the output layer. Thus the final output by the NN is $\hat{y} = (\sigma((x \text{ o } t)_1 W_1 + b) \text{o } t_2)W_2$. This equation can be simplified as

$$\hat{y} = (\sigma(x(t_1 W_1) + b)\, t_2 W_2$$

where t_1 and t_2 can be denoted as diagonal matrices to denote the binary vectors. During training, a new binary vector is sampled for each sample from the Bernoulli distribution that goes through forward pass and backward pass while none such sampling exists for testing. Each weight matrix is usually divided by p_i to maintain constant magnitude.

2.4.2 Dropout applied to RNNs.

Empirical results show that conventional dropout networks don't work effectively on RNNs as they do on feed-forward networks because recurrent networks down the signal and amplifies the noise for long-duration sequences as illustrated by Bayer et al[11]. However, a regularization technique introduced by Zaremba et al [12] is explained below on LSTM cells which seems to avoid overfitting and above-mentioned problems.

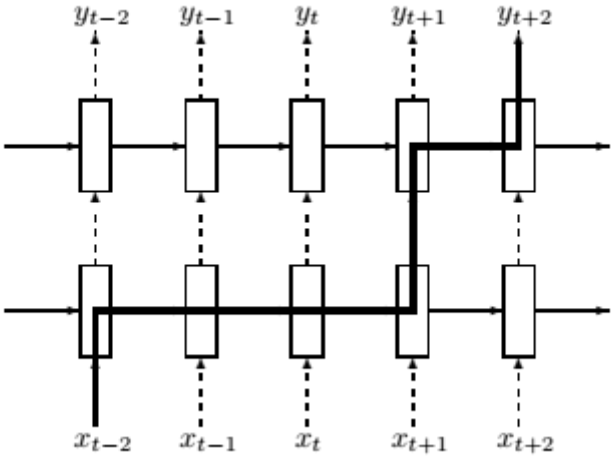


Fig 2.7: Dropout application in RNN. The thickest line shows the flow of information, while the dashed line indicates the non-recurrent path where the dropouts could be applied. [12]

According to him the dropout mechanism must be applied to the non-recurrent section of the LSTM/RNN cell to reduce overfitting. The modified LSTM equation w.r.t equations (4)-(9) with dropout operator D is as follows:

$$i_t = \sigma(W_{xi}^T.D(x_t) + W_{hi}^T.h_{t-1} + b_i) \text{ -(10)}$$
$$f_t = \sigma(W_{xf}^T.D(x_t) + W_{hf}^T.h_{t-1} + b_f) \text{ -(11)}$$
$$o_t = \sigma(W_{xo}^T.D(x_t) + W_{ho}^T.h_{t-1} + b_o) \text{ -(12)}$$
$$g_t = \tanh(W_{xg}^T.D(x_t) + W_{hg}^T.h_{t-1} + b_g) \text{ -(13)}$$
$$c_t = f_t * c_{t-1} + i_t * g_t \text{ - (14)}$$
$$y_t = h_t = o_t * \tanh(c_t) \text{ -(15)}$$

The dropout operator D is applied either to the input vector or output from the previous layer of the same timestep. Since it doesn't affect the timesteps, it prevents any loss of long-distance information. If there are L hidden levels to our LSTM then L+1 times our data would be corrupted. Thus by using non-recurrent connections we can reduce overfitting and provide a better regularisation to our LSTM cell.

2.4.3 Variational Dropout by Google Brain

It is a dropout mechanism that is inferred from the Bayesian interpretation of deep learning. Introduced by Gal and Ghahramani[13] in variational dropout only once a dropout mask is sampled which is repeatedly used for input, output or recurrent layers for all the time steps in forward or backward pass. We have applied the variational dropout technique only to the input and output layer in our project because of its superior performance. Due to complex mathematics involved in Bayesian statistics, one can refer to this paper for a better understanding of the concept.

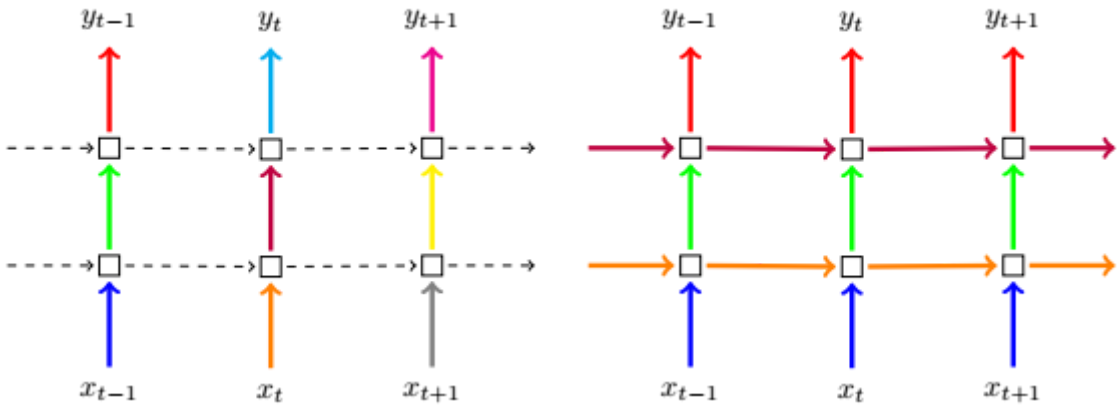


Fig 2.8: Variational Dropout. Source: Adapted from [13]

Figure on the left displays a native dropout wherein different masks are applied at different levels as indicated through multi-coloured arrows. Figure on the right shows Variational Dropout wherein the same mask is applied for the timesteps, recurrent layers or levels as indicated from same coloured arrows. Here each square displays a LSTM unit with horizontal arrows indicating timesteps and vertical arrows indicating input and output levels.

2.5 BottleNeck Features (BNF)

BUT Phonexia Bottleneck Feature extractor [14] is a NN where the hidden layer is of lower dimension than the surrounding layers- referred to as bottleneck layer. This bottleneck layer encodes the information in the input vector when propagated in the NN. The layer is the by-product of the NN i.e after training on a particular task (in our case phone state classification), the bottleneck layer is directed as an output layer and the features are extracted. The vectors extracted are usually robust to noise and overfitting. This NN is Multilingually trained to have language-independent phoneme representation. Though the BUT Phoneixa is initially trained on 3 separate language corpus, the pre-trained model on Babel-17 multilingual was of appropriate choice for our LID task.

2.5.1 Primary Feature Extraction

24 log Mel scale filter bank coefficients are extracted of 25ms frame at every 10ms of the speech. Each utterance passes through global mean normalization. The features are then concatenated with the context of 10 frames (± 5) to each log filter bank vector. Each individual feature in the vector is passed through the hamming window in the time trajectory and then projected onto 6 DCT bases(0th to 5th) which result in a 6 x 24 = 144-dimensional input vector to the NN.

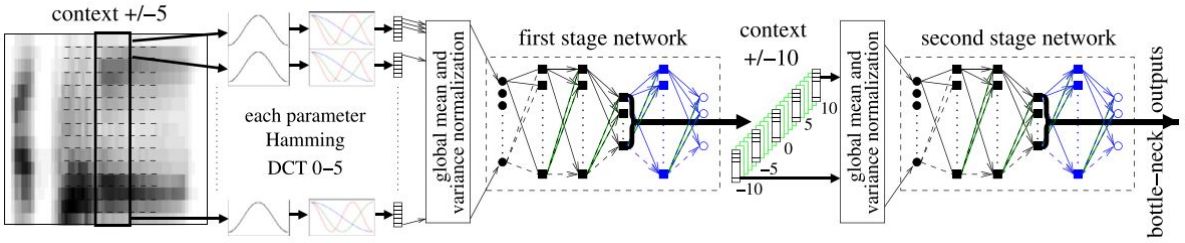


Fig 2.9: Block Diagram of SBN feature extractor. The blue part of the NN is for the training. Source : Adapted from [14]

2.5.2 Stacked Bottleneck (SBN) Architecture.

In SBN topology two standard bottleneck feature extractor are concatenated. The standard bottleneck network has an 80-dimensional bottleneck layer. The configuration of this network

is 144 x D_{HL} x D_{HL} x 80 x D_{HL} x D_{out}. where the hidden layer D_{HL} has 1500 perceptrons for Babel-17 multilingual corpus. All layers utilize sigmoid activation units. The linear bottleneck layer and block softmax are exceptions which are explained later.

The second network is similar to first except the input vector. It takes BNF from the first network that is sampled at t-10, t-5,t,t+5,t+10 to give a 400-dimensional feature vector, giving a context of 31 frames of the initial input. The bottleneck features of this network are used as final feature vectors. The target output of this network is a vector of phonemes of 17 languages stacked together [13].

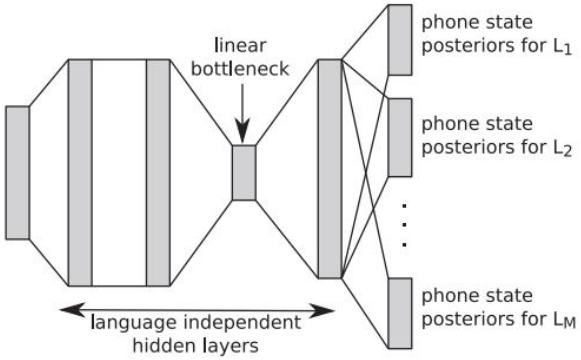


Fig 2.10: Block Softmax. Source: Adapted from [14]

The network goes through multilingual training where more than one language is fed into the bottleneck feature extractor so that the features can encode phonemes independent of their respective language. Here the block softmax activation has proved to be effective. In Block softmax, the output layer is divided into parts representing a collection of phoneme states according to the different languages. During the training, the output vectors that belong to the target language are activated and softmax is applied for the phoneme state classification.

2.6 Senones

Senones are defined as sub-phonetic Markov states of Hidden Markov Models (HMM) [15]. Since there are plenty of states in HMM, the state-dependent output distributions are clustered as senones. Clustering of states than models (tri-phone states), with acoustic similarity, leads to parameter sharing of senone states in different models that ease the training of multiple phoneme models.

2.6.1 LID-Senones

Though BNF or senones are good acoustic feature representations of a particular language, they are trained with phonetic labels rather than language labels. This leads to the poor classification of language with similar phonetic content for example Indian Languages, as

they are encoded with similar acoustic features. To overcome this problem using an end-to-end system, a DNN-CNN model was introduced by Jin et al[16]. This model used BN features and passed it through a stack of convolutional-pooling layers to form language discriminative units namely LID-senones.

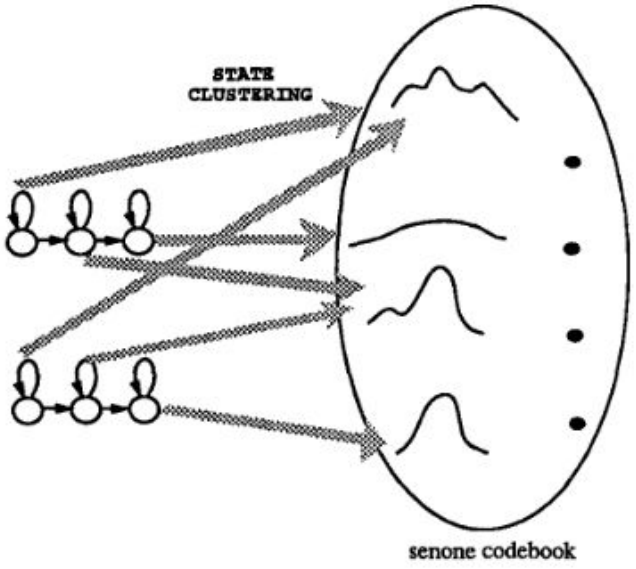


Fig 2.11:Clustering of states- Senones and Formation of senone notebook.
Source adapted from [15]

These units were later classified after being averaged over a context time using spatial pyramid pooling to form utterance level representation form frame-level features. They also termed the model as LID-net.

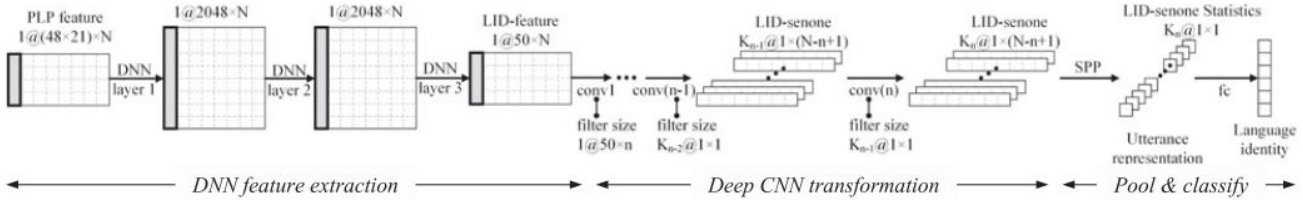


Fig 2.12:End-to-end LID-net system. Source : Adapted from [16]

2.6.2 LID-seq-senones

Taking inspiration from the LID-senones, an end-to-end network with 2 BLSTM was used [17]. The model was trained on language-specific labels using the BNFs as input. The output obtained at the last BLSTM layer was used as the utterance level representation of the speech. These representations covered a total context of 31+N-1 frames (wherein each BNF covered a context of 31 frames and N is variable due to variable speech length) which allowed the BLSTM layer to capture locally available language-specific cues and capture temporal

variations in BNF to discriminate between the languages. Hence these representations were termed LID-sequential-senones or LID-seq-senones.

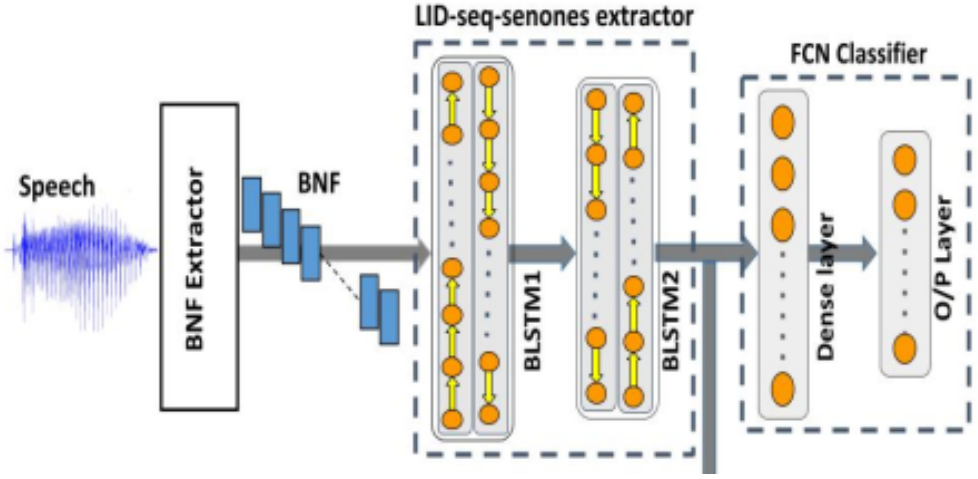


Fig 2.13: Lid-seq-senones extraction with classification layer.
Source: Adapted from [17]

2.7 Attention Mechanism

The attention mechanism[18] is a method in which input features that have high chances of deducing the output are given more weightage using the attention weight vector and their weighted sum of values by attention vector is used to infer the target vector like the pixel of an image or next phoneme in our case. It is similar to bokeh effect in photography wherein the object at focus is given more weightage to determine the object characteristics than the blurred background or how we expect "vegetable/fruit" to be more correlated in a sentence to the word "food" than colour like "green/yellow".

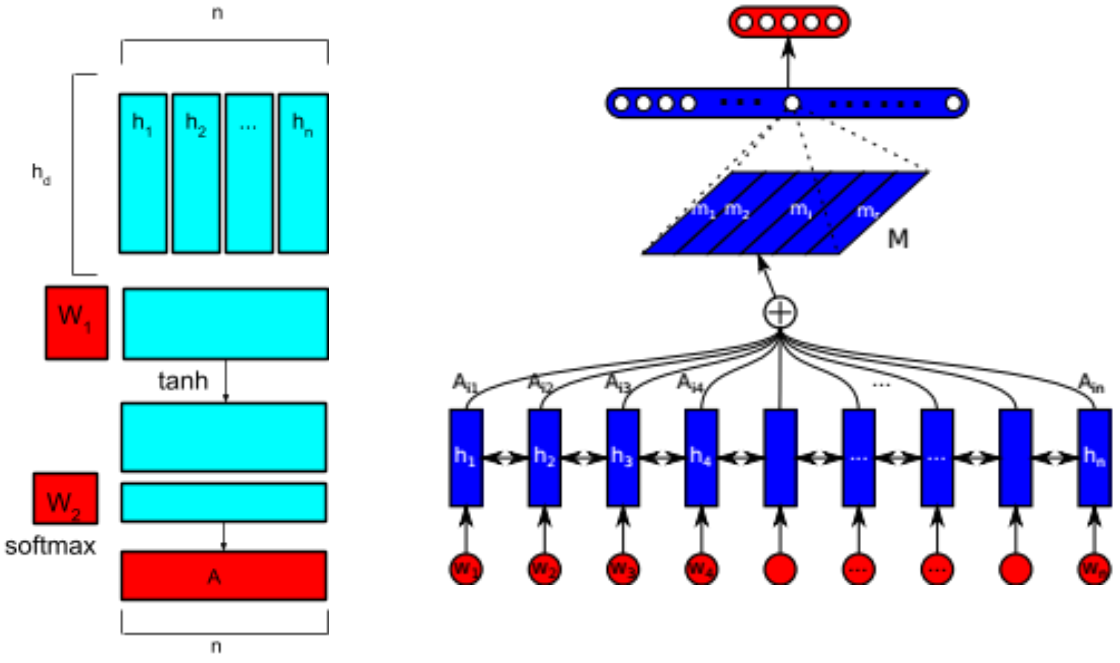
2.7.1 Self-Attentive Word Embeddings.

For extracting word embeddings in DNN based ASR generally a frame-level feature extractor is followed by a pooling layer that aggregates the last hidden state of the RNN unit of each frame-level feature to represent utterance level vector [17]. With this pooling layer, fixed-length representation can be obtained from variable length speech utterances. However, a huge assumption is made that each frame is of equal importance. Usually, after Voice Activity Detector (VAD) a few non-speech or silent frames are still left which discriminates little to none among the extracted phonemes or LID-seq-senones. To prevent such problems self-attention can be used to replace the average pooling layer.

Self-attention is an attention mechanism wherein input features interact among themselves, also called intra-attention, to find the attention weight vector for presuming the desired result.

It derives distinctive features from each frame-level features for the utterance level representation and with direct access to the hidden state it reduces the burden of long-term memorization as shown by Lin et al [19].

Let us represent the output vector H = $\{h_1,h_2,..,h_N\}$ where N is the total number of phonemes in a sentence and h_n represent the final hidden state of each phone frame of dimension d_h. Thus the size of H is d_h x N. The attention mechanism takes the whole input vector H and outputs an annotation vector A = softmax(tanh(H^TW_1)W_2). Here W_1 and W_2 are a weight matrix of size d_h x d_r and d_r x d_a respectively where d_a is a hyperparameter that represents the number of attention heads. It is usually set to 1 in most cases.



a) b)

Fig 2.14: Self-attention mechanism. a) Blue colour stands for hidden weights while the red colour stands for input and output vectors. b) Shows how annotation weighted vector A is multiplied with the last hidden states vector of each phoneme frame H to represent utterance level representation.

The annotation vector A is of size N x d_a that represents the different weight for each h_n. The major attribute of A vector is to focus on a special set of semantics in the spoken sentence. To be able to find multiple semantics in the sentence that generally occurs in long-duration segments, multiple perceptrons or attention layers are added to focus on contrasting aspects of the segment. The softmax at the final layer ensures all the weights are normalised. To represent the final utterance level representation E the attention weight matrix A is multiplied to hidden state vector H to give E = HA of h_d x h_a dimensional vector.

2.8 Deep Domain Adaptation using Gradient Reversal Layer

Deep Feedforward Networks have exhibited state-of-the-art results for problems with huge amounts of labelled dataset. For the inadequately labelled dataset, synthetic data is integrated to drive these networks. However, due to the deviation between source and real-time test dataset they suffer to operate well. This problem of learning a classifier that is invariant to deviation in data distributions is called Domain Adaptation (DA). DA aims to create mappings from the supervised source domain to unsupervised target domain so that source trained classifiers can work effectively on the target domain.

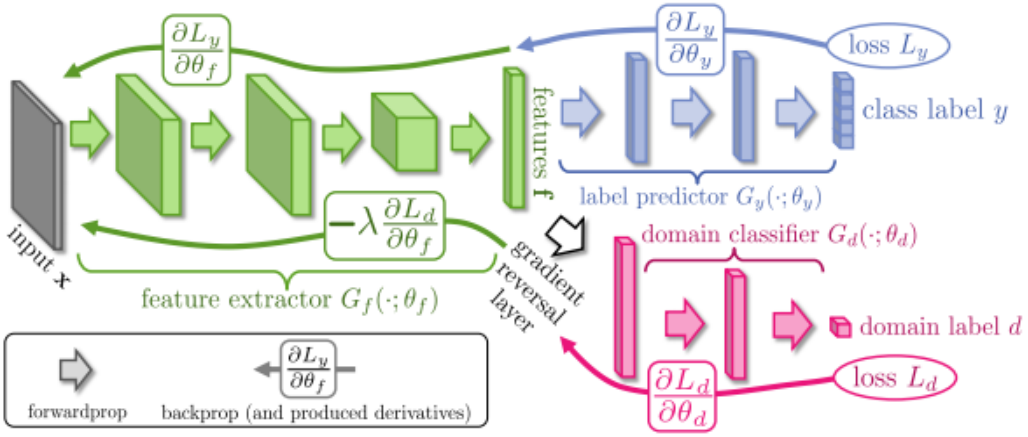


Fig 2.15: Deep Domain Adaptation architecture. Green colour network indicates the feature extractor, blue colour network indicate the discriminative classifier. Both combined represent a standard feed-forward architecture. The red color network is a domain classifier that is joined through GRL with the feature network. This GRL ensures that the features extracted are domain-invariant. Source: Adapted from [20]

While usual DA methods use fixed feature representations followed by domain transformations, Deep Domain Adaptation [20] tries to encapsulate feature representation and domain adaptation into one such that the features learnt are domain invariant and discriminative. All this is brought about by jointly optimising the feature extractor with discriminative classifiers. The primary feature classifier and an auxiliary domain label classifier are optimised to minimize the empirical loss while the parameters of feature extractors are optimised to minimize the feature discriminative classifier and maximize the loss of domain classifier that the features extracted are discriminative and domain-invariant. While the whole network can be trained using simple backpropagation, the auxiliary network is trained using Gradient Reversal Layer(GRL) [20] wherein the input is constant during

forward propagation and a negative scalar is multiplied to the gradient for backpropagation. The negative scalar λ determines the strength of the domain-classifier on the standard empirical loss. The training of this method is explained below using Speaker Invariant Training.

2.9 Speaker Invariant Training via adversarial learning

Though the DNN based acoustic models have achieved exceptional results for phoneme classification, they have failed to reproduce such results for a dataset containing diversified speakers. This is due to the high inter-speaker variability introduced into senones that makes this speaker-independent phoneme classification an arduous task. Looking into the recent success of adversarial learning in reducing domain-interference, The Speaker invariant training (SIT) by Meng et al [21], is an unsupervised training method that addresses inter-speaker variability using Adversarial learning with GRL.

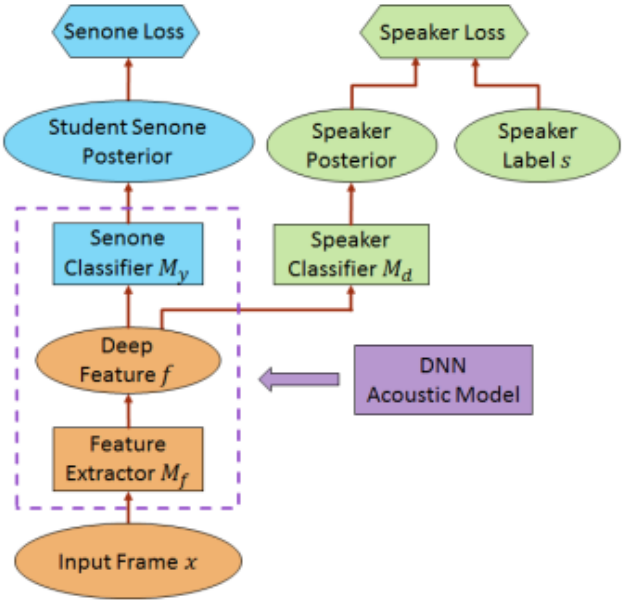


Fig 2.16: Framework for Speaker Invariant training via Adversarial Learning.
Source: Adapted from [21]

The SIT consists of DNN based senone classifier and an auxiliary speaker classifier that are collectively optimised with the primary task of minimizing the senone classification loss and the secondary task of mini-maximizing the speaker classification loss. Through adversarial training, this network learns senone discriminative and speaker-invariant features that are later classified using a fully connected network (FCN).

2.9.1 Working

Let $X = \{x_1, x_2, .., x_N\}$ denote the input speech frame vector, $Y = \{y_1, y_2, .., y_N\}$ denote the output senone units, $S = \{s_1, s_2, .., s_N\}$ denote the respective speaker labels. Let the feature extractor network that maps input vector X to feature vector $F = \{f_1, f_2, .., f_N\}$ be denoted by M_f with parameters θ_f. The N denotes the total number of speech frames. The senone discriminative network parameterised by θ_y be M_y that maps feature vector F to senone posteriori probability $p(q|f;\theta_y)$, where $q \varepsilon Q$ denoting a senone in senone subspace.

$$M_y(f_i) = M_y(M_f(x_i)) = p_y(q|x_i;\theta_f,\theta_y) \text{ - (16)}$$

The speaker classifier network be M_s with parameters θ_s that maps F to speaker posterior probability $p_s(a|x_i;\theta_f,\theta_s)$ where $a \varepsilon A$ denoting a speaker in the speaker subspace.

$$M_s(f_i) = M_s(M_f(x_i)) = p_s(a|x_i;\theta_f,\theta_s) \text{ - (17)}$$

For the features to be senone discriminative the loss of senone prediction is minimized by

$$L_{senone}(\theta_f,\theta_y) = -\sum_{i}^{N} p_y(y_i|x_i;\theta_f,\theta_y)\, M_y(M_f(x_i)) \text{ - (18)}$$

For the feature F to be independent of speaker intra-variation the M_y and M_s are adversarially trained such that θ_f is minimized for senone discriminative loss for while θ_f is maximized for speaker classification loss to generate features that confuse M_s, Thus the features produces are senone discriminative and speaker-invariant.

$$L_{speaker}(\theta_f,\theta_s) = \sum_{i}^{N} p_s(s_i|x_i;\theta_f,\theta_s) = \sum_{i}^{N} \sum_{a \varepsilon A}^{N} 1[a = s_i]\log(M_s(M_f(x_i))) \text{ - (19)}$$

The total loss function of this adversarial training is denoted by

$$L_{total}(\theta_f,\theta_s,\theta_y) = L_{senone}(\theta_f,\theta_y) - \lambda L_{speaker}(\theta_f,\theta_s) \text{ - (20)}$$

where λ act as a trade-off between senone classification and speaker classification. The parameters. The Gradient Reversal Layer is added between the speaker classification layer and feature extraction layer. In the forward propagation the input feature F passes through the speaker classification network as constant while for backward propagation the derivative of the speaker classification w.r.t the feature classification gets multiplied by the negative scalar λ. This function can be denoted as

$$R_\lambda(x_i) = x_i \text{ - (21)}$$
$$\frac{dR\lambda}{dx} = -\lambda I \text{ -(22)}$$

where I is the identity matrix. The function that is minimized using stochastic gradient descent is represented as

$$L_{total}(\theta_f,\theta_s,\theta_y) = -\sum_{i}^{N} p_y(y_i|x_i;\theta_f,\theta_y)\, M_y(M_f(x_i)) - \sum_{i}^{N}\sum_{a\,\varepsilon\,A}^{N} 1[a = s_i]\log(M_s(R_\lambda(M_f(x_i)))) \quad (23)$$

2.10 Data Augmentation

Data Augmentation is a plan of action used for regularisation and to avoid overfitting of the dataset by increasing the quantity of it, thus generalising it to large domain subspace and making it robust to the unseen domain. Speed perturbation, channel perturbation and spec augmentation are techniques that we have used to increase the robustness of our LID-system. While speed and channel perturbation are directly applied to the raw signal, spec augmentation by google brain is applied on the filter bank coefficients.

2.10.1 Speed Perturbation

Speed perturbation [6] is a label-preserving transformation that produces a warped time signal. Given a signal x(t), a warped signal x(αt) is produced by speed perturbation, where α is a speed factor. The Fourier Transform (FT) of x(αt) , $\alpha^{-1}\dot{x}(\alpha^{-1}w)$, produces a shifted version of ẋ(w), which is also visible in the mel spectrogram. Speed perturbation also alters the length and number of frames of the original signal, thus producing more variety within the dataset. This method can be easily incorporated owing to low computational cost using a cross-platform audio processing library called SOX.

2.10.2 Channel Perturbation

Motivated by the theory that multiple microphone setups have different frequency responses, an audio signal can be heard differently if recorded through a different microphone setup. Channel perturbation uses this method to increase the channel diversity of the dataset which contains recordings from a single microphone setup. It augments the data by passing audio through a set of bandpass filters (BPF) of different bandpass frequencies to emulate dissimilar microphone setup as shown in the fig below.

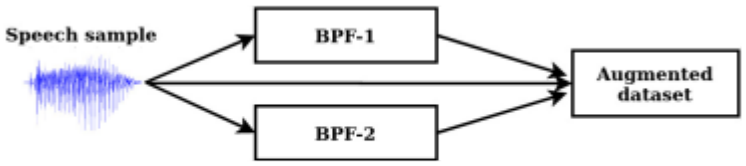


Fig 2.17: Channel Perturbation

2.10.3 Specaugment by Google brain

Specaugment by Daniel et al [22] , is an augmentation technique that is directly applied on the mel-spectrogram of the audio signal rather than the raw audio itself. This method is computationally cheap as it treats the spectrogram as images. First augmentation technique is time warping, that squeezes or stretches the data by shifting the spectrogram using interpolation techniques. Secondly are time and frequency masking , inspired from cutout augmentation from computer vision, that puts a mask on a time block or set of Mel frequencies and replaces it with the mean value of the spectrogram or zero. Figure below shows the above augmentation technique, where the vertical and horizontal axis represents Mel-frequencies and time steps of the spectrogram.

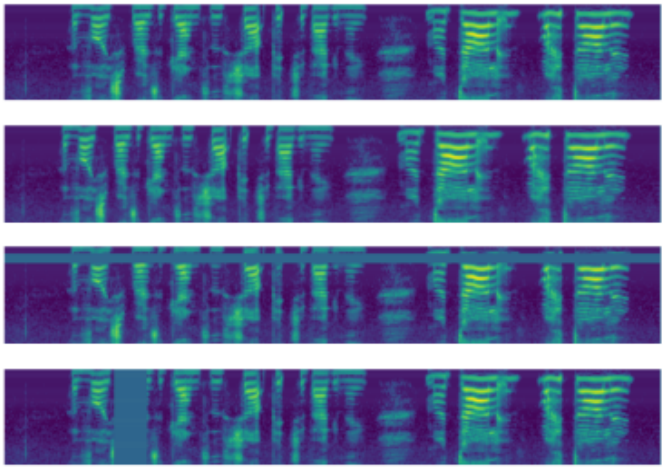


Fig 2.18: Specaugment. Specaugment is applied to the topmost spectrogram. Figures below represent the Mel spectrogram of time warping, frequency masking and time masking respectively .

After understanding methods in acoustic modelling using RNNs, Adversarial training and data augmentation, the first time all three techniques are combined to evaluate their performance on low resource languages with transcribed clean speech signals. We show that our proposed method outperforms, when compared to methods used individually and increases the robustness of the LID systems to an unseen target domain. This method addresses the problems of the limited channel and speaker diversity and domain adaptation when used with augmented synthetic data. The end-to-end pipeline of the proposed method is reviewed below in the Methodology chapter.

CHAPTER 3
Methodology

The block diagram of the proposed method for the domain-invariant LID system is given in below fig 20. Firstly, raw speech samples are augmented using the channel and speed perturbation. This augmented dataset combined with the original dataset is used to train our AMTL based LID system. Secondly, our DNN based feature extractor parameterised by θ_f determines the utterance level representation (u-vector) of the given speech sample. Thirdly, u-vector is classified by multiple FCN, a primarily language classifier parameterized by θ_l, and two auxiliary channel and speaker classifiers parameterized by θ_c and θ_s. Our LID system with parameters $\theta_{net} = \{\theta_f, \theta_l, \theta_c, \theta_s\}$ is then optimised by adversarially training speaker classifier and feature extractor, likewise channel classifier and feature extractor to obtain u-vector embeddings that are speaker and channel invariant.

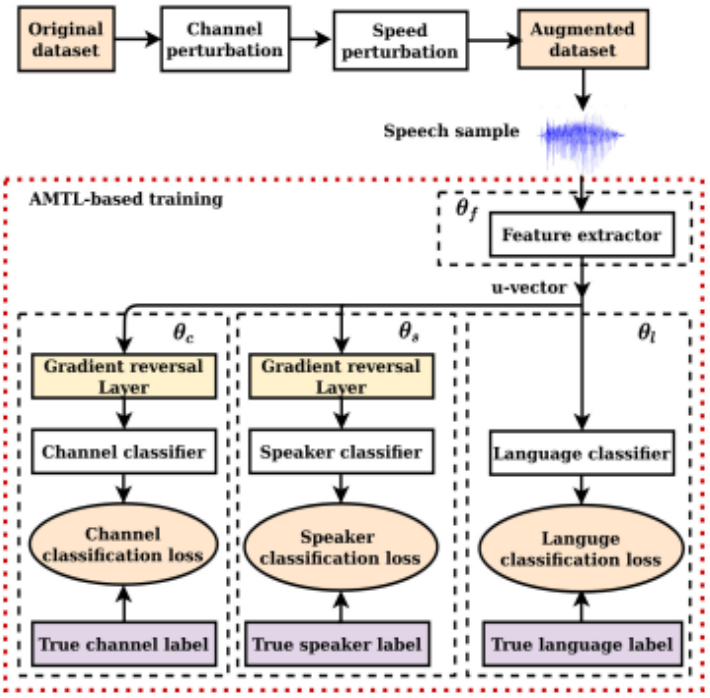


Fig 3.1: Block diagram of proposed AMTL based LID-system

3.1 Dataset Used

The functioning of any NN architecture is highly analogous to the quality of the data used to train it. It is because any algorithm used is made to learn inferences from the dataset itself to make future predictions or labels. To train our DNN architecture, speech samples from IIT-Madras-Indic TTS dataset were used. This dataset has speech samples sampled at 48Khz, recorded by a high-quality single microphone [23], in an echo and noise-free environment.

For each language, two speakers were selected (one male and one female) and they were made to read out a particular script. This script was initially used for Text-to-speech translation and therefore did not contain any code-mixing (intermixing of two languages). With 7 hours of the dataset for each language and little to no channel and speaker variation, it mimics the low resource environment. The 7 hours of the dataset is also divided into 6 hours of training and 1 hour of validation dataset, which also act as in-domain test data.

The IIT-Mandi Indian language dataset was used to test our LID system. Each language has a set of 6 speakers (3 male and 3 female) and their recordings are taken from educational or news channels from youtube. Each recording contains a different microphone set up and background noise which makes this a highly channel diverse dataset. It also contains high pitch variation which makes it a challenge for extracting similar LID-seq-senones. This dataset represents the target or the unseen dataset for domain adaptation. Though Audacity was used to extract only language-specific dataset does contain few intra-samples from the English language.

The recordings were segregated into 8 to 15 seconds speech samples and were downsampled to 8Khz for training and testing of the LID system. The list of languages, their total hours and number of speakers for training (including validation) and testing are given in Table 3.1

Table 3.1: Indian languages, number of speakers and duration in hours

Language	IITM-TTS dataset (Train + Validation)		IIT-Mandi Dataset (Unseen Test Dataset)	
	Hours	Speakers	Hours	Speakers
Assamese	7.21	2	0.5	6
Bengali	7.02	2	0.5	6
Gujarathi	7.10	2	0.5	6
Hindi	7.02	2	0.5	6
Kannada	7.20	2	0.5	6
Malayalam	7.12	2	0.5	6
Manipuri	7.04	2	0.5	6
Odia	7.10	2	0.5	6
Telugu	7.14	2	0.5	6

3.2 Data Augmentation using the channel and speed perturbation

Our primitive low resource dataset though recorded through excellent microphones has very little channel diversity. To tackle this problem we proposed channel augmentation. It works on the principle that different microphones are selectively sensitive to different frequencies which make the audio hear differently. To achieve this we pass the raw signal through the set of two band-pass filters which mimics two microphones. Combining this augmented data with original data leads to a 3 fold increase in the dataset.

The frequency responses of these filters are selected such that each augmented audio is heard differently from the primitive one. In our case, we have chosen passbands of 100 to 2500 Hz and 500 to 3500 Hz for BPF-1 and BPF-2 respectively. The spectrogram below shows the augmented audio B) for BPF-1 and C) for BPF-2 of original audio A). It is visible below how BPF-2 has suppressed even the pitch frequency from the original sample.

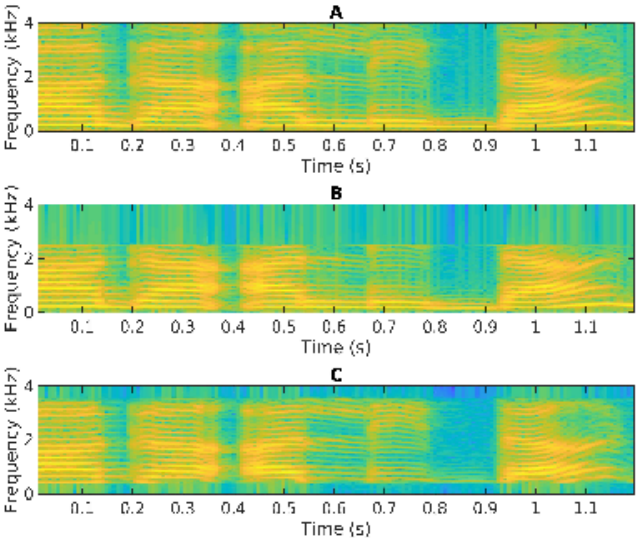


Fig 3.2: Spectrogram of original and channel perturbed signals

To add further variation to the dataset, we apply speed perturbation to the channel perturbed dataset. As explained in chapter 2 speed perturbation produces the augmented data by resampling the signal which causes a shift in the spectrogram. This method brings variation in length, either squeeze or elongate the signal that causes a shift in FFT energy bins. The speed is modified to 90% and 110% of the original sampling rate as suggested in [paper number] to the primitive data. In Fig 21, we can see the speed perturbed signals. A) denotes the signal with 90% speed of the original signal denoted by B), where the energy is shifted to the lower FFT bins and C) denotes the signal with 110% speed, where the energy is shifted to higher FFT bins.

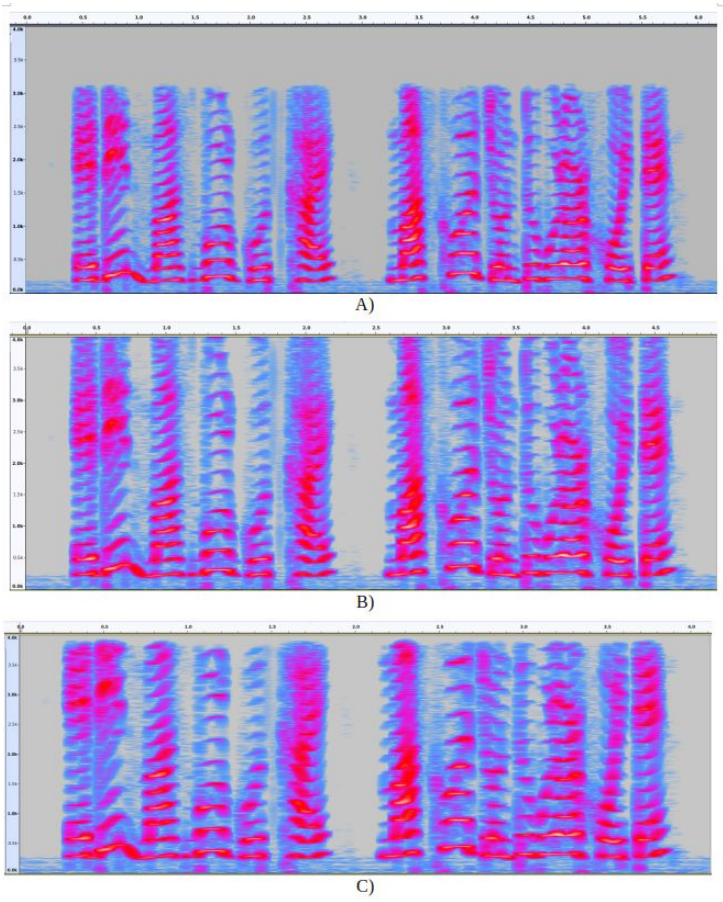


Fig 3.3: Spectrogram of original and speed perturbed signals.

When speed perturbation is applied to the channel perturbed dataset, there is a total increase of 9 fold from the original dataset. This augmented dataset is then used to train our DNN based LID system. We did an experiment of applying speed perturbation first and then channel perturbation but it didn't give any increase in performance. While The channel perturbation is implemented using MATLAB, speed perturbation is implemented using sox sound-processing toolkit.

3.3 Front-end feature extractor (θ_f)

The front-end feature extractor is composed of BUT phoenixa bottleneck feature extractor (BNF), two bi-directional LSTM layer (BLSTM) and a self-attention layer to represent the utterance level embeddings or u-vector. The BNF extractor which is pre-trained on 17-languages for identifying 3096 phonemes, converts the speech segments into a variational length sequence of 80-dimensional BNFs. Each BNF has a background of 31 frames or 325 ms of the input speech segment. It is assumed that the BNF extractor covers phonemes of all the languages that are present in our dataset.

These BNFs are converted into a batch of units, wherein each unit is a chunk of fixed N BNF frames. This batch is then passed through two BLSTM layers to obtain regional language cues and temporal variations for language discriminative features. The activation of an input unit obtained at the last hidden of the ultimate BLSTM layer is called LID-seq-senones. The LID-seq-senones covers the 31+N-1 context of the input speech segment. The LID-seq-senones are independent units like senones and are collectively used to represent the speech segment.

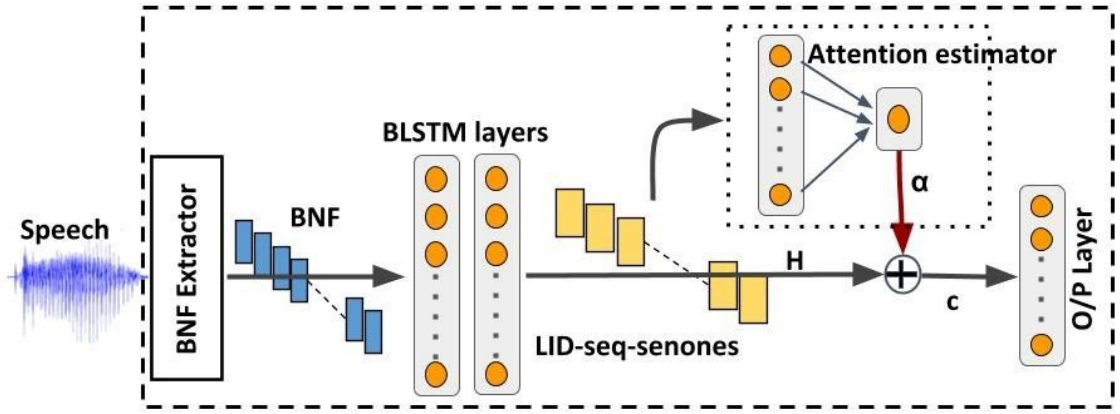


Fig 3.4: Front-end feature extractor for obtaining u-vector denoted by letter c.

To have a fixed-length utterance representation, all the LID-seq-senones of the speech utterance are passed through a self-attention layer. These LID-seq-senones interact with each other to find the weight factor and then use it to calculate the weighted mean of these LID-seq-senones to obtain the final u-vector. This u-vector though being language discriminative contains domain-specific information like that of channel and speaker which makes it hard for the network to perform for an unseen domain. To make the network robust to such intricacies we propose to use AMTL which is explained in the below section.

3.4 AMTL for domain-invariant features

As in Fig 20, the AMTL based LID-system contains three classifiers. Every classifier's output layer is present with a softmax layer which helps in estimating the posterior probability of the class outputs for a given input x.

Let the posterior probability of the outputs of the language classifier, speaker classifier and channel classifier be expressed as $P(y_l|x;\theta_f,\theta_l)$, $P(y_s|x;\theta_f,\theta_s)$ and $P(y_c|x;\theta_f,\theta_c)$ respectively, where $y_l \varepsilon \{l_1,l_2,..,l_{Nl}\}$, $y_s \varepsilon \{s_1,s_2,..,s_{Ns}\}$ and $y_c \varepsilon \{c_1,c_2,..,c_{Nc}\}$ denote the language,speaker and channel labels and Nl, Ns and Nc denotes the total number of language, speaker and channel labels. With K training examples the cross-entropy loss of each classifier is expressed below as L_l, L_s and L_c respectively

$$L_L = -\sum_{i}^{K} \log(P_l(y_l^i|x^i;\theta_f,\theta_l) \text{ - (24)}$$

$$L_s = -\sum_{i}^{K} \log(P_s(y_s^i|x^i;\theta_f,\theta_s) \text{ - (25)}$$

$$L_c = -\sum_{i}^{K} \log(P_c(y_c^i|x^i;\theta_f,\theta_s) \text{ - (26)}$$

The total loss L_t for adversarial training of the model is given as

$$L_t(\theta_f,\theta_l,\theta_c,\theta_s) = L_l(\theta_f,\theta_l) - \lambda_l L_s(\theta_f,\theta_s) - \lambda_l L_c(\theta_f,\theta_c) \text{ - (27)}$$

Here λ_1 and λ_2 denote the hyper-parameters that decide the trade-off made for the speaker and channel classification with the language classification. The parameters $\theta_{net} = \{\theta_f,\theta_l,\theta_c,\theta_s\}$ are jointly optimised to train the network. While $\theta_l,\theta_c,\theta_s$ are optimised to minimize their label loss, the θ_f is optimized to minimize the language classification loss and maximize the speaker and channel classification loss. These parameters are trained using the stochastic gradient descent as mentioned below.

$$\theta_f^{'} = \theta_f - \mu \left(\frac{\partial Ll}{\partial \theta f} - \lambda_1 \frac{\partial Ls}{\partial \theta f} - \lambda_1 \frac{\partial Lc}{\partial \theta f} \right) \text{ - (28)}$$

$$\theta_l^{'} = \theta_l - \mu \frac{\partial Ll}{\partial \theta l} \text{ - (29)}$$

$$\theta_s^{'} = \theta_s - \mu \frac{\partial Ls}{\partial \theta s} \text{ - (30)}$$

$$\theta_c^{'} = \theta_c - \mu \frac{\partial Lc}{\partial \theta c} \text{ - (31)}$$

The $\theta_f^{'},\theta_l^{'},\theta_c^{'}$ and $\theta_s^{'}$ are the updated parameters of the network after the i^{th} training example with μ as the learning rate. The gradient reversal layer is added between feature extractor and speaker classifier as well as between feature extractor and channel classifier for the adversarial training.

With this AMTL training of the LID-seq-senones, when the parameters will reach the saddle point, the utterance level embedding obtained should be speaker and channel invariant along with language discriminability, making this network a robust, domain-invariant LID system.

CHAPTER 4
Experiments and Result Analysis

In this chapter, we will discuss the effectiveness of the baseline LID system, systems trained using data-augmentation, dropout mechanism, AMTL or their respective combinations. These models are evaluated on their accuracy (in %) and C_{avg} values, a closed set multi-language cost function provided in NIST Language Recognition Evaluation 2015 [24], whose lower value indicates better performance. The networks are implemented using a python programming language in PyTorch1.5.1 framework [25] and essential libraries like NumPy, Scipy, Pandas, Matplotlib, Scikit and speech recognition toolkit Kaldi.

For each network, the BNFs used are consistent and are extracted using the pre-trained BUT-phoenixa feature extractor. From these sequences of BNFs, we create a sequence of units which is given as input to the BLSTM layer of the network, where each unit is a cluster of 35 BNFs frames that span approximately 665ms of speech.

4.1 Baseline System and dropout

An end-to-end DNN acoustic model (LID-net) shown in fig 22 is taken as a baseline model to compare with suggested data augmentation and AMTL training. This model consists of a feature extractor layer and a language classifier and is trained on primitive IITM-TTS dataset. The BLSTM layer in the feature extractor has 128 (blstm 1) and 64 (blstm 2) hidden units respectively. The utterance level embedding, u-vector obtained after the self-attention layer is passed through a FCN containing 128 nodes followed by a tanh activation function. Following this, the utterance level features are passed through a FCN of 9 nodes with softmax output that represent the respective languages. The results obtained on IITM-TTS and IIT Mandi Indian language dataset are mentioned in 1st row of Table 4.1.

Table 4.1: Performance of the baseline system and dropout mechanism

LID system	IITM-TTS		IIT-Mandi	
	Acc	Cavg	Acc	Cavg
LIDnet	98.20	1.04	23.45	43.54
LIDnet + dropout	97.50	1.33	27.1	41.36

We can see from the result that the base LID-system overfitted to the domain conditions of the training data. Though it performed well on seen IITM-TTS data, it performed poorly on the unseen IIT Mandi data due to domain variation. To avoid this over-fitting, we

straightaway used a dropout mechanism with dropout probability 0.2 due to its success in other ANN models. The results of this method are mentioned in the 2nd row of Table 4.1.
This method did perform better than the baseline model but faced the same problem as the baseline system. To add domain variation and address the above problem we move to the next section that shows the effectiveness of data augmentation.

4.2 Effect of Data Augmentation

To test the effectiveness of the data augmentation technique, we first implemented our proposed channel perturbation method. This method led to a 3 fold increase in our primitive dataset. To capture more data complexity, the number of hidden units in BLSTM layers were changed to 192 and 96 respectively. The results of training our baseline system with channel perturbation are shown in 1st row (LID-net + ch) of Table 4.2. Likewise, we trained our model using speed perturbation with a similar increase in the dataset and hidden units whose results are shown in the 2nd row of Table 4.2 (LID-net + sp). We can infer from these results that both perturbations were able to include the complexity introduced by the augmented dataset and outperformed our baseline system on IIT Mandi dataset.

Afterwards, we combined the above-mentioned augmentation techniques. Firstly, channel perturbation was applied followed by speed perturbation. Since it led to a 9 fold increase in the dataset, we changed the BLSTM hidden units to 320 and 128 respectively. The results of this augmentation are given in the 3rd row (LID-net + ch + sp) of Table 4.2. It is evident from the results that this method improved over our baseline as well as the individual augmentation technique on unseen target dataset. We also infer that these two augmentation techniques are also complementary. It is because speed perturbation shifts all frequencies while channel perturbation only cut-off few frequencies leaving other frequencies intact. These methods affect the signal directly; they affect the BNFs differently.

Later, we also applied SpecAugmentation to our dataset because of its low computational cost compared to our proposed method and its success in LibriSpeech dataset. The results of this method are shown in the 4th row (LID-net + SpecAug) of Table 4.2. We can infer that this method performed equivalent to channel and speed and improved over our baseline method. However, it failed to provide the same data complexity and improvement as a combined channel and speed perturbation.

It should also be looked upon that all four augmentation methods have performed equivalently on the validation set. This is because all the augmentation methods have an original dataset and therefore the models have memorized this particular domain. Next, we look upon the adversarial training and its effect on the in-domain dataset.

Table 4.2: Performance of LID-systems trained using Data augmentation techniques

LID system	IITM-TTS		IIT-Mandi	
	Acc	Cavg	Acc	Cavg
LIDnet + ch	98.30	1.02	30.37	39.17
LIDnet + spd	97.60	1.70	28.68	41.38
LIDnet + ch + spd	98.66	.93	33.55	36.95
LIDnet + SpecAug	98.49	.98	31.87	38.71

4.3 Effect of AMTL Training

We first applied AMTL training to our primitive dataset, intending to find speaker invariant features (no channel invariant features due to the absence of channel diversity in the original dataset). To implement this we took our baseline model with primary language classifier and added an auxiliary speaker classifier after the feature extractor. This speaker classifier had a FCN of 128 nodes followed by an output layer of 18 nodes that represented speakers in the dataset. We were able to seize speaker-independent u-vector embeddings by adversarially training feature extractor and speaker classifier. The results of this LID system (LID-net + AMTL) are shown on the 1st row of table 4.3. This method improved over baseline and individual data augmentation methods.

Next, we experimented with adversarial training on our channel augmented dataset (3 fold). To the above-mentioned model, we added one auxiliary channel classifier as shown in Fig 3.1. This channel classifier contains a FCN of 128 nodes followed by 3 output nodes representing channels (original + 2 perturbed). The results of this method (LID-net+ch+AMTL) are shown on the 2nd row of Table 4.3. We can infer that this method extracted both channel and speaker invariant u-vector embeddings that led to an increase in the performance as compared to speaker invariant training. Finally, we applied adversarial training to our 9 fold dataset (channel and speed perturbed dataset) using the same model. This model (LID-net+ch+spd+AMTL) results are shown on the 3rd row of Table 4.3.

We can see from the results that LID-net+ch+spd+AMTL has outperformed every other model that we tested on the IIT-Mandi dataset. While the data augmentation generalised the model to the larger domain sub-space, the AMTL training made sure that the features extracted are less exposed to the imbalance between the seen and unseen target domain.

Table 4.3: Performance of LID-systems trained using Adversarial Training

LID system	IITM-TTS		IIT-Mandi	
	Acc	Cavg	Acc	Cavg
LID-net + AMTL	94.33	3.46	32.20	38.02
LID-net + ch + AMTL	93.50	3.78	38.44	34.62
LID-net + ch + spd + AMTL	92.35	4.42	40.20	31.12

We can further observe that models trained using AMTL ill performed on IITM-TTS validation set as compared to non AMTL based LID-systems. This is because AMTL-based models were forced to not utilise the in-domain conditions as opposed to other LID-systems for language classification. However, our main objective of this project was to extract such embeddings that are LID specific and independent of the in-domain variations.

Though the results improved with data augmentation and AMTL based training, their accuracies were relatively low to incorporate in real-life scenarios. It also indicates that we need huge amounts of a good quality dataset as well as new methods to create a robust LID-systems for low-resource languages.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE OF WORK

5.1 Work Conclusion

In this project, we aimed to develop a robust LID system to an unseen target domain for low resource languages. We proposed an end-to-end DNN acoustic model trained using data augmentation and AMTL. We also proposed channel perturbation, a data augmentation technique to increase the channel diversity of the dataset. While the channel perturbation combined with speed perturbation helped in the better generalisation of the domain subspace, the AMTL training helped in producing domain-invariant features making the LID system robust to mismatched domains. Both techniques combined provided a significant improvement over our previously proposed LID-system on IIT Mandi Indian Language Dataset.

5.2 Future Scope of Work

Though our model provides improved performance it's far from 100% accuracy. In future, we would like to use Transformers for feature extraction due to their competitive results in ASR against traditional LSTM models or the use of hierarchical models for their unsupervised variational length feature representations.

Further, with the advent of multi-resolution analysis in the field of computer vision, we could work on multiresolution speeches to find variations and similarity within the domain itself and use it to remove domain-specific features while simultaneously extracting more LID specific features. With sparse dataset or low resource condition, we could also look into Generative Adversarial Networks [GANs] for the production of a synthetic dataset.

Lastly, end-to-end DNN architectures are highly data-hungry and computationally expensive which are not so helpful for moderate or low resources problems. Meta-Learning is one technique that has been successful in recent Few Short Learning (FSL) problems and has shown a great tendency to converge faster and better fine-tuning of the model. With the ability to learn better algorithms, parameter initialization and network optimization we would like to visit meta-learning for training our LID system.

REFERENCES

[1]. M. Abdelwahab and C. Busso, "Domain adversarial for acoustic emotion recognition," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 12, pp. 2423–2435, 2018

[2]. Z. Meng, Y. Zhao, J. Li, and Y. Gong, "Adversarial speaker verification," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP-2019), pp. 6216–6220.

[3]. Y. Shinohara, "Adversarial multi-task learning of deep neural networks for robust speech recognition." in INTERSPEECH- 2016, pp. 2369–2372.

[4]. Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury, "Data augmentation for deep neural network acoustic modeling," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 23, no. 9, pp. 1469– 1477, 2015

[5]. Lan Wang, Mark JF Gales, and Philip C Woodland, "Unsupervised training for mandarin broadcast news and conversation transcription," in 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP 07. IEEE, 2007, vol. 4, pp. IV–353.

[6]. Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur, "Audio augmentation for speech recognition," in Sixteenth Annual Conference of the International Speech Communication Association, 2015.

[7]. Aurelien Geron, "Introduction to ANN",in Title of Hands on Machine Learning, O'Riley publications, 1st edition, CA, USA, 2017.

[8]. Aurelien Geron, "Recurrent Neural Network", in Title Of Hands on Machine Learning, O'Riley publications, 1st edition, CA, USA, 2017.

[9]. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory.Neural computation,9(8):1735–1780, 1997

[10]. Srivastava and Hinton, "Dropout : A simple way to prevent Neural Networks from Overfitting",in Journal of Machine Learning Research 15 (2014), University of Toronto, Canada, 1929-1958.

[11]. Bayer, Justin, Osendorfer, Christian, Chen, Nutan, Urban, Sebastian, and van der Smagt, Patrick. On fast dropout and its applicability to recurrent networks. arXiv preprint arXiv:1311.0701, 2013.

[12]. Zaremba, Sutskever and Vinyals. On Recurrent Neural Network Regularization . arXiv:1311.0701, 2014

[13]. Gal and Ghahramani. On A theoretically Grounded Application of Dropout in Recurrent Neural Network, in NIPS, University of Cambridge,UK,2016.

[14]. Anna Silnova, Pavel Matejka, Ondrej Glembek, Oldrich Plchot, Ondrej Novotny, Frantisek Grezl, Petr Schwarz, Lukas Burget, and Jan Cernocky, "BUT/phonexia bottleneck feature extractor," in Proc. Odyssey 2018 The Speaker and Language Recognition Workshop, 2018, pp. 283–287.

[15]. Mei Hwang. 2001. *Subphonetic Acoustic Modeling for Speaker-Independent Continuous Speech Recognition*. Technical Report. Carnegie Mellon University, USA.

[16]. Ma Jin, Yan Song, Ian McLoughlin, and Li-Rong Dai, "LID-senones and their statistics for language identification," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 1, pp. 171–183, 2018.

[17]. H. Muralikrishna, S. Pulkit, J. Anuksha, and A.D. Dileep, "Spoken language identification using bidirectional lstm based lid sequential senones," in the proceedings of 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2019), pp. 320–326.

[18]. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate.CoRR, abs/1409.0473, 2014.

[19]. Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio ,"A Structured Self-attentive Sentence Embedding", in International Conference on Learning Representation, 2017.

[20]. Yaroslav Ganin and Victor Lempitsky, "Unsupervised domain adaptation by backpropagation," in International conference on machine learning, 2015, pp. 1180–1189.

[21]. Z. Meng, J. Li, Z. Chen, et al., "Speaker-invariant training via adversarial learning," in Proc. ICASSP, 2018.

[22]. Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk,and Quoc V Le. Specaugment: A simple data augmentation method for automatic speechrecognition.arXiv preprint arXiv:1904.08779, 2019

[23]. Arun Baby, Anju Leela Thomas, NL Nishanthi, TTS Consortium, et al., "Resources for indian languages," in Proceedings of Text, Speech and Dialogue, 2016.

[24]. "The 2015 NIST Language Recognition Evaluation plan (lre15)," https://www.nist.gov/itl/iad/mig/2015-language-recognition-evaluation, 2015.

[25]. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in pytorch," in proceedings of NIPS-W, 2017.

PROJECT DETAILS

Student Details			
Student Name	**Shantanu Kapoor**		
Register Number	160907430	Section / Roll No	A /45
Email Address	shantanu.kapoor13@gmail.com	Phone No (M)	7877630652
Project Details			
Project Title	**Spoken Language Identification in Mismatch Domain**		
Project Duration	4 months	Date of reporting	July 2020
Organization Details			
Organization Name	**Indian Institute of Technology , Mandi**		
Full postal address with pin code	IIT Mandi - Parashar Road, Tehsil Sadar, Near Kataula, Kamand, Himachal Pradesh 175005		
Website address	http://www.iitmandi.ac.in/		
Supervisor Details			
Supervisor Name	**Dr Dileep A. D.**		
Designation	Associate Professor		
Full contact address with pin code	IIT Mandi - Parashar Road, Tehsil Sadar, Near Kataula, Kamand, Himachal Pradesh 175005		
Email address	addileep@iitmandi.ac.in	Phone No (M)	+911905300047
Internal Guide Details			
Faculty Name	**Dr. Aparna U.**		
Full contact address with pin code	Dept. of E&C Engg., Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	aparna.u@manipal.edu		