eman ta zabal zazu

Universidad del País Vasco   Euskal Herriko Unibertsitatea

Noisy Speech Recognition using Kaldi and Neural Architectures

Author: Ander González Docasal

Advisors: Vassilis Tsiaras, George P. Kafentzis, Yannis Stylianou

hap/lap

Hizkuntzaren Azterketa eta Prozesamendua
Language Analysis and Processing

Final Thesis

February 2018

Thesis written and presented within the Erasmus+ programme at

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus
700 13 Heraklion, Crete, Greece

Departments: Computer Systems and Languages, Computational Architectures and Technologies, Computational Science and Artificial Intelligence, Basque Language and Communication, Communications Engineer.

Noisy Speech Recognition using Kaldi and Neural Architectures

Ander González Docasal

Hizkuntzaren Azterketa eta Prozesamendua
Language Analysis and Processing

Final Thesis

University of the Basque Country
Computer Science Faculty
Universidad del País Vasco – Euskal Herriko Unibertsitatea
Facultad de Informática – Informatika Fakultatea
Pº Manuel Lardizabal 1
20018 Donostia-San Sebastián, Gipuzkoa, Spain

Thesis written and presented within the Erasmus+ programme at

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus
700 13 Heraklion, Crete, Greece

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

Noisy Speech Recognition using Kaldi and Neural Architectures

Abstract

The goal of an Automatic Speech Recognition (ASR) system is to transform a set of acoustic features into a sequence of words. It mainly consists of various parts: the feature extraction part which extracts information from a speech signal; the acoustic model, in charge of the conversion from speech to phonemes; and the language model that transforms the detected phonemes into the most probable sequence of words.

Throughout their history, these systems were built with statistical methods, mainly Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM). However, in recent years the use of neural architectures such as Deep, Convolutional and Recurrent Neural Networks (DNN, CNN and RNN), have improved the achieved results significantly. Moreover, freely available tools made ASR research develop quickly. Kaldi is one of the most known and widely used ASR systems. It includes a set of neural network packages —nnet1, nnet2 and nnet3— which can be used for implementing the acoustic model. These are fast, accurate and able to handle huge databases since they distribute the load on clusters of machines. However, Kaldi's slow development cycle implies that new neural architectures may be introduced many years after their publications.

Therefore, in this work we substitute the neural acoustic model of Kaldi by our own implementations written in TensorFlow. TensorFlow has the largest community of users and the best support among the available deep learning libraries. By substituting the Acoustic Model of Kaldi with different architectures and testing their performance on the well-known database Aurora-4, we managed to reduce Word Error Rate (WER) by 3.17 % (baseline 15.14 %) when using a CNN architecture. Also, focusing on just the clean subset of the TEST part of the database, a further improvement has been achieved once implementing a CNN + RNN structure, from a 4.54 % WER with the CNNs alone to a 4.13 % with this architecture.

This work is therefore believed to improve the results on obtained by one of the widely used ASR tools simply by implementing more advanced deep learning techniques, which could be executed by more powerful and dedicated external programs.

For future work, a further analysis on more complex convolutional networks could lead to a better performance in this particular database and, in general, in noisy environments. Finally, further improvement of convolutional *and* recurrent architectures is suggested in clean and noise-free conditions, since they have been shown to obtain the best results in this specific circumstances.

Hizketa Zaratatsuaren Azterketa Kaldi eta Arkitektura Neuronalen bidez

Laburpena

Hizketa Automatikoki Ezagutzeko (ASR) sistema baten helburua tasun akustikoen multzo bat hitz sekuentzia batean bihurtzea da. Ondorengo atalez osatuta dago: tasunen erausketa, hizkuntza-informazioa audio seinaletik erauzten du tasun akustikoko bektore gisa; eredu akustikoa, bektore akustikoak fonematan bihurtzearen arduraduna; eta hizkuntza-eredua, hautemandako fonemekin probabilitate gehien duen hitz sekuentzia itzultzen du.

Haien historia osoan zehar, sistema hauek metodo estatistikoak erabilita eraikitzen ziren, batez ere Markoven Ezkutuko Ereduak (HMM) eta Gaussen Eredu Mistoak (GMM). Hala ere, azkenengo urteetean arkitektura neuronalak erabiliz, hala nola Sare Neuronal Sakonak, Konboluziokoak eta Errepikariak (DNN, CNN eta RNN), lehendabizi lortutako emaitzak modu esanguratsuan hobetzea lortu da. Kaldi gehien ezagutzen eta erabiltzen diren ASR sistemetako bat da. Sare neuronalak ezartzen dituen zenbait pakete (nnet1, nnet2 eta nnet3) ditu barne. Hauek eredu akustikoa inplementatzeko erabil daitezke azkarrak, zehatzak eta datu-base handiak erabiltzeko gai direlako, azken hau zama makina multzoetan banatzen. Hala ere, Kaldik duen garapen ziklo motela dela eta, arkitektura neuronal berriak haien argitalpenetik urte asko igaro arte ez dira ezarriko.

Beraz, lan honetan Kaldiren eredu akustikoa TensorFlow programazio-lengoaian guk idatzitako inplementazioekin ordezkatuko da. TensorFlowk erabiltzaile-elkarte handiena eta euskarririk hoberena ditu ikaskuntza sakoneko beste liburutegiekin konparatuta, alegia. Kaldiren eredu akustikoa beste arkitektura ezberdinekin ordezkatzean Aurora-4 deritzon datubasearekin, lehenengo % 15.14ko hitz-errore-tasako (WER) emaitzak % 3.17 puntutan hobetu ahal izan dira Konboluziozko Sare Neuronalekin entrenatzean. Halaber, TEST datubaseko submultzo garbian bakarrik fokatzean, emaitzak are gehiago hobetzea lortu da CNN + RNN egitura bat ezartzean; konkretuki, CNN bakarrik erabiltzean lortutako % 4.54ko WERa % 4.13 arte murriztu da arkitektura hau erabilita.

Beraz, lan honek ASR sistema zabalenetako batekin lortzen diren emaitzak soilik ikaskuntza sakoneko teknika aurreratuagoak inplementatzen hobe daitezkela frogatzen du. Izan ere, hauek ardura bakarreko beste programa boteretsuagoren bidez exekuta daitezkeela ere erakusten du.

Hurrengo lanetarako, CNN konplexuagoetan analisi sakonagoak egiteak ASR sisteman errendimendu hobea izatea erakar lezake datu-base konkretu honetan eta, orokorrean, inguru zaratatsuetan. Hala ere, egoera garbietan lan eginez gero CNN-etan *eta* RNN-etan

fokatu beharko lizateke, hauek izan baitira baldintza hauekin emaitza hoberenak lortu dituztenak.

Αναγνώριση Θωρυβώδους Ομιλίας με χρήση του Kaldi και Αρχιτεκτονικές Νευρωνικών Δικτύων

Περίληψη

Ο στόχος ενός Συστήματος Αυτόματης Αναγνώρισης Ομιλίας (ASR) είναι να μετασχηματίσει ένα σύνολο ακουστικών χαρακτηριστικών σε μια ακολουθία λέξεων. Αποτελείται κυρίως από διάφορα μέρη: η εξαγωγή χαρακτηριστικών που εξάγει τις πληροφορίες από ένα ηχητικό σήμα. το ακουστικό μοντέλο, υπεύθυνο για τη μετατροπή από φωνή σε φωνήματα. και το μοντέλο γλώσσας, το οποίο μετατρέπει τα ανιχνευθέντα φωνήματα στην πιο πιθανή ακολουθία λέξεων.

Κατά τη διάρκεια της ιστορίας τους, αυτά τα συστήματα κατασκευάστηκαν με στατιστικές μεθόδους, κυρίως μοντέλα Hidden Markov Models (HMM) και Gaussian Mixture Models (GMM). Ωστόσο, τα τελευταία χρόνια η χρήση νευρωνικών αρχιτεκτονικών όπως τα Deep, Convolutional και Recurrent Neural Networks (DNN, CNN και RNN) έχουν βελτιώσει σημαντικά τα αποτελέσματα. Το Kaldi είναι ένα από τα πιο γνωστά και ευρέως χρησιμοποιούμενα συστήματα ASR. Περιλαμβάνει ένα σύνολο πακέτων νευρωνικού δικτύου (nnet1, nnet2 και nnet3) τα οποία μπορούν να χρησιμοποιηθούν για την υλοποίηση του ακουστικού μοντέλου. Αυτά είναι γρήγορα, ακριβή και ικανά να χειρίζονται τεράστιες βάσεις δεδομένων αφού κατανέμουν το φορτίο σε συστοιχίες μηχανών. Ωστόσο, ο αργός κύκλος ανάπτυξης του Kaldi υποδηλώνει ότι νέες νευρωνικές αρχιτεκτονικές μπορούν να υλοποιηθούν πολλά χρόνια μετά τις δημοσιεύσεις τους.

Επομένως, σε αυτό το έργο το νευρωνικό ακουστικό μοντέλο του Kaldi θα αντικατασταθεί από τις δικές μας υλοποιήσεις γραμμένες στο TensorFlow. Το TensorFlow έχει τη μεγαλύτερη κοινότητα χρηστών και την καλύτερη υποστήριξη από τις διαθέσιμες βιβλιοθήκες deep learning. Αντικαθιστώντας το ακουστικό μοντέλο του Kaldi με διαφορετικές αρχιτεκτονικές με τη βάση δεδομένων του Aurora-4, τα προηγούμενα αποτελέσματα του Kaldi 15.14 μέσο Word Error Rate (WER) κατάφερε να βελτιωθεί σε 3.17 ποσοστιαίες μονάδες όταν εκπαιδεύτηκε με Convolutional Νευρωνικά Δίκτυα. Επίσης, εστιάζοντας στο καθαρό υποσύνολο της βάσης δεδομένων Test, επιτεύχθηκε περαιτέρω βελτίωση από την υλοποίηση μιας δομής του CNN + RNN, από 4.54 του WER μόνο στα CNN σε 4.13 με αυτήν την αρχιτεκτονική.

Συνεπώς, πιστεύεται ότι οι εργασίες αυτές βελτιώνουν τα αποτελέσματα σε ένα από τα ευρύτερα χρησιμοποιούμενα συστήματα ASR, απλώς εφαρμόζοντας πιο προηγμένες τεχνικές deep learning, οι οποίες θα μπορούσαν να εκτελεστούν με πιο ισχυρά και ειδικά προγράμματα, όπως αποδεικνύεται στην παρούσα αναφορά.

Για μελλοντικές εργασίες, μια περαιτέρω ανάλυση για πιο περίπλοκα convolutional δίκτυα θα μπορούσε να οδηγήσει σε καλύτερη απόδοση του συστήματος ASR στη συγκεκριμένη βάση δεδομένων και, γενικά, σε θορυβώδη περιβάλλοντα. Παρ 'όλα αυτά, σε καθαρές και χωρίς θόρυβο συνθήκες, προτείνεται περαιτέρω βελτίωση των συνθετικών και επαναλαμβανόμενων αρχιτεκτονικών, καθώς αυτές θεωρήθηκαν ότι

επιτυγχάνουν τα καλύτερα αποτελέσματα σε αυτές τις συγκεκριμένες περιστάσεις.

Acknowledgements

This thesis would not have been possible without the collaboration of the University of Crete, where, under the Erasmus+ programme, all the work regarding this thesis has been done. from all the people that advised me and helped in the process of working in this thesis, I would like to specially thank Vassilis Tsiaras, the main supervisor of this project, since without him all this work would have been impossible to achieve. Also, I would like to thank Nikos Komodakis for his help in the investigations. To George Kafentzis, who has been always supportive and helpful in any doubt or problem I had had along these months. And last, but not least, Yannis Stylianou, whom I thank this exchange.

I would also refer in these acknowledgements to all the professors in the Master of Language Analysis and Processing of the University of the Basque Country that have guided me in order to be able to participate on the Erasmus+ programme. Specially to the professors on speech technologies Inma Hernáez and Eva Navas, and to the Master's Coordinator Olatz Arregi.

I am also grateful to all of those with whom I have worked here in the Speech Processing laboratory. Also, to all those people that I have met during my stay here in Crete. You all have made my stay here a wonderful experience that I will never forget.

And finally, the most important part of this experience, those who have still helped me even though I was kilometers away from them. My family, who offered their help and support in everything they could during my stay. My friends that have always been accessible and actively speaking with me. And finally, to my girlfriend, who has been always for me every single day I was on the island, and even kindly accepted my departure and encouraged me to enjoy it as much as I could.

Thank you all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nobody doubts that in these years the interests on characterising and studying speech signals have increased. The number of applications using this technology is rapidly increasing, including commercial products like Alexa from Amazon, Siri from Apple and Google Assistant. It is also worth mentioning that a huge number of universities have a research group focused on developing techniques for synthesising or analysing speech. Even international conferences dedicated to this specific area are held in various parts of the globe, such as InterSpeech or Odyssey.

In the last few years, synthesised speech has reached a quality almost near real human voice with the implementation of WaveNet [27]. Also, speech recognition systems, those that obtain a sequence of words from a raw audio input, have achieved really decent results in this decade [28]. Anyway, the computational power needed for these two applications to reach this performance peak is far away from being easily achieved by low budget research teams.

1.1 Automatic Speech Recognition Systems

The main task of an Automatic Speech Recognition (ASR) system is to map a sequence of acoustic features into a sequence of words [14]. This problem presents a challenging task quite far from being solved in every situation such as in noisy conditions, although in some specific environments important results have been achieved [28] [22]. ASR

ASR systems typically consist on different blocks that perform a sequence of tasks. Since the input of the whole system will be an audio signal, the first steps should deal with the problems related to them: background noise, different speakers, reverberation or quality of the microphone just to mention a few. In order to solve these, the first task of the ASR system is to transform the audio signal into a sequence of vectors of acoustic features from the fed speech signal. These acoustic features are intended to isolate the information relevant to the actual speech, where the linguistic knowledge relies. Then, the last set of blocks is needed to compute the most probable outcome of words given a specific sequence of acoustic vectors, by computing first the most probable phonemes and then, with that information, the most probable sequence of words.

Kaldi is the name of one of the most known ASR systems nowadays, characterised by its philosophy of being open-source and easy to use [19]. It includes a large set of programs dedicated to many different algorithms corresponding to the various tasks inside an ASR. Its major features include Finite State Transducers (FSTs), signal processing, linear algebra and statistic models. Nevertheless, the basis for building neural architectures using this software is still not as developed as in other dedicated softwares.

FSTs

1.2 Neural Networks

In this work, one of the building blocks of the Kaldi ASR system will be programmed using deep learning algorithms, a subset of machine learning algorithms built upon Neural Networks (NNs). The main idea behind Neural Networks is to model a task by a function dependant on, obviously, the input, a huge set of parameters to be tuned, and some kind of non-linearity. By modifying the structure of the network and its parameters with a vast number of examples, the program results in a function that better fits the desired task.

NN

These machine learning algorithms have been directing the research of many different fields in data processing and computer sciences —from graphics in areas such as face recognition [25] and image generation [9] to Natural Language Processing (NLP) in a wide range of fields such as question answering [16], sentence representation [15] and in the area this thesis is focused on, speech recognition [21] [11] [8].

NLP

The main reason for the use of NNs in recent literature relies on that they have been seen to achieve an equal or even better performance than the previous state-of-the-art technologies [16] [15] [25] and the current trend suggests that in the future they will yield even better results.

1.3 Motivation

Even though Kaldi includes three codebases that allow the user to work with NNs, these do not cover the majority of known deep learning architectures. In addition, compared to the different libraries built for the use of these machine learning technologies, their flexibility is not as extended, though they are supposed to keep growing.

A huge variety of deep learning architectures could be applied to the speech recognition task, since the possibilities are almost endless. Therefore, a previous evaluation of some of the techniques could enhance the creation of more specific libraries whose performance has been already tested achieving accurate results. The more range of tested architectures whose performance approximates to the state-of-the-art, the more options could be considered when developing new speech recognition tools.

Basically, these are the main reasons why this work is oriented in substituting one of the core elements of this ASR system with an external deep learning program. This way, the libraries created for a range variety of machine learning tasks can be used for this specific duty, which offer a wider range of possibilities to be tested and therefore potentially finer results.

1.4 Thesis Contribution

With this approach, the work is focused on verifying that the use of different deep architectures can lead to a significant improvement on the results of the previous works based on statistical methods, since it has already been proven [16] [15] [25].

Also, it will clear the path for the users of the Kaldi ASR system that would like to improve their results by using external programs. Since the preprocessing of the audio and feature extraction made by Kaldi are quite straightforward, a user non experienced on signal processing could easily get the material needed for a more specific task on machine learning. Therefore, this works proves that a symbiosis of different tools, each specialised in different areas, can be achieved in order to obtain more fruitful results.

Finally, as stated before, these broad tests on various architectures could lead to the application of some of the techniques used on the Kaldi ASR system. Since all of them have been already analysed and used on it, the contributors could base the new features to add to Kaldi on the experiments performed during this work.

1.5 Structure of the Thesis

During this thesis the Acoustic Model of Kaldi will be substituted with different neural architectures built in TensorFlow. This whole process will be divided as follows: The Chapter 2 explains more deeply the elements of an ASR system that have been used all along this work, the Acoustic Model and the Feature Extraction modules. In the same chapter the underlying structures of the neural architectures that have been used in this work are explained.

The Chapter 3 is divided in two sections. The first will introduce the tools that have been employed for the research: the Aurora-4 database, Kaldi and TensorFlow. The second section includes the whole experimental part. It contains three subsections for each of the architectures that have been tested: DNNs, CNNs and RNNs. The results are displayed in a series of tables that contain the number of parameters to be trained and the scores for each of the partitions on the TEST dataset, as well as their mean and standard deviation. Also, some conclusions will be provided after each of the results. Since the best results are obtained with the use of filter banks instead of MFCCs, from CNNs onwards the networks will only be fed with these features.

Finally, the Chapter 4 explains the whole conclusions that we found during the research, as well as some guidelines for future work. The most remarkable note will be that these architectures actually achieve better results than naïve Kaldi, and therefore it should have them implemented on the next years.

Chapter 2

State-of-the-Art

In this chapter the basic techniques used all over this work will be presented. This will include an overview on Automatic Speech Recognition systems, specifically on their most important parts regarding this thesis: the Acoustic model and Feature extraction.

In addition, an introduction on neural architectures is also provided. It includes an explanation about the different elements that will be used all over the experimental part.

2.1 Automatic Speech Recognition Systems

As stated before, the task of an ASR system is to map a set of acoustic inputs into a sequence of words for, among other applications, transcribing speech. Of the different areas the ASR systems cover, the most difficult tasks happen to be those including a big set of vocabulary —more than 5,000 different words— and continuous speech —where the words are uttered naturally and not between silences [14].

2.1.1 Acoustic Model

Speech recognition systems tend to characterise the acoustic information of an utterance as it were produced as the output of a so called *noisy channel* [14] when fed with the text version of the signal. This is, the main idea is to consider the opposite problem to the one being solved: in this case, instead of getting a sequence of words from a given audio signal (the *actual* problem to solve), it is considering what sequence of words will produce that given audio signal. This way, the task will be reduced to find the most probable sequence of words that will produce the original acoustic signal from all possible sentences.

Noisy channel

This may result in a problem, since two different utterances of a single sentence will never be exactly the same, due to the set of problems previously discussed. Also, the set of all possible sentences in a language is infinite and computing the probabilities of each particular sentence will therefore be useless. In order to overcome these problems, two new tasks should be presented: the characterisation of a speech signal into a set of acoustic features, nowadays resolved by different approaches such as the MFCC, Δ and Δ^2 parameters [17]; and the decoding or search between the sentences to obtain the most probable one, using for example the Viterbi or the A^* algorithms [14].

Once supposing the characterisation problem is overcome, the speech signal $s(t)$ will be represented by a set of acoustic observations $O = o_1, \dots, o_t$, and its corresponding sentence will be $W = w_1, \dots, w_n$. Then, the most probable outcome will be computed by:

$$\hat{W} = \max_W P(W|O)$$

Which, by the use of Bayes' rule, can be reduced to:

$$\hat{W} = \max_W \frac{P(O|W)P(W)}{P(O)}$$

And, since the probability of the observations $P(O)$ will be the same $\forall W$, that probability is not relevant for guessing the maximum, thus:

$$\hat{W} = \max_W P(O|W)P(W)$$

The probability of a single sentence of words of occurring in a language $P(W)$ is computed by the *linguistic model*, whereas the probability of an output sentence given a set of words $P(O|W)$ is computed by the *acoustic model*, the main objective of this work.

Linguistic model

Acoustic model

2.1.2 Feature Extraction

In order to use the audio data for speech recognition the raw files should be preprocessed so they can be converted in a vector of numeric values in what is called *feature extraction*. The most extended technique is the extraction of the *Mel Frequency Cepstral Coefficients* (MFCCs) [17].

Feature extraction MFCC

The raw audio data is first divided into chunks containing 20 to 40 ms of samples, with a separation between them of around 50 % of the length of the frame. Then, to each frame x_n of length N samples ($n \in \{0, \dots, N-1\}$) a windowing process is applied, typically using a *Hamming* window w_n:

Hamming windowing

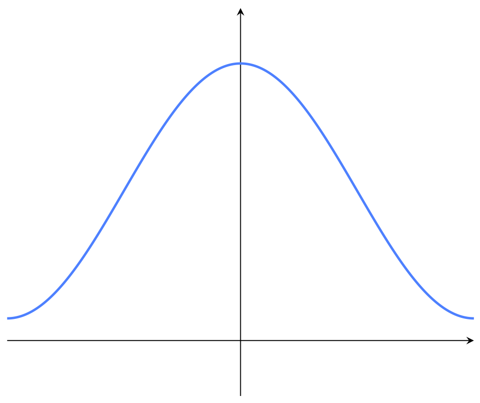

Figure 2.1: Hamming Window

in the following way:

$$x_n \quad \leftarrow \quad x_n \cdot w_n$$

Then the Fourier Transform is applied in order to convert the samples from time domain to frequency domain.

$$X(\omega) = \mathcal{F}[x_n]$$

Since the human hearing perception is not linear with the frequency scale, another transformation should be applied, one that transforms $X(\omega)$ to the *Mel scale*. Let f be such that $\omega = 2\pi f$, then the Mel scale frequency m is computed as

Mel scale

$$\begin{aligned} m &= 2595 \log\left(1 + \frac{f}{700}\right) \\ &= 1127 \ln\left(1 + \frac{f}{700}\right) \end{aligned}$$

Then, a *filter bank*, a series of K triangular filters $\Lambda_k(m)$, are applied to the signal $X(m)$ for each of the desired Mel frequencies, with the objective of simulating the response of the human auditory system [26]. Then result of the integral (discrete sum) along the frequencies inside the filter stored [5]:

Filter bank

$$fb_k = \int_0^\infty \Lambda_k(m) \left\|X(m)\right\| \, dm \qquad \left(= \sum_{\xi=1}^{M} \Lambda_k(\xi) \left\|X(\xi)\right\| \right)$$

Lastly, a Discrete Cosine Transform is applied in order to get the MFCCs, computed in the following way [5]:

$$\text{MFCC}_i = \sum_{k=1}^{K} \log(fb_k) \cos \frac{(2k-1)i\pi}{2M}$$

Where $i \in \{1, \ldots, C\}$, usually $C < K$ [5].

2.2 Neural Networks

Neuron or perceptron

Neural Networks consist in a set of single objects called *neurons* [10] or *perceptrons* which compute a single function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ that depends on a set of elements known as the weights vector $\vec{w} \in \mathbb{R}^n$, the bias $b \in \mathbb{R}$ and the activation function $\phi : \mathbb{R} \longrightarrow [p, q]$ where $p, q \in \mathbb{R}$ in the following way:

$$f(\vec{x}) = \phi(\vec{x} \cdot \vec{w} + b) = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Threshold

$\sigma(x)$

$\text{ReLU}(x)$

$\tanh(x)$

The main role of the activation function $\phi(\cdot)$ is to squash the results of the operation $\vec{x} \cdot \vec{w} + b$ into a range of natural numbers $[p, q]$ —usually $[0, 1]$ or $[-1, 1]$— and to provide non-linearity. A set of different activation functions include the *threshold function*, the *sigmoid function* known as $\sigma(\cdot)$, the *Rectified linear unit* $\text{ReLU}(x)$[1] or the *hyperbolic tangent function* $\tanh(x)$:

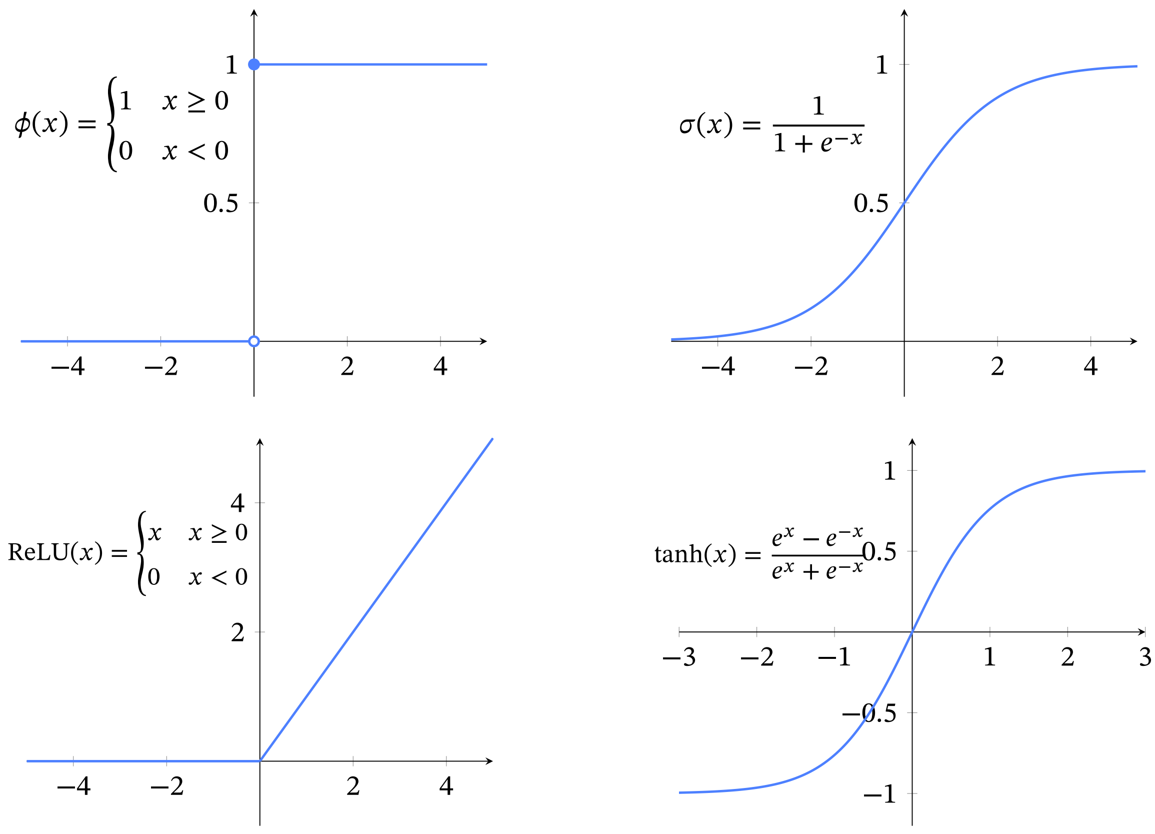

Figure 2.2: The threshold (above left), sigmoid (above right), ReLU (below left) and hyperbolic tangent (below right) functions.

[1]Note that the ReLU function only has an effect on negative values, since its output lies in the domain $[0, +\infty)$. Nevertheless, its input is usually superiorly bounded due to the attributes of the network, and so is the output.

2.2.1 Neural Networks

Layer

Once defined a perceptron, these are usually structured in what it is called a *layer*, a set of m neurons which all receive the same input $\vec{x} \in \mathbb{R}^n$ and compute an output $\vec{y} \in \mathbb{R}^m$ in such a way that the j-th element of $\vec{y}$, y_j, is the output of the function of the j-th neuron $f_j(\vec{x})$:

$$y_j = f_j(\vec{x}) = \phi(\vec{x} \cdot \vec{w}_j + b_j) = \phi\left(\sum_{i=1}^{n} x_i w_{ij} + b_j\right)$$

Which, supposing that $\vec{x}$ and $\vec{y}$ are both row vectors, can be expressed in a vectorial way in the form:

$$\vec{y} = \phi\left(\vec{x}W + \vec{b}\right)$$

where the element w_{ij} of the matrix W is the i-th weight of the j-th neuron, the element b_j of the vector $\vec{b}$ is its bias and the function $\phi(\vec{x})$ is applied elementwise.

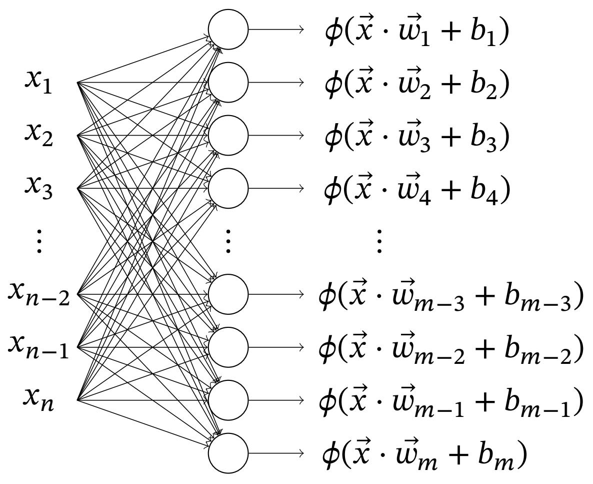
x_1
x_2
x_3
$\vdots$
x_{n-2}
x_{n-1}
x_n
$\phi(\vec{x} \cdot \vec{w}_1 + b_1)$
$\phi(\vec{x} \cdot \vec{w}_2 + b_2)$
$\phi(\vec{x} \cdot \vec{w}_3 + b_3)$
$\phi(\vec{x} \cdot \vec{w}_4 + b_4)$
$\vdots$
$\phi(\vec{x} \cdot \vec{w}_{m-3} + b_{m-3})$
$\phi(\vec{x} \cdot \vec{w}_{m-2} + b_{m-2})$
$\phi(\vec{x} \cdot \vec{w}_{m-1} + b_{m-1})$
$\phi(\vec{x} \cdot \vec{w}_m + b_m)$

Figure 2.3: Diagram of a perceptron layer.

Neural network

If a set of ℓ layers is placed consecutively in such a way that the output of the layer k is feed as input to the layer $k+1$, then it is called a *neural network*. The overall function of the network is highly non-linear and dependant in a great number of parameters, basically the ℓ matrices $W^{(k)}$ and biases $\vec{b}^{(k)}$.

$$\vec{y} = \phi\left(\cdots\phi\left(\phi\left(\vec{x}W^{(1)} + \vec{b}^{(1)}\right)W^{(2)} + \vec{b}^{(2)}\right)\cdots W^{(\ell)} + \vec{b}^{(\ell)}\right)$$

Input/output layer

Hidden layer

DNN

The first and last layers of the network are called the *input* and *output layers*, while any other layer in between is called a *hidden layer*. If the number of layers is greater than 2, that is, when the network includes hidden layers, then the network is called a *deep neural network* (DNN) [4].

These networks are called *feedforward networks* since the input is passed through the different layers and not backwards in order to get an output [10], or *dense layers* since every neuron in one layer is connected every neuron on the next layer. Feedforward network

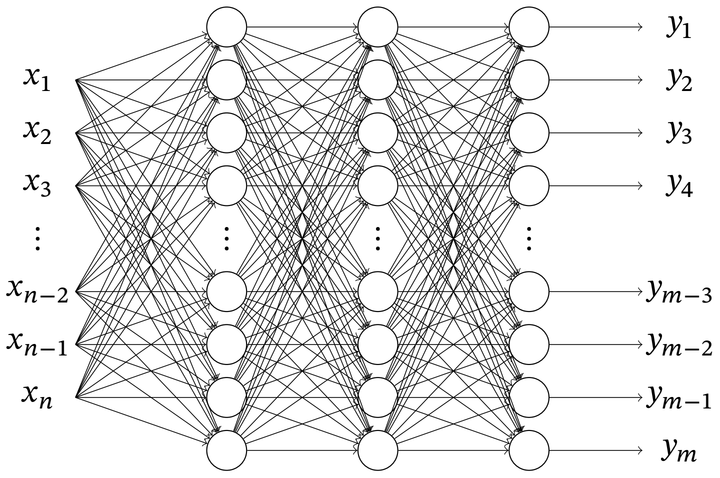

Figure 2.4: Diagram of a neural network.

2.2.2 Cost Function and Training

The main idea behind building a NN is to make it learn from a series of data. Being $\vec{x}$ the input of the network and $\vec{y}$ its output, the function that the network computes is

$$\vec{y} = F(\vec{x})$$

This function F is not only dependant on the input $\vec{x}$, but also on the set of parameters in the network $W^{(\ell)}$ and $b^{(\ell)}$. Let Θ be a vector that contains all the parameters in a row, then the function F is

$$\vec{y} = F(\vec{x}, \Theta)$$

Then, given a set of paired examples $(\vec{x}, \vec{y}_l)$ where $\vec{y}_l$ is the desired value of $F(\vec{x})$, then the computed $\vec{y}$ should be similar to $\vec{y}_l$.

Cost Function

Any function that can be used to measure the performance of a NN (this is, measuring the similarity between $\vec{y}$ and $\vec{y}_l$) is called the *cost function* [10] of the NN. This function should be positive and decrease in value for a better performance.

SGD

In order for the cost function to be minimum, different techniques are proposed in bibliography, such as *Stochastic Gradient Descent* (SGD). This approach uses the value of the gradient of the cost function for minimisation. Let $C(\Theta)$ be the cost function of a NN. Then, the gradient is computed as

$$\frac{\partial C(\Theta)}{\partial \Theta}$$

Then, as stated in [10], the parameter Θ should be updated in the form

$$\Theta \leftarrow \Theta - \alpha \frac{\partial C(\Theta)}{\partial \Theta}$$

where $\alpha > 0$ is called the *learning rate* [10].

Learning rate

Normally, instead of computing the gradient for each sample produced, the dataset is divided in *batches* of m samples and the gradient is computed for all of them [13]. This practice speeds up the whole computation process leading to a more efficient training.

Batch

2.2.3 Batch Normalisation

Another technique that may help in the training of NNs is *batch normalisation* (BN). As S. Ioffe et al. suggest in [13], each dimension $x^{(k)}$ is normalised first according to its mean $\mu^{(k)}$ and standard deviation $\sigma^{(k)}$:

Batch Normalisation

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu^{(k)}}{\sigma^{(k)}}$$

The parameters $\mu^{(k)}$ and $\sigma^{(k)}$ are calculated for each of the computed batches. Then, two new learnable parameters are added, $\gamma^{(k)}$ and $\beta^{(k)}$ in such a way that

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

whenever the relation between x_i and y_i does not have any activation functions. In that case, the function is applied after the BN:

$$y_i^{(k)} = \phi(\gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)})$$

As it is stated in [13], this technique helps in a faster training and in achieving a higher accuracy.

2.2.4 Dropout

One of the major problems in machine learning is the so called *overfitting*, where the learning system performs well in the examples used for training but it fails in others that never went through the program. With the objective of avoiding such an issue, N. Srivastava et al. [24] decided to eliminate some of the connections inside a network so not all the data goes through the parameters.

Overfitting

In a dropout layer, each of the connections on every neuron of the previous layer is sent to the next with a probability p, called the *droput probability*.

Dropout probability

As shown in [24], they could achieve results near to the state-of-the-art performance by applying dropout on both DNN and CNN.

2.2.5 Convolutional Network

Convolution
CNN
Kernel

A convolutional layer inside a neural network is a layer that performs a linear operation called *convolution*, and a convolutional network (CNN) is a neural network that includes at least one convolutional layer inside [7]. This convolution operation requires two tensors to be executed: the *input* I, which has the information about the data; and the *kernel* K. If I and K are 1-D tensors, then:

$$(I * K)_i = \sum_{\eta=-\infty}^{\infty} I_\eta \cdot K_{\eta-i} = \sum_{\eta=-\infty}^{\infty} I_{\eta-i} \cdot K_\eta$$

In the case of 2-D tensors, then:

$$(I * K)_{ij} = \sum_{\eta=-\infty}^{\infty} \sum_{\xi=-\infty}^{\infty} I_{\eta\xi} \cdot K_{\eta-i,\xi-j} = \sum_{\eta=-\infty}^{\infty} \sum_{\xi=-\infty}^{\infty} I_{\eta-i,\xi-j} \cdot K_{\eta\xi}$$

And generalising:

$$\begin{aligned}(I * K)_{i_1,\ldots,i_k} &= \sum_{\eta_1=-\infty}^{\infty} \cdots \sum_{\eta_k=-\infty}^{\infty} I_{\eta_1,\ldots,\eta_k} \cdot K_{\eta_1-i_1,\ldots,\eta_k-i_k} \\ &= \sum_{\eta_1=-\infty}^{\infty} \cdots \sum_{\eta_k=-\infty}^{\infty} I_{\eta_1-i_1,\ldots,\eta_k-i_k} \cdot K_{\eta_1,\ldots,\eta_k}\end{aligned}$$

In the case the kernel is not flipped but computed in the same direction as the tensor, then the operation is called *cross-correlation*. In the case of 2-D tensors:

$$CC(I,K)_{ij} = \sum_{\eta=-\infty}^{\infty} \sum_{\xi=-\infty}^{\infty} I_{\eta\xi} \cdot K_{\eta+i,\xi+j} = \sum_{\eta=-\infty}^{\infty} \sum_{\xi=-\infty}^{\infty} I_{\eta+i,\xi+j} \cdot K_{\eta\xi}$$

If the output is restricted only to the values where the kernel K is completely embedded inside the input I, it is called a *valid* convolution.

Valid convolution

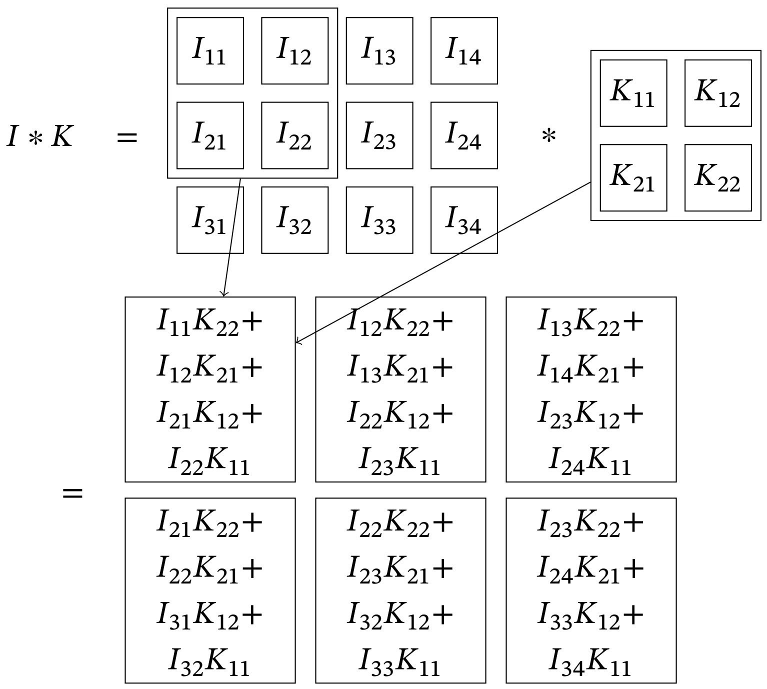
$I * K$
$=$
I_{11} I_{12} I_{13} I_{14}
I_{21} I_{22} I_{23} I_{24}
I_{31} I_{32} I_{33} I_{34}
$*$
K_{11} K_{12}
K_{21} K_{22}
$=$
$I_{11}K_{22}+ I_{12}K_{21}+ I_{21}K_{12}+ I_{22}K_{11}$
$I_{12}K_{22}+ I_{13}K_{21}+ I_{22}K_{12}+ I_{23}K_{11}$
$I_{13}K_{22}+ I_{14}K_{21}+ I_{23}K_{12}+ I_{24}K_{11}$
$I_{21}K_{22}+ I_{22}K_{21}+ I_{31}K_{12}+ I_{32}K_{11}$
$I_{22}K_{22}+ I_{23}K_{21}+ I_{32}K_{12}+ I_{33}K_{11}$
$I_{23}K_{22}+ I_{24}K_{21}+ I_{33}K_{12}+ I_{34}K_{11}$

Figure 2.5: Example of a valid convolution of two matrices. Notice how the kernel matrix is flipped both vertically and horizontally.

2.2.6 Recurrent Neural Networks

If the input of a neural network depends on time in a discrete way, then the network could be arranged in a way such as the input of any layer in a given time $\vec{x}_t$ is influenced somehow by the p previous inputs or already computed outputs $\{\vec{v}_{t-k}\}_{k=1}^{p}$. In this case, the network is called *recurrent* (RNN) [10].

RNN

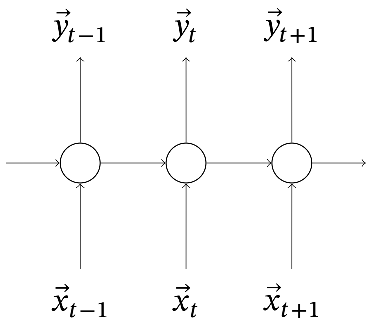

Figure 2.6: Simplified diagram of a recurrent network where each neuron depends on the output of the previous sample.

If the output of the network depends also, if possible regarding to the data, on q future inputs $\{\vec{x}_{t+k}\}_{k=1}^{q}$, then the network is also called *bidirectional* [23].

Bidirectional network

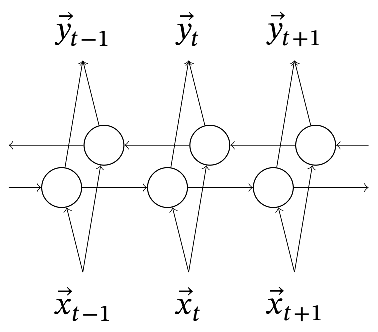

Figure 2.7: Diagram of a simplified bidirectional recurrent network where each output depends on the previous and next neurons.

LSTM A particular case of RNNs is one that uses *Long Short-Term Memory* (LSTM) cells [12] [6]. This type of neurons are used for processing inputs which is context-dependant, this is, that the input of a particular time is influenced by past or future samples. They are designed intentionally for retaining relevant information and forgetting what is not important at a particular moment [12].

The function computed by these cells is more complex than the one from a perceptron. This depends on some intermediate functions called the *input*, *output* and *forget gates* $\vec{i}_t$, $\vec{o}_t$ and $\vec{f}_t$; the *cell activation vector* $\vec{c}_t$; and the *cell output activation vector* $\vec{m}_t$, in the way described by the following expressions [21]:

$$
\begin{aligned}
\vec{i}_t &= \sigma(\vec{x}_t W_{ix} + \vec{m}_{t-1} W_{im} + \vec{c}_{t-1} W_{ic} + \vec{b}_i) \\
\vec{f}_t &= \sigma(\vec{x}_t W_{fx} + \vec{m}_{t-1} W_{fm} + \vec{c}_{t-1} W_{fc} + \vec{b}_f) \\
\vec{c}_t &= \vec{f}_t \odot \vec{c}_{t-1} + \vec{i}_t \odot \phi_c(\vec{x}_t W_{cx} + \vec{m}_{t-1} W_{cm} + \vec{b}_c) \\
\vec{o}_t &= \sigma(\vec{x}_t W_{ox} + \vec{m}_{t-1} W_{om} + \vec{c}_t W_{oc} + \vec{b}_o) \\
\vec{m}_t &= \vec{o}_t \odot \phi_m(\vec{c}_t) \\
\vec{y}_t &= \phi_y(\vec{m}_t W_{ym} + \vec{b}_y)
\end{aligned}
$$

where the symbol $\odot$ denotes an element-wise multiplication, the W_{**} are weight matrices, the $\vec{b}_*$ are bias vectors, the function ϕ_c and ϕ_m are generally the tanh function, and the function ϕ_y is the *output activation function*.

If the architecture of the network includes LSTM cells that read future inputs, if possible, BiLSTM then it is said that the network contains *bidirectional LSTMs* or, abbreviating, *BiLSTMs*.

Another type of recurrent unit was presented in [3] which was based on the LSTM but with GRU the aim of being simpler to compute and implement, the Gated Recurrent Unit (GRU). Let $\vec{h}_t$ be the hidden state of one of this cells at the instant t. The next state depends on the intermediate functions $\vec{r}_t$ and $\vec{z}_t$ called the *reset gate* and the *update gate* respectively, which are computed by the following expressions:

$$\begin{aligned}
\vec{r}_t &= \sigma(\vec{x}_t W_{rx} + \vec{h}_{t-1} W_{rh}) \\
\vec{z}_t &= \sigma(\vec{x}_t W_{zx} + \vec{h}_{t-1} W_{zh}) \\
\vec{\tilde{h}}_t &= \phi(\vec{x}_t W_{hx} + (\vec{r}_t \odot \vec{h}_{t-1}) W_{hh}) \\
\vec{h}_t &= \vec{z}_t \odot \vec{h}_{t-1} + (1 - \vec{z}_t) \odot \vec{\tilde{h}}_t
\end{aligned}$$

where, similarly to the LSTMs, the W_{**} are weight matrices and ϕ is the *activation function* of the cell.[2]

NAS

Finally, the implementation on Neural Architecture Search (NAS) will also be used on this work [31]. Its main goal is to find the most suitable architecture for a given problem by using RNNs. Since the structure of any given neural architecture could be specified by a string, it is also possible to use *another* architecture, named *the controller*, in order to generate the previously stated string, the *child network*. As it is stated in [31], it could find an architecture whose performance was better than most of those created by humans.

2.3 Deep Learning Architectures for ASR Systems

Hybrid architecture

End-to-end architecture

Although the way in which a deep learning algorithm is applied to an ASR system varies among the literature, two major approaches can be distinguished: the *hybrid* architecture which includes some type of neural network inside a statistical system, such as the use of networks along with a Hidden Markov Model (HMM) [2] [20]; and the *end-to-end* architecture, that relies solely in a deep learning architecture that gets as input the acoustic features and outputs its transcription [30] [29].

The range of architectures found in the different submitted articles varies in a significant way, possibly correctly stating that every single paper has a unique set of layers. Though that might be the case, among the authors a preferred technique can be found, in which the use of CNNs accompanied by RNNs, mostly (Bi)LSTMs, such as in [29] [22] [28], which can be seen that get more promising results than when using CNNs alone [30].

[2]Note that the operation $1 - \vec{v}$ will yield a vector such that its j-th element is $1 - v_j$

Chapter 3

Experiments and Results

3.1 Tools and Resources

3.1.1 Aurora-4

Aurora-4 [18] is the audio database used throughout this work. It is based on a first instance on the Wall Street Journal (WSJ) database, more specifically the SI-84 WSJ0 database, composed by a set of very clean utterations in terms of noise. Since the performance of systems that use a low SNR database decays as background noise level increases, the following recorded backgrounds are directly added to the database:

Aurora-4

WSJ database

- Street traffic
- Train terminals and stations
- Cars
- Babble
- Restaurants
- Airports

The added noise has been recorded by two types of microphones:

- A head-mounted Sennheiser HMD-414 close-talking microphone.
- A microphone selected from a group of other 18 common type microphones, such as Crown PCC-160, Sony ECM-50PS and Nakamichi CM100.

The TRAIN database includes 7138 utterances composed by:

- For each of the 2 types of microphones (Training Set 1 & 2):
  - 893 clean samples (directly taken from the database)
  - For each the 6 different types of noise:
    - ∗ 446 samples with added noise, each with a randomly chosen SNR varying from 10 to 20 dB in steps of 1 dB.

The TEST database contains 4620 utterances: 14 different Test Sets composed by randomly selecting 330 utterances for each of the 7 conditions (6 with noise + 1 clean) and each of the 2 microphones. The noise in this case has a randomly chosen SNR from 5 to 15 dB in steps of 1 dB.

The reader is encouraged to check the complete report available online for a more technical review of the datasend along with different performance tests.

3.1.2 Kaldi

Kaldi

Kaldi [19] is a speech recognition toolkit with aims of having an easy to extend code. It mainly uses a set of Finite State Transducers (FSTs) and linear algebra and it has a large community of users that test and support it.

It includes a large set of tools and programs needed for speech recognition such as data preprocessing, feature extraction, HMMs, decision trees and neural networks. For more information the reader may please to review its main website `http://kaldi-asr.org`.

The main architecture model is composed by a set of modules named that mainly compute the following operations:

- Data preparation
- Feature extraction
- Acoustic modelling
- Language modelling

The purpose of this work is to be able to substitute the acoustic model that Kaldi generates with a NN architecture. To do so, the program will need the previous steps of Data preparation and Feature extraction. Then, the NN will take as input the results of those operations and return as output the input of the next step, the Language model.

3.1.3 TensorFlow

TensorFlow

TensorFlow [1] is a multiplatform library for coding a wide variety of machine learning algorithms, implemented mainly in Python though it includes ports in Java, C and Go.

It is based on a set of computations built in the form of a graph, where its direction represents the flow of the data and each node is an operation. The data is generally present in the form of tensors, multidimensional arrays that contain a specific type of data.

The library is implemented so it can be run on GPUs by using the NVIDIA tools CUDA and cuDNN, which increase notably the speed of computation.

3.2 Experiments

Once all the data is prepared, the aim is to build a neural architecture that could substitute the Acoustic Model in Kaldi. This program will receive as input the MFCCs and Filter Banks computed in previous stages and its output will be used for the following stages. The program will train using the 7138 utterances of the TRAIN subset of the database Aurora-4.

In order to measure the performance of the architecture, the *Word Error Rate* (WER) of the whole ASR system will be calculated, which measures the ratio the ratio of incorrect to total words. For that, the network should use a set of utterances that has never seen before, which will be the TEST subset of the database. Therefore, the output will consist in 14 different measures of the WER, along with their mean and standard deviation. For comparison, the mean of the output when using Kaldi alone is 15.14. The aim of the experiments is getting any improvement in this value. WER

Since the dataset has been recorded with two different microphones of different qualities, it is expected that the performance of the network will be noticeably better for the high-quality microphone. In addition, the most probable result is to have lower error rate in the clean environments than in their noisy counterparts.

3.2.1 Deep Neural Networks

First, an approach using Deep Neural Networks will be performed. These would be trained with MFCC features and Filter Banks.

The input of the Network consists on the acoustic features of 11 consecutive frames. Each frame consists on 39 parameters for the MFCCs (13 parameters, 13 Δ and 13 Δ^2) and 41 for the Filter Banks (40 Filter Banks + Energy). Therefore, each input vector has dimension $11 \cdot 39 = 429$ for the MFCCs and $11 \cdot 41 = 451$ for the Filter Banks.

The output of the Network gives the probability of the triphone uttered in the 11 consecutive frames of the input. There is a total of 2316 clusters of triphone combinations.

Each layer consists in a matrix of weights and a vector of biases. Given the characteristics of the input of the network, the dimension of the first weight matrix will be $429 \times w$ for the MFCC or $451 \times w$ for the Filter Banks, where $w \in \mathbb{N}$. Similarly, the dimension of the last weight matrix will be $w \times 2316$, and the dimension of the last bias vector will be also 2316. The rest of weight matrices and vector biases will have dimension $w \times w$ and w respectively. The activation function will be the sigmoid function $\sigma(\vec{x})$ for each layer.

The first experiment will consist on 7 hidden layers (with a total of 9 layers) where $w = 2048$. This architecture will be tested with both the Filter Banks and MFCCs. The WER of the results are shown on the table 3.1.

	30 849 292	30 804 236
TEST dataset	WER (MFCC)	WER (FB)
`airport-1`	10.59	9.10
`airport-2`	23.84	22.21
`babble-1`	10.16	9.38
`babble-2`	25.22	23.50
`car-1`	6.69	5.77
`car-2`	15.54	15.08
`clean-1`	**5.62**	**4.63**
`clean-2`	12.18	9.60
`restaurant-1`	14.01	12.44
`restaurant-2`	**27.82**	25.65
`street-1`	13.06	11.34
`street-2`	26.04	25.01
`train-1`	13.23	11.79
`train-2`	27.48	**26.28**
Mean	16.53	15.13
St. dev.	7.88	7.76

Table 3.1: WER of the Kaldi ASR system with a Deep architecture applied to acoustic features containing MFCCs and Filter Banks. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The number of hidden layers ascends to 7, giving a total of 9 layers. The dimensions of the weight matrix correspond to 2048 neurons where the dimension is not constrained by the characteristics of input and output of the network.

As expected, the best results are achieved in the `clean-1` TEST dataset. This is, a clean signal using a high-quality microphone. Moreover, the results on the datasets in which the data was recorded with an ordinary microphone show a higher error rate than their counterparts, also a predictable result. However, the mean value of the scores of the MFCCs is higher than the 15.14 achieved by using the tools provided by Kaldi, and a similar value on Filter Banks. This implies a further research in more advanced architectures so that the score improves.

3.2.2 Convolutional Neural Networks

After the previous evaluations on Deep Neural Networks, the next approach will cover the use of Convolutional Neural Networks. Since the previous results of the table 3.1 were better for the Filter Banks in the 14 TEST databases, the following experiments will only use this data as input on the network.

From now on, the Neural Network will perform convolution operations, its input will now be two-dimensional. Therefore, the previous input of 451-dimensional vectors will be separated in a grid formed by the 41 acoustic features and the 11 frames, thus having dimension 41×11. Then, a series of convolution and pooling operations will be performed. The convolutions, after performing the operation, will encode the information in different channels. This is, each convolution will have a number of input channels and a number of output channels.

The first experiment will have this architecture:

- A convolution by a 3×3 filter from 1 to 64 channels followed by a ReLU.
- A convolution by a 3×3 filter from 64 to 64 channels followed by a ReLU.
- A max pooling operation with dimension 2×1.
- A convolution by a 3×3 filter from 64 to 128 channels followed by a ReLU.
- A convolution by a 3×3 filter from 128 to 128 channels followed by a ReLU.
- A max pooling operation with dimension 2×1.
- A convolution by a 3×3 filter from 128 to 128 channels followed by a ReLU.
- A convolution by a 3×3 filter from 128 to 128 channels followed by a ReLU.
- A max pooling operation with dimension 2×2.
- A convolution by a 3×3 filter from 128 to 256 channels followed by a ReLU.
- A convolution by a 3×3 filter from 256 to 256 channels followed by a ReLU.
- A max pooling operation with dimension 2×2.
- The output is flattened into a vector.
- A dense layer with 1024 neurons.

The results of the network are in the table 3.2.

	3 812 172
TEST dataset	WER
`airport-1`	7.66
`airport-2`	18.23
`babble-1`	7.66
`babble-2`	19.73
`car-1`	4.91
`car-2`	10.67
`clean-1`	**4.18**
`clean-2`	8.13
`restaurant-1`	9.47
`restaurant-2`	20.38
`street-1`	9.38
`street-2`	20.96
`train-1`	10.48
`train-2`	**22.04**
Mean	12.42
St. dev.	6.38

Table 3.2: WER of the Kaldi ASR system with a Convolutional Neural Network applied to Filter Banks. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

With this new architecture, we managed to obtain a more reliable ASR system than the one predefined by Kaldi. The WER score dropped from 15.14 to 12.42, with an improvement on 2.72 points.

After this experiment, a batch normalisation followed by a ReLU activation function will be applied before each pooling. Thus, the ReLU after each convolution will be removed. The results are displayed in the table 3.3.

	3 812 172	3 812 172
TEST dataset	WER (–BN)	WER (+BN)
airport-1	7.66	7.73
airport-2	18.23	18.87
babble-1	7.66	7.60
babble-2	19.73	19.78
car-1	4.91	4.76
car-2	10.67	9.99
clean-1	**4.18**	**4.39**
clean-2	8.13	6.86
restaurant-1	9.47	9.38
restaurant-2	20.38	21.71
street-1	9.38	9.70
street-2	20.96	**22.14**
train-1	10.48	9.23
train-2	**22.04**	22.10
Mean	12.42	12.45
St. dev.	6.38	6.81

Table 3.3: WER of the Kaldi ASR system with a Convolutional Neural Network applied to Filter Banks, featuring or not Batch Normalisation. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

It can therefore be stated that applying a Batch Normalisation in this way does not make a very noticeable change in this particular problem.

Finally, Dropouts will be added after the 2nd and 4th ReLU activation functions. The results are shown in the table 3.4

	3 812 172	3 812 172
TEST dataset	WER (–Dropout)	WER (+Dropout)
`airport-1`	7.66	7.19
`airport-2`	18.23	17.22
`babble-1`	7.66	7.90
`babble-2`	19.73	18.98
`car-1`	4.91	5.36
`car-2`	10.67	11.25
`clean-1`	**4.18**	**4.54**
`clean-2`	8.13	7.51
`restaurant-1`	9.47	8.97
`restaurant-2`	20.38	19.93
`street-1`	9.38	8.80
`street-2`	20.96	20.01
`train-1`	10.48	9.42
`train-2`	**22.04**	**20.40**
Mean	12.42	11.96
St. dev.	6.38	5.95

Table 3.4: WER of the Kaldi ASR system with a Convolutional Neural Network applied to Filter Banks, featuring or not Dropouts. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

As exposed in the results, the use of dropouts decreases the accuracy of the ASR system for the clean environment but the overall mean taking into account the noisy conditions is improved.

3.2.3 Recurrent Neural Networks

After experimenting with CNNs, the next step will be introducing a new recurrent architecture. The first step will cover the use of RNNs alone, and afterwards an architecture using CNNs followed by RNNs.

As stated before, this phase will be covering the use of different RNN architectures. The different experiments will include:

- 1 layer of 256 LSTMs + 1 dense layer.
- 1 layer of 512 LSTMs + 1 dense layer.

- 3 layers of 256 LSTMs + 1 dense layer.
- 3 layers of 256 LSTMs with 0.85 of forget bias + 1 dense layer.
- 3 layers of 512 LSTMs + 1 dense layer.
- 3 layers of 256 LSTMs + 3 layers of 512 LSTMs + 1 dense layer.
- 3 layers of 512 GRUs + 1 dense layer.
- 3 layers of 256 GRUs + 2 dense layer.
- 3 layers of 256 NASes + 1 dense layer.
- 3 layers of 256 layer-normalising LSTMs + 1 dense layer.
- A BiLSTM composed by 2 layers of 256 LSTMs forward and backward + 1 dense layer.
- A BiLSTM composed by 2 layers of 512 LSTMs forward and backward + 1 dense layer.

The results can be seen in the tables 3.5, 3.6 and 3.7.

	6 829 324	14 180 620	7 879 948	7 879 948
TEST dataset	1×256 LSTM	1×512 LSTM	3×256 LSTM	3×256 LSTM (+FB)
`airport-1`	17.78	10.63	8.65	10.33
`airport-2`	34.49	24.64	21.43	23.43
`babble-1`	16.57	10.24	8.71	9.43
`babble-2`	36.41	26.06	21.63	24.19
`car-1`	11.40	6.65	5.01	6.50
`car-2`	27.52	17.26	13.58	15.36
`clean-1`	**9.49**	**5.70**	**4.61**	**5.29**
`clean-2`	19.75	11.34	8.24	10.48
`restaurant-1`	21.95	13.30	11.58	12.59
`restaurant-2`	**38.74**	26.84	23.86	26.49
`street-1`	20.01	12.01	10.84	12.07
`street-2`	38.58	26.51	23.05	25.99
`train-1`	20.23	12.91	10.13	12.52
`train-2`	38.30	**29.24**	**24.79**	**28.58**
Mean	25.09	16.67	**14.01**	15.95
St. dev.	10.41	8.25	7.33	8.04

Table 3.5: (I) WER of the Kaldi ASR system with different recurrent architectures. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

	18 379 020	20 175 116	1 046 832	2 836 748
TEST dataset	3×512 LSTM	3×256 $+ 3 \times 512$ LSTM	3×512 GRU $+ 1$ DL	3×256 GRU $+ 2$ DL
`airport-1`	10.63	11.66	29.12	22.36
`airport-2`	24.64	24.96	46.05	37.64
`babble-1`	10.24	11.26	34.62	22.32
`babble-2`	26.06	26.25	51.35	39.29
`car-1`	6.65	7.21	15.92	13.64
`car-2`	17.26	16.10	36.56	32.45
`clean-1`	**5.70**	**5.55**	**12.37**	**11.56**
`clean-2`	11.34	10.82	26.75	25.07
`restaurant-1`	13.30	13.36	43.64	28.81
`restaurant-2`	26.84	27.67	58.58	43.25
`street-1`	12.01	12.93	42.41	26.99
`street-2`	26.51	27.42	**61.87**	**45.56**
`train-1`	12.91	14.25	41.79	27.55
`train-2`	**29.24**	**29.55**	59.31	45.15
Mean	16.67	17.07	40.02	30.12
St. dev.	8.25	8.29	15.35	10.97

Table 3.6: (II) WER of the Kaldi ASR system with different recurrent architectures. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

	9 229 580	7 884 556	15 231 244	34 654 476
Test dataset	3 × 256 NAS	3 × 256 LSTM (+Norm)	2 × 256 BiLSTM	2 × 512 BiLSTM
`airport-1`	16.63	10.84	10.35	11.47
`airport-2`	32.08	27.29	23.91	25.56
`babble-1`	15.97	11.43	9.88	10.57
`babble-2`	32.19	28.92	25.01	27.52
`car-1`	9.73	8.14	6.58	7.60
`car-2`	21.63	19.69	15.73	18.59
`clean-1`	**8.16**	**7.12**	**5.38**	**5.92**
`clean-2`	15.06	12.40	10.31	11.99
`restaurant-1`	19.07	14.42	12.52	13.36
`restaurant-2`	33.66	30.82	27.33	29.63
`street-1`	16.61	13.77	12.59	13.06
`street-2`	33.55	31.35	26.77	29.48
`train-1`	17.34	14.20	12.42	14.29
`train-2`	**34.39**	**32.32**	**28.92**	**30.86**
Mean	21.86	18.76	16.26	17.85
St. dev.	9.38	9.34	8.29	8.89

Table 3.7: (III) WER of the Kaldi ASR system with different recurrent architectures. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted. The architecture of the network is previously described.

As shown in the data, the best performance is achieved by using 3 layers of 256 LSTM cells, both for the clean environment and the rest of noisy signals. The poor results achieved by the GRUs could be due to the small number of parameters used for their training. Anyway, instead of investigating further on GRUs, the use of LSTMs will be preferred since a better result than naïve Kaldi has been achieved with them.

After these experiments, the task will be focused on joining the convolutional and recurrent architectures, applying the best CNN architecture followed by the best RNN architecture: this is, the Dropout CNN followed by the 3 layers of 256 LSTMs. The results are in the following table:

	3 812 172	7 879 948	9 781 071
TEST dataset	CNN	RNN	CNN +RNN
airport-1	7.19	8.65	7.42
airport-2	17.22	21.43	18.49
babble-1	7.90	8.71	7.96
babble-2	18.98	21.63	20.72
car-1	5.36	5.01	4.76
car-2	11.25	13.58	10.48
clean-1	**4.54**	**4.61**	**4.13**
clean-2	7.51	8.24	6.63
restaurant-1	8.97	11.58	9.58
restaurant-2	19.93	23.86	22.32
street-1	8.80	10.84	9.25
street-2	20.01	23.05	21.99
train-1	9.42	10.13	9.36
train-2	**20.40**	**24.79**	**22.45**
Mean	11.96	14.01	12.54
St. dev.	5.95	7.33	6.98

Table 3.8: WER of the Kaldi ASR system when using the best CNN architecture, the best RNN architecture and both. The number on top of each architecture represents the number of trainable parameters. The best and worst results are highlighted.

Although applying the RNN after the CNN improves the result of the WER in environments without external noise, the overall score taking into account the signals with noise gets worse results after than with CNNs alone. This is, in noisy conditions it is preferable to use only a convolutional architecture instead of complementing it with a recurrent block afterwards.

Chapter 4

Conclusions and Future Work

At the very beginning, the first experiments served to test the performance of a deep neural network on the Acoustic Model of Kaldi. This lead to the conclusion that Filter Banks behaved in a more efficient way than MFCC scores, since a lower error was achieved. Nevertheless, the results were practically similar to the ones that can be expected from Kaldi alone, and therefore a much deeper research should be done in more advanced neural architectures.

Afterwards, a much greater improvement could be achieved by using convolutional neural architectures. Once applied the dropout on various steps on the network, the WER of 11.96 was accomplished. This means that the previous performance of the Kaldi ASR system was improved in 3.17 points.

Finally, the use of recurrent networks was studied with a great number of architectures and models. The results obtained suggested that from every experiment, the best performant design was that involving three layers of 256 LSTM cells, though its performance is not comparable to the one of the CNNs with its 14.01 WER, still better than Kaldi alone. In the last phase of the research, both convolutional and recurrent structures were combined in a single unit. The results worsen when comparing them with the mean achieved by CNNs alone, but when checking the score on the clean environment they yielded even finer results than the previous experiments, with a total of 4.13 WER compared to the 4.54 on CNNs alone.

Sumarising, and in view of the results achieved with the experiments, it can be stated that the best architecture for speech signals featuring noisy conditions is the one involving CNNs. Nevertheless, it should not be forgotten that for the clean environment a better outcome can be achieved if the architecture is complemented with a recurrent network. Thus, a further analysis on more complex convolutional architectures could lead to more improvement of the ASR system, since these have been seen to operate in a better way than the rest in overall.

It should be made clear that the number of networks that could have been tested is enormous. Therefore, better results might be achieved by implementing any of the architectures out of this work. Since in these recent years the progress made in the areas of machine learning and especially on neural networks has escalated notably, it might be the case that in future times the architectures tested in this work are overcome by new programs whose performances outpass this investigation.

In any case, in terms of immediate future, the first attempts could lead to the implementation of these tested neural architectures in Kaldi, in accordance with the *flavour* of this toolkit. This means a further implementation in C++ following the guidelines found in their documentation.

Besides, a further analysis on the features fed to the network could also provide a more interesting outcome. A different set of input parameters, for example increasing the number of Filter Banks or MFCCs or performing a previous normalisation of the data, may result on a better performance of the overall network.

Bibliography

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., AND PENN, G. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on* (2012), IEEE, pp. 4277–4280.

[3] CHO, K., VAN MERRIENBOER, B., GÜLÇEHRE, Ç., BOUGARES, F., SCHWENK, H., AND BENGIO, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR abs/1406.1078* (2014).

[4] DENG, L., AND YU, D. Deep learning: Methods and applications. Tech. rep., Microsoft Research, May 2014.

[5] GANCHEV, T., FAKOTAKIS, N., AND KOKKINAKIS, G. Comparative evaluation of various MFCC implementations on the speaker verification task. In *Proceedings of the SPECOM* (2005), vol. 1, pp. 191–194.

[6] GERS, F. A., SCHMIDHUBER, J., AND CUMMINS, F. Learning to forget: Continual prediction with lstm. *Neural Computation 12* (1999), 2451–2471.

[7] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] GRAVES, A., MOHAMED, A.-R., AND HINTON, G. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (2013), IEEE, pp. 6645–6649.

[9] GREGOR, K., DANIHELKA, I., GRAVES, A., AND WIERSTRA, D. DRAW: A recurrent neural network for image generation. *CoRR abs/1502.04623* (2015).

[10] HAYKIN, S. *Neural Networks: A Comprehensive Foundation.* Tom Robbins, 1999.

[11] HINTON, G., DENG, L., YU, D., DAHL, G. E., MOHAMED, A.-R., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T. N., ET AL. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine 29*, 6 (2012), 82–97.

[12] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[13] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR abs/1502.03167* (2015).

[14] JURAFSKY, D., AND MARTIN, J. H. *Speech and Language Processing (2nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[15] KALCHBRENNER, N., GREFENSTETTE, E., AND BLUNSOM, P. A convolutional neural network for modelling sentences. *CoRR abs/1404.2188* (2014).

[16] KUMAR, A., IRSOY, O., ONDRUSKA, P., IYYER, M., BRADBURY, J., GULRAJANI, I., ZHONG, V., PAULUS, R., AND SOCHER, R. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of The 33rd International Conference on Machine Learning* (New York, New York, USA, 20–22 Jun 2016), M. F. Balcan and K. Q. Weinberger, Eds., vol. 48 of *Proceedings of Machine Learning Research*, PMLR, pp. 1378–1387.

[17] MUDA, L., BEGAM, M., AND ELAMVAZUTHI, I. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *CoRR abs/1003.4083* (2010).

[18] PARIJAR, N., AND PICONE, J. Aurora working group: DSR front end LVCSR evaluation AU/384/02. Tech. rep., Institute for Signal and Information Processing, Department of Electrical and Computer Engineering, Mississippi State University, December 2002.

[19] POVEY, D., GHOSHAL, A., BOULIANNE, G., BURGET, L., GLEMBEK, O., GOEL, N., HANNEMANN, M., MOTLICEK, P., QIAN, Y., SCHWARZ, P., SILOVSKY, J., STEMMER, G., AND VESELY, K. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding* (Dec. 2011), IEEE Signal Processing Society. IEEE Catalog No.: CFP11SRW-USB.

[20] SAINATH, T. N., MOHAMED, A.-R., KINGSBURY, B., AND RAMABHADRAN, B. Deep convolutional neural networks for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (2013), IEEE, pp. 8614–8618.

[21] SAK, H., SENIOR, A., AND BEAUFAYS, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association* (2014).

[22] SAON, G., KUO, H. J., RENNIE, S. J., AND PICHENY, M. The IBM 2015 english conversational telephone speech recognition system. *CoRR abs/1505.05899* (2015).

[23] SCHUSTER, M., AND PALIWAL, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing 45*, 11 (1997), 2673–2681.

[24] SRIVASTAVA, N., HINTON, G. E., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research 15*, 1 (2014), 1929–1958.

[25] SUN, Y., LIANG, D., WANG, X., AND TANG, X. Deepid3: Face recognition with very deep neural networks. *CoRR abs/1502.00873* (2015).

[26] TIWARI, V. Mfcc and its applications in speaker recognition.

[27] VAN DEN OORD, A., DIELEMAN, S., ZEN, H., SIMONYAN, K., VINYALS, O., GRAVES, A., KALCHBRENNER, N., SENIOR, A. W., AND KAVUKCUOGLU, K. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499* (2016).

[28] XIONG, W., DROPPO, J., HUANG, X., SEIDE, F., SELTZER, M., STOLCKE, A., YU, D., AND ZWEIG, G. The microsoft 2017 conversational speech recognition system. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on* (2017), IEEE, pp. 5255–5259.

[29] ZHANG, Y., CHAN, W., AND JAITLY, N. Very deep convolutional networks for end-to-end speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on* (2017), IEEE, pp. 4845–4849.

[30] ZHANG, Y., PEZESHKI, M., BRAKEL, P., ZHANG, S., LAURENT, C., BENGIO, Y., AND COURVILLE, A. C. Towards end-to-end speech recognition with deep convolutional neural networks. *CoRR abs/1701.02720* (2017).

[31] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *CoRR abs/1611.01578* (2016).