DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

A Semi-Supervised Approach to Automatic Speech Recognition Training For the Icelandic Language

ATLI THOR SIGURGEIRSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A Semi-Supervised Approach to Automatic Speech Recognition Training for the Icelandic Language

ATLI THOR SIGURGEIRSSON

Master's in Computer Science
Date: July 11, 2019
Supervisor: Dr. Mats Nordahl
Examiner: Dr. Olov Engwall
School of Electrical Engineering and Computer Science
Host company: Reykjavik University,
principal supervisor: Dr. Jón Guðnason
Swedish title: En semi-övervakad metod för automatisk språkigenkänning för det isländska språket

Abstract

Recent advances in deep learning have enabled certain systems to approach or even achieve human parity in certain tasks, including automatic speech recognition. These new state-of-the-art speech recognition models are most often dependent on vast amounts of expensive high-quality labeled speech data for supervised training. In this work, we consider ways of leveraging unlabeled data for unsupervised training to reduce this costly data dependency. Six altered models are compared to a baseline sequence-to-sequence speech recognition model under three different low resource conditions. We show that for all three conditions, a semi-supervised approach surpasses the quality of the baseline.

Sammanfattning

Nya framsteg inom djupinlärning har gjort det möjligt för vissa system att närma sig eller till och med uppnå mänsklig paritet i vissa uppgifter, inklusive automatisk taligenkänning. Dessa nya state-of-the-art-metoder är oftast beroende av stora mängder av dyr, högkvalitativ och märkt data för övervakad träning. I det här arbetet undersöker vi olika sätt att utnyttja omärkt data för oövervakad träning för att minska beroendet av dyr data. Sex modifierade modeller jämförs under låga resursförhållanden med en sekvens-till-sekvens taligenkänningsmodell som baslinje. Vi visar att för alla tre förhållanden överträffar en delvis övervakad träning baslinjens kvalitet.

Acknowledgements

I would like to thank my supervisor, Dr. Mats Nordahl, for his precious guidance and support during the writing of this work. I would like to state my sincere gratitude for the opportunities that Dr. Jón Guðnason, the principal supervisor of this work, has granted me and his vital help and skillful suggestions. I would additionally like to thank the members of the Language and Voice Lab at Reykjavik University for their technical and academic guidance.

Finally, I would like to express my warmest gratitude to my parents, Sigurgeir and Þóra, and my sisters, Fanney and Karen, for their indispensable support and all the great times we have spent together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The research area of this work is in the field of natural language processing (NLP), specifically in speech recognition. automatic speech recognition (ASR) models aim to translate spoken language into the corresponding text. As technology has advanced and data has become more available, the progression of ASR technology has been astounding. Today, ASR is used widely and recently has become an integral part of how we communicate with our devices, by voice. The state-of-the-art ASR models today are built on deep neural networks (DNNs) and have reached human parity [1].

The quality of such models is highly coupled with the amount of data and processing power available. Neural ASR models are trained on hundreds of hours of transcribed speech, a source of data that for many smaller languages is hard to come by. Even more recently, training ASR models end-to-end has become a popular research topic. End-to-end model combine every module that constitutes traditional ASR modelling, described in Section 2.3, into a single model which is trained with a single loss function. Such models often require even more data. An example of an end-to-end ASR model is the *Listen, Attend and Spell* model [2] or LAS for short. LAS was trained on a very large corpus consisting of 2000 hours of hand transcribed data.

Reducing this resource need enables cheaper and faster development of ASR for many languages that as of today do not have access to state-of-the-art ASR. One way of reducing resource cost is to take advantage of both unlabeled speech and text data that is widely available at a low cost. Unsupervised training methods could be used to augment existing neural ASR models to reduce their necessity for parallel corpora by making use of unlabeled data.

This work will be conducted at the language and voice lab of *Cadia* [1] and implemented specifically for the Icelandic language.

1.1 Research Question

The objective of this work is to investigate whether unsupervised training methods can be used to reduce the need for expensive resources that the state-of-the-art neural ASR models require. The ASR model will be trained and evaluated on Icelandic, using open sourced Icelandic data [2]. The research question is stated below.

Can unlabeled data be leveraged using unsupervised training methods to improve the performance of a baseline neural ASR model for the Icelandic language, trained on a limited amount of transcribed speech data?

1.2 Ethical and Social Relevance

As technology has advanced, so have our ways of interacting with it. Many digital services have now enabled direct conversational based controls, the Google Assistant, Amazon's Alexa and Apple's Siri to name a few. These new ways of interaction are however only available for the very largest languages, caused by the extensive development costs of such systems. Amazon's Alexa only supports 7 languages according to development documentation [3], to take an example. According to [3], only 5% of the 7,000 languages spoken at the time will survive the ever stronger digital transition of our societies. The rate of languages dying has never been higher and this rate has been contributed to multiple different factors.

According to [4], the factors threatening the sustainability of the Icelandic language specifically are both societal and technological. Societal threats includes the significant boost of tourism in Iceland which has led to more foreign nationals being able to take over service jobs in Iceland. Technological threats include the increased usage of smartphones and tablets, media services such

[1] Iceland's interdisciplinary research center in artificial intelligence spanning the School of Computer Science, the School of Science and Engineering, and the School of Business at Reykjavik University.

[2] https://www.malfong.is/?lang=en

[3] https://developer.amazon.com/docs/

as YouTube and Netflix which mainly serve content in English and voice controlled devices. What makes these technological threats harder to deal with is the fact that ever younger kids interact with smart devices, 58% of 3-5 year old children with access to smartphones or tablets started using the devices when they were less than 2 year old. Speech transcription is important to make dif-

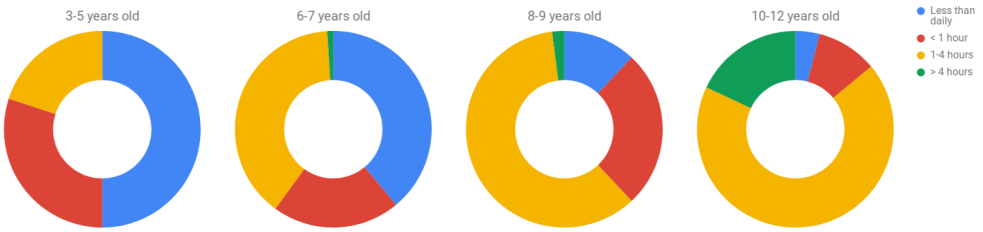


Figure 1.1: The usage of smartphones and tablets by Icelandic children by age groups (Based on [4])

ferent types of media available for everyone. For smaller societies, the cost of providing subtitles for televised media can prove to be economically unsustainable, even for socially important broadcasts such as news or large TV events. This results in the media being inconsumable for certain groups of society, such as those hard of hearing, deaf individuals and immigrants to name a few. The development of ASR for small societies can make the production of subtitles more economically sustainable.

Constructing data-dependent models, such as the one described in this work, requires careful ethical considerations with regard to the data used. The data used has to be legally sourced and when the data is sourced from regular citizens it should be done so privately and with full consent. The transcribed data used in this work required donating participants to sign a participation agreement and the donated data of each user is linked to an anonymous id [5].

Bias in training data is another source of ethical concerns. If the number of utterances spoken by men in the training data far outnumbers the number of utterances spoken by women, it is likely that the trained ASR model will perform better overall on new utterances from men. In a similar way the age of participants donating their voice and geographical location are factors that can bias the training data in a negative way, to name a couple of examples. The transcribed dataset used in this work consists of 63,215 (51.3%) sentences spoken by 303 (53.8%) male participants and 60,012 (48.7%) sentences spoken by 260 (46.2%) female participants of varying age. The geographical location of participants is often highly coupled with the participant's dialect. For

many languages, the different dialects of a language is a source of bias that is hard to account for. Regional dialects exist in Iceland but the difference in pronunciation between each one is very limited [6].

In the broader perspective, reducing ASR development cost is especially beneficial for poor societies. This is particularly true for models that are trained end-to-end which normally require much more extensive training. The estimated cost of training NLP models can be compared using the cloud computing cost of these models if hosted by the same provider. In [7], the cost of 5 different language models is compared, ranging from $41 up to $3,201,722. The cost correlates well with the number of trainable parameters that the models consists and the number of training iterations needed for convergence. This high training cost puts some complex state-of-the-art NLP models out of reach for poorer and less developed societies.

In [7], the carbon footprint produced by training the same 5 models is additionally compared. The power consumption of the models compared ranges from approximately 500W up to 12,000W, corresponding with estimated 26-626,155 pounds of CO_2 emitted based on the energy composition of main cloud computing providers. For comparison, the average carbon footprint of the whole lifetime of a car including fuel consumption corresponds with estimated 126,000 pounds of CO_2. Reducing the training cost of complex ASR models could therefore theoretically have the added benefit of reducing the large carbon footprint produced by such models.

1.3 Outline

Chapters 2-3 are dedicated to discussion on background literature and research and related work focusing on automatic speech recognition. In Chapter 2 we roughly describe the history of ASR development both in general and for the Icelandic language. A detailed description of ASR development follows the historical background, from feature extraction to decoding. Chapter 3 contains a review of state-of-the-art ASR models with careful consideration of sequence-to-sequence models.

Chapters 4-5 are dedicated to methodology and results. In Chapter 4 we describe each component of the proposed model and some additional necessary background study. The training procedure of each component of the proposed model is additionally described in detail. Chapter 5 gives a brief overview of training and model hyperparameters and other technical and design choices relevant to the performed experiments.

Chapter 6 gives an in-depth description of the result of training each model

component as well as a comparison between differently configured models under different resource conditions. Finally, we discuss future work and conclusions in Chapter 7.

Chapter 2

Automatic Speech Recognition

Automatic speech recognition is a well studied field which has a relatively long history. The technological foundation that ASR rests on is very extensive and here a description of some of the important objectives of ASR is given. Special consideration is made for feature extraction, acoustic modelling and language modelling which all are important objectives for the methodology of this work.

2.1 A Brief History of ASR

The first documented speech recognizer was *Audrey*, a fully analog digit recognizer developed at Bell Laboratories in 1952 [8]. The Audrey ASR achieved respectable accuracy, but only if it was specially adapted to the speaker. The very limited potential of ASR at the time made them economically unattractive. Audrey, for example, required substantial power and maintenance and would in most cases be less efficient at entering digits then by simply dialing in the digits.

It was not until the 1980s that a significant step forward was taken in the field of ASR when ASR research shifted to more statistical approaches mainly driven by the research of hidden Markov models (HMMs). [9]. It was thought that a speech signal could be well modeled both in the frequency domain, using a Gaussian mixture model, and the temporal domain using HMMs [10]. This turned out to be the case and using HMMs became the standard procedure to develop ASR, see Section 2.3.2.

Another statistical approach was reintroduced in the 1980s for acoustic modeling in ASR, the artificial neural networks (ANNs). Early attempts with

ANNs focused on recognizing a small corpus of phonemes [1] or words [11]. At the time, there was no good way of dealing with temporal data, like speech. Recurrent neural networks (RNNs) were introduced shortly after as a solution to this temporal issue. A combination of HMMs and RNNs became a prominent way of doing statistical acoustic modeling [12].

In recent years, ASR development has been dominated by deep recurrent neural networks, edging past human parity using very complex models [1], or achieving promising results with more straight forward models [2] and vast amounts of data.

ASR development has always been heavily dependent on labeled data and supervised learning. Many efforts have been made to bring unsupervised ASR learning closer to reality, [13], [14], [15] to name a few, but so far there exists no way of connecting the speech domain to the text domain without supervision. These efforts are very much ongoing, in the hopes of reducing the vast needs of state-of-the-art ASR models.

2.2 ASR development in Iceland

Icelandic language technology was virtually non-existent at the turn of the century [16]. There was very limited academic interest in language technology, such that no language technology courses were taught at Icelandic universities, nor was any research being carried out in the field.

This situation has changed dramatically in recent years as more and more have voiced their concern that the Icelandic language would become less relevant as the world becomes more connected. In terms of speech recognition, an isolated word speech recognizer, or the Icelandic speech recognition project *Hjal* [17], was created in 2004. The Hjal recognizer managed a 97% recognition rate, but no ASR existed for continuous speech at the time.

In recent years, larger Icelandic speech corpora have been created, most prominently the *Almannaromur* corpora [18]. Almannaromur is a large multi-speaker corpus of utterances donated by over 500 participants. A smaller corpora was collected in 2017 with the *Eyra* acoustic data collecting tool-chain [19].

Additionally, a large corpus of transcribed speeches made at the Icelandic parliament, the Althing, has been made open for use by the public. Promising results have been achieved using the Althing ASR corpus in [20], where a word error rate, see Section 2.3.6, of 14.76% was obtained using a long-short term

[1] A unit of sound that distinguishes a particular word from another.

memory (LSTM) recurrent neural network.

2.3 Technical Components of ASR

The goal of an ASR model is to find the most probable word sequence, $\mathbf{W}^*$, given the input speech signal. If each word was always pronounced in the same way, finding the correct word given the speech would be a relatively simple task but this is of course not the case and words are almost never pronounced exactly the same. To take these different pronunciations into consideration, ASR models are largely statistical models.

The input of an ASR model is the speech signal, represented as a sequence of acoustic feature vectors, $\mathbf{X}$. We can then formulate the most probable word sequence with

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmax}}\, P(\mathbf{W}|\mathbf{X}) \tag{2.1}$$

By applying Bayes' rule we can show that the posterior probability of the word sequence, given the speech signal is

$$P(\mathbf{W}|\mathbf{X}) = \frac{P(\mathbf{X}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{X})} \tag{2.2}$$

And using both 2.1 and 2.2 we can show that

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmax}}\, P(\mathbf{X}|\mathbf{W})P(\mathbf{W}) \tag{2.3}$$

The conditional likelihood $P(\mathbf{X}|\mathbf{W})$ is referred to as the acoustic model, and the prior probability of the word sequence is referred to as the language model (LM), $P(\mathbf{W})$.

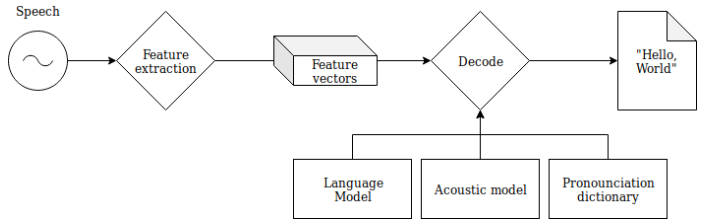


Figure 2.1: Typical components of ASR

Figure 2.1 shows the common building blocks of continuous ASR, of which some are described in detail in the following sections.

2.3.1 Feature Extraction

The goal of speech feature extraction is to represent the original speech waveform in such a way that the loss of information that tells two phonemes apart is minimized [21] at the same time as the dimensionality of the signal is reduced. The speech signal carries a lot of information that is not relevant to the spoken utterance, such as environmental sounds and noise. We therefore aim for the extracted features to ignore the irrelevant content of the signal and emphasize the content that makes detection of different phones easier.

A very common representation of the speech signal in the ASR domain are the Mel-frequency cepstral coefficients (MFCCs) [22]. The derivation of MFCCs varies between implementations but mainly consists of three important steps which are discussed later in this section.

Sound waves are pressure waves transmitted through a medium and are measured by the pressure the sound wave applies to a surface like a diaphragm in a microphone or the eardrum in the middle ear [23]. The square of the *amplitude* of the signal is proportional to the energy that the speech signal carries and therefore proportional to the pressure applied by the sound waves of speech. A speaking person produces a continuous analog *speech signal*. A pure tone [2] waveform is shown in the time domain in Figure 2.2. A recorded

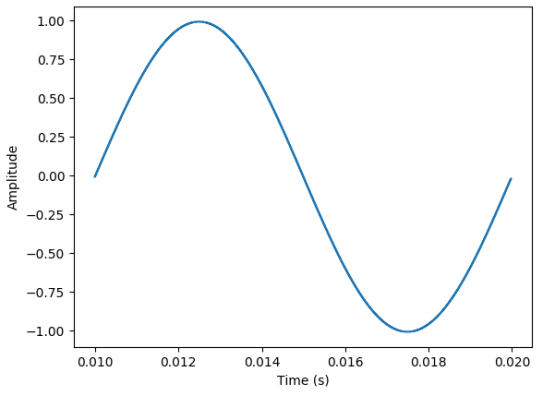


Figure 2.2: A 100 Hz *analogue* sine wave

speech signal cannot be stored in its continuous form and the signals we deal with are most often temporally quantized. To convert the continuous value of the analog speech signal to a discrete digital representation we *sample* from the continuous signal. When we sample a speech signal in time we first define

[2] A signal that contains only a single frequency component

the sampling period, T. At each sampling period, we measure the amplitude of the signal. The measured amplitude is digitally stored as a binary number. A bit-depth of 16 is common for storing audio and is the bit depth, meaning the amplitude samples are in the range $[-2^{15}; 2^{15} - 1]$. The sampling rate is given as $f_s = 1/T$. Speech signals need to be sampled thousands of times a second to produce audible results. The sampling rate of data used in this work is 16 kHz which is a widely used sampling rate[5]. It is important to note that the highest frequency we can represent is determined by the sampling frequency. This frequency is referred to as the *Nyquist* frequency, $f_{Nyquist}$, and is equal

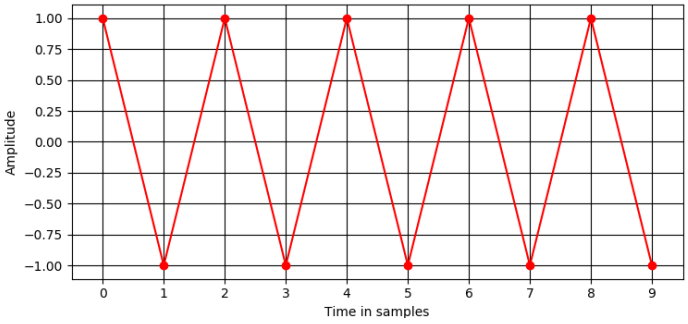


Figure 2.3: A demonstration of the Nyquist frequency

to half the sampling frequency, f_s [24]. To understand why we refer to Figure 2.3. The highest frequency is achieved when the period between the peaks and troughs of the waveform is minimized. For a sampling frequency f_s, this period equals the period of two samples, resulting in $f_{Nyquist} = \frac{1}{2} f_s$. This means for the sampling frequency of 16kHz used in this work, the highest frequency of speech we can represent is equal to 8kHz. The sine waveforms

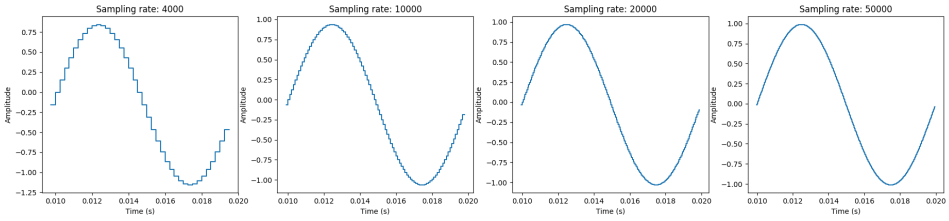


Figure 2.4: Discrete sine waves for different sampling rates [Hz]

for 4 different sampling rates are shown in Figure 2.4. All of the signals shown in Figure 2.4 are discrete and have the step-like nature that is visible for the lowest sampling rate of 4 kHz. Speech signals are however not pure tone but

contain multiple frequency components. A more complex waveform is shown in Figure 2.5 which is made of 4 sine waves of different frequencies. . The

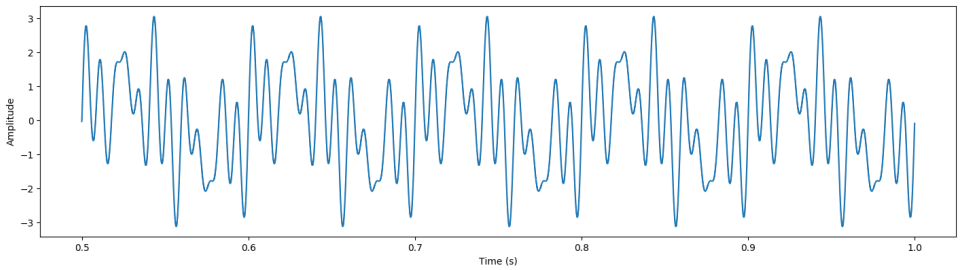


Figure 2.5: A signal made up of multiple frequencies

frequency components of speech are important characteristics to consider for feature extraction and therefore seek to decompose the speech signal into the underlying frequency components.

The First step of extracting MFCCs is the application of the discrete Fourier transformation (DFT) to obtain the frequency domain representation and the power spectrum [3] of the speech signal. When we represent the power spectrum over the whole signal as a function of time it is referred to as the *spectrogram* of the signal. To determine the spectrum of the signal we use Fourier analysis.

The *Fourier principle* tells us that any periodic signal can be broken into its sum of constituent simple[4] frequency components. The *Fourier transform* is used to transform a signal sampled in time into a signal sampled in frequency. The continuous Fourier transform of a signal $s(t)$ at a certain frequency f is given as

$$S(f) = \int_{-\infty}^{\infty} s(t) e^{-i2\pi i f t} dt \tag{2.4}$$

Using Euler's formula we note that

$$e^{-2\pi i f t} = \cos(-2\pi f t) + i \sin(-2\pi f t) \tag{2.5}$$

The magnitude $|S(f)| = \sqrt{\text{Re}(S(f))^2 + \text{Im}(S(f))^2}$ corresponds with the amount of frequency f in the signal and the *argument* of $S(f)$, the angle formed between the positive real axis and the line connecting $S(f)$ to the origin

[3] The distribution of power, the amount of energy transferred per unit time, over the frequency components that a signal consists of.

[4] A pure tone signal, e.g. the sine wave.

(marked as θ in Figure 2.6), corresponds to the phase offset of the frequency f. The inverse Fourier transform is given by

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{i2\pi ft} df \tag{2.6}$$

If we assume the frequency f is in Hz and the time t is in seconds, then we can understand $e^{-2\pi ift}$ as a point in the complex plan rotating around the origin with an angular frequency of $-2\pi ft$ as shown in Figure 2.6 where the point completes f cycles per second. By inspection of Equation 2.5, we can tell that

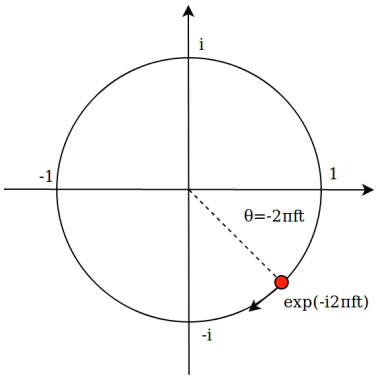


Figure 2.6: Showing $e^{-2\pi ift}$ in the complex plane

as $e^{-2\pi ift}$ rotates, the real part of Equation 2.5 changes as a cosine wave and the imaginary part as a sine wave. We can show this if we consider $e^{-2\pi ift}$ as a function of time as we do in Figure 2.7. By evaluating Equation 2.4 for all possible values of f we produce the *frequency-domain* function $S(f)$ of the original signal, a combination of sines and cosines like the ones shown in Figure 2.7. The frequency domain is shown for a specific signal in Figure 2.8 and it shows how the amount of each frequency component is represented in amplitude in the frequency domain. As the speech signal is sampled we require a discrete version of the Fourier transform. Speech is also a time-variant signal where each phoneme corresponds to a varying number of steps in the signal [23]. Taking a Fourier transform over the whole signal does not yield the local representation needed to represent each phone of the utterance separately. The amplitude envelope of a speech signal is shown in figure 2.9 and demonstrates this need for a local representation. A version of the Fourier transform that is quantized in time is the short-time Fourier transform (STFT). We assume that the frequency components over a small period, Δt, are stationary. We split the signal into overlapping *frames* of length Δt and take the DFT of each

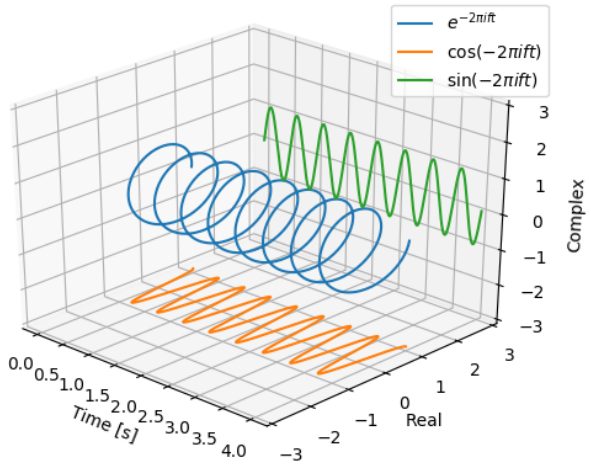


Figure 2.7: The real and imaginary parts of $e^{-2\pi ift}$ for $f = 2$Hz shown as projections on the real and imaginary axis

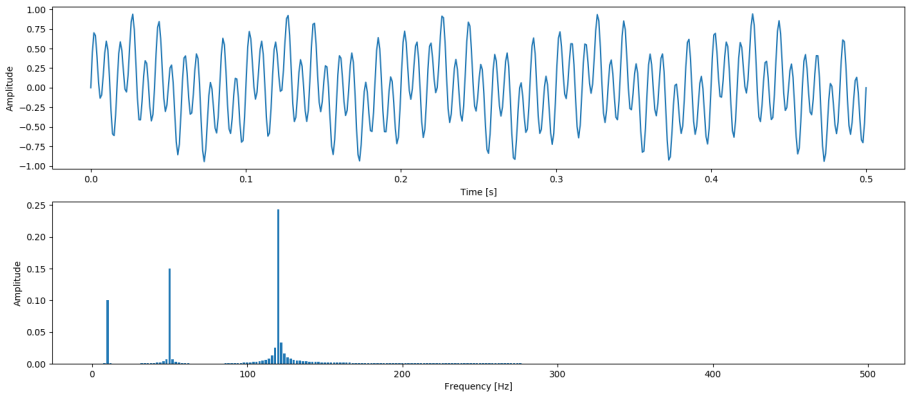


Figure 2.8: A comparison of the time and frequency domain of a signal given by $s(t) = 0.2\sin(10 \times 2\pi t) + 0.3\sin(50 \times 2\pi t) + 0.5\sin(120 \times 2\pi t)$

frame. The signal is split into equal length chunks using a *window function*, i.e a function that is zero-valued everywhere outside a determined interval. A common window function to use is the *Hann* window [25] defined as

$$h(n) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi n}{M-1}\right) \quad 0 \leq n \leq M-1 \tag{2.7}$$

The Fourier transform assumes periodicity in the input and applying the rectangular window does not produce a periodical input leading to unwanted arti-

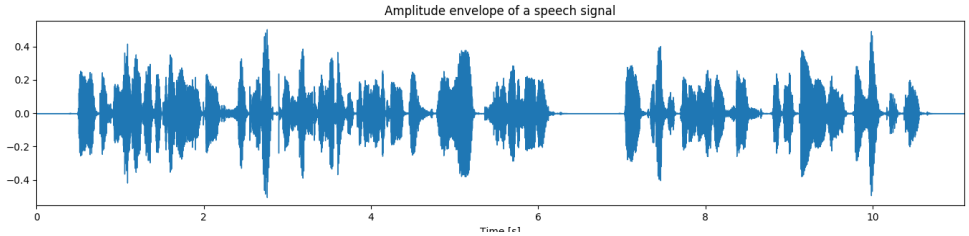


Figure 2.9: Amplitude envelope of the speech signal, the text spoken: "Undir þessa rekstrareiningu fellur allur verslanarekstur á íslandi með mat og sérvöru, auk reksturs í Svíþjóð vegna Bónus og Hagkaups."

facts in the output referred to as *spectral leakage* [26]. Typically the frequency

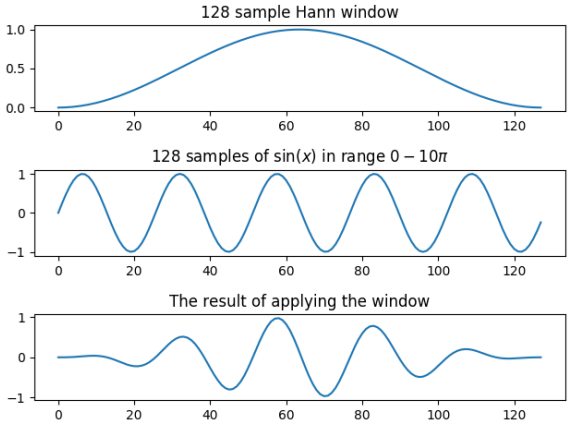


Figure 2.10: A demonstration of applying the Hann function

is quantized as well to speed up computations and the fast Fourier transform (FFT) is used. FFT is any version of the Discrete Fourier Transform (DFT) which can be evaluated in $\Theta(n \log n)$ operations instead of $\Theta(n^2)$ operations and a very common approach is the Cooley-Tukey algorithm [27]. The DFT of a signal $s(t)$ is given as

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-i2\pi k \frac{n}{N}} \quad 1 \leq k \leq K \tag{2.8}$$

where $s_i(n)$ represents the n-th sample of the i-th frame of $s(t)$ and $h(n)$ is the N-sample long Hann window. K is a parameter, representing the number of frequency components in the FFT. Typically, a 512 point FFT is performed but it is important that the number of points is larger than the number of samples per window to avoid loss of precision. The power spectrum of the kth

frequency component of the ith frame is proportional to the corresponding frequency component magnitude

$$P_i(k) = \frac{1}{N}|S_i(k)|^2 \tag{2.9}$$

The next step in the MFCC procedure is to calculate the *Mel* filter bank energies. This is achieved by multiplying the power spectrum in Equation 2.9 with each filter bank and adding up the resulting coefficients [22]. Normally 40 filters are used, distributed on the Mel scale, a non-linear scale which aims to mimic the human perception of sound [28]. In an experiment, 10 participants were asked to determine the perceived equal distance between pitches. As the pitches increase in frequency, the perceived distance between them increases as well [28]. Our way of hearing likely evolved parallel to the evolution of speech production and for that reason is our hearing most responsive to signals in the frequency range of $200 - 5600$Hz [23]. The *basilar membrane* in the *cochlea* of the inner ear varies in the shape and firmness resulting in the membrane vibrating over a larger area for lower frequencies than higher frequencies, resulting in our non-linear perception of pitch with regards to frequency [23].

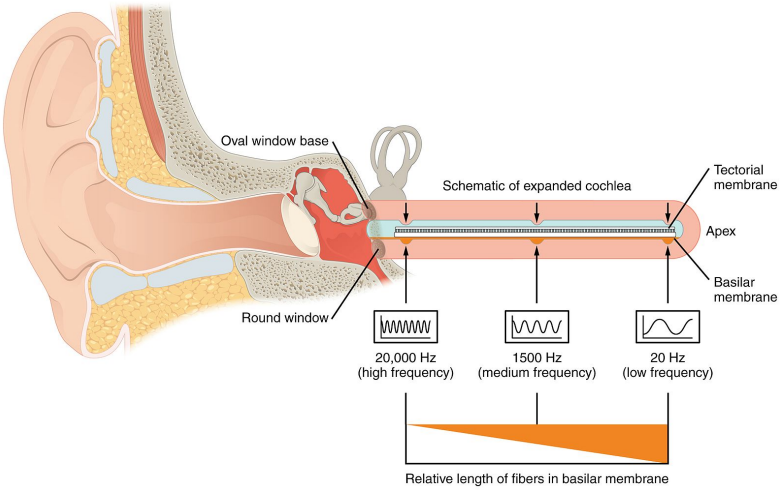


Figure 2.11: The cochlea which is naturally spiraled is shown unrolled. High frequencies are perceived at the base of the cochlea and lower frequencies deeper near the *apex*. (Attribution: OpenStax, CC BY 4.0[6])

[6]https://creativecommons.org/licenses/by/4.0

There is no definitive Mel-scale formula for mapping f Hertz to m Mel but one widely used was proposed by Gunnar Fant [29]

$$m = \frac{1000}{\log 2} \log\left(1 + \frac{f}{1000}\right) \tag{2.10}$$

In this work, the formulation from the MATLAB auditory toolbox of Slaney is used, where the mapping is linear for frequencies below 1 kHz and otherwise logarithmic [30].

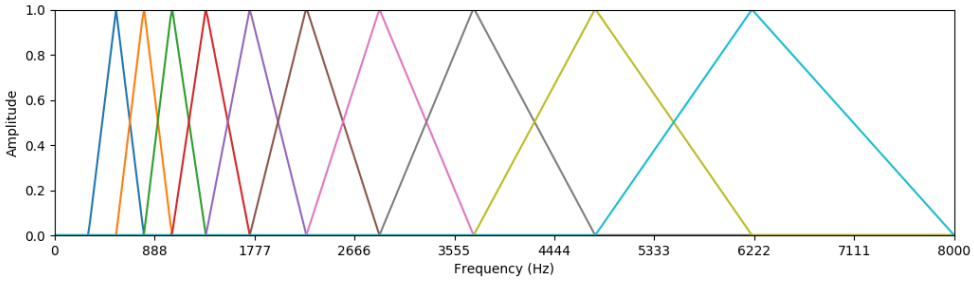


Figure 2.12: A 10 mel filter bank ranging from 300Hz to 8kHz

Each of the filters is triangular where the signal response is the highest at the center of the nearest FFT frequency bin and decreases linearly towards 0 at the frequencies which the adjacent filters give their highest response. This is demonstrated in figure 2.12. Formally, the response of the m-th filter for the k-th FFT frequency bin is given as

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.11}$$

where $f(m)$ is the nearest FFT bin of the m-th filter. An example of a linear spectrogram is compared to a Mel-scale spectrogram in Figure 2.13. Another well known perceptual scale is the *Bark scale* [31]. A common approach to convert f Hertz to b Barks is given as

$$b = 13 \arctan\left(0.76 \frac{f}{\text{kHz}}\right) + 3.5 \arctan\left(\frac{f}{7.5\text{kHz}}^2\right) \tag{2.12}$$

The Bark scale is similar to the Mel scale in that it is linear for certain low frequencies and logarithmic otherwise. The Bark scale is however designed in

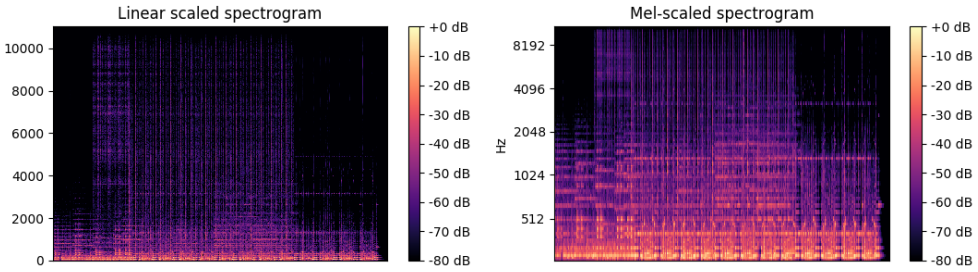


Figure 2.13: A comparison of a linear and a Mel-scale spectrogram

such a way that Barks 1 to 24 cover the first 24 critical bandwidths of hearing, increasing in frequency [23] [31].

The last step of acquiring the MFCCs is to apply the Discrete Cosine Transformation (DCT) <replace with acronym. DCT is applied to decorrelate features in pattern space and proved especially useful for the hidden Markov models with Gaussian mixture modelled emission discussed in Section <add section ref>. Applying the DCT leads to fewer non-zero off-diagonal values in the covariance matrices of the Gaussian mixtures which results in the approximations of the Gaussian mixtures to be much more feasible [21].

Correlated features are less of a problem in the context of deep learning since deep neural networks are capable of modelling the covariance structure of the data without any prior preprocessing. Therefore, the DCT step is commonly omitted when extracting features for DNN ASR modelling to favor higher dimensional features [32].

2.3.2 Acoustic Models

A single phoneme corresponds with multiple consecutive feature vectors. We can formulate the probability of the acoustic model over the whole word sequence $\mathbf{W}$ as a product of phoneme likelihoods instead. Let us assume that the word sequence $\mathbf{w}$ has n phonemes, $w_1, w_2, \ldots w_n$. Then the corresponding feature vector sequence $\mathbf{X} = \mathbf{x_1 x_2} \ldots \mathbf{x_m}, \mathbf{x_i} \in \mathbb{R}^d$ can be broken into equal amount of consecutive groups, n, where each group of feature vectors, X_i corresponds with a single phoneme. Then the likelihood can be written as

$$P(\mathbf{X}|\mathbf{W}) = P(X_1|w_1)P(X_2|w_2)\ldots P(X_n|w_n) \tag{2.13}$$

again, where e.g. the first phoneme corresponds with t_1 feature vectors

$$P(X_1|w_1) = P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t_1}|w_1) \tag{2.14}$$

If however, the case was that each phoneme in the sequence corresponded with a single frame, e.g. $P(X_1|w_1) = P(\mathbf{x_1}|w_1)$, then each of the conditional probabilities in Equation 2.13 could be modeled with a Gaussian distribution, and with enough training samples we could estimate the mean and variance for each conditional probability. For the general case, we need a more dynamic system since each phoneme can correspond to a varying number of feature vectors.

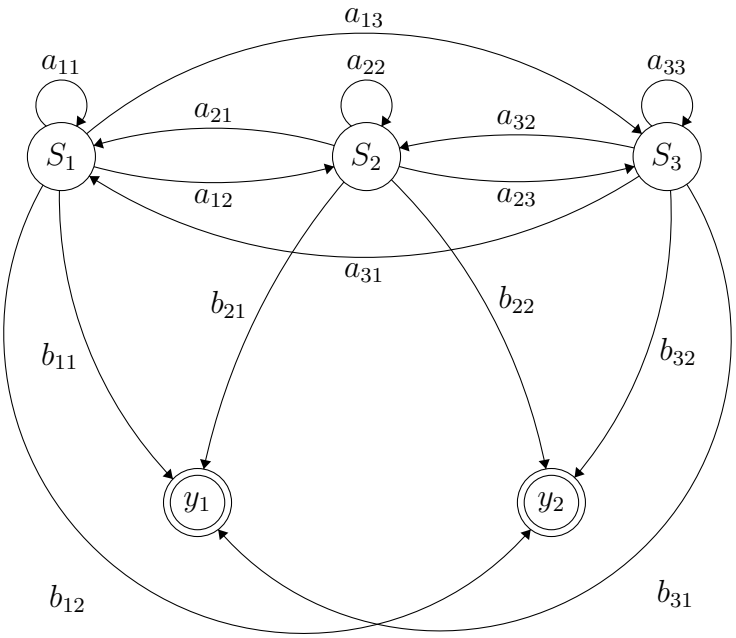


Figure 2.14: An example of a hidden Markov model with three hidden states and two emissions

Hidden Markov models are dynamic by design and provide a fitting framework for constructing acoustic models. At each time step, the model transitions to a state and an observation is emitted, see Figure 2.14. Each HMM is parameterized by the transition probabilities, the emission probabilities, and the starting probabilities. The transition probabilities, a_{ij} describe the likelihood of transitioning between state and the emission probabilities b_{ij}, describe the likelihood of each state emitting each part of the sequence. The starting probabilities describe the likelihood of the model starting in each state. HMMs are furthermore modeled with three key assumptions.

- The probability of a state only depends on the previous state, $P(S_t|S_{t-1}, \ldots, S_1) = P(S_t|S_{t-1})$. This is referred to as the *Markov*

assumption, which describes Markov processes.

- The probability of the current observation is independent of the previous observations and depends only on the current state, $P(x_t|S_t, S_{t-1}, \ldots, S_1, x_{t-1}, \ldots, x_1) = P(x_t|S_t)$. This is referred to as the observation independence assumption

- The probability of transitioning from state j to state i is independent of when the transition takes place, whether at step t_1 or t_2 $P(S_{t_1} = i|S_{t_1-1} = j) = P(S_{t_2} = i|S_{t_2-1})$

A *Markov Process* describes a sequence of events where the probability of each event in the sequence only depends on the previous event. Let us take a simple example of a Markov process using the model shown in Figure 2.14. We know that the model has emitted the sequence $Y = y_1 y_2$ but we are not aware of the process itself, i.e. which state emitted each part of the sequence.

When we use HMMs for acoustic modeling, the states of the HMM are phonemes and the emissions are feature vectors. More specifically, normally each phoneme is represented by multiple hidden states and each phoneme is modeled by a single HMM [21]. In Figure 2.15, we demonstrate how the topology of an acoustic HMM might look and how an observation sequence, $x_{1:7}$, might be emitted by the HMM that models the /a/ phoneme, here using 3 hidden states. At each step, the model transitions from one state to another, emitting a hidden vector at each step. In the example shown in Figure 2.15, the first hidden state emits the first three vectors of the sequence. This capability of the HMM is particularly handy for this task since phonemes correspond to a varying number of feature vectors.

The emission probabilities are often modeled using m-component Gaussian mixture models (GMMs), since the acoustic vectors associated with a certain state often follow a strongly non-Gaussian distribution [7]. For a given feature vector emission $\mathbf{x}$, the emission probability of each state, j, is given by

$$b_j(\mathbf{x}) = P(\mathbf{x}|\text{state} = j) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mathbf{x}, \mu_{jm}, \mathbf{\Sigma}_{jm}) \tag{2.15}$$

where c_{jm} is the mixing coefficient for each Gaussian distribution of the j-th GMM and $\mathcal{N}(\mathbf{x}, \mu_{jm}, \mathbf{\Sigma}_{jm})$ is the multivariate Gaussian distribution [8].

[7] By non-Gaussian, we mean a probability distribution that cannot be modeled by a single Gaussian distribution. For example, Gaussian distributions are always *unimodal* while GMM distributions can be *multimodal*.

[8] The probability density function (PDF) of the multivariate Gaussian distribution is given as $f(\mathbf{x} = [x_1, \ldots, x_k]|\mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu)\right)$

ANNs can be used to model the emission probabilities of acoustic HMMs [33]. Convolutional Neural Networks (CNN) have been used for this particular task as well [34]. Following the same structure as in [33], a CNN is used to apply convolutions along the frequency axis of the input spectrogram, which gives the model more tolerance for small shifts in frequency, caused by speaker or environment variations.

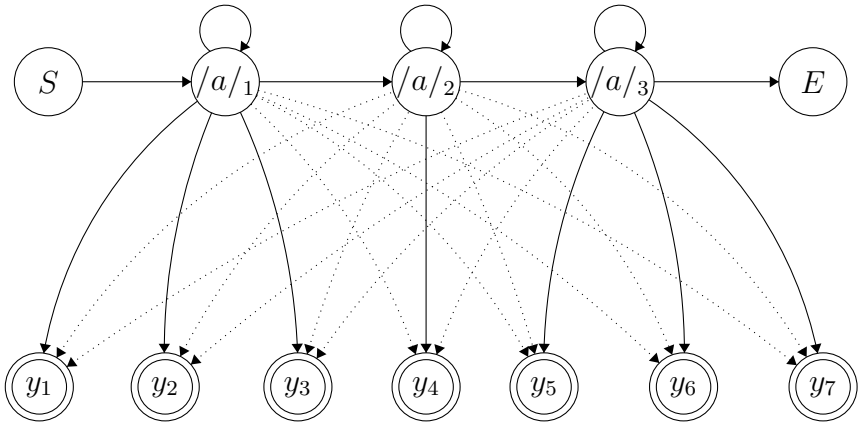


Figure 2.15: A 3-state HMM-based phone model for the /a/ phone (note: emissions denoted using $y_{1:7}$)

In Figure 2.15 we show how each hidden state of an HMM acoustic model corresponds with a varying amount of observations. For a whole utterance, the most likely word sequence given the sequence of feature vectors can be found by evaluating all possible sequences of states of each phone HMM and selecting the word sequence that corresponds with the state sequence with the maximum probability. In the example in Figure 2.15 the most likely hidden state sequence given the observation sequence $Y = y_1, \ldots, y_7$ has been determined to be (/a/$_1$, /a/$_2$, /a/$_3$). The state sequence can be found using the Viterbi algorithm, a dynamic programming approach that takes into consideration the transition and emission probabilities of the model [35].

The model parameters have to be tuned, by training, for it to achieve good results. For training, our goal is to estimate the parameters of the HMM, $\boldsymbol{\lambda} = [\{a_{ij}\}, \{b_j()\}]$, given an observation sequence. The parameters are determined using the *forward-backward* algorithm, an iterative Expectation Maximization (EM) algorithm [10]. Such algorithms estimate parameters of a model given observations Y by maximizing a likelihood function $P(Y|\boldsymbol{\lambda})$.

2.3.3 Language Models

As stated in Equation 2.1, the most likely word sequence does not only depend on the acoustic model but also the prior $P(\mathbf{W})$ which in this context is referred to as the language model. The prior is given by

$$P(\mathbf{W}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, \ldots, w_1) \text{ , where } \mathbf{W} = w_1 w_2 \ldots w_K \tag{2.16}$$

N-gram language models truncate this conditioning down to the last n words that preceded the current one. Equation 2.16 then becomes

$$P(\mathbf{W}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, \ldots, w_{k-n+1}) \tag{2.17}$$

Most commonly, the 3 nearest preceding words are used for language modelling.

The quality of language models is often measured in their *perplexity*, H, which is defined by

$$H = -\lim_{K\to\infty} \frac{1}{K} \log_2 P(w_1, \ldots, w_K) \tag{2.18}$$

A low perplexity value suggests that the language model is a good predictor.

The issue with n-gram language models is data sparsity. To calculate the probability of an n-gram as described in Equation 2.17, we divide the number of occurrences of the sequence with the number of occurrences of first $n-1$ occurrences. For example, to predict the probability of the word "here" to come after "wait right" using a 3-gram, we get

$$P(\text{"here"}|\text{"right", "wait"}) \approx \frac{\#\text{"wait right here"}}{\#\text{"wait right"}} \tag{2.19}$$

But these sequences might very well be very uncommon, leading to these estimates being ill-founded. One way of solving the data sparsity issue is to assign each word to a *class*. Our language model could have a few hundred classes and data sparsity would still be much less of an issue. Another approach to deal with language model data sparsity is found in Katz's *back-off model* [36]. Katz's model uses a combination of $1, \ldots, n$-gram models where the conditional probability estimate of a word is determined by progressively fewer preceding words when data sparsity is an issue. Formally, the conditional probability of a word w_i given the n preceding words, $w_{i-n+1}, \ldots, w_{i-1}$

is defined as

$$P(w_i|w_{i-n+1},\ldots,w_{i-1}) = \begin{cases} d_{w_{i-n+1},\ldots,w_i}\frac{C(w_{i-n+1},\ldots,w_i)}{C(w_{i-n+1},\ldots,w_{i-1})} & \text{if } C(w_{i-n+1},\ldots,w_i) > K \\ \alpha_{w_{i-n+1},\ldots,w_i}P(w_i|w_{i-n+2},\ldots,w_{i-1}) & \text{otherwise} \end{cases} \tag{2.20}$$

where d, α are weights, K a hyper parameter and $C(\mathbf{w})$ is the number of times the word sequence $\mathbf{w}$ appears in the training data. Often, K is chosen as 0 meaning that if the n preceding words of a word w_i appear at least once in the training data, we use the n-gram estimate. Otherwise we consider the $(n-1)$-gram estimates and so on.

Feedforward neural networks have been used for language modelling [37]. Both traditional n-gram LMs and the neural network approach proposed in [37] have a static size *context*, meaning that the probability of a certain word depends only on the last n-words. Recurrent neural networks (RNNs), on the other hand, can use the entire preceding sequence to predict a word, however long it has become [38]. The RNN used in [38] is the simple Elman network [39]. The Elman network is a very simple RNN, that takes in a concatenation of the feature representation of the current word and the output of the hidden layer from the previous step. The output of the network is the prediction of the next word.

Another type of an LM is the *skip-gram* [40], often referred to as a member of the *word2vec* model family. Both the LM proposed in [37] and the continuous bag of words model mentioned in [40] take as input some neighboring *context* of words. The skip-gram instead takes in the current word and predicts words in a given range from the current one, for example with a window size of 2 the words would be $w_{t-2}, w_{t-1}, w_{t+1}w_{t+2}$ given w_t. The formal objective of the skip-gram is to maximize the average log-probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log P(w_{t+j}|w_t) \tag{2.21}$$

where c is the size of the window. We train the skip-gram on a large corpus of source text, running sequentially through it. Iteratively, the model learns the relationship between words that often appear close to each other in context.

Similarly to the NNLM, the skip-gram learns a feature vector representation of each word. In [40], they found that simple algebra rules could be applied to answer questions about the relationship of words. For example, to find a word that is similar to "small" in the same way that "big" is similar to

"biggest", the vector x can be computed by

$$x = \text{vector("biggest")} - \text{vector("big")} + \text{vector("small")} \tag{2.22}$$

then the vector in the vector space that is closest to x could be found and that word is used to answer the question.

2.3.4 Pronunciation Dictionaries

Pronunciation dictionaries, or lexicons, are used by some ASR models during decoding. Models that predict phonemes, like the hidden Markov model with Gaussian mixture emissions (GMM-HMM) discussed in Section 2.3.2, need a pronunciation dictionary to decode the words from the phonemes. The dictionary provides a mapping, from a sequence of phonemes to a word. To deal with the multiple pronunciations of words, pronunciation dictionaries map multiple phoneme sequences to each word [41]. In this way, the dictionary models what is called *within-word variation*. Some lexicons offer limited multi-word variation, i.e. multiple phoneme mappings for sequences of words [41].

2.3.5 Decoding

The final step of the ASR process is decoding. During decoding, depending on the type of ASR model, we find suitable candidates for Equation 2.1. That is, given the input speech signal, we aim to find the most probable sequence of words. A dynamic programming solution could be used, but the search space quickly becomes unwieldy. It is therefore important to prune the search space as early as possible.

Most of these algorithms heuristically produce estimates of the n-best word sequences [42], or *hypotheses*. Often, these n-best sequences are stored in word lattices [43] which are convenient and compact structures for storing these hypotheses. A simple word lattice is shown in Figure 2.16. Each path from the leftmost vertex in the graph, representing the start of the utterance, to the rightmost vertex, representing the end, is a decoding hypothesis.

Many DNN ASR models use beam search during decoding [2]. Beam search is a breadth-first search algorithm where at each level of the search tree instead of greedily choosing the most probable hypothesis, we expand a fixed amount of hypotheses, denoted by a parameter β. For an ASR model that predicts one word at a time using $\beta = 5$, we would start out with the top 5 guesses of the first word. We then do this recursively for each guess until we have reached the desired amount of guesses, depending on the length of the utterance.

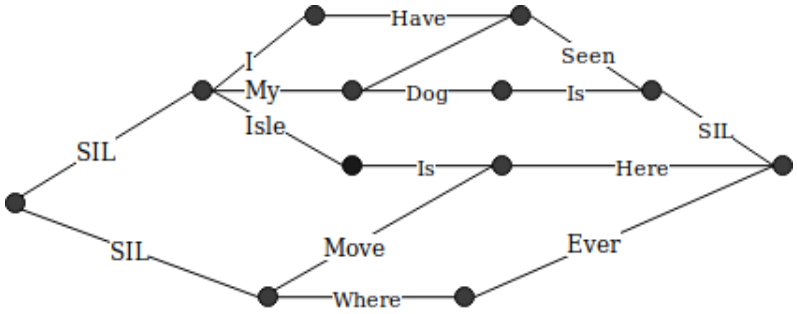


Figure 2.16: An example of a word lattice

2.3.6 Quality Measurements

The most common metric for measuring the quality of ASR systems is the word error rate (WER). The lengths of the hypothesis and the reference are often different and therefore comparing a word at a time, or character, from each sequence will not be sufficient to measure quality. The WER of a predicted hypothesis h, and a reference r is given by

$$\text{WER}(h, r) = \frac{S + D + I}{N} \tag{2.23}$$

where

- S is the number of word substitutions to change h into r
- D is the number of word deletions to change h into r
- I is the number of word insertions to change h into r
- N is the number of words in the reference to change h into r

where the error rate in 2.23 has been minimized. This equals measuring the *Levenshtein distance* between the hypothesis and the reference [44]. The Levenshtein distance is a dynamic programming algorithm and the distance between h and r is given as $\text{lev}(|h|, |s|)$ where $|h|$ and $|s|$ is the string length of the hypothesis and the reference and where

$$\text{lev}_{h,r}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{h,r}(i-1,j) + 1 \\ \text{lev}_{h,r}(i,j-1) + 1 \\ \text{lev}_{h,r}(i-1,j-1) + 1_{(h_i \neq r_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{2.24}$$

where $1_{(h_i \neq r_j)}$ is the indicator function

$$1_{(h_i \neq r_j)} = \begin{cases} 1 & \text{if } h_i \neq r_j \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

Chapter 3

Background

Here, additional background and related work is discussed. This chapter has two main subjects. First we discuss different types of ASR models, focusing on sequence-to-sequence models and the modern state-of-the-art. This also includes a detailed background on recurrent neural networks, attention based models and generative adversarial networks.

The second subject addresses theory and work related to machine learning model architecture and training, focusing on learning rules, activation functions and weight initialization.

3.1 Recurrent Neural Networks in ASR

RNNs by their design incorporate the temporal nature of data like speech into neural networks, which normally have a hard time to deal with the data of varying length.

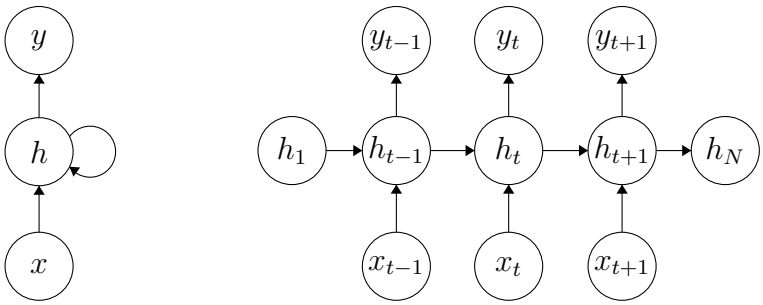


Figure 3.1: Folded and unfolded diagrams of typical RNNs

Figure 3.1 shows a diagram of a simple RNN, folded on the left and unfolded on the right. Given an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ the RNN will

compute a hidden sequence $\mathbf{h} = (h_1, h_2, \ldots, h_T)$ and an output sequence $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ by iteration of all time steps using the Equations

$$h_t = \sigma(\mathbf{W}_{xh}x_t + \mathbf{W}_{hh}h_{t-1} + b_h) \tag{3.1}$$

$$y_t = \mathbf{W}_{hy}h_t + by \tag{3.2}$$

where $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$, b_h, $\mathbf{W}_{hy}$, and b_y are trainable weights and σ is a sigmoid activation function [45], see Section 3.5. The main difference therefore between feed-forward NNs and RNNs is the idea of a loop in the network as shown in Figure 3.1. This loop allows the network to adjust its context as needed and have memory of events that happened many time steps earlier. Vanishing gradients are however an issue for these simple RNNs since they have a limited capacity to handle long-term dependencies [46]. The long-short term memory (LSTM) [45] RNN was proposed as a solution to the vanishing gradient problem that simpler RNNs deal with. Each LSTM node can manipulate its cell

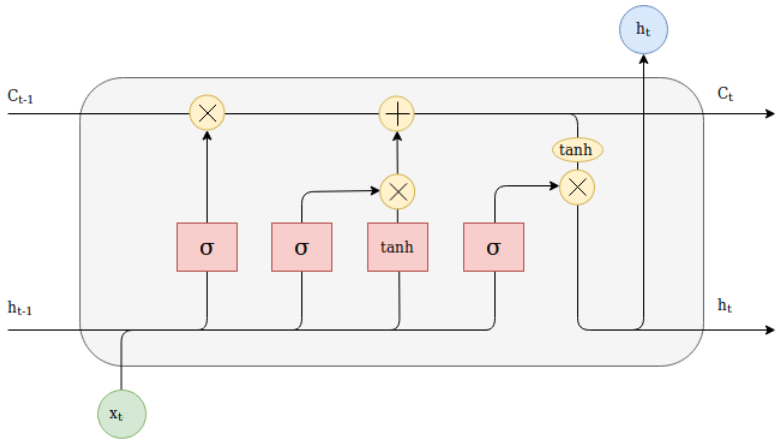


Figure 3.2: LSTM diagram

state, its *memory*, by adding to it, subtracting from it or leaving it unchanged. This is achieved by implementing three trainable gates.

The *forget gate layer* is used to decide to what extent a cell forgets the previous cell state c_{t-1}. Based on each scalar value in h_{t-1} and the current input x_t, the forget gate layer outputs a number between 0 and 1 for each value in the cell state

$$f_t = \sigma(\mathbf{W}_{xf}x_t + \mathbf{W}_{hf}h_{t-1} + b_f) \tag{3.3}$$

where σ is the element-wise sigmoid activation function. For the forget layer to completely forget $c_{t-1}[i]$, it will set $f_t[i] = 0$ and $f_t[i] = 1$ to keep it in the

cell state unaltered. These values are pointwise multiplied with the previous cell state as shown in Figure 3.2, in the top left.

Equally important to forgetting old information is to store new information. In LSTMs, this is a two-step procedure. First, to decide which values of the cell state to update, h_{t-1} and x_t are passed through a sigmoid layer, the *input gate layer*. Similarly to the forget gate layer, the input gate layer outputs 0 if the particular element of the cell state should be left unchanged. The output of the input gate is given by

$$i_t = \sigma(\mathbf{W}_{xi}x_t + \mathbf{W}_{hi}h_{t-1} + b_i) \tag{3.4}$$

Additionally the candidate elements are generated with tanh-activation (see Section 3.5)

$$\widetilde{c}_t = \tanh(\mathbf{W}_{xc}x_t + \mathbf{W}_{hc}h_{t-1} + b_c) \tag{3.5}$$

These are the values that will perhaps be added to the cell state. The candidate elements are then element-wise multiplied by the output of the input gate. The new cell state is then given by

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t \tag{3.6}$$

where $\odot$ denotes the element-wise product, or the *Hadamard product* [1]. The output, h_t, will be based on the cell state but a refined version of it. The *output gate layer* is used to achieve this. Based on h_{t-1} and x_t, we decide which values to output. That decision is stored in o_t. Next, the cell state is tanh-activated and then multiplied by o_t to only output the desired part of c_t.

$$o_t = \sigma(\mathbf{W}_{xo}x_t + \mathbf{W}_{co}c_{t-1} + b_o) \tag{3.7}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.8}$$

This is a common version of the LSTM cell, but there are many different versions in use, e.g. the popular peephole BLSTM [47]. These differences are however minor and the computation of the hidden sequence is very similar across different types of LSTMs. LSTMs have been used both for acoustic modeling [48], where it surpassed the state-of-the-art performance of other deep neural network (DNN) acoustic models and for language modeling [49], where they have been successfully applied and obtain lower perplexity compared to simpler RNN LMs.

[1] The Hadamard product of two matrices, A and B, of identical dimensions is denoted as $A \odot B$ where $(A \odot B)_{ij} = A_{ij}B_{ij}$

Conventional RNNs, including LSTM networks, are however only capable of making use of the previous context to calculate the current output. Bidirectional recurrent neural networks (BiRNNs) combine two layers, each processing the sequence of data in opposite directions and thus making use of both past and future context.

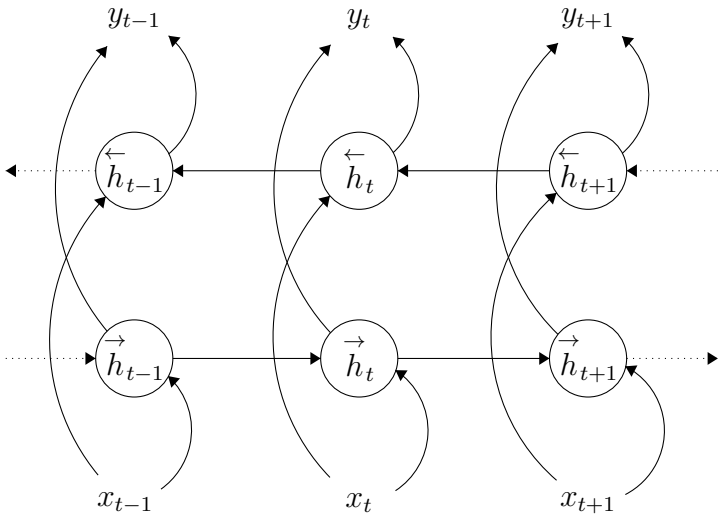


Figure 3.3: Bidirectional RNN topology

Such networks, therefore, calculate two hidden sequences and those are then combined to calculate the output sequence

$$\overrightarrow{h} = \sigma(\mathbf{W}_{x\overrightarrow{h}} x_t + \mathbf{W}_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \tag{3.9}$$

$$\overleftarrow{h} = \sigma(\mathbf{W}_{x\overleftarrow{h}} x_t + \mathbf{W}_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \tag{3.10}$$

$$y_t = \mathbf{W}_{\overrightarrow{h}y} \overrightarrow{h} + \mathbf{W}_{\overleftarrow{h}y} \overleftarrow{h} + b_y \tag{3.11}$$

A commonly used BiRNN is the bidirectional long-short term memory (BLSTM) network [50]. BLSTMs have been used for acoustic modelling [51] and are commonly used for language modelling. The BLSTM networks uses the same LSTM cells as regular LSTM networks and the bidirectional versions of Equations 3.3-3.8 are achieved similarly as Equations 3.9-3.11 compared to Equations 3.1-3.2.

When used in acoustic modeling, BLSTM networks are normally stacked to create deep BLSTM networks. By doing so, these networks become capable of building up differently leveled representations of the input sequence [52].

3.2 Sequence-To-Sequence Learning in ASR

In many tasks, such as speech recognition and text to speech synthesis, we aim to predict sequences of varying lengths, given input sequences of varying lengths. Unaltered DNNs or RNNs cannot be used to achieve this end-to-end and without any major assumptions of the sequence structures. To train GMM-HMM ASR models the targets have to be aligned at the frame-level, which is a costly and daunting task.

In recent years, the focus of ASR development has in many ways shifted to sequence-to-sequence learning from the more conventional GMM-HMM models. When we talk about sequence-to-sequence learning in the ASR domain, we mean predicting a sequence of words directly from the corresponding speech signal, as opposed to predicting phonemes or by any means dissecting the learning tasks into smaller units. Furthermore, when we refer to end-to-end training, we mean that the whole model is trained jointly with a single loss function. In the ASR domain, that means training a model without the use of e.g. separate language models or any other auxiliary components.

3.2.1 Connectionist Temporal Classification

In [53], a method to do away with the frame-level alignment for end-to-end training is proposed. Let us assume that we want to predict the text sequence $\mathbf{y}$ from the speech signal $\mathbf{x}$. We then define the set $\mathcal{B}(\mathbf{y}, \mathbf{x})$ as the set containing all strings that:

- are of length $|\mathbf{x}|$
- are identical to $\mathbf{y}$ after removing any repeated targets and *blank* symbols, denoted by "<b>".

For example, for the target $\mathbf{y}$ ="hello" where $|\mathbf{x}| = 10$ then $\mathcal{B}(\mathbf{y}, \mathbf{x})$ includes for example

- "<b><b><b><b><b>hello"
- "h<b>e<b>l<b>l<b>o<b>"
- "h<b>eee<b>ll<b>o"

In this way, $\mathcal{B}(\mathbf{y}, \mathbf{x})$ is the set of *all* frame level alignments for the utterance. Using connectionist temporal classification (CTC), the acoustic model is de-

fined as

$$P(\mathbf{y}|\mathbf{x}) = \sum_{\hat{\mathbf{y}} \in \mathcal{B}(\mathbf{y},\mathbf{x})} \prod_{t=1}^{|\mathbf{x}|} P(\hat{y}_t, \mathbf{x}), \quad \hat{\mathbf{y}} = \hat{y}_1 \dots \hat{y}_T \tag{3.12}$$

Each conditional in Equation 3.12 is estimated with a recurrent neural network, referred to as the *encoder*. At each encoder time step, the encoder produces a vector $\mathbf{h}_t^{enc}$ which is softmaxed[2] over all possible characters. By design, the CTC is very similar to an acoustic model.

In [53], the WER of a CTC ASR is compared to that of a GMM-HMM on the TIMIT speech corpus, and the CTC is shown to outperform the GMM-HMM with a WER score of $30.51 \pm 0.19\%$ versus $31.57 \pm 0.06\%$.

3.2.2 The RNN Transducer

The RNN transducer [54] is similar to the CTC model, but an additional *prediction network* over possible labels is added for encoding. Here, the prediction network can be thought of as a language model and the encoder as the acoustic model. Both the prediction network and the encoder are modeled with recurrent neural networks. At each step of decoding, the prediction network receives the previous label prediction, y_{u-1} as input and produces an output vector $\mathbf{p}_u$ over all possible output labels. Along with the encoder output, the prediction network output at every time step is passed on to a *joint network*. The joint network computes at every time step

$$\mathbf{h}_{t,u}^{\text{joint}} = \tanh(A\mathbf{h}_t^{\text{enc}} + B\mathbf{p}_u + b) \tag{3.13}$$

$$\mathbf{z}_{t,u} = D\mathbf{h}_{t,u}^{\text{joint}} + d \tag{3.14}$$

where A, B, D, b and d are trainable parameters of the model. $\mathbf{z}_{t,u}$ are treated as logits and are passed to a softmax, together defining a probability distribution over possible output labels u for each frame t of the speech signal.

In [54], the error rate of a CTC model was compared to that of the RNN transducer on the TIMIT speech corpus. The RNN transducer achieved a lower error rate, 23.2% compared to the 25.2% achieved by the CTC model.

3.2.3 The RNN encoder-decoder

The *RNN encoder-decoder* model proposed in [55] in 2014 was a breakthrough for sequence-to-sequence learning. The solution was to use one RNN, referred

[2]The softmax function turns a vector of arbitrary values into a vector of probabilities and is defined as $\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

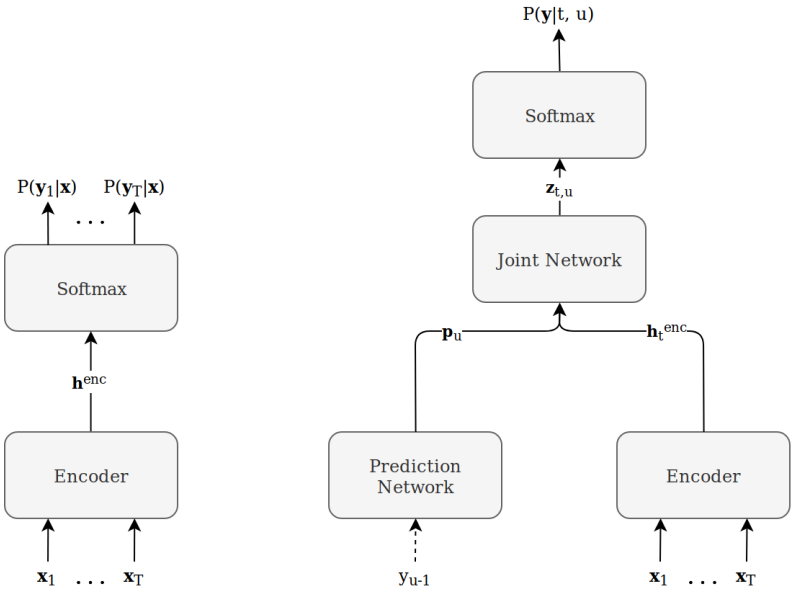


Figure 3.4: The topology of CTC on the left, and RNN transducer on the right

to as the *encoder*, to read the input sequence and map the output to a large fixed-size context vector c. Then another RNN referred to as the *decoder*, would take as input the context vector and predict the variable-length output sequence. In the simple example shown in Figure 3.5, the encoder is shown on

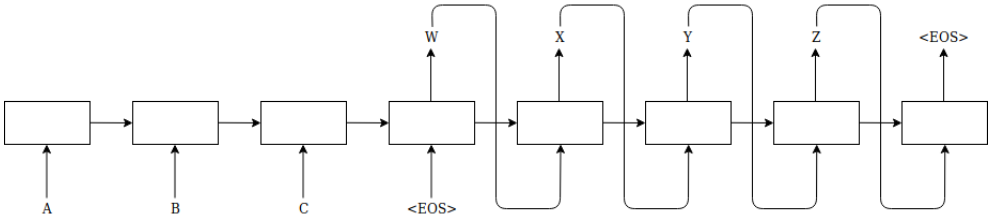


Figure 3.5: The topology of a simple sequence-to-sequence learning model

the left and takes as input the sequence "ABC<eos>" where "<eos>" is some token representing *end of sequence*. This sequence is fed to the encoder one symbol at a time. The fixed-size representation of the input is the value of the last hidden state of the encoder, which is the hidden state of the LSTM cell that receives "<eos>" as input. The decoder, shown on the right, takes as input at the first time step the context vector c. At each following time step,

the previous emission is supplied as input. The encoder stops predicting after emitting the end of sentence token.

Interestingly, in [55] they found that by reversing the order of the input sequence, the model becomes even more capable of learning long-term dependencies. Instead of the model mapping (a, b, c) to (α, β, γ), the model maps (c, b, a) to (α, β, γ). By doing so a is in close proximity to α which enables the model to *make the connection* earlier. Additionally, by reversing the input sequence, the average time lag stays the same between words in the input and the output but at the same time, the minimum time lag is made smaller. They furthermore found that deep LSTM networks significantly outperformed the single-layered LSTM network.

In general, RNNs estimate the probability of an output sequence $\mathbf{y} = (y_1, \ldots, y_T)$, given an input sequence $\mathbf{x} = (x_1, \ldots, x_T)$ where both sequences are of equal length. In sequence-to-sequence learning we aim to estimate

$$P(y_1, y_2, \ldots, y_{T'} | x_1, x_2, \ldots x_T) \tag{3.15}$$

where T' is not necessarily equal to T. By encoding the input sequence as fixed size vector c, Equation 3.15 can be written as

$$P(\mathbf{y}) = \prod_{t=1}^{T'} P(y_t | c, y_1, \ldots, y_{t-1}) \tag{3.16}$$

where c is the context vector and $\mathbf{y} = (y_1, \ldots, y_{T'})$. Each distribution in the product on the right in Equation 3.16 is represented with a softmax over all the possible outputs, every character in the case of character-level ASR models. By ending each input sequence with the end-of-sequence token, we enable the model to define a distribution over any arbitrarily sized sequences, that is, the decoder can continue to decode for any arbitrary time until "<eos>" is emitted.

3.2.4 Attention-based Models

Using *attention* in sequence-to-sequence learning has been an integral part of encoder-decoder models after it was proposed in [56]. In [56] the authors conjecture that the fixed size context vector used in sequence-to-sequence models like the encoder-decoder RNN, is a bottleneck. Specifically, the issue of compressing the necessary information of long input sequences into the context vector. In [56], another type of sequence-to-sequence model is proposed, the *sequence-to-sequence model with attention*. Instead of encoding the input sequence into a fixed size context vector, the model encodes the input sequence

to a sequence of vectors and the decoder chooses a set of those adaptively while decoding. This new model achieved state-of-the-art performance on the English-to-French machine translation task [55] and the attention mechanism has been extensively adapted and used in machine translation, ASR and TTS in recent years. Like was explained in Section 3.2, the RNN encoder-decoder

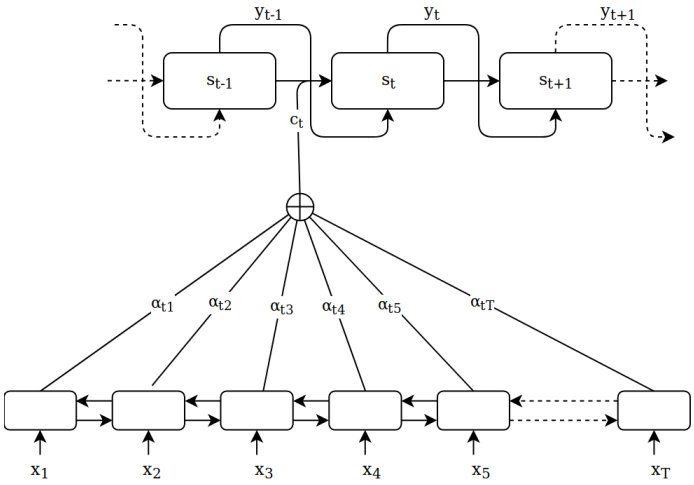


Figure 3.6: The topology of the model proposed in [56]

model encodes all the information of the input sequence into a context vector or the last hidden state emitted by the encoder, h_T. When the sequences are long the amount of information needed for the translation becomes too unwieldy to be carried over by a single hidden state. The attention-based model, on the other hand, takes all the emitted encoder hidden states into consideration.

What that means is that instead of each conditional probability in Equation 3.16 being only conditioned on the context vector, attention-based models condition the output at each step on the whole input sequence $\mathbf{x}$, that is

$$P(\mathbf{y}) = \prod_{t=1}^{T'} P(y_t|y_1, \ldots, y_{i-1}, \mathbf{x}) \tag{3.17}$$

where each conditional is estimated with $g(y_{t-1}, s_t, c_t)$, a multi-layered function with softmax on top. The decoder state s_t is calculated similarly to how the LSTM state was calculated in Chapter 2.3, but in [56], the state is dependent on the previous output y_{t-1} and the context vector c_t. At each decoder

time step, we calculate the context vector

$$c_i = \alpha_{i1}\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ \vdots \\ h_{1n} \end{bmatrix} + \alpha_{i2}\begin{bmatrix} h_{21} \\ h_{22} \\ h_{23} \\ \vdots \\ h_{2n} \end{bmatrix} + \cdots + \alpha_{iT}\begin{bmatrix} h_{T1} \\ h_{T2} \\ h_{T3} \\ \vdots \\ h_{Tn} \end{bmatrix} = \sum_j^T \alpha_{ij} h_j \tag{3.18}$$

where the weight α_{ij} for each of the encoder time step's hidden state is computed with the softmax function over all the encoder time steps

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k^T \exp(e_{ik})},$$
$$e_{ij} = a(s_{i-1}, h_j)$$

where a is an *alignment model* and s_{i-1} is the previous hidden state of the decoder. The *energy* value e_{ij}, the output of the alignment model, should reflect how well the input at time step j and the output at time step j match. In [56] this alignment model is parametrized with a simple feed forward network with tanh activation

$$a(s_{i-1}, h_j) = \mathbf{v}_a^{\mathrm{T}} \tanh(\mathbf{W}_a s_{i-1} + \mathbf{U}_a h_j)$$

The weighted sum in 3.18 represents the expected encoder hidden state emission, taking the expectation over all possible alignments. We can, therefore, understand attention as a way of providing a probability distribution over encoding steps. In this way, the decoder is given the responsibility to choose a part of the input sequence to *attend* to, thus relieving the burden of encoding all the needed information in a single hidden state from the encoder.

A good way of inspecting alignment models during training is to visualize the attention values, α_{ij} for all encoder and decoder time steps. Visualizing these values gives us an idea about what the decoder decides it should attend to at every step.

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1T'} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{2T'} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3T'} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{T1} & \alpha_{T2} & \alpha_{T3} & \dots & \alpha_{TT'} \end{bmatrix} \tag{3.19}$$

If for example, the alignment values on the diagonal in the matrix in 3.19, the α_{ii} values if the matrix is square, would commonly be the highest values,

then we could theorize that the alignment between the source and the target is quite monotonic. This is often the case for tasks like ASR. In machine translation tasks, although depending on the language, this is not necessarily the case and often there exists a non-monotonic alignment between the source and target. This describes well the power of using attention for sequence-to-sequence learning.

Listen, Attend and Spell

Listen, attend and spell (LAS) [2] is a type of a sequence-to-sequence model with attention made specifically for the ASR task. Importantly, LAS predicts characters rather than words which means out-of-vocabulary (OOV) words can be handled automatically. The LAS model consists of an RNN encoder, referred to as the *listener*, and an attention-based RNN decoder, which we will refer to as the *speller*.

Sequence-to-sequence ASR models often are often plagued by a very slow convergence rate since the number of input speech signal frames can be in the thousands. LAS alleviates the input dimension problem by *down-sampling* the input with a a Pyramid-shaped Bidirectional Long-Short Term Memory (pBLSTM) encoder, similar to the one used in [57]. The
pBLSTM encoder, or the listener, used in LAS had three layers. Each layer reduces the time resolution by a factor of two, bringing the final resolution down by a factor of eight.

The listener produces a vector of listener outputs, $\mathbf{h} = \text{Listen}(\mathbf{x})$. Using, $\mathbf{h}$, a context vector c_i is created at every decoder time step. The previous context vector, c_{i-1}, and the previous character prediction, y_{i-1} are fed into the speller which then emits the current decoder state s_i

$$s_i = \text{RNN}(s_{i-1}, y_{i-1}, c_{i-1}) \tag{3.20}$$

LAS uses BLSTM cells for the speller, and the RNN function is formally described in Section 4.1.3. The state s_i is then turned into a probability distribution over possible output labels with a simple feed forward network, Ω, with softmax

$$P(y_i|\mathbf{x}, y_{<i}) = \text{softmax}(\Omega(s_i)) \tag{3.21}$$

Beam search is used during decoding and in [2], they found that the LAS model had a bias for shorter utterances. Therefore, the probability of each sequence is normalized by its character length, denoted by $|\mathbf{y}|_c$. Additionally, for some

experiments, a language model was added to rescore each beam in the beam search. The complete distribution that the model approximates is given by

$$S(\mathbf{y}|\mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|_c} + \lambda \log P_{LM}(\mathbf{y}) \tag{3.22}$$

where λ is a language model weight and $P_{LM}(\mathbf{y})$ is the language model.

In [2], they found that without using attention, the model overfitted to the training data and without the pyramid BLSTM down-sampling the model converged too slowly. Previous sequence-to-sequence models used in ASR had only been applied to phoneme sequences, rather than a voice signal. LAS achieves a word error rate (WER) of 14.1%, without any modifications, on a subset of the Google voice search task. LAS, however, requires a large amount of transcribed speech for training, LAS was trained on 2000 hours of data [2], which often is not available under low resource conditions.

In [58], multiple sequence-to-sequence ASR models are compared, including Listen, Attend and Spell. Compared to both CTC-based models and RNN transducer models, a version of LAS proved to be the best sequence-to-sequence ASR model on 5 different test sets, achieving a 6.3% WER on a clean dictation dataset, compared to 6.6% WER achieved by RNN transducer.

The Transformer

In 2017, another breakthrough in encoder-decoder modelling was made with the introduction of the *Transformer* model [59]. Up to that point, RNNs were seen as an integral part of sequence-to-sequence learning, but the transformer model managed to achieve state-of-the-art results on the English to French translation task without any RNNs.

The transformer model uses *self-attention* to represent inputs and outputs without any convolutions or sequence-aligned RNNs. Self-attention allows a model to connect different positions of the sequence to compute a representation of the entire sequence. Additionally, the transformer relies heavily on *multi-head attention*. Instead of performing a single attention function, the inputs are linearly projected *h* times using learned linear projection networks. The attention is then applied in parallel to all *h* instances. This aided the model to generalize, similar to ensemble learning.

Another new and interesting approach to attention is called *area attention* [60]. Area attention enables models to attend to dynamic areas of adjacent items in the memory. Models using the more conventional attention mechanisms can attend to some fixed size item at a time, a single frame in the case

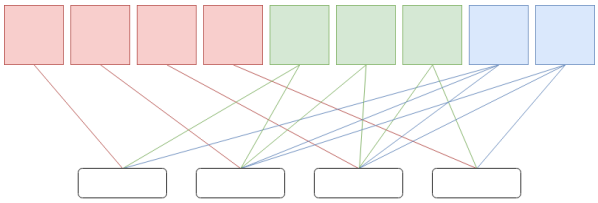


Figure 3.7: Area attention shown for a sequence of length 4 and maximum allowed area size is 3

of ASR for example, or a word in the case of MT. Area attention introduces a way of attending to groups of consecutive items at the same time. Importantly, the size the area is determined with learning, allowing the model to vary its granularity while attending to the input sequence. In Figure 3.7 this is shown for sequences of length 4 and where a constraint of 3 has been set on the size of attention areas. At the bottom, the original memory is shown, the encoder outputs of the encoder in an encoder-decoder RNN for example. The red squares represent 1-sized memory or the same type of memory that is attended to in [56]. The green squares show all possible 2-sized memories and so on. The attention weights will then allow the model to choose not only the location in memory to align the output to, but also the size of the area at that location.

In [60], the conventional attention mechanisms of the Transformer model [59] were replaced with area attention. The strong performance baselines of the Transformer model were all improved by the introduction of area attention.

3.3 Generative Adversarial Networks and Unsupervised Learning

An interesting model type for unsupervised learning tasks, especially under low-resource conditions, is the generative adversarial network (GAN) [61]. A typical GAN consists of a *generator* network, G, and a *discriminator* network, D. The goal of the generator is to produce samples that are likely to be sampled from some ground truth data source. The goal of the discriminator is to tell apart synthetic samples generated by the generator and real samples from the data source. The generator outputs a single scalar for each sample between 0 an 1, with the goal of assigning 1 to sample from the data source and 0 to the synthetic data. The Discriminator is trained to maximize

$$\log(D(\mathbf{x})) + \log(1 - D(G(\mathbf{z})) \tag{3.23}$$

where $\mathbf{x}$ is a ground truth sample and $\mathbf{z}$ is some prior distribution that the discriminator samples from to generate the synthetic data. At the same time, the generator is trained to minimize

$$\log(1 - D(G(\mathbf{z})) \tag{3.24}$$

By training both the discriminator and the generator at the same time, the discriminator becomes better at predicting the ground truth and at the same time the generator becomes better at synthetically generating data that is likely to be sampled from the ground truth data distribution.

Attempts have been made both in machine translation tasks and ASR to reduce the need for labeled data, such as a bilingual dictionary in the case of machine translation. In [62], the authors noted that word embedding spaces are often similar between languages, meaning that word embeddings that are close in one language tend to be close in other languages as well, even in very different languages. They proposed to use a small bilingual dictionary of only 5000 words as a base point and from it learn a mapping from the source embedding space to the target space. Although the performance demonstrated in [62] was promising it was simply outperformed by previous supervised methods.

In [63], a word translation method without any parallel data is proposed. The method uses two large monolingual corpora and adversarial methods to learn a mapping from the source language to the target. Starting with two distributions of word embeddings, one for each language. The goal is to find a rotation matrix, $\mathbf{W}$, which closely aligns the source distribution X and the target distribution Y in such a way that $\mathbf{W}X$ becomes similar to Y. Adversarial training is used to achieve this. A discriminator is trained to discriminate between samples from $\mathbf{W}X$ and Y while $\mathbf{W}$ is trained to maximize similarities between $\mathbf{W}X$ and Y. The embedding similarity observation made in [62] is an important key observation for the rotation to result in close alignments.

In [63], the mapping is further improved by employing similar techniques as in [62], by generating synthetic parallel dictionaries iteratively and using embeddings from this synthetic bilingual dictionary as anchor points when reevaluating $\mathbf{W}$.

In [64] unsupervised machine translation without any parallel data is investigated. Here, the authors use a sequence-to-sequence model with attention to construct a sentence in the target language from a noisy sentence from the source language. For each language, there is an encoder which transforms a sentence to the shared latent space and a decoder to retrieve a sentence into either the source or target language.

The suggested method starts by generating a synthetic dictionary as proposed in [63] and then at each iteration the encoders and decoders are tuned to minimize loss of reconstructing a sentence from a noisy source sentence. These altered encoders and decoders are then used to generate new translations. Noise is added to the source sentences since, without any constraints, the auto-encoders would simply learn to copy the input without understanding the structure of the sentence. Adversarial training is used to promote alignment of source and target language latent distributions. A discriminator is employed to determine whether an encoding of a sentence in the latent space comes from the source language or the target language. The encoder is trained to fool the discriminator and thus promotes alignment in the latent space.

In [15], ideas from [63] and [2] are combined to create a semi-supervised sequence-to-sequence ASR using adversarial training. The goal in [15] is to leverage the vast amount of unlabeled speech and text available, along with a small parallel corpus, in ASR training. To achieve this, a baseline sequence-to-sequence ASR is used and in [15] that model is very similar to LAS [2]. Additionally, the model consists of a text autoencoder, speech autoencoder, and a discriminator network. The text autoencoder uses the decoder of the baseline ASR model for decoding and the speech autoencoder partially uses the encoder of the baseline ASR model for encoding.

Adversarial training is used in [15] and the text encoder outputs are treated as ground truth data and the speech encoder is treated as the generator. With this adversarial training setup, the goal is to encourage the text and speech encoders to share an output embedding space. More specifically, we encourage the speech encoder to *imitate* the text encoder which has been trained to efficiently encode text features. Another integral part of the training procedure is to train the speech autoencoder. As mentioned in [15], speech autoencoders are different from text autoencoders since we need to encode both linguistic and non-linguistic features of the speech signal. The ASR encoder is designed to encode the linguistic content of the speech signal and in [15], another encoder is added to capture non-linguistic content. The speech autoencoder is trained to efficiently encode speech from unlabeled speech data and therefore we achieve additional unsupervised training of the baseline ASR encoder.

Since the key observation used in [64] does not hold with unlabeled ASR datasets, a semi-supervised approach is used in [15] to bootstrap the training of the ASR using a relatively small aligned corpus. Then both the text and speech autoencoders can be trained independently. Periodically, adversarial training is performed using the discriminator. In [15], the proposed semi-supervised model is compared to the baseline model using 2.5, 5 and 10 hours of tran-

scribed speech. In all situations, the proposed model achieves a lower word error rate. They additionally showed that adversarial training was important to achieve these results.

3.4 Learning Rules

Given some labelled training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, typical machine learning problems focus on training parameters $\boldsymbol{\theta}$ of a model, such that it becomes capable of robustly predicting the labels of some unseen feature data, $\mathbf{x}$. Some quality measure, $\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta})$ is used to determine the predictive power of the model. This measure is commonly a cost function, a *loss* function, which is to be minimized. To minimize the loss function, we derive the gradient of the loss function given the training data, w.r.t. the model parameters $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta})|_{(x,y)\in\mathcal{D}}$. The model parameters are then updated using *gradient descent*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}_t)|_{(x,y)\in\mathcal{D}} \tag{3.25}$$

Where η_t is the step size of the update, often referred to as the *learning rate* [65]. To efficiently calculate the gradients needed in Equation 3.25, we use *error backpropagation*. In very simple terms, backpropagation recursively applies the chain rule

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(y)}{\partial y}\frac{\partial g(x)}{\partial x} \quad \text{where} \quad y = g(x) \tag{3.26}$$

to calculate the gradient of the loss function w.r.t. each parameter of the model, given the value of the loss function [65].

This method of minimizing the loss is called stochastic or mini-batch *Gradient Descent* and is normally employed for this task [65]. Many different learning rules have been proposed and here we introduce four differently complex versions leading up to ADADELTA in Section 3.4.4, which is the learning rule used in this work as listed in Section 5.4.

3.4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) involves updating the model parameters, θ iteratively in the direction of steepest descent, following the gradient

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta_{\boldsymbol{\theta}_t} \tag{3.27}$$

where $\Delta_{\boldsymbol{\theta}_t}$ is the negative gradient of the loss function with regards to the parameters as shown in Equation 3.25. The learning rate in Equation 3.25 controls the size of the step we take along the gradient per parameter update. One issue with SGD is that we have to, based on some assumptions, choose a reasonable learning rate. A too low learning rate leads to slow convergence and too high learning rate can lead to the model to diverge from the minimum. Therefore, typically some tuning procedure is performed to choose a good learning rate.

3.4.2 SGD with Momentum

Methods have been proposed where, based on a heuristic, learning is slowed down near the minima and increased where suitable. If the learning rate is not annealed when learning approaches the minima, we risk jumping repeatedly over and around the minima. A simple extension to the SGD algorithm to affect learning rate in this way is using *momentum* [66]. Momentum allows for learning to be fast in the dimension where the gradient is continually in the same direction, and slow it down in dimensions where the gradient direction changes

$$\Delta_{\boldsymbol{\theta}_t} = \rho\Delta_{\boldsymbol{\theta}_{t-1}} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}_t) \tag{3.28}$$

Here, ρ is the momentum constant that controls the *decay* of the previous update. If we imagine for example, that we are doing SGD along a narrow valley in the solution surface, the gradient along the valley will be small but continuously in the same direction. The gradient across the valley will most often be higher but oscillates in each direction. Applying momentum will encourage gradient updates along the valley and thus avoids getting stuck in a local gradient *whirlpool* that slows convergence.

3.4.3 ADAGRAD

ADAGRAD is an example of a learning method where the learning rate of each dimension is dynamic. The update rule in ADAGRAD

$$\Delta_{\boldsymbol{\theta}_t} = \frac{-\eta}{\sqrt{\sum_{\tau=1}^{t}\mathbf{g}_\tau^2}}\mathbf{g}_t, \quad \mathbf{g}_t = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}_t) \tag{3.29}$$

The denominator in the update calculates the per-dimension ℓ^2-norm of all previous gradients. This results in a per-dimension dynamic learning rate. The

higher gradients in a dimension, the larger the denominator and therefore the overall learning rate is decreased. The opposite applies for dimensions with small gradients [67]. ADAGRAD alleviates the burden of carefully choosing an optimal learning rate. Although the base learning rate η has to be chosen, it has a lot less of an effect as compared to standard SGD. One issue with ADAGRAD is its sensitivity to initial conditions. If the gradients are very large at the start of training, it will slow down convergence significantly for the rest of the training. Additionally, the continually growing denominator in Equation 3.29 will eventually result in a learning rate of 0, which might be an issue for hard learning tasks.

3.4.4 ADADELTA

An approach to combat the issues of ADAGRAD, is called *ADADELTA* [68]. To deal with the ever decaying learning rate, the sum of squared gradients is restricted to a fixed window size of recent gradient updates. The running average of the sum of squared gradients at time t, $E[\mathbf{g}^2]_t$, is given by

$$E[\mathbf{g}^2]_t = \rho E[\mathbf{g}^2]_{t-1} + (1-\rho)\mathbf{g}_t^2 \tag{3.30}$$

where ρ is a constant similar to the momentum constant. Like in ADAGRAD, we take the square root of this sum. Taking the square root of $E[g^2]_t$ basically results in the root mean square [3] over the windowed gradients up to time t

$$\text{RMS}[\mathbf{g}]_t = \sqrt{E[\mathbf{g}^2]_t + \epsilon} \tag{3.31}$$

where ϵ is a small constant to ensure numerical stability in calculations. The update is then given with

$$\Delta_{\theta_t} = -\frac{\eta}{\text{RMS}[\mathbf{g}]_t}\mathbf{g}_t \tag{3.32}$$

In [68], the ADADELTA algorithm is compared to standard SGD, SGD with momentum and ADAGRAD in a handwritten digit classification task. The ADADELTA algorithm manages to match up to the fast initial convergence of ADAGRAD. While ADAGRAD eventually slows down due to the ever decaying learning rate, ADADELTA reached convergence faster.

[3] $\text{RMS}[x] = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)}$

3.5 Activation Functions

For the sake of comparison we list all activation functions used in this work in Sections 3.5.1-3.5.4. The reasoning for their use is partially explained here as well as in the description of the model components in which they are used.

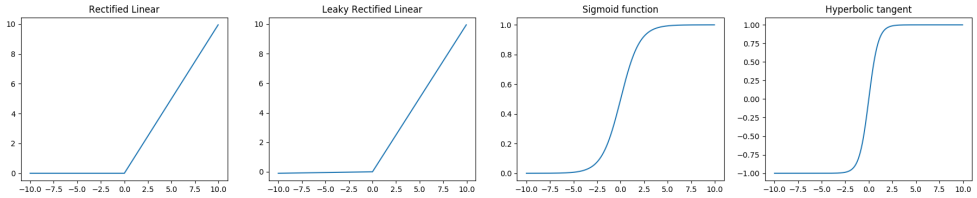


Figure 3.8: A comparison of the 4 activation functions used in this work

3.5.1 The Sigmoid Function

The sigmoid function, or the standard logistic function, is a frequently used activation function and is defined by

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.33}$$

Sigmoid functions are nonlinear and therefore give neural networks the capability of approximating nonlinear functions.

3.5.2 The Hyperbolic Tangent Function

The hyperbolic tangent, which is also a sigmoid function, is often used as an activation function in neural networks. The function is defined by

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.34}$$

One reason for using the hyperbolic tangent rather than the sigmoid function, is that the hyperbolic tangent produces more aggressive gradients around zero, which can be beneficial if convergence is slow. A comparison of the derivatives of the hyperbolic tangent and the sigmoid function is shown in Figure 3.9. Additionally, as opposed to the standard logistic function, tanh is symmetric

about the origin, resulting in the average of the output being close to zero [69]. This has the same effect as normalizing the input data and aids convergence [4].

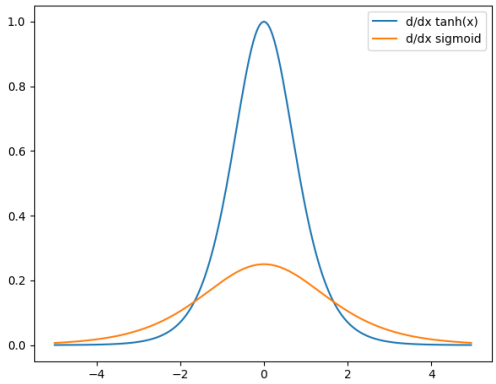


Figure 3.9: The derivatives of the sigmoid function and the hyperbolic tangent

3.5.3 The Rectified Linear Function

A unit that has rectified linear activation is referred to as a rectified linear unit (ReLU). The rectified linear function returns its input if it is larger than 0, otherwise, it returns 0

$$\text{ReLU}(x) = \max(0, x) \tag{3.35}$$

One thing to note is that ReLU is not differentiable at $x = 0$, but we can define the derivative of ReLU at $x = 0$ to be 0. The range of ReLU is $[0; \infty)$. The sigmoid functions taper off, the standard logistic function has the range $(0; 1)$ and tanh has the range $(-1; 1)$. If the input vector $\mathbf{x}$ of a layer with ReLU activations is high, the outputs will be high, while the sigmoid functions taper the output down. what this means is that using ReLU can be more efficient, allowing the network to learn faster when the model is far off target, but sigmoid functions tend to be more stable.

When it comes to gradient calculations, ReLU is much less computationally expensive as compared to the sigmoid functions, which is another reason why ReLU is often used for activations.

[4] One way to think about this is to imagine training a neural network on a dataset of only positive samples. In that case, the gradient update of all the weights of the network will have the same sign. This can lead to the gradient direction to zigzag, which leads to inefficient training

3.5.4 Leaky Rectified Linear Function

The leaky rectified linear unit (LReLU) is similar to the regular ReLU but has a non-zero gradient over the entire domain, unlike the regular ReLU. The leaky ReLU is given by

$$\text{LReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & \text{otherwise} \end{cases} \tag{3.36}$$

In [70], the authors theorize that leaky ReLUs will allow for more robust parameter optimization by sacrificing *hard-zero* sparsity for a gradient over the whole domain. The derivatives of these two rectifier functions are given as

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

$$\frac{\partial \text{LReLU}(x)}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ 0.01 & \text{otherwise} \end{cases} \tag{3.38}$$

One potential issue with ReLU is that the gradient of ReLU is always zero if the unit is not active. If a unit is never activated initially, a gradient based optimization might never changes the weights of the unit such that it activates.

In [70] however, they only manage to slightly improve the rate of convergence using leaky ReLUs as compared to regular ReLUs.

3.6 Weight Initialization

The models discussed in Section 3.2 rely largely on RNNs and LSTM networks are often used, for example in the LAS model discussed in Section 3.2.4. LSTM networks rely heavily on sigmoid activations for gating as discussed in Section 3.1. When training deep networks using sigmoid activations it is necessary to carefully consider the initialization of model weights. For this, *LeCun* initialization [69] is often used. Using LeCun initialization, each weight value w_{ij} in the weight matrix $\mathbf{W}$ is drawn from a zero-centered normal distribution[5], such that $w_{ij} \sim \mathcal{N}(0, \frac{1}{m})$ where m is the *fan-in*, or the number of unit inputs. This initialization strategy ensures that the output distribution of each layer in the model will start off approximately the same.

[5]The univariate normal PDF centered at the mean, μ, with standard deviation σ^2 is given as $\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{w\sigma^2}}$

The output of sigmoid functions saturates quickly with larger inputs and that kills the gradients. Therefore, we initialize the weights in such a way that the sigmoid is activated primarily in the linear region to begin with. This allows for confident learning in the beginning as well as learning the linear part of the mapping first, before learning the more complex nonlinear part of the mapping. In Figure 3.10, we show an example of a bad weight initialization. Here, we have a model $M(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, that given some training data $\mathcal{D}$ tries to learn to predict a class y in $\{0, 1\}$. In the first row in Figure 3.10, we show the value $M(\mathbf{x})$ scattered for all training samples as well as the sigmoid function. In the second row, we show the gradient value, $\partial M(\mathbf{x})/\partial \mathbf{x}$ for all $\mathbf{x} \in \mathcal{D}$. For the model showcased in Figure 3.10, we initialized the weights

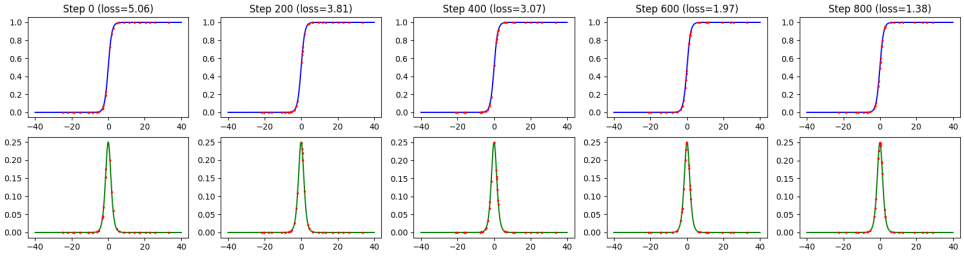


Figure 3.10: Learning is slow for too high initialization of weights

with $W_{ij} \sim \mathcal{U}(0, 2)$ [6]. This weight initialization strategy has not worked well and we can see why in Figure 3.10. The sigmoid is not activated in the linear region and the values are saturated, which results in vanishing gradient values. Figure 3.11 shows the result of the same experiment, using LeCun

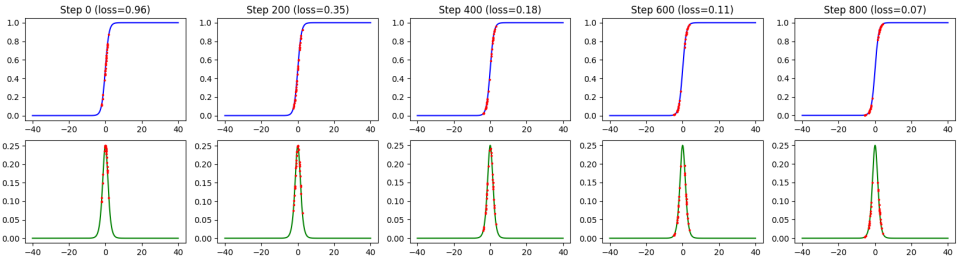


Figure 3.11: LeCun initialization of weights results in faster training

initialization. We can see that the model learns quickly and confidently. It is

[6] The uniform PDF over the range $[a; b]$ is given as $\frac{1}{b-a}$

reasonable to consider why we do not just initialize each weight parameter as 0. First off, if the value of each weight is initialized with the same value, back-propagation will update each weight parameter of each layer equally, making most of the units in each layer useless. Furthermore, for deeper networks, too small weight initializations will result in vanishing gradients. For the sake of

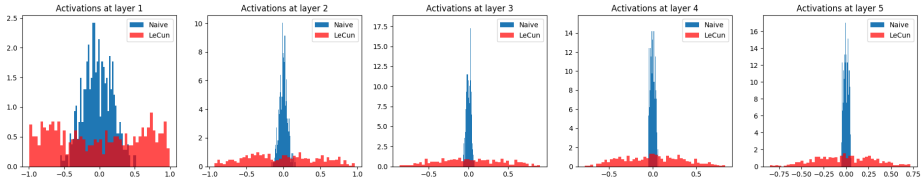


Figure 3.12: Comparing activations of different weight initialization strategies in a deep network

experiment, a 5 layer network with 500 hidden units with tanh activation is created. In the first experiment, the weights are normally distributed around zero with a very low standard deviation. In the second experiment, LeCun initialization is used. The activation values for those experiments are shown in Figure 3.12. The activations of the first model quickly shrink. During back-propagation, small weights will calculate small gradients since the gradients are proportional to the value of weights. For the experimental model used here, using any more than 7 layers, leads to vanishing gradient issues for the naively initialized model. It is therefore also important when using sigmoid activations to initialize weights large enough.

Chapter 4

Architecture and Methodology

We aim to make changes to a baseline sequence-to-sequence ASR model which enables us to further train the model in an unsupervised manner. Using ideas from [15], we propose a baseline similar to LAS [2] and then make three key modifications to leverage unlabeled data. The proposed model consists of

- The baseline ASR, similar to LAS
- A text autoencoder
- A speech autoencoder
- A discriminator network for adversarial training

where both autoencoders share parameters with the baseline. We will now describe the model implemented in this work in full and the corresponding training procedures.

Although the baseline ASR is trained end-to-end, the three other components are trained separately. We start by doing unsupervised training to create a pre-trained seed model. By doing this, the goal is to tune the parameters of the baseline without any supervision. The baseline is finally trained on the limited amount of transcribed speech data available. Using the pretrained baseline as a seed model, The training procedure performed for each experiment can therefore be summarized by the following steps

1. Train the text autoencoder.
2. Adversarially train the discriminator and the baseline ASR encoder.
3. Train the speech autoencoder.

4. Train the baseline ASR model on a limited amount of transcribed speech data.

The first three procedures of unsupervised training are performed multiple times in this order. The reason for this particular order is that in [15], text autoencoder and adversarial training were shown to have the largest positive impact on training and validation results. These procedures will affect the baseline model by sharing parameters with the baseline or by adversarial training and is further explained in Sections 4.1-4.4 Parameter sharing and overall

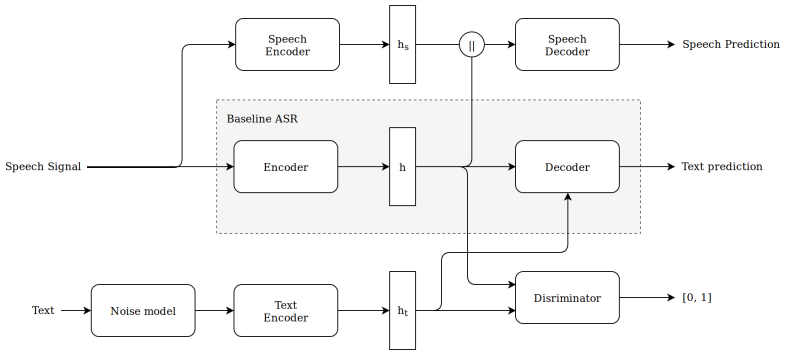


Figure 4.1: The topology of the entire model shows how parts of the model interact during training

model topology is shown in Figure 4.1. One thing that is not included in that figure and the previous discussion is the language model. The reason for this is that the language model used in this work is trained separately and no other component depends on it.

4.1 The Baseline Model

The baseline ASR model used in this work is very similar to the one described in LAS. The model contains three main modules, the listener, the speller and attention which correspond to the same modules of the LAS model. A novel diagram of this baseline model is shown in Figure 4.2. The baseline ASR model is trained end-to-end on a labeled speech corpus. During training, we minimize the *cross entropy loss* between the target text and the text prediction. To motivate this choice of an error metric we first need to discuss classifications in general.

Classification problems in machine learning use statistical models to approximate the probability distribution over possible labels, given training data

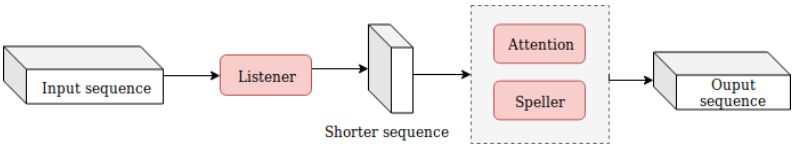


Figure 4.2: Simplified diagram of the LAS model

$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^d$ and normally $y_i \in \mathbb{Z}$ for $i = 1, \dots, n$. That is, we aim to maximize the posterior probabilities, which are parameterized by $\boldsymbol{\theta}$ by the model

$$P(y|\mathbf{x}) \equiv P(y|\mathbf{x}; \theta) \tag{4.1}$$

By using Bayes' rule we can deduce that maximizing the likelihood of the data, $P(\mathbf{x}|y) \equiv P(\mathbf{x}|y; \boldsymbol{\theta})$ corresponds to maximizing the posterior since $P(\mathbf{x}|y; \boldsymbol{\theta}) \propto P(y|\mathbf{x}; \boldsymbol{\theta})$. One frequentist approach to maximizing the posterior is to maximize the likelihood of the training data. This is referred to as maximum likelihood estimation (MLE) and the optimal parameter value $\boldsymbol{\theta}^*$ is given as

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmax}} \frac{1}{|\mathcal{D}|} \prod_{(\mathbf{x},y)\in\mathcal{D}} P(\mathbf{x}|y; \boldsymbol{\theta}) \tag{4.2}$$

where we have made the assumption that the label distributions are independent and identically distributed. Since the logarithm is a monotonically increasing function, Equation 4.2 corresponds to maximizing the log likelihood

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmax}} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log P(\mathbf{x}|y; \boldsymbol{\theta}) \tag{4.3}$$

which is preferred in optimization since otherwise the likelihoods would become increasingly small until we run out of possible precision.

A common loss function for classification problems is the cross entropy loss. For a classification problem with C classes and a training dataset $\mathcal{D}$ the cross entropy is given as

$$\mathcal{L}(\mathcal{D}, \boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \log \left(\frac{\exp(s_y)}{\sum_{k=i}^{C} \exp(s_k)} \right) \quad \text{where} \quad s_k = P(k|\mathbf{x}, \boldsymbol{\theta}) \tag{4.4}$$

where we softmax the model output for each sample to ensure that the prediction is in fact a probability. Equation 4.3 can be written as

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \log P(\mathbf{x}|y;\theta) \tag{4.5}$$

By comparing Equations 4.5 and 4.4, we can see that minimizing the cross entropy loss is equal to minimizing the negative log likelihood, which is in turn equal to maximizing the log likelihood.

When training the model, a part of the encoder input is the ground truth of the previous encoder time step, y_{t-1}. During inference however, the ground truth is not available and we always supply the previous prediction instead. This can lead to an undertrained model since the decoder is only used to correct previous predictions, not possibly wrong previous predictions. To alleviate this effect, we supply the actual previous prediction instead of the ground truth as proposed in [71]. With a certain probability, instead of supplying the ground truth to the current encoding step, we sample from the previous prediction's probability distribution. The probability of doing teacher forcing instead, i.e. supplying the ground truth, is referred to as the *teacher forcing rate* in section 5.2.

4.1.1 The Listener

The listener is an acoustic model encoder. The listener is a multi-layer recurrent neural network and uses bidirectional long short-term memory cells. The listener has 4 layers. The first layer operates directly on the input. Typically, the output of a deep BLSTM network at time step i and layer j is given by

$$h_i^j = \text{BLSTM}(h_{i-1}^j, h_i^{j-1}) \tag{4.6}$$

That is, the input to the i-th cell in the j-th layer is the output from the i-th cell in the previous layer. The decoder however uses pBLSTM cells which means that the input to the i-th cell in the j-th layer is a concatenation of the two cells in the previous layer that are *closest* to it as shown in Figure 4.3. Formally, we write it as

$$h_i^j = \text{pBLSTM}(h_{i-1}^j, (h_{2i}^{j-1}, h_{2i+1}^{j-1})) \tag{4.7}$$

The concatenation performed by each pBLSTM will double the hidden dimension of each input and at the same time, the temporal dimension will be reduced by a factor of two. After running through the three pBLSTM layers of the listener, the temporal dimension has been decreased in total by a factor of 8.

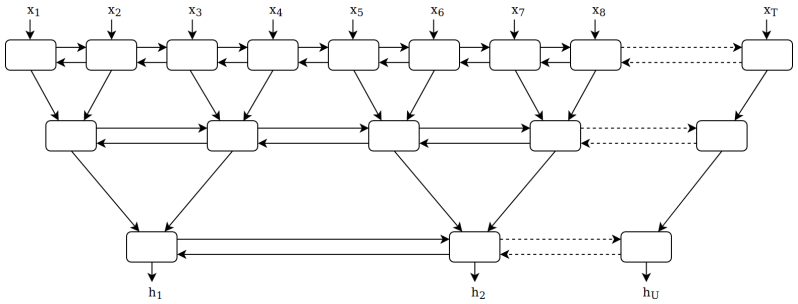


Figure 4.3: The topology of the listener

4.1.2 The Attention Module

The attention mechanisms used in the model proposed in [56] are categorized as *additive*, LAS however uses *content-based attention*, where the alignment model uses cosine similarity

$$e_{i,j} = a(s_i, h_j) = \phi(s_i)^T \psi(h_j)$$

where ϕ and ψ are simple feed forward networks. Here, the attention mechanism learns to align hidden state outputs from the encoder, the down-sampled input, to characters in the outputs. As with the attention model described in Section 3.2.4, a context vector c_i is computed by the attention mechanism, which is a linear blend of all the listener outputs

$$c_i = \sum_{j=1}^{J} \alpha_{i,j} h_j \tag{4.8}$$

where $\alpha_{i,j}$ represent a probability distribution over listener timesteps, where $\alpha_{i,j} = \text{softmax}(e_{i,j})$.

4.1.3 The Speller

The speller is a character decoder, consisting of two LSTM layers. At every time step, the speller emits a hidden state which is transformed into a character prediction with a simple feed-forward network with softmax over all possible characters. At every timestep the first LSTM cell, i.e. the one in the first layer, receives the previous character prediction as input or the "<sos>" token at the first time step, representing the *start of sentence*. The state of each speller LSTM cell, s_i is given by the previous decoder state, s_{i-1} the previous character predicted y_{i-1} and the context vector c_{i-1}. The input to each cell is the

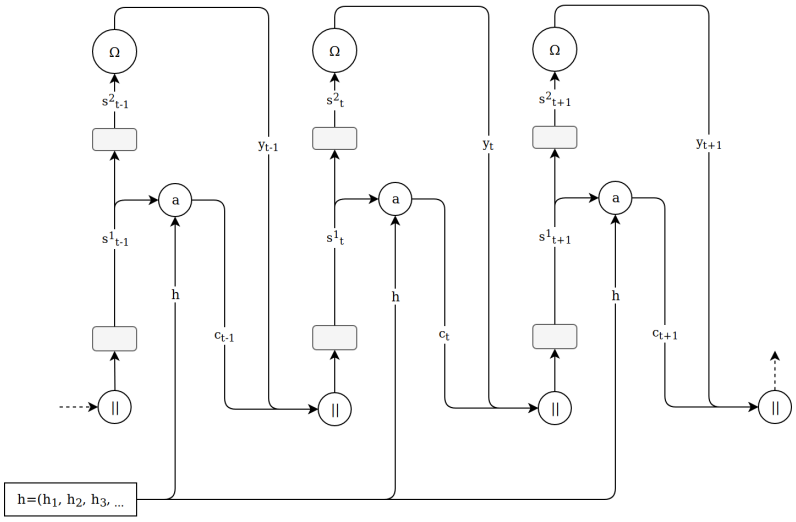


Figure 4.4: The topology of the speller where a represent the attention module

concatenation of the previous prediction of and the context vector , $[y_{i-1}, c_{i-1}]$. At each decoding step, a hidden state is emitted by the speller. A simple feed-forward network with softmax over characters will transform each state s_i into a character prediction y_i.

$$x_t = y_{t-1} \| c_{t-1} \tag{4.9}$$
$$f^1_t = \sigma(\mathbf{W}^1_{xf} x_t + \mathbf{W}^1_{sf} s^1_{t-1} + b^1_f) \tag{4.10}$$
$$i^1_t = \sigma(\mathbf{W}^1_{xi} x_t + \mathbf{W}^1_{hi} h_{t-1} + b^1_i) \tag{4.11}$$
$$k^1_t = f^1_t \odot k^1_{t-1} + i^1_t \odot \tanh(\mathbf{W}^1_{xk} x_t + \mathbf{W}^1_{sk} s^1_{t-1} + b^1_k) \tag{4.12}$$
$$o^1_t = \sigma(\mathbf{W}^1_{xo} x_t + \mathbf{W}^1_{ko} k^1_{t-1} + b^1_o) \tag{4.13}$$
$$s^1_t = o^1_t \odot \tanh(k^1_t) \tag{4.14}$$

where $\|$ denotes concatenation and to avoid confusion, we denote the LSTM cell state with k rather than c since we denote the context with c. The second layer LSTM emits the final state s^2_t with similar calculations

$$f^2_t = \sigma(\mathbf{W}^2_{xf} s^1_t + \mathbf{W}^2_{sf} s^2_{t-1} + b^2_f) \tag{4.15}$$
$$i^2_t = \sigma(\mathbf{W}^2_{xi} x_t + \mathbf{W}^2_{hi} h_{t-1} + b^2_i) \tag{4.16}$$
$$k^2_t = f^2_t \odot k^1_{t-1} + i^2_t \odot \tanh(\mathbf{W}^2_{xk} s^1_t + \mathbf{W}^2_{sk} s^2_{t-1} + b^2_k) \tag{4.17}$$
$$o^2_t = \sigma(\mathbf{W}^2_{xo} s^1_t + \mathbf{W}^2_{ko} k^2_{t-1} + b^2_o) \tag{4.18}$$
$$s^2_t = o^2_t \odot \tanh(k^2_t) \tag{4.19}$$

Finally, the probability over all possible characters is given by

$$y_t = \text{softmax}(\Omega(s_t^2)) \tag{4.20}$$

where Ω is a simple single layer feed-forward network. The character prediction at each time step is the character with the highest probability. A detailed diagram of the speller's topology is shown in Figure 4.4 where a denotes the attention module and h contains all encoder outputs.

4.2 The Speech Autoencoder

The speech autoencoder consists of the baseline model ASR encoder, an additional speech encoder used to capture non-linguistic content of the speech signal and a speech decoder. The speech autoencoder is trained on an unlabeled speech corpus. By training the speech autoencoder, we aim to minimize the difference between the input and the output. To that end, we minimize the *smooth L1 loss*. The element wise smooth L1 loss for a ground truth and prediction pair, $y, \hat{y} \in \mathbb{R}^n$ is given by

$$\mathcal{L}(\hat{y}, y) = \frac{1}{n} \sum_i z_i \tag{4.21}$$

where z_i is given by

$$z_i = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & \text{if} \quad |\hat{y}_i - y_i| < 1 \\ |\hat{y}_i - y_i| - \frac{1}{2}, & \text{otherwise} \end{cases} \tag{4.22}$$

The smooth L1 loss is therefore quadratic for small terms and linear for larger terms. By keeping the loss linear for larger terms, the goal is for the loss criterion to be less sensitive to outliers and prevent exploding gradients.

4.2.1 The Speech Encoder

The speech encoder is convolutional neural network (CNN). It consists of three convolutional layers with batch normalization, ReLU non-linearities (see Section 3.5) and Max-pooling. The first layer convolves in frequency, while the second two convolve in time. In each layer, we max-pool the input, with increasingly sized kernels. The last max-pool is over the whole input, which produces a single vector representing the whole utterance. Convolutional neural networks have been widely used for image classification tasks, largely since

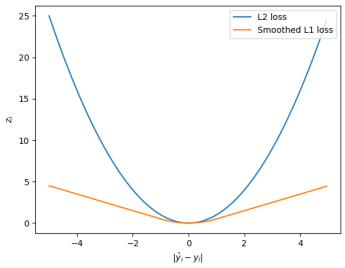


Figure 4.5: Smooth L1 loss compared to L2 loss

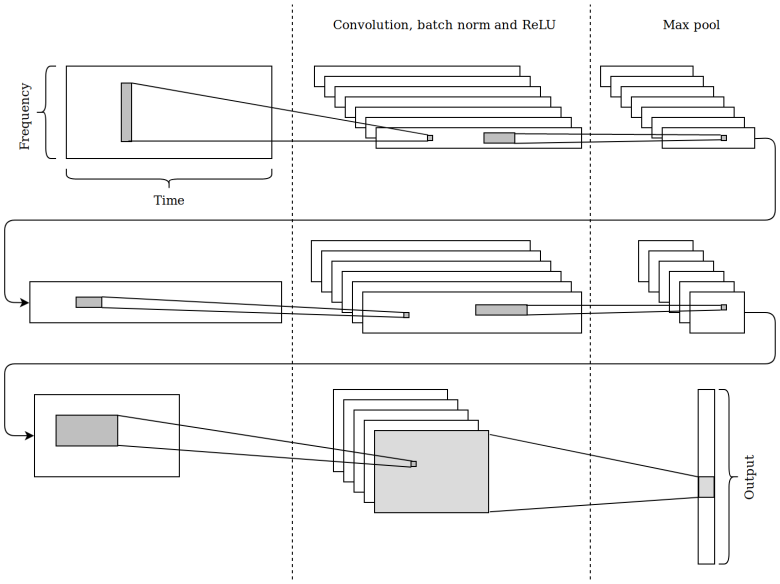


Figure 4.6: Visualization of all layers of the speech encoder

CNNs are invariant to transformations on the input and handle inputs of grid-like topology well. Consider the task of recognizing handwritten digits. Each sample of each digit will vary slightly in rotation and scale. A feed-forward network could be trained on enough data to recognize digits, but such a network would not utilize the fact that there is a strong relationship between neighboring pixels in images. CNNs, on the other hand, take this property into consideration. [65]. The Convolutional neural network normally consists of convolutional layers, layers of non-linearities, pooling layers and fully connected layers. Here, we will consider two-dimensional discrete inputs where the input can consist of multiple *channels*. Images are often stored in three different color channels, red green and blue. In this work, the input only has 1 channel containing the amplitude in the input spectrograms. A convolution

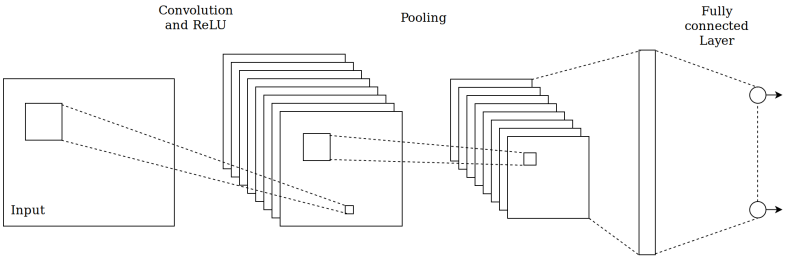


Figure 4.7: Diagram of a simple CNN

of two functions, f and g is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{4.23}$$

The convolutional layer of CNN performs a discrete convolution over the input x and a two-dimensional kernel K

$$(X * K)(i,j) = \sum_m \sum_n X(i,j)K(i-m, j-n) \tag{4.24}$$

Each kernel can be thought of as a trainable weight matrix. Each kernel runs over the input, convolving a part of the input at a time with the kernel weights. This kernel convolution operation is what gives rise to the transformation invariance of CNNs. In the simple example shown in Figure 4.8, a single kernel

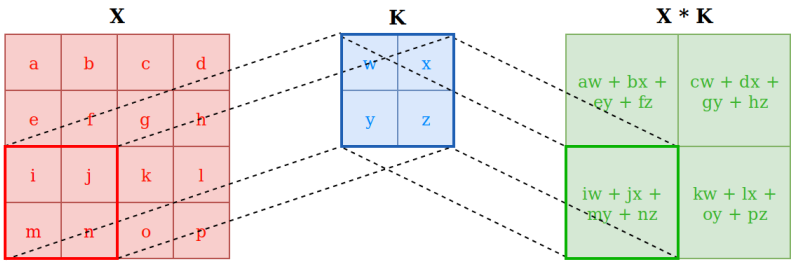


Figure 4.8: The input X is convolved with the kernel K

is used to produce a single response map, $(X * K)$, but often multiple kernels are used to produce multiple response maps as shown in figure 4.7. The kernel has a height and width of 2 and in this example and is shifted by 2 spaces for each application of the kernel. This shift is referred to as the *stride* of the convolutional layer.

Next, some non-linear activation is applied to the result, e.g. a rectified linear unit (ReLU) layer which increases the non-linearity of the decision surface. Finally to improve generalization, a subsampling layer is used to achieve shift-invariance. For this, max-pooling is most commonly used [72]. Max-pooling is applied by taking the maximum of the region a certain filter sees at any time and supplying it to the output. We do so by, similarly to the convolution layers, moving a frame over the convolution layers and taking the maximum of that frame. We show the result of a max-pooling layer in Figure 4.9. Again, we

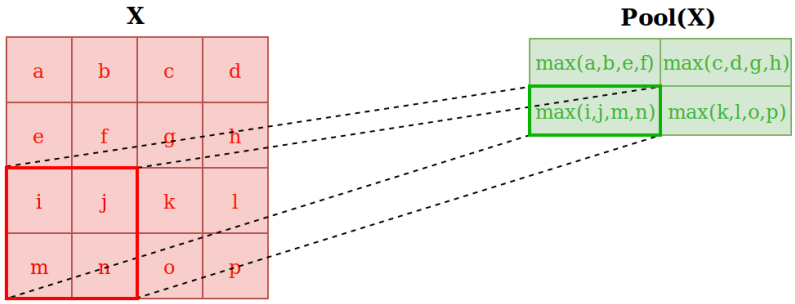


Figure 4.9: The feature map is max-pooled with a 2-dimensional kernel

have decided to use a 2-dimensional kernel, with height and width of 2 and a stride of 2. This order of operations is often repeated multiple times, but in Figure 4.7 a simpler example is shown where with a fully connected layer the feature map produced by each kernel is concatenated into a single output feature vector.

Batch normalization is a transformation that is applied batch-wise to training data, which allows for higher learning rates and less careful parameter initialization [73] and is often used when training CNNs. To apply batch normalization on $\mathcal{B} = (x_1, \ldots, x_m)$, first we calculate the batch mean and variance

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{4.25}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{4.26}$$

Then, each sample is normalized

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{4.27}$$

And finally, each sample is scaled and shifted by trainable parameters from the parameter vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$

$$y_i = \gamma_i \hat{x}_i + \beta_i \tag{4.28}$$

4.2.2 The Speech Decoder

The speech decoder predicts a single uncompressed frame of the speech signal, that is the decoder predicts the frames that the listener takes as input. The input to the speech decoder is a concatenation of the whole utterance vector representation from the speech encoder and a single state from the listener. The speech decoder is a feed-forward network, consisting of two leaky ReLU layers and a linear layer. By incorporating the ASR encoder in this way with

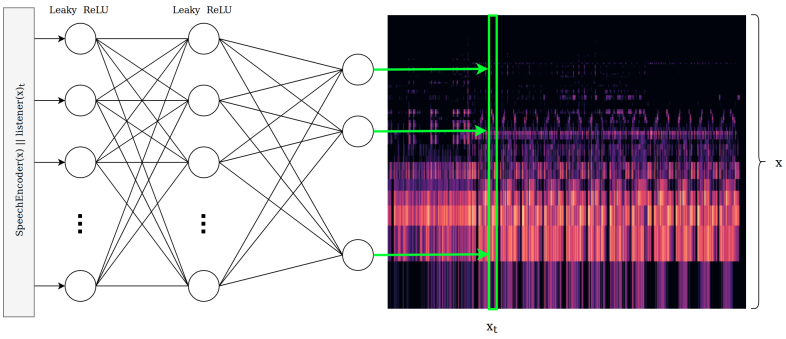


Figure 4.10: The speech decoder predicts a single frame from the original signal at a time

the task of capturing the linguistic content of the speech signal, we aim to pre-train the baseline ASR in an unsupervised manner. Theoretically, by training the speech autoencoder, the ASR encoder should become better at efficiently encoding the linguistic contents of the signal and that should result in improved accuracy.

4.3 The Text Autoencoder

The text autoencoder consists of three components, the text encoder, a *noise model* and a text decoder. The text decoder is the ASR decoder of the baseline model and is shared with the text autoencoder.

Noise is added to the character inputs to make the encoder more generalized since encoding characters is a very simple learning task. Following [15],

we only drop characters with a certain probability and do not shuffle or add characters to each sequence.

The text encoder consists of a *character embedding* [74] and a deep LSTM network. Character embeddings are based on skip-grams and similarly are trained to find a representation of the input character such that it becomes a good predictor for the surrounding characters in the sequence. The output of the text encoder is then passed on to the listener component to be decoded.

The text decoder has a similar purpose to the speech encoder, incorporate unsupervised training into the baseline ASR training. In the case of the text encoder, which shares parameters with the baseline ASR decoder, we aim to pre-train the ASR encoder in an unsupervised manner which theoretically could improve decoding results.

The text autoencoder is trained on an unlabeled text corpus. The text autoencoder is trained with the same goal as the speech autoencoder, to minimize the difference between the input and the output. Cross entropy is also used as a loss metric for the text autoencoder.

4.4 The Discriminator

The discriminator is a simple feed-forward network. The discriminator receives as input, a vector which is either an output from the listener or the text encoder. In Figure 4.11, the simple topology of the discriminator is shown. We indicate a single output from the listener with the subscript t_1, in reference to the listener encoding steps, and from the text encoder with t_2, in reference to the text encoder steps. The discriminator outputs a scalar value in the range

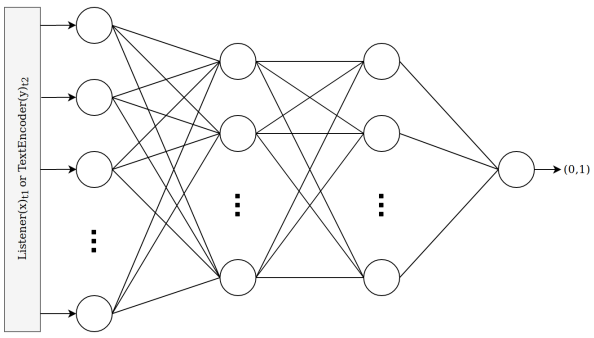


Figure 4.11: The discriminator seeks to predict whether the input was generated by the listener or the text encoder

$(0, 1)$. For adversarial training, we treat the output of the text encoder as the

ground truth and the listener is treated as the generator. The discriminator is trained to assign high valued labels to samples generated by the text encoder and low valued labels to the listener.

The discriminator outputs predictions in the range $[0, 1]$ and is trained to output high values for ground truth predictions and low values for the generator predictions. As in [15] and further explained in section 6.3.2, we will treat the text encoder as the ground truth source and the ASR encoder as the generator for adversarial training.

As recommended in [75] and implemented in [15], we apply *label smoothing* to the ground truth labels during training. That is, instead of supplying $\{1, \ldots, 1\}$ as training labels for ground truth samples to the discriminator, we supply $\{1 - \epsilon, \ldots, 1 - \epsilon\}$ where ϵ is some small value. The reasoning behind this is to encourage the discriminator to estimate soft probabilities rather than hard probabilities which tend to lead to overfitting. For a labelled speech and text pair $(\mathbf{x}, y)$, the discriminator, parameterized by $\boldsymbol{\theta}_D$, seeks to maximize

$$\boldsymbol{\theta}_G^* = \underset{\boldsymbol{\theta}_D}{\operatorname{argmax}} \log(D(\Theta(y))) + \log(1 - D(G(x)) \tag{4.29}$$

where D is the discriminator, G is the ASR encoder generator and Θ is the text encoder. The generator, parameterized by $\boldsymbol{\theta}_G$ is trained in parallel with the discriminator during adversarial training. The generator seeks to maximize

$$\boldsymbol{\theta}_G^* = \underset{\boldsymbol{\theta}_G}{\operatorname{argmax}} \log(D(G(x)) \tag{4.30}$$

To achieve this, the discriminator and generator are trained to minimize the binary cross entropy (BCE) loss, a special case of cross entropy for classification problems with two possible output labels. For a training dataset $\mathcal{D}$ and model parameters $\boldsymbol{\theta}$, BCE loss is given by

$$\mathcal{L}(\mathcal{D}, \boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} y \log P(y|\mathbf{x}; \boldsymbol{\theta}) + (1 - y) \log(1 - P(y|\mathbf{x}; \boldsymbol{\theta})) \tag{4.31}$$

4.5 The Language Model

In both [2] and [15], word-level n-gram language models were used to rescore beams in the beam search during inference. Here, we opt for a simpler RNN character-level language model. The main part of the language model is a deep LSTM. The LSTM receives a character y_t and outputs a prediction for the next

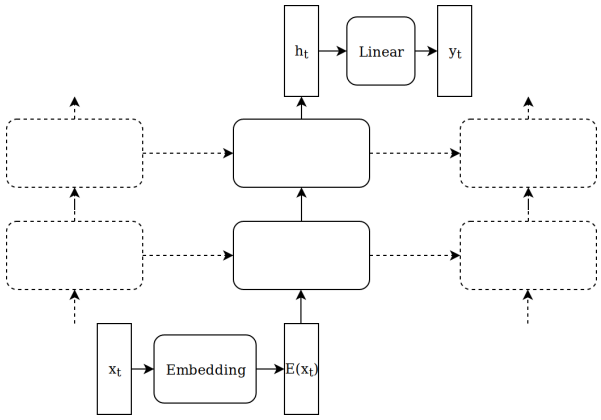


Figure 4.12: One prediction step of the language model

character y_{t+1}. As with the text autoencoder, we use a character embedding to encode each character as a distributed vector. The character embedding is then fed through a deep RNN, outputting a corresponding hidden vector. The hidden vector is finally passed through a linear layer to achieve the dimensionality of the character space.

The language model is trained on a large text corpus. During training we try to minimize the *cross entropy loss* of each prediction.

Chapter 5

Experimental Setup

5.1 Choice of Data

The largest open-sourced parallel text and speech corpus for Icelandic is the *Almannarómur* corpus. Over 500 participants recorded themselves reading multiple sentences out loud, up to 350 utterances per participant. Text from Icelandic news articles collected in 2010-2011 constitutes a large part of the corpus. Additionally, rare Icelandic words, names of people and places were added to the corpus. The total composition of the Almannarómur corpus is shown in Table 5.1. URLs were added since at the time URLs could not contain special Icelandic characters. All in all, over 120,000 utterances were

News stories	50%
Rare tri-phones	10%
Names of streets	10%
Names of people	10%
Miscellaneous	10%
Countries and capitals	5%
URLs	5%

Table 5.1: The composition of the Almannarómur corpus

recorded in over 150 hours of speech recordings. Later, a manually verified subset of Almannarómur called *Málrómur* [5]. The original recordings were manually inspected by evaluators and a clean corpus was created. Of the original 120,000 utterances, about 108,000 were deemed correct. The most common source of error was participants reading the text partially or completely

wrong. Additionally, long silences were trimmed off both ends of the recordings. This is likely to be beneficial for the alignment model of the ASR.

Since the language model is only trained on text data, we can use additional text data sources for that part of the training procedure. The *Risamálheild* dataset [76] includes more than 1 billion running words, mostly sourced from news articles, parliamentary speeches, adjudications, books and other contemporary texts. Each word in the dataset is stored with its *tag* and its *lemma*. By lemma, or just dictionary form, we are referring to the source word that all inflected forms of that word belong to. For example, the word "better" belongs to the lemma "good". The most common tags in the dataset are adverb, conjunction and preposition. The composition of the Risamálheild corpus is shown in Table 5.2, As the character-level language model used here is quite

Text Genre	Word Count	No. of docs.	Period
Newspaper Articles	796,526,434	3,029,985	1998-2017
Parliamentary Speeches	210,699,883	380,557	1911-2017
Adjudications	92,696,289	23,634	1999-2017
Published Books	5,729,543	120	1980-2008
Transcribed Radio/Tv News	54,129,050	313,749	2004-2017
Sports News Websites	47,431,733	280,838	2002-2017
Regulations	26,038,153	12,038	1275-2017
Current Affair Blogs	10,511,776	43,678	1973-2017
Informational Articles	10,796,107	55,091	2000-2107
Lifestyle	4,027,506	14,671	2010-2017
Total	1,261,026,503	4,154,528	2010-2017

Table 5.2: The composition of the Risamálheild corpus

simple, we do not need to train on the complete corpus since the model converges relatively early. Instead, we use a smaller sample of the Risamalheild corpus, consisting of contemporary news articles.

5.2 Model Configuration

All hyperparameters are listed in Table 5.3 Here, n-m-LSTM represents an n-layer LSTM with m-dimensional state sizes. FC-<a>-n-m represents a fully connected layer with n inputs and m inputs, using <a> for activation. Conv-[m,n]-k-ReLU-BN represents a convolutional layer, with a kernel size of $[m, n]$, k output channels and batch normalization is applied after the convolution.

Spectral analysis	40-band mel-based filter bank, stride: 10ms, window size: 25 ms, Hann window
Listener	1-256-BLSTM $\rightarrow$ 3-256-pBLSTM
Attention	
ϕ	FC-Tanh-256-128
ψ	FC-Tanh-512-128
Speller	2-256-LSTM (teacher forcing rate $= 0.9$)
Text autoencoder	
Encoder	128-D character embedding $\rightarrow$ 2-256-LSTM
Speech autoencoder	
Encoder	Conv-[36,1]-32-ReLU-BN $\rightarrow$ Max pooling: stride=1, shape=[3,1] $\rightarrow$ Conv-[1,5]-64-ReLU-BN $\rightarrow$ Max pooling: stride=1, shape=[5,1] $\rightarrow$ Conv-[1,3]-256-RelU-BN $\rightarrow$ Max pooling: stride=1, shape=[*, *]
Decoder	FC-LReLU-(512+256)-(512+256) $\rightarrow$ FC-LReLU-(512+256)-(512+256) $\rightarrow$ FC-LReLU-(512+256)-(8*40)
Discriminator	FC-ReLU-512-128 $\rightarrow$ FC-ReLU-128-128 $\rightarrow$ FC-sigmoid-128-1 (label smoothing: $\epsilon = 0.1$)
Language model	128-D character embedding $\rightarrow$ 2-128-LSTM $\rightarrow$ FC-linar-128-50
language model weight (λ)	0.1
Noise rate	0.2

Table 5.3: Training and model hyperparameters

The last pooling layer of the speech autoencoder CNN uses a kernel size such that the pooling is operated over the entire utterance, resulting in a single vector representation of the input utterance.

5.3 Preprocessing

Each utterance is represented as a 40-band Mel-base spectrogram as listed in Table 5.3. However, all text data, the LM corpus and the ASR label data, is additionally preprocessed. Characters that are considered in this project are

The Latin alphabet	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Icelandic characters	á, é, í, ó, ú, ý, æ, ö, þ, ð
Numerical symbols	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special characters	(" ": space) , (".": period), (",": comma) ("?": question mark)
Tokens	("<" sos token), (">" eos token), ("$" unknown)

Table 5.4: Character composition of labels and predictions

listed in Table 5.4. Each training string sample starts with the "<sos>" token and ends with the "<eos>" token, representing the start and end of the sentence. The model has to learn to emit the <eos> token, otherwise, predictions could go on forever. We append the <sos> token since *something* needs to be input at the first time step of the decoder.

Each string is cleaned of any other symbols, lower-cased and redundant spaces are collapsed. For example, the string "Halló ég heiti : Atli." becomes "<halló ég heiti $ atli.>".

During training, these strings are collected in batches. To achieve that, the samples need to be padded up to the maximum length of the batch. Here, we pad the ends of each string with the <sos> token. The reason for this is that the model is never taught to predict the <sos> token anyway since the loss of predicted <sos> tokens is discarded. We could have used a special pad token for this, but that would have made the character space larger than necessary.

5.4 Training Configuration

For language model training, a batch size of 128 was used, both for training and validation. A batch size of 64 was used for text autoencoder training. All other training procedures trained on and validated batches of 32 samples. The baseline ASR model was validated every 3,000 global training steps, the language model every 10,000 steps and all other parts of the model were validated every 1,000 steps.

First, just the baseline ASR model was trained until convergence. For this, all available transcribed data was used. Then, separate baseline ASR models were generated under different resource conditions as in [15]. The first on approximately 2.5 hours of transcribed speech, the second on 5 hours and the last on 10 hours. For each of these three models, we additionally perform text

and speech autoencoder training as well as adversarial training as explained in Chapter 4.

The default batch size used during training and validation is 32 and was used for the ASR, speech autoencoder, and discriminator training and validation. A batch size of 128 is used for text autoencoder training. A batch size of 128 is used for LM training, where each sample is 200 characters long.

For all training procedures the ADADELTA learning rule is used which is described in detail in Section 3.4.

5.5 Technical Setup

All code is written in *Python* [1]. *Pytorch* [2] was used for model construction and data loading. *Librosa* was used for spectral analysis and feature extraction. [3]. All models were trained on hardware supplied by the Language and Voice lab at Reykjavik University, which includes 5 CUDA ready GPUs.

[1] https://www.python.org/
[2] `https://pytorch.org`
[3] `https://librosa.github.io`

Chapter 6

Results

The results of all experiments performed are here described in detail. We compare the performance of up to 6 models under three different resource conditions. The baseline without any unsupervised training is evaluated for all three resource conditions but to further asses it we train it on all available data and report the results in Section 6.1. The most important results are reported in Sections 6.3-6.4 where we describe the results of both unsupervised training and the supervised baseline training that follows as described in Chapter 4. The language model is also evaluated and finally the best performing model is evaluated using beam search and the language model for decoding.

6.1 Baseline Training Using All Available Data

First the baseline is trained on all available transcribed speech data to gauge how well the baseline performs under good resource conditions. The results of these experiments do not affect the results in Sections 6.3-6.4 and are only reported for the sake of comparison and to get a reference for the rest of the experimentation.

Using the Malromur corpus, we trained the baseline ASR for about 120,000 iterations. Each batch contains 32 samples, resulting in about 3.8 million predictions, or about 30 epochs. The baseline ASR performed very well during training, and confidently converged to very high accuracy and low loss and error. It should be noted here that the WER metric used in this work is averaged over the words in the label. After only a small amount of training, the baseline ASR will often emit too long predictions. Because of this implementation

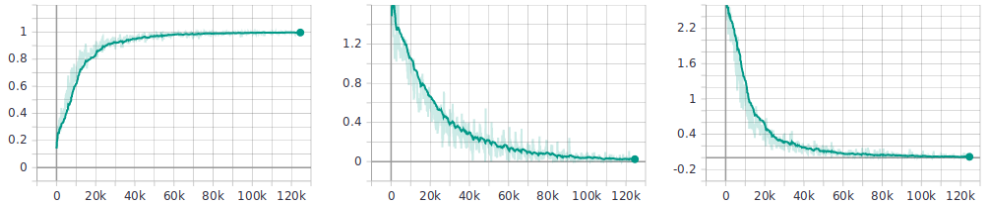


Figure 6.1: Training accuracy, word error rate and loss of baseline ASR

of the metric, the error rate often starts above 1.0. The accuracy metric is a simple character level comparison for the ground truth $\mathbf{y} = (y_1, \dots y_N)$ and prediction $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_N)$

$$\text{accuracy}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\sum_i f_{\text{equiv}}(\hat{y}_i, y_i)}{N} \tag{6.1}$$

$$f_{\text{equiv}}(a, b) = \begin{cases} 1, \text{ if } \quad a = b \\ 0, \text{ otherwise} \end{cases} \tag{6.2}$$

The attention weights are also visualized for some validation samples, shown in Figure 6.2. As we can see, the alignment model is effectively aligning the speech signal to the text. The prediction made by the model is shown above each alignment figure. As we can see, the prediction at step $6,000$ is longer than the one at step $102,000$ resulting in larger attention weights, as reflected in Figure 6.2. The alignment figure at step $102,000$ is also the most *confident* of the three. In the figures, completely black corresponds with a weight of 0 and completely white with a weight of 1. The sum of each row in the figures is 1, and as we can see in the first two figures, the model has a hard time aligning a couple of characters in the output, resulting in less sharp attention. The final alignment figure shows a very sharp alignment. We can also see that the last two characters of the output, "um", are aligned to the same encoder output. The validation accuracy, error rate and loss for the same model are shown in

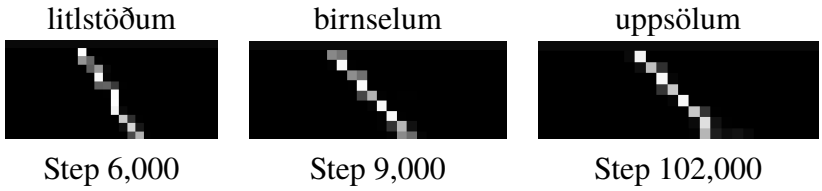


Figure 6.2: Attention weights for the short utterance "uppsölum."

Figure 6.3. Both the accuracy and error curves are promising and show that the model is generalizing quite well. Worryingly though, the loss curve does not

show improvements. It is hard to determine the cause of this since we assume that accuracy and loss are negatively correlated. It is likely that the cause of

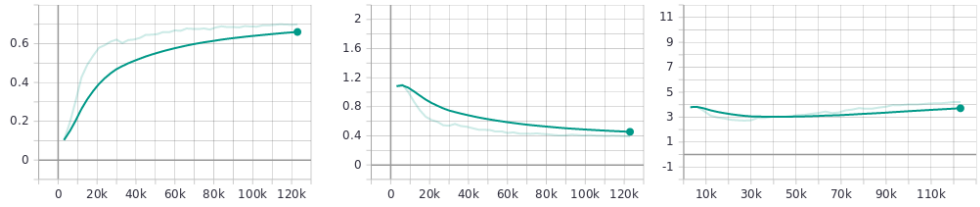


Figure 6.3: Validation accuracy, word error rate and loss of baseline ASR

this bad loss curve is technical since the validation results are otherwise good and the model performs quite well regardless. The baseline ASR trained on

Ground truth	Prediction
"ástríkur"	"ástríkur"
"kirkjuhvoli"	"kirkjuhvoli"
"borgin hætti við niðurskurð"	"borgin hætt við neðurskurð"
"í þjóðbúningi á þjóðminjasafni"	"í þjóðbúningi á þjóð minni safni"
"nokkuð um minniháttar umferðaróhöpp"	"nokkur um minniháttar umfyrðarókork"
"röng ákvörðun samkvæmt endursýningu"	"raunkákurinn samkvæmt endir singingu"
"hér má fylgjast með þróun eldsneytisverðs"	"hér má fylgjast með þróun eldsneytisins"
"ragna komst áfram á kýpur"	"ragna komst áfram á kípur"
"fjórmenningarnir eru sömuleiðis allir erlendir ríkisborgarar"	"fjórmenningarnir eru sveinu leiðis allir erlendir ríkisborgarar"
"steindi.blogspot.com"	"styndi punkt blorð spor kom"

Table 6.1: A selection of validation predictions made by the baseline ASR

all available data reached a character level accuracy of approximately 68% and validation WER of $45,0\%$ without LM rescoring. LAS achieves a WER of $14,1\%$ on the same task, but it should be noted that LAS was trained on 2000 hours of data in the form of three million transcribed utterances opposed to the $100,000$ utterances used in this work. A sample of validation predictions are shown in Table 6.1. Based on manual inspection, the model struggles more with longer phrases. This should not come as a surprise since alignment is more complex for longer sequences. Nevertheless, Table 6.1, shows very promising results. In most cases, the prediction *sounds* similar to the ground truth, suggesting that the model makes good acoustic predictions. For example, the model predicts "kípur" instead of "kýpur" and "endir singingu" instead of "endursýningu". In both of those cases, the predictions have a very similar pronunciation in Icelandic.

Another interesting observation shown in Table 6.1 is the prediction of the utterance for the URL "steindi.blogspot.com". The model emits "punkt" instead of "." but the full stop symbol is called "punktur" in Icelandic. There are many URLs in the dataset and in most cases, the model has learned to emit the URL. For example, "bb.is" is predicted as "bb.is" and "ordabok.is" as "ordabok.is". The reason for why the model emits "punkt" instead of "." in the example above is not clear, but the model may have just learned to transcribe "punktur is", the pronunciation of ".is", as the single unit corresponding with ".is".

6.2 Language Model

Since the language model is not dependant on any other model parts for making predictions, we choose to begin by training the language model. At every step, the training loss is recorded. Periodically, the LM generates a 100 character long string, starting from the <sos> token. We can inspect these generated strings to get a basic idea about the performance of the LM. Since the LM will be used during ASR decoding, the language model is capable of predicting the same characters as the ASR. Some examples of generated strings are

Step	Generated string
0	"mnríýtcrd<yej4y88?ký,luöí?efðrön1sc,,mkþvicívge<ðð$a,ýb?47mdóý1k2rf8n9mntvék9þ23,sn?8cvníoþru.v.t3x"
600	"ohlatineinni bætti ryng sem sér leini herkar þr sinnar eigart neugastondar til aðkópa fygggjanni va"
4,800	". við sjálfsmenn og stað. tali bent að hann hafi verið verið það síðast í skýrt við að aftur en að"
27,400	"rafluknar olíða gömul hliðsson, bakteríur úr öðrum sem getur ekki beits segir þrí þó að mestur um sum"
70,000	"amma hennar norður hafi verið að keyra rafeindum þess að sjálfum ekki yfir þau sem verður með að vera"

Table 6.2: Strings generated by a language model during training

shown in Table 6.2. Although not all readers are familiar with the Icelandic language, it should be clear that with more training, the LM becomes better at generating eligible Icelandic. The LM was trained for 5 epochs, which equated approximately 70,000 iterations with a batch size of 128 where each sample is a 200 character long string. That equals about 17.9×10^8 character predictions.

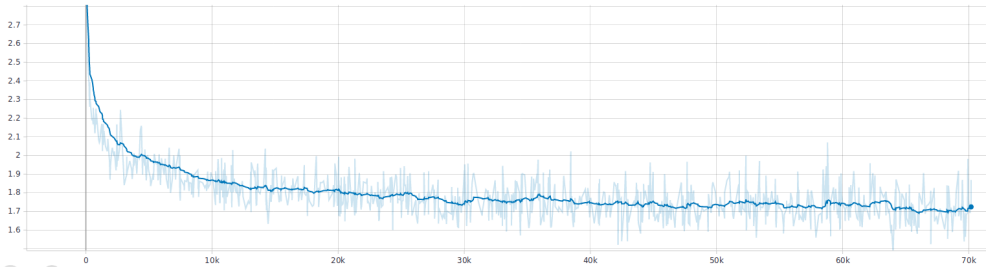


Figure 6.4: The learning process of the RNN LM

As discussed earlier, during training, the LM receives the previous prediction with a certain probability, instead of the ground truth to determine the current prediction. This is done to make the LM more generalized, and more capable of correcting bad previous predictions.

The total loss of each batch with labels $\mathbf{Y} = \{y_1, \ldots, y_n\}$ and predictions $\hat{\mathbf{Y}} = \{\hat{y}_1, \ldots \hat{y}_n\}$, is the average cross entropy of samples in the batch

$$\mathcal{L}_B(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1\ldots n} \mathcal{L}(y_i, \hat{y}_i) \tag{6.3}$$

where we have used the cross entropy loss function defined in Equation 4.4 In Table 6.3, LM predictions for the same string are compared for different

ρ	Prediction	Accuracy
0.0	" nsirtdistíanð araðleik et lit a sr aknn gruilun e sðym s tmn"	12.6%
0.1	" sarlanan þiveðaldafi s0ðmennaóka earsa oal r seimmtileein rg"	6.2%
0.2	" sllrdir eónn a sei eeed dega verðð aerii ir semtt smíaran"	10.8%
0.3	" akransg iað hajú áardurigur se þ ma i esearna óttirikui hn nn"	20.0%
0.4	" araimis oa hehir misgmður haedsess aeg heörtarir tari rsakuang"	21.5%
0.5	" akramíl ureiui verið aktsua beaðsettssið sarrtu eey rlarlann na"	43.1%
0.6	" nar elaioetjt eiðð dám iseátðtaáetmæsg ifretkssemie ena sikuarg"	9.2%
0.7	" m vófbl ssaiur verið akm emðaað heðiaá sysaörnunarsveitar enn þ"	41.5%
0.8	" t eðtnd oefur verið akmdi sil aats iðearargunarsveitar erdkng"	56.9%
0.9	" akramíl oafur verið akmdi asúflaats við srörgunarsveitar ann na"	64.6%
1.0	" akramíl oafur verið akmdi sil aats við þrörgunarsveitar ann na"	70.8%

Table 6.3: The LM tries to predict "[s]júkrabíll hefur verið sendur til móts við björgunarsveitarmennina" with different teacher forcing rates (ρ)

rates of using the ground truth to determine the current prediction. We can see that if we always sample from the previous prediction we get a very low character level accuracy. High teacher forcing rates result in high character

level accuracies. If we were to use the LM for ASR decoding during inference, we would always sample from the previous prediction since we do not know the ground truth beforehand. This LM is much better fitted to rescore the predictions generated by the baseline ASR model.

6.3 Unsupervised Training of Auxiliary Components

Here we will describe the training results of the text autoencoder, speech autoencoder as well as results of adversarial training. As described earlier, we start by training the text autoencoder. Following that, adversarial training takes place, using the now pre-trained ASR encoder as a generator. Finally, we train the speech autoencoder, again using the pre-trained ASR encoder. We store the intermittent version of the baseline ASR before each phase, resulting in 4 different pre-trained models. We will from now on use the following naming convention.

- M_1: The baseline ASR model described in Chapter 4 without any unsupervised pre-training.
- M_2: Baseline ASR with text autoencoder training.
- M_3: Baseline ASR with text autoencoder training and adversarial training
- M_4: Baseline ASR with text and speech autoencoder training and adversarial training.

6.3.1 Text Autoencoder

The text autoencoder was trained on a mixture of the Malromur corpus and the Risamalheild corpus. Using a batch size of 128 samples, the text autoencoder was trained for 30 epochs, or just about $175,000$ iterations, resulting in about $11,2$ million strings being encoded and decoded.

We train the text autoencoder to minimize the character level cross entropy. For a set of string predictions $\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_n\}$ and corresponding ground truth labels $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, the batch level loss is given as

$$\mathcal{L}_B(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{n \times s} \sum_{i=1 \ldots n} \sum_{j=1 \ldots s} \mathcal{L}(\hat{\mathbf{y}}_i[j], \mathbf{y}_i[j]) \tag{6.4}$$

where have used the cross entropy loss defined in Equation 4.4 and s is the length of each string in the batch. The validation and training loss curves of

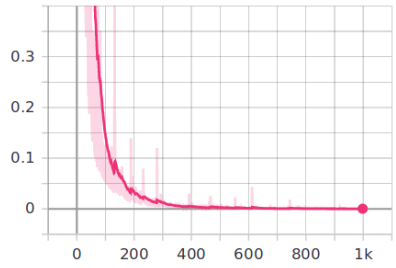
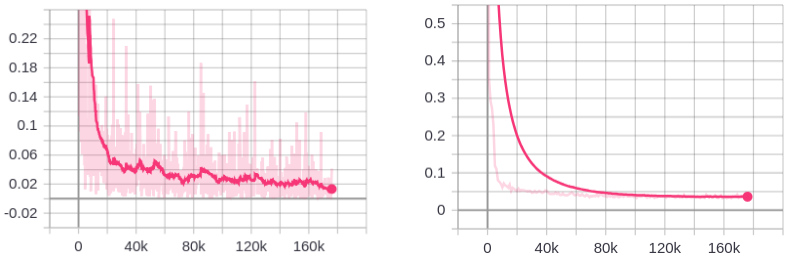


Figure 6.5: The text autoencoder fitted to a single sample

the text autoencoder are shown in Figure 6.6. Interestingly, the loss of the text autoencoder seems to bounce back and forth, which was also observed when it was made to overfit to a single sample as shown in Figure 6.5. This might suggest a too high learning rate, but could be caused by other factors. Regardless of this, the text autoencoder converges to a validation loss of about 0.03 and a training loss of approximately 0.01. Based on decoding results, it



Train loss Validation loss

Figure 6.6: Loss curves for text autoencoder training

seems like the autoencoder is biased towards the character "s". That is, the ASR decoder will often append the "s" character to predictions. For example, decoding "snanna" from "nanna" or decoding "seinar" from "einar". It is hard to explain what exactly causes this and hard to correct for. Regardless of this, it should be clear that the text autoencoder is very capable. Lastly, we take a look at the performance of the text autoencoder, comparing different text drop rates. The results are shown in Figure 6.4. As we can see, the text autoencoder performs quite well on this never before seen phrase. The autoencoder practically manages to recreate the original string up until a drop rate of 0.3.

ξ	Dropped text	Prediction
0.0	"þetta tauganet er að reyna að skilja texta"	"þetta tauganet er að reyna að skilja texta"
0.1	"þetta auganet r að reyna ð skilja ta"	"þetta tauganet er að reyna að skilja texta"
0.2	"þetta tauaet r að reynað skija texta"	"þetta tauga et er að reyna að skilja texta"
0.3	"þettga r aðena skljaexta"	"þetta tauganet er aa reyna að skilja t"
0.4	"þtatunet era rna að ja"	"þetta anet er að reyna að ilja"

Table 6.4: Decoding results of the original string "þetta tauganet er að reyna að skilja texta" shown for different drop rates (ξ)

6.3.2 Adversarial Training

The discriminator and the generator, the ASR encoder, are trained together. Given a discriminator training batch of real and synthetic hidden vector predictions and the corresponding labels, $\mathcal{B} = \{(\hat{y}_1^r, 1-a), \ldots, (\hat{y}_n^r, 1-a), (\hat{y}_1^f, 0), \ldots, (\hat{y}_m^f, 0)\}$, the total discriminator loss is given as

$$\mathcal{L}_B^G(\mathcal{B}) = \frac{1}{n} \sum_{i=1\ldots n} \text{BCE}(\hat{y}_i^r, 1-a) + \frac{1}{m} \sum_{i=1\ldots m} \text{BCE}(\hat{y}_i^f, 0) \tag{6.5}$$

where $\hat{y}_i^r$ and $\hat{y}_i^f$ are real and synthetic hidden vector predictions respectively. Similarly, the loss minimized by the generator for a batch of *discriminator* predictions and corresponding labels, $\mathcal{B} = \{(\hat{y}_1^f), \ldots, (\hat{y}_k^f)\}$, is given as

$$\mathcal{L}_B^D(\mathcal{B}) = \frac{1}{k} \sum_{i=1\ldots k} \text{BCE}(\hat{y}_i^f, 1) \tag{6.6}$$

As was mentioned in Section 4.4, we use label smoothing for adversarial training, and the discriminator is trained to minimize the binary cross entropy loss of predictions and labels. Because of this, the lowest loss of a real data label $1-a$, where $a \in (0,1)$ is the smoothing value, is when the discriminator predicts exactly $1-a$. In that case, the loss value is, achieved using Equation 4.31

$$\begin{aligned} \mathcal{L}(1-a, 1-a) &= (1-a)\log(1-a) \\ &\quad - (1-(1-a))\log(1-(1-a)) \\ &= -(1-a)\log(1-a) - (a)\log(a) \end{aligned} \tag{6.7}$$

For $a = 1/10$, the minimum loss comes out as approximately 0.32. This is in fact the loss value that the discriminator quickly converges to. There is no theoretical maximum BCE loss value, since for example, $\hat{y} = 0$ and $y = 1$, the loss tends to infinity. This fact is visualized in Figure 6.7 where we have plotted

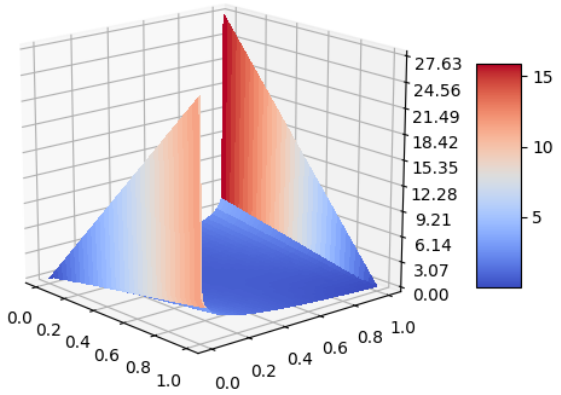
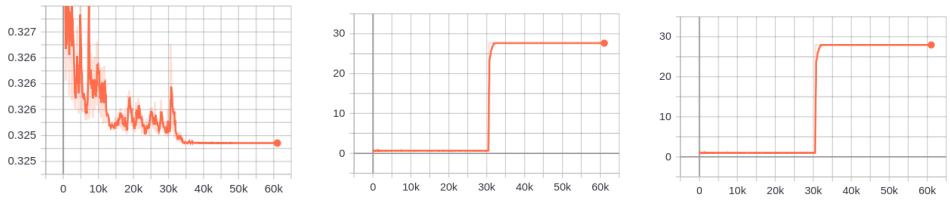


Figure 6.7: BCE loss as a function of $\hat{y}$ and y.

the BCE loss for values of $(\hat{y}, y)$ in the $(0; 1)$ range. By experimentation, we find that the BCE implementation in PyTorch numerically caps the maximum loss at approximately 27.6.

The amount of synthetic data that the discriminator is trained on depends on the length of each utterance and therefore, the batch sizes of these two modules cannot be considered the same. For example, an utterance can be several thousand frames long and the ASR encoder would then emit several hundred hidden vectors if we account for the down-sampling performed by the encoder. At the same time, the corresponding text label, which is the input to the text encoder data distribution, could be less than 50 characters long, resulting in less than 50 hidden vectors. However, in terms of the number of training utterances, the modules are trained on an equal amount of data. To start with,

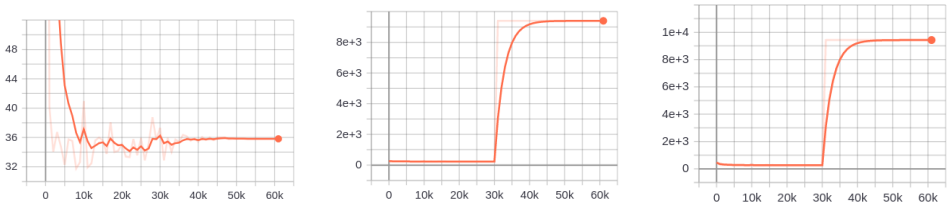


Real Loss Fake Loss Total Loss

Figure 6.8: Loss curves for discriminator training

training seemed to be going quite well. Based on Figure 6.8, the discriminator reaches the lowest achievable training loss for real samples. For about 30,000 training steps, the generator did not manage to fool the discriminator. Then, however, something changed and the discriminator starting predicting

all generated samples as real as the loss of fake samples shows in Figure 6.8. The same behavior is visible for validation samples as shown in Figure 6.9. And this, of course, is mirrored for the generator, which reaches the minimum training loss as shown in Figure 6.10 The goal of adversarial training in this



Real Loss Synthetic Loss Total Loss

Figure 6.9: Loss curves for discriminator validation (Note: losses not averaged over validation samples)

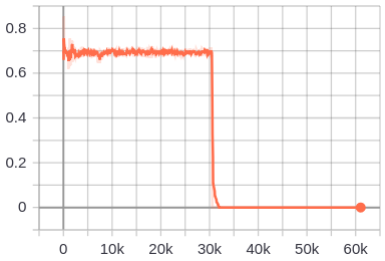


Figure 6.10: Generator training loss

context is to encourage the ASR encoder to share an embedding space with the text autoencoder. Based on the figures above, one could be fooled to think that that goal has been achieved but it turns out that the generator learns to output the same hidden vector for all different inputs. This is likely the result of *mode collapse*, the issue of the generator learning to emit the same output for any input [75]. We can visualize this using principal component analysis (PCA) [77]. In Figures 6.11 and 6.12 we show a 3-component PCA visualization of the generated hidden vectors after no training and 50 steps of training. After only 50 steps, the synthetic hidden vectors, shown in red, have become very similar. Before any training, both types of hidden vectors consist of small values, seemingly distributed and centered around the 0 vector. After only 50 steps of training we clearly observe mode collapse.

The issue of mode collapse is a very common point of failure when training GANs and has been described in detail [78]. At some point during training,

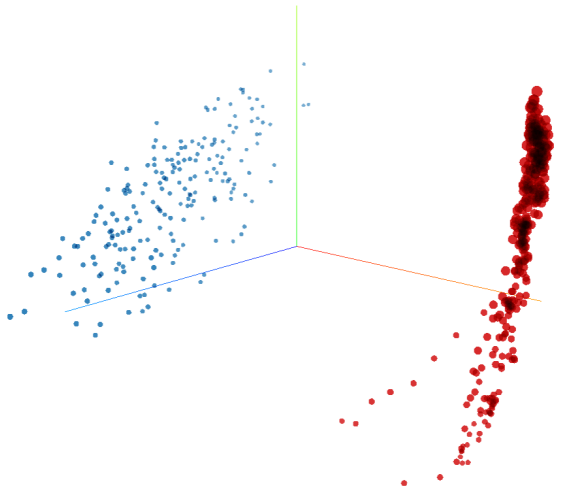


Figure 6.11: PCA visualization before any training (real hidden vectors in blue, synthetic hidden vectors in red)

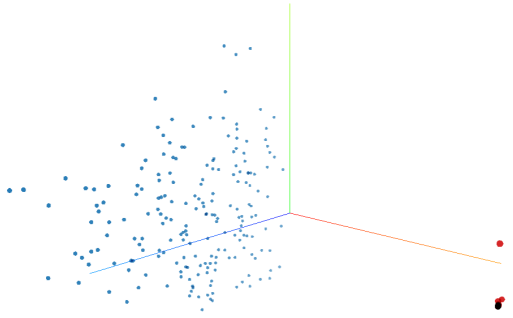
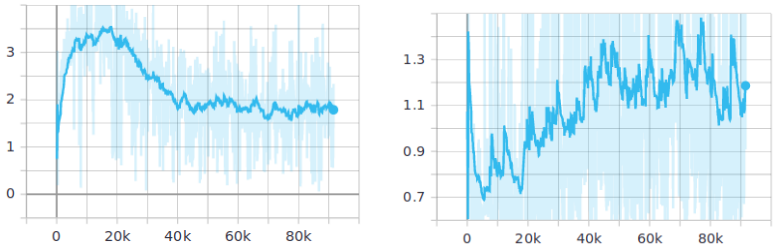


Figure 6.12: PCA visualization after 50 training steps (real hidden vectors in blue, synthetic hidden vectors in red)

the generator identifies some mode in the output distribution that the discriminator highly believes is sampled from the ground truth distribution. The generator then places all of it's output probability distribution mass on that mode. Ultimately, the discriminator learns that samples around that mode are likely synthetic and learns to classify them as such. Instead of the generator learning to produce samples distributed around many different modes, the generator simply moves all the mass to some other mode that the discriminator currently believes represents real samples. This procedure then repeats, resulting in non-convergence. In light of this, experiments were made with different activation functions, model topologies, and optimizers. The same issue persist

which seems to suggest that the issue is caused by mode collapse. One of the configurations lead to more sensible results and did not display the same aggressive learning as before. In this configuration, the discriminator model was expanded a bit. For activations, we use leaky ReLU instead of regular ReLU and we add one more hidden layer. As explained earlier, there is not a balance in the amount of synthetically generated data and data sampled from the ground truth distribution. For this new experiment, we add more real data by leveraging the language model data we have available. Using this new adver-



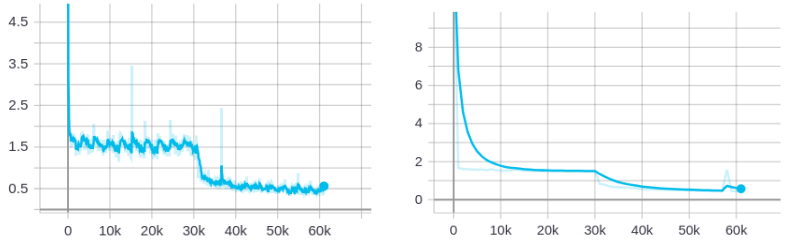
Generator training loss Discriminator training loss

Figure 6.13: Adversarial training curves for M_6

sarial configuration, a new pre-trained ASR is proposed and we refer to that model as M_6. This model was first pre-trained using the text autoencoder, then the speech autoencoder and finally adversarial training was performed.

6.3.3 Speech Autoencoder

The final auxiliary component to be trained is the speech autoencoder. And since the speech autoencoder shares parameters with the ASR encoder, the training of the speech encoder will inadvertently be influenced by the adversarial training that precedes it. This fact is visible in Figure 6.14 which shows a clear drop in the loss for both training and validation after about 30 thousand global steps. The speech autoencoder performs quite well, reaching training loss of approximately 0.38 and validation loss of 0.44. As explained in Section 4.2, the speech autoencoder is trained to minimize the smooth L1 loss. The spectrogram, S, is stored as a two dimensional array where $S[i, j]$ is a scalar, representing the amplitude of the j-th frequency component of the i-th frame. The loss is therefore averaged over all frames and frequency components of the whole batch. That is, the total loss of the batch labels $\mathbf{Y} = \{S_1, \dots S_n\}$



Train Loss Evaluation Loss

Figure 6.14: Speech Autoencoder training and validation loss curves

with predictions $\hat{\mathbf{Y}} = \{\hat{S}_1, \ldots \hat{S}_n\}$, is given as

$$\mathcal{L}_B(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n \times t_b \times f} \sum_{k=1\ldots n} \sum_{i=1\ldots t_b} \sum_{j=1\ldots f} \mathcal{L}(\mathbf{S}_k[i,j], \hat{\mathbf{S}}_k[i,j]) \tag{6.8}$$

where we have used the element-wise smooth L1 loss defined in Equation 4.21, t_b is the maximum number of frames in the batch and f is the number of frequency bands. The minimum loss achievable is therefore zero, while the maximum loss depends on the spectrogram input.

In Figure 6.15 we show the input to the speech autoencoder, a spectrogram for one of the validation utterances. In Figure 6.16 we show the outputs of the speech autoencoder for this particular input at certain steps during training. As we can see, after about 32,000 steps, the speech autoencoder output has greatly improved. As it is the goal of the ASR encoder to capture linguistic features of the signal, i.e. features that change over time as opposed to utterance level features, we theorize that it is the ASR encoder that makes these improvements around the 32,000th step.

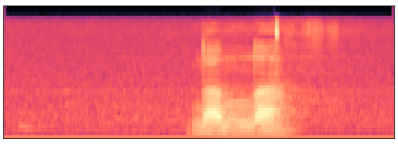


Figure 6.15: An example of a ground truth spectrogram

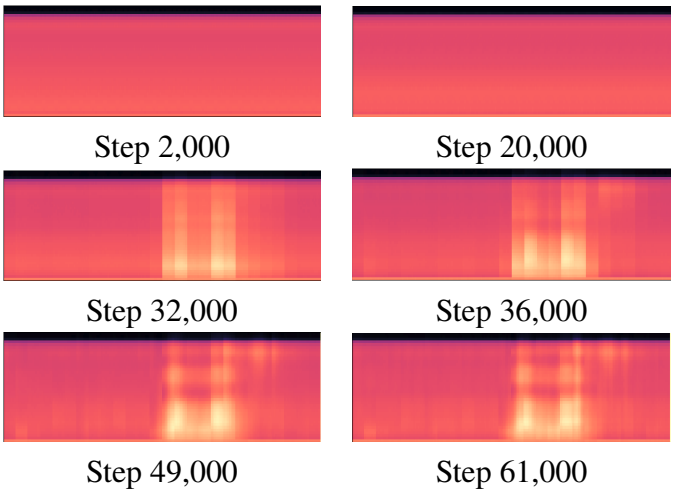


Figure 6.16: Speech Autoencoder outputs at different steps during training

After discovering the issues with adversarial training, an additional model without adversarial training was created. This model, which we will refer to as M_5, represents the baseline ASR with both text and speech autoencoder training. As can be seen in Figure 6.17, the speech autoencoder converges

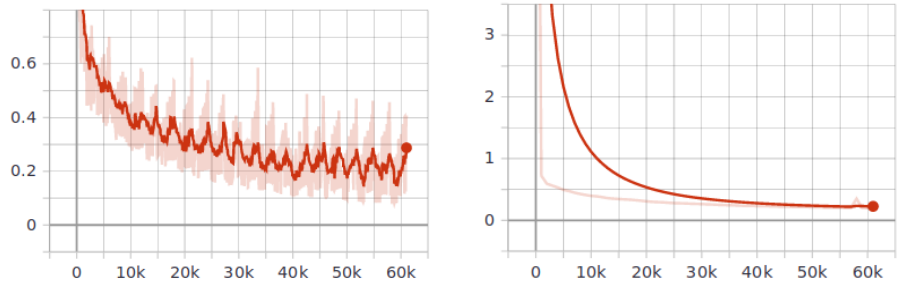
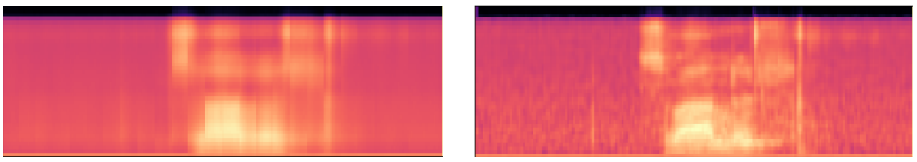


Figure 6.17: Speech Autoencoder learning curves, without prior adversarial training

faster and does not get stuck in local optima when prior adversarial training is omitted. In this case, the speech autoencoder reaches training loss of 0.18 and validation loss of 0.22 after approximately 60,000 steps, compared to 0.64 and 0.45 respectively of the prior speech autoencoder, that of the M_4 model. Finally, in Figure 6.18, we show the difference in outputs made by these two speech autoencoders.



M_4 M_5

Figure 6.18: Comparing the outputs of the speech autoencoders of models M_4 and M_5 for the same validation sample. Original sample is the one shown in Figure 6.15

6.4 Supervised Training of Baseline ASR

In this section, we compare the training results using all different types of pre-trained models under three different resource conditions. The 6 different ASR models are summarized in Table 6.5. We start by comparing the first four models which were originally proposed.

It should be noted that in this section, inference is performed without a language model. Additionally, inference results are determined by choosing the character with the highest probability at each time step without length normalization like in Equation 3.22 or beam search.

	TAE	ADV	SAE	ADV-2
M_1	-	-	-	-
M_2	1	-	-	-
M_3	1	2	3	-
M_4	1	2	3	-
M_5	1	-	2	-
M_6	1	-	2	3

Table 6.5: The order of auxiliary component training for each model type. (TAE = text autoencoder, SAE = speech autoencoder, ADV = adversarial training , ADV-2 = alternative adversarial training proposed in Section 6.3.2

We start with the first four models trained on 2.5 hours of transcribed speech. As we can see in Figures 6.19 and 6.20, the only model that beats the baseline (M_1) is the model pre-trained using the text autoencoder, M_2. Even though the validation loss metric is questionable, as explained previously, the training loss of M_1 and M_2 in Figure 6.19 suggests that the models become over-trained after about 10,000 steps, which corresponds with the losses shown in Figure 6.20. It is clear that in terms of each metric on both training and validation

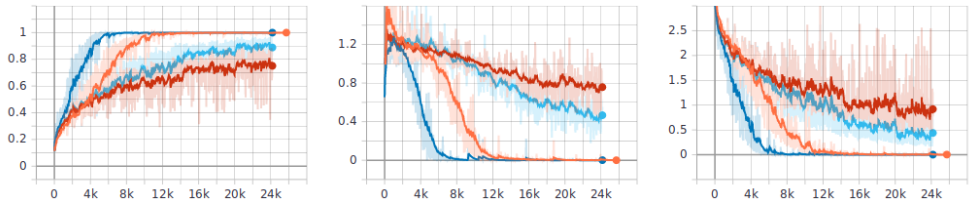


Figure 6.19: Training accuracy, error and loss for all model types trained on 2.5 hours of transcribed data. (M_1: orange, M_2: dark blue, M_3: dark red, M_4: light blue)

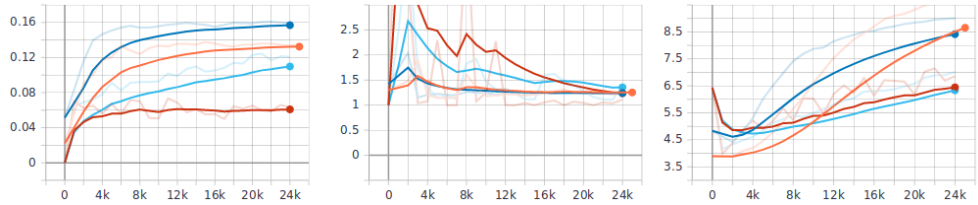


Figure 6.20: Validation accuracy, error and loss for all model types trained on 2.5 hours of transcribed data. (M_1: orange, M_2: dark blue, M_3: dark red, M_4: light blue)

sets, M_2 achieves the best results, reaching validation accuracy of 0.16. To put that in perspective, a randomly generated prediction for a string sequence of length n reaches accuracy of $\frac{1}{50^n}$ on average where 50 is the number of text tokens used in this work. For $n = 10$, the average accuracy of randomly generated sequences is basically zero, or approximately 1.02×10^{-17}. The average length of samples in the training data is much higher than 10 characters meaning that the results in Table 6.6 suggest that all models are much better than simply guessing. In Table 6.7, we show examples of predictions generated

	Accuracy	Error
M_1	0.13	1.27
M_2	0.16	1.25
M_3	0.06	1.42
M_4	0.11	1.39

Table 6.6: Validation accuracy and error for 2.5 hours of training data

by these models using only 2.5 hours of transcribed speech after a different amount of training. Surprisingly, even though M_1 and M_2 are far from correctly recognizing the example words in Table 6.7, they seem to be becoming

better as training progresses. For comparison, M_3 and M_4 make very limited progress. The error rate of both M_1 and M_2 are still above 1 suggesting that both are still emitting too long predictions. The words in bold in Table 6.7 are the best predictions in terms of character level accuracy. Next, we compare the

	$y =$"haukamýri"	$y =$"hægindi"	$y =$"uppsölum"
$M_1(\mathbf{x})$	*"hann hann haf stöðum í" "lungnað verður kómk"* ***"marglið"***	*"áeir" "heimi"* ***"helli"***	*"skarði" "lustúljur" "fustvöluum"*
$M_2(\mathbf{x})$	*"fram var ein að" "mennirgarðarhefur"* ***"menni gr"***	*"fenni" "heimi"* ***"helli"***	*"fimmtudagsiðum" "vingsi"* ***"liðsstöðum"***
$M_3(\mathbf{x})$	*"sellið er stað stið að skipa" "steinarnir rekur sama stað" "steinarnir rekur sama stað"*	*"sellið er stað stið að skipa" "hrafnsteinur" "steinholti"*	*"setein heldur í hann stöðum" "steinarnir reka sama" "steinholti"*
$M_4(\mathbf{x})$	*"steinn er stað skipa skipa" "mynda byggðar gerði" "ljósvegjumenn"*	*"steinn er stað stöðum" "háeiri" "harryirgili"*	*"steinn er stað stöðum" "við stórum við komið" "leitaðiar"*

Table 6.7: Validation predictions at training steps 1,000, 10,000 and 20,000, using 2.5 hours of transcribed speech

results of training these four models on 5 hours of transcribed speech. Based on Figures 6.21 and 6.22, M_2 seems to again be the best model. The validation error if M_1 and M_2 is drastically improved as shown in Table 6.8 while M_3 and M_4 show no measurable improvement.

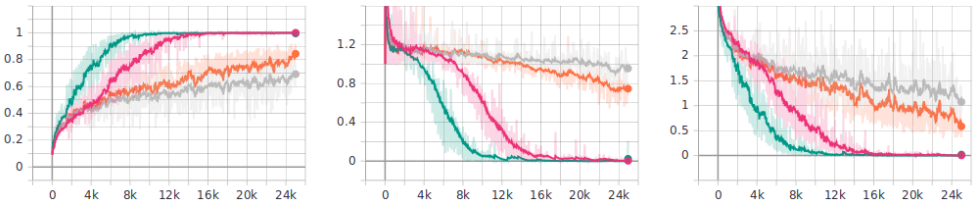


Figure 6.21: Training accuracy, error and loss for all model types trained on 5 hours of transcribed data. (M_1: pink, M_2: green, M_3: gray, M_4: orange)

	Accuracy	Error
M_1	0.19	1.35
M_2	0.23	1.18
M_3	0.06	1.37
M_4	0.11	1.42

Table 6.8: Validation accuracy and error for 5 hours of training data

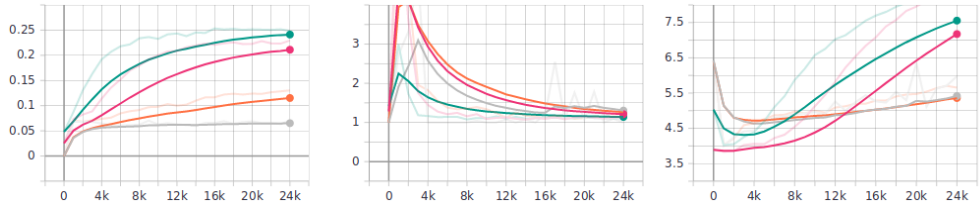


Figure 6.22: Validation accuracy, error and loss for all model types trained on 5 hours of transcribed data. (M_1: pink, M_2: green, M_3: gray, M_4: orange)

	$y =$"haukamýri"	$y =$"hægindi"	$y =$"uppsölum"
$M_1(\mathbf{x})$	*"hann herði herði herði" "herganri" "**hergarmiður**"*	*"hanna" & "heiði" & "**heiði**"*	*"hannar er" & "hlufsumum" & "hékkustöðum"*
$M_2(\mathbf{x})$	*"hann er að hann verði" & "herri kaun henn" & "hergnei"*	*"hrauni" & "hauði" & "**haríði**"*	*"hraunsnesi" & "gufustöðum" & "**hauskistöðum**"*
$M_3(\mathbf{x})$	*"hann hefti sér stjórnar" & "hlíð" & "hlíð"*	*"hann hefur er að" & "hlíð" & "hlíð"*	*"hefur er að stjórnar" & "hlíðar" & "hlíð"*
$M_4(\mathbf{x})$	*"hann herð st freinn hefti sann herði" & "melgerði" & "laufallilu"*	*"hraunsteiin" & "hergill" & "harii"*	*"hann herð st freinn" & "hestins stirlin" & "lyngsi"*

Table 6.9: Validation predictions at training steps 1,000, 10,000 and 20,000, using 5 hours of transcribed speech

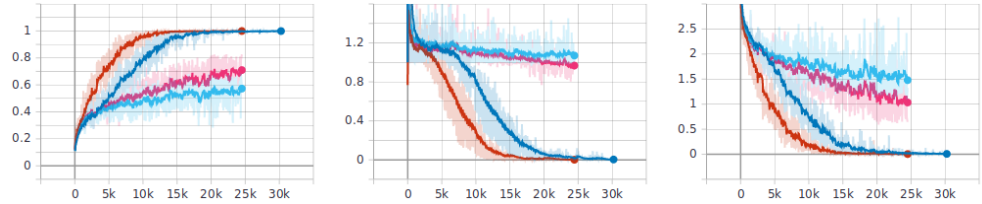


Figure 6.23: Training accuracy, error and loss for all model types trained on 10 hours of transcribed data. (M_1: dark blue, M_2: dark red, M_3: light blue, M_4: pink)

	Accuracy	Error
M_1	0.29	1.11
M_2	0.34	1.04
M_3	0.07	1.30
M_4	0.13	1.27

Table 6.10: Validation accuracy and error for 10 hours of training data

Finally, we compare the results of the four model using 10 hours of tran-

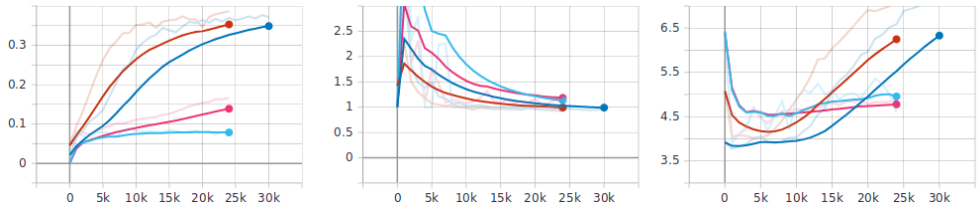


Figure 6.24: Validation accuracy, error and loss for all model types trained on 10 hours of transcribed data. (M_1: dark blue, M_2: dark red, M_3: light blue, M_4: pink)

scribed speech training data. Not surprisingly, M_2 ends up being the best model in this case, beating out M_1 once again. In Table 6.11 we see the first completely correct prediction, where M_2 correctly predicts "uppsölum" after training for 20,000 steps. After the same amount of training, M_1 predicts "unnsölum" which although close is less accurate. The attention weights of

	$y =$"haukamýri"	$y =$"hægindi"	$y =$"uppsölum"
$M_1(\mathbf{x})$	*"starnar" & "fett við heim" & "$aupablýmu"*	*"starnar er stöðum" & "helíð" & "hælingi"*	*"standar" & "urnsölum" & "unnsölum"*
$M_2(\mathbf{x})$	*"haga er er hann" & "höngurlíðu" & "**haukarlíður**"*	*"höfði" & "hægi" & "**hægingi**"*	*"sigurber" & "uppssölum" & "**uppsölum**"*
$M_3(\mathbf{x})$	*"hannar er stað stað stað" & "hafnargötu" & "hafni"*	*"hannar er stað stöðum" & "hafnargötu" & "hafnar"*	*"hannar er stað stað" & "haga" & "hafni"*
$M_4(\mathbf{x})$	*"hann steins á samastaði" & "verður í dag" & "menn"*	*"hannarstöðum" & "högn" & "hrænikeri"*	*"steinnist í stöðum" & "heilsubraut" & "miðsönum"*

Table 6.11: Validation predictions at training steps 1,000, 10,000 and 20,000, using 10 hours of transcribed speech

the four models trained on 10 hours of transcribed speech are shown at different steps of training in Figure 6.25. Interestingly, we can see that M_2 starts focusing it's attention much sooner than M_1. Both models end up with similar attention weights at step 24,000 however. Comparing M_4 to M_3 we see yet again that adversarial training has a negative impact on ASR performance and M_3 does not attend to the input at all. In Figure 6.26 we show that there is a clear correlation between the amount of training data and accuracy for M_1 and M_2. Both models consistently perform better when supplied with more data. Finally, we consider the last two models proposed, M_5 and M_6. Both of these models were trained under the same three resource conditions. It is clear from the results shown in Figure 6.27 that M_5 manages much better than M_6. However, compared to previous models pre-trained using adversarial training, M_6 performs much stronger. Inspecting the attention weights of M_6 trained

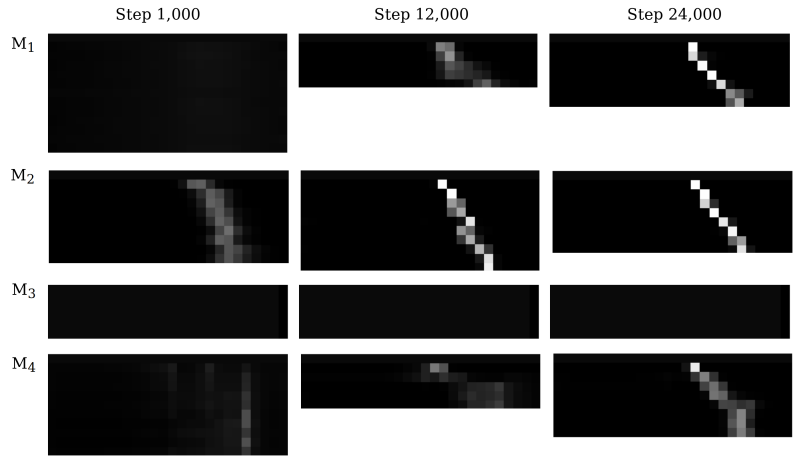
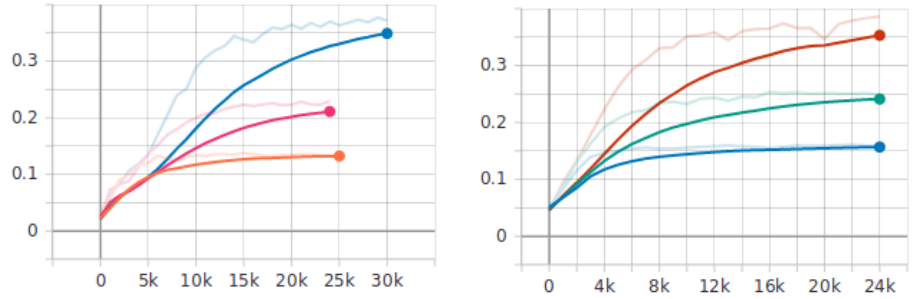


Figure 6.25: Attention weights when attending to the validation utterance of "íris"



M_1 M_2

Figure 6.26: Comparing validation accuracy across resource conditions for M_1 and M_2, showing that validation accuracy correlates well with the amount of training data (For both model types, best accuracy is reached with 10 hours of transcribed speech, then 5 hours and worst accuracy with 2.5 hours)

on 10 hours of transcribed speech, we can see that when compared to M_3, it manages to attend to the input much better. The attention weights of M_6 are shown in Figure 6.28. M_5 achieves slightly better results than M_2, reaching accuracy of 0.17 for 2.5 hours of data, 0.26 for 5 hours and 0.35 for 10 hours.

6.5 ASR Inference

Now that we have concluded that M_5 achieves the highest accuracy under all resource conditions, we will consider inference results using beam search and language model rescoring. Here, the hypothesis is chosen as in [2], following

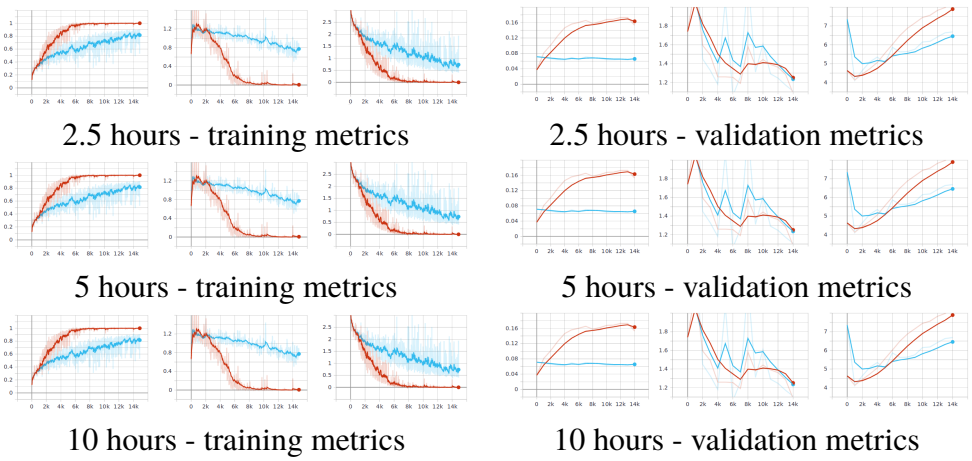


Figure 6.27: Training and validation accuracy, error and loss for M_5 and M_6. For each metric, M_5 is better than M_6

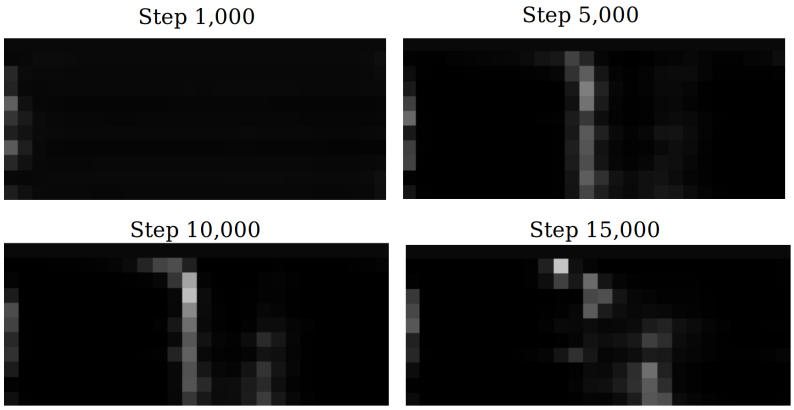


Figure 6.28: The attention weights of M_6 when attending to the validation utterance of "svanur"

Equation 3.22. The results are shown in Table 6.12 For this experiment a beam

Training data	Direct Inference	Beam search	Beam search w. LM
2.5 hours	0.17	0.20	0.23
5 hours	0.26	0.34	0.35
10 hours	0.35	0.41	0.40

Table 6.12: Test Accuracy of the best produced model

width of 8 was used which determines the amount of hypothesis we consider at each step of decoding. As we can see demonstrated in Table 6.12, beam search does in fact increase the accuracy of the model. Adding language model

rescoring has less of an effect but a word level LM is more likely to increase the accuracy further.

Chapter 7

Discussion

7.1 Conclusions and Future Work

Based on the results in Chapter 6, a hybrid semi-supervised training procedure offers some performance gains under all three resource conditions considered. We can therefore answer the research question stated in Chapter 1 positively. Despite this relative success, it was disheartening to see the best model proposed in [15] perform so poorly in this work. The role of adversarial training is said to be critical for obtaining the best model in [15]. Without adversarial training, the baseline ASR model is said to struggle with finding a correspondence between the speech input and the text output. However, a comparison of the performance of the discriminator and the generator is not listed in [15] which makes it difficult to compare adversarial training results in this work to that in [15].

The concrete source of this performance discrepancy has not yet been established but is most likely the result of mode collapse. This is partially based on experiments performed after the results in Chapter 6 were obtained. Adversarial training was tested on the same model topology, but with much smaller hidden representations. In this case, the ASR encoder was changed to emit 3-dimensional hidden vectors and the text encoder was changed in the same way. This is a drastic dimensionality reduction since the model proposed emitted 512-dimensional hidden vectors. Regardless of this drastic change, the generator never manages to catch up with the discriminator, as shown in Figure 7.1. This could suggest that a higher learning rate for the generator could be beneficial. To deal with the likely issue of mode collapse, *minibatch discrimination* is proposed [78]. The key to minibatch discrimination is to append information about the *closeness* of samples in the minibatch to the input of the

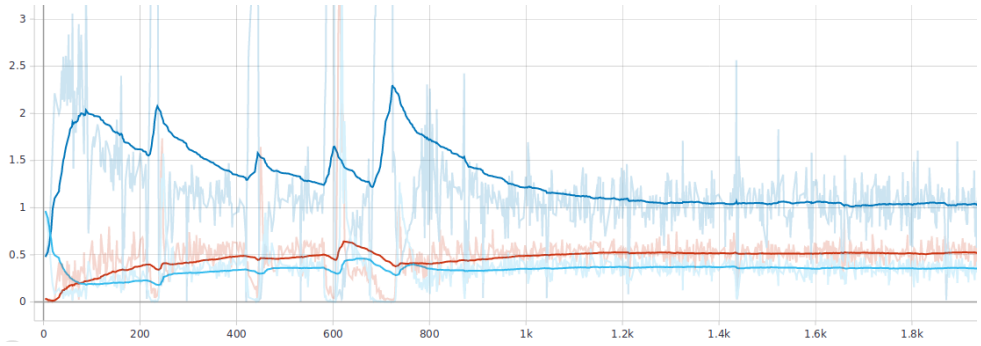


Figure 7.1: The learning results of adversarial training using a much simpler model. Generator loss shown in dark blue, discriminator loss of synthetic samples in light blue and real samples in red

discriminator. In this way, the discriminator can detect if a sample is unusually close to other generated samples in latent space. This weakens the generator race towards a single mode. Implementing minibatch discrimination, feature matching [78] or other means of reducing mode collapse is therefore a high priority future work.

Transfer learning, the act of leveraging data in one domain to learn relationships in a different related domain, is another method that has been used in ASR development to deal with the low resource issue, two of which are compared in [79]. The first one is *Heterogeneous Transfer Learning*, in which a single base model is simultaneously trained on multiple languages. The second one is *Model Adaptation*, where a model is first trained on a language under good resource conditions and then re-trained on the low resource language using the pre-learned weights as a starting point. This second method has shown promising results, such as in [80], where a multi-layer perceptron was first trained on English and other resourceful languages and then successfully adapted to low resource languages like Vietnamese.

A common source of error in DNN ASR models is extremely rare or out-of-vocabulary words. The main approach to address this issue is to incorporate a language model during decoding as explained in Section 2.3.3. More complex approaches have been proposed to address the issue of rare words. One of those is to create a synthetic speech and text corpora using a TTS and use it with a spelling correction model to directly deal with the error distribution of the baseline ASR model [81]. This approach has shown good results but is also much more computationally complex.

The language model used in this work was a character level RNN language model. Conventionally, language models predict sequences of words, rather

than sequences of characters, and estimate n-gram probabilities as mentioned in Section 2.3.3. In general, word level LMs outperform character level LMs but are harder to train. It is therefore of interest to research ways of incorporating a word level LM into the decoding procedure of the proposed model. In [82] and [15] the *Weighted Finite State Transducer* (WFST) framework is used to convert a word level language model into a character level one. In [83], a neural LM is proposed that only relies on character inputs, but predictions are made on the word level.

As mentioned in 7.1, the adversarial training procedure proposed is likely faulty as a result of some technical aspects. To achieve the results reported in [15], a correct implementation of the adversarial training procedure is needed. Replication work such as this one in the field of machine learning remains complicated for several reasons but mainly because of resources and unclear model formulation. This work is heavily inspired by the work in [15] and [2]. The first one does not describe in detail how training was performed, specifically how long each model component was trained and how adversarial training was configured. The formulation of model components, specifically the speech encoder and the discriminator, is additionally so short that it allows for multiple interpretations.

7.2 Contributions

In this work we have compared up to 6 semi-supervised ASR models for the Icelandic language under 3 different resource conditions. These semi-supervised ASR models are inspired by the work in [2], [15]. Design decisions have been made in this work that are not directly inspired by that work, most notably

- The down-sampling procedure of the ASR encoder of the baseline model
- The adversarial training setup
- The CNN speech encoder design

Additionally, the results reported here are much more detailed than compared to [15] allowing for a more in-depth comparison of these different models and the baseline.

All components, including the baseline model, the language model, the speech and text autoencoders and the discriminator were created from scratch in PyTorch as well as the procedures that facilitate both supervised and unsupervised training.

7.3 Final Words

Although the results obtained in Chapter 6 are somewhat disheartening, the importance of the subject remains high. The goal of reducing the resource needs of the state-of-the-art DNN ASR of today is a noble one and the research of this topic is the foremost course to achieve that goal. Leveraging unlabeled data for unsupervised training in a hybrid setup as proposed is a step in the direction of the lofty idea of zero resource ASR generation.

Acronyms

ANN artificial neural network.

ASR automatic speech recognition.

BCE binary cross entropy.

BiRNN bidirectional recurrent neural network.

BLSTM bidirectional long-short term memory.

CNN convolutional neural network.

CTC connectionist temporal classification.

DFT discrete Fourier transformation.

DNN deep neural network.

FFT fast Fourier transform.

GAN generative adversarial network.

GMM Gaussian mixture model.

GMM-HMM hidden Markov model with Gaussian mixture emissions.

HMM hidden Markov model.

LAS listen, attend and spell.

LM language model.

LReLU leaky rectified linear unit.

LSTM long-short term memory.

MFCC Mel-frequency cepstral coefficient.

MLE maximum likelihood estimation.

NLP natural language processing.

NN neural network.

pBLSTM pyramid-shaped bidirectional long-short term memory.

PCA principal component analysis.

ReLU rectified linear unit.

RNN recurrent neural network.

SGD stochastic gradient descent.

STFT short-time Fourier transform.

WER word error rate.

Bibliography

[1] Wayne Xiong et al. “Achieving human parity in conversational speech recognition”. In: *arXiv preprint arXiv:1610.05256* (2016).

[2] William Chan et al. “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 4960–4964.

[3] András Kornai. “Digital language death”. In: *PloS one* 8.10 (2013), e77056.

[4] Eiríkur Rögnvaldsson and Sigríður Sigurjónsdottir. “Stafrænt sambýli íslensku og ensku”. In: *Ráðstefnurit Netlu–Menntakvika 2018* (Dec. 2018).

[5] Steinþór Steingrímsson et al. “Málrómur: A manually verified corpus of recorded Icelandic speech”. In: *Proceedings of the 21st Nordic Conference on Computational Linguistics*. 2017, pp. 237–240.

[6] Björn Guðfinnsson. “Mállýzkur II.” In: *Reykjavík: Ísafoldarprentsmiðja* (1946).

[7] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *arXiv preprint arXiv:1906.02243* (2019).

[8] KH Davis, R Biddulph, and Stephen Balashek. “Automatic recognition of spoken digits”. In: *The Journal of the Acoustical Society of America* 24.6 (1952), pp. 637–642.

[9] Biing-Hwang Juang and Lawrence R Rabiner. “Automatic speech recognition–a brief history of the technology development”. In: *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara* 1 (2005), p. 67.

[10] Lawrence R Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[11] Richard P Lippmann. “Review of neural networks for speech recognition”. In: *Neural computation* 1.1 (1989), pp. 1–38.

[12] Dan J Kershaw, Anthony J Robinson, and Mike Hochberg. “Context-dependent classes in a hybrid recurrent network-HMM speech recognition system”. In: *Advances in Neural Information Processing Systems*. 1996, pp. 750–756.

[13] Maarten Versteegh et al. “The zero resource speech challenge 2015”. In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[14] Yi-Chen Chen et al. “Towards unsupervised automatic speech recognition trained by unaligned speech and text only”. In: *arXiv preprint arXiv:1803.10952* (2018).

[15] Jennifer Drexler and James Glass. “Combining End-to-End and Adversarial Training for Low-Resource Speech Recognition”. In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2018, pp. 361–368.

[16] Eiríkur Rögnvaldsson. “Icelandic language technology: an overview”. In: *Language, Languages and New Technologies: ICT in the Service of Languages. Contributions to the Annual Conference*. 2010, pp. 187–195.

[17] Eiríkur Rögnvaldsson. “The Icelandic speech recognition project Hjal”. In: *Nordisk Sprogteknologi. Årbog* (2003), pp. 239–242.

[18] Jón Guðnason et al. “Almannaromur: An open icelandic speech corpus”. In: *Spoken Language Technologies for Under-Resourced Languages*. 2012.

[19] Jón Guðnason et al. “Building ASR Corpora Using Eyra.” In: *INTERSPEECH*. 2017, pp. 2173–2177.

[20] Inga Rún Helgadóttir et al. “Building an ASR Corpus Using Althingi’s Parliamentary Speeches.” In: *INTERSPEECH*. 2017, pp. 2163–2167.

[21] Mark Gales, Steve Young, et al. “The application of hidden Markov models in speech recognition”. In: *Foundations and Trends® in Signal Processing* 1.3 (2008), pp. 195–304.

[22] Louis CW Pols et al. “Spectral analysis and identification of Dutch vowels in monosyllabic words”. In: (1977).

[23] Douglas O’shaughnessy. *Speech communication: human and machine*. Universities press, 1987.

[24] Claude Elwood Shannon. “Communication in the presence of noise”. In: *Proceedings of the IEEE* 86.2 (1998), pp. 447–457.

[25] Alan V Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.

[26] Fredric J Harris. “On the use of windows for harmonic analysis with the discrete Fourier transform”. In: *Proceedings of the IEEE* 66.1 (1978), pp. 51–83.

[27] James W Cooley and John W Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[28] Stanley Smith Stevens, John Volkmann, and Edwin B Newman. “A scale for the measurement of the psychological magnitude pitch”. In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190.

[29] Gunnar Fant. “Analysis and synthesis of speech processes”. In: *Manual of phonetics* 2 (1968), pp. 173–277.

[30] Malcolm Slaney. “Auditory toolbox”. In: *Interval Research Corporation, Tech. Rep* 10.1998 (1998).

[31] Eberhard Zwicker. “Subdivision of the audible frequency range into critical bands (Frequenzgruppen)”. In: *The Journal of the Acoustical Society of America* 33.2 (1961), pp. 248–248.

[32] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. “Comparison of different implementations of MFCC”. In: *Journal of Computer science and Technology* 16.6 (2001), pp. 582–589.

[33] Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*. Vol. 247. Springer Science & Business Media, 2012.

[34] Ossama Abdel-Hamid et al. “Convolutional neural networks for speech recognition”. In: *IEEE/ACM Transactions on audio, speech, and language processing* 22.10 (2014), pp. 1533–1545.

[35] Andrew Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE transactions on Information Theory* 13.2 (1967), pp. 260–269.

[36] Slava Katz. “Estimation of probabilities from sparse data for the language model component of a speech recognizer”. In: *IEEE transactions on acoustics, speech, and signal processing* 35.3 (1987), pp. 400–401.

[37] Yoshua Bengio et al. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[38] Tomáš Mikolov et al. “Recurrent neural network based language model”. In: *Eleventh annual conference of the international speech communication association*. 2010.

[39] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[40] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[41] Helmer Strik and Catia Cucchiarini. “Modeling pronunciation variation for ASR: A survey of the literature”. In: *Speech Communication* 29.2-4 (1999), pp. 225–246.

[42] Mari Ostendorf et al. “Integration of diverse recognition methodologies through reevaluation of N-best sentence hypotheses”. In: *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*. 1991.

[43] Frederick Richardson, Mari Ostendorf, and Jan Robin Rohlicek. “Lattice-based search strategies for large vocabulary speech recognition”. In: *1995 International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. IEEE. 1995, pp. 576–579.

[44] Vladimir I Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Soviet physics doklady*. Vol. 10. 8. 1966, pp. 707–710.

[45] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[46] Sepp Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. In: *Diploma, Technische Universität München* 91.1 (1991).

[47] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. “Learning precise timing with LSTM recurrent networks”. In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

[48] Haşim Sak, Andrew Senior, and Françoise Beaufays. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”. In: *Fifteenth annual conference of the international speech communication association*. 2014.

[49] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “LSTM neural networks for language modeling”. In: *Thirteenth annual conference of the international speech communication association*. 2012.

[50] Alex Graves and Jürgen Schmidhuber. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *Neural networks : the official journal of the International Neural Network Society* 18 (July 2005), pp. 602–10. DOI: 10.1016/j.neunet.2005.06.042.

[51] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. “Hybrid speech recognition with deep bidirectional LSTM”. In: *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE. 2013, pp. 273–278.

[52] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[53] Alex Graves et al. “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 369–376.

[54] Alex Graves. “Sequence transduction with recurrent neural networks”. In: *arXiv preprint arXiv:1211.3711* (2012).

[55] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[56] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[57] Jan Koutnik et al. “A Clockwork RNN”. In: *International Conference on Machine Learning*. 2014, pp. 1863–1871.

[58] Rohit Prabhavalkar et al. “A Comparison of Sequence-to-Sequence Models for Speech Recognition.” In: *Interspeech*. 2017, pp. 939–943.

[59] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[60] Yang Li et al. “Area Attention”. In: 2019.

[61] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[62] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. “Exploiting similarities among languages for machine translation”. In: *arXiv preprint arXiv:1309.4168* (2013).

[63] Alexis Conneau et al. “Word translation without parallel data”. In: *arXiv preprint arXiv:1710.04087* (2017).

[64] Guillaume Lample et al. “Unsupervised machine translation using monolingual corpora only”. In: *arXiv preprint arXiv:1711.00043* (2017).

[65] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[66] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. “Learning representations by back-propagating errors”. In: *Cognitive modeling* 5.3 (1988), p. 1.

[67] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” In: vol. 12. Jan. 2010, pp. 257–269.

[68] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[69] Yann A LeCun et al. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[70] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[71] Samy Bengio et al. “Scheduled sampling for sequence prediction with recurrent neural networks”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1171–1179.

[72] Fu-Jie Huang Marc’Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. “Unsupervised learning of invariant feature hierarchies with applications to object recognition”. In: *Proc. Computer Vision and Pattern Recognition Conference (CVPR’07). IEEE Press*. Vol. 127. 2007.

[73] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[74] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[75] Ian Goodfellow. “NIPS 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016).

[76] Steinþór Steingrímsson et al. “Risamálheild: A Very Large Icelandic Text Corpus”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*. 2018.

[77] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[78] Tim Salimans et al. “Improved techniques for training gans”. In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[79] Dong Wang and Thomas Fang Zheng. “Transfer learning for speech and language processing”. In: *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE. 2015, pp. 1225–1237.

[80] Ngoc Thang Vu and Tanja Schultz. “Multilingual multilayer perceptron for rapid language adaptation between and across language families.” In: *Interspeech*. 2013, pp. 515–519.

[81] Jinxi Guo, Tara N Sainath, and Ron J Weiss. “A spelling correction model for end-to-end speech recognition”. In: *arXiv preprint arXiv:1902.07178* (2019).

[82] Dzmitry Bahdanau et al. “End-to-end attention-based large vocabulary speech recognition”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 4945–4949.

[83] Yoon Kim et al. “Character-aware neural language models”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[84] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. “Which training methods for GANs do actually Converge?” In: *arXiv preprint arXiv:1801.04406* (2018).

[85] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. “Discrete cosine transform”. In: *IEEE transactions on Computers* 100.1 (1974), pp. 90–93.

TRITA EECS-EX