Finite state models for recognition and validation of read prompts

Aku Rouhe

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.7.2017

Thesis supervisor:

Prof. Mikko Kurimo

Thesis advisor:

M.Sc Reima Karhila

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Aku Rouhe

Title: Finite state models for recognition and validation of read prompts

Date: 31.7.2017 Language: English Number of pages: 7+76

Department of Signal Processing and Acoustics

Professorship: Speech Recognition

Supervisor: Prof. Mikko Kurimo

Advisor: M.Sc Reima Karhila

An automatic speech recognition system has to combine acoustic and linguistic information. Therefore the search space spans multiple layers. Finite state models and weighted finite state transducers in particular can efficiently represent this search space by modeling each layer as a transducer and combining them using generic weighted finite state transducer algorithms. When recognising a text prompt being read aloud, the prompt gives a good estimate of what is going to be said. However human reading naturally produces some deviations from the text, called miscues. The purpose of this thesis is to create a system which accurately recognises recordings of reading. A miscue tolerant finite state language model is implemented and compared against two traditional approaches, an N-gram model and forced alignment.
The recognition result will ultimately be used to validate the recording as fit for further automatic processing in a spoken foreign language exam, which Project DigiTala is designing for the Finnish matriculation examination. The computerisation of the matriculation examination in Finland makes the use of such automatic tools possible.
This thesis first introduces the context for the task of recognising and validating reading. Then it explores three methodologies needed to solve the task: automatic speech recognition, finite state models, and the modeling of reading. Next it recounts the implementation of the miscue tolerant finite state language models and the two baseline methods. After that it describes experiments which show that the miscue tolerant finite state language models solve the task of this thesis significantly better than the baseline methods. Finally the thesis concludes with a discussion of the results and future work.

Keywords: automatic speech recognition, weighted finite state transducer, language modeling, reading miscues

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Aku Rouhe

Työn nimi: Äärelliset tilamallit lukupuheen tunnistamisessa ja tarkastamisessa

Päivämäärä: 31.7.2017 Kieli: Englanti Sivumäärä: 7+76

Akustiikan ja signaalin käsittelyn laitos

Professuuri: Puheentunnistus

Työn valvoja: Prof. Mikko Kurimo

Työn ohjaaja: DI Reima Karhila

Automaattinen puheentunnistusjärjestelmä yhdistää akustista ja kielellistä tietoa, joten sen hakuavaruus on monitasoinen. Tämän hakuavaruuden voi esittää tehokkaasti äärellisillä tilamalleilla. Erityisesti painotetut äärelliset tilamuuttajat voivat esittää jokaista hakuavaruuden tasoa ja nämä muuttajat voidaan yhdistää yleisillä muuttaja-algoritmeilla. Kun tunnistetaan ääneen lukemista syötteestä, syöte rajaa hakuavaruutta hyvin. Ihmiset kuitenkin poikkeavat tekstistä hieman. Kutsun näitä lukupoikkeamiksi, koska ne ovat luonnollinen osa taitavaakin lukemista, eivätkä siis suoranaisesti lukuvirheitä. Tämän diplomityön tavoite on luoda järjestelmä, joka tunnistaa lukupuheäänitteitä tarkasti. Tätä varten totetutetaan lukupoikkeamia sietävä äärellisen tilan kielimalli, jota verrataan kahteen perinteiseen menetelmään, N-gram malleihin ja pakotettuun kohdistukseen.
Lukupuheen tunnistustulosta käytetään, kun tarkastetaan sopiiko äänite seuraaviin automaattisiin käsittelyvaiheisiin puhutussa vieraan kielen kokeessa. DigiTala-projekti muotoilee puhuttua osiota vieraan kielen ylioppilaskokeisiin. Ylioppilaskokeiden sähköistäminen mahdollistaa tällaisten automaattisten menetelmien käytön. Kokeet sekä englanninkielisellä simuloidulla aineistolla että ruotsinkielisellä tosimaailman aineistolla osoittavat että lukupoikkeamia sietävä äärellisen tilan kielimalli ratkaisee diplomityön ongelmanasettelun. Vaikealla tosimaailman aineistolla saadaan 3.77 ± 0.47 prosentuaalinen sanavirhemäärä.

Avainsanat: automaattinen puheentunnistus, painotetut äärelliset tilamuuttajat, kielimallit, lukupoikkeamat

Acknowledgements

I thank anyone who skims through this thesis or at least looks at the pictures. I have enjoyed writing it and I hope it shows.

I have enjoyed working intensely on this narrow topic and the feeling of grokking it. I thank my supervisor, Prof. Mikko Kurimo, for this opportunity and for the privilege of his trust. This work has been funded by Svenska Folkskolans Vänner and the Finnish Matriculation Examination board. I thank the Aalto Science-IT project for providing cluster computation resources.

I have enjoyed being surrounded by frighteningly clever people. It has helped in learning perhaps the single most important lesson in Aalto: to stop viewing my capabilities as fixed. Most of the important stuff is difficult and should be treated with care, not the frustration of wanting to understand it instantly. Specifically I want to thank Peter Smit for patiently answering my Kaldi questions. Also quite specifically I wish to apologise to these surrounding people for the open-back headphone bleed.

I have enjoyed the company and the advice of my advisor Reima Karhila. I am amused by his sharp commentary and I appreciate his ability to keep the big picture in view.

Finally I wish to thank Doora, with whom all this time has felt meaningful and compelling.

Espoo, 30.7.2017 Aku Rouhe

Contents

1 Introduction

What is assessed becomes valued. What is valued gets taught. This is the definition of *washback* in education. Nowhere is it more apparent than in high-stakes examinations, where futures are decided. Washback has a dual nature. It is seen as driving students to disregard material that is not relevant to their success in the test. Yet it is also seen as a power tool for shaping education; it is the darling of policy makers.[1]

The story of this thesis began eleven years ago in the Ministry of Education and Culture. For long the consensus had been to shift more focus toward oral skills in foreign language education. What better way to accomplish this than implement an oral test as part of the revered matriculation examinations? The twist was that testing oral proficiency is not straight forward, especially so in the large scale standardised manner that the matriculation examination requires. There were a number of problems that had to be solved first. The hard truth is that listening is slow and human reviewer time is expensive. Simply arranging an oral test was deemed problematic: simultaneous tests were technologically impossible and testing students one-by-one would have meant the test could not have been the same for everyone. [2]

A decade later the matriculation examination is currently being computerised. *Project DigiTala* believes computerisation is the key to making oral testing possible on a large scale. A computerised test can be administered to everyone simultaneously. Speech processing technologies promise to help in test scoring. A pilot version of an oral test was created and administered in a few high schools, gathering both data and user experiences. [3] Adding an oral part to foreign language matriculation exams was recommended by a task force set up by the Ministry of Education and Culture[4]. The first language to get the oral part will be Swedish.

Aalto University provides the speech processing technology in Project Digitala. Just how much automatic speech processing can help in scoring is still a crucial, unanswered question. It is a vital component in making the oral test feasible. Oral proficiency is not one thing, and should not be measured on a one dimensional scale. Thus a plethora of different question types are needed, ranging from reading text aloud to answering questions orally or communicating with a partner. And thus a plethora of speech processing tools are needed. It is important to research the possible speech processing tools thoroughly to enable a diverse spoken test. Otherwise the negative side of washback may incentivise students to study for the quirks of automatic speech processing instead of better oral communication.

Automatically analysing articulation and then predicting human reviewer scores based on that analysis is a particularly hot topic in Project DigiTala. A computer predicting scores has to be extremely reliable, but it could save a great amount of reviewer time. As a preprocessing step to articulation analysis, the speech is transcribed in words and the speech sounds are segmented into small units. The preprocessing step needs to work well, or the following analysis is done on false premises.

The task of reading aloud may be the lowest hanging fruit for automatic transcribing and segmentation. The content of read speech can be controlled very well.

Therefore it is a good place to start. The topic of this thesis is to implement the preprocessing for reading tasks. The preprocessing is done by *automatic speech recognition*, turning speech into text by computer.

In technical terms this thesis aims to implement the following task: given a text and a recording of someone reading it, transcribe what is actually said and segment the recording into small sound units. The text to be read is referred to as a *prompt* and the corresponding recording of speech is called an *utterance*. Transcribing the text automatically is called *recognising* the read prompt.

In the future automatic recognition and segmentation are also used to *validate* the utterance: to make sure the recording is fit for the articulation analysis system. It does not matter if the recording is rejected as fault of the reader or as fault of the recognition system, because in both cases the automatic articulation analysis should not be used. Instead the recording can be flagged for human review or the student might even be reprompted on the spot, if the recognition and validation system is fast enough.

To recognise and validate the prompt, it is necessary to model reading. A model that predicts the next words in an utterance well makes recognition more accurate. To this end *finite state models* turn out to work well. They can efficiently represent the word sequences that humans produce when reading and they integrate well with automatic speech recognition. The work in creating these finite state models is also the basis for an article and an accompanying demonstration system which will appear in the 2017 Interspeech conference[5].

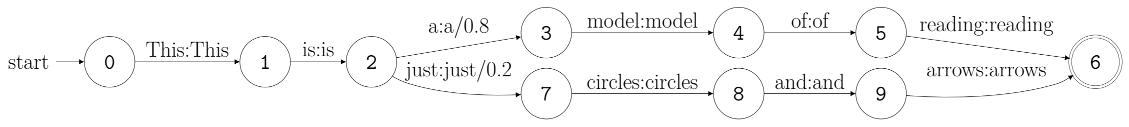


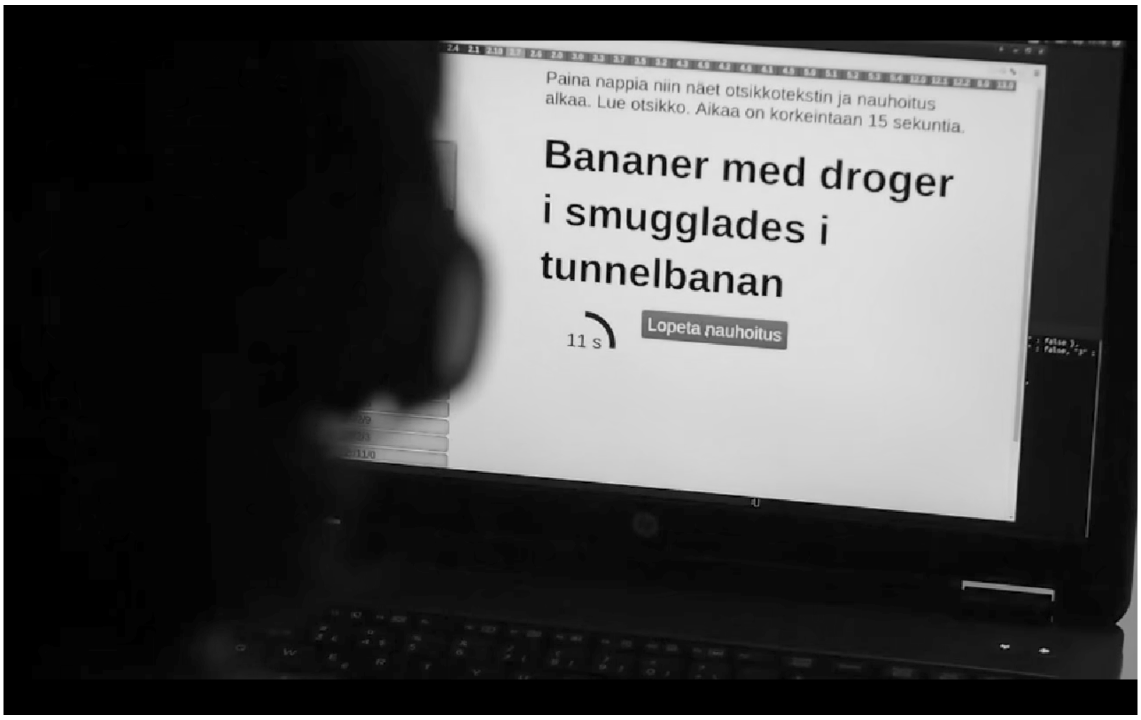
Figure 1: *A preview of a finite state model for reading.*

Figure 1 gives a glimpse of what finite state models for recognition and validation of read prompts look like. They are introduced in sectionsec:finitestatemodels. This thesis builds from a theoretical background towards creating models of reading and then comparing them against conventional techniques in experiments with in-domain data. First automatic speech recognition is introduced. This thesis is concerned with three things in particular: where does the model of reading fit in an automatic speech recognition system, how do a speech recogniser's different subsystems affect the goal of this thesis, and how are existing speech recognition systems used to accomplish the task. Then finite state models are presented both on their own, so as to provide a basis for creating the reading models, and as used in speech recognition, to show how the models are integrated. Next the seemingly ordinary act of reading is studied in a systematic way to provide the last piece of the puzzle of modeling reading and complete the theoretical background.

In the practical part of the thesis, conventional techniques are first proposed as a baseline and then my implementation of finite state models of reading is described in more detail. Then the conducted experiments and their results are recounted. Finally the thesis concludes with a discussion of the results, some ideas for future work and some reflection. Now, before delving into the theoretical background, the DigiTala pilot test is briefly introduced.

1.1 Pilot test

The DigiTala pilot test served two purposes. Firstly it prototyped a computerised spoken test. The feedback was positive. The diverse and engaging tasks in the pilot were liked by the students too. The reading aloud tasks had short tongue-twisters and longer multi-sentence prompts. The other tasks were answering questions in speech, a simulated voice-call, giving directions based on a map, describing a picture aloud and even a pair-wise discussion about summer job opportunities. Figure 2 shows the test from the test taker's perspective. The test was administered with a browser based interface. The responses were saved into a server machine in the classroom.


Paina nappia niin näet otsikkotekstin ja nauhoitus alkaa. Lue otsikko. Aikaa on korkeintaan 15 sekuntia.

Bananer med droger i smugglades i tunnelbanan

Figure 2: *A test taker's view of the DigiTala pilot test. The test taker is reading a tongue-twisting short prompt. The picture is a screen capture from the DigiTala project introduction video,* `https://www.youtube.com/watch?v=p3_UTsjBYJY`

Gathering these responses for further use was the second purpose. This way some data could be bootstrapped to begin experimenting with various speech processing tools. This data is also used as the test data in this thesis.

2 Automatic speech recognition

The first form of language was speech. The written form is secondary[6]. To speak is something an infant picks up, to write is something a child practices. Speech comes naturally without thinking and we can talk while driving, talk while cooking and sometimes we talk while sleeping.

We are also great at listening to speech. We understand what is being said and simultaneously notice the tone the speaker is using. We can detect sarcasm. We can focus on one speaker in a room full of interesting conversations[7].

All of our language abilities are quite effortless. Yet our communication is imprecise and full of ambiguity. We do not always notice this; instead we automatically draw from a wealth of context and real-world knowledge and find a suitable interpretation, which is almost always the correct one.[8]

Computers do not have that intuition, they are lifeless and are best understood without thinking in terms of human abilities, i.e. without anthropomorphism. We may use active words to describe computers: they compute seemingly instantaneously, follow orders to the letter and remember everything exactly. Yet computers are passive. So if we want a computer to understand speech, what we actually have to do is to construct an exact sequence of processor instructions that reads the audio signal and computes from it a desirable representation, usually text, of that spoken phrase. This gets us half way, this is a good start. Then we must consider actually deciphering the intent behind the text, and construct yet more exact processor instructions for that task. Language is a cooperative game between speaker and listener [9] and the computer is not even trying.

Automatic speech recognition is the art of programming, or teaching, a computer to turn speech into its text representation. It is the art of getting to the half way point. The word teaching is appropriate, because the modern approaches are based on *machine learning* [10]. The reverse task of turning text into speech by machine is called *speech synthesis*. The task of comprehending the meaning of the text is called *natural language understanding*.

2.1 A very brief history

Turning speech into text has been an active research topic since the 1950s and remains one today. Commercial applications were feasible in the 1990s, but they were all designed for some particular task, such as answering inquiries about airline flights. Limiting the domain of the system allowed crucial constraints on the complexity.[10] With smart phones, voice operated personal assistant systems were introduced into the everyday. In 2015, Baidu research published a preprint in arXiv claiming to surpass human transcription ability in many read speech tasks [11]. A year later in 2016, in the same forum, Microsoft research published results that they had surpassed a human benchmark in English conversational speech recognition [12].

Conversational speech is widely considered the most difficult to predict. However, cutting edge results may set performance expectations too high. All in all it is difficult to say what the current state of the art is, because it depends on the application. Moreover, it depends on the language: the state of the art results quoted for Finnish in a task similar to the conversational English result of the last paragraph are somewhat worse [13]. Historically, automatic speech recognition research has been spearheaded in English tasks, perhaps due to its status as the research lingua franca[14]. Counter-intuitively, this has also allowed relatively small language areas to cultivate fruitful research, by focusing on the particulars of their language. This is evidenced for example in Finnish with the development of Morfessor[15].

Speech synthesis could be considered a solved problem, but only in the sense arbitrary text can be synthesised intelligibly. Already in 2002 researchers were looking for "perfect synthesis for all of the people all of the time", which includes not only sounding natural, but being able to use an appropriate style and type of voice.[16] Natural language understanding is a broad field, with some successes for example in [17, 18], but much progress to be made.

2.2 What is machine learning?

In conventional programming, the programmers take a task, input step-by-step instructions in code, and the computer accomplishes the desired operation by following these instructions faithfully. Machine learning, in contrast, is the art of programming the computer so it learns to perform the operation without the salient detailed instructions. Typically this is done because we simply do not have a step-by-step algorithm.[19] Humans are able to listen to speech and decode its message, but we do not know how exactly we do it. We do not experience decoding the sounds, it happens subconsciously and automatically.

The lack of clear algorithm is often compensated for by abundance of data. We program a more or less generic learning method, typically utilising *pattern recognition* and statistical methods. Then we feed this method a set of inputs, or observations, and expect it to extrapolate a *model* which can perform our task.

By more generic learning method we mean that there are frameworks which theoretically allow us to approximate any input-to-output relation we want [20], which in practice have been successfully used for many very difficult problems. By less generic we mean that we often want to include all the knowledge we do have about the specific task we are trying to accomplish. The more specific and restricted we can make our task, the better we can usually accomplish it.

2.2.1 It is all about data

In the context of automatic speech recognition, even the earliest systems were based on matching spoken digits (0 to 9) to templates estimated from data [10]. More modern systems are built from very large datasets, or *corpora* (the singular is *corpus*), with possibly over a thousand hours of recorded speech and gigabytes of text. The text datasets can be simply large texts collections. The audio datasets are also large collections of recorded speech, but they need an accurate accompanying transcription. There is a truism in machine learning:

> There is no data like more data.

While not all data is equally valuable[21], it is almost always possible to take advantage of more data.

The dataset that is used for learning is called a *training set*. The type of data in the training set will give an inherent *bias*. If, for example, the training is constructed from 1970's rock music lyrics, the data gives quite a peculiar view of English overall, although it does match its own domain well. An essential problem in machine learning is that the models become too specific to the training set and do not describe any other observations well. This is called *overfitting*. To counter this, a part of the training set is held out from the model learning. Model performance is evaluated on this *validation set* (also called a *development set*) to check for overfitting. Often we have some choices to make about the learning methods, such as which kind of statistical distribution to use; evaluation on the validation set is also used to look for the best choices. Since the validation set is used to make these kinds of choices, it will also impart some bias to the model. Therefore, when we wish to estimate the true performance of the final model, it has to be done on a completely separate dataset, a *test set*, to get a meaningful estimate. [19]

There are an infinite number of possible expressions of language, so gathering statistics about all of them is impossible. This problem is not just theoretical; there are so many different expressions of language that people actually use that it is very difficult to gather a training set with sufficient representation. This is called the *data sparsity* problem [22]. In other words, most machine learning models have some number of parameters, e.g. the mean and variance in one dimension of the training data, and to estimate a large number of parameters a large number of training instances are needed.

2.2.2 Cross validation

The technique of training a model on one dataset and then testing its performance on another is called *cross validation*. Typically just one dataset is collected, from which a portion is held out as the test set right at the start. The rest of the data is used as a training and validation set. In the end, the final model is tested on the test set. The test set should be large enough to give statistically meaningful estimates.

This means that a part of the collected data is not used for building the model. In case the dataset is already small to begin with, this can be problematic. One solution is to use *k-fold cross validation*, where the dataset is divided into k folds and the training and testing procedure is repeated k times. Each fold serves as the test set in one train-test-iteration and the rest of the folds make up the training set. The results are gathered from each iteration's test set. Effectively, the whole dataset functions as both the training set and the test set, but in a way that does not train and test on the same samples simultaneously. A typical choice of k is 10.[23] The downside of k-fold cross validation is the increased computational cost.

2.3 System overview

This section presents the structure of a typical automatic speech recognition system. This is also the structure of the system used in this thesis. In machine learning terms, automatic speech recognition can be considered a *classification* task, where we attempt classify incoming audio segments into words.

The automatic speech recognition system has two major sources of knowledge: an *acoustic model* and a *language model*. The acoustic model represents the sounds of speech. The language model defines how words form phrases. Connecting these two models is the *lexicon*. It defines how the speech sounds form words. Different languages have different sounds, different ways of mapping the sounds to words and different ways of connecting words to form phrases.

With the acoustic model, we can take an unknown audio signal and search for sounds that resemble the sounds of our chosen language. With the lexicon, we can take a sequence of these resemblances and find words that could be constructed from them. With the language model we can take a possible sequence of words and say how well it conforms to the ways the language usually forms word sequences.

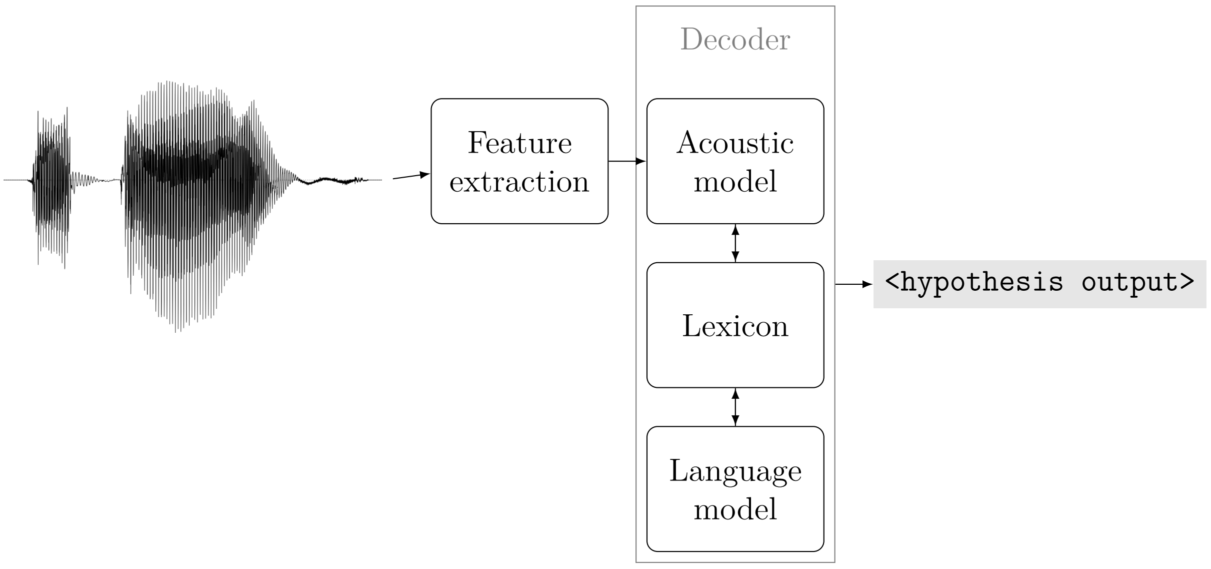


Figure 3: *A typical automatic speech recognition system structure in decoding.*

Having a separate the acoustic model and language model is not the only possible choice, but it has many benefits. It allows for them to be constructed separately, which has quite fruitfully allowed researchers to focus on smaller, better defined portions of the whole automatic speech recognition task and improve it incrementally. It allows us to represent the innate heterogeneity of automatic speech recognition. [24] It allows for taking one model and using it in many different contexts. For example we can take a good acoustic model and use it with multiple language models.

2.3.1 Formulating the decoding task

To offer a high level view of what questions the acoustic and the language models answer, the mathematical formula of speech recognition is presented here. Taking in the audio, combining information from the acoustic model, the language model and the lexicon, and then finding the best matching sequence of words is the task of the *decoder*.

Figure 3 shows a schematic of a typical automatic speech recognition system in decoding. Mathematically the search problem of the decoder can be defined as the problem of finding the most probable sequence of words $\hat{w}$ given a set of observations $\boldsymbol{X}$,

$$\hat{w} = \arg\max_{w}\{P(w|\boldsymbol{X})\}. \tag{1}$$

We wish to express this probability of word sequences given observations $P(w|\boldsymbol{X})$ in terms of the acoustic and language models. The problem is transformed with the Bayes' theorem:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}. \tag{2}$$

Given the $\arg\max$ in equation 1 we may discard the denominator. Thus we get the equivalent task:

$$\hat{w} = \arg\max_{w}\{P(\boldsymbol{X}|w)P(w)\}. \tag{3}$$

Here, $P(\boldsymbol{X}|w)$ is the likelihood of seeing observation sequence $\boldsymbol{X}$ given a hypothesised word sequence w, which is what an acoustic model directly computes. $P(w)$ is the probability of word sequence w, which is exactly what the language model gives us. In practice this product is calculated in the logarithm domain, with a *language model scale* α, which allows for controlling how much weight, or trust, we place on each model. [25] Then equation 3 looks like this:

$$\hat{w} = \arg\max_{w}\{\log P(\boldsymbol{X}|w) + \alpha \log P(w)\}. \tag{4}$$

2.4 Acoustic modeling

A speech signal has information in the way the signal changes through time and in how the slices of time look in the frequency domain, as shown in the time-frequency plot, or *spectrogram*, in figure 4. The acoustic model represents both the frequency domain features and the passage of time.

The output classes into which the acoustic model classifies the incoming feature vector sequences could be chosen arbitrarily. Each word could have a separate class. However each new word would need a lot of training instances of just that word and large vocabularies would need a huge number of classes, which is not desirable. Instead smaller output units based on *phones* are used in practice. Just as the written language is built from letters, phones form the alphabet of speech: they are the smallest distinct unit of speech. A language has a finite set of phones from which all words may be built and thus the vocabulary size does not affect the number acoustic output classes. English can be divided into about 45 phones[26], though the divisions

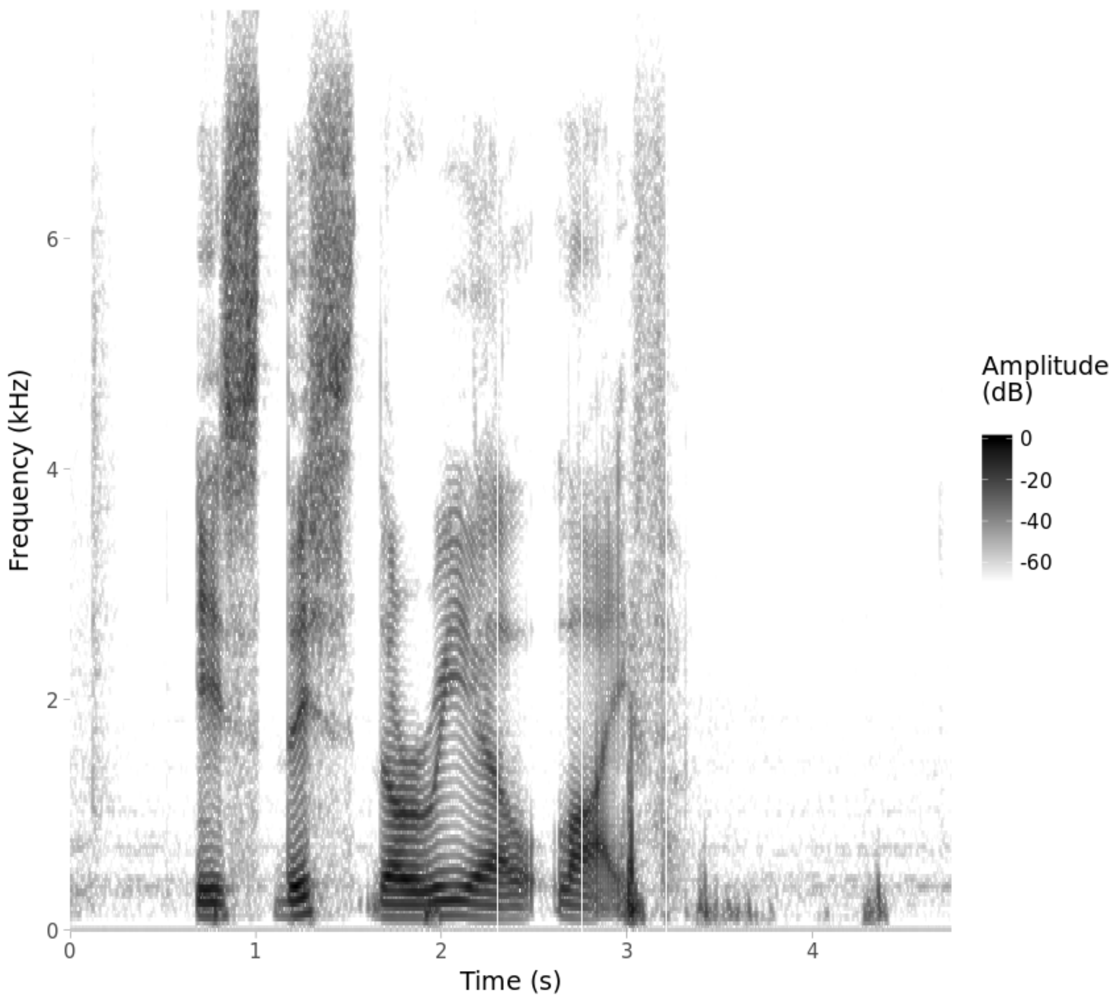


Figure 4: *A spectrogram of the phrase* `Is this the real life?` *The* `/z/` *and* `/f/` *sounds can be found visually: they are tall dark columns from wide bandwidth noise and don't have undulations from a harmonic structure.*

are somewhat arbitrary. The phones are often denoted by slashes, e.g. `/dh/`, `/ih/`, and `/z/` for the pronunciation of `this`.

Although it is said that phones are the smallest distinct unit, the realisation of a particular phone varies greatly depending on the neighbouring phones being pronounced. To take this into account, the actual output unit classically used in acoustic modelling has been the triphone, i.e. a phone with two neighbour phones as context, e.g. `dh-ih+z` for the middle triphone in the word `this`. Furthermore the classes which the acoustic model uses internally are even finer grained: they are nodes of a graph that represents the passage of time.

Acoustic modeling begins with the audio. The first step is to extract some salient measurements, or *feature vectors*, from the signal. This step is called *feature extraction*. After briefly describing feature extraction, the acoustic model is built in two phases. First the graph that represents time is introduced and then the *feature space* is explored.

- Unique speech organs controlled by a unique brain
- Mood and emotions
- Situational effects (such as talking differently in loud environments due to the Lombard effect[27])
- The acoustic environment
- The recording equipment and how they are used

Table 1: *Factors which make speech signals varied*

2.4.1 Feature extraction

What makes acoustic modeling difficult is that speech essentially never sounds the same. Reasons for this are listed in table 1. It means that a speech signal carries with it a lot of information that is not essential for recognition. Before the audio signal is fed to the acoustic model, it is preprocessed to remove as much redundant information as possible and to emphasize characteristics which are important for classification. The end result is a sequence of feature vectors, which correspond to short (typically some 30ms), overlapping segments of the audio. For a long time, *mel-frequency cepstral coefficients* (MFCC) have been the standard feature type. Table 2 shows the steps in extracting MFCC features. The resulting MFCC feature vectors lie in a continuous feature space.

To better capture the temporal characteristics in speech, the difference of consecutive vectors is classically appended to the vectors; and then the difference of the difference (delta and delta-delta features)[26]. The features can also be transformed to make the acoustic model match them better[29, 30].

2.4.2 The time axis

The passage of time in a digital application is inherently a matter of taking discrete steps. It can be encoded as moving through paths between nodes in a graph: each transition takes one time step and at each time step we are at a given node. In automatic speech recognition a very specific graph, a *Markov chain*, is typically used as an approximation. It is a *stochastic* graph where the probability of moving to a given node depends only on the current node. This means the graph has the *Markov property*. Figure 5 depicts a Markov chain which models the process of pronouncing `is`. The first time steps steps will consist of producing the `/ih/` sound, taking the self-loop transition back to the `/ih/` node. At some point the speaker will move onto the `/z/` sound, and eventually stop that too.

- First, the audio is converted from a discrete sampled form into the magnitude spectrum by short-time fourier transform. Unique sample values do not give any information; the information is in their relationships, which are naturally represented in the frequency domain. The temporal resolution is usually such that each feature vector covers 30ms, but the time from the previous vector, ie. the *frame shift* is only 10ms. The intuition is to capture an approximately stationary slice in time: the fundamental frequency of an adult male is typically around 100Hz, leading to a 10ms period. At a 16kHz sampling rate, a 30ms frame gives 480 point vector to start with.
- The spectral representation is filtered by a mel-scaled filter bank, which reduces the number of features and approximates human hearing. As speech has evolved dependent on hearing, it is intuitive to assume that human hearing should capture the essential in speech.
- The mel-spectral representation is then converted to the cepstral domain by taking a logarithm and a discrete cosine transformation. The cepstrum is a sort of spectrum of a spectrum. Some higher order cepstral coefficients can be discarded; they represent finer details in the frequency domain. Thus a very compact vector representation of for example just 13 dimensions is reached.

Table 2: *Steps in extrating mel-frequency cepstrum coefficient features[28].*

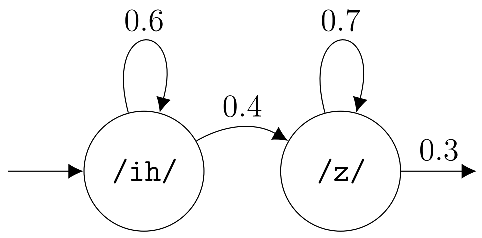


Figure 5: *A Markov chain for pronouncing the word* `is`*. The transitions into and out of the chain are used to combine many different chains. The transition probabilities in the graphic are simply illustrative.*

Upon entry, each node produces a new feature vector, which is the only thing we see in reality. From this point of view, we can state the decoding task as: finding the sequence of transitions which the speaker took, that ended up producing the sequence of feature vectors that we got. Since the actual sequence is not seen, the model is called a *hidden Markov model*. Figure 6 depicts a hidden Markov model of `is`. The hidden Markov model is *generative*: each state has a probability distribution over the feature space and is thought to generate, or *emit*, the feature vectors randomly according to this distribution. The core of the acoustic model is estimating this probability distribution.

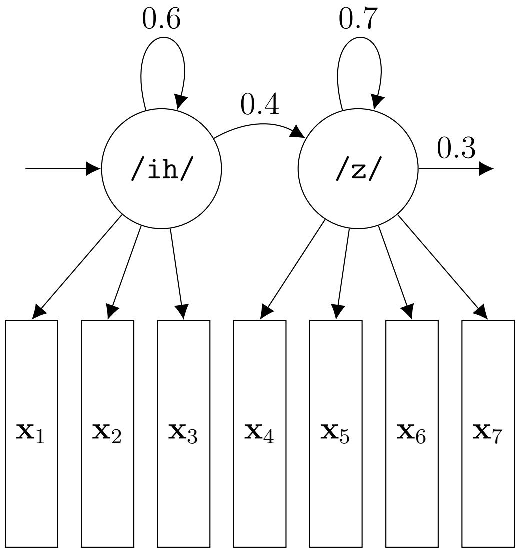


Figure 6: *A hidden Markov model of pronouncing the word* `is`*. The underlying Markov chain is the same as in figure 5, but we only see the feature vectors* $\boldsymbol{x}_{1\ldots7}$*. Note that in these visual examples we are assuming a left-to-right topology, which is also a typical constraint. In other words, this means the states are forced to be in strictly chronological order. Note also that the amount of feature vectors is too low; this is just for illustration purposes.*

As previously stated, the typical class of an acoustic model is the triphone. Each class is represented by a hidden Markov model that has some three to five states. This gives us the final representation of the pronunciation of `is`, shown in figure 7

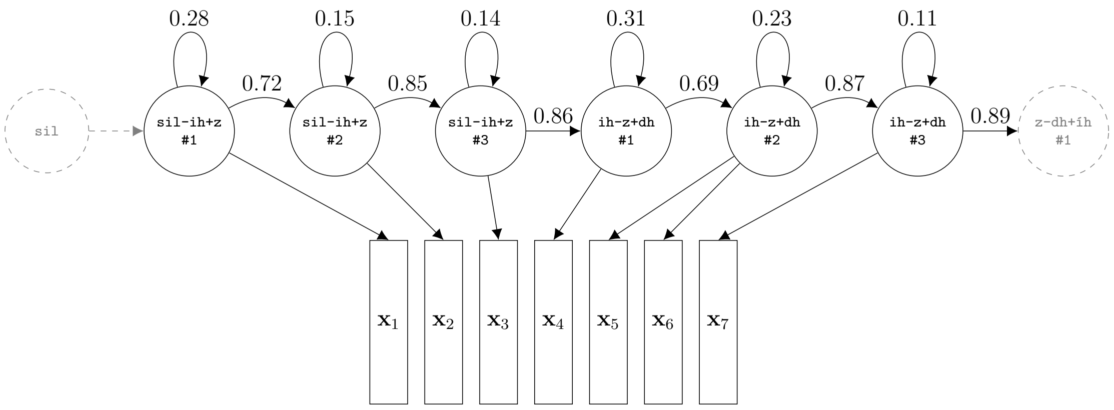


Figure 7: *A triphone tristate hidden Markov model of pronouncing the word* `is`*. The first triphone of a word depends on the previous word and the last depends on the next word. Here we are assuming the utterance beginning (from silence) with* `is` *and continuing with* `this`*.*

2.4.3 Feature space representation

The slices of time are represented by the feature vectors. The acoustic model internally classifies feature vectors into the states of the hidden Markov models of each triphone. Each feature vector is a collection of continuous variables, and so each state of each hidden Markov model has a multi-dimensional probability distribution which gives, for a particular feature vector, the probability of having been emitted by that state. Each multi-dimensional probability distribution is described by a set of parameters.

All of this gives a very large number of parameters, which requires a large number of training instances. In practice, most triphones are rare or never used and conversely a minority of all possible triphones take up a large majority of all heard instances of the language and consequently the training data.[26, 31] We encounter the problem of data sparsity, as described in section 2.2.1. A number of methods exist to more accurately model the rare or unseen triphones. A classic approach is to tie the parameters of a phone in similar contexts together, such as the middle state in both `dh-ih+z` and `f-ih-sh`, basically reducing the number of classes overall [26]. Another more recent idea has been to create a subspace of the features and construct each state's distribution as a lower dimensional vector in this space [32].

The question still remains of how to represent the probability distribution. There are two main paradigms: *Gaussian mixture models* and *deep neural networks*. Gaussian mixture models are an older method that still has some use cases, while deep neural networks have become the standard for recognition if enough training data is available, because they achieve better accuracy.

2.4.4 Gaussian mixture models

A single *multivariate Gaussian distribution* $\boldsymbol{\mathcal{N}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a generalisation of the familiar Gaussian distribution into many multiple dimensions. The familiar univariate Gaussian probability density function is:

$$\phi(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

This generalises to multiple dimensions quite simply. The biggest difference is that the variance σ^2 of the univariate distribution becomes covariance matrix $\boldsymbol{\Sigma}$:

$$\boldsymbol{\phi}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})},$$

where $|2\pi\boldsymbol{\Sigma}|$ signifies a determinant. Although MFCC features are not fully decorrelated, their covariances should be quite small. In order to reduce the amount of parameters to be estimated, the covariance matrix is often constrained to be diagonal, i.e. the covariances are zeroed and only the variances in each dimension remain. Figure 8 shows a multivariate Gaussian distribution with zero covariances. The projections of the distribution to the component dimensions show two familiar univariate Gaussian distributions.

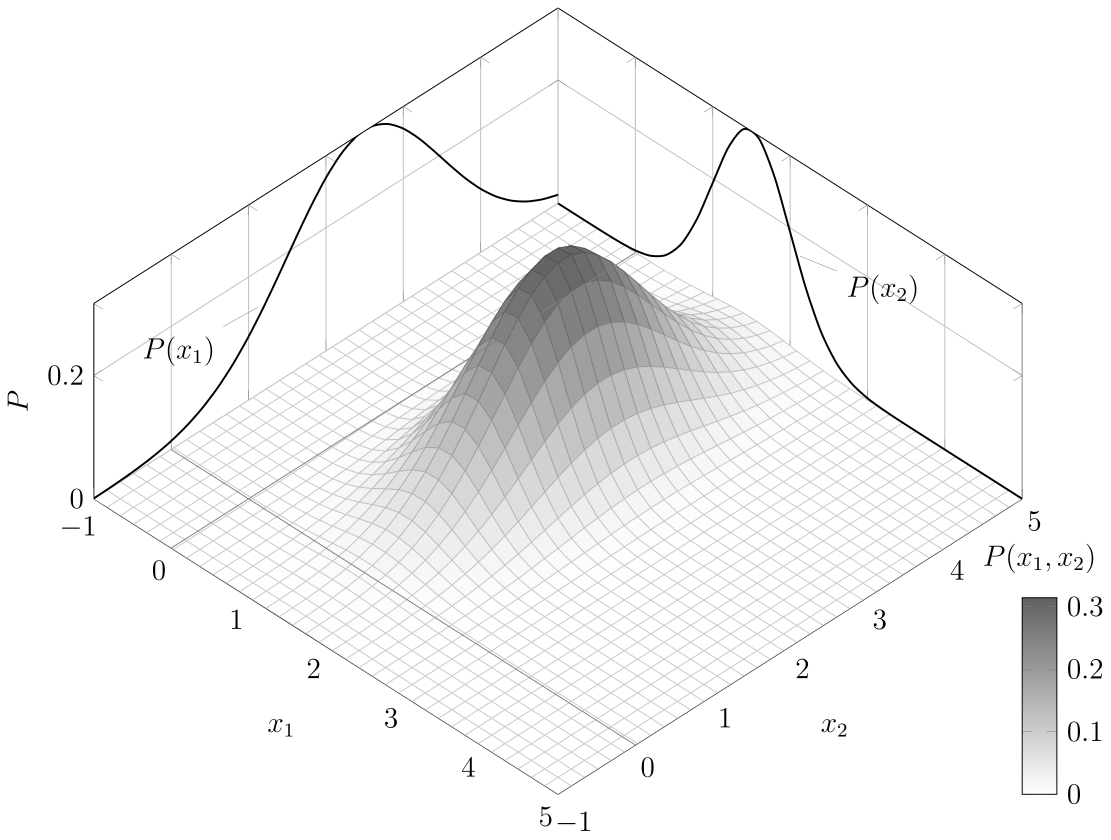


Figure 8: *A single multivariate Gaussian distribution. This is more specifically a distribution of just two variables, a bivariate distribution, since graphical representations in higher dimensions would be unhelpful. The feature space in automatic speech recognition is in reality much higher dimensional.*

A Gaussian mixture model is built by linearly combining multiple multivariate Gaussian distributions. This gives a very flexible distribution of the form:

$$\boldsymbol{\phi}(\boldsymbol{x}|c_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K) = \sum_{i=1}^{K} c_i \boldsymbol{\mathcal{N}}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i).$$

Here, c_K are the mixture component coefficients, which must sum to unity. Figure 9 shows a mixture of multivariate Gaussians. The projections of the distribution to the component dimensions now show more complex forms than with a single multivariate Gaussian.

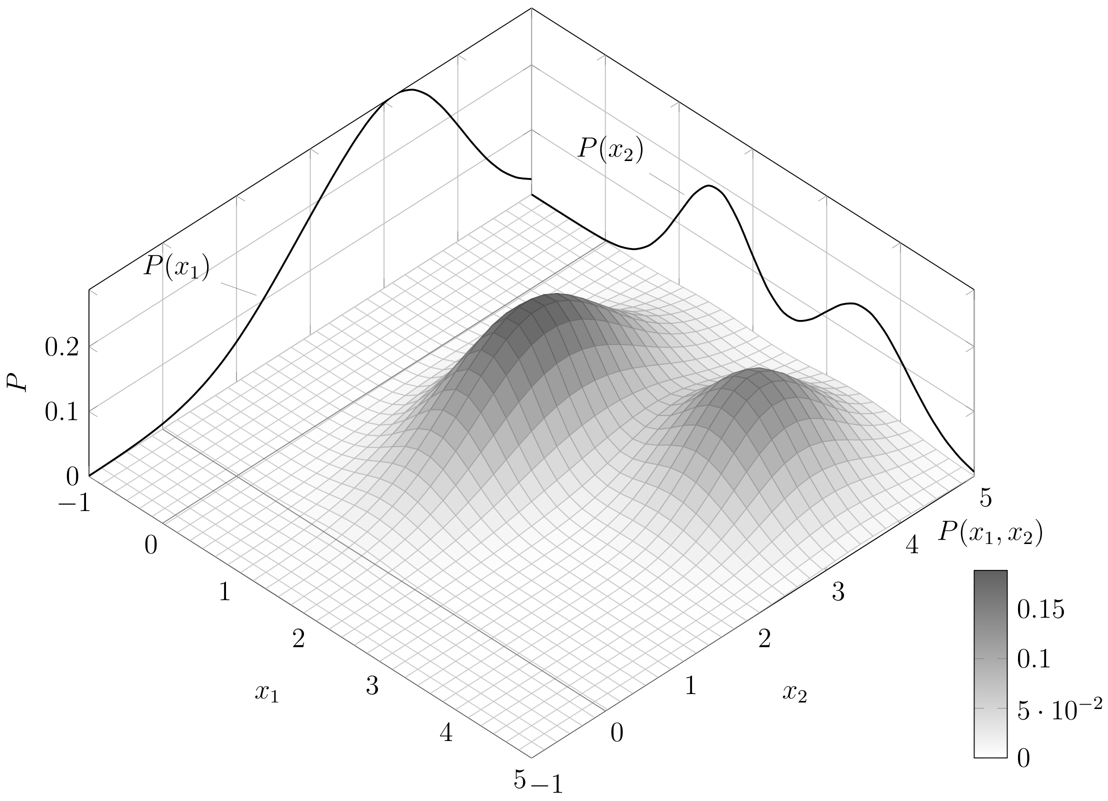


Figure 9: *A mixture of multivariate Gaussian distributions.*

The training procedure for hidden Markov models with Gaussian mixture model distributions is unified, and simultaneously estimates transition probabilities, which Gaussian mixture component a state belongs to, the mean vector and the diagonal (co)variance matrix for each Gaussian mixture component, and mixture component coefficients. While the training equations are not in the scope of this thesis, it should be noted how this type of training produces a model that in decoding aligns the hypothesis well temporally.

2.4.5 Deep neural networks in acoustic modeling

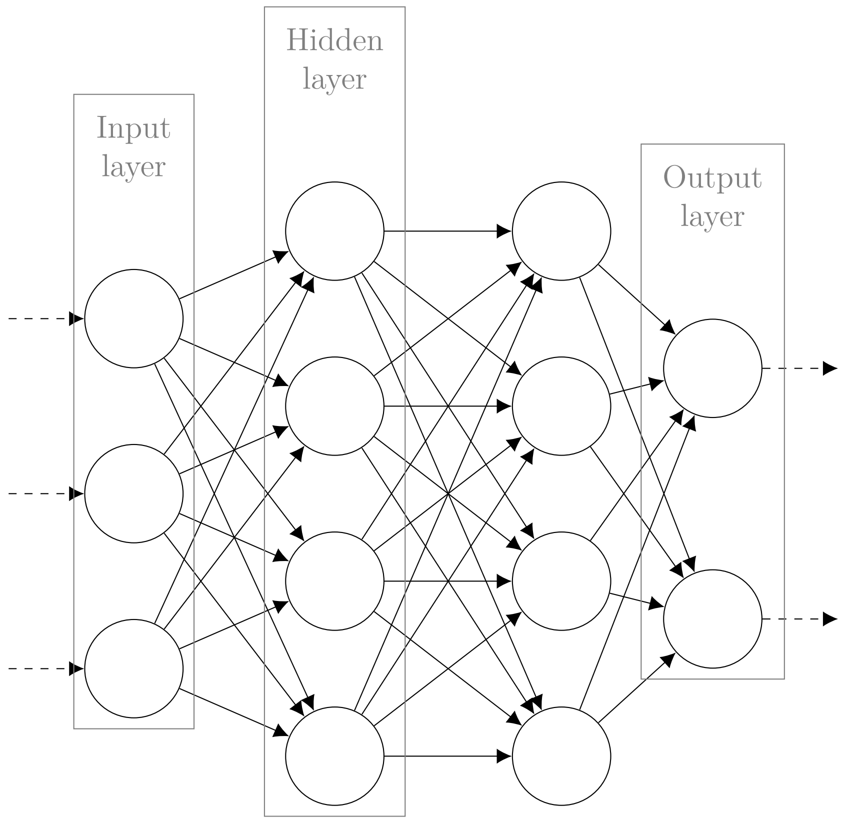


Figure 10: *A generic artificial neural network structure. The circle shapes are called neurons.*

Artificial neural networks are a large family of machine learning methods. The family is very diverse, but figure 10 shows a general structure that conveys the basic computational idea. The network is organised in *layers*. Each layer has some amount of *neurons*. The each feature vector component becomes the value of one neuron in the input layer. After the input layer come a certain amount of *hidden layers*, called so because they are not visible at the input or the output. Each neuron forwards its output value to each neuron in the next layer. Figure 11 shows the structure of a basic neuron. It has a weight for each input from the previous layer. The inputs from the previous layer are weighted, then summed together with a bias term. This sum is then ran through an activation function, which is typically nonlinear. The value of the activation function is the neuron's output value. The nonlinearity gives the artificial network the ability to represent nonlinear input-output relations. This type of neuron is called the perceptron. The networks are typically trained by *backpropagation of errors* from the output to the input, which is beyond the scope of this thesis, but means that the artificial neural network can learn to minimise any function which we can differentiate for the weights of the network. This function is then called the *objective function*. Different types of neural networks are constructed with different arrangements of layers, different types of neurons and different objective functions.

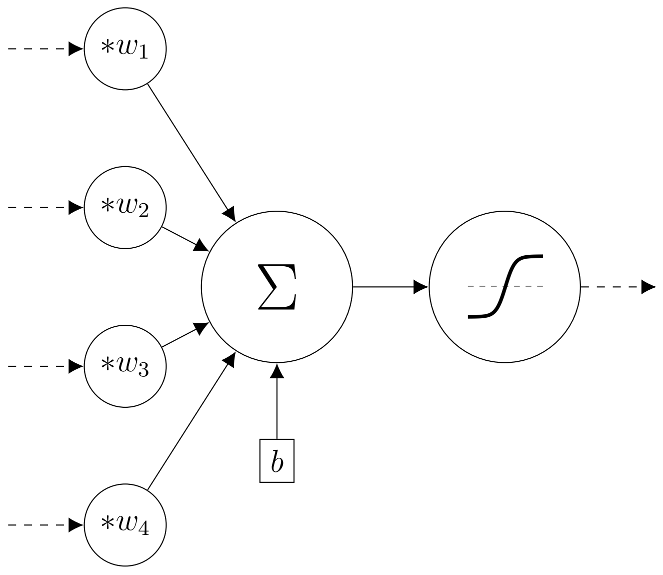


Figure 11: *A basic neuron type called a perceptron. The learned parameters of a neuron are the weights for each input and the bias. Typically the nonlinearity in the activation unit does not learn its parameters.*

The first neural network acoustic models did not perform much better than the Gaussian mixture models of their time. In the 21st century, methods have been found and computing capacity has increased enough to make it possible to teach neural networks of very many layers and many neurons per layer. These very large neural networks are called deep neural networks and the teaching of these deep networks is called *deep learning*. Though artificial neural networks have long achieved competitive results, the traditional non-deep models do not outperform other methods[33]. Deep learning has however allowed incredible leaps in the state-of-the-art in many different artificial intelligence tasks[34]. This has happened for acoustic modeling too[35]. The many layers of a deep neural network learn to represent features of gradually increasing complexity. Remarkably the network learns these representations from the data; they do not need to be hard coded by experts. This makes the deep learning methods very flexible.

There are two main branches of neural network acoustic models. The first is simply having the neural network predict the likelihoods of hidden Markov model states given an acoustic feature vector, $P(state|\boldsymbol{x})$. This is then transformed into the emission probability $P(\boldsymbol{x}|state)$ simply via equation 2. Another architecture, called connectionist temporal classification, has also had some notable successes[36, 11]. This method learns to output character sequences directly by using a specific objective function. An important part of the idea is that since in pure recognition tasks the time-alignment is not needed, it is a waste to learn to create it. An objective function called *lattice free maximum mutual information* is a sort of fusion of connectionist temporal classification and traditional hidden Markov model deep neural networks. It is typically used in conjunction with a reduced output frame rate.[37] Though this method gives good transcription accuracy, the temporal resolution may suffer from the reduced output frame rate and the connectionist temporal classification -like objective function.

2.5 Language modeling

The language model represents the patterns and structure found in the chosen language. This is not the formally defined grammar of the language, but knowledge about sequences of words, i.e. knowledge about the ways in which words are used to form phrases.

In equation 3 the language model gives the probability $P(w)$. The word sequences for which this probability is zero cannot be recognised; they are ruled out by the language model. This way the language model can be used to drastically reduce the search space of the recogniser. In the general case the zero probability is an underestimate and needs to be avoided[38].

The language model can provide good guesses when the acoustic model alone cannot. For example, the language model can be used to disambiguate similar sounding words, even homophones, words with the same pronunciation but different meaning or spelling. Consider the sentence `"Come here and describe the songs you hear."` The words `here` and `hear` sound exactly the same, yet there is intuitively very little ambiguity about which one is meant each time.

2.5.1 N-gram models

The traditional way to construct language models for automatic speech recognition is by the approximation that the probability of a given word only depends on the N-long history of words. These word N-length word sequences are called *N-grams*. A three word history, meaning the word itself and two previous words, is called a *trigram*. Similarly, a two word sequence is called a *bigram* and single words can be called *unigrams*. The probability of a word sequence $P(w)$ can be deconstructed into a sequence, a chain of multiplications using the chain rule[1]:

$$P(w) = P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)...P(w_N|w_{N-1}, ..., w_1).$$

In this equation the conditional probability $P(w_N|w_{N-1}, ..., w_1)$ is called the N-gram probability. It is the probability of a word appearing next, given the $N-1$ previous words. [26]

The N-gram probabilities can be estimated from a text corpus simply counting their frequencies, with

$$P(w_N|w_{N-1}, ..., w_1) = \frac{f(w_1, ..., w_N)}{f(w_1, ...w_{N-1})},$$

where $f(w)$ is the frequency of the word sequence w. Let's use the lyrics to Bohemian Rhapsody as a toy example. The word sequence `Galileo Figaro` appears once,

[1] $P(a, b) = P(a|b)P(b)$

and the word `Galileo` appears five times altogether. We can then declare that the bigram probability $P(\texttt{Figaro}|\texttt{Galileo}) = \frac{1}{5}$ for a bigram model trained on that text.

This basic idea has some very attractive qualities. It is easy to use and to understand. In some languages it captures local dependencies well. There is a major problem in this approach. For a vocabulary of size V, there are V^N N-grams. This number becomes so large so quickly that most possible N-grams don't appear even once in the training data. Even a small 1000 word vocabulary has 1 000 000 000 possible trigrams. It would obviously be an underestimate to say that all the unseen word sequences are impossible. This is the archetypical example of the data sparsity problem mentioned in section 2.2.1. It is mitigated by *smoothing*.

2.5.2 N-gram smoothing

N-gram models are built from discrete event counts, counts of each word sequence. Therefore the probability distribution which maximises the likelihood of seeing the data that we have, the *maximum likelihood* estimate, would be a spiky distribution with zero probability for any unseen sequences.[39] It is intuitive in the sense that, if the data is all the knowledge we have, for all we know the other sequences are forbidden and will never be seen. But in the case of language the data is not really all we know; we know that all sequences should have some small probability. It is easy to produce ungrammatical sentences. We can redistribute some of the probability mass from the spikes in our maximum likelihood distribution to all the unseen sequences. This redistribution is called *smoothing*, and just the removal of probability mass is called *discounting*.

There are many smoothing methods, as Goodman and Chen present in [40] and [38], and whom we summarise here. A very simple one would be to add one to all N-gram counts, thus pretending all sequences are seen at least once. This indeed leaves no impossible sequences. Effectively, it takes probability mass from the seen sequences and gives it to the unseen sequences. But it makes all unseen sequences equally likely. For example, it would make `fantastic life` and `fantastic vitae` just as probable (if both are unseen), even though `vitae` is a much rarer word. The rarity would be well accounted for by the unigram probability. In fact for good performance we need to combine information from many models, which often means N-gram models of different lengths, but can also mean other knowledge sources. These basic steps in building better N-gram models are then:

1. Discount the original counts of sequences to
   (a) account for the excessive spikiness
   (b) have some probability mass for further operations
2. Combine with some lower order models.
3. Take care of zero probability sequences. This typically happens as part of step 1a or step 2 or both.

The *Kneser-Ney* smoothing method is presented here, and used later in experiments.

The Kneser-Ney scheme does not simply consult, or *backoff* to, the (N-1)-gram in case of missing N-gram; it takes into account that some words complete many N-grams naturally whilst some words are only used in very specific contexts. The word `life` will be found preceded by a wide range of adjectives. In a human-resources domain corpus the word `vitae` may appear quite often, giving a high unigram probability, but only in the context of `curriculum vitae`, which is already well matched by the bigram model. The backoff probability should be low. This idea is introduced on top of *absolute discounting*, where a discounting constant δ is subtracted from the N-gram count, giving:

$$P(w_N|w_{N-1...0}) = \begin{cases} \frac{max(f(w_1,...,w_N)-\delta,0)}{f(w_1,...w_{N-1})}, & \text{if } f(w_1,...,w_N) > 1 \\ \alpha\frac{\textsc{NumContexts}(w_N)}{\sum_{v=1}^{V}\textsc{NumContexts(v)}}, & \text{otherwise} \end{cases}, \quad (5)$$

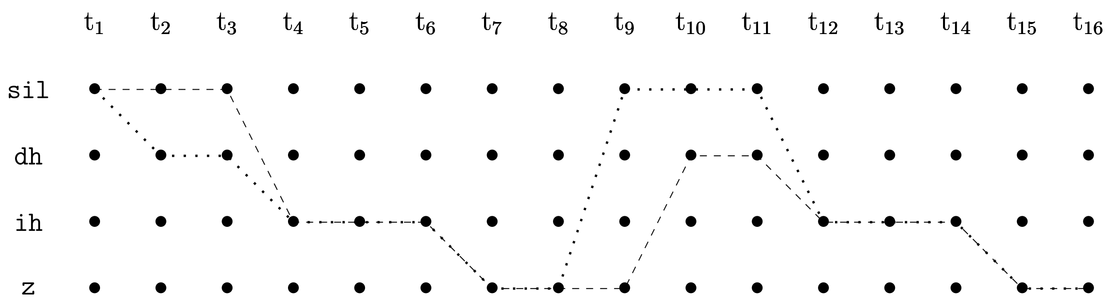
where v is a word in the whole vocabulary V and α is a normalising variable which is determined by the constraint that the whole probability distribution sums to unity.

Manning and Schütze find that the Kneser-Ney backoff model (and particularly a modified variant introduced by Goodman and Chen) performs well in most situations.Although the Kneser-Ney variants are still an adequate baseline, recently recurrent neural network language models have outperformed all others given enough training data [41, 42].

2.6 Decoding and aligning

With the acoustic and language model likelihoods ready to be plugged into the speech recognition equation 3, it is time to find out how this equation is actually computed. Computation with the acoustic model takes approximately the same amount of time for each incoming feature vector and is quite fast. The word sequence search space size is defined by the language model. In typical modern applications which require *large vocabulary continuous speech recognition*, the large amount of different words we want to search for and the variety of sequences we can build from them make the search space large to begin with and infinitely large for arbitrarily long utterances. To make decoding computable in reasonable time, i.e. *tractable*, not all possible paths are explored. The decoder implements a search algorithm like *beam search*, where only the most promising paths are kept and the rest are *pruned* away at the end of processing each feature vector.

Figure 12 represents the search space of the decoder. On the horizontal axis is time. One feature vector is in each time step. Each dot in the grid is a possible choice of acoustic class for that time step. Exactly one class is ultimately chosen for each time step. Each dot has an feature space acoustic likelihood computed from that time step's feature vector with that class' acoustic model. Each path in the grid accumulates Markov model transition weights and language model weights. The Markov model weights are added at each time step corresponding on which class-to-class transition is taken. The language model weights are only added at the beginning of each new word. The starting point of each new word could be at the start of any phone. However traditional tristate hidden Markov model structures take at least three time steps to traverse through, so each phone takes at least three time steps.



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16}
sil
dh
ih
z

Figure 12: *The search space of a decoder. The dotted line represents a path hypothesis for* **`this is`***, the dashed line for* **`is this`***. The naive implementation has to try all paths and word boundary positions. The labels t_i denote the time steps, and for simplicity this figure uses context independent phones.*

The search space of the decoder is not trivial to represent. A compact representation is valuable for memory and computation time considerations. Since the search prunes unlikely paths away, a compact representation can also give a better recognition accuracy; less paths need to be pruned, so the correct transcript can be found even if it looked unlikely in the beginning. It turns out the finite state models introduced in section 3 can create such a compact representation.

A task closely related to decoding is *alignment*. In alignment the word sequence, and thereby the phone sequence, and thereby the acoustic class sequence, are already known. This is the case when training an acoustic model, when the training transcripts are available. The exact path through the grid of figure 12 is still not trivial to find. In fact at the granularity of individual time steps, the borders of each phone and word are not exactly defined. The task of alignment is to compute the most likely path through the grid. The acoustic model likelihoods define this path.

2.7 System quality metrics

In automatic speech recognition, performance is a many faceted question. If it is separated from any particular task, the system takes an audio signal and should produce the transcription, and we'd want to measure how close the system's prediction is to the correct answer. However any real task the system is used for will have its own goals; the speech recognition system should actually be designed to best serve the task.

We wish to measure the performance of an engineering system for two reasons: to know how good it is and to compare different systems. In speech recognition it can be difficult to give a precise threshold for performance, after which the system is good enough to have an overall benefit. Sometimes speech recognition systems can simply be compared to other ways of achieving the same end. As an input method, speech recognition can simply be compared in speed against typing, and in fact modern system can be considerably faster[43]. This measure conveniently takes into account that the speech recognition system makes some errors, which the user has to correct by hand, and because of the switch of medium, the corrections are probably more costly than typing errors.

2.7.1 Recognition performance

A one dimensional, universal score is helpful. It allows comparing different systems on the same metric and it allows for more intuitive understanding of the system's performance. In speech recognition, the metric is *Levenshtein distance*[44]. It is a measure of the distance from one string to another. When used as a performance metric it is measured from a hypothesis string to a reference string. It is typically measured with words as units. In that case, the percentage $\frac{\text{Levenshtein Distance}}{\text{Number of words in target}} * 100\%$ gives the *word error rate*. *Agglutinative languages*, such as Finnish, build words from small units and thus end up with a huge vocabulary of words very close in their pronunciation. In their case, a one character recognition error is hard to avoid but may not hinder understanding very much. Therefore the word error rate may be too punitive and instead letters are used as the unit of distance calculation, giving the *letter error rate*.

The metric counts three different types of errors:

- substitutions of a unit with another,
- insertions of extra units,
  and
- deletions of units (i.e. missing units).

The distance is an integer quantity of errors, so that the error count is minimized. This minimisation is needed since a substitution can be interchanged with an insertion and a deletion. The shortest path of correct units, substitutions, insertions, and deletions from a test string to the target string is the *Levenshtein alignment* of the two strings.

The word error rate is presented as a percentage, but it can exceed 100%, since the amount of insertions is unbounded. Thus it cannot be directly interpreted as the probability of getting a word wrong. Moreover, a speech recognition system's errors are not randomly distributed among the words of the reference, but instead they often co-occur and form patterns.

2.7.2 Significance testing for word error rates

Every measurement has some level of uncertainty associated with it. Traditionally in statistics some form of significance testing is performed to estimate whether a finding is trustworthy. It is particularly important if the amount of data is small, when the law of large numbers does not apply. Unfortunately word error rate is not well suited to typical methods. The approach in speech recognition has been the use of standardised training and test sets. Algorithms can be compared on these and the data sets have a plethora of results published on them so adequate benchmarks are available.

In many cases, some form of statistical significance testing would still be desirable. Bisani and Ney (of Kneser-Ney backoff fame) have introduced a method for estimating confidence intervals for word error rates on a test set[45]. The method is based on *bootstrapping* in which essentially many smaller subsets are sampled *with replacement* (i.e. the same sample may be picked multiple times) from the initial test set. This way a multitude of word error rate estimates are produced, which yield an average word error rate and, crucially, its confidence intervals.

A measurement of the word error rate of an automatic speech recognition system on a particular test set may tell a lot about the speech recognition system, but it may also tell a lot about the test set. The test set may just happen to be very well designed for that algorithm. It seems the best algorithms stand the test of multiple standardised test sets.

2.7.3 Optimising for different types of errors

Given a classifier that has a particular level of performance over the whole test set, we may also be interested in the type of errors that it makes. Particularly this is of interest in case the classifier makes a diagnosis of some phenomenon happening or not. A medical example is typical: detection of a disease. The rate of correctly detecting a disease would be called the *true positive rate* and the amount of false alarms per the amount of healthy people would be the *false positive rate*. A basic trade-off usually exists between these two quantities simply through setting the threshold for detection. The lower the threshold is set, the more diseases are detected, but also more false alarms happen.

This trade-off is often visualised as a *receiver operating characteristic* curve, which plots the true positive rate as a function of the false positive rate. Figure 13 gives an example of such a plot. Typically a point is searched for where either a given detection rate is satisfied or an adequately low false positive rate occurs.

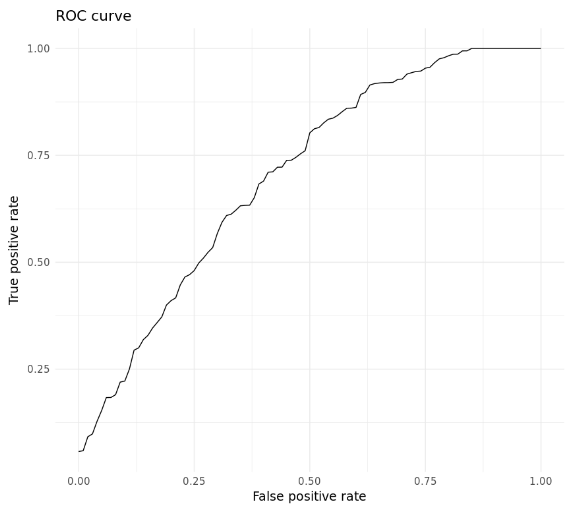


Figure 13: *An example of a receiver operating characteristic curve.*

2.7.4 Processing time

It is important to note that processing time is a significant resource in any automatic speech recognition system. Basically any allowed increase in processing time affords using a larger model and a more thorough search in the hypothesis space. Many applications have the constraint that they must respond to user speech input in a reasonable time and optimising for recognition speed is one aspect in system performance. In other applications, it simply becomes unfeasible to process large amounts of data if the system is too slow.

The measure usually reported for a speech recognition system is the *real-time factor*. It is the ratio of processing time to the utterance length, i.e. a speech recognition system that computes the output in 0.3 seconds for an utterance of 3 seconds has a real-time factor of 0.1.

2.8 Use of speech recognition in this thesis

In conclusion it can be said a speech recognition system is very heterogeneous. The path from an audio signal to a transcription begins with feature extraction. Then acoustic model likelihoods are computed for each feature vector. The decoder then searches the lexicon and the language model for the most likely word sequence. All this has to be done fast. The training of the acoustic and language models though might have taken days, because the training sets need to be large.

The bulk of the work in this thesis is just language modeling, but for experiments all the other parts are needed too. A Swedish acoustic model and lexicon was provided by Peter Smit, and a standard recipe was used to train an English model. Acoustic models are characterised by the data used for training, so they are briefly described here. Lastly, we look at the speech recognition toolkit used in this thesis, *Kaldi*.

2.8.1 Wall Street Journal

The *Wall Street Journal* corpus is one of the standardised datasets that are used to compare speech recognition models. It has approximately 73 hours of speech data for training from 245 speakers. The speech data is clean and the speech is read from randomly selected snippets of the Wall Street Journal. The English has a general American English accent. [46]. Wall Street Journal's domain is business news from a North American perspective.

A standard recipe included with the Kaldi toolkit was used to train Gaussian mixture model-hidden Markov models for this thesis. In detail, it is *speaker adaptively trained* on feature vector windows, which are compressed through *linear discriminant analysis* and a *maximum likelihood linear transformation*. The word error rate on the test set is 10.88% using an N-gram language model provided with the corpus. There are in fact two test sets in the Wall Street Journal corpus standard set division, in this thesis the *Eval93*-set is used throughout.

2.8.2 Swedish data

Three corpora of Swedish data are used. The first is from the pilot test data collection efforts in project DigiTala: approximately 800 Finnish high school students around the country, each recording about 10 minutes of speech. This is in-domain data. It will be referred to as the DigiTala corpus. The second corpus is the Talko corpus[2]. It consists of 40 hours of fluent Swedish in a comprehensive variety of dialects spoken in Finland. The third corpus is the Swedish Språkbanken corpus[3]. It has over 400 hours of read Swedish speech, but the dialects are from Sweden.

A deep neural network acoustic model was trained from the combination of these three corpora. In detail, it is a *time-delay neural network* trained using the lattice-free maximum mutual information objective function. The word error rate measured on read texts from 20 speakers in the DigiTala corpus, using a 400 000 word N-gram language model from the Språkbanken corpus, is on average 39.30%. This large value is explained by the large variance of the results. Figure 14 shows a boxplot of the word error rates for individual speakers. Some outliers have word error rates exceeding 100%. Without outliers, the average word error rate drops to 29.38%. However this high word error rates may not reflect the true performance of the model because the test data was not well annotated when performing the tests; it is reported here only referentially.

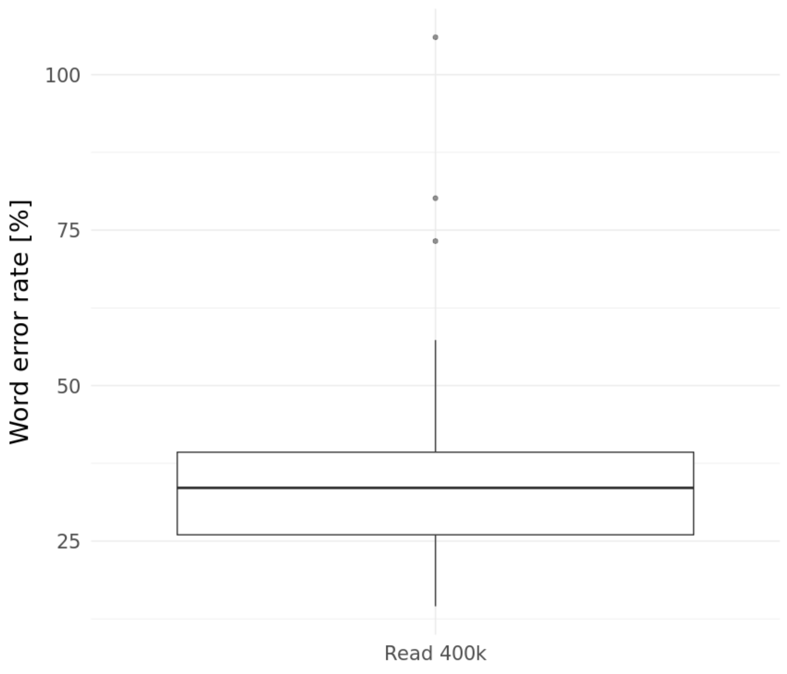


Figure 14: *A boxplot of the word error rates with the Swedish deep neural network acoustic model. Some individual speakers have huge error rates, so they should be excluded as outliers.*

[2]http://www.sls.fi/en/talko_en
[3]http://www.nb.no/Tilbud/Forske/Spraakbanken

In the process of training the deep neural network, a Gaussian mixture model was also created. The deep neural network has better accuracy, and should be used for the actual decoding. However its performance in temporal alignment may be worse, as discussed in section 2.4.5. Therefore the temporal segmentation outputs from the preprocessing system implemented in this thesis should be computed by realigning the transcript produced by the deep neural network model with the Gaussian mixture model.

2.8.3 Kaldi

The Kaldi toolkit is a modern open source speech recognition toolkit aimed at speech recognition research [47]. It provides the main building blocks for automatic speech recognition systems and a wide variety of algorithms and recipes. There are many other toolkits that provide similar functionality, like HTK, Sphinx, and even Aalto university's AaltoASR. Kaldi is chosen for this thesis because it has a modern, extensible design and great performance. Kaldi also has first-class support for finite state transducers, which turn out to be useful for implementing language models for recognising read speech, and which we will introduce in the next section.

3 Finite state models

Computer science is the study of computation. There are many ways of formally representing computation, each with different purposes. Some purposes are mostly theoretical, answering a question like: "What can be computed?". In that case the constraints of finite memory and processing time may not be of interest and a researcher may use a model like the *Turing machine.*[48] Our purposes are practical, but it turns out we still benefit from using a formalism which fits our problems. The *finite state machine* model can represent the hidden Markov models of context dependent triphones, the lexicon and many types of language models, including N-gram models. Furthermore it can represent the cascaded combination of these models as used by the decoder and it can do this in an efficient manner.[49]

Turing machines are not physically implementable, finite state machines are. However it may be unhelpful to think about finite state machines as tangible boxes turned on with a big red power switch. You don't feed the inputs in on a conveyor belt and wait to see what comes out the other end. Instead, the finite state machine framework allows us to design in our minds such a conveyor-belt-machine and then get the mathematical representation of that machine. If the question is: "What can be computed?", part of the answer is at least, "the output of a finite state machine".

In this section we will first look at the parts which make up the different types of finite state machines. Then we will look at the operations that can be performed on and with them. All the while we will be building a toy example part by part alongside. At the end of the section we will show how finite state machines represent the different components of the decoding cascade in automatic speech recognition.

3.1 Finite state machines

Finite state machines have, as the name suggests, a finite set of internal states. They represent different static configurations or positions. One of these states must be the *initial state*, the state at which machine starts, and a possibly empty subset of them are *final states*, states where the machine may stop. In mathematical notation the set of states is denoted with S, the initial state s_0 and the set of final states F.

The example we are building is a puzzle game where the player is in a strange mansion and the objective is to get out. The reader may be familiar with text-only puzzle computer games; the finite state machine we are building represents the game's state for the internal code. Each state is a location in the puzzle and figure 15 shows the layout.

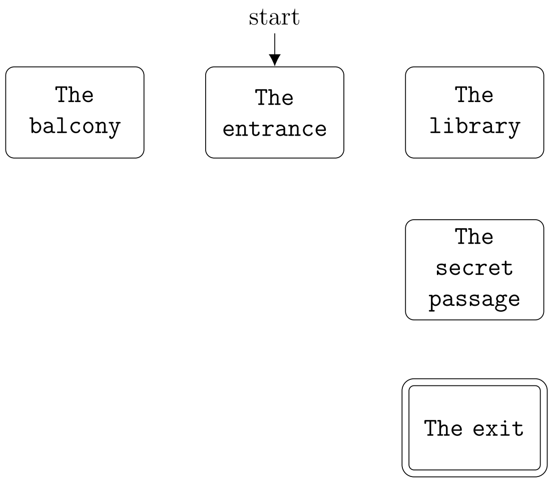


Figure 15: *The mansion layout. The exit is a final state, which are drawn with double borders.*

A finite state machine also has a set of *input symbols*, noted with Σ. At each step a finite state machine reads one input symbol. The next state of the machine is decided entirely on the current state and the read input. The finite state machine has a set of *transitions* E, which connect two states $q_i, q_k \in S$ via an input label $x \in \Sigma$, i.e. $E \subseteq S \times \Sigma \times S$. In the puzzle game, the inputs are actions the player can take, such as `go west`. The player chooses an action, inputs it to the finite state machine and the finite state machine computes the next state. Figure 16 shows the mansion the with actions the player can take in each room and where each action takes the player. The puzzle can now be solved.

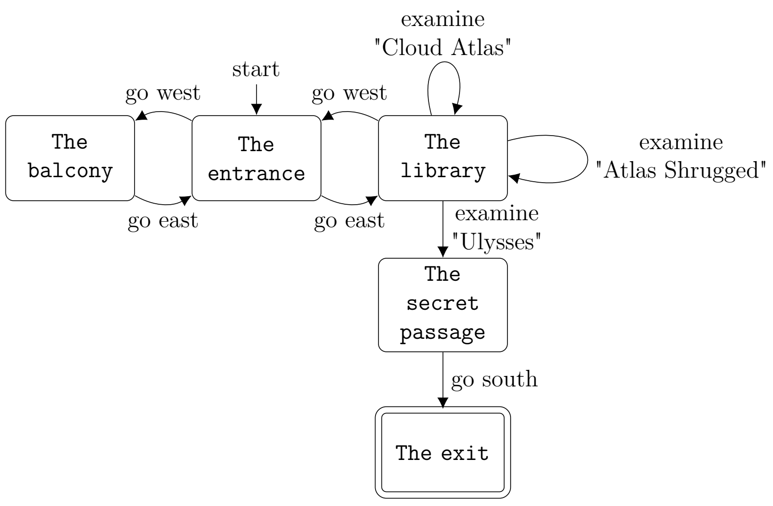


Figure 16: *The mansion with actions available in each room. The transitions are represented as arcs between the different states.The twist in this puzzle is that the player must examine right book in the library to be taken into the secret passage.*

At each state in the puzzle only some actions can be taken. The player cannot try to examine "Cloud Atlas" in the balcony, since the books are stored in the library. It is typical that transitions are only defined for a subset of all state-via-input-symbol-to-state combinations and other combinations will not be accepted. An input symbol sequence that uses undefined transitions is rejected. Similarly, sequences ending in non-final states are rejected. In many cases this property of accepting certain sequences and rejecting others is the reason to construct the machine. This type of finite state machine is called a *finite state acceptor*, and it accepts a *regular language* of the input symbols[48].

3.2 Finite state transducers

The output of a finite state acceptor is just an acceptance verdict. To build more complicated output, a finite state machine can write an *output symbol* at each transition. In this case the machine is called a *finite state transducer*, since it converts, or transduces, the input sequence into an output sequence.

In addition to the finite state machine components the finite state transducer has a set of output symbols Ω, which may be completely different than the input set. In the puzzle example the game lacks narrative. To fix that we will add some interesting text to each transition to be displayed to the player on each action. They will take some page space, so we add text identifiers: each will correspond to one output text. Figure 17 shows the puzzle model with output labels added. Table 5 lists the actual output texts for the reader's entertainment.

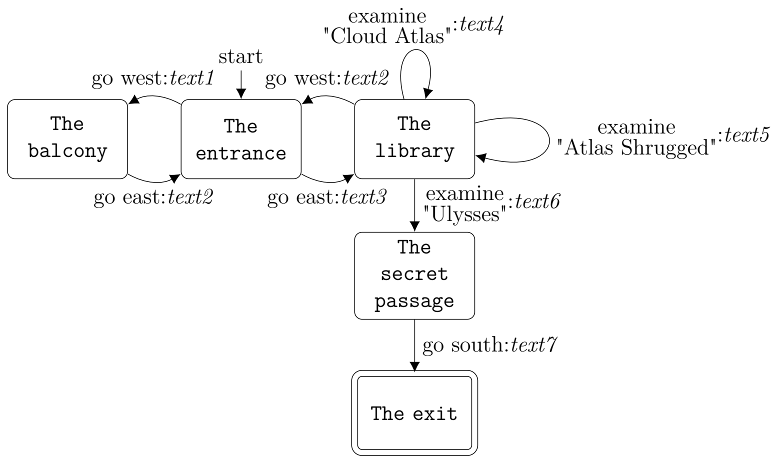


Figure 17: *The mansion with output labels. This finite state model now transduces a sequence of player actions into a sequence of text identifiers.*

There are still two properties of finite state transducers that we will introduce: empty labels and weights. Then we will look at some powerful, generic operations on the transducers. The finite state models shown so far have been useful for representing things in an abstract, yet code-implementable way. However, weighted finite state transducers derive most of their power from some generic mathematical operations which allow them to be combined and made as compact as possible.

3.2.1 ϵ - empty label

So far all transitions have consumed an input symbol and output another symbol. Epsilon, ϵ, is a special label which does not consume or output a symbol.

When placed on the output side, it allows consuming more than one input symbol per one output symbol. This is a typical situation: different domains may have different occurrence intervals. One symbol of one domain may consist of many symbols of another domain, for example words and letters. In the puzzle example, we will get word sequences as input from a command line interface, or maybe even a speech recognition system. We will have to build the input symbols from the word tokens. This is an archetypal task for a finite state transducer and figure 18 shows the transducer that solves the task. Epsilons are used on the outputs to map from many single word input symbols to multi-word output symbols. This transducer has some flaws, so we will come back and improve it when introducing some of the operations on transducers.

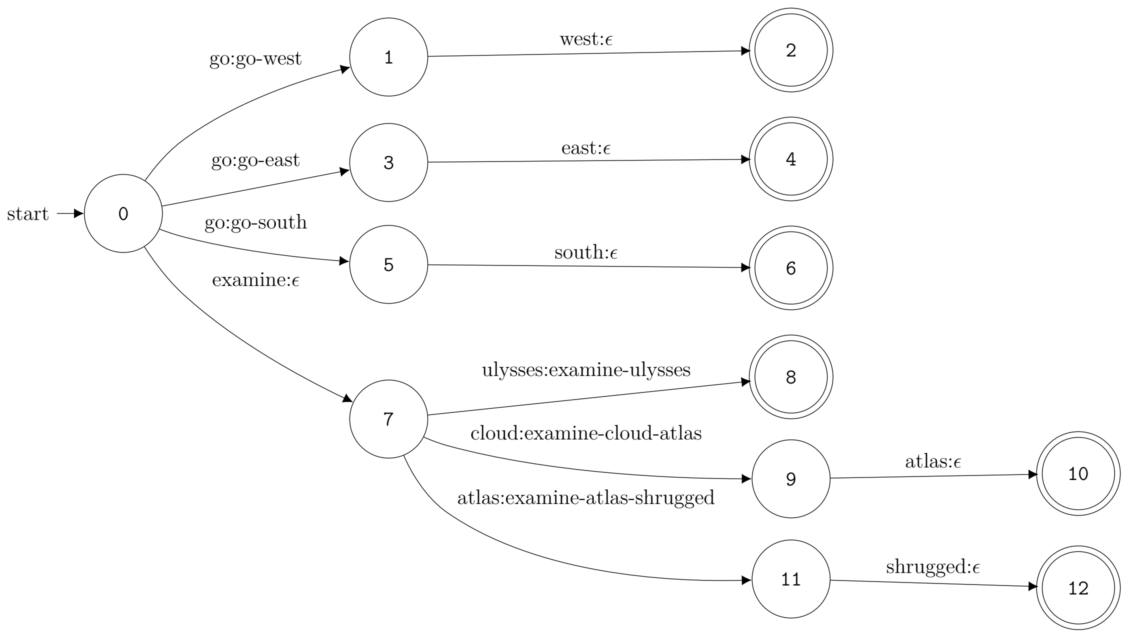


Figure 18: *Word token to puzzle input transducer. The epsilons on the output side allow consuming multiple input symbols per output symbol. Notice that now the states themselves do not have a name. It could be said the states correspond to positions in utterances, such as having just said the word* ***`examine`****. Now that we are working with more traditional transducers, we switch to circle shapes. This is the typical notation for finite state transducers in speech and language technologies.*

An ϵ input label is a more complicated matter. In many applications, each transition is thought to correspond to one time step or similar unit[48]. The empty input label represents an instant traversal; it should take zero steps. Like output side ϵ labels, it can be used to transduce between domains of different occurrence intervals, such as from words to letters.

3.2.2 Weighted models

Transitions in finite state models can be augmented with a weight. The weights encode the cost, duration, probability or other similar measure of that transition. That quantity is accumulated from all the transitions on an input sequence takes to calculate the total weight of the transduction. Additionally, the final states have a weight, which is added to the weight of an accepted transduction.[49] The final weights form a set ρ. In the puzzle example, we can add weights to the wrong action choices. This way, a score can be computed for each played game.

The useful generic properties and operations on finite state transducers have weighted versions. These versions require that the weights form a *semiring*. A semiring is like a limited algebra, which does not require all the properties we might be used to in basic arithmetic over real numbers. It is defined with two binary operations over a value set $\mathbb{K}$: an addition operation $\oplus$ and a multiplication operation $\otimes$. The addition operation is associative, commutative and has *identity* $\bar{0}$. The identity value preserves the value of the other operand: $a \oplus \bar{0} = a$. The multiplication operation is associative, distributes with respect to $\oplus$ and has an identity value $\bar{1}$. The multiplication operation also has as *annihilator* the identity of addition, $\bar{0}$. Any operation with its annihilator has the annihilator as result: $a \otimes \bar{0} = \bar{0}$. [50] The big difference over the usual *ring* algebra is that a semiring does not require that each value has an *additive inverse*, i.e. a value which yields $\bar{0}$ in addition. For example, arithmetic with natural numbers $\mathbb{N}$ is a semiring, but not a ring, since values in $\mathbb{N}$ have no negative counter parts to sum to zero. Table 3 lists the semirings we will use in automatic speech recognition.

SEMIRING	$\mathbb{K}$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Probability	$\{x \in \mathbb{R}_+ \vert 0 \leq x \leq 1\}$	$+$	$\cdot$	0	1
Inverse log probability	$\mathbb{R}_+ \cup \{\infty\}$	$\oplus_{log}$	$+$	∞	0
Tropical	$\mathbb{R}_+ \cup \{\infty\}$	*min*	$+$	∞	0

Table 3: *Semirings used in automatic speech recognition.* $a \oplus_{log} b = -log(e^{-a} + e^{-b})$

Now that all the properties of weighted finite state transducers have been presented, a final version of the puzzle game example is constructed. It has a meaningful interpretation of all the different properties. Table 4 has the puzzle in a text format, so that the reader is reminded of the many aspects which make up a weighted finite state transducer.

3.3 Operations with weighted finite state transducers

In this section we will introduce algorithms which operate on weighted finite state transducers. Most algorithms take just one transducer and return a transducer that fulfils the same relations between sequences with the same total weight as the original, but has changed in some beneficial way, like being smaller in size. Other algorithms take two transducers and compare or construct something from them.

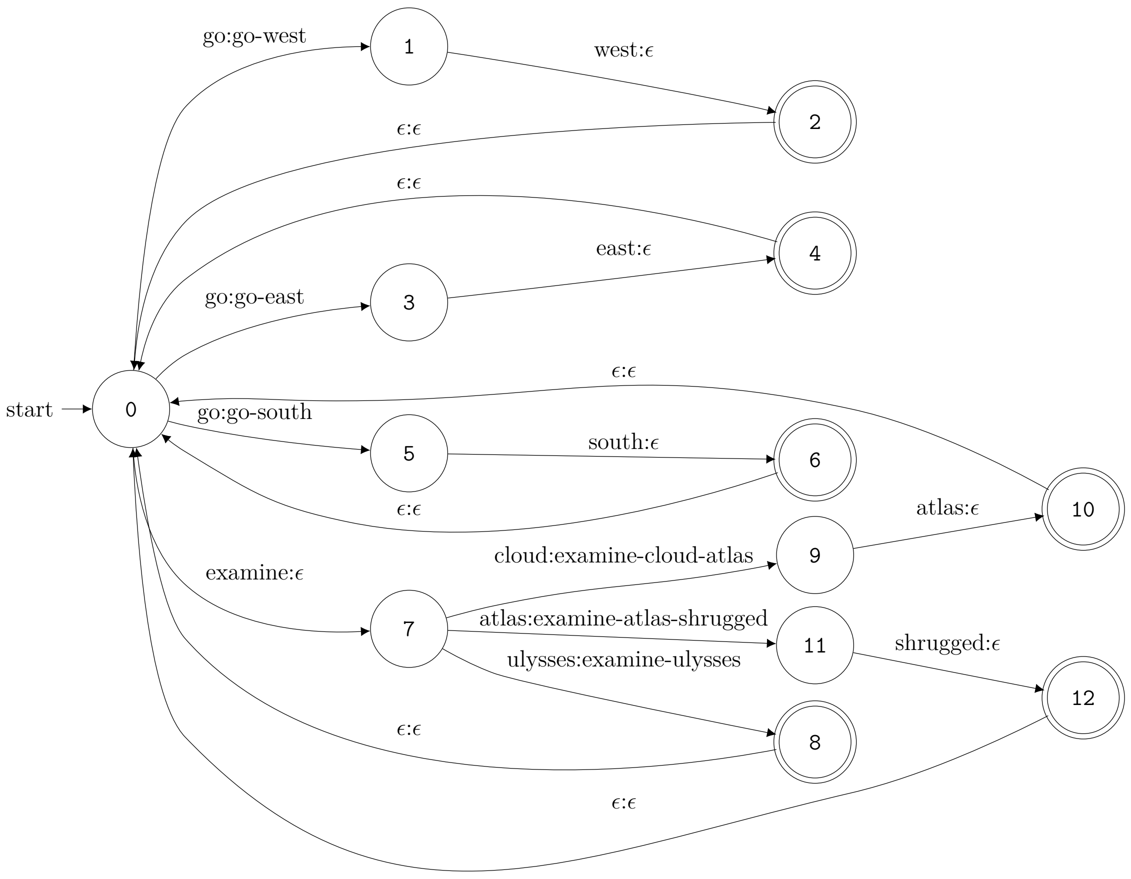


Figure 19: *The Kleene plus closure of the word the input transducer. It can now transduce multiple commands in sequence.*

The algorithms presented here are the primary reason why expressing a task in this unified framework is beneficial.

We will look in detail at three algorithms most important for our work: *composition*, *determinisation*, and *minimisation*. Other algorithms are described more briefly.

3.3.1 Basic transducer manipulation

Weighted finite state transducers can be combined in series with *concatenation* and in parallel with *union* and repeated zero or more times with the *Kleene star* operation. The *Kleene plus* operation allows the transducer to repeat once or more.

The word input transducer of figure 18 can only transduce one sequence at a time. The first improvement is to allow it to transduce many commands in sequence. This is done with the Kleene plus operation, and figure 19 shows the result. Essentially, it adds an ϵ:ϵ transition from the end of each phrase back to the start.

Inversion creates a reverse transducer which transduces from the accepting states toward the initial state. *Projection* can be used to obtain an acceptor by keeping either the input or the output labels. Often this is used to read a result after some transducer manipulation by projecting and then reading the sequence of labels.

3.3.2 Composition

As will be demonstrated in section 3.4, particularly in automatic speech recognition the power of finite state transducers comes from creating processing chains with other finite state models. Like an electrical transducer, a finite state transducer can act as a link and feed another model by having its output symbols serve as the input symbols of the second. When combined in this way, the two models form a new transducer which transduces from the input symbols of the first transducer to the output symbols of the second transducer. This operation is called composition.

This algorithm can be used to create complex relations from simple transducers[49]. For example a Levenshtein distance measuring (see section 2.7.1) transducer can be created with composition[4]. The transducers from figures 19 and 17 are now composed to transduce straight from text input into output text identifiers. Figure 20 shows the result. It's easy to see now that the Kleene plus closure of the original word input transducer was necessary for this composition to give a non-null result, because `The exit` is not reachable with a single command.

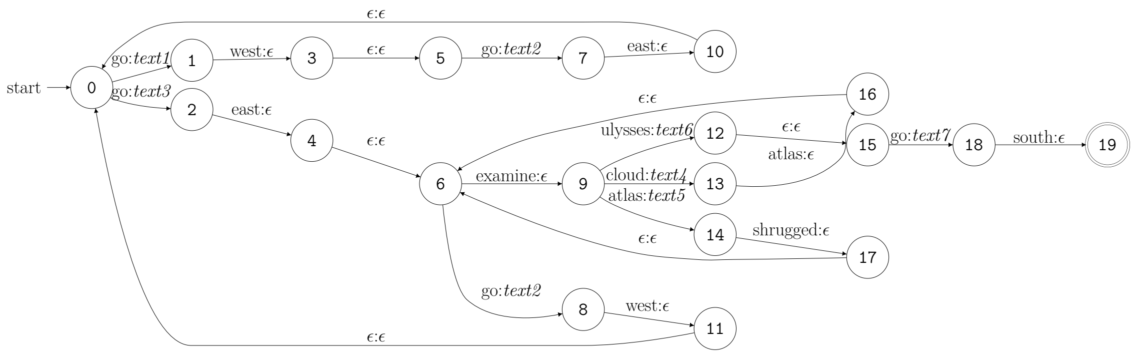


Figure 20: *The result of composing the transducer of figure 19 on the left with the transducer from figure 17. The state are again given arbitrary number identifiers because of the implementation of the algorithm. Note that some of the states can be traced back to those mansion rooms: 0 is `The entrance`, 6 is `The library`. Some correspond to states where the transducer awaits input.*

The weighted version of composition requires some care regarding ϵ labels [49]. The total weight through a path in the composed transducer should equal the accumulation of weights on corresponding paths in the two original transducers.

3.3.3 Determinisation

A *deterministic* transducer has at each state at most one transition leaving with any input label and no ϵ input labels. The deterministic transducer has at most one successful path for any given sequence of inputs, which means it uses the least

[4]This example is from, and is elaborated at: http://www.openfst.org/twiki/bin/view/FST/FstExamples

amount of steps of all transducers to process an input sequence. Only one path needs to be followed.

The weighted determinisation algorithm takes a non-deterministic transducer and returns a new deterministic transducer which transduces any input sequence to the same output as the original transducer with the same weight. However, the algorithm has an exponential complexity in the worst case and does halt for all inputs.[49] One example are ambiguous transducers, where a given input might be transduced to different output sequences or weight or weights. In some determinisation implementations the epsilon input labels are not removed, and are instead treated like a normal input label.

The composed word input to text identifier output transducer from figure 20 is now determinised. After determinisation the `go west/east/south` paths share a common transition which consumes `go`. Figure 21. The number of states is reduced from 20 to 14 and there are no ϵ input labels remaining. Particularly the $\epsilon : \epsilon$ transitions, which do not transduce anything, added by the Kleene plus closure are simply removed.

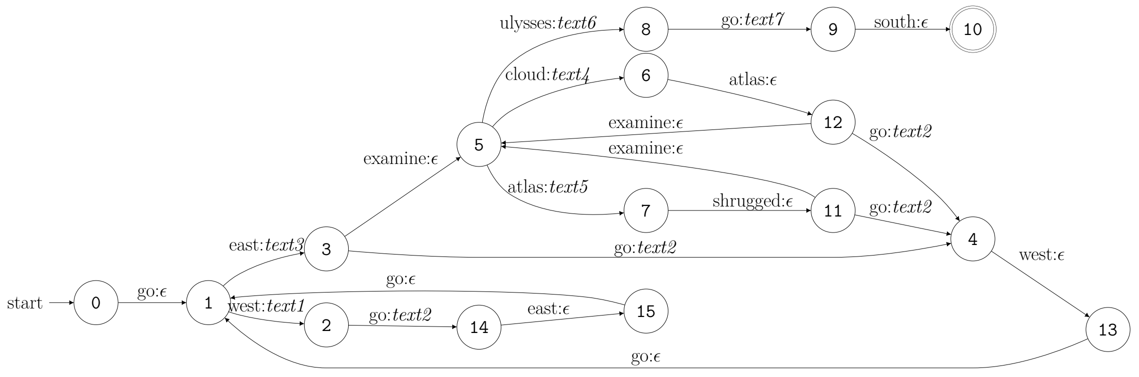


Figure 21: *The result of determinising the transducer from figure 20. Some states are merged to delete all but transition with a given input label. Some states inherit the transitions from the states to which they had $\epsilon : \epsilon$ transitions.*

3.3.4 Minimisation

A *minimal* transducer is one that uses the smallest possible amount of states to describe its input-output relationship. This is obviously beneficial in terms of memory consumption. The minimisation algorithm takes a deterministic transducer and returns a minimal deterministic transducer which transduces equally to the original.

As part of the weighted minimisation algorithm, the weights are redistributed along the paths so that weight is accumulated as close to the initial states as possible. The total weight for any path stays the same. This is both necessary for the minimisation algorithm and beneficial in any approach that uses a pruned search (like described in section 2.3.1), as unpromising paths are detected as early as possible.[49] The operation is called *weight pushing*.

A final, minimised version of the determinised word input to text identifier output transducer is shown in figure 22. This example did not use weights, so no weight pushing is needed. The number of states is further reduced from 14 to 12.

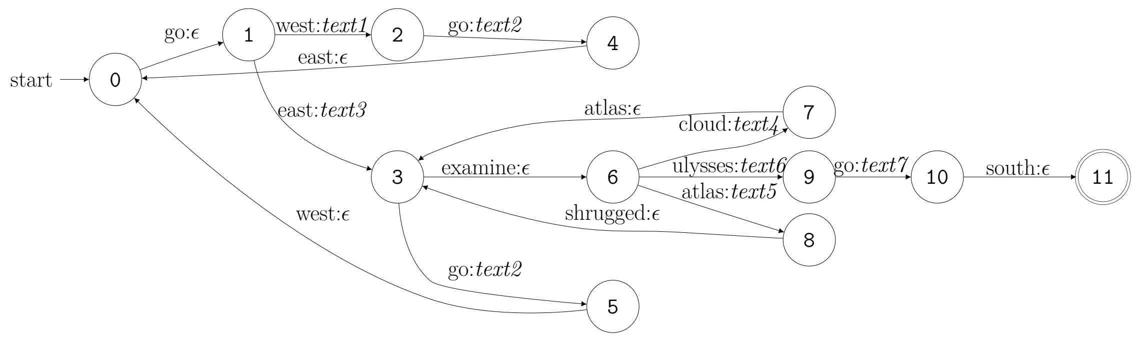


Figure 22: *The result of minimising the transducer from figure 21. Essentially, the states **11** and **12** in figure 21 are merged into state 3, which corresponds to **The library** in the original puzzle layout.*

3.4 Speech recognition decoding graphs

The decoder of an automatic speech recognition system matches acoustic observation sequences to word sequences. Figure 12 presented the search space of the decoder as a grid of acoustic classes and time steps.

The decoder searches for paths through acoustic classes which form words. The language model defines the search space: which words should be searched. The lexicon defines the sequences of phones that form a word. The corresponding sequence of context dependent phones can be derived from a sequence of context independent phones. The hidden Markov models of each context dependent phone give the corresponding sequence of acoustic classes and the probabilities of moving from one acoustic class to the next.

Each part in this four layer structure can be represented by a weighted finite state transducer. The language model is represented by transducer G, for grammar. The lexicon is represented by transducer L, for lexicon. The hidden Markov models will be represented by transducer H, for hidden Markov model. And solving the phonetic context dependencies is transducer C, for context dependency. Together they form a structure of $H \circ C \circ L \circ G$, where $\circ$ is the composition operation. [49]

In other words, weighted finite state transducers can be used to create a graph of the whole search space of a decoder. The operations described in section 3.3 allow us to make the graph, and thus the search space, very compact. Next we will briefly show how each layer is created, focusing on the grammar and lexicon transducers. Then we describe how the layers are composed and how the resulting graph is optimised.

3.4.1 Creating the transducer layers

The lexicon is a transducer from phones to words. If a word has multiple pronunciations, the probability of each pronunciation may be given as a weight. The lexicon is not determinisable in the general case, for example homophones cannot be determinised. This problem is obvious in figure 23. It can be made determinisable by adding special *disambiguation symbols* at the ends of words. The last epsilon input label is turned in to `#1`, or `#2...k` for k different homophonous words.

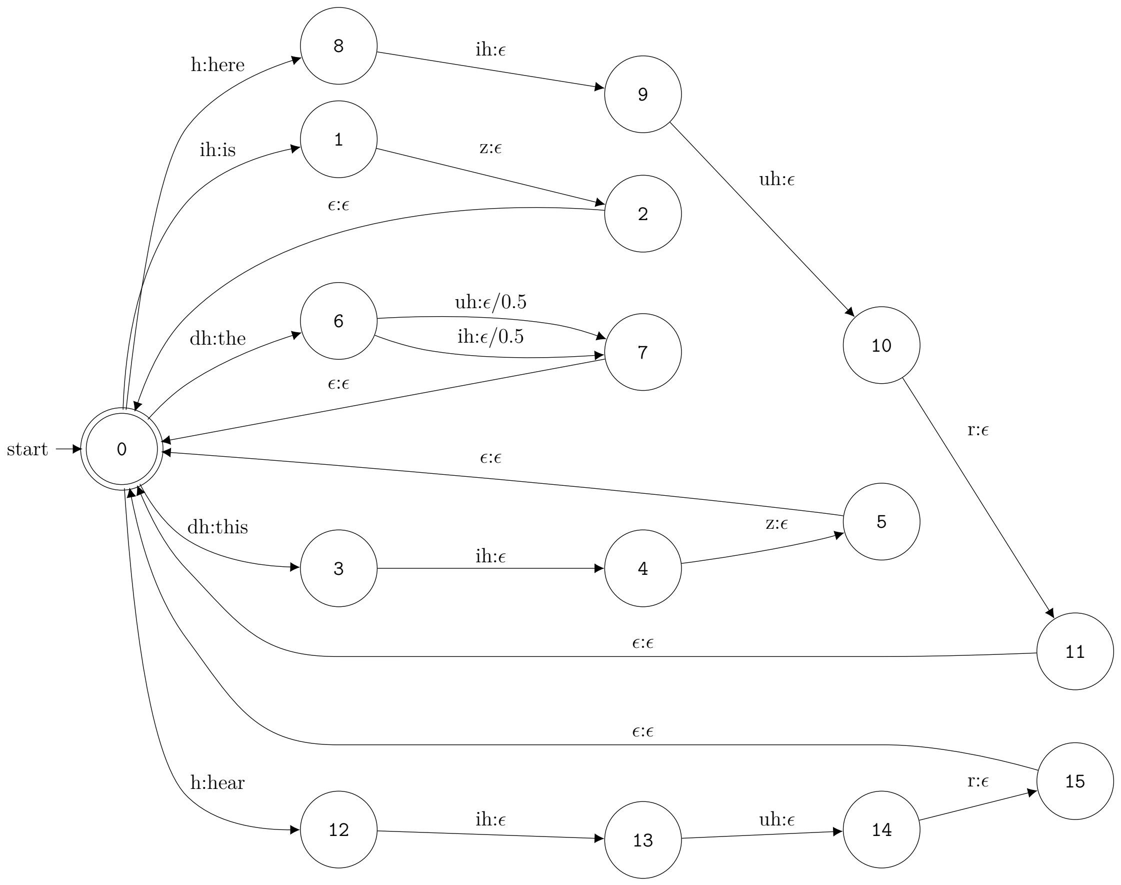


Figure 23: *A lexicon transducer. It maps phone sequences to words. The word* **`the`** *has two pronunciations. The words* **`hear`** *and* **`here`** *are homophones.*

A traditional language model would be a weighted finite state acceptor, as it simply assigns weights to word sequences, but for practical reasons we represent it as an identity transducer which maps words to themselves. This way we get the actual words as output, and not just their weight. Each state in an N-gram language model transducer represents some history of words. The transitions leaving that state have as weight the conditional probability of seeing the input label word after the history associated with that state. This is of course the N-gram probability from section 2.5.1. There are also backoff states for each history: the transition to this state is an $\epsilon : \epsilon$ transition and represents the decision to use a lower level model. The transition weight is the probability mass reserved by discounting. The backoff state has transitions of the labels and the weights of the model being backed off to. In fact, if the model being backed off to is a lower level N-gram, then the backoff transition simply leads to a state of shorter history. Figure 24 shows an example of an N-gram language model transducer. Following the arcs in the image we find that:

- There is no path from state `3`, i.e. the history of `this`, with the label `the`. The sequence `this the` indeed seems unlikely. It can still be decoded by following the backoff arc.
- There is a direct path for `is this`: `2 3`, but also a path through the backoff state: `2 1 3`. This non-determinism is problematic, especially since the paths give different weights. It's a consequence of using the unigram zero word history state as the global backoff state. This too can be solved by replacing the epsilon label on the backoff arc with a disambiguation symbol (we will notate it as `#0`).

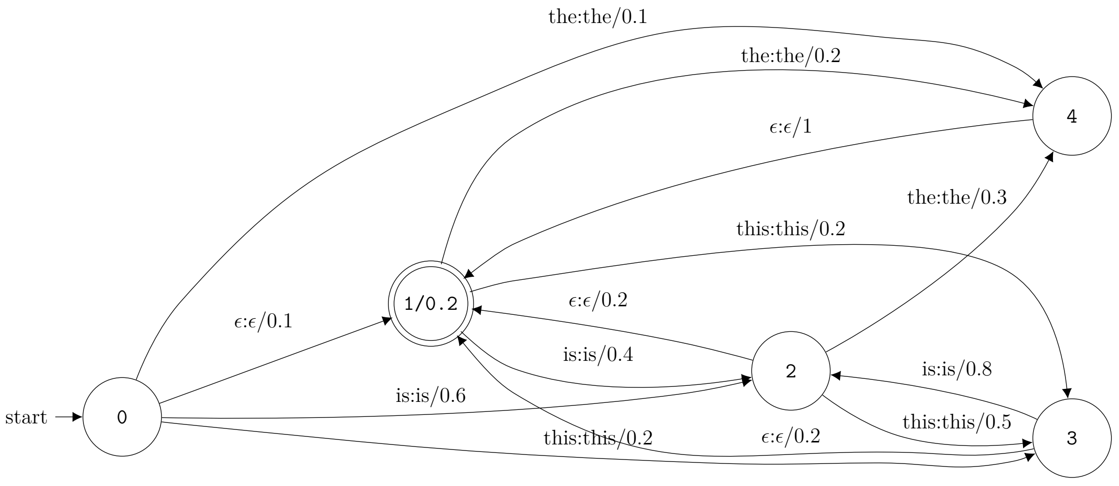


Figure 24: *A simple bigram language model transducer. It maps words to words, the weights (in the probability semiring) give the language model probabilities. State `0` has the special history of sentence beginning. State `1` is the backoff state. The language model has three words: `is`, `this`, and `the`.*

It seems clear that a phone sequence can be transduced from corresponding context dependent phones. Surely the sequence `sil-ih+z ih-z+dh dh-ih+z ih-z+sil` gives `ih z dh ih z`. The structure of the context dependency is slightly more complicated than initially seems, because the context phones from the future are not known in advance. The deterministic versions of the C transducer take this into account.

The Markov model structure of figure 5 already looks quite like the transducers of this section, and indeed the transitions are simply augmented with input and output labels to get a weighted finite state transducer. The output label on the transition into a hidden Markov model should be the context dependent phone. As the graph we are constructing will represent the decoder search space, the input labels will just be identifier symbols for the probability distributions of the acoustic model. These are then used by the decoder to follow search paths through different acoustic classes. However, in many implementations the hidden Markov model structure is implemented by the decoder[49] and other special arrangements are possible[37].

3.4.2 Optimising the search graph

The composition of $H \circ C \circ L \circ G$ is usually started with $L \circ G$, then $C \circ (L \circ G)$ is constructed and finally the hidden Markov model structure is added with $H \circ (C \circ (L \circ G))$. This structure lends itself to optimisation and correct handling of the disambiguation symbols introduced with G and L.

It should not be a huge twist for the reader that the graph is optimised primarily with the determinisation and minimisation algorithms. They can be applied at many points during the composition. One example could be $min(det(H \circ (C \circ min(det(L \circ G)))))$, where $min()$ is the minimisation and $det()$ the determinisation operation. However, there are many different graph creation recipes. There is some interplay between decoder implementation and graph creation. Some finite state transducer libraries perform much faster when the transitions in the actual data structure are sorted by some criterion, such as alphabetically for input labels. Generally, optimised finite state transducer search graphs give very fast decoding performance while retaining a size of the same order of magnitude as the language model[49].

Static graphs give a speed boost in decoding, since a lot of computation is done beforehand in the graph creation process. However, this may not work together well with dynamic models, such as a language model which is augmented by the results of an online database search. For these applications, the search graph may use parts that are loaded as needed, or *lazily*. Of course the drawback is that these parts cannot be optimised globally. [50]

T_{puzzle}

- S, the set of states
  - `The entrance`
  - `The balcony`
  - `The library`
  - `The secret passage`
  - `The exit`
- s_0, the initial state
  - `The entrance`
- F, the set of final states
  - `The exit`
- Σ, the set of input symbols
  - go west
  - go east
  - go south
  - examine "Ulysses"
  - examine "Cloud Atlas"
  - examine "Atlas Shrugged"
- Ω, the set of output symbols
  - *text1*
  - *text2*
  - *text3*
  - *text4*
  - *text5*
  - *text6*
  - *text7*
- E, the set of transitions
  - `The entrance` to `The balcony`, go west:*text1*/1.0
  - `The balcony` to `The entrance`, go east:*text2*/0.0
  - `The entrance` to `The library`, go east:*text3*/0.0
  - `The library` to `The entrance`, go west:*text2*/1.0
  - `The library` to `The library`, examine "Cloud Atlas":*text4*/1.0
  - `The library` to `The library`, examine "Atlas Shrugged":*text5*/1.0
  - `The library` to `The secret passage`, examine "Ulysses":*text6*/0.0
  - `The secret passage` to `The exit`, go south:*text7*/0.0
- ρ, the set of final weights
  - The exit/0.0

Table 4: *The final version of the puzzle weighted finite state transducer. It defines a septuple* $T_{puzzle} = (S, s_0, F, \Sigma, \Omega, E, \rho)$. *The text identifiers of* Ω *refer to flavor texts which are provided for the reader's entertainment in table 5.*

text1	You open the small balcony door and step outside. The night air is surprisingly warm and humid. You can hear someone in the distance. You peek down at the railing but the fall is too high.
text2	You return to the entrance. The marble floor announces your every step with a sharp clack. The bare and dimly lit room makes you shudder.
text3	You enter the quiet library. There is an intimidating shelf of high-brow literature and an inviting recliner with just the right light for reading.
text4	You grasp Cloud Atlas and skim the back cover. The concept seems a little too clever but you've heard good things.
text5	You hold Atlas Shrugged in your hands like a mirror. You imagine a reflection: an older, colder you with an unhealthy sense of entitlement.
text6	You begin to grab Ulysses and suddenly the world around you starts carouseling. The bookshelf revolves and you are taken into a hidden passageway. You take the book with you, swearing to finally find the time to actually read it.
text7	At the end of the passage you find a plain door which takes you outside. You have solved the puzzle.

Table 5: *The texts shown to the player in the puzzle example game. Each text corresponds to a text identifier.*

4 Modeling reading aloud

At first glance the task of recognising read speech seems not a decoding problem, but a problem of aligning the words of the text temporally. However oral reading is not simply one exact process of pronouncing each successive word after another. Even experienced readers deviate from the text from time to time. For example they may repeat a word. This alone makes a simple alignment ill-founded.

In fact the reading process has been studied in *psycholinguistics*, the study of relationships between linguistic behaviour and psychological phenomena, both from the view point of acquisition and use. As the subjects in this thesis are high school students, it is reasonable to assume they have acquired a mature level of reading ability, and the interest is in what psycholinguistics can tell us about the use of the reading ability. Kenneth S. Goodman developed a method for dissecting the reading process called *miscue analysis*. Goodman says[5]:

> "A miscue is defined as an observed response that does not match what the person listening to the reading expects to hear."

The term miscue is used to avoid a value judgment implied by a term like error. Deviations from the text may not be harmful or destructive. Self-correction is a natural part of reading. An efficient reader's goal is not word-for-word accuracy per se, but correct conveyal of the meaning. In contrast, a struggling reader may resort to treating the text as "grammatical nonsense", which may retain a decent level of letter-wise accuracy, but involve a loss of meaning. The deviations from the expected arise from a complex, multifaceted process, where only one part is scanning and decoding the letters into phonetic forms. Semantic, phonological, graphic and syntactic information are all used by the reader. [51] I believe this suggests two things. Firstly inferring a qualitative analysis of the reader's process is a hard problem. Secondly it is not fair to expect the reader to provide a word-for-word rendition of the text and especially the reader should not automatically be punished for it. The deviation from the text may in hindsight be a positive indication of the reader's ability, yet this hindsight will be impossible if automatic speech recognition cannot capture the deviation.

Fortunately the engineering task is to accurately predict the actual outcome of reading rather than make accurate inferences of how the outcome came to be. Thus what works counts, and we may test some conventional speech recognition methods. Either the methods solve the task well and may be used or they provide a valuable adequate baseline. As stated in the beginning of this section, the task is close to alignment, so forced alignment might be a good starting point. N-gram models are a classic language model and they can decode word sequences which diverge from the prompt. First these two ideas are described in the next subsections. Then section 5 is dedicated to exploring the idea of task-based grammars with explicit paths for miscues.

[5]Unfortunately I could not find the original source, but the quote is attributed to Goodman here: `http://www.ucs.louisiana.edu/~jsd6498/damico/miscue.html`

As a side note, Goodman also lists and categorises miscues comprehensively into the *Goodman taxonomy of miscues* in [52]. It could provide ideas of what to look for, but detecting miscues is not trivial, and the list cannot simply be written as an algorithm. Thus miscue types should be look at on a case-by-case basis.

4.1 Forced alignment

Forced alignment of the prompt text on to the utterance forces the words to appear in the text order. The forced alignment procedure may fail, producing no output. The prompt is expressed as a sequence of acoustic classes and the feature vectors are searched for the most likely points where the acoustic classes transitioned to the next one.

A danger in forced alignment is that it strives to always produce some kind of alignment. This may result in alignments that are nonsensical. A miscue may throw the forced alignment procedure off course, as for example a missing word in utterance means the text is aligned into silence or on top of some other word. Though silence typically may be handled well, since it is explicitly modeled as an acoustic class, the phone classes are somewhat confusable. It is difficult to automatically detect these cases and a human test is expensive.

In the Kaldi toolkit, forced alignment simply uses finite state acceptors which only accept the prompt word sequence. Figure 25 shows an example of such an acceptor.

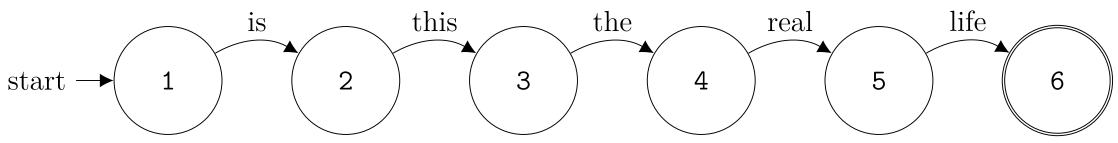


Figure 25: *A finite state acceptor for the phrase* `Is this the real life?`*. It is simply a chain from one word to another.*

4.2 Biased N-gram models

Included in the Kaldi toolkit are a set of cleanup scripts for corpora that have not been manually reviewed or expert transcribed. They are similar to the procedures documented in [53], but the original scripts in that work are not open source[6]. It is thus difficult to credit the work correctly. As part of the cleanup procedure the audio is divided into small segments and an N-gram language model is created from the text to which the audio is assumed to correspond.

As stated in section 2.5.1, N-gram models are normally trained on a large text corpus. To benefit from the strong prior knowledge on the utterance, an N-gram can be trained on just the prompt text. Of course this way almost all word histories are seen just once, and thus only the word that followed in the text is given any

[6]As reported by Kaldi author Daniel Povey here: `https://groups.google.com/forum/#!topic/kaldi-help/K9ANiNCNV1M`

probability mass. The application of N-gram smoothing distributes some of this mass to other words in the prompt. Thus the N-gram remains biased towards the original prompt but allows other orderings, repetitions and omissions. The scripts implement Kneser-Ney smoothing.

These biased N-gram models have some parameters to consider. The amount of discounting used gives a convenient control of the level of bias. The effect of the length of history, i.e. the N-degree, is difficult to reason about. A longer history perhaps works well when a reader is right on track and reading word-for-word. A too long history might make the model too biased. The scripts also allow adding a list of words with predefined probabilities, after which the model is normalised. This is used later to add miscue possibilities like spoken noise.

4.3 Reading tutors

Most previous work on developing automatic speech recognition for reading aloud has been done for the purpose of tutoring children learning to read. I believe these automatic *reading tutors* and the recognition of non-native language reading by high school students have very similar considerations technically.

The children may pronounce as native speakers, but recognising children's speech is more challenging than adult speech, because their fundamental frequency is higher and there is less children's speech training data available. The high school students match the adult speech corpora physiologically, but their non-native accent produces an acoustic mismatch. Neither target group thus matches the adult speech corpora perfectly and well matching acoustic data is hard to find. Even though the children struggle with the actual visual scanning of letters and the high school students struggle with understanding the language, it is logical to think the long and uncommon words are the most difficult for both groups. Furthermore both groups struggle with connecting the written form to the pronunciation, even if for different reasons.

In particular, the work done since 1993 in Carnegie Mellon University's still ongoing Project LISTEN[7] has looked automatic reading tutoring very thoroughly. When refining the finite state language models described in section 5, I found a paper describing the technical design and reasoning behind Project LISTEN's language models. I discovered that my design was very close to theirs, which somewhat validated the approach I had taken and provided a wealth of knowledge. Section 5.1 refers back to the technical lessons learned from Project LISTEN, but first the basic idea is outlined.

5 Miscue tolerant finite state language models

The task is to decode an utterance, and the utterance is a reading of a prompt text, so reading miscues will lead to deviations from the text. One idea is to explicitly represent the search space as the prompt text with deviating paths for the miscues.

[7] http://www.cs.cmu.edu/~./listen/

As shown in section 3 weighted finite state transducers can efficiently represent the search space for decoding.

The acceptor in figure 25 is the starting point for creating this miscue tolerating finite state language model. Each state corresponds to a point in the prompt text. If reading was a linear process of scanning letters and then reciting the corresponding phones, the states would be the places where the reader is looking at that instant: the space separating the incoming transition's word from the outgoing one. Other paths are added, representing miscues. For example a word may be repeated, perhaps more confidently as it has now been properly understood. Figure **??** shows the acceptor augmented with repetition transitions for each word. The transitions now have weights: the miscues are assumed less frequent than expected words, so this is reflected in the probabilities of each path. Table 6 lists the miscues implemented in this thesis.

A caveat with the approach of having one state per position in the text is that after a jump in the text, the reader is assumed to continue from that position forward. In reality more complicated self-corrections are possible, such as where the reader notices they misread one word from a few words ago and only repronounce that one. These paths are possible in the miscue tolerant finite state models of this thesis, but they incur a double penalty in the weight. Thus they will only be found if the acoustics strongly suggest it.

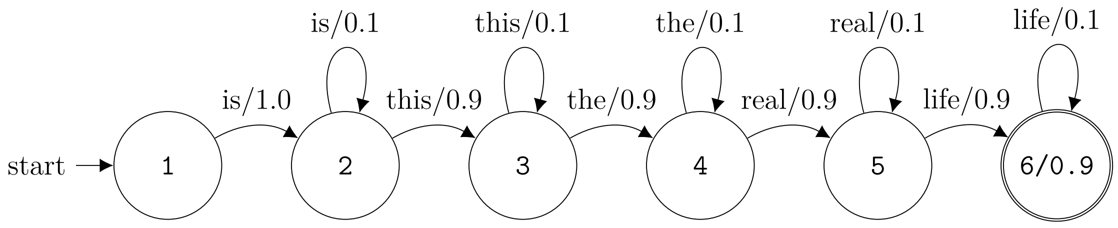


Figure 26: *A weighted finite state acceptor which has transitions representing the repetition of each word.*

As stated in section 4.3, I cannot claim that this is an original idea, even if I thought of it independently. Project LISTEN has already been using it for many years. Fortunately that simply means they have already tested on a wide scale what works and what does not. The next section describes the miscues that they found useful to implement in the finite state models.

5.1 Lessons from Project LISTEN

Jack Mostow describes the technical aspects of Project LISTEN's speech recognition in [54], which this section is based on. Commendably, he also mentions the solutions they tried but which turned out not to help; many times an improvement in one particular are came with a cost of decreased overall accuracy. In summary, he writes:

> "Rely on realism. The better we model children's oral reading, the better we can track it. What if any models boost tracking accuracy dramatically?"

Repetition:	saying the just read word again.
Skipping:	omitting one word.
Jumping forward:	omitting multiple words.
Jumping backward:	restarting from an earlier point than the latest word in the prompt.
Premature end:	stopping reading altogether before reaching the end of the prompt.
Spoken noise:	a model of speech-like sound which catch additions of words not in the prompt and help tolerate miscellaneous noises.
Truncation:	a word pronounced only partially from the start. This requires augmenting the lexicon. New pronunciations tagged with a truncation prefix are added for each word for all phonetic lengths from $n - 2$ to 2 phones, where n is the word's length in phones.

Table 6: *Miscues in the miscue tolerant finite state language models of this thesis.*

Table 6 lists the miscues that are used in this thesis. All but the spoken noise model are also in Project LISTEN's models. In the Kaldi recipes, a spoken noise acoustic model is often trained if the training transcripts have special markings for it. It seems Project LISTEN also had a separate path for restarting the whole prompt. It is not implemented separately in this thesis because it is included in jumping backward. The truncation miscue was the only lexical modification that did not ultimately hurt overall accuracy by hallucinating more miscues in Project LISTEN.

An important point raised by Mostow is that it may be enough to detect any miscue: for many purposes correctly identifying each miscue is not as important as being able to notice a deviation from the original text. Many miscues, like truncation, can serve a dual purpose of both representing a particular type of real-world phenomenon and a general purpose distractor. This way, fewer miscue models may suffice, providing better accuracy and less hallucinations. Mostow mentions trying more complicated state arrangements, which could solve the double penalty caveat of described in the last section. However the more involved models mainly served to make decoding much slower.

Project LISTEN also uses its own alignment algorithm, called MultiMatch. It is not implemented here, but Mostow reports it would serve reading better than the standard Levenshtein alignment.Another key finding that is out of scope of this thesis is that the durations of silence between words is a good gauge of reading fluency. Better tracking through better language models probably improves this gauge. Extracting the inter-word latency measure could provide valuable information at virtually no added computational cost.

5.2 Keeping the models deterministic

Some care is needed in order to keep the resulting finite state acceptors deterministic, or at least determinisable in a reasonable amount of time. As mentioned in section 3.3.3, the determinisation algorithm has an exponential worst case complexity, and does not halt for all inputs. In fact this insight came from initial tests where the compilation of some search graphs took many minutes or crashed. If the compilation succeeded, the graph was orders of magnitude larger in memory consumption than normal.

The jumps forwards and backwards were implemented by $\epsilon : \epsilon$ transitions. It seemed at first glance that disambiguation symbols would fix the problem, as they work well in the classic lexicon and N-gram determinisation cases. However, the combination of the different miscue paths allow infinite loops.

The strategy to counter this is to consume an input label at each outgoing transition. Instead of an $\epsilon : \epsilon$ transition going to the state where the next word to read is `is`, there is a transition with an `is` input label going to the state of having just read `is`. This takes care of infinite loops that do not consume labels.

5.2.1 Homophones

Particular care has been taken in dealing with homophones. This includes having the same word multiple times in a prompt text, i.e. here a word is considered homophonic with itself. Homophones are not rare since some words like `the` are very common and since discussing a subject often involves repeatedly using the subject's name. More homophones are added by the truncation miscues, as they add shorter pronunciations which are thus more likely to overlap.

If the words are different, the lexicon building automatically adds disambiguation symbols. This solves determinisation, but it does not solve the ambiguity: either word could be transcribed as the result. It is possible that the weights on the transitions solve the ambiguity, but due to symmetries, some truly ambiguous cases can also arise. If the words are the same, it is the determinisation algorithm's worst case complexity. Furthermore it is also a truly ambiguous case.

Fortunately simply by taking a definitive stance on which of the competing homophones should be used in each case, the ambiguity is solved, determinisation is guaranteed and fast and only one path needs to be searched. Coupled with the design choice that each transition should consume an input label, our finite state transducers actually stay deterministic from the start.

A map from each word to a list of its homophones is compiled when creating the lexicon. Algorithm 1 shows how the miscues are added: essentially each time the transition adding method is called, it first checks of transitions with homophone input labels already exist. Thus miscue path adding functions can be written without thinking about other miscue adding functions, instead their calling order determines which competing paths are chosen. The redundancy of the homophone map makes the algorithm efficient; there is an entry copy for each word of a set of homophones.

Algorithm 1: Homophone checking before adding an arc

```
States(Map: state id → list of transitions)
Homophones(Map: word token → Set of word tokens)  /* Maps a word to a
  set of its homophones. */

Method AddTransition(from: state id, to: state id, input: word token, weight:
  float):
    foreach transition ∈ States[from] do
        if input ∈ Homophones[transition.label] then
            Return                  /* Conflict found, do not add Arc */
        end
    end
    AddArc(from, to, input, weight)          /* No conflict, add Arc */
end
```

5.3 Weights

The weight on each transition defines the prior probability of taking that transition. Each miscue type has its own weight representing the frequency of that miscue. The weights should ideally be estimated on a separate, large, in-domain corpus by counting the frequencies of each miscue. Furthermore the miscue frequencies are probably quite speaker dependant. This requires a lot of data since there are quite many parameters to estimate. In practice the weights only need to reflect the fact that correct words are much more likely than miscues. Therefore I have made educated guesses for each weight to serve as agreeable default values.

The calculation of the weight for the jumps backward and forward has a jump length dependent factor. The probability of the miscue is lower the longer the jump. The intuition is that the reader focuses their attention on a few nearby words.

There is an additional parameter, which controls the bias of the model. It is used to boost the probability of the correct words. Having just one parameter to adjust requires less data. Furthermore it is also convenient for controlling the trade-off between detecting miscues and not hallucinating miscues.

5.4 Output labels

In case the miscues should be explicitly labeled, a tag can be added to the output. In that case, a finite state transducer is used instead of an acceptor. Figure 27 shows an example. There are epsilon input label, but they are only used to output multiple labels per consumed input, forming a chain.

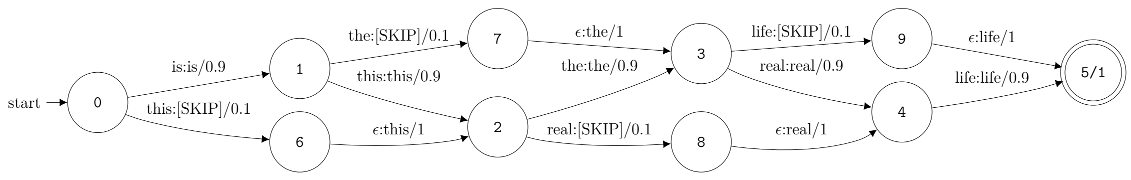


Figure 27: *A finite state transducer language model which tags the skipping miscue. Notice how the last word cannot be skipped. This is because there is no input label to consume after the last word. The premature end miscue after the second last word serves the same purpose of not saying the last word.*

Remarkably, this miscue tagging transducer can also be used afterwards to obtain the same tagging. This is because the transducers made with the above recipes are deterministic by nature: they transduce a sequence of words to one unique output sequence. The transcript from automatic speech recognition is turned into a finite state transducer with the identical input and output labels. An example is shown in figure 28. This transducer is then composed on the left with the miscue tolerant finite state language model, producing a transducer with just one transition per state. This is projected and the result is read off the resulting acceptor, as shown in figure 29. This way the output from the biased N-gram language models of section 4.2 can be interpreted as miscue detection.

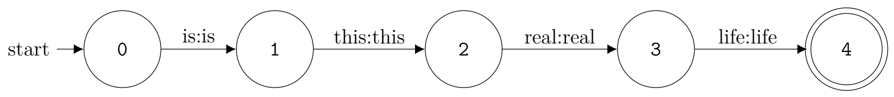


Figure 28: *A transducer representing a transcript from a speech recognition system. The speaker was reading* `Is this the real life?` *but skipped* `the`*.*

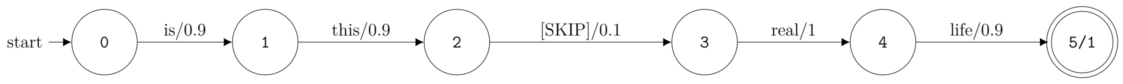


Figure 29: *An acceptor after composing the transducers from figure 28 and 27 and projecting on the output labels. The result is the accepted sequence.*

5.5 Implementation code

The creation of miscue tolerant finite state language models is handled by scripts written in the Python programming language[8]. Additionally, some Bash shell scripts handle connecting logic. The language models are technically speech recognition toolkit agnostic, but only integration with the Kaldi toolkit is facilitated by the connecting logic.

The scripts output a text file specifying a weighted finite state transducer in the OpenFst format[55]. The Kaldi toolkit links against the OpenFst libraries.

The scripts are open source with a permissive licence. They are released in a Github repository[9]. The Github repository also contains some scripts for calculating miscue fetching accuracies which are used in experiments to balance false positives and miscue detection.

[8]Python version 2.7, `https://www.python.org/`

[9]`https://github.com/Gastron/miscue-tolerant-lm-fst`

6 Experiments

This section narrates the empirical part of the thesis. The first experiments were carried out on the Wall Street Journal corpus with the goal of rapidly validating ideas. A well thought out recipe for acoustic model training is available in the Kaldi repository, as well as the facilities for obtaining a suitable lexicon. The speech is quite clean, so any progress in language modeling should not get masked by poor acoustic model performance. There are few miscues in the Wall Street Journal corpus, so they were simulated by splicing the recordings and manipulating the transcripts.

When the tests on simulated data ultimately showed that the miscue tolerant finite state language models worked in theory, it was time to test them in practice. The DigiTala data set had in-domain data with real miscues. The only thing missing was reference transcripts for the recordings of reading. To test miscue detection, both the original prompt text and the actual utterance transcripts are needed. Labeling had to be done manually. Once the dataset was labeled, the miscue tolerant finite state language models were tested against forced alignment and biased N-gram language models.

First the Wall Street Journal corpus experiments are recounted from dataset construction to results. Then the in-domain experiments are described, starting from manually labeling the miscues and ending with the main empirical results of this thesis.

6.1 Wall Street Journal experiments

Work on the Wall Street Journal corpus began with running the Kaldi acoustic model training script interactively. It served as both an introduction to the Kaldi toolkit and its conventions and of course to provide a decent acoustic model. The acoustic model was used to temporally align the transcripts to the recordings at the word level. With these temporal alignments, some miscues could be simulated.

First the miscue simulation procedures are described. Next the choices for the free parameters of the language model are briefly explained. The word error rates are then reported on the Wall Street Journal test set with simulated miscues.

6.1.1 Simulating miscues

Some miscues lend themselves well to being simulated from existing recordings. In the terms of table 6 repetition and jumping backward are achieved by looping the recording: playing up to a certain point, then restarting from an earlier point and playing till the end. Skipping, jumping forward and ending prematurely are achieved by cutting the recording: playing up to some point and then continuing from a later point. Sudden cuts and loops may sound unnatural to a human ear. As long as care was taken to swiftly fade the recording out and in at the boundaries to avoid loud pops, the acoustic model did not break down.

Essentially a miscue involving jumping between points in the text has a natural counterpart in the audio domain. With the temporal alignments, these jumping miscues can easily be simulated both in the audio and the text domains in a coordinated manner. The original text serves as the prompt, while the transcript with the simulated miscues is the reference. The original text is fed into the language model creation scripts, with the hope that the resulting language model can decode the simulated miscues as notated in the reference.

The truncation miscue can also be simulated with a loop at the phone level, though phone level splicing may require more care as single phones are very short temporally. I had simply not implemented the truncation miscue in the finite state language models, so there was no reason to provide a test for it.

The spoken noise miscue was implemented in the finite state models at an early stage, since the Kaldi acoustic models readily enabled it, but it is more difficult to simulate. Speech synthesis can be used to generate extraneous speech with matching voices, but it would have been much too complicated for the sake of preliminary testing. Instead just the transcripts were modified so that a randomly chosen word was simply deleted, and the audio was kept intact. This time the modified transcript served as the prompt text. The language model then would not be expecting the deleted word. As long as the word appeared only once in the original text, it could not be decoded directly. Instead it would be speech like noise in the audio, just as needed.

Three separate test datasets were created, one where cuts were made in the recordings at randomly chosen word boundaries, one where loops were introduced so that randomly chosen passages were said twice, and one where randomly chosen words were deleted from the transcripts. The same was done for the development set. For each experiment the language model was created with all miscue paths. This way, each miscue's effect on the whole model could be tested separately.

6.1.2 Parameter optimisation

A few iterations on the development set of the Wall Street Journal corpus showed that a substantial boost factor for the correct word, as described in section 5.3, gives the lowest word error rates. This reflects the fact that the corpus is quite clean and the acoustic model can distinguish the words quite well.

The boost parameter was not increased limitlessly. Doing so could have given even lower word error rates, but the goal of these experiments was not to minimise the word error rates. The goal was to validate an idea. An extremely high boost factor would be a special case, but the default parameters should also give adequate results. Thus a boost factor of 1000 was chosen as a reasonable compromise.

6.1.3 Results

The word error rates on the three datasets with miscue tolerant language models are shown in table 7. The word error rate in large vocabulary decoding was 10.88%, as stated in section 2.8.1. The error rate on each dataset is at least halved compared to large vocabulary decoding, so the miscue tolerant finite state language model implementation seems sound.

Simulation type	**WER[%]**
Cutting speech	3.22
Looping speech	1.95
Tampering transcript	5.41

Table 7: *The word error rates with miscue tolerant language models on the three simulated datasets. The speech cutting simulation produces forward jumping miscues, the speech looping simulation produces backward jumping ones, and the transcript tampering simulation produces spoken noise.*

The forced alignment procedure is bound to fail on these simulated datasets, because each and every utterance has a large discrepancy between the prompt and the correct transcript. However in a control experiment the biased N-gram model produced very similar word error rates as the miscue tolerant language model. It cannot explicitly decode the spoken noise, so the transcript tampering set was not tested. Clearly testing on actual in-domain data is needed to better compare the two models.

6.2 DigiTala experiments

The in-domain test data is considerably more difficult than for automatic speech recognition than the Wall Street Journal corpus. The sources of difficulty can be attributed coarsely to two categories.

Firstly the technical quality of the recordings is sometimes adverse. The DigiTala pilot tests were done in normal classrooms with on average ten students taking the test at once. The students used headset microphones, but other students' voices are clearly audible in the recordings. Figure 30 is a photograph from classroom with the level acoustic isolation that can be reasonably expected at the scale of the examinations. Furthermore, some in some recordings the speech is not audible at all.

Secondly the language was not fluent. Although the actual matriculation examinations are taken very seriously, in the pilot tests the students were not motivated to do their best. If anything, we test givers tried to create a relaxed atmosphere. The students' skill level was very varied: there were some native Finnish Swedish speakers, but also some inexpert speakers who had only completed one course of Swedish in high school.

First the work in labeling the miscues of the dataset is recounted. Then the free parameter choices are justified. Next some details are exposed about making the word error rates of the different method comparable. Finally the word error rates for all the compared models are reported and a trade-off is explored between detecting miscues and hallucinating them, i.e. decoding correct words or silences as miscues.

6.2.1 Labeling miscues

The constraints of this thesis required that I manually label the test dataset myself. Transcribing a recording manually is a laborious task. The industry standard for

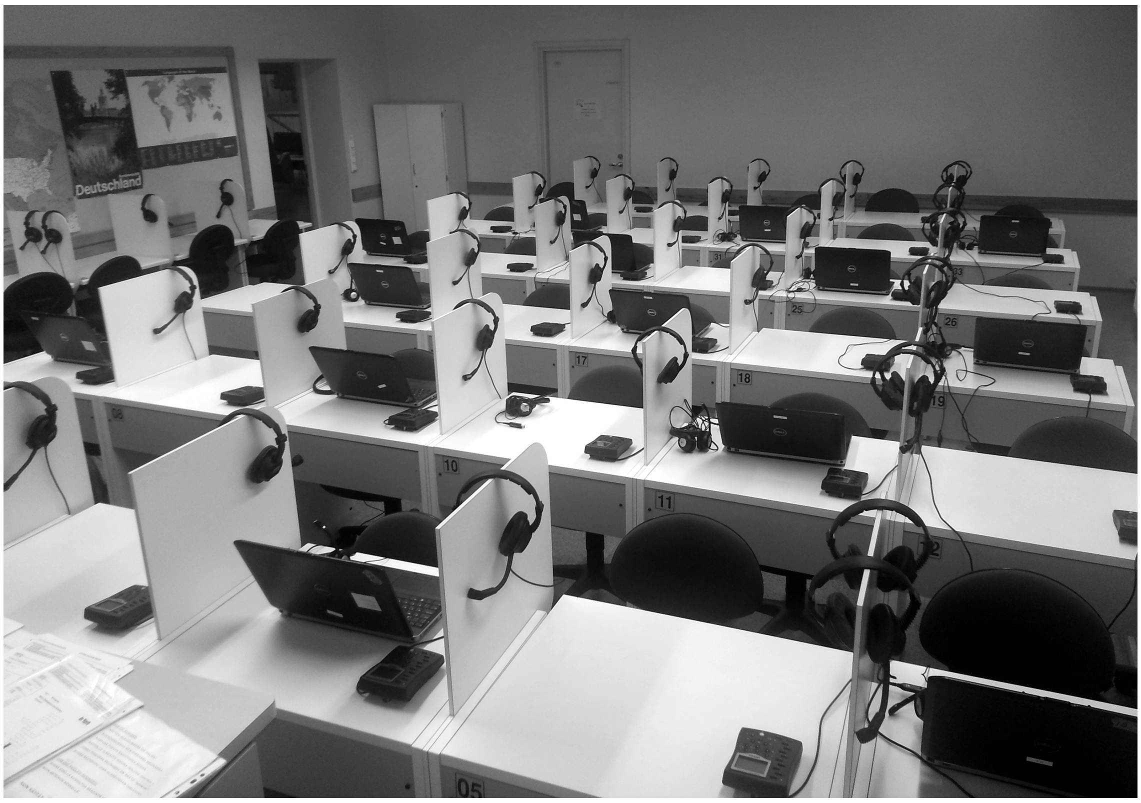


Figure 30: *A classroom in pilot test configuration. The test was done on laptops recording through headsets. Though the screens and the seating patterns provide visual privacy, the acoustic environment has little in terms of isolation.*

professionals is quoted as an hour of work for fifteen minutes of audio[10]. Since that time was not available, a faster approach was needed.

As stated in section 4, the original prompt gives a very good prior on the actual utterance. Often the utterance in fact contains no miscues, the original text is a good transcript as is. A small program was written which plays a recording and displays a text edit box, in which the original prompt is given as a default answer. This way, a miscueless reading can be processed in real-time and accepted with a single click.

To further improve the speed, the transcription was not done in a precise orthographic way. An orthographic transcription might have been more valuable in the future, but in this thesis it was not needed. Instead complete words were transcribed as is, but other words were tagged mispronounced, hesitated or truncated. The mispronounced tag was used for words which I thought where the reader clearly used the wrong phones. Pronunciations which are specific to some dialect, especially differences between Swedish spoken in Finland and Sweden, were not marked as mispronounced. The hesitation tag was used for words where the pronunciation was exceedingly slow and clearly the word was not familiar to the reader. The truncation tag was used like the miscue: the cases where the reader starts pronouncing a word,

[10]In the Wikipedia article: https://en.wikipedia.org/wiki/Transcription_(service), accessed 17.7.2017

but stops midway through. In addition, speech like noises and use of Finnish was marked with a spoken noise tag and other loud noises were labeled as non-spoken noise.

These tags were chosen because they seemed very profuse during early corpus exploration. During the labeling process I felt like the tags adequately represented all the encountered data. Table 8 lists the dataset in numbers. Although the total number of miscues may seem low at first glance, the jumps can result in utterances drastically different from the prompts.

Coverage	
Utterances	1023
Total time	3 hours 50 minutes
Speakers	149
Most utterances per speaker	11
Speakers with only one utterance	45
Different prompts	15
Short prompts	10
Long, multi-sentence prompts	5
Utterances from long prompts	142
Total words	20300
Percentage of words from long prompts	67%
Miscues	
Correct words	16515
Repetitions	171
Skips	119
Jumps	130
Premature ends	112
Truncations	241
Spoken noises	281
Miscues in total	1054

Table 8: *The DigiTala dataset in numbers. The jumps forward and backward in the text are reported together here. The premature endings are not annotated.*

6.2.2 Making the results comparable

A choice had the be made about words tagged as mispronounced or hesitated. Since they cannot be explicitly decoded with those tags, the tags were in the end simply removed, giving just normal words. The mispronounced words should be handled by the automatic articulation analysis and the hesitations could be picked up by an automatic prosody analysis. Thus both types of words should ideally be passed on as normal by the preprocessing system. Additionally since the forced alignment procedure cannot detect truncations, the truncation labels were simply removed when calculating the forced alignment word error rates, leaving the words as if they were pronounced fully.

The forced alignments were not actually computed. Instead it was assumed that each time, the whole prompt aligned. The reason is that although the computing the alignments is quite simple, making sure the alignments are sensible is not trivial. When a reasonably large search space is used, like with the miscue tolerant and biased N-gram language models, a correct transcription can be assumed not to come about by chance. However as stated in section 4.1, the forced alignment will most of the time result in some sort of alignment. The assumption that an alignment can be forced every time will increase the word error rate in utterances that are just silence and it will decrease the word error rate when the alignment would have failed completely. This should be reflected in an increased variance of the word error rate.

In the Kaldi scoring scripts, it typical to remove the spoken noise labels from the references and the hypotheses. On one hand it is a miscue that is explicitly modeled. On the other hand it has no practical value after recognition. Having the miscue explicitly modeled is a way to make the models more robust to unexpected sounds; it may catch all types of noise. Therefore when calculating the word error rates, the spoken noise miscue should be removed. However when characterising the miscue detection rate, the spoken noise miscue is kept in the transcripts.

6.2.3 Parameter choices

The DigiTala dataset was so small that a separate test set could not be afforded. Instead 10-fold cross validation was used. In each iteration, the training folds were used to find good values for the free parameters of the miscue tolerant finite state models and the biased N-gram models. For the miscue tolerant models, the correct word boosting factor was optimised, and for the biased N-gram models, both the N-degree and the discounting constant were optimised. In addition, the language model scale was optimised for both models. The folds were the same for both model types.

First a list of plausible values were determined for each parameter. The optimal parameters were found by decoding the training set with models of each combination of values and finding the model with the best word error rate. Because of this computationally expensive parameter search method, the value combinations are not at local word error rate minima. However in practice the models are not sensitive to the exact values of the parameters and I feel confident these values are in the optimal order of magnitude.

Table 9 shows the chosen parameter values for each fold. There are no big differences between the folds. The average values over the ten folds were used when calculating the miscue detection rate at different hallucination rates, except crucially the correct-word-boosting factor and the discounting constant were varied. This way different hallucination rates were achieved. The expectation is that the more biased the model is toward the prompt word sequence, the less miscues it hallucinates and also the less miscues it detects. First the word error rates are reported and then these miscue detection rates are characterised.

	Miscue tolerant		Biased N-gram		
Fold	Correct-boost	LM-scale	N-degree	Discount	LM-scale
1	1.0	1.8	3	0.1	1.3
2	1.0	1.8	3	0.1	1.3
3	1.0	1.7	3	0.1	1.3
4	1.0	1.8	3	0.1	1.3
5	1.0	1.8	3	0.1	1.3
6	1.0	1.7	3	0.1	1.3
7	1.0	1.7	4	0.1	1.2
8	1.0	1.9	3	0.1	1.3
9	1.0	1.6	3	0.1	1.3
10	1.0	1.7	3	0.1	1.3

Table 9: *The chosen parameter values at each fold. LM-scale is short for the language model scale.*

6.2.4 Measuring word error rates

The word error rates are shown in table 10 for the whole DigiTala dataset, compiled from the test sets of each cross validation fold. Since the dataset was small, confidence intervals are computed using Bisani and Ney's bootstrapping method described in section 2.7.2. The intervals are at the 95% confidence level. For both biased N-grams and miscue tolerant finite state language models the interval size relative to the mean value is approximately 12%. For forced alignment, the interval is approximately 24%. As expected in section 6.2.2, the variance is higher for forced alignment.

Method	WER [%]
Forced alignment	13.90 ± 3.37
Biased N-gram	4.16 ± 0.48
Miscue tolerant	3.77 ± 0.47

Table 10: *The word error rates (WER) for the different methods, with 95% confidence intervals. The confidence intervals of the biased N-gram models and the miscue tolerant models overlap.*

In all the other word error rate values presented here, the spoken noise labels have been removed from the transcripts, but table 11 shows the word error rates without the removal. I report this result because intriguingly the biased N-gram performs better in when measured this way.

The word error rates for each fold are reported in table 12. Each fold by itself is just a tenth of the data, so the test set size is just 102. With such a small test set, the Bisani and Ney bootstrapping method gives nonsensical results: impossibly large and negative word error rates. Thus no confidence intervals are reported.

Table 8 mentions 138 utterances from longer, multi-sentence prompts. These can be isolated from the test folds and the word error rate can be calculated separately

Method	**WER** [%]
Forced alignment	14.39 ± 3.32
Biased N-gram	4.86 ± 0.55
Miscue tolerant	5.31 ± 0.61

Table 11: *The word error rates (WER) for the different methods without removing the spoken noise labels, with 95% confidence intervals. The confidence intervals of the biased N-gram models and the miscue tolerant models overlap again, but here the biased N-gram model has a better mean value. The forced alignment procedure cannot detect spoken noise labels, so every instance increases its word error rate.*

	WER [%]	
Fold	Miscue tolerant	Biased N-gram
1	3.30	4.07
2	3.82	3.91
3	4.69	5.06
4	4.18	4.93
5	3.64	3.92
6	3.87	4.38
7	3.31	3.63
8	2.58	2.81
9	5.65	5.87
10	3.27	3.75

Table 12: *The word error rates for each fold. The miscue tolerant method has a lower word error rate in every fold.*

for both the long utterances and the remaining short ones. Table 13 lists the results.

		WER [%]	
	Miscue tolerant	Biased N-gram	Forced alignment
Long prompts	2.40	2.48	13.83
Short prompts	6.38	7.35	13.96

Table 13: *The word error rates for long and short prompts.*

While there are large differences between word error rates of long and short prompts, dividing the recognition results by individual prompts gives an even more detailed view of the results and secondly of the data, too. Table 14 lists both the prompts, their total number and the speech recognition results on them.

	WER [%]		
	Miscue tolerant	Biased N-gram	Forced alignment
Prompt 1 *Allt fler högskolestudenter pluggar på distans i Sverige.* total utterances: 88	2.84	2.70	1.70
Prompt 2 *Ansökningstiden till juridiska fakulteten går ut 9.4.2015.* total utterances: 91	19.79	21.61	75.04
Prompt 3 *Bananer med droger i smugglades i tunnelbanan.* total utterances: 91	3.08	3.85	2.47
Prompt 4 *Bilreparatören fick en schock när han öppnade bagageluckan.* total utterances: 85	5.32	5.60	6.72
Prompt 5 *Den moderna mormodern tågluffar med barnbarnen.* total utterances: 92	5.12	6.83	5.80
Prompt 6 *Dödsrisken 7,3% mindre bland cyklister med skyddshjälm.* total utterances: 90	10.99	13.40	24.53
Prompt 7 *Kyligt väder försenade jordgubbsskörden.* total utterances: 84	6.61	5.79	7.44
Prompt 8 *Rekordmånga ålänningar gör frivillig värnplikt.* total utterances: 88	5.54	8.53	6.18

	WER [%]		
	Miscue tolerant	Biased N-gram	Forced alignment
Prompt 9 *Välfärdsstaten klarar inte av den ökande arbetslösheten.* total utterances: 83	3.51	3.34	2.84
Prompt 10 *Vi köper begagnade kläder i gott skick.* total utterances: 89	2.45	3.83	5.67
Prompt 11 *Hej peter! Jag försökte ringa dig, men din mobil var avstängd. Var är du? Hoppas att du hör mitt meddelande snart. Eva ligger på sjukhus! Hon råkade ut för en bilolycka i morse, men det är ingen fara med henne.* *Eva åkte till jobbet med min bil. I den stora korsningen på Vasa-gatan kom en buss som körde för fort och kunde inte stanna vid rödljuset. Som du kanske vet var det kyligt och jättehalt på morgonen. Eva hann inte stoppa sin bil utan körde rakt in i bussen. Hon kände sig yr, hon hade ont i huvudet, ryggen, och ena benet. Därför fördes hon till sjukhus.* *Till all lycka mår Eva ganska bra nu. Läkaren tror att hon kommer att skrivas ut i övermorgon, eller kanske på fredag. Ska vi hälsa på henne på sjukhuset? Hon är i rum nummer 28b på fjärde våningen. Om du vill så följer jag gärna med. Jag kan visa rummet för dig, sedan det är svårt att hitta på sjukhuset. Min bil är okörbar just nu, så det blir du som får skjutsa mig. Men ring mig när du hör det här meddelandet.* total utterances: 43	2.32	2.47	8.64

	WER [%]		
	Miscue tolerant	Biased N-gram	Forced alignment
Prompt 12 *Det är kul att prova på något nytt som det här elektriska talprovet på dator. Får jag framföra en liten dikt för dig?* *Tjejer och killar i höstmörkret,* *orkar ni ännu läsa?* *Ljusen på bordet värmer så skönt.* *Se ut och hör hur snögubben ropar på gården:* *"Snälla vän köp mig en morot till näsa."* total utterances: 93	2.56	2.51	23.72
Prompt 13 *Hej peter! Jag försökte ringa dig, men din mobil var avstängd. Var är du? Hoppas att du hör mitt meddelande snart. Eva ligger på sjukhus! Hon råkade ut för en bilolycka i morse, men det är ingen fara med henne.* total utterances: 2	2.47	1.23	1.23
Prompt 14 *Eva åkte till jobbet med min bil. I den stora korsningen på Vasa-gatan kom en buss som körde för fort och kunde inte stanna vid rödljuset. Som du kanske vet var det kyligt och jättehalt på morgonen. Eva hann inte stoppa sin bil utan körde rakt in i bussen. Hon kände sig yr, hon hade ont i huvudet, ryggen, och ena benet. Därför fördes hon till sjukhus.* total utterances: 2	0.00	0.00	2.19

	WER [%]		
	Miscue tolerant	Biased N-gram	Forced alignment
Prompt 15 *Till all lycka mår Eva ganska bra nu. Läkaren tror att hon kommer att skrivas ut i övermorgon, eller kanske på fredag. Ska vi hälsa på henne på sjukhuset? Hon är i rum nummer 28b på fjärde våningen. Om du vill så följer jag gärna med. Jag kan visa rummet för dig, sedan det är svårt att hitta på sjukhuset. Min bil är okörbar just nu, så det blir du som får skjutsa mig. Men ring mig när du hör det här meddelandet.* total utterances: 2	2.37	1.18	0.59

Table 14: *Word error rates on the different prompts. The last three prompts only have two utterances each. This is because for two students Prompt 11 was split into three parts.*

The real-time factor in all of these experiments is below 0.1, much faster than real-time. This is not surprising considering the compact search space of the language models, the reduced output rate of the deep neural network acoustic models and the use of a fast scientific computing cluster.

6.2.5 Measuring miscue detection

To measure the miscue detection rate, each word in the reference and the hypothesis transcripts were tagged either correct or with a miscue label. The tagging was done as described in section 5.4. Then the hypotheses were Levenshtein aligned with the references. Each aligned word pair was compared. This way it was determined whether the hypothesis was a correct detection of a correct word or a miscue, or if the hypothesis was a false acceptance of a miscue as a correct word, or if the hypothesis was a hallucination of a miscue from a correct word.

The trade-off between detecting miscues and not hallucinating miscues can be drawn as a receiver operating characteristic curve. Correctly detecting miscues is the true positive rate, which is drawn as a function of the of hallucinating miscues, the false positive rate. Figure 31 shows this curve for the miscue tolerant finite state language models and figure 32 shows the curve for the biased N-gram models.

Table 15 lists the amounts of miscues the two miscue detecting methods found when achieving the results reported in table 11. In Miscue Detection/Miscue Hallucination rates this is 12.6%/2.0% for the biased N-gram models and 22.2%/3.8%for the miscue tolerant models.

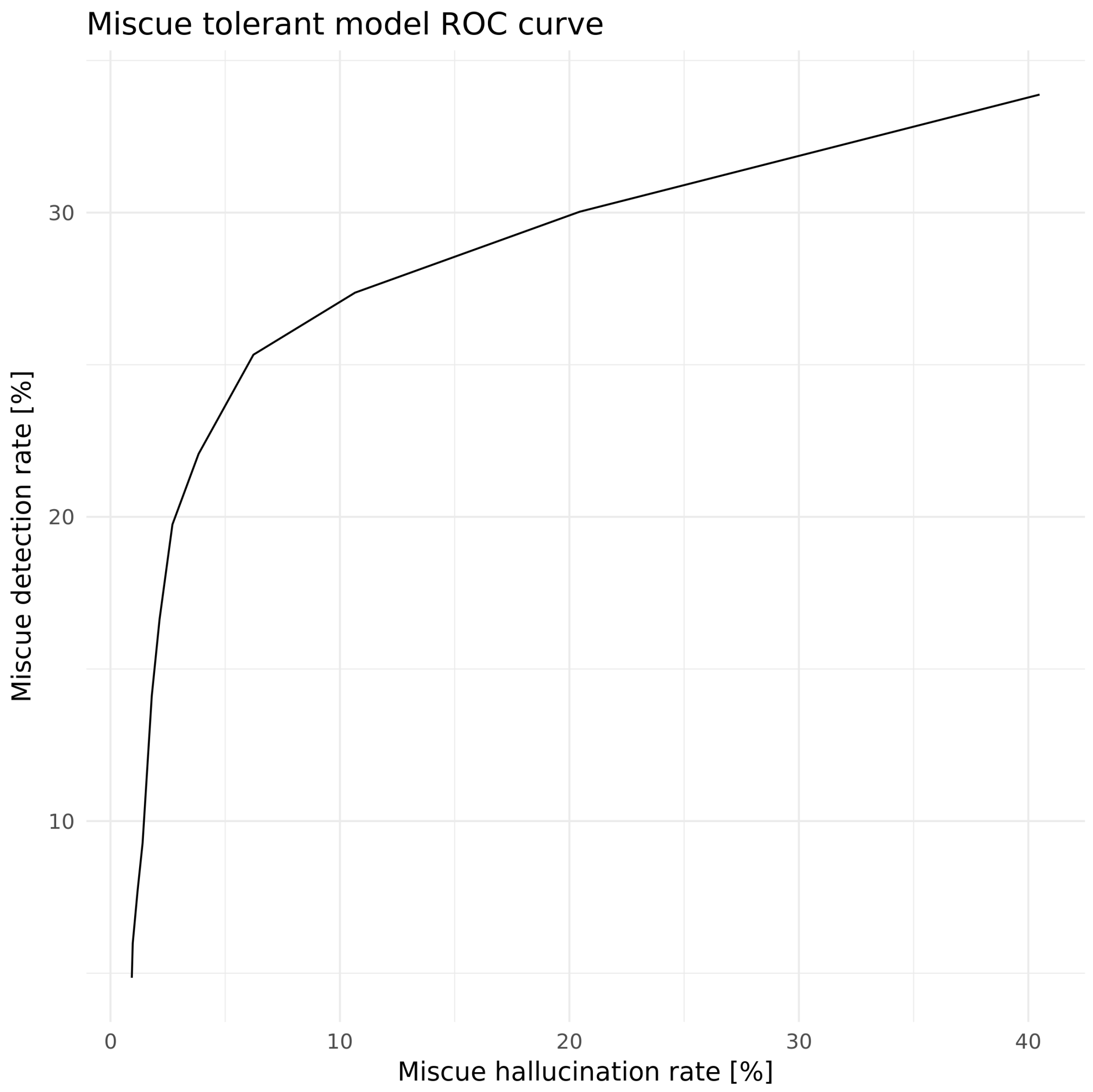


Figure 31: *The receiver operating characteristic curve for the miscue tolerant finite state language model on the DigiTala data, from varying the correct word boosting factor.*

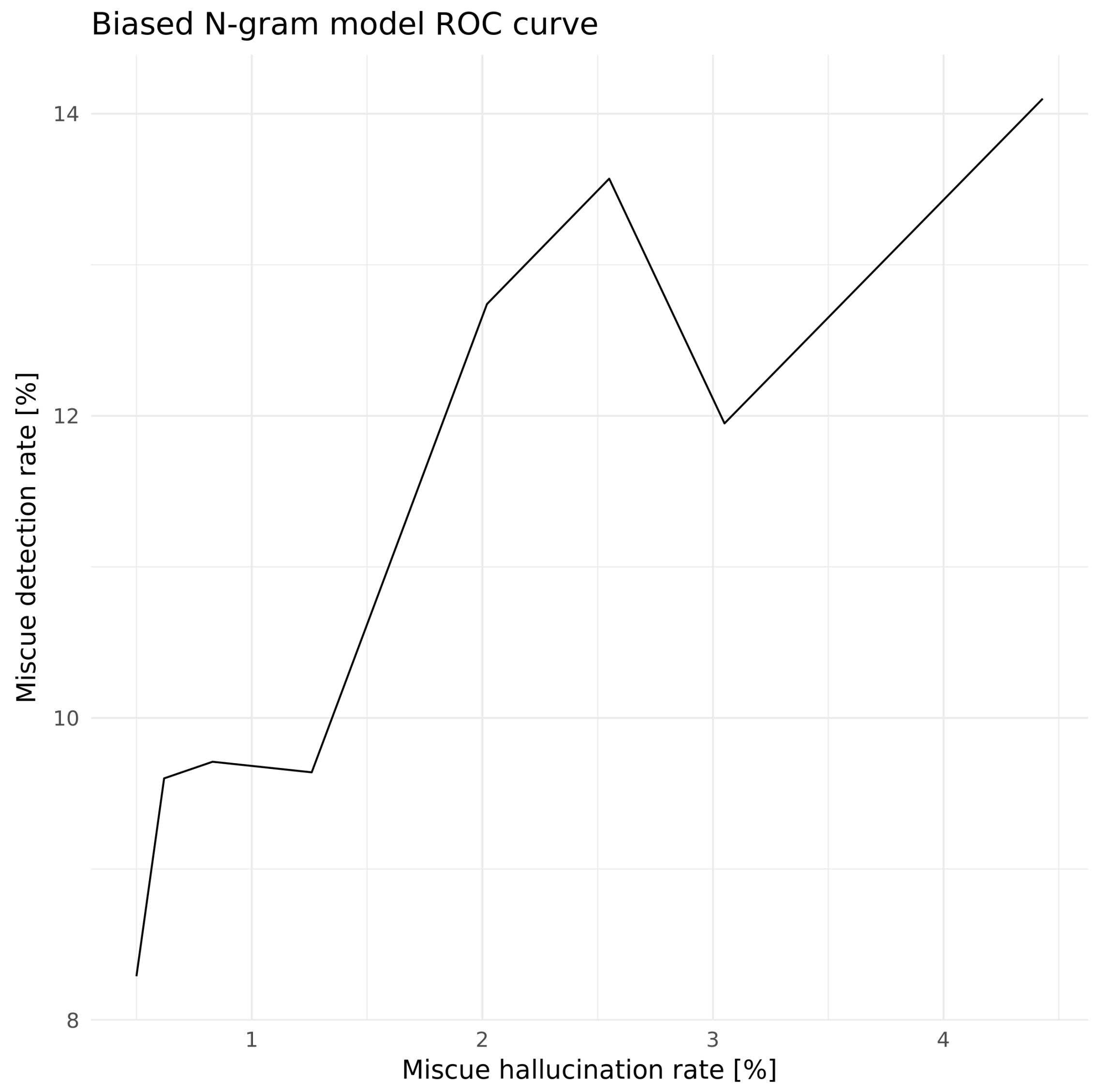


Figure 32: *The receiver operating characteristic curve for the biased N-gram models on the DigiTala data, from varying the discounting constant*

	In corpus	Miscue tolerant	Biased N-gram
Correct words	16515	18198	16176
Repetitions	171	223	182
Skips	119	196	62
Jumps	130	160	234
Premature ends	112	94	84
Truncations	241	227	66
Spoken noises	281	518	213
Miscues in total	1054	1418	841

Table 15: *Miscues detected by the two miscue detecting methods. The higher than corpus amount of correct words by the miscue tolerant models means that the method hallucinated many words.*

7 Discussion

In this section I first present my interpretation of the DigiTala data experiments and their results. I look at the word error rates and the miscue detection separately. At the end of this section I present some ideas for future work.

The Wall Street Journal results are not interpreted further. They gave confidence to test the models on real world data, but I believe the real value of the Wall Street Journal experiments lies in the scripts used to simulate miscues. Those scripts could be used again in the future to test miscue detection in under-resourced languages, to make sure the acoustic models are good enough to detect at least artificially created miscues.

7.1 Interpreting the word error rates

Table 10 presents the main empirical finding of this thesis. The biased N-gram models and the miscue tolerant finite state language models reach low word error rates. The miscue tolerant models have 9.4% better word error rate than the biased N-gram models, but this is difference is not statistically significant with the sample size.

I find five interesting points in the results. Firstly the two miscue detecting methods shined in longer utterances. Secondly the individual prompts have large differences between them. Thirdly the miscue tolerant finite state transducer hallucinates many spoken noises. Fourthly the overall results for forced alignment suggest that a miscue detecting method is needed. Finally some details suggest the miscue tolerant finite state language models represent the utterances better than the biased N-gram models.

7.1.1 Longer utterances were easier

The longer utterances had almost three times smaller word error rates than the short utterances with the two miscue detecting methods. On the other hand the forced alignment word error rates were practically identical between short and long utterances. This suggests the longer utterances were easier to decode. The longer prompts had more common, easy words such as prepositions and conjunctions which are necessary to give the long text some structure. The short prompts on the other hand had many difficult tongue-twisting words. The focus was on interesting idioms rather than sensible sentences.

An error in tracking could lead to catastrophic failure in a long utterance and some errors are inevitable. Thus the results suggest that both miscue detecting methods could tolerate some errors and get back on track.

Although the longer prompts were a minority in the number of prompts, they accounted for 67% of the words in data. Thus good performance on them was key to low word error rates. This may be an important consideration in preparing exams, if each prompt should be scored individually.

7.1.2 Large differences between prompts

Some individual prompts had very large word error rates, particularly prompts 2 and 6. These two prompts had numerals, which proved to be extremely challenging for the students. The date in prompt 2 only had a few correct renditions in all of the readings. Numerals should have some special, more miscue robust paths in the language models and the students need some practise.

In a few individual prompts the forced alignment word error rate was lower than that of either miscue detecting method. This means the miscue detecting methods could do better. If prompt specific parameter estimation is possible, it could be very beneficial. However this would require data from those prompts, and the matriculation examination questions are secret.

It seems that the more difficult the prompt, the worse the recognition result is as well. This is quite logical, but it is unfortunate for matriculation examinations, where some difficult prompts may be needed to bring out the differences between students.

7.1.3 Hallucinated spoken noise

The word error rate of the miscue tolerant models is higher when the spoken noise labels are kept in the transcripts. Table 15 shows that the miscue tolerant models detect almost twice as many spoken noise labels as there actually are. Thus the issue would be solved by a more accurate weight for the spoken noise miscue path.

7.1.4 Miscue detecting methods are needed

The overall word error rate for forced alignment is an order of magnitude larger than the two miscue detecting methods. This alone proves that miscue detecting methods are needed. In addition, the forced alignments may be unreliable in their result. Though the apparent word error rate can sometimes be low, it would be laborious to check the alignments for errors.

The difficult prompts with numerals had particularly high word error rates for forced alignment. In their case, with an over 50% word error rate, the further processing steps might have produced results based on pure luck.

7.1.5 Details in favor of miscue tolerant models

Though the overall word error rates were not statistically significantly different, I think some details favor the miscue tolerant finite state language models.

- On every cross validation training iteration the miscue tolerant models had a noticeably higher language model scale than the biased N-gram models. This suggests the miscue tolerant models provided a more trustworthy predictions about the training data.
- The miscue tolerant models had better miscue detection. Particularly the detection of truncations was much better, as evidenced by table 15. The spoken

noise and truncation miscues were artificially added to the biased N-gram models and they could not leverage the position in the text for predicting.
- The miscue tolerant models can be fine tuned when more specific information becomes available. New types of miscues can be added. It is more difficult to address a specific point in a text, such as a numeral, with the biased N-gram models, because their states represent word histories instead of positions in the text.

I think the biased N-gram models could have a large advantage in slightly longer texts than what is used in this thesis. The Kneser-Ney backoffs become more meaningful in that case, as word histories get more occurrences. Furthermore the biased N-grams could quite easily incorporate a background N-gram model trained on a large text corpus by backing off to it. I don't see a straight-forward way to do that with the miscue tolerant finite state language models.

7.2 Interpreting the miscue detection rate

The receiver operating characteristic curves of section 6.2.5 would lead to believe that the models are actually not very good at detecting miscues. Interpreting this result in light of the low word error rates is more difficult. I think four factors explain the data.

- Firstly the parameters that were not adjusted were taken from the 10-fold cross validation training averages. The cross validation training criterion was achieving as low a word error rate as possible. This goal is very different from detecting miscues: a low average error comes from predicting the average case correctly, while detecting miscues is about catching outliers. Essentially, it all comes down to the bias-variance trade-off.
- Secondly I believe that the low detection rates are in part due the way miscues are labeled. Each miscue is a tagged on one word only and other words are labeled as correct. For example a jump to another part of the text has to be recognised at the exact position. Decoding the jump at an off-by-one position incurs the full penalty of missing a miscue, even though the speech tracking may have been almost as good.
- Thirdly the parameters that were varied in the experiments were not perfectly suited to trading more hallucinated miscues for better detection. The N-gram discounting constant alone could not be adjusted far enough to achieve even a 20% miscue detection rate. The boosting factor of the miscue tolerant finite state language models controlled the trade-off better, but ideally the individual weights of each miscue path should be optimised.
- Finally there are quite many miscue types. They can each be confused with each other. This makes detecting the correct miscue more difficult. On the other hand as mentioned in section 5.1 for good recognition it may be enough to detect that some miscue happened.

There is room for improvement in miscue detection. Fortunately the miscue hallucination rates are quite low, which I feel is important for fairness in a spoken language testing context.

7.3 Future work

In this section I briefly motivate and describe some tangential ideas and improvement opportunities. Primarily I think a validator for the recognition results should be trained and a better parameter estimation procedure for the miscue tolerant finite state language models should be developed. The validator could also be used when cleaning a speech audio corpus for training. Secondarily I believe a spoken language practice tool could be built with the miscue tolerant language models and finally I see some room for improvement in miscue detection.

7.3.1 Rejecting bad recognition results

Since recognising and segmenting the read speech is only the first step in processing the utterances, the results of the speech recognition system should be validated. In normal speech recognition this is very difficult, because there is nothing to compare the results to. In the task of this thesis the validation has a natural reference: the prompt.

There are two reasons to reject a recognition result: the recogniser did not work well or the student's speech did not match the prompt well. In both cases I expect the symptoms to be similar. The decoder probably finds little of the prompt and detects a lot of miscues. Moreover in both cases, the further processing steps would be done on false premises. Thus the validator should calculate some distance measure between the recognition result and the prompt and a threshold should be found by training on both acceptable and unacceptable samples.

Students should not be automatically penalised for miscues, but the prompt text is still the expected recognition result. One or two miscues should not make the distance measure grow too large. Levenshtein distance could be a good starting point, but perhaps some tolerance for jumps in the text should allowed. A long jump in the text, for example restarting the whole prompt, would result in a large Levenshtein distance.

7.3.2 Validator for speech corpus cleaning

Validating an utterance based on the text that is supposed to be said in the recording is not only useful in reading tasks. It is a common problem in acoustic model training that the training transcripts do not match the audio exactly, and the corpus needs some cleaning. Using the miscue tolerant finite state language models should also be researched in this corpus cleaning domain. In fact the biased N-gram models were initially from Kaldi scripts for this cleaning purpose.

7.3.3 Better parameter estimation

While boosting parameter of the miscue tolerant finite state language models could be trained by cross validation, the individual weights of each miscue path used ad hoc values in this thesis. The 10-fold cross validation was too computationally expensive to search for optimal values.

Fortunately the individual weights could be estimated from their occurrence frequencies in a training set. Robust estimation probably requires more data than is available currently.

7.3.4 Reading practise application

The high school students should be able to practise on a computerised spoken language interface. Reading for the automatic speech recogniser in a browser application would be a simple exercise. It lacks the communicative aspects of spoken language, but it could help learning on some level. It would also teach the student about speaking to automatic speech recognition and it would hopefully build trust in the automatic speech processing tools. If the student permits it, their audio data could also be stored for acoustic model training.

The recogniser could track the student's speech in real time with miscue tolerant language models. This could encourage the reader. The inter-word latency score mentioned in section 5.1 could provide some quantified feedback for the student, and once the automatic articulation analysis is available it should be integrated into the system too.

7.3.5 Better miscue detection

Miscue detection should be improved. Better parameter estimation for the miscue tolerant finite state language models should help. Unfortunately improving the language model can only go so far toward better miscue detection.

I think improving the acoustic information should be the basis for good miscue detection. In the end miscues remain the unexpected event, so the recognition system has to rely on the acoustic information to detect them. In the DigiTala project, improving the recording conditions could provide substantial improvements in the acoustic information. More in-domain acoustic training data is also needed. If spoken tests are also introduced to the high school language courses, the data gathered there could be used for acoustic model adaptation.

8 Conclusions

Computerisation opens up the possibility of adding an oral test to foreign language matriculation exams. The high stakes nature of the matriculation examinations means whatever is assessed in them, is also valued by the students. Project DigiTala has created a plausible prototype for the oral test, but research in speech processing tools is vital for making the massive work load of scoring the tests manageable. That research will come in small yet significant steps.

The footprint of this thesis is a method for transcribing and segmenting read prompts. It is to be used as a preprocessing step in automatic articulation analysis, possibly followed by automatic score prediction. The word error rates on the DigiTala test data suggest both the biased N-gram models and the miscue tolerant finite state language models were successful. The miscue tolerant finite state models appear the best choice for this task: they achieved the lowest word error rate and they are tailor-made so they are amenable to further development.

Along the way two other discoveries were made. Firstly the miscue tolerant finite state models are with trivial changes able to tag miscues in transcripts produced by any method. This is useful for feedback. Secondly miscues can be simulated given just recordings and a time aligned transcript.

Some errors are inevitable in automatic speech recognition, but how much is acceptable is hard to answer. It depends on the following processing steps, on the robustness of the automatic articulation analysis and the stability of the prediction of scores. An upper limit for an acceptable word error rate could be found by characterising the effect of a one word error in the predicted score and setting a maximum score error. However the preprocessing, articulation analysis and score prediction should be optimised jointly.

Project DigiTala suffers from a chicken-and-egg problem: once the computerised tests are rolling, they will continuously provide more data for research and development, but to get them rolling more data is needed. Once more data is available the tests in this thesis should be repeated. More data would also enable better tuning of the parameters of the tested models.

To get to the tangible results, I studied automatic speech recognition, finite state models and a psycholinguistic model of reading. Throughout automatic speech recognition, I ran into data sparsity. There is always a model that can benefit from more data. The real challenge is to do as much as possible with as little data as possible. Finite state transducers are truly fascinating. They can represent complicated relationships so compactly, that it is easy to run out of short term memory trying to reason about them. Peeking inside an unfamiliar field of study was captivating. The challenge is missing all of the sense of context that comes with years of experience. Fortunately one thing a university teaches is the ability to read scientific material. In the end it is just similar curious heads beating against different walls.

What is assessed becomes what is valued. This statement about washback rings true in a more general sense. I think the common danger is that what is easy to assess becomes what is assessed. In this particular thesis the quality of temporal alignments is seen as difficult measure. The effect of the temporal segmentation on the articulation analysis might be surprisingly large and in the future it should be quantified. As for a wider perspective, in all engineering tasks it is good practice to quantify progress and set measurable goals to strive for.

Let us make sure the goals are not those that are the easiest to measure, but those that are the most meaningful.

References

[1] L. Cheng, “Washback or backwash: A review of the impact of testing on teaching and learning.” Education Resources Information Center, 2000.

[2] E.-R. Pirhonen and co., “Lukiokoulutuksen suullisen kielitaidon arviointityöryhmän muistio.” https://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/80105/tr26.pdf?sequence=1, 2006.

[3] R. Karhila, A. Rouhe, P. Smit, A. Mansikkaniemi, H. Kallio, E. Lindroos, R. Hildén, M. Vainio, M. Kurimo, *et al.*, “Digitala: An augmented test and review process prototype for high-stakes spoken foreign language examination.,” in *INTERSPEECH*, pp. 784–785, 2016.

[4] T. F. Ministry, “Gaudeamus igitur - ylioppilastutkinnon kehittäminen.” http://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/79746/OKM16_2017.pdf?sequence=1, 2017.

[5] A. Rouhe, R. Karhila, P. Smit, and M. Kurimo, “Reading validation for pronunciation evaluation in the digitala project,” in *Interspeech*, 2017.

[6] O. Aaltonen, R. Aulanko, A. Iivonen, A. Klippi, M. Vainio, L. Alivuotila, P. Eskelinen-Rönkä, M. Lehtinen, and H. Ylönen, “Puhuva ihminen,” *Puhetieteiden perusteet. Helsinki: Otava*, 2009.

[7] A. W. Bronkhorst, “The cocktail party phenomenon: A review of research on speech intelligibility in multiple-talker conditions,” *Acta Acustica united with Acustica*, vol. 86, no. 1, pp. 117–128, 2000.

[8] D. Kahneman, *Thinking, fast and slow.* Macmillan, 2011.

[9] H. P. Grice, P. Cole, J. Morgan, *et al.*, “Logic and conversation,” *1975*, pp. 41–58, 1975.

[10] B.-H. Juang and L. R. Rabiner, “Automatic speech recognition–a brief history of the technology development,” *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, vol. 1, p. 67, 2005.

[11] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, *et al.*, “Deep speech 2: End-to-end speech recognition in english and mandarin,” *arXiv preprint arXiv:1512.02595*, 2015.

[12] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, “Achieving human parity in conversational speech recognition,” *arXiv preprint arXiv:1610.05256*, 2016.

[13] P. Smit, S. Virpioja, and M. Kurimo, “Improved subword modeling for WFST-based speech recognition,” in *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017.

[14] D. van Weijen, “The language of (future) scientific communication.” https://www.researchtrends.com/issue-31-november-2012/the-language-of-future-scientific-communication/, 2012. Online, accessed: 26.5.2017.

[15] M. Creutz and K. Lagus, *Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0.* Helsinki University of Technology, 2005.

[16] A. W. Black, “Perfect synthesis for all of the people all of the time,” in *Speech Synthesis, 2002. Proceedings of 2002 IEEE Workshop on*, pp. 167–170, IEEE, 2002.

[17] T. Winograd, “Understanding natural language,” *Cognitive psychology*, vol. 3, no. 1, pp. 1–191, 1972.

[18] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.

[19] E. Alpaydin, *Introduction to machine learning.* MIT press, 2014.

[20] K. Hornik, M. Stinchcombe, and H. White, “Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks,” *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[21] I. Rehbein and J. Ruppenhofer, “There’s no data like more data? revisiting the impact of data size on a classification task..” http://www.sfb632.uni-potsdam.de/~rehbein/papers/lrec2010_rehbein_ruppenhofer.pdf, 2010.

[22] B. Allison, D. Guthrie, and L. Guthrie, “Another look at the data sparsity problem,” in *International Conference on Text, Speech and Dialogue*, pp. 327–334, Springer, 2006.

[23] R. Kohavi *et al.*, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Ijcai*, vol. 14, pp. 1137–1145, Stanford, CA, 1995.

[24] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*, vol. 247. Springer Science & Business Media, 1994.

[25] M. Gales and S. Young, “The application of hidden markov models in speech recognition,” *Foundations and trends in signal processing*, vol. 1, no. 3, pp. 195–304, 2008.

[26] S. Young, “A review of large-vocabulary continuous-speech,” *IEEE signal processing magazine*, vol. 13, no. 5, p. 45, 1996.

[27] E. Lombard, “Le signe de l'elevation de la voix [11],” *Ann. Maladies Oreille, Larynx, Nez, Pharynx*, vol. 37, no. 101-119, p. 25, 1911.

[28] F. Zheng, G. Zhang, and Z. Song, “Comparison of different implementations of mfcc,” *Journal of Computer science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[29] H. Wakita, “Normalization of vowels by vocal-tract length and its application to vowel identification,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 2, pp. 183–192, 1977.

[30] M. J. Gales, “Maximum likelihood linear transformations for hmm-based speech recognition,” *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998.

[31] T. Ko and B. Mak, “Eigentriphones: A basis for context-dependent acoustic modeling,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4892–4895, May 2011.

[32] D. Povey, L. Burget, M. Agarwal, P. Akyazi, F. Kai, A. Ghoshal, O. Glembek, N. Goel, M. Karafiát, A. Rastrow, R. C. Rose, P. Schwarz, and S. Thomas, “The subspace gaussian mixture model—a structured model for speech recognition,” *Computer Speech & Language*, vol. 25, no. 2, pp. 404 – 439, 2011. Language and speech issues in the engineering of companionable dialogue systems.

[33] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, “Do we need hundreds of classifiers to solve real world classification problems,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[34] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, Nov 2012.

[36] A. Graves and N. Jaitly, “Towards end-to-end speech recognition with recurrent neural networks.,” in *ICML*, vol. 14, pp. 1764–1772, 2014.

[37] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, “Purely sequence-trained neural networks for asr based on lattice-free mmi.,” in *INTERSPEECH*, pp. 2751–2755, 2016.

[38] J. T. Goodman, “A bit of progress in language modeling extended version,” 2001.

[11]Original article unavailable, see `http://paul.sobriquet.net/wp-content/uploads/2007/02/lombard-1911-p-h-mason-2006.pdf` for a translation

[39] C. D. Manning and H. Schütze, *Foundation of Statistical Natural Language Processing.* MIT Press, 1999.

[40] F. Stanley and J. Goodmani, “An empirical study of smoothing techniques for language modeling,” *Computer Speech and Language*, vol. 13, pp. 359–394, 1999.

[41] G. Saon and J.-T. Chien, “Large-vocabulary continuous speech recognition systems: A look at some recent advances,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 18–33, 2012.

[42] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, “Extensions of recurrent neural network language model,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5528–5531, May 2011.

[43] S. Ruan, J. O. Wobbrock, K. Liou, A. Ng, and J. Landay, “Speech is 3x faster than typing for english and mandarin text entry on mobile devices,” *arXiv preprint*, vol. arXiv:1608.07323 [cs.HC], 2016.

[44] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[45] M. Bisani and H. Ney, “Bootstrap estimates for confidence intervals in asr performance evaluation,” in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP’04). IEEE International Conference on*, vol. 1, pp. I–409, IEEE, 2004.

[46] D. B. Paul and J. M. Baker, “The design for the wall street journal-based csr corpus,” in *Proceedings of the workshop on Speech and Natural Language*, pp. 357–362, Association for Computational Linguistics, 1992.

[47] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, “The kaldi speech recognition toolkit,” in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.

[48] Y. A. Feldman in *Encyclopedia of Computer Science and Technology* (A. Kent and J. G. Williams, eds.), ch. Finite-State Machines, pp. 73–104, CRC Press, 1992.

[49] M. Mohri, F. Pereira, and M. Riley, “Speech recognition with weighted finite-state transducers,” in *Springer Handbook of Speech Processing*, pp. 559–584, Springer, 2008.

[50] T. Hori and A. Nakamura, “Speech recognition algorithms using weighted finite-state transducers,” *Synthesis Lectures on Speech and Audio Processing*, vol. 9, no. 1, pp. 1–162, 2013.

[51] K. S. Goodman, "Miscues: Windows on the reading process," in *Miscue Analysis: Applications to Reading Instruction.*, ERIC, 1973.

[52] K. S. Goodman and C. L. Burke, "Theoretically based studies of patterns of miscues in oral reading performance. final report.," 1973.

[53] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 5206–5210, IEEE, 2015.

[54] J. Mostow, "Why and how our automated reading tutor listens," in *Proceedings of the International Symposium on Automatic Detection of Errors in Pronunciation Training (ISADEPT)*, pp. 43–52, 2012.

[55] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, vol. 4783 of *Lecture Notes in Computer Science*, pp. 11–23, Springer, 2007. `http://www.openfst.org`.