Contents

Symbols and abbreviations

Symbols

$\boldsymbol{\Delta}$	delta feature vector
$\boldsymbol{\mu}$	mean vector of a GMM
$\boldsymbol{\Sigma}$	covariance matrix of a GMM
$\boldsymbol{o}_t$	observation, i.e., feature vector, at time t
$\boldsymbol{w} = w_1, ..., w_N$	a sequence of words or other units of text

Abbreviations

AM	acoustic model
ANN	artificial neural network
ASR	automatic speech recognition
BERT	bidirectional encoder representations from transformers
BPE	byte pair encoding
CER	character error rate
CMN	cepstral mean normalisation
CMVN	cepstral mean and variance normalisation
DNN	deep neural network
EM	expectation maximisation
fMLLR	feature space maximum likelihood linear regression
FSA	finite-state acceptor
FST	finite-state transducer
GLUE	the general language understanding evaluation
GMM	Gaussian mixture model
GRU	gated recurrent unit
HMM	hidden Markov model
LDA	latent Dirichlet allocation
LDA	linear discriminant analysis
LF-MMI	lattice-free MMI
LM	language model
LSTM	long short-term memory
LVCSR	large vocabulary continuous speech recognition
MAP	maximum a posteriori
MCE	minimum classification error
MFCC	mel-frequency cepstral coefficient
ML	maximum likelihood
MLE	maximum likelihood estimation
MLLR	maximum likelihood linear regression
MLLT	maximum likelihood linear transformation
MLP	multilayer perceptron
MMI	maximum mutual information

MPE	minimum phone error
NCE	noise contrastive estimation
NLP	natural language processing
NLU	natural language understanding
NN	neural network
NNLM	neural network language model
OOV	out of vocabulary
PDF	probability density function
PPL	perplexity
ReLU	rectified linear unit
RNN	recurrent neural network
SAT	speaker adaptive training
SD	speaker dependent
SI	speaker independent
TDNN	time delay neural network
VAD	voice activity detection, here synonymous with speech activity detection
VT	Viterbi training
VTLN	vocal tract length normalisation
WER	word error rate
WFST	weighted finite-state transducer

1 Introduction

1.1 The ASR task

Automatic speech recognition (ASR) is the task of converting speech into text. The direct application of ASR is useful in many situations, for example to transcribe patient notes dictated by medical doctors. ASR has also an increasing number of use cases in systems with longer pipelines that achieve some speech-initiated task, such as *voice user interfaces* or *speech-to-speech translation* systems. The ASR piece can often be a bottleneck in the pipeline, determining to a large degree the accuracy of the whole system.

In the conventional ASR system, the task is divided into subtasks. The first subtask, called feature extraction, is to divide the audio signal into segments, and convert the segments into feature vectors, also called observations. The observations are a compressed representation of the audio signal. The two most significant subtasks are *acoustic modelling* and a *language modelling*. A language model (LM) generates an *a priori* probability distribution over possible word sequences. For example, the transcription "the god of thunder was Zeus" should probably be assigned a larger probability than "the god of thunder was juice" even before any speech audio is processed. An acoustic model (AM) outputs likelihoods of observations conditional on phoneme sequences. The phoneme sequences are mapped to words by a *lexicon*, after which these a posteriori probabilities can be combined with the a priori probabilities of the language model, yielding an estimation of the optimal transcription for a given speech audio.

Acoustic and language modelling are achieved using machine learning models, primarily deep neural networks (DNNs), which are estimated based on training data. Modelling acoustics requires a parallel corpus of speech and correct transcriptions, whereas modelling language only requires text.

1.2 Spontaneous, conversational Finnish

The difficulty of ASR depends on how varied and noisy the speech audio signals are. The confined problem of recognising a few different words pronounced clearly by one speaker was essentially solved years ago. However, speech recognition becomes more difficult when the speech is continuous and recorded in differing noise conditions from many speakers. Current state-of-the-art ASR systems are nearing human-level recognition accuracy also in the *large vocabulary continuous speech recognition* (LVCSR) task if the speech is planned and pronounced clearly, as it is, for example, in broadcast news or parliament sessions. Yet, spontaneous, informal conversations remain a challenging type of speech to recognise, and the gap between human- and machine-level accuracy is still large for this type of speech.

The difficulty depends also on the language. The most obvious factor is the availability of training data. The state-of-the-art ASR systems are based on DNNs that require large training data sets. For languages such as Finnish, the resources are more limited than for the most widely spoken languages in the world, which makes

the ASR task harder. Thus, achieving accurate ASR for small languages will require more work.

Some inherent idiosyncratic properties of Finnish should also be taken into account when developing a Finnish ASR system. Finnish is a morphologically rich language. Suffixes and other conjugations perform grammatical functions, such as cases, which in other languages such as English would be denoted by separate words. This makes the number of word types in the vocabulary large, requiring not only a considerable amount of computational resources but also much text for training the LM. This problem can be avoided by segmenting words into smaller units, referred to as *subwords* (Hirsimäki et al., 2006).

However, the morphology of colloquial spoken Finnish often differs from that of the formal language, with some of the suffixes and other inflections often being omitted or changed to an incorrect one. For example, it is common to replace first person plural inflections with the passive voice verb inflection ("me ollaan" instead of the correct form "me olemme") or to use the incorrect singular form when the third person plural form should be used ("ne on" instead of the correct "ne ovat"). Other common characteristics of informal Finnish include shortening words (e.g., from "minä" to "mä") and/or combining words (e.g., "oliko se" to "olikse"). The differences between formal and informal Finnish pose difficulties when modelling informal language: text from formal sources such as books or newspapers does not resemble colloquial Finnish very closely. However, another relevant feature of the Finnish language is its phonemic orthography, i.e., the fact that one letter generally corresponds to one phoneme, and vice versa. As a result, it is possible to also *write* colloquial Finnish as it is spoken. It is therefore possible to find written text that resembles the spoken language, and use this to model colloquial Finnish. The above mentioned incorrect inflections are examples of informal Finnish that is also written in the incorrect way, as it is pronounced.

Colloquial Finnish is written, for example, on online forums, especially in direct-message conversations. Enarvi et al. (2013) collected a conversational Finnish text corpus from Internet forums by searching for key phrases that indicate informal conversations.

1.3 The research questions and scope of the thesis

The purpose of this thesis is to improve the ASR accuracy for the spontaneous, conversational DSPCON corpus. The aim is to evaluate recent acoustic and language modelling methods on the data and assess whether they can improve the ASR accuracy. Furthermore, the ASR pipeline, which includes the acoustic models and language models built with different toolkits, is rebuilt using the newest versions of the toolkits. For example, utilising the latest Kaldi recipes for acoustic modelling improves the ASR accuracy compared to older recipes.

In the previous decade, i-vectors (Dehak et al., 2010) have been used to model speaker and channel variability. I-vectors can be added to the features, providing information that enables the ASR system to adapt to differing channels and speakers during both training and testing. Snyder et al. (2018) described how also DNNs can

be utilised to create speaker embeddings, called x-vectors, which model differences between speakers. In this work, the effects of i-vectors and x-vectors on the ASR accuracy are evaluated and compared on the conversational Finnish data. The experiments aim to determine whether one of the methods is more suitable for this type of data.

Vaswani et al. (2017) introduced a neural network architecture for language modelling, called the Transformer, which is based solely on (self-)attention mechanisms (Bahdanau et al., 2014). Since then, the state-of-the-art language models have been of the transformer model type for many language processing tasks. In this thesis, Transformer-XL (Dai et al., 2019) language models are trained and evaluated on the conversational Finnish data. The goal is to determine whether the transformer models achieve better results than the RNN models on the conversational Finnish data. Other language modelling experiments in this work include tuning the hyperparameters and comparing word and subword vocabularies for n-gram LMs, RNN LMs and transformer-XL LMs.

1. Can the recognition accuracy be improved using new Kaldi recipes?
2. i-vectors vs x-vectors:
    - Can x-vectors achieve better results than the previously used i-vectors?
    - Do extractors pre-trained on a different language (English) with more data improve the ASR accuracy?
    - Is it beneficial to concatenate i- and x-vectors
3. RNN vs Transformer language models:
    - Can transformer models achieve better results than the RNN models on the conversational Finnish data?
    -

The background theory of the utilised acoustic and language modelling methods is described in three separate chapters. Chapter 2 defines some of the main concepts of machine learning as well as the basic structure of an ASR system. Chapter 3 describes the relevant language modelling techniques: the n-gram, RNN, and transformer language models. Acoustic modelling and the Kaldi toolkit are described in Chapter 4. Chapter 5 presents the experiments and results. Chapter 6 discusses the results and describes possible future work on the subject. Chapter 7 concludes the thesis.

2 Machine learning

A *model* is a simplified or abstract representation of a phenomenon, an artefact that aims to hold relevant and distilled information about its real-world counterpart. The term is very general: there are physical, statistical, and conceptual models, just to name a few different types. Brains, too, build models, conscious and subconscious, of phenomena that they encounter in their environment, which is the basis of how animals are able to perceive and act in an environment. Language is a prominent phenomenon in the environment that humans live in, and the human brain naturally develops models to process and generate language. Humans learn a language gradually through exposure to it. Similarly, to build a statistical model of a language or its pronunciation, machine learning algorithms are used. This chapter briefly introduces some of the main concepts that are central to the language and acoustic modelling algorithms described respectively in the next two chapters.

A mathematical model is often formulated as a function that generates an output given an input, emulating how the real-world phenomenon behaves. More generally, a model can be any kind of mapping from a domain to another. The acoustic and language models used in ASR are formulated as probabilistic models that output probabilities for a given input: how probable is a sequence of words, or how likely is a sequence of audio observations conditional on a particular phoneme sequence.

There are two directions, ways to create a model of a phenomenon: top-down or bottom-up[1]. Designing from top down means roughly to analyse the task on a high-level and deducing from that how the model should function in the lower levels. For example, a chess program can be programmed top-down using if-else clauses, e.g. "if pawn is in D7 and D8 is empty, move pawn to D8". To build a competent chess program this way requires a good understanding of what moves are beneficial. Top-down design of models is an exclusively human undertaking. A bottom-up generation of a model is the more usual way models come to being. In this approach, the abstract model is estimated from data gathered from the manifest world. In the chess example, the designing work can be delegated to an algorithm that performs bottom-up estimation of the desired behaviour using machine learning. In practice, the program learns by trial and error, preferring (reinforcing) those moves that have preceded winning. When the actual decisions of moves are left for the algorithm, the programmer herself need not understand the game. By merely defining the learning algorithm and leaving the model to teach itself, the model can eventually surpass the human-level competence, as has happened with most board games, among other tasks. Some tasks are inherently so complex that no one has seriously tried to build rule-based models of them. As the behaviour to be modelled becomes too complex and non-linear for a human to design it, the only way is to teach it to the machine by brute-force trial and error.

[1] In mathematics, solving a function in a top-down fashion is called an analytical or a symbolic solution, and bottom up a numerical solution.

2.1 Fitness criterion

The mapping from input to output is determined by the parameters θ that are learned. There are different approaches to defining how good a model is, given a training dataset. Using the maximum likelihood estimation (MLE), the parameters are adjusted to maximise how probable the model judges the data set. The maximum likelihood estimate θ_{ML} of the model parameters is the point in the parameter space that maximises the likelihood of the observed data. The confidence of how good the point estimate is is related to its variance. If alternative samplings of the observations are likely to change the estimate, the variance is larger. In contrast to the MLE approach, the Bayesian approach is to generate a full probability distribution function for θ. When the model is used to estimate the probability of an observation, the distribution is integrated over the parameter space, and the probabilities of the observations given by each parameter point estimate is weighted by how probable the parameter point is. This way the confidence on the parameter estimate is incorporated in the data likelihood. However, using a full probability distribution instead of a point estimate is computationally expensive, and therefore the MLE method is the more commonly used approach in machine learning.

2.2 Optimisation

Depending on the criterion for an optimal model, a optimisation method is chosen and applied to tune the free parameters of a model. When the model to be estimated includes latent variables, the maximum likelihood estimate can be computed using the expectation maximisation (EM) algorithm. A latent (i.e., hidden) variable is a variable that is not directly observed, but can affect the observed variables. A general example of a latent variable is some phenomenon in the real world that is not directly observed, but whose properties are inferred from measurements that are observable. Two types of models that include latent variables are GMMs and HMMs, both of which are central to the conventional implementation methods in automatic speech recognition. In GMMs, the identity of the mixture component c of a data point o is a latent variable, and the probability distribution $P(o)$ is determined by the joint distribution $P(o, c) = P(o|c)P(c)$. Hidden Markov models are named after the latent, hidden sequence that generates an observed sequence[2]; in speech recognition the hidden sequence can be a sequence of phones that generate the sequence of processed audio signal chunks referred to as features.

Maximising the expectedness of the data can be turned around and seen as minimising how surprising the training data are to the model by modifying the model. Expectation maximisation is an iterative method to finding a local maximum for the likelihood of a set of data given a model.

The EM algorithm consists of iterating the expectation step (E-step) and the maximisation step (M-step). The E-step

[2]They are also named after the Russian mathematician Andrey Markov.

2.3 Artificial neural networks

Artificial neural networks (ANN) are a family of model types, historically based on the concept of a perceptron (Rosenblatt, 1957), which functions similarly to a biological neuron, and is therefore called an artificial neuron, or just neuron. The core functioning, common to the perceptron and the biological neuron, is that an input either activates the output or not based on an activation function. In the case of a perceptron, the activation function is a unit step function: if the input is above a scalar threshold the output is activated. Perceptron learns a binary classifier by tuning this threshold based on information from training examples. The idea of a perceptron has since been generalised and extended into the wide range of different kinds of artificial neural networks, but the core idea of a neuron remains.

A multilayer perceptron (MLP) stacks neurons in multiple layers that are connected in the forward direction from the input to the output layer. An MLP includes an input layer of neurons with linear or non-linear activation functions, one or more hidden layers, and an output layer. The layered network learns the weights of the connections through backpropagation.

An important class of ANNs are deep neural networks (DNN), called "deep" because they include many hidden layers. MLPs are a subclass of DNNs; other subclasses include, for instance, convolutional NNs (CNN) and recurrent NNs (RNN), which are particularly useful in speech recognition since they are capable of modelling long-term time dependencies in data.

2.4 Machine learning for speech recognition

The first ASR systems were developed in the 1960s. These machines could recognise a few words, for instance the digits, by matching templates of the words with the data. In the 1970s and 80s, the HMMs were demonstrated useful in ASR, after which HHM-based systems have remained as the state-of-the-art technology, until the breakthrough of the neural models during the previous decade.

In the conventional ASR system, the task is divided into subtasks. The first subtask, called feature extraction, is to divide the audio signal into T segments, and convert the segments into feature vectors, also called observations, $\boldsymbol{O} = \boldsymbol{o}_1, ..., \boldsymbol{o}_T$. The observations are a compressed representation of the audio signal. The task is then to find $\text{argmax}_{\boldsymbol{w}} P(\boldsymbol{w}|\boldsymbol{O})$, where $\boldsymbol{w} = w_1, ..., w_N$ is a word sequence. This probability is not practicable to compute directly, but by Bayes' rule it can be expanded to

$$\underset{\boldsymbol{w}}{\text{argmax}}\, P(\boldsymbol{w}|\boldsymbol{O}) = \underset{\boldsymbol{w}}{\text{argmax}}\, \frac{P(\boldsymbol{w})P(\boldsymbol{O}|\boldsymbol{w})}{P(\boldsymbol{O})} = \underset{\boldsymbol{w}}{\text{argmax}}\{P(\boldsymbol{w})P(\boldsymbol{O}|\boldsymbol{w})\} \tag{2.1}$$

The probability of the observations $P(\boldsymbol{O})$ in the denominator is not relevant in finding the best transcription ($\text{argmax}_{\boldsymbol{w}}$) for the observations, which leaves the product in the numerator to be estimated. This product includes the two most significant subtasks: *acoustic modelling* and a *language modelling*. A language model (LM) generates an *a priori* probability distribution $P(\boldsymbol{w})$ over possible word sequences. For example,

the transcription "the god of thunder was Zeus" should probably be assigned a larger probability than "the god of thunder was juicy" even before any speech audio is processed. An acoustic model (AM) outputs likelihoods of observations conditional on phoneme sequences. The phoneme sequences are mapped to words by a *lexicon* (also called a *pronunciation dictionary*), yielding $P(\boldsymbol{O}|\boldsymbol{w})$. To avoid numerical underflow, the probabilities are converted to the logarithmic domain. A scalar weight λ is added to determine how significant the LM probabilities are compared to the AM probabilities.

$$\underset{\boldsymbol{w}}{\operatorname{argmax}}\{P(\boldsymbol{w})^{\lambda}P(\boldsymbol{O}|\boldsymbol{w})\} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\{\lambda \log\{P(\boldsymbol{w})\} + \log\{P(\boldsymbol{O}|\boldsymbol{w})\}\} \tag{2.2}$$

Acoustic and language modelling are achieved using machine learning models, primarily deep neural networks (DNNs), which are estimated based on training data. To model acoustics a parallel corpus of speech and text is needed, whereas to model language only text is needed.

In the past few years, end-to-end (E2E) speech recognition systems have achieved promising results (e.g., by Hannun et al. (2014); Chan et al. (2016)). An E2E system dispenses with the division to an LM and an AM, and instead learns a mapping from (preprocessed) audio straight to the transcription. This makes the training procedure simpler since only one model is trained instead of multiple. However, it has been shown that E2E models can still benefit from, for example, incorporating an external language model (Toshniwal et al., 2018) or speaker embeddings (Rouhe et al., 2020), into an E2E system, making it arguably no longer a pure E2E model, depending on how "E2E" is defined. Results such as these indicate that pure E2E systems will not completely supplant conventional ASR systems, or systems that include multiple separately trained models, any time soon, although E2E systems benefit from the simplified training procedure. The state-of-the-art results are still obtained with the conventional systems in many ASR tasks, and for this reason, this thesis explores methods in this conventional paradigm.

3 Statistical language modelling

A language model defines the probability distribution $P(\boldsymbol{w})$, from Eq. 2.1, over word sequences $\boldsymbol{w} = w_1, ..., w_N$. Typically, $P(\boldsymbol{w})$ is calculated as the product of the constituent word probabilities:

$$P(w_1, ..., w_N) = \prod_{i=1}^{N} P(w_i|context) \tag{3.1}$$

What constitutes the context depends on the method. The constant-order n-gram model uses a simple context of $n-1$ previous tokens. In more involved models, the context can also be of variable length and include subsequent tokens (bi-directional context). Often the context encompasses the all the previous words $w_1, ..., w_{i-1}$ in the sequence.

In statistical language modelling, the distribution is estimated based on statistics drawn from a training corpus. The statistical approach is in contrast with linguistically motivated models of language that take into account, for example, the grammaticality of a string when determining its probability.

3.1 Choice of vocabulary

A language model defines a set of units that the output sequence can include, called the vocabulary. One natural choice of vocabulary is to include in it the word types that appear in the training corpus. However, using smaller units has some benefits over a word vocabulary. If the words are segmented into smaller pieces, or all the way to individual characters, the vocabulary is smaller. This decreases computation and memory requirements for neural language models (see Section 3.3), for instance. Another benefit of using a *subword* vocabulary is that words absent in the training corpus can possibly be composed of the subword units, which means that the out-of-vocabulary (OOV) rate is smaller. The downside of subword vocabularies is that each sentence includes a larger number of units to be processed. The number of units in the context should therefore be larger if the units are shorter. This creates difficulties especially with n-gram language models (Section 3.2), because the number of possible n-grams increases exponentially w.r.t. n. NNLMs are better able to model long contexts, and the benefits of using shorter units typically outweigh the costs, which is why modern SOTA NNLMs, such as BERT (see Section 3.3.2), often use subword units.

The segmentation is an minimisation problem with a trade-off between two desiderata: a small vocabulary size and a small number of units in the corpus. As one of these decreases, the other increases (as a general rule), so an optimal compromise ought to be sought. When segmenting words into subword units it is therefore important to take into account the frequency of a character string or word. In general, frequent character strings should be added to the vocabulary so that the number of units in the corpus decreases.

Byte pair encoding (BPE) (Gage, 1994) is a compression method that can be, and is commonly, applied to segmenting words into smaller pieces (Sennrich et al., 2015).

The algorithm first divides the training corpus into characters. It then combines the two characters or subwords that are most frequently adjacent and combines these into a new subword. The algorithm iterates this for a number of times to generate the vocabulary that minimises the size of the training corpus given the size of the vocabulary.

The Morfessor family of algorithms (Creutz and Lagus, 2002, 2007) segments words in a more linguistically-motivated way. The segmenting model learns an optimal way to segment words into subwords based on a criterion, for example the maximum likelihood of the training data given the model.

3.2 n-gram language models

An n-gram language model bases the prediction of the last token in a sequence of n tokens on statistics gathered from a training corpus. The probability of a word sequence is the product of the probabilities of the constituent words, and the probability of each word is conditional on $n-1$ previous words (i.e., the context). The Eq. 3.1 becomes

$$P(w_1, ..., w_N) = \prod_{i=1}^{N} P(w_i | w_{i-(n-1)}, ..., w_{i-1}) \tag{3.2}$$

A simple method to define the probability of an n-gram is to let the probability of each word be its normalised frequency in the context:

$$P(w_i | w_{i-(n-1)}, ..., w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, ..., w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, ..., w_{i-1})} \tag{3.3}$$

This is called the maximum likelihood estimate of $P(w_i | w_{i-(n-1)}, ..., w_{i-1})$ since it maximises the likelihood of this specific training data set (Chen and Goodman, 1998). As in machine learning in general, however, the aim is to use the training data to distil from it a generalising model which predicts patterns also in previously unseen data instead of building a model that maximally accounts for the training data set.

To improve the predictions, *smoothing* can be applied to the simple occurrence counts by redistributing probability mass from the most common n-grams to the less common ones, i.e., *discounting* the most frequent n-grams. Low occurrence counts become a problem especially when n is high, since the number of possible n-grams increases exponentially as n increases. If there is an insufficient number of examples of the n-grams of the desired order, the information of lower-order n-grams (unigrams,...,(n-1)-grams) can be used; the model can *backoff* to the lower orders to make the probability distribution smoother. The lower-order n-gram scores can also be *interpolated* with the scores of the n-grams of the nominal order n.

Backing off to lower orders revives problems that motivated using higher-order n-grams in the first place. One of them is that of two words that are equally frequent, and thus have the same unigram probability, one may have a very specific kind of context in which it almost always appears whereas the other appears in various contexts. Given a novel context, which is why backing off to unigrams is necessary,

the former is statistically less probable to appear in it than the latter, but this is not captured by unigram statistics. Kneser and Ney (1995) introduced a smoothing method, which has become commonly used, where the number of different bigram contexts of a word correlate with the unigram backoff probability. The number of seen bigram types where the word w is the latter word, i.e., the number of types of the previous token w' that at least once precede w in the corpus, can be expressed as $|\{w' : \text{count}(w', w) > 0\}|$. This count is normalised by the number of all word types that are seen as the first word of a bigram to get the KN unigram probability:

$$P_{\text{KN}}(w) = \frac{|\{w' : \text{count}(w', w) > 0\}|}{\sum_{w''} |\{w' : (w'w'') > 0\}|} \tag{3.4}$$

For a Kneser-Ney smoothed bigram model, the unigram and bigram statistics are then interpolated:

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(\text{count}(w_{i-1}w_i) - d, 0)}{\text{count}(w_{i-1})} \lambda(w_{i-1}) P_{\text{KN}}(w_i) \tag{3.5}$$

where d is a discount constant, usually $0 < d < 1$, and λ is a normalising factor that defines how the discounted probability mass is redistributed:

$$\lambda(w_{i-1}) = \frac{d}{\sum_{w'} \text{count}(w_{i-1}, w')} |\{w' : \text{count}(w_{i-1}, w') > 0\}| \tag{3.6}$$

The discount is normalised by the sum of the counts of all bigrams where the w_{i-1} is the first word, and the normalised discount is multiplied by the number of word types that have followed w_{i-1}, i.e., the number of word types that have been discounted, so that $P_{\text{KN}}(w_i|w_{i-1})$ over all w_i equals to one.

The bigram formulation can be generalised to higher-order n-grams:

$$\begin{aligned} P_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = & \frac{\max(\text{count}_{\text{KN}}(w_{i-n+1}^{i-1}, w_i) - d, 0)}{\sum_{w'} \text{count}_{\text{KN}}(w_{i-n+1}^{i-1}, w')} \\ & + \ \lambda(w_{i-n+1}^{i-1}) P_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{3.7}$$

where w_{i-n+1}^{i-1} is the $n-1$ words before w_i and count_{KN} is the count for the highest order and the number of different words that precede the n-gram for the lower orders (Jurafsky and Martin, 2019).

In the modified Kneser-Ney smoothing, introduced by Chen and Goodman (1998), the discount constants are different for n-grams that have one, two, or more than two occurrences, changing also λ for the distribution to still sum to one. This is motivated by empirical results suggesting that the optimal d depends on the frequency.

A common approach to making an n-gram model more efficient is to *prune* n-grams that are least relevant. The simplest way to prune n-grams from the model is to use a frequency cut-off below which the n-grams are omitted from the model. Other methods include comparing the log-probability of the model and the model where an n-gram has been removed, and removing the n-gram if the difference is small (Siivola et al., 2007b).

3.3 Neural language models

n-gram language models treat each word as an element in a set with no other attributes besides an ID.

Bengio et al. (2003) proposed a method able to take into account the similarity of words, and able to generalise exploiting the similarities. Words can be similar to each other in different ways: "cat" is similar to "dog", and "cat" is similar to "whiskers", but in a different way. Different kinds of similarities can be represented as dimensions in a vector that encodes a word. By this method, words are embedded into a vector space that encodes relations between words. Similarity of words corresponds to proximity in the embedding space, and can be measured by the cosine similarity of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

$$\cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|} \tag{3.8}$$

or some other similarity metric. Being able to measure the similarity of two words allows for generalising from word sequences in the language modelling training text to sequences that have not been encountered. If an embedding space holds the information that "cat" is similar to "dog" in the particular way that "cat" is, the language model can generalise from a seen sentence "the cat is running" to a new sentence "the dog is running".

One of the seminal word embedding models was published by Mikolov et al. (2013a), called Word2Vec. The goal of this model is to be able to train word embeddings using large data sets of billions of words and to create embedding spaces that encode multiple degrees of similarity between words. Substructures of the embedding space can be probed by the word analogy task, for example asking which word is closest in the vector space when the word "man" is subtracted from "king", and the word "woman" is added. The intuitive answer "queen" is given by Word2Vec embedding space (Mikolov et al., 2013a). The paper introduced two methods for computing the Word2Vec embeddings. In both methods the idea is to train a shallow feedforward neural network with one hidden layer for a classification task and then use the learned hidden layer weights as the vector representation of the word. The Continuous Bag-of-Words Model (CBOW) learns the embeddings by predicting the current word given a few (e.g. 4) previous words and a few next words. The method uses a bag-of-words, i.e. the order of the words is not taken into account. The continuous Skip-gram model is similar to the CBOW, but instead of predicting the current word given a context, the training task is to predict the context of a given word. Using the negative sampling method (Mikolov et al., 2013b), a small subset of the words outside the context are sampled. This way the task does not require to classify all the other outside-of-the-context words as "not in the context" but only some sample in the order of 10.

Learning to embed words by an affine transform is usually incorporated into modern NNLMs as the first layer or first few layers of the network. The embedding is learned in the same training process as the language modelling task. Initialising the embedding of an NNLM from a pre-trained word embedding model is also possible.

3.3.1 Recurrence

Feedforward networks feed the output of a layer to the next layer until the output layer. The network output at time t is therefore dependent only on the input at t. In natural languages, each unit of speech or text is strongly connected to the previous units. Both language models and acoustic models ought to capture dependencies across many time steps, and RNNs have been successfully applied to these tasks. In a simple RNN, the output of a layer is fed back to the same layer in the next time step, creating a type of memory of previous states of the network. Other recurrence methods are also possible, but all RNNs are based on some type of cycle between the network connections, creating a dependence of the current output on the previous outputs. This enables modelling dependencies between time steps.

Long short-term memory
A widely used type of recurrent NN is called a long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997). The term "short-term" alludes to the activations of recurrent connections as type of memory, in contrast with "long-term" memory in the form of the weights of the connections that change by learning. The short-term memory is made longer and more expressive than in the usual RNN memory by utilising special gate layers that comprise a memory cell unit. The LSTM cell contains in total four neural network layers that perform different gating functions and interact to jointly learn which information should be memorised, i.e., which of the processed previous inputs contain the relevant *context* that the next output should depend on. The gate layers are called the forget gate, the add gate, the input gate, and the output gate. Figure 1 illustrates the LSTM cell with its four gate layers.

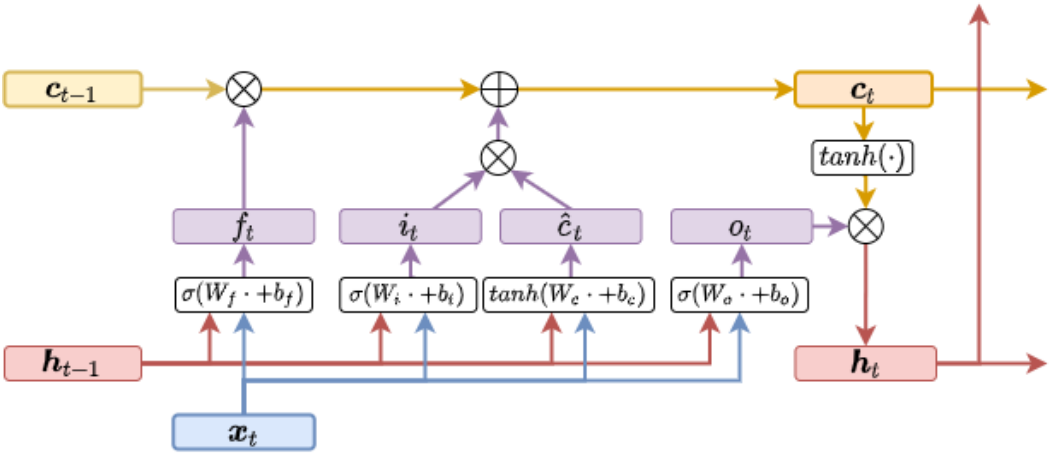

Figure 1: An LSTM cell. At time step t the cell inputs the input vector $\boldsymbol{x}_t$, the previous output (the high-level representation) $\boldsymbol{h}_{t-1}$, and the previous context vector $\boldsymbol{c}_{t-1}$.

The forget gate inputs the previous output h_{t-1} and the current input x_t and

outputs a vector that decides which elements in the previous context vector (aka cell state) c_{t-1} should be kept in the memory and how completely, on a scale from 0 to 1.

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \tag{3.9}$$
$$k_t = f_t \circ c_{t-1} \tag{3.10}$$

where $\circ$ represents the element-wise (also known as Hadamard) product of the vectors. In a first version of the LSTM published by Hochreiter and Schmidhuber (1997), there is no forget gate but a self-recurrent connection, which is equivalent to f being always unity. This is called the *constant error carousel* because it retains information from the previous cell states and alleviates the problem of vanishing or exploding gradients. Gers et al. (1999) introduced the forget gate to avoid problems caused by continual input streams that are not segmented by ends at which the LSTM state could reset. The forget gate incorporates to the LSTM cell learnable parameters whose purpose is to discard information when it becomes useless.

The add gate extracts from h_{t-1} and x_t a new candidate context vector $\hat{c}_t$ that could be added to the context in lieu of the forgot information. The hyperbolic tangent squishes the values between -1 and 1. The input gate i_t selects from $\hat{c}_t$ the new context that is added to the context.

$$\hat{c}_t = \tanh(U_c h_{t-1} + W_c x_t) \tag{3.11}$$
$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \tag{3.12}$$
$$j_t = \hat{c}_t \circ i_t \tag{3.13}$$
$$c_t = k_t + j_t \tag{3.14}$$

The output gate o_t generates the current hidden state h_t from the newly computed current context c_t:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \tag{3.15}$$
$$h_t = o_t \circ \tanh(c_t) \tag{3.16}$$

The LSTM cells can be stacked on top of each other to create a deep LSTM network. Each layer feeds the output to the input of the next layer, and the recurrent connection feeds the context vector and the output vector also to the same cell in the next time step.

LSTMs are inherently uni-directional, as the input sequence units are processed one-by-one. If an application benefits from bi-directional context, two separate cells can be used for the two directions. Figure 2 illustrates a bidirectional LSTM network.

Figure 2: A bidirectional LSTM network.

3.3.2 Attention

Bahdanau et al. (2014) developed a mechanism that has been widely adopted in speech and language processing systems, called *attention*. The term refers to a search technique in which a model learns to determine which parts of the input are most relevant for an output at a particular time step: which parts of the input should be attended to when generating an output.

Background: seq2seq encoder-decoder models
Attention was introduced in the context of machine translation with sequence-to-sequence (Sutskever et al., 2014) encoder-decoder systems. "Sequence-to-sequence" means that the system maps an input sequence to an output sequence; in machine translation these are a sentence in two different languages. "Encoder-decoder" is a neural network architecture type that first encodes the input sequence as a high-level representation and then a decodes the output using this representation vector. The encoder typically uses a feedforward neural network to project the tokens in the sequence into an embedding space, and an RNN, e.g., an LSTM network, to turn the word embedding sequence (multiple vectors) into the single context[3] vector $\boldsymbol{c}$.

Also the decoder is usually an LSTM network, followed by a softmax layer to generate the output word sequence $\boldsymbol{w} = w_1, ..., w_N$ by determining a probability distribution $p(w_i|w_1, ..., w_{i-1}, \boldsymbol{c})$ over the words given the context vector and previous outputs.

$$p(\boldsymbol{w}) = \prod_{i=1}^{N} p(w_i|w_1, ..., w_{i-1}, \boldsymbol{c}) \tag{3.17}$$

In the conventional encoder-decoder model, the context vector is the output of the encoder LSTM (or GRU) cell, as described in Section 3.3.1, after processing the last input token, and the context vector initialises the first decoder state:

$$\boldsymbol{c} = \boldsymbol{h}_U^e \tag{3.18}$$

$$\boldsymbol{h}_0^d = \boldsymbol{c} \tag{3.19}$$

where the superscripts e and d distinguish the encoder and decoder state vectors.

If the decoder is an LSTM network, the probability distribution depends on the previous output w_{i-1}, the encoder output $\boldsymbol{c}$, and the decoder LSTM state $\boldsymbol{h}_i^d$. Call this function g:

$$p(w_i|w_1, ..., w_{i-1}, \boldsymbol{c}) = g(w_{i-1}, \boldsymbol{h}_i^d, \boldsymbol{c}) \tag{3.20}$$

The decoder is therefore an *autoregressive* generator of the output sequence, generating one unit in the sequence at a time based on the previously generated units (Graves, 2013).

[3]The term "context" is used in the context LSTMs and attention; it denotes different kinds context vectors in LSTMs and in the attention mechanism.

An attentive decoder

Attention can be used in the decoder to help find the relevant information from the encoder output. Whereas the traditional decoder bases the output word probabilities on the single context vector $\boldsymbol{c}$ common to one sequence, attention computes a separate $\boldsymbol{c}_i$ vector for each time step i, and uses all of the encoder states $H^e = \boldsymbol{h}^e_1, ..., \boldsymbol{h}^e_U$ to generate the context vectors $\boldsymbol{c}_1, ..., \boldsymbol{c}_N$. The output probability distribution from Eq. 3.20 becomes

$$p(w_i|w_1, ..., w_{i-1}, \boldsymbol{c}_i) = g(w_{i-1}, \boldsymbol{c}_i, H^e) \tag{3.21}$$

The decoder LSTM network state depends on the current context vector, along with the previous decoder state and the previous output:

$$\boldsymbol{h}^d_i = f(w_{i-1}, \boldsymbol{h}^d_{i-1}, \boldsymbol{c}_i) \tag{3.22}$$

The context vector $\boldsymbol{c}_j$ is a weighted sum of annotations $\boldsymbol{h}^e_i$, which are typically the concatenated outputs of the forward and backward layers in a bi-directional LSTM network.

$$\boldsymbol{h}^e_j = [\overrightarrow{\boldsymbol{h}}^e_j; \overleftarrow{\boldsymbol{h}}^e_j] \tag{3.23}$$

The weights α_{ij} aim to capture how relevant each encoder state $\boldsymbol{h}^e_i$ is to the decoder output at the current time step. First, a scoring function gives the pair of encoder and decoder state vectors a value e_{ij} that indicates how relevant the encoder state at index j is to the decoder state at index i. This could be achieved by a simple dot product of the vectors, but a more adaptive way to assess the similarity is to learn the similarity function. The scoring function is parametrised with a weight matrix W_s:

$$score(\boldsymbol{h}^d_{i-1}, \boldsymbol{h}^e_j) = \boldsymbol{h}^d_{i-1} W_s \boldsymbol{h}^e_j \tag{3.24}$$

$$e_{ij} = score(\boldsymbol{h}^d_{i-1}, \boldsymbol{h}^e_j) \tag{3.25}$$

The scoring function is called the *alignment model* by Bahdanau et al. (2014) as it aligns the encoder states with the decoder outputs. As the encoder states are dependent on the inputs at the same time step, the alignment is between the input sequence and the output sequence. In the application of machine translation, this means an alignment of the words in the source and target languages. If the application is end-to-end ASR, the alignment would be between observations and output tokens (e.g., characters, as in the influential paper by Chan et al. (2016)). The alignment is soft, i.e., each input affects many outputs with variable weights. The scores are normalised with the usual softmax function to ensure that they are positive values which add up to one:

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})} \tag{3.26}$$

The softmax also increases the relative differences between the values, drowning the irrelevant units and emphasising the relevant ones. Finally, the context vector is calculated by the weighted sum of the encoder states:

$$\boldsymbol{c}_i = \sum_j \alpha_{ij} \boldsymbol{h}^e_j \tag{3.27}$$

Figure 3 illustrates the attention mechanism used in an encoder-decoder model.

Figure 3: Flow chart of the attention calculations.

Self-attention and transformers

The encoder-decoder described in the previous subsection applies the attention mechanism in between two recurrent neural networks. As a deviation from this prevailing method, Vaswani et al. (2017) asserted that "Attention Is All You Need", in the title of their paper describing the transformer network. The network architecture proposed in their paper evades the use of recurrence altogether, applying instead feedforward networks and *multi-head self-attention* blocks, which are briefly described in this subsection. One good reason to avoid recurrence is its inherently sequential nature: the state of an LSTM cell depends on the previous state, which means they need to be computed in succession. Attention mechanism has no such restriction, which enables parallelising the computation, decreasing significantly the required training time.

The basic attention described in the previous section aligns the input sequence with the output sequence. Self-attention, also called intra-attention, aligns the input sequence with itself, as well as with the output sequence. By relating each unit in the input sequence with the other units, the self-attention mechanism aims to compute a more informative encoding of the units. This idea is based on the fair assumption that there are dependencies across the sequence, not necessarily related to how many indices apart the dependent units are. For example, if the input sequence is "Iris was the goddess of the rainbow and the messenger of the Olympian gods while Arke, her twin sister, became the messenger of the Titans." it is probably useful to attend to "Iris" when encoding "her", to "Arke" when encoding "messenger", and to both "Iris" and "Arke" when encoding "sister". At least, this is roughly how a human reader would perceive stronger relations between some words than others. Vaswani et al. (2017) noted that the attention mechanism connects words within a sequence in a way that is usually similar to how a human reader does.

The self-attention mechanism is an interplay between three input vectors called *key*, *value* and *query*, which map to an output vector. These vectors are derived from the input token embeddings by multiplying each with a single matrix of learned parameters, i.e., performing a linear projection on the input. This typically reduces the dimensionality of the input vector to the key, value and query vectors. For each position in the input sequence, the query vector of the input at that position is paired with the keys of every other positions. For each pair, a score is calculated. The idea here is similar to the basic attention score in the previous subsection (Eq. 3.24): to determine how relevant the other input units in the sequence are when encoding this input unit. The particular implementation of self-attention described by Vaswani et al. (2017) is called scaled dot product attention, in which the score is calculated by a dot product of the key and query, scaled by the square root of the dimension d_k of the key vector. In practice, the vectors are stacked together to compose matrices,

simplifying the calculations:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.28}$$

The score is fed through a softmax function, as in Eq. 3.26. The softmax output, which can be thought of as probabilities of how much each position should be attended to, are multiplied by the value vectors of the particular positions, i.e., the value vectors are weighted by the softmaxed scores. The attention function outputs are summed together and normalised to create the encoding for the particular input unit.

The attention mechanism works nicely as is, but Vaswani et al. (2017) noted that it is beneficial to compute multiple different linear projections for the keys, values and queries, and perform the attention function separately on these different branches, or *heads*. The outputs of the heads are concatenated and a final linear projection generates the output of the multi-head attention.

Since the attention mechanism treats each position equally, an explicit representation of each position is required to model the information expressed by the order of the units in a sequence. A *positional encoding* is calculated for each unit and summed with the input embedding. Either learned or fixed positional encodings are possible, and they can be based on the absolute position in the sequence, or relative to the length of the sequence. Vaswani et al. (2017) evaluated a few positional encoding functions and ended up using a fixed function that uses the fixed positions *pos*:

$$PositinalEncoding(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{3.29}$$

where i is the dimension, and d is the number of dimensions.

The transformer network follows the encoder-decoder structure. The encoder includes six stacked 2-part layers that each include the multi-head self-attention mechanism followed by a fully-connected feedforward network. Residual connections (He et al., 2016) are added around the attention and around the feedforward layers, and the layer outputs are normalised. The residual connections require that the outputs and inputs are of the same dimensionality.

The decoder is similar to the encoder, but an additional layer is integrated to it to perform multi-head attention to the encoder output. The transformer output is also fed to a multi-head attention. This output is masked so that the subsequent positions cannot be attended to, but the units depend only on the known previous positions.

After the publication of the transformer network, its variants have been applied to many natural language processing and understanding tasks with success. Currently, the common benchmark for these kinds of general language models is the General Language Understanding Evaluation (GLUE), which includes tasks such as classifying sentence sentiment or entailment (Wang et al., 2018). GLUE aims to assess whether a system is able to extract a variety of information from text, similarly to how humans understand text by extracting meaning from it.

Radford et al. (2018) pre-trained a large transformer *generatively* on a large corpus of diverse text and applied *discriminative* fine-tuning to modify the model for a specific task with in-domain text. They call the model GPT, or generative pre-trained transformer. GPT was able to achieve SOTA results on many tasks similar to the GLUE score (which released after GPT), demonstrating the efficacy of general generative pre-training followed by task-specific fine-tuning, and further fortifying the position of transformers as a replacement of RNNs in language modelling.

Devlin et al. (2018) developed another prominent transformer system called BERT or Bidirectional Encoder Representations from Transformers. Two of the main design choices that made BERT stand out and improve upon the results of the GPT and other models were bi-directionality and multi-task learning. GPT was trained to predict subsequent words given a previous context, but BERT training utilises masking to predict words given both previous and subsequent words in the context. Masking is a deeper way to use the bi-directional context than concatenating two encodings derived from left and right context separately, as described in Section 3.3.1 with bLSTMs. Because the same encoding model can see tokens on either side of the masked token, dependencies across left and right contexts are possible to model, too. BERT was pre-trained on a second task besides the masked language modelling task. The second task was to predict whether a given sentence follows the current sentence. This task aims to train a more high-level understanding of the sentences, needed in many down-stream tasks such as question answering. This also makes the pre-training resemble the fine-tuning more, as fine-tuning could be done, for example, to train the model to answer questions. The next sentence prediction is implemented by adding a special binary classification token as the first token in the output sequence of embeddings. In other tasks, the classification embedding can be used differently. The input consists of three embedding types that are summed together: token embeddings, segment embeddings and the positional encoding. The segment embedding encodes the information which sentence of the two an input token belongs to. The sentence pair is also separated by an additional separating token. The 2-task pre-training was deemed important in the ablation experiments of Devlin et al. (2018).

While the transformer has proven more effective than RNNs for NLU tasks and supplanted RNNs in many applications of language modelling, recurrence itself has made a return of sorts. Dai et al. (2019) addressed some problems caused by dispensing with recurrence entirely and utilising attention only, pertaining to what they call *context fragmentation*. As a corpus of training text is fed to a model such as BERT, it needs to be segmented into smaller input snippets due to memory restrictions of the computation. In text, there are usually natural linguistic segments such as sentences, paragraphs and documents, but segmenting the text into these would effect segments of very uneven lengths. In practice, a corpus is usually segmented into fixed-length segments of a few hundred characters. These segments cannot conform to the natural linguistic boundaries, so the used context may lack some relevant context as, e.g., sentences are sometimes cut in half. Dai et al. (2019) proposed a remedy for the context fragmentation, incorporated in their system called Transformer-XL (extra long). Their solution is based on two modifications

of the transformer: reusing the hidden states of previous segments in a recurrent manner and using relative positional encodings instead of absolute ones. The second modification is necessary to avoid temporal confusion when reusing hidden states.

Reusing the previous hidden states is implemented by simply concatenating the hidden states of two consecutive segments. This creates a recurrence in the network whereby the utilised context extends beyond two segments, since all of the previous segments affect the current segment hidden state to some degree. One detail that makes this recurrence different from the recurrence commonly implemented in RNNs is that the reused hidden states are taken from one layer below instead of the same layer. Consequently, the longest possible dependency length increases linearly as more layers added to the network.

When the previous state is reused, the positional encoding is also carried over from the previous state. This obviously creates confusion, and the whole purpose of encoding positional information is undermined, if the usual positional encoding function (Eq. 3.29) is used. Therefore, Dai et al. (2019) use a different positional encoding scheme which avoids this problem. As noted before, in the traditional transformer, the positional encoding $\boldsymbol{u}_i$ is summed with the token embedding $\boldsymbol{x}_i$. This sum is linearly projected using a learned weight matrices W_k and W_q to respectively generate the key vector $\boldsymbol{k}$ and query vector $\boldsymbol{q}$.

$$\boldsymbol{u}_i^{\text{abs}} = PositinalEncoding(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{3.30}$$

$$\boldsymbol{q}_i = W_q(\boldsymbol{x}_i + \boldsymbol{u}_i^{\text{abs}}) \tag{3.31}$$

$$\boldsymbol{k}_j = W_k(\boldsymbol{x}_j + \boldsymbol{u}_j^{\text{abs}}) \tag{3.32}$$

And the score of a key-query pair is simply their product (omitting the scaling factor for simplicity). This product can be expanded:

$$\begin{aligned} score_{\text{abs}}(\boldsymbol{q}_i, \boldsymbol{k}_j) &= W_q(\boldsymbol{x}_i + \boldsymbol{u}_i^{\text{abs}})W_k(\boldsymbol{x}_j + \boldsymbol{u}_j^{\text{abs}}) \\ &= \boldsymbol{x}_i W_q W_k \boldsymbol{x}_j + \boldsymbol{x}_i W_q W_k \boldsymbol{u}_j^{\text{abs}} \\ &+ \boldsymbol{u}_i^{\text{abs}} W_q W_k \boldsymbol{x}_j + \boldsymbol{u}_i^{\text{abs}} W_q W_k \boldsymbol{u}_j^{\text{abs}} \end{aligned} \tag{3.33}$$

To convert the absolute positional encodings to relative, Dai et al. (2019) change the scoring function to

$$\begin{aligned} score_{\text{rel}}(\boldsymbol{q}_i, \boldsymbol{k}_j) &= \boldsymbol{x}_i W_q W_{k,x} \boldsymbol{x}_j + \boldsymbol{x}_i W_q W_{k,u} \boldsymbol{u}_{i-j}^{\text{rel}} \\ &+ \boldsymbol{v}^x W_{k,x} \boldsymbol{x}_j + \boldsymbol{v}^u W_{k,u} \boldsymbol{u}_{i-j}^{\text{rel}} \end{aligned} \tag{3.34}$$

where $\boldsymbol{u}_{i-j}^{\text{rel}}$ is a relative counterpart to $\boldsymbol{u}_i^{\text{abs}}$. When the positional encoding is relative, only the distance between i and j is taken into account. Furthermore, $\boldsymbol{v}^x$ and $\boldsymbol{v}^u$ are learnable vectors of parameters which replace $\boldsymbol{u}_i^{\text{abs}} W_q$. This is motivated by the fact that the query vector is equal (not multiplied by $\boldsymbol{x}_i$ in these terms) for all query positions, which suggests that the attentive bias towards different positions should also be equal for each query position. The weights for content $W_{k,x}$ and position $W_{k,u}$ are also distinguished from each other.

This formulation allows for an intuitive interpretation: the first term encodes content-based attention, the second term encodes content-dependent positional information, the third term encodes global content bias, and the last term encodes global positional bias (Dai et al., 2019).

4 Acoustic modelling and the Kaldi toolkit

Kaldi[4] is a toolkit for automatic speech recognition (Povey et al., 2011), used in the ASR experiments in this study. This chapter aims to give an overview of the most relevant theoretical underpinnings of Kaldi ASR systems, as well as some of the practical details of using Kaldi.

4.1 Feature extraction and feature-space transforms

A speech audio signal contains a lot of information that is irrelevant for converting the signal to text. The first step of ASR is to find the features of the signal that contain the information about what is being said. An assumption is made that the speech signal does not change meaningfully in a time frame of about 10 milliseconds so that the signal can be divided into frames with this time resolution. The frames overlap so that each frame is about 20 or 25 milliseconds, and a tapered window function, such as Hamming, is applied to (i.e., multiplied by) each frame. This window function removes the discontinuities that occur on the borders of frames, and the overlapping compensates for the tapering of the window function so that the distorting effect on the signal statistics is minimised .

Mel-frequency cepstral coefficients

The stationary frames' frequency components can be then computed with the Fourier transform. A commonly used method is to extract the MFCCs by applying a logarithmic mel-scale filterbank to the frequency spectrum, and lastly computing the DCT. The log mel-scale emphasises the lower frequencies emulating the way humans perceive sound, i.e., the way the human inner ear recognises lower frequencies with higher frequency resolution. The DCT decorrelates the coefficients so that the use of diagonal covariance matrices is possible in the subsequent stages of the modelling, namely when using GMMs to model the HMM state emissions (Section 4.2). The use of diagonal covariance matrices greatly reduces the number of free parameters, but the trade-off is that correlation between feature vector elements is not modelled.

Delta and delta-delta features

An MFCC vector encodes only the stationary frequency features of a frame. However, a speech signal varies in time, and this variation carries meaning about which phones are uttered. It is therefore useful to add information to the feature vectors about how the signal changes in time. Information about temporal change and about change of temporal change is extracted from the MFCCs by calculating the differences and second-order differences of adjacent coefficients. These features are called the delta ($\boldsymbol{\Delta}$) and delta-delta ($\boldsymbol{\Delta\Delta}$), or acceleration, features. The delta feature vector $\boldsymbol{\Delta}_t$ corresponding to the MFCC vector $\boldsymbol{c}_t$ (or the time step of that vector) is calculated by

[4] "Kaldi" is one in the series of coffee-related names for computer science projects. The Kaldi documentation explains: "According to legend, Kaldi was the Ethiopian goatherder who discovered the coffee plant."

subtracting the weighted previous vector(s) from the weighted subsequent vector(s) and normalising the sum:

$$\boldsymbol{\Delta}_t = \frac{\sum_{\theta=1}^{\Theta} \theta(\boldsymbol{c}_{t+\theta} - \boldsymbol{c}_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{4.1}$$

In Kaldi, the default window length $\boldsymbol{\Theta}$ is 2, so the $\boldsymbol{\Delta}$s are computed by multiplying the MFCCs with a sliding window of values $[-2, -1, 0, 1, 2]$ and then normalising by dividing by $2 * (1^2 + 2^2) = 10$. The $\boldsymbol{\Delta\Delta}$s are computed by applying the same method to the $\boldsymbol{\Delta}$ features. The first and last MFCCs are replicated to fill the window (Young et al., 2015).

Cepstral mean and variance normalisation

The cepstral mean normalisation (CMN) (Rosenberg et al., 1994) and cepstral mean and variance normalisation (CMVN) (Viikki and Laurila, 1998) are methods to make the features more useful in noisy conditions. In these techniques, the MFCC feature vectors are normalised to have a zero mean, and in CMVN unit variance, over a sliding finite segment. After the normalisation, clean and noisy MFCCs are more similar, which mitigates the performance reduction caused by noisy environments.

The variance of the MFCCs of a noisy speech signal is generally lower than the variance of those of a clean signal. By requiring the variance be constantly unity, noisy and clean speech MFCCs resemble each other more closely. Similarly, when noise is added to a signal, the mean changes, and by requiring the mean to be zero the characteristics of clean and noisy signals become more alike. Normalising variability between the speech signals is in general important in training an ASR system that ought to recognise different types of speech by different speakers in different recording conditions.

The topic of handling meaningless variability, i.e., variability that does not contribute to the phoneme content of the utterance, between the utterances is revisited in Section 4.5 which describes speaker adaptive training.

Dimensionality reduction and feature-space transforms

Features can be compressed by a dimensionality reduction method. Two common methods for this are the principal component analysis (PCA) (Pearson, 1901) and linear discriminant analysis (LDA), also called Fisher discriminant analysis (Martínez and Kak, 2001). The general idea of these methods is to find a linear combination of variables that best explain the observations. PCA achieves this by performing an orthogonal transformation that transforms the data matrix into a space where the dimensions, called the principal components, are ordered by their variances in decreasing order. After this, dimensionality reduction is achieved by pruning the dimensions with the lowest variance since they contribute the least to the information content of the features. LDA, on the other hand, searches for those dimensions that best discriminate between classes. For LDA, labelled training data is needed.

MLLT is a feature orthogonalizing transform that makes the features more accurately modeled by diagonal-covariance Gaussians

Differences between GMM and DNN input
GMMs are sensitive to the number of dimensions of the feature vectors: even a small increase in the vector length will increase the number of GMM parameters substantially. The usual number of dimensions used with GMMs is about 40. With DNNs, however, the input vector determines only the width of the input layer–widths of the other layers are not constrained by the dimensionality of the input features. Therefore, DNNs can learn to use longer feature vectors, and often the used number of dimensions is a few hundred (Rath et al., 2013).

4.2 Modelling phonemes with hidden Markov models

Estimating the likelihoods of observations given phonemes is achieved by creating a HMM for each phoneme. The phoneme-specific HMMs generate likelihoods of the observed sequences which can be used to map observations to phonemes. This way the task becomes to estimate the parameters of the HMMs so that each of them models the associated phoneme as accurately as possible.

A hidden Markov model consists of a hidden Markov chain, also called regime, and the observation sequence, i.e., feature vectors. Each observation $\boldsymbol{o}_t$ has a probability $b_i(\boldsymbol{o}_t)$ of being generated when a hidden state i is entered. In other words, the observation is a probabilistic function of the hidden state. A state's emission probabilities are represented by a PDF, typically a mixture of multivariate Gaussian densities

$$b_i(\boldsymbol{o}_t) = \sum_{m=1}^{M_j} c_{jm} \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) \tag{4.2}$$

where $\boldsymbol{\mu}_{jm}$ is the mean vector, $\boldsymbol{\Sigma}_{jm}$ is the covariance matrix and c_{jm} is mixture weight for mixture component m in state j. The Gaussian mixture density is

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{o}-\boldsymbol{\mu})} \tag{4.3}$$

Because a mixture of Gaussians can assume arbitrary shapes, they can model non-Gaussian phenomena; no restricting assumption is made about the shape of the PDF when a GMM is used.

HMMs are used to model sequences, but an observation is independent of past observations. Instead, the regime has a memory, although the shortest possible: the probability of being a certain state in the next time step depends only on the current state and not the previous states. This independence of the previous transitions is called the Markov assumption. Each hidden state pair, represented in Kaldi by an arc from a state to another, is associated with a transition probability a_{ij} that describes how probable it is to move from state i to state j.

All in all, a HMM is defined by the set of states $S = s_1, s_2, ...s_N$, the transition probability matrix $A = a_{11}, ..., a_{ij}, ..., a_{NN}$, the emission probabilities $B = b_i(\boldsymbol{o}_t)$, and the initial probability distribution $\pi = \pi_1, \pi_2, ...\pi_N$ that models the probability of a state being the first state in the hidden sequence.

The typical HMM topology for a phoneme is a left-to-right model, also called the Bakis model, with three emitting states that each have a transition to the next state and a self-loop. The model also includes a fourth non-emitting final state that has no outbound transitions. However, in the Kaldi chain models (see Section 4.4), the topology is reduced to have only one emitting state due to a lower time resolution used. In Kaldi, the phoneme topology is defined in the lang/topo file.

After initialising a HMM for each phoneme, the parameters, i.e. means, covariances and mixture weights need to be estimated in the process referred to as "training" of the model.

4.3 Training HMM/GMM AMs and finding the hidden state sequence

The speech recognition system is trained in a supervised manner, meaning that the training data consists of a parallel corpus of speech audio and corresponding correct transcription. However, the task of the acoustic model is not as straightforward as finding a label for an input vector[5]. The reference transcription can be of arbitrary length but the AM is required to map each observation to a HMM state, generating an equal-length *alignment* of observations and states. The states correspond to phonemes, so an alignment can be mapped to a transcription of the audio, given a lexicon that maps the phoneme sequences to word sequences.

ML is one criterion of determining the best HMM parameters; others include the MMI criterion which is discussed in Section 4.3.3. Section 2 was a brief discussion about different fitness criteria and optimisation algorithms. It was noted there that for finding the maximum likelihood estimate of a model with latent variables, expectation maximisation is a commonly used optimisation algorithm.

4.3.1 The Baum-Welch algorithm

The Baum-Welch algorithm is an expectation maximisation algorithm for estimating the HMM parameters. Here, the task is to maximise the likelihood $P(\boldsymbol{O}|M)$ of the observations $\boldsymbol{O}$ given the parameters of the HMM M. If the HMM had only one state j, the maximum likelihood estimate $\hat{\boldsymbol{\mu}}_j$ would simply be the average of the observations, and $\hat{\boldsymbol{\Sigma}}_j$, too, could be determined directly, using the covariance definition. In practice, there are many states which is why the parameters need to be estimated numerically, iteratively. However, the initial parameter values can be taken from simple statistics of the observations. Initially, the observations are divided equally between the states and the means and variances of the states are taken from the average values.

The maximum likelihood estimates for the parameters are

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} L_j(t)\boldsymbol{o}_t}{\sum_{t=1}^{T} L_j(t)} \tag{4.4}$$

[5]In End-to-end ASR the whole ASR task is simplified to outputting an arbitrary length grapheme sequence given the sequence of observed features

and

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} L_j(t)(\boldsymbol{o}_t - \boldsymbol{\mu}_j)(\boldsymbol{o}_t - \boldsymbol{\mu}_j)^\top}{\sum_{t=1}^{T} L_j(t)} \tag{4.5}$$

The numerator and denominator sums for both parameter groups are *accumulated* from the observations. This is the M-step in this expectation maximisation algorithm. The E-step includes finding the optimal alignment given the current HMM parameters. This is achieved by the forward-backward algorithm.

The state occupation probability

$$L_j(t) = P(x(t) = j|\boldsymbol{O}, M), \tag{4.6}$$

i.e., the probability of being in state j at time t, is calculated using using the forward-backward algorithm. The forward probability $\alpha_j(t)$ and backward probability $\beta_j(t)$ are defined as

$$\alpha_j(t) = P(\boldsymbol{o}_1, ..., \boldsymbol{o}_t, x(t) = j|M) \tag{4.7}$$

$$\beta_j(t) = P(\boldsymbol{o}_{t+1}, ..., \boldsymbol{o}_T|x(t) = j, M) \tag{4.8}$$

Spelled out, $\alpha_j(t)$ is the probability of the partial observation sequence up to time t and that M is in state j at the time step. The backward probability is the probability of the partial observation sequence at the subsequent time steps up to the last vector, given that at the current time the model state is j. The forward probability is a joint probability of the observations and the state, whereas the backward probability of the observations is conditional on the state. This allows for the state occupation probability to be determined by the product of the forward and backward probabilities (from Eqs. 4.6, 4.7, and 4.8)

$$L_j(t) = \frac{\alpha_j(t)\beta_j(t)}{P(\boldsymbol{O}|M)} \tag{4.9}$$

α and β are calculated respectively using the recursions

$$\alpha_j(t) = \left[\sum_{i=2}^{N-1} \alpha_i(t-1)a_{ij}\right] b_j(\boldsymbol{o}_t) \tag{4.10}$$

$$\beta_i(t) = \sum_{j=2}^{N-1} \beta_j(t+1)a_{ij}b_j(\boldsymbol{o}_{t+1}) \tag{4.11}$$

and the initial conditions,

$$\alpha_1(1) = 1, \quad \alpha_j(1) = a_{ij}b_j(\boldsymbol{o}_1) \tag{4.12}$$

$$\beta_i(T) = a_{iN} \tag{4.13}$$

for $1 < j < N$ and the final conditions

$$\alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T)a_{iN} \tag{4.14}$$

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j}b_j(\boldsymbol{o}_1)\beta_j(1) \tag{4.15}$$

where the limits of the sums exclude the states 1 and N because they are non-emitting. The recursion of Eq. 4.10 calculates the forward probabilities (of seeing the specified observations and being at the state j) by summing all possible forward probabilities for all possible predecessor states i weighted by the transition probability a_{ij}.

From Eqs. 4.7, 4.14, and 4.15 it follows that calculating the forward probability also yields the total likelihood $P(\boldsymbol{O}|M) = \alpha_N(T)$.

4.3.2 Viterbi training

An alternative approach to the Baum-Welch algorithm is an iterative procedure called Viterbi training (VT), also called Viterbi extraction or Baum-Viterbi algorithm since it involves the Baum re-estimation (Eqs. 4.4 and 4.5) and the Viterbi algorithm (Lember et al., 2008). Instead of maximising the likelihood of all the data as in Eq. 4.10, in VT the probability of only the most likely hidden sequence is maximised

$$\phi_N(T) = \max_i \{\phi_i(T) a_{iN}\} \tag{4.16}$$

for $1 < i < N$ where

$$\phi_j(t) = \max_i \{\phi_i(t-1) a_{ij}\} b_j(\boldsymbol{o}_t) \tag{4.17}$$

and initially

$$\phi_1(1) = 1 \tag{4.18}$$

$$\phi_j(1) = a_{1j} b_j(\boldsymbol{o}_t). \tag{4.19}$$

for $1 < j < N$. This alignment process finds an arc ij for each observation $\boldsymbol{o}_t$.

In order to find the most likely hidden sequence the Viterbi algorithm is applied. The Viterbi algorithm creates paths through the sequence by selecting for each time step the maximum of the previous state likelihoods multiplied by the transition probability to the state. After finding the maximum of the preceding state probabilities for each time step, the sequence is traced back starting from the last unit and selecting the best preceding tags for each tag. This traceback determines the most likely sequence. In practice, the algorithm becomes computationally too expensive to calculate for long sequences and with large vocabulary, so to approximate the most likely sequence beam search is used, which keeps only the k best hypotheses of each hidden state.

Viterbi training results in an approximation of the maximum likelihood estimate, which was computed in the Baum-Welch algorithm (in theory; beam search makes also the Baum-Welch an approximation). The approximation is convenient, since using only the best hidden sequence for updating the HMM parameters makes the Viterbi training is computationally less expensive than the Baum-Welch algorithm. Viterbi training is used in the Kaldi toolkit for estimating the HMM/GMM acoustic models in the standard recipes.

4.3.3 A discriminative training criterion: MMI

The MLE method described in Section 4.3 aims to maximise the likelihood of the observed sequence given the most probable HMM in Viterbi training, or all the

possible HMMs in Baum-Welch. The MLE method maximises the likelihood of the observations for all of the competing HMMs independently of each other. However, the ultimate aim is to find the HMM that most accurately models the observation sequence, so it would make sense to also try to find meaningful differences between HMMs. In discriminative training, instead of maximising the likelihood of the data given the model, a model is trained to discriminate between the classes, which in this case are different phoneme sequences, corresponding to the HMMs. This way, in principle, more of the model capacity is used to model the boundaries between different HMMs, instead of using it to model just the relations between individual HMMs and alignments.

The discriminative objective function can be simply the difference between the correct classifications (e.g., a phoneme sequence) for a set of examples and the classifications assigned to them by the model. This is called the minimum classification error (MCE) criterion (Juang et al., 1997). Another type of objective function is the maximum mutual information (MMI) (Bahl et al., 1986) criterion

$$\mathcal{F}_{\mathrm{MMI}}(M) = \sum_{r=1}^{R} \log \frac{P(\boldsymbol{w}_r)P(\boldsymbol{O}_r|\boldsymbol{w}_r)}{\sum_{\boldsymbol{w}} P(\boldsymbol{w})P(\boldsymbol{O}_r|\boldsymbol{w})} \tag{4.20}$$

where $\boldsymbol{w}_r$ is the correct transcription for the r'th speech file (Povey, 2005). The numerator is the log-probability of the output sequence, and the denominator is the log-probability of all possible output sequences. This way the probability of a particular sequence is normalised by the probability of all sequences. In other words, the probability of all possible sequences is *minimised*, while maximising the probability of the correct output sequence. Since the correct sequence is included also in the denominator, the maximum value of the objective function is zero.

Optimisation w.r.t the MMI criterion is achieved by the extended Baum-Welch (EBW) algorithm. The Gaussian parameter updating formulas are reminiscent of the Baum-Welch updating formulas (Eqs. 4.4 and 4.5), whence the name. The EBW is described for example by Jiang (2010).

4.3.4 Phone context, state tying, and phonetic decision trees

Phones of the same phoneme sound different when flanked by different phonemes. For this reason, contextual information is modelled, too, by taking into account the preceding and subsequent phonemes of the modelled phoneme. These phonemes with left and right context of one are called triphones, and each can be assigned a HMM (Schwartz et al., 1985).

When considering triphones instead of monophones, the number of possible phonemes increases to the power of three. This means each class of triphone will include fewer instances in the training data, which brings difficulties in estimating the state output PDFs. To alleviate the data sparseness, the states of the phoneme HMMs are can be tied together so that the parameters of the output distributions of those states are shared. This makes the estimation of the parameters more robust because there are more training data occurrences, and also makes the total system

more compact with fewer parameters (Young, 1992). States are clustered based on a chosen metric of similarity.

In tree-based clustering as described by Young et al. (1994), the states are divided into branches in a top-down optimisation procedure. Starting from the root node, the question that maximises the likelihood is selected for the node, with the data on each side of the divide being modelled by a single Gaussian. In phonetic decision trees the questions are about the context of the phone, e.g. "Is the phone on the left of the current phone a fricative?". After the procedure, the leaves of the tree are the state clusters in which the states are tied. In the final stage, leaves can be merged if the likelihood does not decrease more than a threshold value.

After a tree has been constructed for the states of the triphone models, also previously unseen triphones can be synthesised by traversing the tree to the appropriate leaf node, i.e. cluster, by answering the questions about that triphone's context and using the tied states of that cluster.

4.4 Deep neural networks for acoustic modelling

In the previous decade, deep neural networks achieved state-of-the-art results in acoustic modelling, supplanting the Gaussian mixture models as the most accurate method to classify observations into phoneme classes. In the HMM/DNN hybrid approach, DNNs provide *pseudo-likelihoods* of the observations for each HMM state. The approach defined in Eq. 2.2 works well for traditional ASR systems that use GMMs for acoustic modelling. However, if the generative GMM is replaced with a discriminative neural network model, it does not produce an acoustic likelihood $P(\boldsymbol{o}_t|\boldsymbol{s}(t))$ but a state level posterior probability $P(\boldsymbol{s}(t)|\boldsymbol{o}_t)$, which is a problem because the decoding (Eq. 2.2) relies on the likelihoods. This mismatch can be bypassed by applying the Bayes' rule to produce pseudo-likelihoods:

$$P(\boldsymbol{x}_t|\boldsymbol{s}(t)) = \frac{P(\boldsymbol{s}(t)|\boldsymbol{x}_t)P(\boldsymbol{x}_t)}{P(\boldsymbol{s}(t))} \propto \frac{P(\boldsymbol{s}(t)|\boldsymbol{x}_t)}{P(\boldsymbol{s}(t))} \tag{4.21}$$

The state priors $P(\boldsymbol{s}(t))$ can be gathered from corpus frequencies (Bourlard and Morgan, 2012).

4.4.1 TDNNs and RNNs

Two common types of DNN used for acoustic modelling are RNNs and time delay neural networks (TDNN). Both RNNs and TDNNs are inherently suited to modelling time-series data, where it is important to capture long-term time dependencies, such as speech. However, the two types of DNN are different in how they achieve this.

RNNs (see Section 3.3.1)

TDNNs

(Peddinti et al., 2015)

4.4.2 Sequence-level lattice-free MMI

Currently, the state-of-the-art implementations of DNNs in Kaldi are trained using lattice-free MMI (LF-MMI) training criterion, as described by Povey et al. (2016). These are called "chain" models in Kaldi. LF-MMI is a sequence discriminative criterion, which means that the aim is to maximise the conditional log-likelihood (Eq. 4.20) of the correct transcript on the sequence level. In the traditional MMI approach, a cross-entropy system is trained to generate lattices for a weak language model, and the lattices are used to approximate the possible word sequences for the discriminative objective function denominator. In LF-MMI, however, the possible word sequences are not approximated with a lattice, but a phone-level language model is computed so that the sum in the denominator is not approximated, but can be computed exactly. This is possible since a phone-level LM requires significantly less mamory than a normal word-level LM. The phone LM is represented as an FST, created in a similar manner as the normal decoding FST described in Section 4.8. In this case there is no lexicon and grammar FSTs but a phone grammar FST, so the graph consists of the component FSTs H, C, and P. The numerator FST uses a lattice to represent the utterance. The numerator FST is composed with the denominator FST so that the phoneme LM of the denominator removes illegal output sequences. The composition also ensures that the objective function value is negative.

4.4.3 Regularisation of DNN AMs

Regularisation is important in DNN training to impede overfitting. Kaldi chain models incorporate a number of regularisation methods in the DNN architectures. One of the methods is called *L2-regularisation*, which applies the euclidean norm to penalise elements in the output vector that tend to blow up. This is achieved by subtracting $\frac{1}{2}c||\boldsymbol{y}||_2^2$ from the objective function of each frame output $\boldsymbol{y}$, where c is a user-set scaling factor, e.g. 0.005 (Povey et al., 2016). In Kaldi, L2-regularisation is applied on the outputs; note that this method is different from the L2-regularisation applied to the weights, which is another commonly used regularisation method.

Another widely used regularisation method for deep neural nets are *dropout layers* which randomly sample a subset of the neurons and their connections which are omitted, dropped out, during an iteration of the training (Srivastava et al., 2014). This prevents the neurons from relying on the other neuron's outputs too much, i.e., prevents too much *co-adaptation* among the parameters. Dropping out random units in training shifts the unit of adaptation down from large groups of neurons to individual neurons, since it is not guaranteed that the other neurons are present. This principle is explained through an analogy to sexual selection, in the above mentioned original paper. Since genes are mixed randomly with another set of genes from a conspecific between every generation, individual genes cannot rely on specific other genes to be present in the future. This inhibits aggregation of large interdependent gene complexes, in which genes function well only with the group of co-adapted genes. These kinds of complexes would become rigid, and fragile to the inevitable mutations that will always occur quite randomly: new genes would be difficult to

incorporate in the complex. This kind of rigidity is analogous to overfitting to the training data and failing to find generalisable rules that apply to new data samples from the hidden distribution, which regularisation attempts to lessen.

A third type of regularisation technique is called *leaky HMM* in Kaldi. Transitioning from any state a to any state b is allowed once per frame with the probability of a small (typically around 0.01) *leaky-hmm-coefficient* times the probability of state b. The aim is to effect a gradual forgetting of the context, since transitioning to a random state is equal to stopping and restarting the HMM (Povey et al., 2016). This reduces the overfitting caused by too much memorising of the training data sequences.

Another technique to regularise the sequence-level training is to add a separate output layer to the network that learns the cross entropy objective, as well as a separate last hidden layer. This means the network has two output branches with two separate weight matrices in each branch. After the training, the cross entropy branch can be discarded, and the main, sequence output branch is left to be used in decoding. This technique is abbreviated as `xent_regularize` in the Kaldi code, and the associated hyperparameter is a scaling factor for the cross entropy objective, typically 0.1 because its dynamic range is naturally larger than that of the MMI objective function (Povey et al., 2016).

4.5 Speaker-adaptive training

Variability between speakers poses a challenge to an ASR system. Each speaker may have an idiosyncratic voice, distinct style of pronunciation as well as distinct recording conditions, which can degrade the ASR performance, as the training speech set and test speech set differ from each other. This section describes some of the methods to account for inter-speaker variability by adapting either the model or the features to a particular speaker. An exhaustive overview of all the used methods is beyond the scope of this thesis.

The adaptation can be done either in testing or training, usually respectively referred to as *speaker adaptation* and *speaker adaptive training* (SAT). Speaker adaptation can be done by modifying a speaker-independent (SI) model to create personal, speaker-dependent (SD) models for each speaker. This can be done by taking a small number of speech data from the speaker and using this adapt a SI model to the specific speaker (Shinoda, 2011). In SAT, speaker-specific information is incorporated in the training of the model. When a model is trained using SAT, it naturally benefits to use an adaptation scheme also in testing. Both GMM- and DNN-based AMs have been shown to benefit from speaker adaptation as well as SAT. A commonly used SAT method for GMMs is the feature-space MLLR. For DNNs, speaker embeddings can be appended to the speech feature vectors. These methods are described in this section.

SAT requires identifying each speaker from the metadata that indicates the speaker ID. Adaptation can be useful even if the speaker-identifying metadata is absent, in which case a separate adaptation is learned for each utterance, as if each utterance were spoken by a different person. In this case, the adaptation is

for inter-utterance variability in general, so speaker-adaptive training becomes a misnomer. Furthermore, the adaptation can, of course, be done w.r.t any attribute of the data that has been labelled in the metadata, and thus these methods could be termed more generally *attribute-aware training* (Rownicka et al., 2019).

With a HMM/GMM acoustic model, speaker adaptive training can be done by representing each speaker's distinct qualities as a transform in the feature space or in the model space. A feature space transform is applied on the observation vectors, and model space transform on the mean and variance of the GMM. Furthermore, a model space transform can be unconstrained or constrained, the former being separate transforms for the means and variances and the latter using the same transform for both (Gales, 1998).

Using the model space transform, the aim is to learn the optimal model $\boldsymbol{M}$ and adaptation $\boldsymbol{G}^{(r)}$ for speaker r

$$(\hat{\boldsymbol{M}}, \hat{\boldsymbol{G}}) = \underset{(\boldsymbol{M},\boldsymbol{G})}{\operatorname{argmax}} \prod_{r=1}^{R} P(\boldsymbol{O}^{(r)}; \boldsymbol{G}^{(r)}(\boldsymbol{M})) \tag{4.22}$$

where $\hat{\boldsymbol{M}}$ are the model parameters (Anastasakos et al., 1996). The feature space transform is analogous, only transforming the observations instead of the model.

A common SAT transform is the maximum likelihood linear regression where $\boldsymbol{G}(\boldsymbol{M})$ is an affine[6] transformation defined by the matrix $\boldsymbol{A}$ and bias $\boldsymbol{b}$

$$\boldsymbol{\mu}^{(r)} = \boldsymbol{A}^{(r)}\boldsymbol{\mu} + \boldsymbol{b}^{(r)} \tag{4.23}$$

Whether an adaptation modifies the model or the features is in some cases only a question of interpretation when describing the method, and a question of choosing among two equivalent implementations. The constrained MLLR (CMLLR) can be represented as a feature space transform (Gales, 1998), and is therefore also called feature-space MLLR or fMLLR. The transformation projects the features from the speaker-specific space to the speaker-normalised space.

In practice, in the Kaldi implementation, affine transform is applied by appending a 1 to the feature vector, and multiplying it with the linear transform $\boldsymbol{A}$ concatenated with the constant offset (bias) $\boldsymbol{b}$: $\begin{bmatrix}\boldsymbol{A}; \boldsymbol{b}\end{bmatrix} \begin{bmatrix}\boldsymbol{o} \\ 1\end{bmatrix}$. The Kaldi program `transform-feats` is used to multiply the feature vectors with transform matrices.

As well as GMM parameters, also DNN acoustic models (see Section 4.4) can be estimated speaker-adaptively. Adapting can be performed in the feature-space by appending or transforming the observations or in the model-space by modifying the DNN AM parameters.

A common method is to extract i-vectors from speakers (Dehak et al., 2010), optionally perform a transformation of the i-vectors using a control network, and append them to the features that are fed to the DNN (Miao et al., 2015). i-vectors have usually a hundred or a few hundred dimensions that encode properties of a

[6] *Linear* regression is thus a slight misnomer.

speaker as well as the environment, enabling the AM to generalise more robustly to speech in different conditions. The original paper by Dehak et al. (2010) referred to the i-vector as the *total factors* $\boldsymbol{w}$ in the *total variability space* $\boldsymbol{T}$ because it models both speaker and channel variability in contrast with joint factor analysis (Kenny, 2005) which makes a distinction between the two sources of variability.

The total variability space $\boldsymbol{T}$ models the variability between utterances. i-vector extractor training starts by estimating a speaker-independent GMM called an *universal background model*, or UBM. The purpose of the UBM is to represent the general, or universal, characteristics of speech, i.e., it is the speaker-independent model. Baum-Welch statistics are obtained from the UBM at the frame level. The i-vector is then defined as the mean vector of the posterior Gaussian distribution conditioned on the Baum-Welch statistics for a given utterance (Dehak et al., 2010). In practice it is a MAP estimate (Kenny et al., 2005). The mean vector (i-vector) is actually a concatenation of the mean vectors of the mixture components, called a supervector (Campbell et al., 2006). A speaker-specific utterance supervector $\boldsymbol{M}$ is composed of the factors

$$\boldsymbol{M} = \boldsymbol{m} + \boldsymbol{T}\boldsymbol{w} \tag{4.24}$$

where $\boldsymbol{m}$ is the UBM supervector. The extractor compresses the high-dimensional statistics from the UBM into the dense i-vector representation for a given utterance.

Another approach to modelling speaker characteristics is to train a feedforward deep neural network to project speakers into an embedding space that models the speaker variance. Snyder et al. (2017) introduced a *speaker embedding* method where the DNN is trained to classify speakers from variable-length segments. The DNN learns to assign each speaker an embedding space vector which can be then used in the AM training similarly to an i-vector. Snyder et al. (2018) describe the use of speaker embeddings, which they call x-vectors, in speaker recognition. In contrast with i-vectors, x-vectors model only the speaker characteristics since the DNN is trained to identify speakers.

Speaker embeddings can be extracted either online or offline. This refers to which frames can be used in the extraction. When streamed audio is transcribed on the fly, the transcribed utterance is partial: only the frames up to time t can be be used in the embedding instead of the complete utterance. Online decoding is simulated in the standard recipes in Kaldi, and the embeddings are extracted using partial utterances every 10 frames or so. In online extraction, the features are carried over from previous utterances of the same speaker. Offline extraction can be used when there is no need to stream the features or you do not want to simulate this application). Offline extraction is standard when using the embeddings in speaker recognition instead of speech recognition.

4.6 Silence and pronunciation probability modelling

The Kaldi toolkit allows also for modelling the probability of an optional silence between specific words, and the probability of different pronunciations of a word. These methods have been found to improve WER results by a small but consistent margin (Chen et al., 2015).

Differences of pronunciation are significant in many languages. However, in Finnish, which has a phonemic orthography (see Section 4.7), pronunciation probabilities are not applicable. If the mapping from phonemes to graphemes is one-to-one, there are no alternative pronunciations for a word.

The probability of an optional silence phone between two words is estimated from statistics collected from alignments. When using a subword vocabulary, it is important to indicate which boundaries are actual word boundaries, because the optional silence should not be inserted in the middle of a word. Smit et al. (2017) described and implemented a method to build the lexicon FST in such a way that restricts the use optional silence to only word boundaries.

4.7 Mapping words to phoneme sequences

The hidden state sequence decoded from a HMM corresponds to a phoneme sequence, but the ultimate aim is to generate a word sequence for a given observation sequence. For mapping words to phoneme sequences, the ASR system applies a *lexicon*, also referred to as *pronouncing dictionary*, or just *dictionary*.

In Finnish there is generally a one-to-one mapping from letters to phonemes. In the jargon of linguistics, Finnish has a *phonemic orthography*. This makes creating a lexicon very simple, as each letter can be assigned to a phoneme, and the word-to-phonemes mapping follows trivially.

[oov]	SPN
!SIL	SIL
[laugh]	SPN
[reject]	NSN
+i+	I
+loma	L O M A
+ssa	S S A
avoim+	A V O I M
zoom+	T S O O M
äitiys+	AE I T I Y S
överi	OE V E R I
über	Y Y B E R

Table 2: An example of a lexicon. The first entries have special phonemes: spoken noise (SPN), silence (SIL) and non-spoken noise (NSN). "oov" refers to out-of-vocabulary words. "+" is the intra-word boundary marker in subword vocabularies. Note also the inaccurate pronunciation of "zoom": "T S U U M" would be more accurate.

A lexicon for a language like English has to be constructed largely by hand, as most of the mapping from letters to phonemes does not have a sound [7] logic behind it. This is due to historical change in the pronunciations of words that was

[7] Ignore the pun.

not translated into corresponding change in their written forms (e.g., during the Great Vowel Shift) as well as loan words from other languages which have different rules of orthography (Crystal and Potter, 2020). For a language that does not have a phonemic orthography but some constant patterns of pronunciation, a machine learning approach can also be used to learn a mapping for the dictionary.

Similar change can be observed in Finnish as words are imported from other languages, mainly from English. Loan words often do not follow the Finnish orthography, which means that the one-to-one mapping from letters to phonemes is no longer accurate for these words. For example, "googlata" is pronounced "G U U G L A T A"[8] instead of following the Finnish orthographic rules to pronounce it "G O O G L A T A", or alternatively spelling it "guuglata". In this study these inaccuracies are ignored, and the Finnish lexicon is generated automatically using the simple letter-to-phoneme correspondence, where a letter corresponds to a single phoneme, usually denoted by itself (with a few exceptions, e.g., "c"->"K", "q"->"K V"). Table 2 is an example of a few lexicon entries that follow the word-to-phoneme mapping used in the experiments of this thesis.

4.8 Weighted finite-state transducers in Kaldi

Weighted finite-state transducers are a type of automaton in which a transition has an input label, an output label, and a weight. A special case of FST is a finite-state acceptor (FSA) where the input and output labels of a transition are equal. A FST, having both input and output labels, maps an input sequence to an output sequence when a path is taken through it. Figure 4 displays a simple example of an FST. The

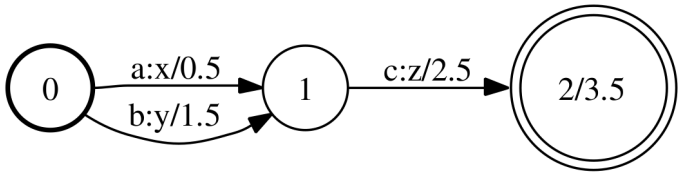

Figure 4: A weighted finite-state transducer (Allauzen et al., 2007).

example transducer has three states, the initial state (0) denoted with a bold circle and final state (2) with a double circle. The arcs between states have input labels {a, b, c}, output labels {x, y, z}, and real number (floating point) weights associated with them. Also the final state has a weight. This transducer would map the input sequence "ac" to the output sequence "xz" with the weight calculated by summing the individual arc weights, in this case adding up to 6.5.

FSTs are used in Kaldi to encode many of the components of the ASR system. In training, the lexicon is represented as a FST. In decoding, the complete ASR system is encoded into a FST, called a *decoding graph* before the actual decoding, as the graph creation is typically the most resource-hungry step in the process. In Kaldi

[8]These are not official phonetic alphabet (they are the phoneme symbols that are used in this thesis) but hopefully the example still makes sense.

chain models, FSTs are used in the LF-MMI training to encode the denominator and numerator of the objective function (see Section 4.4).

Kaldi decoding graph creation

Kaldi composes the $HCLG$ decoding graph from four component transducers: the HMM H, the context C, the lexicon L, and the LM (or grammar) G (Povey et al., 2012).

The n-gram language model that is used in decoding is converted from the ARPA format, described in **??**, into the G transducer using the `arpa2fst` program. The purpose of G is not to transduce a sequence from one domain to another, but to assign weights to the possible word sequences. For this reason, the input and output labels are equal, making it technically a finite-state acceptor. The ARPA model lists the conditional base-10 logarithmic probabilities for each n-gram. These give weights to the arcs of the corresponding paths in the acceptor, the arc weights having an inverse relation to the probabilities (i.e., they are negated) and using natural logarithm instead of 10-base. Referring to the weight of decoding graph FST as "cost" is more intuitive, so in this text this term is used, too. A complete path through the acceptor corresponds to a word sequence and the cost of the path indicates how *un*likely the sequence is. If a probability of the highest-order n-gram has not been explicitly specified in the n-gram model, a path is taken through the backoff node which corresponds to the lower-order backoff n-grams and the costs are determined by the associated backoff probabilities. Since the model can recursively back off all the way to unigrams, any word sequence is accepted and given a cost. The backoff arcs do not have a word label, which raises the problem that there are multiple paths for a single input sequence, making processing the graph inefficient[9]. The backoff arcs are therefore assigned a *disambiguation symbol* `"#0"` as the input label. The disambiguation symbol allows for an operation (in practice, an algorithm) called *determinisation*. A deterministic transducer has the property that one input string matches at most one path (Mohri et al., 2008). The acceptor also dismisses sentence start and end tokens of the LM (`<s>` and `</s>` or whatever they are) since these are not wanted in the speech transcripts. The number of word types used in speech recognition can also be limited to make the decoding graph of feasible size, in which case some types are omitted from the LM acceptor.

The lexicon transducer L maps a phoneme sequence input to an output consisting of one word. If the lexicon has multiple words with the same pronunciation, or when a phoneme sequence is a part of multiple word pronunciations, word disambiguation symbols {`#1`, `#2`,...} are needed to ensure each phoneme sequence has only one possible word output. Ambiguity in the transducer means that there could be multiple paths matching one input string. See Section 4.7 for discussion about orthography and Section 4.6 for discussion about the case where one word has multiple pronunciations.

The grammar acceptor G and lexicon transducer L are combined to compose a new transducer LG that maps a phoneme sequence input to a word sequence output. The composition is done in the program `fsttablecompose`, and the new transducer

[9] more disadvantages?

is determinised in the program `fstdeterminizestar`.

C is the transducer from context-dependent phonemes to context-independent phonemes. By composing C with LG, the triphones are mapped to words. C is built dynamically in the process of composing it with the existing LG FST. The final FST is the HMM, which maps the state transitions to triphones, or more generally any context-dependent phonemes. After H is composed with the CLG, the $HCLG$ is complete and can be used for lattice generation and decoding.

4.9 Lattices and n-best lists

The decoding graph built by the Kaldi programs encodes all possible word sequences and the corresponding HMM state sequences. A sequence of observations is decoded with the graph in the manner described by Povey et al. (2012). First, a WFSA U is built from the observations that encodes the acoustic weights for each arc of each state transition that can emit each observation in the sequence. U includes $T + 1$ states with arcs between the states that correspond to the HMM states at the time step t. The costs of the arcs encode the acoustic log-likelihoods. The utterance-specific *search graph* S that is traversed during decoding is generated by composing U with the decoding graph. The search graph is not searched completely, but a subset of the best paths are selected by *beam pruning*. When generating the lattice that encodes the subset of the best paths, some of the desiderata are: that the lattice includes all word sequences within the beam size α that represents a log-likelihood difference to the optimal path; that the scores and alignments in the lattice are accurate; that the lattice does not contain duplicate paths of the same word sequence (Povey et al., 2012). Figure 5 depicts a word lattice.

An alternative to a word lattice is to simply create lists of n best transcriptions of the utterance. Though sometimes simpler to create and use, n-best lists are less efficient than lattices. In n-best lists, there are often transcriptions that differ only by a few words, which is redundant. n-best lists are capable of encoding far fewer transcriptions than lattices in the same amount of memory, and are therefore usually much more restrictive when, for instance, doing rescoring with a language model.

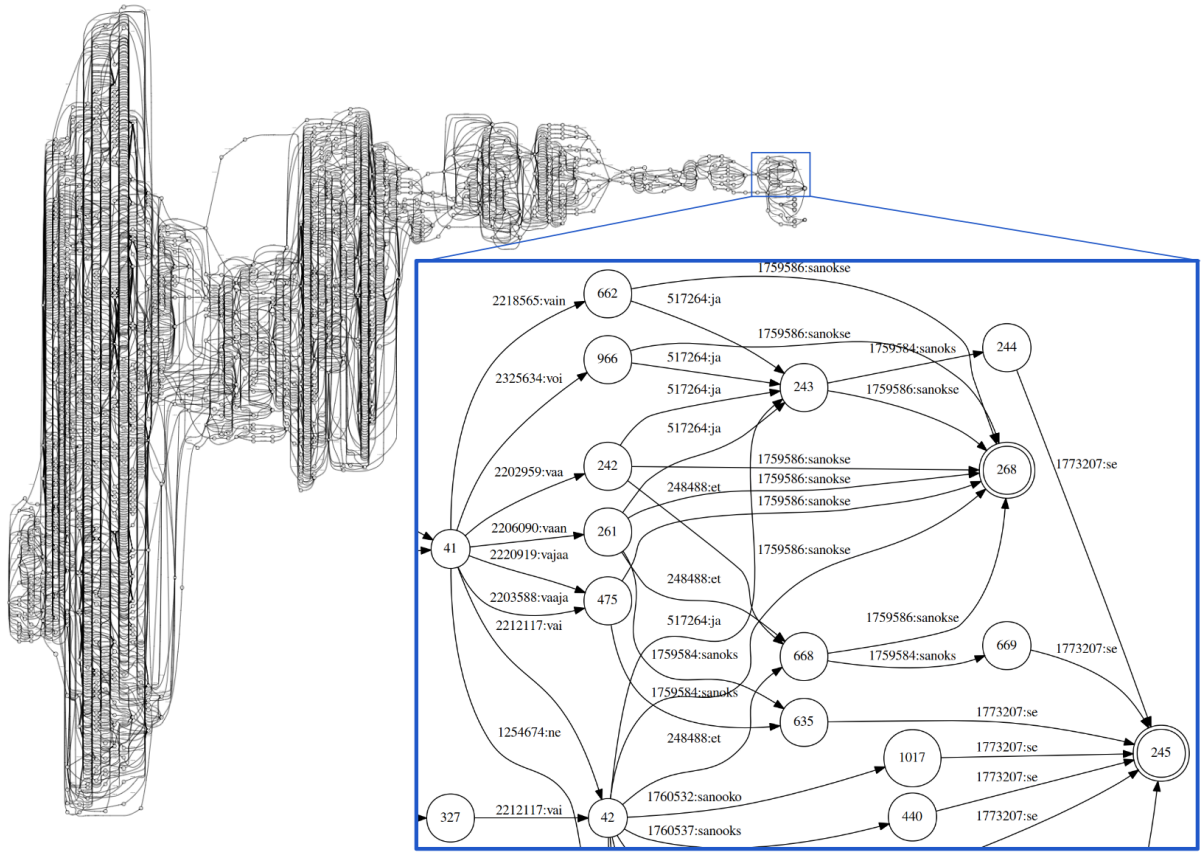

Figure 5: A word lattice FST generated by one of the ASR systems built during the thesis. The correct transcription is "nii voi joo siis ei se varmaa siis emmä usko et sitä ongelmaa niinku se mä vaan mietin niinku nyt tota et et et mitä se tuomas sano sanoks se niinku et ne avaimet on niinku menny sen roskiksen mukana niinku mä ehdotin vai sanoks se". Each edge of the lattice is assigned a word as well as acoustic and language model costs (not printed in the figure). The numbers in the figure are identifiers of nodes and words. As a path is taken through the lattice, a transcription is generated.

5 Experiments

5.1 Acoustic modelling experiments

5.1.1 Speech corpora

The acoustic model training, development and test sets are the same as in (Enarvi et al., 2017). The training speech data are 85 hours of read and spontaneous speech from three different sources. The SPEECON corpus consists of 550 speakers reading 30 sentences and 30 single words as well as speaking 10 spontaneous sentences (Iskra et al., 2002). The DSPCON[10] corpus consists of 5281 spontaneous sentences from 218 different male students and 24 female students, totalling 9.8 hours (Enarvi et al., 2018). The third source is the FinDialogue part of the FinINTAS corpus (Lennes et al., 2009). The development and test sets contain additionally spontaneous utterances from radio shows, referred to as RadioCon. Table 3 lists the acoustic modelling data sets.

Corpus	# of speakers				# of utterances				# of hours
	total	train	devel	eval	total	train	devel	eval	
SPEECON	549	549	0	0	5499	5499	0	0	18.8
FinDialogue	22	22	0	0	6338	6338	0	0	10.4
DSPCON	243	192	40	11	5219	4129	863	227	9.7
RadioCon	11	0	5	6	440	0	126	314	0.5
total train	763				15966				36.6
total devel	45				989				2.0
total eval	17				541				0.7

Table 3: Speech corpora. Training data "train", development data "devel", and evaluation data "eval".

Speed perturbation is a data augmentation method in which the speed of the audio is increased or decreased (Ko et al., 2015). This method is used for the DNN AM training data, augmenting the speech data by changing its speed by a factor of 0.9 and 1.1, increasing the amount of data by a factor of three. As the quality, or domain, of the three different data sets is different, it could be useful to augment only the in-domain data, i.e., DSPCON. A system is trained with the whole dataset augmented, as well as one system augmenting only DSPCON. The difference

5.1.2 HMM/GMM acoustic model architecture and training

Different features are used in GMMs and DNNs. The GMMs input 13 MFCCs and their Δ and $\Delta\Delta$ features, amounting to a 39-dimensional feature vector. Cepstral mean and variance normalisation is applied to the features per speaker.

[10] `http://urn.fi/urn:nbn:fi:lb-201708251`

The GMM phoneme model set was trained for the most part by following the Kaldi recipe for the Wall Street Journal corpus[11]. The monophone model set is trained with a subset of 2000 shortest utterances. The features are first aligned equally with the states, after which 40 iterations of Viterbi training are performed to estimate the HMM/GMM model set and generate an improved alignment. The number of Gaussians is increased in between iterations, reaching a total of 1000 Gaussians.

The triphone model set is initialised from the monophone models and their alignments, and is trained in a three consecutive steps. Subsets of the data set is selected, including 4000 utterances for the first step, 8000 utterances for the second step, and then using the whole 15000 utterance training data set for the third triphone training step. The first step (`train_deltas.sh`) is similar to monophone training, using the delta and delta-delta features in addition to the MFCCs, and increasing the total number of Gaussians to 10k. The second step (`train_lda_mllt.sh`) splices the MFCC features, reduces the dimensionality back to 40 using LDA, and estimates an MLLT transform. The MLLT transform is applied on the features and new estimates of the models are computed, increasing the total number of Gaussians to 15k. The third and final triphone training step (`train_sat.sh`) does speaker adaptive training utilising fMMLR. The total number of Gaussians is increased to 40k.

After the triphone model set is trained, silence probabilities (see Section 4.6) are estimated from the statistics of the training data alignments. Since the Finnish lexicon is a one-to-one mapping from letters to phonemes, there are no optional pronunciations of words for which to calculate pronunciation probabilities.

An MMI (see Section 4.3.3) system is trained based on the triphone alignments and silence probabilities. The MMI model generates the alignments that are used to train the deep neural network acoustic model.

The GMM triphone model set was kept fixed throughout the experiments, after first running a couple of experiments to tune the model. These experiments included evaluating the MMI model compared to a fourth triphone training step and experimenting on a couple of different

5.1.3 HMM/DNN acoustic model architecture and training

The DNN AM inputs higher-resolution features than the GMM AM. 40-dimensional MFCCs are extracted from the data set, which has been augmented using speed perturbation.

The DNN is a Kaldi chain model, trained on the LF-MMI objective

5.1.4 Speaker embedding experiments

In the speaker embedding experiments, pretrained extractors are used as well as extractors trained on the AM training data.

[11] `https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/run.sh`

Pretrained VoxCeleb i-vector and x-vector extractors
Pretrained i-vector and x-vector extractors trained on the VoxCeleb data (Nagrani et al., 2017; Chung et al., 2018) are used in the experiments. The VoxCeleb1 data contains about 100k utterances from 1251 celebrities and the VoxCeleb2 data contains about 1M utterances from over 6000 speakers. The code that was used for training the extractors is in the Kaldi repository[12][13] and the pretrained extractors are downloaded online[14].

The VoxCeleb i-vector extractor was trained on 24 MFCCs and their delta and delta-delta coefficients. An energy-based VAD system is used to select the voiced frames for both i-vector and x-vector systems. The i-vector system UBM is a GMM that has 2048 full-covariance component Gaussians. The i-vectors have 400 dimensions, and are subsequently reduced to 200 dimensions using an LDA model.

The VoxCeleb x-vector extractor is trained on the VoxCeleb speech data that has been augmented in various ways. The MUSAN corpus (Snyder et al., 2015) of music, noise and speech as well as simulated room impulse response (Ko et al., 2017) are used to generate noise for the speech data. The noise is added to the speech data and a subset of the noisy audio files is randomly selected and pooled with the clean audio files. This increases the amount of data roughly twofold. The x-vector extractor was trained on 30 MFCCs with their deltas and delta-deltas. The DNN is a TDNN with ReLU non-linearities where the five first layers use frame-level training with a temporal context of a few adjacent frames. The architecure is described by Snyder et al. (2017). After the frame-level layers is a pooling layer that aggregates the frame-level outputs, calculating the mean and standard deviation of the whole segment. The last two layers before softmax operate on the segment level statistics. The x-vector is extracted from the penultimate layer (the 6th layer), with 512 dimensions. An LDA model reduces the x-vector dimensionality to 200.

Pretrained i-vector extractor trained on Yle and Parliament data
A pretrained i-vector extractor was used as a comparison to the models trained in this work. This extractor was trained on the Yle and Parliament speech data.

i-vector extractor trained on the conversational Finnish data
Most of Kaldi recipes for training a HMM/DNN model include i-vectors as the speaker-adaptation method. In this thesis, the extractor for the baseline i-vectors is trained similarly to the Switchboard recipe. The extractor is trained on the same data as the acoustic model for extracting online i-vectors. The speed perturbation data augmentation is used also for the extractor training data. Both online and offline i-vectors are extracted and compared.

x-vector extractor trained on the conversational Finnish data
An x-vector extractor is trained also on the conversational data. This model uses the same setup and hyperparameters as the Voxceleb x-vector extractor. The data is

[12] i-vectors: `https://github.com/kaldi-asr/kaldi/tree/master/egs/voxceleb/v1`
[13] x-vectors: `https://github.com/kaldi-asr/kaldi/tree/master/egs/voxceleb/v2`
[14] `https://kaldi-asr.org/models/m7`

augmented similarly to the Voxceleb model, increasing the number of utterances up to about 100k.

Domain adaptation by finetuning the pretrained extractors
Since the VoxCeleb extractor models are trained on English speech, it is possible that the extractors are suboptimal for Finnish. The extractors are trained on task of identifying speakers based on an utterance. There could be differences in the two languages in what kind of acoustic cues are utilised to identify the speaker. As with other modelling tasks, such as acoustic modelling, the extractor model could benefit from finetuning the large pretrained model on a smaller set of in-domain (or closer to the test domain) speech corpus.

The Voxceleb x-vector extractor was finetuned with the augmented Finnish speech corpus. The softmax layer was replaced so that the number of target classes (speakers) was compatible with the finetuning data. Apart from the ouptu layer, all the other layers and their parameters were kept from the pretrained model, and the model was trained on the Finnish data for one epoch. This improves the evaluation set WER result a little (Table 4).

Replacing the last hidden layer of the extractor model was experimented on, but this did not improve the WER results. Adding a new layer after the 7th layer was also tried, but here too the results were worse than for the pretrained model.

Combining i-vectors and x-vectors
Since the method of modelling speaker variability is different in the two embedding types (see Section 4.5), it is possible that they encode different, not completely overlapping information. Therefore it might benefit to combine the two speaker embeddings. Simply concatenating the two vectors improved the WER results by a small margin, compared to using only x- or i-vectors. In Table 4 this method is called the i-x-ensemble vectors.

Results
Table 4 presents the results of the speaker embedding experiments.

The fact that the pretrained speaker embedding extractors get better results than the embeddings trained on the conversational speech data suggest that the domain of the speech is not as important as the quantity of the speech. The domain means the language as well as the type of speech, since the Voxceleb speech is in English.

5.1.5 Discussion

5.2 Language modelling experiments

5.2.1 Text corpora

Two text corpora were used for training the language models: the DSPCON speech corpus transcriptions and WEBCON corpus collected from Internet forums. Table 5 lists the sizes of the LM corpora, in total roughly 76M tokens.

Method	Data set	devel	eval
None		25.1	27.3
i-vectors	conv speech	23.9	26.5
	VoxCeleb pretrained	23.9	25.5
	Yle, parliament pretrained	**23.3**	25.8
x-vectors	conv speech	24.3	26.3
	VoxCeleb pretrained	23.7	25.5
	VoxCeleb pretrained and conv speech finetuned	24.2	25.3
i-x-ensemble	conv speech	24.0	25.8
	VoxCeleb	23.7	**25.1**

Table 4: The WER results for the word-based system using different speaker embeddings concatenated with the input features. This is the 1st-pass, using a 4-gram language model.

LM training corpus	# of word tokens
DSP conversational speech corpus transcriptions	61k
WEB corpus, conversational written text corpus	75.9M

Table 5: The two language modelling corpora. DSPCON is more similar to the development and test corpora than the WEBCON, but it is significantly smaller. The two coprpora are used to generate separate n-gram models, which are interpolated. The interpolation weights are optimised on the development set.

5.2.2 Decoding with n-gram LM

The word-based systems use an n-gram LM with $n = 4$ for the first-pass decoding and lattice generation. The SRILM (Stolcke, 2002) is used to train KN-discounted n-gram in the ARPA format (see Section 3.2). The two corpora are interpolated with weights that are optimised on the development data, similarly to (Enarvi et al., 2017).

The subword-based systems use an n-gram LM with $n = 5$ for the first-pass decoding and lattice generation. Different constant and variable values of n were evaluated on the development data to find a suitable value. The VariKN (Siivola et al., 2007a) tool was used to evaluate variable-order pruned n-grams.

5.2.3 Rescoring lattices with LSTM LM

The lattices generated in the first pass are rescored with an LSTM language model.

The LSTM language model was mostly the same as in (Enarvi et al., 2017). The projection layer has a dimension of 500, after which there is one LSTM layer followed by four highway layers, all with 1500 dimensions. There are dropout layers in between all of the layers, with a dropout rate of 0.2.

In order to weight the in-domain DSPCON corpus more, the WEBCON is

Language model	Devel set WER
Constant-order trigram	25.2
Constant-order 4-gram	**24.5**
Constant-order 5-gram	24.6
Variable-order n-gram	24.9

Table 6: Subword-based system 1st-pass results on the development data set. The AM uses the VoxCeleb x-i-ensemble vectors, making the results comparable with the last line of Table 4 which presented the word-based system results.

	# of n-grams				
	Word	**Subword**			
	4-gram	**trigram**	**4-gram**	**5-gram**	**VariKN** (38-gram)
unigrams	2,427,251	42,698	42,698	42,698	42,699
bigrams	22,606,146	9,032,698	9,032,698	9,032,698	9,032,513
trigrams	4,762,607	10,288,565	9,626,856	9,626,856	7,591,231
4-grams	2,573,404		7,788,695	7,101,982	4,750,328
5-grams				3,601,421	1,539,610
total	32,369,408	10,288,565	26,490,947	29,405,655	24,505,381

Table 7: Number of n-grams per order for the n-gram language models.

subsampled with a factor of 0.2 at each epoch. That is, only a random subset of 20% of the WEBCON corpus is used, resampled at each epoch. The vocabulary is reduced to the 100k most common words to reduce the memory requirements.

For the subword models, the cross-entropy objective is used. For the word models, the training objective is noise contrastive estimation (NCE) (Gutmann and Hyvärinen, 2010) to limit the memory requirements of the word models. This objective NCE samples random words and learns to classify words as training words or noise words. The higher the number of noise samples the slower but more stable the training. The number of noise samples is set to 500 per one training word. The noise words are the same for each batch of training words, and the batch size is set to 24. The noise dampening hyperparameter determines whether noise words are sampled uniformly (0) or according to the unigram distribution (1). This is set to 0.5. The Adagrad (Duchi et al., 2011) optimisation method is used with a learning rate of 0.1.

The maximum length of the token sequence that is processed limits the memory of the LSTM network. It determines how far back the LM can learn dependencies. As the subword tokens are shorter, i.e., fewer characters long, they should probably have a longer context than word tokens. A few different sequence lengths were applied to see how this hyperparameter affects the second-pass WER results. The results are presented in Table 8.

In the decoding, three pruning methods were utilised. Firstly, the standard method of defining a beam width was used: if the logarithmic probability difference

Sequence length	Word LM	Subword LM
8	24.3	
15	**24.2**	
25	24.5	23.5
40	24.8	**23.1**

Table 8: WER results on the evaluation set for word and subword based models with different sequence lengths.

between the best token and another token was more than 650, the other token was pruned. Secondly, the maximum number of tokens per a lattice node was set to 62. Finally, if at some time step there are multiple tokens with the same previous words, only the best is ketp. This context length, or recombination order, is set to 22.

5.2.4 Rescoring n best hypotheses with Transformer-XL LM

The Transformer-Xl models rescored the n-best lists generated from the first or second pass. The use of n-best lists instead of lattices is due to its computational simplicity. n-best lists are suboptimal compared to lattices, but the rescoring of lattices with transformers is not as straight forward as with RNNs, since the state of a position depends not only on the previous state, but on all of the states of other positions, too. This has been done before (e.g., (Irie et al., 2019)), but implementing it was beyond the scope of this thesis work.

The Transformer-Xl model hyperparamters were optimised for the development set to some degree. The network embeds the input to 256 dimensions. The model layers have a total dimensionality of 400. The attention layers have 8 heads with 40 dimensions each. The number of tokens to predict is set to 70. **todo: more details about hyperparams and training**

2nd pass results
When the n-best lists were generated from the n-gram LM decoding, the rescoring improved the development results about 17%, from the first pass 29.0% WER, listed in Table **??**. These numbers are not comparable to those of the Sections 5.2.2 and 5.2.3, since a different acoustic model was used. The table shows the effect of scoring more hypotheses: 50, 200 or 1000. Iterating the training for more steps brings little improvement after 130k training steps, when rescoring the 50-best list.

3rd pass results
The three-pass rescoring scheme utilised all three LM model types. Lattices were generated from the first pass with an n-gram LM, and rescored with an LSTM LM after which an n-best list of best hypotheses was generated. The n-best list was rescored with the transformer model to determine the optimal transcriptions.

# of hypotheses	Devel WER
50-best list	26.2
200-best list	25.1
1000-best list	24.2

Table 9: The effect of a larger number of hypotheses on the development set WER results. The n-best lists were generated from a first-pass decoding with an n-gram, which achieved a WER of 29.0 in this experiment.

# of training steps	Devel WER
130k	26.2
540k	26.0
800k	26.0

Table 10: The effect of further training on the development set WER results. The n-best lists were generated from a first-pass decoding with an n-gram, which achieved a WER of 29.0 in this experiment.

LM	Devel WER
n-gram	29.0
LSTM	**24.8**
LSTM + n-gram	25.1
Transformer-xl	26.0

Table 11: Comparison of the LSTM and transformer models in the 50-best hypothesis rescoring.

5.2.5 Discussion

5.3 Combined results

Table 12 lists the best results from the 3-pass decoding scheme.

5.4 Comparison of model sizes and training time

			WER (%)	
LM scoring	**Vocab**	**Model details**	**devel**	**eval**
n-gram scores	word	Enarvi et al. (2017)	29.8	31.7
	subword		29.1	31.7
NNLM-rescored lattices	word		25.6	27.9
	subword		**25.0**	**27.1**
n-gram scores	word	AM w/ i-x-vectors, 4-gram LM	23.7	25.1
	subword		24.5	25.2
NNLM-rescored lattices	word	LSTM sequence 15	**21.1**	22.4
	subword	LSTM sequence 40	**21.1**	22.3
Transformer-rescored 50-best list	subword	Transformer-XL	**21.1**	**22.2**

Table 12: The results of the best models.

Vocab	**Model**	**Number of parameters**
	DNN AM	17M
Word	LSTM LM	230M
	Transformer-XL	56M
Subword	LSTM LM	115M
	Transformer-XL	41M

Table 13: The number of learnable parameters in the neural network models after hyperparameter tuning.

6 Discussion

6.1 Possible future work

7 Conclusion

Spontaneous conversational Finnish is a challenging type of speech to recognise due to frequent dysfluencies in sentence structure as well as due to the use of various informal word forms. This thesis work was an effort to improve the ASR accuracy for the conversational DSPCON corpus. The aim was to evaluate recent acoustic and language modelling methods on the data. The main experiments included evaluating the effect of different speaker embedding approaches, and comparing transformer-XL LMs and LSTM LMs with word and subword vocabularies.

In the speaker embedding experiments, the use of i-vectors and x-vectors achieved similar results when trained on the same speech corpus. Both methods decreased the evaluation set WER around 3% when trained on the small Finnish conversational speech corpus, and roughly 7% when trained on the significantly larger, English VoxCeleb corpus. Although i-vectors and x-vectors achieve roughly equal results, the two methods differ in how they are trained or how they embed speakers or utterances in a vector space. It is therefore possible that they encode different types of information about the speakers or utterances. To assess whether the methods encode overlapping information, the two vectors were also concatenated and evaluated on the same ASR task. The concatenated i- and x-vectors achieved a small improvement compared to using only x-vectors (about 2%). However, this improvement was not statistically significant.

Besides the speaker embedding experiments, the acoustic model was tuned for the development data. Since the previous best results presented by Enarvi et al. (2017), the Kaldi recipes have been optimised further. Using the new Kaldi recipes also improved the results. In total, the ASR WER improved 21% (from 31.7% to 25.1%), when comparing the results achieved using n-gram LM scoring.

The language modelling experiments included evaluating different language models with word and subword vocabularies. In the lattice generation phase, evaluating different n-grams showed that a simple 4-gram LM achieved the best WER results for both word-based and subword-based systems.

The LSTM language models were used to rescore the lattices generated using the n-gram language models. It was hypothesised that the sequence length of the context ought to be longer for the subword-based systems, since the tokens are shorter. This was shown to be the case: increasing the sequence length from 25 to 40 improved the WER. Perhaps more surprisingly, the word-based systems benefited from decreasing the sequence length from 25 to 15.

Transformer-XL models were used for rescoring n-best lists generated using n-gram or LSTM LMs. For the word-based system, rescoring the n-best lists with a transformer LM did not improve the results. Rescoring a 50-best list generated after the LSTM rescoring improved the best result by a small margin (0.5%) for the subword-based system, but this was not a statistically significant improvement. The inefficacy of the transformer LM is probably due to the use of n-best lists instead of lattices as well as the small size of the language modelling corpus.

Combining the best acoustic and language models built during this thesis work achieved a WER of 22.2% on the conversational Finnish evaluation set, improving

the WER by 18% compared to previous best result.

References

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: A general and efficient weighted finite-state transducer library. In Holub, J. and Žďárek, J., editors, *Implementation and Application of Automata*, pages 11–23, Berlin, Heidelberg. Springer Berlin Heidelberg.

Anastasakos, T., McDonough, J., Schwartz, R., and Makhoul, J. (1996). A compact model for speaker-adaptive training. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, volume 2, pages 1137–1140. IEEE.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bahl, L., Brown, P., De Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 49–52. IEEE.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bourlard, H. A. and Morgan, N. (2012). *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media.

Campbell, W. M., Sturim, D. E., Reynolds, D. A., and Solomonoff, A. (2006). Svm based speaker verification using a gmm supervector kernel and nap variability compensation. In *2006 IEEE International conference on acoustics speech and signal processing proceedings*, volume 1, pages I–I. IEEE.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE.

Chen, G., Xu, H., Wu, M., Povey, D., and Khudanpur, S. (2015). Pronunciation and silence probability modeling for ASR. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. *Harvard Computer Science Group Technical Report TR-10-98*.

Chung, J. S., Nagrani, A., and Zisserman, A. (2018). Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622*.

Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. *arXiv preprint cs/0205057*.

Creutz, M. and Lagus, K. (2007). Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4(1):1–34.

Crystal, D. and Potter, S. (2020). English language. *Encyclopædia Britannica*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., and Ouellet, P. (2010). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Enarvi, S. et al. (2018). Modeling conversational finnish for automatic speech recognition.

Enarvi, S., Kurimo, M., et al. (2013). Studies on training text selection for conversational finnish language modeling. In *Proceedings of the 10th International Workshop on Spoken Language Translation (IWSLT 2013)*, pages 256–263.

Enarvi, S., Smit, P., Virpioja, S., and Kurimo, M. (2017). Automatic speech recognition with very large conversational Finnish and Estonian vocabularies. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(11):2085–2097.

Gage, P. (1994). A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Gales, M. J. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, 12(2):75–98.

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hirsimäki, T., Creutz, M., Siivola, V., Kurimo, M., Virpioja, S., and Pylkkönen, J. (2006). Unlimited vocabulary speech recognition with morph language models applied to finnish. *Computer Speech & Language*, 20(4):515–541.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Irie, K., Zeyer, A., Schlüter, R., and Ney, H. (2019). Language modeling with deep transformers. *arXiv preprint arXiv:1905.04226*.

Iskra, D., Grosskopf, B., Marasek, K., Heuvel, H., Diehl, F., and Kiessling, A. (2002). Speecon-speech databases for consumer devices: Database specification and validation.

Jiang, H. (2010). Discriminative training of hmms for automatic speech recognition: A survey. *Computer Speech & Language*, 24(4):589–608.

Juang, B.-H., Hou, W., and Lee, C.-H. (1997). Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio processing*, 5(3):257–265.

Jurafsky, D. and Martin, J. H. (2019). *Speech and Language Processing (3rd ed. draft, 16th Oct 2019)*. Web access: `https://web.stanford.edu/~jurafsky/slp3/`.

Kenny, P. (2005). Joint factor analysis of speaker and session variability: Theory and algorithms. *CRIM, Montreal,(Report) CRIM-06/08-13*, 14:28–29.

Kenny, P., Boulianne, G., and Dumouchel, P. (2005). Eigenvoice modeling with sparse training data. *IEEE transactions on speech and audio processing*, 13(3):345–354.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE.

Ko, T., Peddinti, V., Povey, D., and Khudanpur, S. (2015). Audio augmentation for speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Ko, T., Peddinti, V., Povey, D., Seltzer, M. L., and Khudanpur, S. (2017). A study on data augmentation of reverberant speech for robust speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5220–5224. IEEE.

Lember, J., Koloydenko, A., et al. (2008). The adjusted Viterbi training for hidden Markov models. *Bernoulli*, 14(1):180–206.

Lennes, M. et al. (2009). Segmental features in spontaneous and read-aloud finnish. *Phonetics of Russian and Finnish general description of phonetic systems: experimental studies on spontaneous and read-aloud speech*.

Martínez, A. M. and Kak, A. C. (2001). Pca versus lda. *IEEE transactions on pattern analysis and machine intelligence*, 23(2):228–233.

Miao, Y., Zhang, H., and Metze, F. (2015). Speaker adaptive training of deep neural network acoustic models using i-vectors. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):1938–1949.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mohri, M., Pereira, F., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer.

Nagrani, A., Chung, J. S., and Zisserman, A. (2017). Voxceleb: a large-scale speaker identification dataset. *arXiv preprint arXiv:1706.08612*.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Povey, D. (2005). *Discriminative training for large vocabulary speech recognition*. PhD thesis, University of Cambridge.

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

Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motlíček, P., Qian, Y., et al. (2012). Generating exact lattices in the wfst framework. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4213–4216. IEEE.

Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech*, pages 2751–2755.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rath, S. P., Povey, D., Veselỳ, K., and Cernockỳ, J. (2013). Improved feature processing for deep neural networks. In *Interspeech*, pages 109–113.

Rosenberg, A. E., Lee, C.-H., and Soong, F. K. (1994). Cepstral channel normalization techniques for hmm-based speaker verification. In *Third International Conference on Spoken Language Processing*.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Rouhe, A., Kaseva, T., and Kurimo, M. (2020). Speaker-aware training of attention-based end-to-end speech recognition using neural speaker embeddings. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7064–7068. IEEE.

Rownicka, J., Bell, P., and Renals, S. (2019). Embeddings for DNN speaker adaptive training. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 479–486. IEEE.

Schwartz, R., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. (1985). Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *ICASSP'85. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 10, pages 1205–1208. IEEE.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Shinoda, K. (2011). Speaker adaptation techniques for automatic speech recognition. *Proc. APSIPA ASC*, 2011.

Siivola, V., Creutz, M., and Kurimo, M. (2007a). Morfessor and varikn machine learning tools for speech and language technology. In *Eighth Annual Conference of the International Speech Communication Association*.

Siivola, V., Hirsimaki, T., and Virpioja, S. (2007b). On growing and pruning Kneser–Ney smoothed n-gram models. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(5):1617–1624.

Smit, P., Virpioja, S., Kurimo, M., et al. (2017). Improved subword modeling for wfst-based speech recognition. In *INTERSPEECH*, pages 2551–2555.

Snyder, D., Chen, G., and Povey, D. (2015). Musan: A music, speech, and noise corpus. *arXiv preprint arXiv:1510.08484*.

Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. In *Interspeech*, pages 999–1003.

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5329–5333. IEEE.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Stolcke, A. (2002). Srilm-an extensible language modeling toolkit. In *Seventh international conference on spoken language processing*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Toshniwal, S., Kannan, A., Chiu, C.-C., Wu, Y., Sainath, T. N., and Livescu, K. (2018). A comparison of techniques for language model integration in encoder-decoder speech recognition. In *2018 IEEE spoken language technology workshop (SLT)*, pages 369–375. IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Viikki, O. and Laurila, K. (1998). Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1-3):133–147.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Ragni, A., Valtchev, V., Woodland, P., and Zhang, C. (2015). *The HTK Book (version 3.5a)*.

Young, S. J. (1992). The general use of tying in phoneme-based hmm speech recognisers. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 1, pages 569–572. IEEE Computer Society.

Young, S. J., Odell, J. J., and Woodland, P. C. (1994). Tree-based state tying for high accuracy modelling. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.