TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Martin Talimets 163584IAPM

END-TO-END SPEECH RECOGNITION FOR ESTONIAN

Master's thesis

Supervisor: Tanel Alumäe
Senior Researcher

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Talimets 163584IAPM

TÄIELIKULT NÄRVIVÕRKUDEL PÕHINEV KÕNETUVASTUS EESTI KEELELE

Magistritöö

Juhendaja: Tanel Alumäe
Vanemteadur

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Martin Talimets

07.05.2018

Abstract

The thesis is about end-to-end speech recognition system for Estonian language. This method makes training a speech recognition system a lot simpler. It does not require many complex neural network layers or knowledge about a language to train.

We test a few different approaches for end-to-end speech recognition. These tested end-to-end systems include a recurrent neural network (RNN) with connectionist temporal classification (CTC) or an attention based encoder-decoder architecture. These methods result in much lower speech recognition accuracy than the traditional hybrid model that uses hidden Markov models and deep neural networks. Instead, we use a combination of both architectures.

Our experiments show that the proposed method for Estonian language speech recognition system does not perform as good as the traditional method yet. The trained model achieves a word error rate (WER) of 21.9% and a character error rate (CER) of 7.5% on test set. However, when combining this system with the traditional approach, it gave results that are statistically significant improvements to the current best Estonian speech recognition system. The achieved results were 12.0% for WER and 4.0% for CER.

This thesis is written in English and is 46 pages long, including 6 chapters, 10 figures and 4 tables.

Annotatsioon

Täielikult närvivõrkudel põhinev kõnetuvastus eesti keelele

Magistritöö eesmärgiks on uurida täielikult närvivõrkudel põhinevat kõnetuvastust eesti keelele. Täielikult närvivõrkudel põhineva kõnetuvastuse süsteemi treenimine on oluliselt lihtsam kui traditsioonilise sügaval närvivõrgul põhineva süsteemi puhul. See ei vaja mitmeid keerulisi närvivõrkude kihte või teadmisi treenitava keele sõnade, käänete või kasutuse kohta.

Antud magistritöös testitakse erinevaid lähenemisi täielikult närvivõrkudel põhineva kõnetuvastuse süsteemi jaoks. Kõik testitavad süsteemid baseeruvad rekurrentsetel närvivõrkudel, millele on lisatud *connectionist temporal classification* (CTC) või tähelepanu mehhanismil põhinev enkooder-dekooder arhitektuur (*attention based encoder-decoder architecture*). Antud süsteemide tulemused olid silmnähtavalt halvemad traditsioonilisest kõnetuvastusest, mis põhineb sügaval närvivõrgul koos varjatud Markovi mudeliga. Seetõttu kasutame arhitektuuri, mis koosneb rekurrentsest närvivõrgust koos CTC ja tähelepanu mehhanismil põhineva enkooder-dekooder arhitektuuri hübriidiga.

Tehtud eksperimendid näitavad, et välja pakutav täielikult närvivõrkudel põhinev kõnetuvastuse süsteem ei saavuta nii häid tulemusi nagu traditsiooniline kõnetuvastus. Treenitud mudel saavutas sõnade veamääraks (*word error rate*) 21,9% ja tähtede veamääraks (*character error rate*) 7,5% testandmete peal. Traditsioonilise ja täielikult närvivõrkudel põhineva kõnetuvastuse kombineerimisel saavutatud tulemused olid statistiliselt oluliselt paremad praegusest parimast eesti keele kõnetuvastuse süsteemist. Antud süsteem saavutas sõnade veamääraks 12,0% ja tähtede veamääraks 4,0%.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 46 leheküljel, 6 peatükki, 10 joonist, 4 tabelit.

List of abbreviations and terms

BLSTM	*Bidirectional Long Short-Term Memory*
BRNN	*Bidirectional recurrent neural network*
CER	*Character error rate*
CTC	*Connectionist temporal classification*
DNN	*Deep neural network*
HMM	*Hidden Markov model*
LAS	*Listen, attend and spell*
LSTM	*Long Short-Term Memory*
LVCSR	*Large Vocabulary Continuous Speech Recognition*
WER	*Word error rate*
WTN	*Word transition network*
RNN	*Recurrent neural network*

Table of contents

List of figures

List of tables

1 Introduction

The thesis investigates using end-to-end speech recognition system for Estonian language. The model is trained with ESPnet end-to-end speech recognition toolkit using the data acquired mostly from Estonian TV or radio programs and news. It mainly consists of a deep neural network and a hybrid CTC/attention based encoder-decoder architecture.

This chapter describes the problem and methodology for solving it. It outlines the basic differences between using the traditional and end-to-end approach for speech recognition. We define the problem, set a hypothesis for solving it and set target goals for validating the results.

1.1 Problem

The thesis investigates using end-to-end speech recognition for Estonian language. End-to-end method has not been previously researched on Estonian language.

The current research on speech recognition for Estonian language is done using the traditional approach. Traditional speech recognition systems are currently based on very complex components. These components are acoustic models based on hidden Markov models, Gaussian mixture models, deep neural networks, n-gram and neural network based language models, complicated training and decoding algorithms. End-to-end speech recognition replaces all these different components in the traditional pipeline with a single end-to-end deep recurrent neural network (RNN).

1.2 Objective

The objective for this thesis is to test a deep recurrent network model with hybrid CTC/attention based encoder-decoder architecture for Estonian speech recognition.. The model is trained using about 216 hours of Estonian speech recordings from approximately 3500 unique speakers.

The results are evaluated using word error rate and character error rate on n-best lists. Word error rate measures the accuracy by comparing the exact match of words. Character error rate compares each character individually, which usually gives a correctness score better than with word error rate. These scores are calculated for end-to-end speech recognition system and then combined with the traditional approach.

The goal is to achieve a word error rate below 25% and a character error rate below 5% for only using end-to-end speech recognition. For combining end-to-end with traditional approach, the goal is to achieve better results than with traditional approach alone.

1.3 Methodology

The model is trained using different end-to-end speech recognition systems. These systems all include a deep neural network. The difference is the method used for scoring the output of a neural network.

Various systems are tested to see which architecture works best. The results from the best working model are then combined with the results from existing traditional approach. All results are compared and analysed to emphasize main differences between different systems.

1.4 Outline

The first chapter describes the problem and methodology for solving Estonian language end-to-end speech recognition. It outlines the basic differences between using the traditional and end-to-end approach for speech recognition. We define the problem, set a hypothesis for solving it and set target goals for validating the results.

The second chapter gives a detailed overview of the main component used for using an end-to-end speech recognition system, e.g. feature extraction, RNN, LSTM, CTC, attention based encoder-decoder and evaluation metrics.

The third chapter introduces previous related works that are related to this thesis' problem. These alternative works have used end-to-end speech recognition for other languages. The only Estonian language related work concentrates on the traditional method for speech recognition.

The fourth chapter specifies the solution part of the thesis. It gives a detailed overview of the methods used with training the end-to-end speech recognition system. The hybrid CTC/attention based end-to-end architecture and ROVER method is explained.

The fifth chapter is about the experiments. It describes the data used in the experiments, why the ESPnet toolkit is chosen, the used model for training, the results of the experiments and the analysis of the results.

The sixth chapter draws conclusion on how the end-to-end speech recognition system performs on Estonian language. The objectives and hypothesis are analysed whether they were achieved as initially planned.

2 Background

This chapter gives a detailed overview of the main component used for using an end-to-end speech recognition system, e.g. feature extraction, RNN, LSTM, CTC, attention based encoder-decoder and evaluation metrics.

2.1 End-to-end speech recognition

Speech recognition is the process of automatically extracting the word conveyed by speech wave. Traditional speech recognition systems are currently based on very complex components. These systems use components such as hidden Markov models, Gaussian mixture models, deep neural networks, n-gram and neural network based language models. End-to-end speech recognition replaces all these different components in the traditional pipeline with a single end-to-end deep recurrent neural network.

End-to-end speech recognition does not know anything about words or how they are used. It tries to guess each letter on a given audio and combine those to form words. This is different from the traditional approach where vocabulary and n-grams are used. The traditional approach then tries to calculate probabilities of what a person might have said compared to a given vocabulary and n-grams.

The main benefit of using end-to-end speech recognition is that is simplifies the process of training and deployment. Because of the fact that end-to-end does not need a vocabulary or n-gram, it can be used with different languages more easily, when only training data is available. It will also simplify deployment on mobile devices, because it does not use a typical n-gram language model, which takes a lot of disk space.

End-to-end speech recognition system has one important downside. Even though training the system is a lot simpler than in traditional approach, it needs a lot of training data. The system will have to learn different characteristics of a language by itself and that is why it needs to have a sufficient amount of data. Usually, the system needs to have thousands of hours of data to train on. It is possible to train with less data, but the results will not be as good as the system is capable of achieving.

End-to-end speech recognition is mainly based on deep recurrent neural network. A deep recurrent neural network alone is not enough for a speech recognition system. The first attempts used CTC, but it is incapable of learning the language and needs language model to clean up common mistakes. Instead of CTC, attention-based models were tested and they proved to be outperforming previous models, due to the ability of learning all components of a speech recognizer. Both methods still have some benefits over one another and recent works have started using a hybrid CTC/attention based architecture, which is also used in this thesis.

2.2 Feature extraction

The first step in recognizing speech from audio is to extract features. Feature extraction is for removing background noise, emotion and all other useless information to get only the components that can be used for identifying linguistic content. This pre-processing is needed for making neural network's processing easier to recognize text from audio [1].

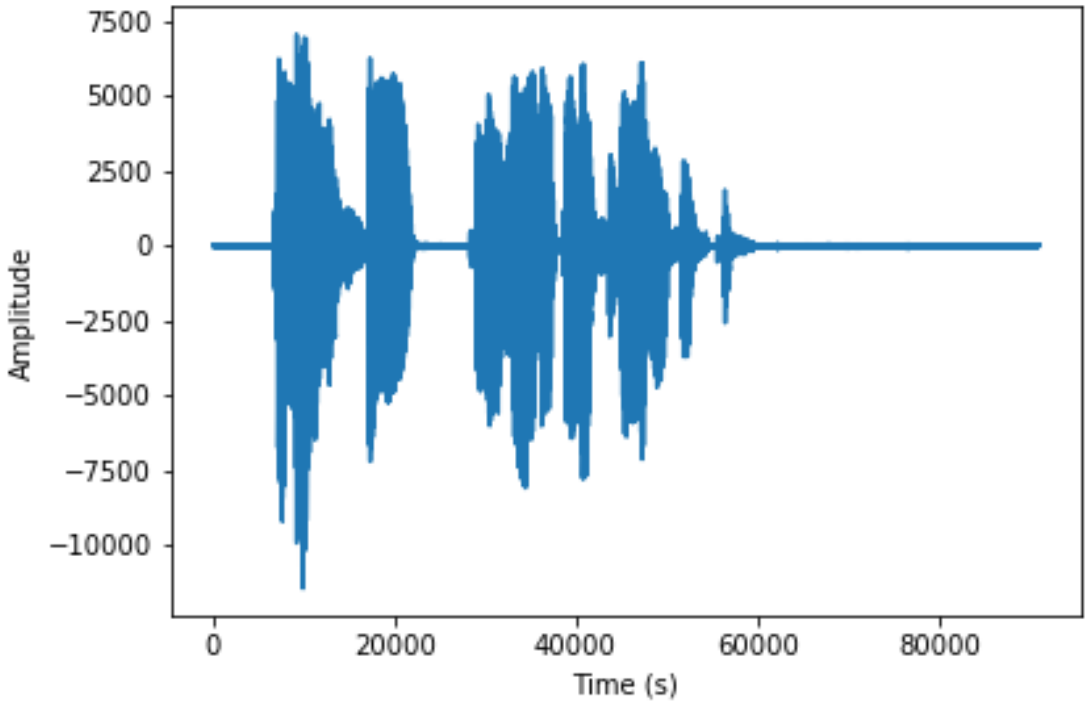

Figure 1. An audio signal wave

For extracting features from audio, the signal is usually framed into 20-40ms frames. Each frame is selected after 10ms, which means that the next frame will have some of its

contents from the previous frame. All of these frames still contain redundant information that require filtering. To remove all sudden endings of an audio signal in each frame, a window function is used, such as the Hamming window. The Hamming window function also allows to counteract the assumption made by the discrete Fourier transform that the signal is infinite and to reduce spectral leakage.

By using discrete Fourier transform, each complex sound wave is broken apart into simple sound waves. It takes a windowed signal as input and outputs a complex number for each frequency band. Each of those sound waves' contained energy is added up to get a score of how important each frequency band is. To better visualize the output of this process, a spectrogram is created using each frame's contained energy scores.

After using the discrete Fourier transform, the spectrogram still has too much information. Triangular filters are applied on a Mel-scale to the power spectrum for extracting frequency bands. The Mel-scale's purpose is to mimic the non-linear human ear perception of sound. Lower frequencies are filtered with narrower and higher frequencies with wider bands. Each of those filters collect energy from a number of frequency bands in discrete Fourier transform.

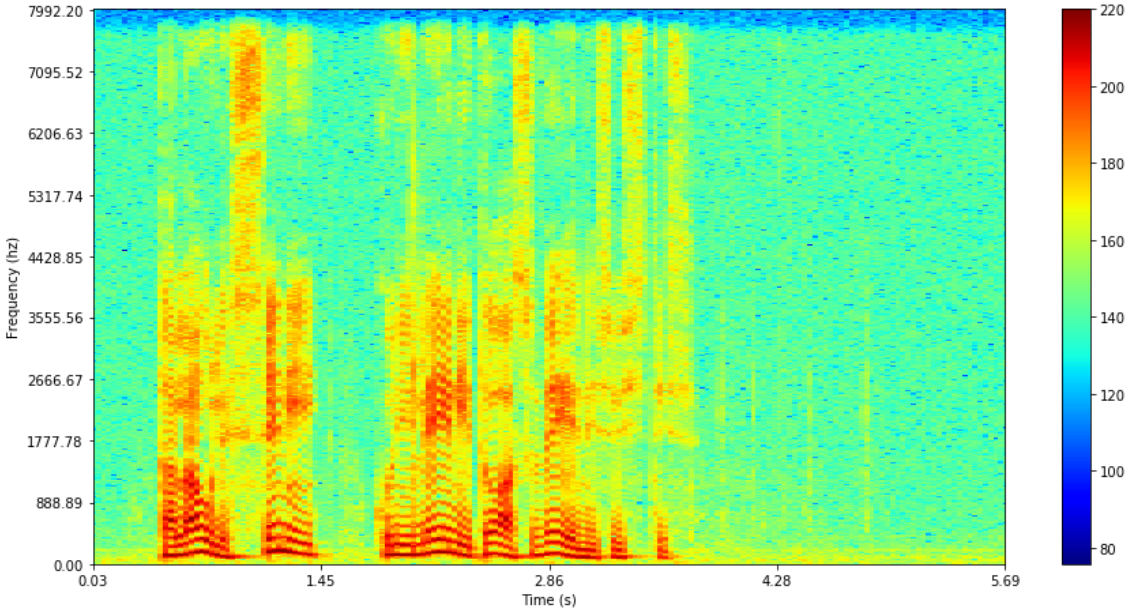

Figure 2. Spectrogram of an audio clip

The spectrogram in Figure 2 visualizes the patterns of low and high pitch frequency ranges. This data is better for a neural network to process, because it can find patterns

more easily. That is why this is the actual representation of audio data that gets fed into the neural network. The neural network will then try to figure out the best possible letter for each of these 20ms frames.

2.3 Recurrent neural network (RNN)

RNN is a type of deep learning model that works best for handling sequential information. RNN assumes, that all inputs and outputs are dependent on each other, unlike the traditional neural network. It keeps a memory of previous outputs and passes those as inputs from one step of the network to the next (Figure 3). This way the network can have a deeper understanding of the statement [2].

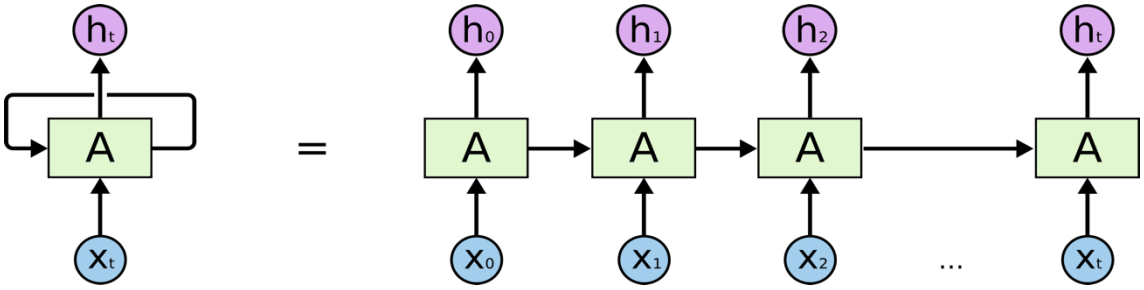

Figure 3. An unrolled RNN [3]

The above figure shows a chunk of neural network (A), that takes (X_t) and previous output as inputs and outputs a value (h_t). The recurrence allows the network to pass information from one step to the next [3]. This is the basic workflow of a RNN, but it is often used with bidirectional to get more accurate results.

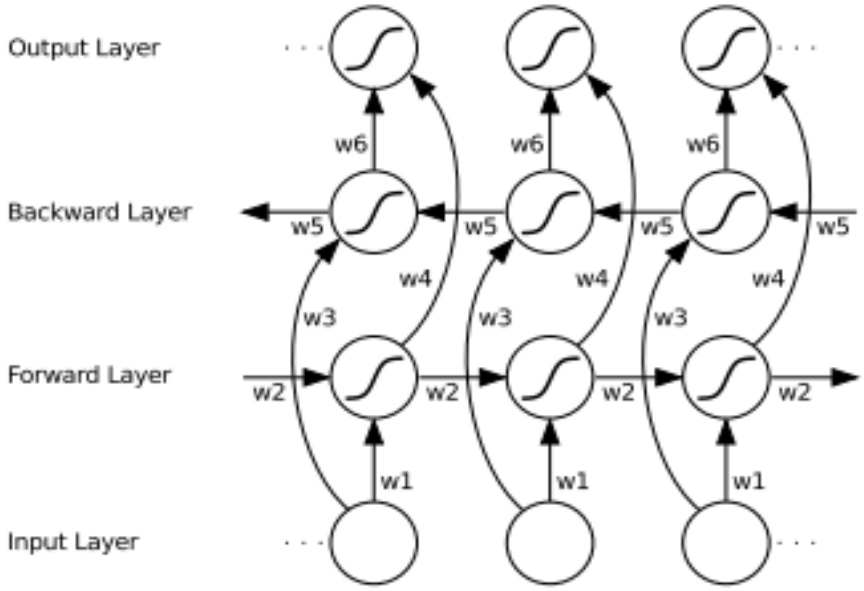

Figure 4. Bidirectional RNN (BRNN) [4]

It is often beneficial to know some information about the future as well as past context. Usually, when classifying a letter in a word, it is beneficial to know about the previous and next letters. This is something that bidirectional RNNs will help to solve. The idea behind BRNN is to have two hidden layers (Figure 4), one for forward and one for backward layer. This way the output layer will have both the past and future context for every point in the input sequence [4].

Standard RNN does not always perform very well. The problem is that RNNs cannot preserve memory from far away in the sequence. RNN makes predictions based on the most recent sequences. This means that the context about the start of a sentence might be lost while predicting the end of the sentence. To solve this problem, RNN is often used with long short-term memory (LSTM) architecture to have the context of a whole sentence always available in memory [5].

2.4 Long Short-Term Memory (LSTM)

Standard RNN architectures have a problem with multiple hidden layers. When passing information from one hidden layer to another, the information might get lost, if there are many layers. LSTM handles this kind of situation and enables RNN to preserve memory throughout the whole learning process.

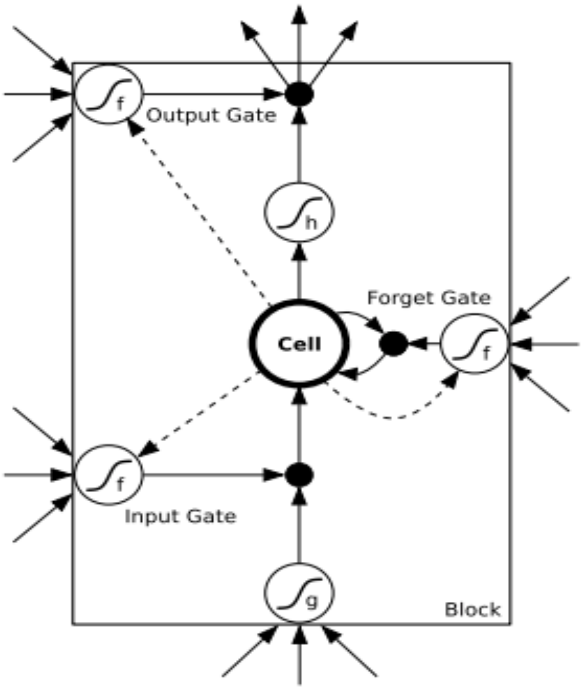

Figure 5. LSTM memory block with one cell [4]

LSTM architecture consists of memory blocks, which are all recurrently connected to each other. Each memory block contains at least one self-connected memory cell [6]. The memory cell allows information to be stored in, written to or read from. It also decides, which information to store and when to allow reading, writing and erasing. This is done using input, output and forget gates that open and close as shown in Figure 5.

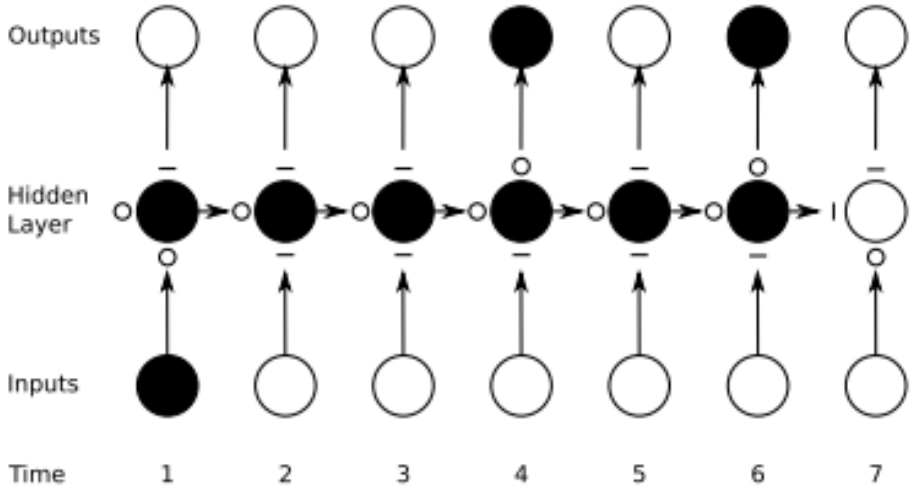

Figure 6. Preservation of gradient information by LSTM [4]

Figure 6 shows the preservation of gradient information by LSTM. Input state, forget and output gates are shown below, to the left and above the hidden layer respectively. (O) represents an entirely open gate and (-) is for a closed gate. Looking at the diagram, LSTM memory cell can preserve information as long as the input gate is closed and forget gate is open. Getting information from output gate does not have any affect memory cell's data.

When using LSTM architecture with bidirectional RNN gives bidirectional LSTM (BLSTM) [4]. Using BLSTM allows to preserve information from the past as well as from the future. This is important, when the understanding of past and future context is needed to find the correct next word in any time.

For better understanding of BLSTM, it can be explained with a simple speech recognition example. Let us say, we need to detect the next word for a sentence starting with "I will go to ". Currently the only available information about the sentence's context is in the past. Finding the correct next word can be difficult, when there are almost limitless possibilities. Now, the BLSTM allows to get context from the future as well. When the sentence continues with "and learn machine learning", the detection for the missing word becomes simpler, because of the extra context about the whole sentence [7].

2.5 Connectionist temporal classification (CTC)

People talk with very different rates of speed which makes training an ASR system a lot more difficult. That is why the alignment between characters in the transcript and audio is always unknown. One way of solving this problem is to manually align all characters to their location in the audio. The major downside is that it's very time consuming when dealing with large datasets. Another option is to use connectionist temporal classification (CTC) which has become a very popular among RNNs [8].

CTC is a type of neural network output and associated scoring function. It is used with RNNs to handle sequential problems. CTC sums over the probability of all possible alignments between the input and the output [4]. Assuming that an input has a length greater than the actual word's length, one option for solving the problem is to collapse all repeating characters.

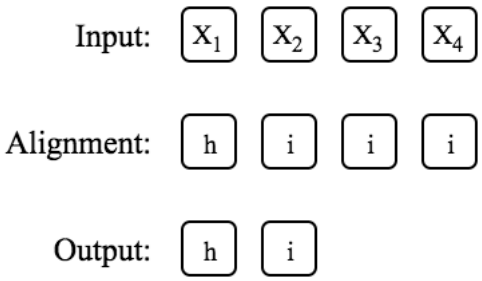

Figure 7. Merging repeated characters

Collapsing all repeating words is not the best way of tackling this problem, because it will remove all repeating characters even when there should be a repetition. That is why CTC uses a token called *blank* (here referred as _). This token is always removed from the output, but used in alignment process.

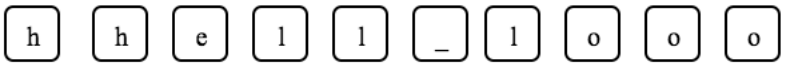

Figure 8. Merging repeated characters with *blank* token

Firstly, CTC merges all repeating characters and secondly, removes all *blank* tokens. The remaining output will be ‘hello’ not ‘helo’, which would be the output without the token.

There are a lot of possible ways of character alignment for every input. CTC loss functions combines all alignments where the output is the same. It then calculates the score for each of these combinations and sums over all scores. While decoding, a character with the highest score for each time step is picked. After that, the duplicate characters are merged and *blank* tokens removed to get the final output [9].

2.6 Attention based encoder-decoder architecture

Encoder-decoder architectures are mostly used to deal with sequences where the input and output length size in unknown. Both encoder and decoder are RNNs. The encoder transforms the input to a higher level representation where the length size is fixed. The decoder then uses this representation and generates output sequences.

When dealing with a simple encoder-decoder architecture, the decoder generates a transcription based on the last hidden state from the encoder. This is not a reasonable approach, because when dealing with sentences containing many words, the encoder will have to encode every information into a single vector. The decoder must then produce a valid output only based on this single vector. While decoding, the decoder has to consider information from the beginning of the sentence and when dealing with RNNs, it is known that long-range information might get lost.

Attention based encoder-decoder architecture solves the problem of encoding everything into one single vector. The attention mechanism allows decoder to get information from all parts of the source sentence at every step of output generation. The model will learn by itself what information is important and should be considered. Each decoder output does not depend on the last vector anymore, but instead on a weighted combination of all the input states [10].

2.7 Evaluation metrics for speech recognition

2.7.1 Word error rate (WER)

WER is a method for calculating the performance of a speech recognition software. This is not always easy to measure, because the correct input length can be different from the detected value length.

$$WER = \frac{S + D + I}{N}$$

Equation 1. WER calculation

Equation 1 shows the equation for calculating WER. S shows the number of substitutions, D is for deletions, I is for insertions and N is for words in the reference [11].

There are three possibilities for an automated speech recognition (ASR) software to make mistakes that WER will calculate:

1) Deletion – ASR system deletes a word
   Correct input: Machines can think
   ASR result: Machines think

2) Insertion – ASR system inserts an unneeded word
   Correct input: Machines can think
   ASR result: Machines can not think
3) Substitution – ASR system substitutes a correct word with an incorrect one
   Correct input: Machines can think
   ASR result: Machines can learn

WER can sometimes give very unreasonable results when dealing with compound words. Sentences like “water melon tastes good” and “watermelon tastes good” are both very well understandable. But the calculated WER would be 50%, which is unfair considering that the actual mistake is only adding an unnecessary white space. This is where character error rate will give more adequate results.

2.7.2 Character error rate (CER)

Another method for calculating the performance of ASR is character error rate. CER is calculated with the minimum number of operations necessary to transform the original text into ASR output. The smaller the number, the more accurate both texts are.

The equation for calculating CER is the same for WER as shown in Equation 1. But for CER, N is for the total number of characters and the minimal number of character substitutions as S, deletions as D and insertions as I, required for transforming original text into automatic transcription [12].

White space and case are also important for CER. While contiguous white spaces are usually considered as one, a word pair “auto mobile” with more than one space between them still gives an accuracy of 10%. When comparing words with different case like “Hello World” and “hello world”, CER sees them as substitutions and calculates an accuracy of 18%.

CER is most commonly used when dealing with languages that have difficult declensions. When the original reference in Estonian is “koerast” and ASR recognises it as “koeras”, the WER would be 100%, but CER is only 14%. For these kind of languages, where a word has many different cases, WER might show a bit unfair results compared to CER. Although the result of WER is high, the word is still readable and in the meaning of the sentence would still be understandable.

3 Related works

The problem of using end-to-end speech recognition for Estonian language has not been investigated in any earlier research.

This chapter introduces previous related works that are related to this thesis' problem. These alternative works have used end-to-end speech recognition for other languages. The only Estonian language related work concentrates on the traditional method for speech recognition.

3.1 End-to-end speech recognition

Deep Speech 2: End-to-end Speech Recognition in English and Mandarin was created in 2015 to show the possibilities of implementing end-to-end speech recognition on very different languages [13]. The system consists of three main components:

1) RNN with one or more convolutional input layers
2) Multiple recurrent layers and one fully connected layer
3) CTC

For training the models, this research uses 11940 hours of labeled speech, which contains 8 million utterances, for English model and for Mandarin, there are 9400 hours of labelled speech, which contains 11 million utterances.

The trained model's WER for English language is comparable with human WER, when the audio is clearly understandable. In these cases, the WER differs between 3-13% using different datasets. When testing with accented or noisy audio, the WER becomes understandably bigger. The difference between human level and the trained model becomes clearer when dealing with accented or noisy audio.

The results for Mandarin language show that end-to-end speech recognition can give better results than an average human speaker. When transcribing short voice-query like utterances, the trained system for Mandarin language works better than human level

performance. The system achieves a WER of 3.7% for 100 random utterances labelled by a committee of 5 and 5.7% for 250 utterances labelled by a single person. A typical Mandarin Chinese speaker achieves approximately 4% for committee labelled utterances and 9.7% for utterances labelled by an individual.

Listen, attend and spell (LAS) is research done in 2015 and has a key improvement over previous end-to-end CTC models. LAS uses a neural network, that transcribes speech utterances to characters. The system has two components: a listener and a speller, which are both jointly learned. The listener is a pyramidal recurrent network encoder that uses filterbank spectra for inputs. The speller is an attention-based recurrent network decoder that sends out characters as outputs [14].

Without using a language model or a dictionary, LAS achieves a WER of 14.1% on a subset of the Google voice search task. The result is not as good as the traditional DMM-HMM models, but still quite good for a system that has not been fully researched and developed.

There have also been many other recent researches about end-to-end speech recognition using LAS, such as [15] [16] [17] [18] [19].

Joint CTC/attention decoding for end-to-end speech recognition is another research for end-to-end speech recognition created in 2017 [20]. Previous works on end-to-end ASR systems have used either CTC or attention architecture. This research has created an end-to-end speech processing toolkit called ESPnet which proposes a hybrid CTC/attention architecture to utilize both advantages in decoding [21].

The testing is done on spontaneous Japanese and Mandarin Chinese datasets. For getting better results, the train set is expanded by linearly scaling the audio lengths by factors of 0.9 and 1.1. It eventually achieved a WER of 29.9% which is better than systems using only CTC.

Using CTC in end-to-end speech recognition is also researched by many others, such as [22] [23] [24] [25] [26].

3.2 Estonian language

Recent improvements in Estonian LVCSR is a paper from 2014 by Tanel Alumäe which uses the traditional method for solving Estonian language speech recognition [27]. This paper is from 2014.

This paper describes a speech-to-text transcription system for semi-spontaneous speech. The system is based on the Kaldi toolkit and uses deep neural network based hidden Markov models (DNN-HMM) as main acoustic models. For restoring the final lattices, the system uses neural network based phone duration models, which gives significant improvements over the basic DNN-HMM architecture.

For training the model, over 100 hours of speech was transcribed and used. The audio contains various speakers and no special processing has been made with it. This system achieves WER of 17.9% on broadcast conversations and 26.3% on conference speeches.

4 Method

This chapter specifies the solution part of the thesis. It gives a detailed overview of the methods used with training the end-to-end speech recognition system. The hybrid CTC/attention based end-to-end architecture and ROVER method is explained.

4.1 Hybrid CTC/attention-based end-to-end architecture

In machine translation, where word order for input and output can be different, the attention-based encoder-decoder works fairly well. It allows nonsequential alignments between each element of the output sequence and acoustic encoder network generated hidden states for each frame of acoustic input. But for speech recognition, word order is the same for input and output except some small within-word deviations that may happen.

Another problem is the different lengths of input and output sequences. The difference in length comes from each speaker's speaking rate and writing system. That makes it difficult for the ASR to track the alignment between input and output. The attention mechanism could solve all these problems, but for better results, a CTC-based alignment will be used for training the model [28].

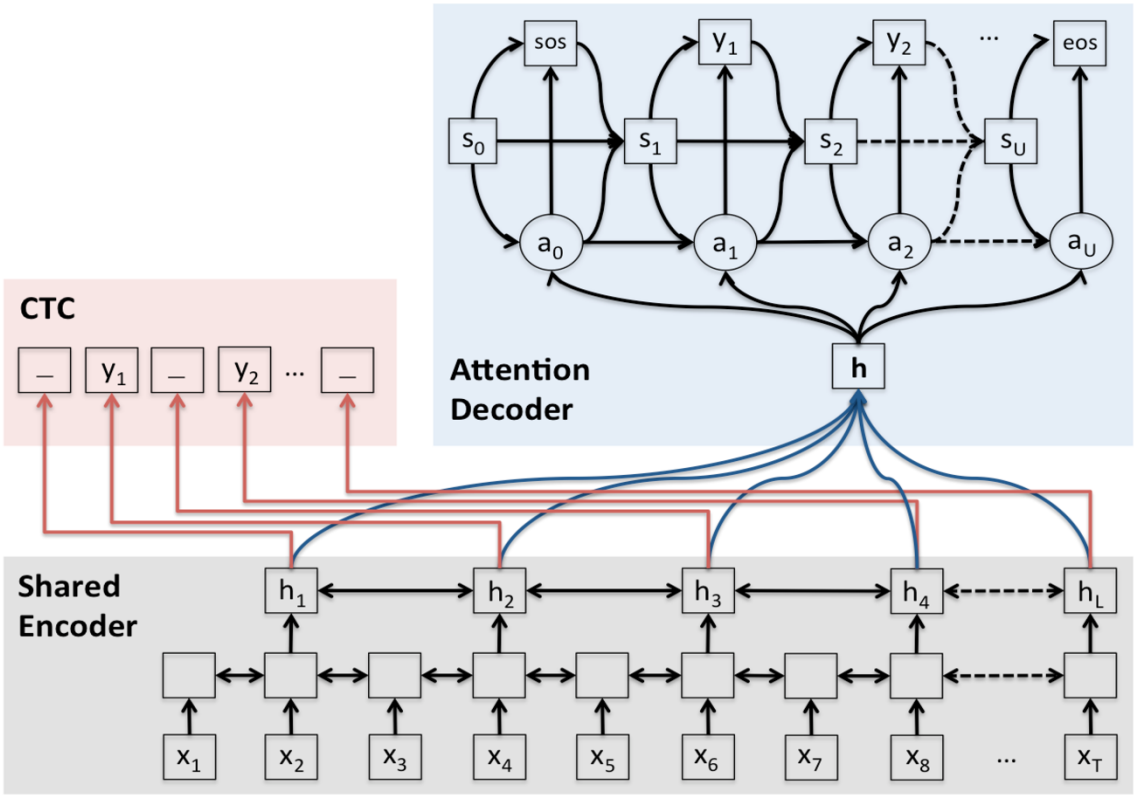

Figure 9. Hybrid CTC/attention-based end-to-end architecture [20]

Hybrid CTC/attention-based end-to-end architecture, as shown in Figure 9, solves both the word order and alignment problem between input and output. This architecture uses a CTC objective function as an auxiliary task for training the attention model encoder. The BLSTM encoder network is shared between CTC and attention model.

The decoding process uses both attention-based and CTC scores. Because of the fact that CTC and attention-based decoder computes scores differently, combining them is nontrivial. A rescoring/one-pass beam search algorithm is used to combine those scores. The outcome of this would eliminate all irregular alignments.

Using this joint architecture, the learning process of the network is quicker and it works better in noisy conditions or with long sentences. The forward-backward algorithm of CTC enforces monotonic alignment between speech and label sequences. This helps to acquire more accurate alignments in noisy conditions. Using CTC as an auxiliary task also improves the speed in estimating alignments without the aid of rough estimates. That way the estimations for alignments in long sequences are not solely dependent on data-driven attention methods [20].

4.2 Combining n-best lists with ROVER method

N-best list is generated by the ASR system and it contains a list of likely possibilities for input sentence which is sorted by the best score. Each possibility is different and has a score of how sure the system is in its correctness. N-best list allows to combine multiple different ASR systems to achieve better results.

For combining multiple n-best lists, a ROVER system is used. The first step for this system is to align all hypothesis transcripts from ASR systems to get one word transition network (WTN). It firstly creates WTNs for all ASR system outputs to be able to combine them.

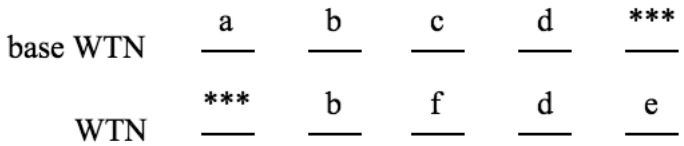

Figure 10. WTNs alignment

A base WTN is chosen from which the composite WTN is developed. All other WTNs are aligned according to the base WTN as shown in Figure 10. For example, if there are 3 different systems, a base WTN is chosen and then one of the remaining WTN is aligned with the base WTN to form a new base WTN. The process is repeated with all other remaining WTNs to eventually get one final composite WTN.

When the final composite WTN is found, a voting module is used to find the best scoring word sequence. The voting module finds the occurrences of each word and accumulates them.

$$Score(w) = \alpha\left(\frac{N(w,i)}{Ns}\right) + (1-\alpha)C(w,i)$$

Equation 2. Scoring formula

The Equation 2 show how the voting is performed. The number of occurrences of word type *w* is accumulated in correspondence set *i* in the array *N(w,i)*. To scale the frequency of occurrence to unity, the array is then divided by the number of combined systems *Ns*.

The measured confidence scores for word *w* create an array *C(w,i)*. The parameter α is trained to be the trade-off between using word frequency and confidence scores.

The voting can be done in three different ways. When setting the α parameter to 1, the information about confidence scores become irrelevant. This way the voting is made by frequency of occurrence. When training the parameter α a priori on the training data, the voting will use confidence scores to find either average or maximum confidence scores. The parameter α can be trained by quantizing the parameter space into a grid of possible WER values and then exhaustively searching for the lowest WER [29].

5 Experiments

This chapter is about the experiments. It describes the data used in the experiments, why the ESPnet toolkit is chosen, the used model for training, the results of the experiments and the analysis of the results.

5.1 Data

Table 1. Used dataset description

Origin	**Speech type**	**Hours**
Radio and TV programs, radio interviews	Semi spontaneous	109
Lectures and conference presentations	Not spontaneous	38
Radio and TV news	Dictated, semi spontaneous	30
A spontaneous phonetic speech corpus from University of Tartu	Spontaneous	29
Speech database from BABEL	Dictated	8
Android app “Kõnele” real spoken data	Dictated	2
Sum		216

This thesis uses about 216 hours of Estonian speech from approximately 3500 unique speakers. Most of the data contains spontaneous speech. Spontaneous speech gives better

results for training the model, because of the different talking speeds and pronunciations of words by various speakers.

Table 1 shows the distribution of used data over different origins. This dataset is divided between training, test and validation sets. Test and validation sets use data from TV news, interviews, talk shows and radio shows. Test data contains 8 hours and 27 minutes of data from approximately 170 unique speakers which is equivalent to 3027 utterances. Validation set contains 7 hours and 28 minutes of data from approximately 190 unique speakers which is equivalent to 2954 utterances. All other data is for training set.

5.2 Traditional approach for Estonian speech recognition

The previous work for solving Estonian speech recognition is made using traditional approach. It uses deep neural network based hidden Markov models (DNN-HMM) as main acoustic models. For better results, a neural network based phone duration models are used for rescoring final lattices.

The system uses the Kaldi toolkit and contains 43 phoneme models, a silence/noise model and a garbage model that collects foreign language and unintelligible words during training. A single silence/noise model is used to map different noises and fillers. The acoustic model is trained using Kaldi Switchboard recipe.

Language model for traditional approach is one of the key components. It consists of bigrams, trigrams and also 4-grams that occur more than once. Before creating the language model, a text normalization is performed, where recapitalization and changing numbers to words are mainly done. The language model vocabulary consists of 200 000 most likely case-sensitive compound-slip units.

This thesis uses n-best list from traditional speech recognition approach to combine this with end-to-end approach. The results for combining the results of two different systems are described in chapter 5.6.

5.3 Choosing the best library

Because of the increasing interest in developing end-to-end speech recognition systems, there are quite many projects related to it. This theses trained models on three different

systems. End-to-end automatic speech recognition system implemented in Tensorflow, an open source project called DeepSpeech by Mozilla and ESPnet end-to-end speech recognition toolkit [30] [31].

Training models on all three systems showed that ESPnet performs significantly better. The use of hybrid CTC/attention architecture has proven to perform better than CTC or attention based alone. ESPnet has got many benefits over the other two systems apart from just performing better. It has been developed by many highly valued experts with a lot of experience in the field of speech recognition. The documentation is very good and includes scripts with different training parameters for some popular data corpuses. One of the other advantages is the fact that ESPnet has similar structure to Kaldi toolkit and uses its functionality. Kaldi toolkit has been widely recognized and used by speech recognition researchers [32].

5.4 Model description

The model is trained by using various parameters. All parameters are taken from the ESPnet script that trains a model for TED talks corpus. This is done because the TED corpus is very similar to the data used in this thesis and has proven itself. Table 2 lists most important parameters used in training the model.

Table 2. Trained model parameters

Parameter	**Value**
Type of encoder network architecture	vggblstmp
Number of encoder layers	6
Number of encoder hidden units	320
Number of encoder projection units	320
Encoder subsampling	1_2_2_1_1
Type of CTC implementation to calculate loss	Chainer

Number of decoder layers	1
Number of decoder hidden units	300
Type of attention architecture	location
Number of attention transformation dimensions	320
Number of attention convolution channels	10
Number of attention convolution filters	100
Multitask learning coefficient	0.5
Batch size	30
Batch size is reduced if the input sequence length is greater than max length	800
Batch size is reduced if the output sequence length is greater than max length	150
Optimization	AdaDelta
Number of maximum epochs	15

5.5 Data augmentation

End-to-end speech recognition requires a lot of data to train a model that has as low as possible WER and CER. Collecting data for training is not that easy and takes a lot of time and resources. That is why data augmentation is used for generating more training data.

There are many options for data augmentation. This thesis uses speed modifications, noise and reverberation to generate more data. The original dataset is modified by decreasing speed by 10% and increasing it by 10%. Noise and reverberation is added for each of these datasets. The used background noises are extended by repetition to cover the whole audio input. Reverberations are only added at a specified time.

By doing these simple modifications, the initial dataset has given 5 more datasets that can all be used during model training. This kind of data augmentation gives more real life data to train on and the trained model actually achieves better results.

The model is trained using original dataset, original dataset with speed modifications and original dataset with speed modifications, noise and reverberation. The results are shown in Table 3.

Table 3. WER and CER results for different datasets

System	**Validation set**		**Test set**	
	WER %	**CER %**	**WER %**	**CER %**
Original dataset	24.7	8.2	23.3	7.8
Original dataset with speed modifications	24.2	8.1	23.5	8.0
Original dataset with speed modifications, noise and reverberation	22.8	7.7	21.9	7.5

As shown in Table 3, WER and CER improves when adding speed modifications to original dataset and testing against validation set. The results for testing against test set show slightly worse results. Using only speed modifications for simulating data does not always give better results. By adding noise and reverberation, the results show promising improvement for both test set and validation set. WER is improved by almost 2% and CER by 0.5%.

5.6 Combining n-best lists

N-best list allows to combine the result of different systems to achieve more accurate results. We use the ROVER method to combine n-best lists from the traditional and end-to-end system which contain 100 different predictions with confidence scores.

Table 4. WER and CER results for combined systems

System	**Validation set**		**Test set**	
	WER %	**CER %**	**WER %**	**CER %**
Traditional approach alone	12.4	4.7	12.7	4.5
Combined with end-to-end system	12.0	4.0	12.0	4.0

As Table 4 shows, the combined system achieves slightly better WER and CER. The combined system benefits from end-to-end system's ability of recognising unknown out of vocabulary words. Traditional approach outputs only words in its vocabulary, therefore commonly making mistakes with names and different cases of a word.

At first thought, the improvement might not look that great. But when looking at the relative improvement on test set, for example, WER improves by 5.5% and CER by 11.1%. This improvement is actually statistically significant according to speech recognition benchmark tests by Makhoul [33].

5.7 Analysis

This chapter analyses the results of previous work done with traditional approach (Kaldi), end-to-end approach (ESPnet) and the combined result of those two methods (ROVER).

Each example starts with the original reference (REF). Spelling errors and completely wrong words are underlined. Asterisk is used when a word is completely missing.

5.7.1 Combined n-best lists better than traditional approach

Combining n-best lists from Kaldi and ESPnet can sometimes improve WER or CER. Both methods have their advantages and when put together, can improve the outcome of

speech recognition. Here are some of the examples where ROVER gives better results than Kaldi.

REF: te rääkisite jah et pooltel tegevus aladel palgad tõusid pooltel ei tõusnud
KALDI: te rääkisite jah et pooltel tegevus aladel palgad tõusid POOLTELE ei tõusnud
ROVER: te rääkisite jah et pooltel tegevus aladel palgad tõusid pooltel ei tõusnud

REF: ja rüüstamistest on need on pildid londoni ees linnadest
KALDI: ja RÜÜSTAMISTE SEDA on need on pildid londoni ees linnadest
ROVER: ja rüüstamistest on need on pildid londoni ees linnadest

REF: eesti meedikud jõuavad afganistani lahingu tegevuse kõrg punktiks ja ühtlasi kõige palavamaks
KALDI: eesti meedikud jõuavad afganistani lahingu tegevuse kõrg punktiks ja ühtlasi kõige PALAVA MAKS
ROVER: eesti meedikud jõuavad afganistani lahingu tegevuse kõrg punktiks ja ühtlasi kõige palavamaks

REF: maht selliseks et see rahuldaks ka kõiki abi vajajaid
KALDI: maht selliseks et see RAHUL PEAKS ka kõiki abi EI VAJA
ROVER: maht selliseks et see rahuldaks ka kõiki abi EI VAJA

REF: aasta aegade vaheldudes kohtume me enamasti palju like
KALDI: aasta aegade VAHELDUS kohtume MEIE enamasti palju like
ROVER: aasta aegade vaheldudes kohtume MEIE enamasti palju like

REF: kosmose agentuur nasa kinnitas kosmose sondi doon jõudmist kääbus planeedi seres orbiidile
KALDI: kosmose AGENTUURI nasa kinnitas kosmose sondi TOON jõudmist KAEBUS planeedi SEE RES orbiidile
ROVER: kosmose AGENTUURI nasa kinnitas kosmose sondi TOON jõudmist KAEBUS planeedi seres orbiidile

REF: õppima saabuval noorel peab olema selge ette kujutus sellest mis teda ees ootab
KALDI: õppima saabuval noorel peab olema SELG ette kujutus sellest mis teda ees ootab
ROVER: õppima saabuval noorel peab olema selge ette kujutus sellest mis teda ees ootab

REF: tüdrukud olid vanematele öelnud et nad teevad päevase välja sõidu
KALDI: tüdrukud olid VANEMATEL öelnud et nad teevad päevase välja sõidu
ROVER: tüdrukud olid vanematele öelnud et nad teevad päevase välja sõidu

These examples show some of the benefits of combining the results from two different approaches to speech recognition. The most common mistake here, that Kaldi system does, is the use of wrong case for a word. Most of the times, ROVER will fix this when ESPnet has a better recognition for the word. The other problem is when Kaldi's language model has no knowledge about a word used in audio. Kaldi then proposes words close to the original, but is unable to come up with a correct word. ESPnet does not depend on language model and therefore ROVER will use a more accurate transcription for the input audio, when dealing with a word missing from train set.

5.7.2 Traditional approach better than combined n-best lists

Combining n-best lists does not always give better results. When combined systems have significantly different results for input, a wrong presumption could get selected by ROVER method. Some of these use cases are listed below.

REF: siis ma nagu pöördun komisjoni poole et las nemad siis seletavad
KALDI: siis ma nagu pöördun komisjoni poole et las nemad siis SELETAVAT
ROVER: siis ma nagu pöördun komisjoni poole et las NÄEVAD siis SELETAVAT

REF: raik küla lähedal oli leitud pink mis bio voolude abil ise liikuma hakanud
KALDI: raik küla lähedal oli LEIDNUD pink mis bio KUULUDA abil ise liikuma hakanud
ROVER: raik küla lähedal oli LEIDNUD pink mis PEO KUULUDA abil ise liikuma hakanud

REF: leiavad eeldatavalt kasutust riigi ameteis
KALDI: leiavad eeldatavalt kasutust riigi AMETIS
ROVER: leiavad eeldatavalt KASUTUSTE riigi AMETIS

REF: isas linde kohtab emastega just nii kaua kui neid tarvis on
KALDI: isas HINDA kohtab HAMMASTEGA just nii kaua kui NEED tarvis on
ROVER: ISA HINDA kohtab HAMMASTEGA just nii kaua kui NEED tarvis on

These results show that on some occasions, the ROVER method will choose the wrong result. This happens when one of the systems higher confidence in a word even when it is not a correct assumption. The system with a correct word might not be too confident about it and that is why it will not be selected by ROVER method.

5.7.3 Traditional approach better than end-to-end approach

This sections compares some of the results where traditional approach works better than end-to-end approach.

REF: tegelikult tähendab see väljend korraldust kõige kõrgemalt ülemuselt
KALDI: tegelikult tähendab see väljend korraldust kõige kõrgemalt ülemuselt
ESPNET: tegelikult tähendab see väljend korraldust kõige kõrgemalt ÜLAMUSELT

REF: riigi kogu liikmetele see eetika koodeks noh ütleme koostada ja kehtivaks tunnistada
KALDI: riigi kogu liikmetele see eetika koodeks noh ütleme koostada ja kehtivaks tunnistada
ESPNET: riigi kogu liikmetele * SEETIKA POODEKS noh ütleme koostada ja kehtivaks TUNNISTAB

REF: kas riigi kogu liikmetel on siis vaja seda eetika koodeksit üldse
KALDI: kas riigi kogu liikmetel on siis vaja seda eetika KOODEKS SIIT üldse
ESPNET: kas riigi kogu liikmetel on siis vaja * EETIKU KOODEKSITE üldse

REF: lihtsate lausetega saab jutu ära rääkida
KALDI: lihtsate lausetega saab jutu ära rääkida
ESPNET: LIHTSATELE ASETEGA saab JUTTU ära rääkida

REF: kolm ilvest ühe korraga
KALDI: kolm ilvest ühe korraga
ESPNET: kolm ILMEST ühe korraga

REF: õpilas kodude vajadus on igas maakonnas kõigest kolmekümne koha ringis
KALDI: õpilas kodude vajadus on igas maakonnas kõigest kolmekümne koha ringis
ESPNET: õpilas kodude VAEDUS on igas maakonnas kõigest kolmekümne koha ringis

REF: riik võib taolist tegevust hukka mõista aga tõele au andes
KALDI: riik võib taolist tegevust hukka mõista aga tõele au andes
ESPNET: riik võib TAOLIS tegevust hukka mõista aga TÕE LAUANDES

Traditional approach works better, when dealing with specific expressions. When an audio contains an expression made out of 2, 3 or more successive words that are also in Kaldi's vocabulary, the system easily outputs correct results. Because ESPnet does not have any vocabulary, it can make mistakes more often even with very simple and short words, when the word is not pronounced clearly or the audio has some background noise.

ESPnet also makes mistakes, when previous word ends with the same letter as the next word starts. In these situations, when the speaker speaks very fast or does not make a pause between those words, mistakes will often occur. Traditional approach knows how to fix these problems with misspelling more efficiently because of its vocabulary and language model.

5.7.4 End-to-end approach better than traditional approach

End-to-end speech recognition has its own advantages over traditional approach. Some of the examples, where end-to-end speech recognition works better are listed below.

REF: et kuidas on teie arvamus ivar tallo kas
KALDI: et kuidas on teie ARVAMUSI VALD ALLA kas
ESPNET: et kuidas on teie ARMAS ILMAR tallo kas

REF: jean paul nerriere'ile tuli mõte et keel mida üle ilmselt räägitakse
KALDI: RUUM POOLNE ERI AIRILE tuli mõte et keel mida üle ilmselt räägitakse
ESPNET: SOOM POOL NERIEERILE tuli mõte et keel mida üle ilmselt räägitakse

REF: ja selle kulutused suureneksid kümme protsenti
KALDI: ja selle kulutused SUURENESID kümme protsenti
ESPNET: ja selle kulutused suureneksid kümme protsenti

REF: ja üüri raha liigub omaniku taskusse sula rahas
KALDI: ja JÜRI raha liigub omaniku taskusse sula rahas
ESPNET: ja üüri raha liigub omaniku taskusse sula rahas

REF: kuidas paigutuvad kõiksuses ümber nii sugused asjad mida me nimetame pakenditeks
KALDI: kuidas PAIGUTAVAD KÕIKSUSE SEE ümber nii sugused asjad mida me nimetame PAKENDITE EKS
ESPNET: kuidas paigutuvad kõiksuses ümber nii sugused asjad mida me nimetame AKENDITEKS

REF: need eksisteerisid serbias georgias ukrainas
KALDI: need EI EKSISTEERI SIIT serbias georgias ukrainas
ESPNET: need eksisteerisid serbias georgias ukrainas

These results show that ESPnet performs better with words which are not in Kaldi's vocabulary. Especially when dealing with names, ESPnet will output more accurate result than Kaldi, which just outputs a word closest to the input its processing. ESPnet still makes mistakes transcribing names, because of the pronunciation. Also, recognizing unknown words is more accurate in end-to-end approach, because it will not try to find a similar word to it, when no match is found.

6 Summary

This chapter draws conclusion on how the end-to-end speech recognition system performs on Estonian language and whether the objectives were achieved as initially planned. The thesis is constructed around end-to-end speech recognition system, which is trained on ESPnet speech recognition toolkit using available Estonian speech recordings.

The best performing model was created using RNN with hybrid CTC/attention based encoder-decoder architecture. The model's main part is the use of both CTC and attention based encoder-decoder. This allowed the use of both methods' advantages and achieve better results. The parameters used for training the model were taken from ESPnet's example project which has a similar amount of training data.

The model was trained using 216 hours of Estonian speech audio. The audio contains mostly programmes, news and interviews from TV and radio. For better results, the data is augmented using speed manipulation, noise and reverberation.

Analysis showed some situations, where end-to-end speech recognition makes common mistakes and where it performed better than traditional approach. The system makes mistakes, when dealing with noisy audio or when the previous word ends with the same letters as the next one starts. However, it proved to be more accurate with words not existing in training set and with names.

This thesis set a goal to create an end-to-end speech recognition system that can achieve a WER below 25% and CER below 5%. Only one of those goals was met. The goal for WER was met as the system achieved a WER of 21.9% on test set. Unfortunately, the best achieved CER was 7.5% on test set. Another objective was to achieve a better result when combining two different methods. This came out positive and a combination of two systems achieved a WER of 12.0% and a CER of 4.0%. This result is statistically significant improvement from the current traditional approach for Estonian speech recognition system.

The thesis proves that end-to-end speech recognition can be used on Estonian language. However, the results are not as good as the traditional approach, it is not ruled out that the growing popularity of end-to-end speech recognition can improve over the years to achieve greater results. It also proved, that using a combination of traditional and end-to-end approach can improve the accuracy of recognising Estonian language from speech.

For future work, it is recommended to gather more data for Estonian language. The thesis did not actually have as much data as end-to-end speech recognition system needs. By only increasing training data, the system could possibly achieve more accurate results. End-to-end speech recognition is also itself a fast growing area. It is not excluded, that in a few years, end-to-end speech recognition would achieve better results than traditional approach.

7 References

[1] H. Xuedong, A. Alex and H. Hsiao-Wuen, “Spoken Language Processing: A guide to theory, algorithm, and system development,” 2001.

[2] D. Britz, “WILDML,” 17 September 2015. [Online]. Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. [Accessed 21 March 2018].

[3] C. Olah, “Colah's blog,” 27 August 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 21 March 2018].

[4] A. Graves, Supervised Sequence Labelling with Recurrent Neural Networks, Berlin: Springer-Verlag Berlin Heidelberg, 2012.

[5] S. Kostadinov, “Towards Data Science,” 2 December 2017. [Online]. Available: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7. [Accessed 13 February 2018].

[6] “A Beginner’s Guide to Recurrent Networks and LSTMs,” [Online]. Available: https://deeplearning4j.org/lstm.html. [Accessed 13 March 2018].

[7] S. Hochreiter and S. Jürgen, “Long short-term memory,” *Neural Computation,* pp. 1735-1780, September 1997.

[8] A. Hannun, “Sequence Modeling with CTC,” *Distill,* 2017.

[9] A. Graves, S. Fernandez, F. Gomez and J. Schmidhuber, *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks,* Pittsburgh, Pennsylvania: ICML '06 Proceedings of the 23rd international conference on Machine learning, 2006.

[10] D. Britz, “WILDML,” 3 January 2016. [Online]. Available: http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/. [Accessed 12 March 2018].

[11] Y. Park, S. Patwardhan, K. Visweswariah and S. C. Gates, “An Empirical Analysis of Word Error Rate and Keyword Error Rate,” in *Ninth Annual Conference of the International Speech Communication Association*, Brisbane, 2008.

[12] I. S. MacKenzie and S. R. William, “A character-level error analysis technique for evaluating text entry methods,” in *Proceedings of the second Nordic conference on Human-computer interaction*, Aarhus, 2002.

[13] D. Amodei, S. Ananthanarayanan, R. B. J. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen and J. Chen, “Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin,” in *International Conference on Machine Learning*, New York City, 2016.

[14] W. Chan, N. Jaitly, Q. V. Le and O. Vinyals, “Listen, Attend and Spell,” in *arXiv preprint arXiv:1508.01211*, 2015.

[15] C. C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, K. Gonina and N. Jaitly, "State-of-the-art speech recognition with sequence-to-sequence models," arXiv, 2017.

[16] A. Tjandra, S. Sakti and S. Nakamura, "Local Monotonic Attention Mechanism for End-to-End Speech and Language Processing," in *In Proceedings of the Eighth International Joint Conference on Natural Language Processing*, Taipei, 2017.

[17] R. Prabhavalkar, T. N. Sainath, B. Li, K. Rao and N. Jaitly, "An analysis of "attention" in sequence-to-sequence models," in *Interspeech*, Stockholm, 2017.

[18] J. Hou, Z. Shiliang and D. Lirong, "Gaussian Prediction based Attention for Online End-to-End Speech Recognition," in *Interspeech*, Stockholm, 2017.

[19] P. M. H. R. S. Doetsch and Ney, "Inverted Alignments for End-to-End Automatic Speech Recognition," *IEEE Journal of Selected Topics in Signal Processing ,* vol. 11, no. 8, pp. 1265-1273, 2017.

[20] S. Kim, T. Hori and S. Watanabe, "Joint CTC/attention decoding for end-to-end speech recognition," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 2017.

[21] S. Watanabe, "ESPnet," 2017. [Online]. Available: https://espnet.github.io/espnet/. [Accessed 3 January 2018].

[22] H. Soltau, H. Liao and H. Sak, "Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary Speech Recognition," in *Interspeech*, Stockholm, 2017.

[23] T. Zenkel, R. Sanabria, F. Metze, J. Niehues, M. Sperber, S. Stüker and A. Waibel, "Comparison of Decoding Strategies for CTC Acoustic Models," in *Interspeech*, Stockholm, 2017.

[24] O. Siohan, "CTC Training of Multi-Phone Acoustic Models for Speech Recognition," in *Interspeech*, Stockholm, 2017.

[25] R. Collobert, C. Puhrsch and G. Synnaeve, "Wav2letter: an end- to-end convnet-based speech recognition system," arXiv, 2016.

[26] Y. Zhou, C. Xiong and R. Socher, "Improving End-to-End Speech Recognition with Policy Learning," arXiv, 2017.

[27] T. Alumäe, "Recent improvents in Estonian LVCSR," in *Fourth International Workshop on Spoken Language Technologies for Under-Resourced Languages*, St. Petersburg, 2014.

[28] S. Watanabe, T. Hori, S. Kim, J. R. Hershey and T. Hayashi, "Hybrid CTC/Attention Architecture for End-to-End Speech Recognition," *IEE Journal,* vol. 11, no. 8, pp. 1240-1253, 2017.

[29] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER)," in *IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, Santa Barbara, CA, USA, 1997.

[30] P. Hitesh, "Automatic Speech Recognition," 2017. [Online]. Available: https://github.com/zzw922cn/Automatic_Speech_Recognition. [Accessed 13 November 2017].

[31] A. Hannun, C. Case and J. Casper, "Project DeepSpeech," 2017. [Online]. Available: https://github.com/mozilla/DeepSpeech. [Accessed 4 December 2017].

[32] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer and K. Vesely, “The Kaldi Speech Recognition Toolkit,” IEEE Signal Processing Society, Hilton Waikoloa Village, Big Island, Hawaii, US, 2011.

[33] D. Pallett, J. Fiscus and J. Garofolo, “Resource Management Corpus: September 1992 Test Set Benchmark Test Results,” Proceedings of ARPA Microelectronics Technology Office Continuous Speech Recognition Workshop, Stanford, CA, 1992.

[34] A. Geitgey, “How to do Speech Recognition with Deep Learning,” *Medium,* 2016.

8 Appendix 1 – Code for applying the model

```
#! /bin/bash

. cmd.sh
. path.sh

stage=1 # general configuration
backend=pytorch
stage=-1       # start from -1 if you need to start from data download
gpu=-1         # use 0 when using GPU on slurm/grid engine, otherwise -1
debugmode=1
dumpdir=dump   # directory to dump full features
N=0            # number of minibatches to be used (mainly for debugging). "0"
uses all minibatches.
verbose=0      # verbose option
resume=        # Resume the training from snapshot

# feature configuration
do_delta=false # true when using CNN

# network archtecture
# encoder related
etype=vggblstmp     # encoder architecture type
elayers=6
eunits=320
eprojs=320
subsample=1_2_2_1_1 # skip every n frame from input to nth layers

# loss related
ctctype=chainer

# decoder related
dlayers=1
dunits=300

# attention related
atype=location
adim=320
aconv_chans=10
aconv_filts=100

# hybrid CTC/attention
mtlalpha=0.5
```

```
# minibatch related
batchsize=30
maxlen_in=800  # if input length  > maxlen_in, batchsize is automatically
reduced
maxlen_out=150 # if output length > maxlen_out, batchsize is automatically
reduced

# optimization related
opt=adadelta
epochs=15

# rnnlm related
lm_weight=1.0

# decoding parameter
beam_size=20
penalty=0.0
maxlenratio=0.0
minlenratio=0.0
ctc_weight=0.3
recog_model=acc.best # set a model to be used for decoding: 'acc.best' or
'loss.best'

# exp tag
tag="1a" # tag for managing experiments.

num_data_reps=1
echo "$0 $@"  # Print the command line for logging
. parse_options.sh || exit 1;
set -e
set -u
set -o pipefail
train_set=train_trim
train_dev=dev_trim
recog_set="dev test"

if [ $stage -le 0 ]; then
  ./local/00_prepare_data.sh
fi

feat_tr_dir=${dumpdir}/${train_set}/delta${do_delta}; mkdir -p ${feat_tr_dir}
feat_dt_dir=${dumpdir}/${train_dev}/delta${do_delta}; mkdir -p ${feat_dt_dir}

if [ ${stage} -le 1 ]; then
    echo "stage 1: Feature Generation"
    # Generate the fbank features; by default 80-dimensional fbanks with
    # pitch on each frame
    for x in test dev train; do
        steps/make_fbank_pitch.sh --cmd "$train_cmd" --nj 32 data/${x} || \
        exit 1;
    done
```

```
    remove_longshortdata.sh --maxchars 400 data/train data/${train_set}
    remove_longshortdata.sh --maxchars 400 data/dev data/${train_dev}

    # compute global CMVN
    compute-cmvn-stats scp:data/${train_set}/feats.scp \
    data/${train_set}/cmvn.ark || exit 1;
    dump.sh --cmd "$train_cmd" --nj 32 --do_delta $do_delta \
    data/${train_set}/feats.scp data/${train_set}/cmvn.ark \
    exp/dump_feats/train ${feat_tr_dir}
    dump.sh --cmd "$train_cmd" --nj 32 --do_delta $do_delta \
    data/${train_dev}/feats.scp data/${train_set}/cmvn.ark exp/dump_feats/\
    dev ${feat_dt_dir}
fi

dict=data/lang_1char/${train_set}_units.txt
echo "dictionary: ${dict}"

if [ ${stage} -le 2 ]; then
    ### Task dependent. You have to check non-linguistic symbols used in the
    # corpus.
    echo "stage 2: Dictionary and Json Data Preparation"
    mkdir -p data/lang_1char/
    echo "<unk> 1" > ${dict} # <unk> must be 1, 0 will be used for "blank" \
    in CTC
    echo "<space> @ a b c d e f g h i j k l m n o p q r s t u v w x y z ü õ \
    ö ä A B C D E F G H I J K L M N O P Q R S T U V W X Y Z š ž Š Ž Ü Õ Ö Ä \
    - '" | tr " " "\n" \
    | sort | uniq | grep -v -e '^\s*$' | awk '{print $0 " " NR+1}' \
    >> ${dict} wc -l ${dict}

    # make json labels
    data2json.sh --feat ${feat_tr_dir}/feats.scp \
    data/${train_set} ${dict} > ${feat_tr_dir}/data.json
    data2json.sh --feat ${feat_dt_dir}/feats.scp \
    data/${train_dev} ${dict} > ${feat_dt_dir}/data.json
fi

if [ -z ${tag} ]; then
    expdir=exp/${train_set}_${etype}_e${elayers}_subsample${subsample}\
    _unit${eunits}_proj${eprojs}_ctc${ctctype}_d${dlayers}_unit${dunits}\
    _${atype}_adim${adim}_aconvc${aconv_chans}_aconvf${aconv_filts}\
    _mtlalpha${mtlalpha}_${opt}_bs${batchsize}_mli${maxlen_in}\
    _mlo${maxlen_out}

    if ${do_delta}; then
        expdir=${expdir}_delta
    fi
else
    expdir=exp/${train_set}_${tag}
fi
```

```
mkdir -p ${expdir}

if [ ${stage} -le 4 ]; then
    echo "stage 3: Network Training"
    ${cuda_cmd} ${expdir}/train.log \
    asr_train.py \
    --gpu ${gpu} \
    --backend ${backend} \
    --outdir ${expdir}/results \
    --debugmode ${debugmode} \
    --dict ${dict} \
    --debugdir ${expdir} \
    --minibatches ${N} \
    --verbose ${verbose} \
    --resume ${resume} \
    --train-feat scp:${feat_tr_dir}/feats.scp \
    --valid-feat scp:${feat_dt_dir}/feats.scp \
    --train-label ${feat_tr_dir}/data.json \
    --valid-label ${feat_dt_dir}/data.json \
    --etype ${etype} \
    --elayers ${elayers} \
    --eunits ${eunits} \
    --eprojs ${eprojs} \
    --subsample ${subsample} \
    --ctc_type ${ctctype} \
    --dlayers ${dlayers} \
    --dunits ${dunits} \
    --atype ${atype} \
    --adim ${adim} \
    --aconv-chans ${aconv_chans} \
    --aconv-filts ${aconv_filts} \
    --mtlalpha ${mtlalpha} \
    --batch-size ${batchsize} \
    --maxlen-in ${maxlen_in} \
    --maxlen-out ${maxlen_out} \
    --opt ${opt} \
    --epochs ${epochs}
Fi

if [ ${stage} -le 5 ]; then
    echo "stage 5: Decoding"
    nj=4
    for rtask in ${recog_set}; do
    (
        decode_dir=decode_${rtask}_beam${beam_size}_e${recog_model}_p\
        ${penalty}_len${minlenratio}-${maxlenratio}_ctcw${ctc_weight}

        # split data
        data=data/${rtask}
        split_data.sh --per-utt ${data} ${nj};
        sdata=${data}/split${nj}utt;
```

```
        # feature extraction
        feats="ark,s,cs:apply-cmvn \
        --norm-vars=truedata/${train_set}/cmvn.ark \
        scp:${sdata}/JOB/feats.scp ark:- |"

        if ${do_delta}; then
            feats="$feats add-deltas ark:- ark:- |"
        fi

        # make json labels for recognition
        data2json.sh ${data} ${dict} > ${data}/data.json

        #### use CPU for decoding
        gpu=-1

        ${decode_cmd} JOB=1:${nj} ${expdir}/${decode_dir}/log/decode.JOB.log\
            asr_recog.py \
            --gpu ${gpu} \
            --backend ${backend} \
            --debugmode ${debugmode} \
            --verbose ${verbose} \
            --recog-feat "$feats" \
            --recog-label ${data}/data.json \
            --result-label ${expdir}/${decode_dir}/data.JOB.json \
            --model ${expdir}/results/model.${recog_model}  \
            --model-conf ${expdir}/results/model.conf  \
            --beam-size ${beam_size} \
            --penalty ${penalty} \
            --maxlenratio ${maxlenratio} \
            --minlenratio ${minlenratio} \
            --ctc-weight ${ctc_weight} \
            --nbest 100 \
            --lm-weight ${lm_weight} &
        wait
        score_sclite.sh --wer true ${expdir}/${decode_dir} ${dict}
    ) &
    done
    wait
    echo "Finished"
fi

if [ ${stage} -le 6 ]; then
    echo "stage 5: Speech speed-augmentation"
    utils/data/perturb_data_dir_speed_3way.sh data/train data/train_sp
fi
```

```
if [ ${stage} -le 7 ]; then
    echo "stage 1: Feature Generation for speed-augmented data"
    # Generate the fbank features; by default 80-dimensional fbanks with
    #pitch on each frame
    remove_longshortdata.sh --maxchars 400 data/train_sp data/${train_set}_sp
    dump.sh --cmd "$train_cmd" --nj 8 --do_delta $do_delta \
    data/${train_set}_sp/feats.scp data/${train_set}/cmvn.ark \
    exp/dump_feats/train_sp ${feat_tr_dir}_sp
fi

if [ ${stage} -le 8 ]; then
    # make json labels
    data2json.sh --feat ${feat_tr_dir}_sp/feats.scp \
    data/${train_set}_sp ${dict} > ${feat_tr_dir}_sp/data.json
fi

if [ -z ${tag} ]; then
    expdir=exp/${train_set}_${etype}_e${elayers}_subsample${subsample}\
    _unit${eunits}_proj${eprojs}_ctc${ctctype}_d${dlayers}_unit${dunits}\
    _${atype}_adim${adim}_aconvc${aconv_chans}_aconvf${aconv_filts}\
    _mtlalpha${mtlalpha}_${opt}_bs${batchsize}_mli${maxlen_in}\
    _mlo${maxlen_out}
    if ${do_delta}; then
        expdir=${expdir}_delta
    fi
else
    expdir=exp/${train_set}_sp_${tag}
fi

mkdir -p ${expdir}

if [ ${stage} -le 9 ]; then
    echo "stage 3: Network Training"
    ${cuda_cmd} ${expdir}/train.log \
    asr_train.py \
    --gpu ${gpu} \
    --backend ${backend} \
    --outdir ${expdir}/results \
    --debugmode ${debugmode} \
    --dict ${dict} \
    --debugdir ${expdir} \
    --minibatches ${N} \
    --verbose ${verbose} \
    --resume ${resume} \
    --train-feat scp:${feat_tr_dir}_sp/feats.scp \
    --valid-feat scp:${feat_dt_dir}/feats.scp \
    --train-label ${feat_tr_dir}_sp/data.json \
    --valid-label ${feat_dt_dir}/data.json \
    --etype ${etype} \
    --elayers ${elayers} \
    --eunits ${eunits} \
```

```
    --eprojs ${eprojs} \
    --subsample ${subsample} \
    --ctc_type ${ctctype} \
    --dlayers ${dlayers} \
    --dunits ${dunits} \
    --atype ${atype} \
    --adim ${adim} \
    --aconv-chans ${aconv_chans} \
    --aconv-filts ${aconv_filts} \
    --mtlalpha ${mtlalpha} \
    --batch-size ${batchsize} \
    --maxlen-in ${maxlen_in} \
    --maxlen-out ${maxlen_out} \
    --opt ${opt} \
    --epochs $[${epochs}/2]
fi

if [ ${stage} -le 10 ]; then
    echo "stage 10: Decoding using speed-perturbed model"
    nj=4

    for rtask in ${recog_set}; do
    (
        decode_dir=decode_${rtask}_beam${beam_size}_e${recog_model}\
        _p${penalty}_len${minlenratio}-${maxlenratio}_ctcw${ctc_weight}

        # split data
        data=data/${rtask}
        split_data.sh --per-utt ${data} ${nj};
        sdata=${data}/split${nj}utt;

        # feature extraction
        feats="ark,s,cs:apply-cmvn --norm-vars=true\
        data/${train_set}/cmvn.ark scp:${sdata}/JOB/feats.scp ark:- |"

        if ${do_delta}; then
            feats="$feats add-deltas ark:- ark:- |"
        fi

        # make json labels for recognition
        data2json.sh ${data} ${dict} > ${data}/data.json

        #### use CPU for decoding
        gpu=-1

        ${decode_cmd} JOB=1:${nj} ${expdir}/${decode_dir}/log/decode.JOB.log\
        asr_recog.py \
        --gpu ${gpu} \
        --backend ${backend} \
        --debugmode ${debugmode} \
        --verbose ${verbose} \
```

```
        --recog-feat "$feats" \
        --recog-label ${data}/data.json \
        --result-label ${expdir}/${decode_dir}/data.JOB.json \
        --model ${expdir}/results/model.${recog_model}  \
        --model-conf ${expdir}/results/model.conf  \
        --beam-size ${beam_size} \
        --penalty ${penalty} \
        --maxlenratio ${maxlenratio} \
        --minlenratio ${minlenratio} \
        --ctc-weight ${ctc_weight} \
        --nbest 100 \
        --lm-weight ${lm_weight} &
        wait

        score_sclite.sh --wer true ${expdir}/${decode_dir} ${dict}

    ) &
    done
    wait
    echo "Finished"
fi


if [ ${stage} -le 11 ]; then
    echo "stage 5: Doing noise and reverberation augmentation"

    if [ ! -d "RIRS_NOISES" ]; then
        # Download the package that includes the real RIRs, simulated RIRs,
        #isotropic noises and point-source noises
        wget --no-check-certificate\
        http://www.openslr.org/resources/28/rirs_noises.zip
        unzip rirs_noises.zip
    fi

    rvb_opts=()
    rvb_opts+=(--rir-set-parameters "0.5,\
    RIRS_NOISES/simulated_rirs/smallroom/rir_list")
    rvb_opts+=(--rir-set-parameters "0.5,\
    RIRS_NOISES/simulated_rirs/mediumroom/rir_list")
    rvb_opts+=(--noise-set-parameters\
    RIRS_NOISES/pointsource_noises/noise_list)

    python steps/data/reverberate_data_dir.py \
    "${rvb_opts[@]}" \
    --prefix "rev" \
    --foreground-snrs "20:10:15:5:0" \
    --background-snrs "20:10:15:5:0" \
    --speech-rvb-probability 1 \
    --pointsource-noise-addition-probability 1 \
    --isotropic-noise-addition-probability 1 \
    --num-replications ${num_data_reps} \
```

```
    --max-noises-per-minute 1 \
    --source-sampling-rate 16000 \
    --include-original-data true \
    data/train_sp data/train_sp_rvb${num_data_reps}_tmp || exit 1;

    local/persist_wav_data_dir.sh --cmd "$train_cmd" --nj 8 \
    data/train_sp_rvb${num_data_reps}_tmp \
    data/train_sp_rvb${num_data_reps} \
    data/train_sp_rvb${num_data_reps}/data || exit 1;
fi

if [ ${stage} -le 12 ]; then
    echo "stage 12: Feature Generation for noise-augmented data"

    #Generate the fbank features; by default 80-dimensional fbanks with pitch
    #on each frame
    steps/make_fbank_pitch.sh --cmd "$train_cmd" --nj 8\
    data/train_sp_rvb${num_data_reps}

    remove_longshortdata.sh --maxchars 400 data/train_sp_rvb${num_data_reps}\
    data/${train_set}_sp_rvb${num_data_reps}

    # compute global CMVN
    compute-cmvn-stats \
    scp:data/${train_set}_sp_rvb${num_data_reps}/feats.scp \
    data/${train_set}_sp_rvb${num_data_reps}/cmvn.ark || exit 1;

    dump.sh --cmd "$train_cmd" --nj 8 --do_delta $do_delta \
    data/${train_set}_sp_rvb${num_data_reps}/feats.scp \
    data/${train_set}_sp_rvb${num_data_reps}/cmvn.ark\
    exp/dump_feats/train_sp_rvb${num_data_reps}\
    ${feat_tr_dir}_sp_rvb${num_data_reps}
fi


if [ ${stage} -le 13 ]; then
    # make json labels
    data2json.sh --feat ${feat_tr_dir}_sp_rvb${num_data_reps}/feats.scp \
    data/${train_set}_sp_rvb${num_data_reps} ${dict} > \
    ${feat_tr_dir}_sp_rvb${num_data_reps}/data.json
fi

if [ -z ${tag} ]; then
    expdir=exp/${train_set}_${etype}_e${elayers}_subsample${subsample}\
    _unit${eunits}_proj${eprojs}_ctc${ctctype}_d${dlayers}_unit${dunits}\
    _${atype}_adim${adim}_aconvc${aconv_chans}_aconvf${aconv_filts}\
    _mtlalpha${mtlalpha}_${opt}_bs${batchsize}_mli${maxlen_in}\
    _mlo${maxlen_out}

    if ${do_delta}; then
        expdir=${expdir}_delta
```

```
    fi
else
    expdir=exp/${train_set}_sp_rvb${num_data_reps}_${tag}
fi
mkdir -p ${expdir}

if [ ${stage} -le 14 ]; then
    echo "stage 14: Network Training"
    ${cuda_cmd} ${expdir}/train.log \
    asr_train.py \
    --gpu ${gpu} \
    --backend ${backend} \
    --outdir ${expdir}/results \
    --debugmode ${debugmode} \
    --dict ${dict} \
    --debugdir ${expdir} \
    --minibatches ${N} \
    --verbose ${verbose} \
    --resume ${resume} \
    --train-feat scp:${feat_tr_dir}_sp_rvb${num_data_reps}/feats.scp \
    --valid-feat scp:${feat_dt_dir}/feats.scp \
    --train-label ${feat_tr_dir}_sp_rvb${num_data_reps}/data.json \
    --valid-label ${feat_dt_dir}/data.json \
    --etype ${etype} \
    --elayers ${elayers} \
    --eunits ${eunits} \
    --eprojs ${eprojs} \
    --subsample ${subsample} \
    --ctc_type ${ctctype} \
    --dlayers ${dlayers} \
    --dunits ${dunits} \
    --atype ${atype} \
    --adim ${adim} \
    --aconv-chans ${aconv_chans} \
    --aconv-filts ${aconv_filts} \
    --mtlalpha ${mtlalpha} \
    --batch-size ${batchsize} \
    --maxlen-in ${maxlen_in} \
    --maxlen-out ${maxlen_out} \
    --opt ${opt} \
    --epochs $[${epochs}/3]
fi

if [ ${stage} -le 15 ]; then
    echo "stage 15: Decoding using noise and speed-perturbed model"
    nj=4

    for rtask in ${recog_set}; do
    (
        decode_dir=decode_${rtask}_beam${beam_size}_e${recog_model}\
        _p${penalty}_len${minlenratio}-${maxlenratio}_ctcw${ctc_weight}
```

```
        # split data
        data=data/${rtask}
        split_data.sh --per-utt ${data} ${nj};
        sdata=${data}/split${nj}utt;

        # feature extraction
        feats="ark,s,cs:apply-cmvn --norm-vars=true\
        data/${train_set}_sp_rvb${num_data_reps}/cmvn.ark\
        scp:${sdata}/JOB/feats.scp ark:- |"

        if ${do_delta}; then
            feats="$feats add-deltas ark:- ark:- |"
        fi

        # make json labels for recognition
        data2json.sh ${data} ${dict} > ${data}/data.json

        #### use CPU for decoding
        gpu=-1

        ${decode_cmd} JOB=1:${nj} ${expdir}/${decode_dir}/log/decode.JOB.log\
        asr_recog.py \
        --gpu ${gpu} \
        --backend ${backend} \
        --debugmode ${debugmode} \
        --verbose ${verbose} \
        --recog-feat "$feats" \
        --recog-label ${data}/data.json \
        --result-label ${expdir}/${decode_dir}/data.JOB.json \
        --model ${expdir}/results/model.${recog_model}  \
        --model-conf ${expdir}/results/model.conf  \
        --beam-size ${beam_size} \
        --penalty ${penalty} \
        --maxlenratio ${maxlenratio} \
        --minlenratio ${minlenratio} \
        --ctc-weight ${ctc_weight} \
        --nbest 100 \
        --lm-weight ${lm_weight} &
        wait

        score_sclite.sh --wer true ${expdir}/${decode_dir} ${dict}

    ) &
    done
    wait
    echo "Finished"
fi
```