CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Speech-to-speech translation using deep learning

FREDRIK BREDMAR

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2016:24

Speech-to-speech translation using deep learning

Fredrik Bredmar

Department of Computer Science and Engineering
Division of Computing Science
University of Gothenburg
Gothenburg, Sweden 2017

Speech-to-speech translation using deep learning
FREDRIK BREDMAR

Supervisor: Mikael Kågebäck, Computer Science and Engineering
Examiner: Peter Damaschke, Computer Science and Engineering

Master's Thesis 2016:24
Department of Computer Science and Engineering
Division of Computing Science
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Speech-to-speech translation using deep learning
Fredrik Bredmar
Department of Computer Science and Engineering
Gothenburg University

Abstract

Current state-of-the-art translation systems for speech-to-speech rely heavily on a text representation for the translation. By transcoding speech to text we lose important information about the characteristics of the voice such as the emotion, pitch and accent. This thesis examine the possibility of using an LSTM neural network model to translate speech-to-speech without the need of a text representation. That is by translating using the raw audio data directly in order to persevere the characteristics of the voice that otherwise get lost in the text transcoding part of the translation process. As part of this research we create a data set of phrases suitable for speech-to-speech translation tasks. The thesis result in a proof of concept system which need to scale the underlying deep neural network in order to work better.

Keywords: Neural Networks, Deep Learning, LSTM, RNN, Speech-to-speech translation

Acknowledgements

First of all I would like to express my sincere gratitude to my supervisor Mikael Kågebäck for the support and guidance throughout this thesis. I am also thankful to Fredrik Johansson for his interest and advice.

Fredrik Bredmar, Gothenburg, October 2016

Contents

1
Introduction

This thesis takes on the problem of speech-to-speech translation with persevered voice characteristics found in the source voice throughout the translation. Based on recent advancements within the scientific computing area combined with the cutting edge technology of deep neural networks we believe that a translation system based solely on the human voice is able to take machine translation to the next level. Today state-of-the-art translation systems rely heavily on the text representation in order to translate. These systems can be split up into three distinct steps. The first step is the speech-to-text part. The core component of this step is usually referred to as the speech recognition component [34]. The following step is the actual translation step. The translation is done within the text domain. The final step is the text-to-speech step. The core component of the final step is the speech synthesizer [5]. With the advancements of neural networks and deep learning each of these components have independently become better the last few years [17, 41, 13].
Previous research show that deep recurrent neural networks have an impressive effectiveness on previously difficult tasks where dependency over time between consecutive examples is important. This thesis aim to push the limits of these deep recurrent neural networks by attempting to translate voice-to-voice without a text representation. This section introduce the reader to the problem of speech-to-speech translations and why a system like this could take speech translation to the next level.

1.1 Motivation

Machine translation has been an ongoing research for a long time. With the globalisation happening the need for good machine translation tools is evident. The current state of machine translation is rather good when it comes to text-to-text translation. However the problem of speech-to-speech translation is yet to get a satisfiable solution. In todays state-of-the-art systems we see a workflow with three separate systems. A speech recognition system that identifies the words spoken and transcode them into text. These systems are robust and keep on evolving. Speech recognition systems make it possible to use the text-to-text translation models as mentioned previously. This yield a text translation of the source speech. With this text translation we can now use a speech syntheziser to go from the translated text to spoken words. These systems are all rather robust and work well for their intended purpose. However for use in a daily conversation the voice of the syntheziser get rather dull. The reason for this is that when we transcode from the speech

signal into a text representation in the speech recognition system we lose important characteristics of the voice that is needed to get a good dynamic voice.
With recent advancements in the area of neural networks and deep learning we are able to solve problems as generating handwriting, translate text-to-text better than before, generate program source code, image captioning and many more interesting problems [24]. An important architecture that is the common ground for many of these neural networks is the Long Short-Term Memory (LSTM) model. By using the benefits of the LSTM network this thesis aim to accomplish a translation system from speech-to-speech without the need of a text representation in order to persevere voice characteristics from the source voice through the translation. This will be done using a sequence to sequence model to represent the source voice and the target voice. The contributions of this thesis aim to show a proof of concept system for translating speech-to-speech. Which will allow for a new type of translation systems which, while persevering translation correctness, adds a more dynamic output voice in terms of mood, gender, pitch, and accent.
A crucial point in complex machine learning systems is the availability of good, large data sets to be suitable for training and testing of developed systems. During our research we have found a large gap in data sets suitable for speech-to-speech translation. Therefore as part of this thesis we aim to contribute a data set suitable for speech-to-speech translation tasks.

1.2 Problem definition

By applying current state-of-the-art algorithms within the deep learning area this project will investigate the performance of a speech-to-speech translation system that persevere voice characteristics throughout the translation. The long short-term memory (LSTM) network will be the core of the system. LSTM networks has been explored within the area of sequence to sequence learning before. Sutskever et al. has used LSTM networks to translate text-to-text with greater results than previous systems [41]. Within a similar area LSTM networks has been used to generate handwriting [18]. In the handwriting generation the generated text kept distinct characteristics of the original text. The data representation for our project is an important difference from previous papers. Text data is rather simple in its representation. Speech audio on the other hand provide more information. Therefore an important part of the project is to investigate possible data representations suitable for the LSTM network. However the richer feature space is the foundation of our belief that speech-to-speech translation can yield better result with higher dynamics than previous systems. The main aim of the project is to persevere the voice characteristics through a translation rather than trying to get a better BLEU [32] score than previous systems.

1.3 Research goals

This thesis project aim to result in two research contributions. The first has previously been described, which is the speech-to-speech translation system. The aim of

this system can be summarized into two distinct points as follows.

- Improved translations by using information embedded in the voice-print that is destroyed when transcoding in text.
- Develop a speech-to-speech system that captures the emotion and characteristics of the input.

The other contribution we aim to achieve is to create a data set for speech-to-speech translation tasks. This data set will be a phrase data set containing phrases between English and French. The data set specifically need to have the following four features.

- large set of phrases where each phrase is available in both English and French
- phrases of variable length
- a natural flow in the speech, as in day-to-day conversation
- a great variety of people talking with different voice characteristics

1.4 Outline

The thesis start with showing previous work in chapter 2 that relates to the aim of this thesis . Both in the context of current state-of-the-art systems for speech-to-speech translation and in the context of previous research based on the type of machine learning methods we want to use as our foundation for this thesis. Then in chapter 3 we introduce the reader to the concept of neural networks in general. Further in section 3.3 we introduce the recurrent neural networks that is the foundation of the work done in this project. Finally we give an in-depth description of the Long short-term memory neural network in section 3.4. The Long short-term memory neural network is the recurrent neural network architecture currently most well-suited for sequence learning where the dependencies may be over long time periods and the mapping between dependencies can be non-linear.
In chapter 3.9 we continue to step by step describe the data processing made to transform our raw movie files into several phrase files in a data representation suitable for sequence to sequence learning in a LSTM neural network. Chapter 4 describe how we implemented our system, in terms of building the network and processing the sequences resulting from the data processing step. We then describe our experiments in chapter 6 and how we evaluated the results. Finally chapter 8 show the results and discuss the results of this thesis and the design choices made. Both in terms of data representation and the system design. With chapter 9 we conclude what we have accomplished within this thesis and look at possible extensions and future work possible in this project.

2
Related work

In this section we go through related work to our problem. The section is split into two parts. The first part focus on actual state-of-the-art translation systems that are being used today as well as research connected to these systems. The second part focus more on related work toward sequence modeling experiments with neural networks, especially the LSTM model to show what previously has been accomplished.

2.1 Systems

In this section we introduce software systems that are capable of translating speech to speech, or would be capable with small extensions or modifications.

Google translate
The translation system from Google mainly work on text, but do have built-in functionality to take the input from a microphone and then output the translated audio through the speakers. As previously described Google translate use the classical speech-to-speech translation style with the use of a speech recognizer for speech to text, then translating the text and finally a speech synthesizer system for generating the audio related to the text. It is worth to mention that the Google translate service is a tough candidate to beat in terms of correct translations due to its well engineered implementation with enormous quantities of input data from many different sources.

Skype translator
The VoIP service from Skype [2] has currently an open beta where it is possible to make near real-time speech translation during calls with their service using deep learning [1]. The foundation of their translation feature, called Skype Translator [15], is built on the same setting as Google's translation service with one addition [2, 30, 33, 27, 38]. Skype also use an acoustic model for mapping the source voice to a translated voice similar to the source voice. At the point of writing this report the acoustic model more or less use two translator voices, one male and one female, and try to map the source voice to the closest acoustic model. Which in theory will yield a more dynamic translated voice in terms of voice characteristics. However the current result is rather bulky and sounds like a robot with small fragments of human elements [29]. Google's translation service provide a more human like speech output although they do not use an acoustic model for their output.

Jibbigo
A speech-to-speech translation application for mobile devices that allows the user to speak a sentence and then the app translates the sentence and output the translation to the user through the speakers [14]. Jibbigo also has support for a two way dialog between participants. This method is however fully dependent on a text representation.

Moses
Although Moses is not an end-system for the speech-to-speech translation model it is worth mentioning for the purpose of potential benchmarking for speech-to-speech models. Moses is a statistical machine translation engine that allows for training statistical models for the purpose of text translations [25]. Since Moses has been shown to yield good results for text translations would be an interesting component of a base system to compare our results to and add components for speech recognition and speech synthesis in order to create a model similar to Google translate and Skype translator but with a know data source.

2.2 Algorithms

In this section we introduce previous research done based on the algorithms and methods used in this thesis. The problems solved in these papers are similar in nature of our problem as in they all solve a sequence-to-sequence problem. The main difference is the problem domain.

Sequence learning
In the paper *Sequence to sequence learning with neural networks* Sutskever et al. introduce the concept of sequence to sequence modeling with neural networks [41]. They more specifically use LSTM networks in order to make text-to-text translations between French and English. With the model introduced in the paper they are able to outperform state-of-the-art systems for machine text translations in terms of BLEU score [32]. An important contribution made by this paper was the discovery that reversing the input sequence gave a distinct increase in the resulting translation. This is due to the fact that the start of the input sequence and the start of the output sequence get closer in the time steps which helps the network in starting the translation. Another strenght shown in the paper by Sutskever et al. is the ability of the network to learn word ordering between the two languages.
The paper by Sutskever et al. is based on a paper by Alex Graves called *Generating sequences with recurrent neural networks* [18] where Graves show the strength of LSTM networks for the task of generating sequences. The paper shows that given a complex structure in the input the LSTM network is possible to persevere this structure when predicting future time steps. One of the experiments made by Graves is that of handwriting generation. In this experiment he shows that given a input sequence of a handwriting sample the network is able to identify characteristics of the input sequence and persevere while generating words learned from several different handwriting inputs. The result of this experiment is closely related to how we believe that the speech-to-speech translation system will work where in our case the

characteristics persevered is those that identify the voice and the actual translation between phrases can be learned through large sets of phrases independent of the specific voice characteristics.

3
Background

In this section we introduce the theory and the recent advancements within the machine learning area that make a speech to speech translation system without a text representation possible. The subsections 3.4, 3.5, and 3.6 aim more toward explaining techniques and technologies used to yield a better system. Finally we describe the process of creating the data set and the techniques used.

3.1 Neural networks

This project's basis is solely dependent on the subfield within machine learning called artificial neural networks, generally referred to as neural networks. In 1989 Dr. Robert Hecht-Neilsen defined neural networks as "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" [11]. Figure 3.2 show an example of a simple neural network. It will be referred to in this section in order to get an overview of the general structure of a neural network. There are three types of layers in a neural network as seen in Figure 3.2. The input layer is the nodes which we feed our input data into, as described in the *data representation* section. The output layer is the output generated by the network based on the given input in the input layer. In our project the output size of the output layer should be equal to the input layer size, that is number of nodes in that layer, since we want to predict new frequencies. The middle layer is called the hidden layer. Simplified the size and the number of hidden layers determine how much information the network can distinguish between. This is the basic structure of the neural network. Now we will describe how the network can actually learn.
To each layer belongs a set of nodes. From each node in a layer is an edge connecting it to each of the nodes in the following layer, just as shown in Figure 3.2. Each node does a predefined computation based on the input from the edges. One classic, yet simple, type of node in the neural network is the McCulloch-Pitts node [28]. An illustration of this node, or neuron as McCulloch and Pitt prefer to call them, can be seen in Figure 3.1. The computations done in the McCulloch-Pitts node is basically a sigmoid function. We sum up the inputs and if they are above a certain threshold, we output 1 otherwise we output 0. As we will see later on there are more complex representations of these nodes, but the McCulloch-Pitts neuron is a good starting point for understanding the basics of neural networks.
In combination with these nodes are the edges illustrated as arrows in Figure 3.2. These edges are more commonly referred to as the weights of the network. What

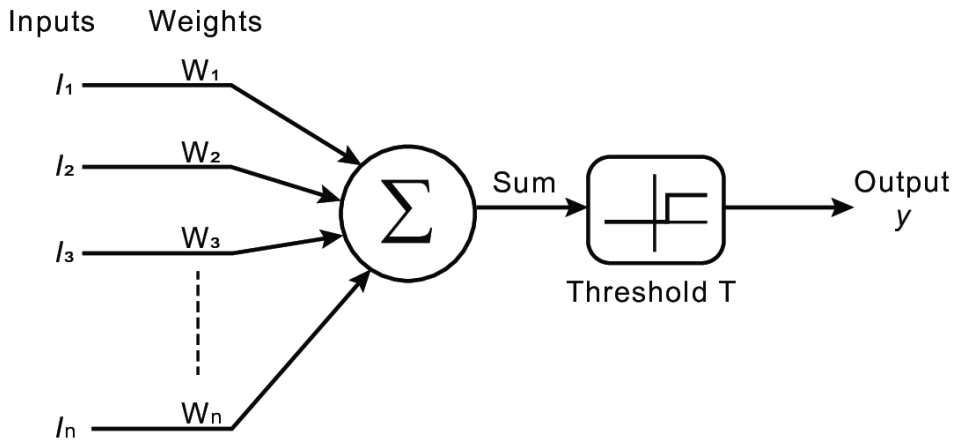

Figure 3.1: The McCulloch-Pitts neuron, a classic neural network node.

they basically do is to multiply the output of a neuron with it's value and then feed it into the next node that the edge is connected to. By updating these weights, depending on the generated output of an example, we can teach the network to distinguish which input data should generate which output data. The procedure of updating the weights is called *backpropagation* which we will describe in the next section.

3.1.1 Backpropagation

The backpropagation part of a neural network's training is where it actually learns. This is where the network update its weights throughout the network in order to make the correct output given an input. The inner workings of the backpropagation algorithm is explained by Rumelhart et al. [35] we will give an overview of the concept in this section.

The backpropagation starts from the output where we compare each output node to the expected output. The difference between our output and the expected output can be computed in many ways, one classic way is by the mean squared error. The function used to compute the error is called our *loss function*. The loss function must be differentiable to work with the backpropagation algorithm. We now want to update all weights in the network to make a closer output to the expected output next time the input is give to the network. We start off by computing the partial derivate of the loss function with respect to the incoming edges to the output node. Each derivative express how much the loss function output depends on each input weight. The weights are now updated and the error of the activation functions in the nodes of the previous layer can be updated in the same way. This update is recursively computed all the way to the input layer in order to update the whole network.

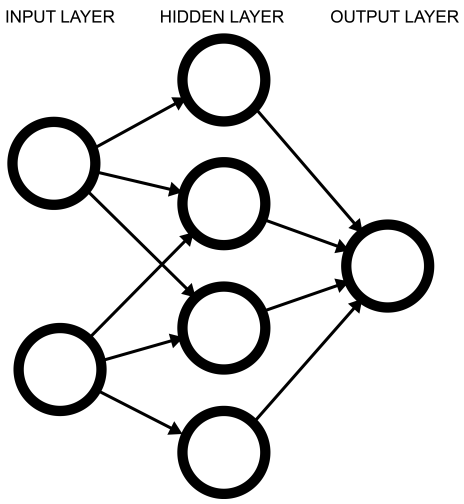

Figure 3.2: An example structure of a simple feedforward neural network.

3.2 Deep Learning

With the advancement of computational power the last $10 - 15$ years a new area in machine learning has evolved. This area is called deep learning. Deep learning as a concept can cover many algorithms within machine learning. However deep learning within neural networks is seen as very large models, usually with very many hidden layers between the output and input layer and each layer can consist of a large set of nodes. The main reason for the progression of deep learning is the introduction of general-purpose computing on graphical processing units (GPUs) [16]. The two main advantages of GPUs computations to CPUs in terms of neural networks is the fast access to large quantities of data to the processing unit so the overhead of moving data is reduced. The other reason is the optimization of the type of computations the GPU handles. GPUs have well optimized functions for computing matrices. Since neural networks rely heavily on matrix operations for its training the GPU enables much faster processing [26].

3.3 Recurrent neural networks

As the deep learning segment evolved previously untested theoretical algorithms became available for testing and usage to solve complex problems. One such algorithm is the recurrent neural networks (RNNs), Jürgen Schmidhuber were an early adpoter of RNNs and lead a research team to many great discoveries within RNNs [36]. The major advancement by the introduction of recurrent neural network is the ability to allow connections between nodes in a further layer to previous layers. That is it allowed for neural networks to be expressed as directed cycles. These neural networks now allow the trained input to be dependent over cycles. Which allowed recurrent neural networks to model the input and output data as sequences, text sentences for example, and model a dependence through these sequences compared to previous neural network algorithms where the input needed to be stationary. Figure 3.3 gives an example of a recurrent neural network architecture.

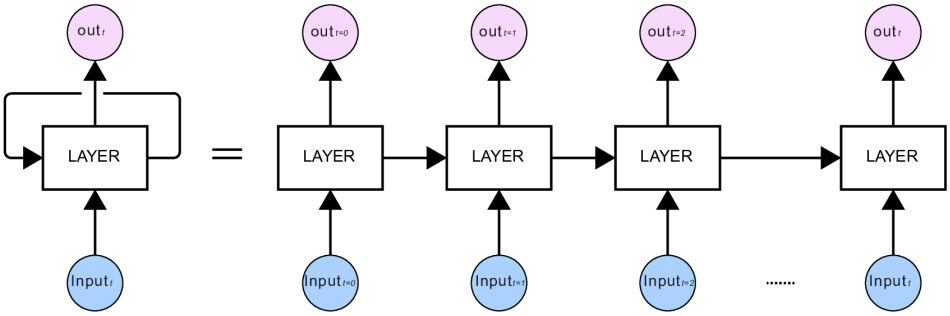

Figure 3.3: Example structure of a recurrent neural network.

3.4 Long short-term memory neural networks

Finally we arrive at the architecture laying the foundation upon which we build our thesis on. With the introduction of recurrent neural networks we could start to model our data as sequences dependent on previous steps over time. One drawback of the recurrent neural network previously described is the long term memory. Recurrent neural networks work well for dependencies within close time steps. However although recurrent neural networks are able to handle long-term dependencies in theory, it has a hard time to accomplish these long-term dependencies in reality [20, 9]. This is the idea and motivation behind the Long short-term memory (LSTM) neural network [21]. By redefining how the nodes keep past information the LSTM networks are able to persevere long-term dependencies much better. This makes handling long-term dependencies natural for LSTM network, rather than something they struggle to learn.

As mentioned previously the LSTM redefine the structure of the node for keeping long-term dependencies. Regular RNNs can use a very simple node structure, like a single non-linear activation function such as the tanh function. The structure of the LSTM network is much more complex. An illustration of the LSTM node can be seen in Figure 3.4. The LSTM node consist of four interacting layers. The core part of the LSTM node is the straight line going through the node as seen in Figure 3.4, this line illustrates the cell state. The cell state is modeled as a matrix with its size being determined by the number of neurons set for the LSTM node. It is within the cell state that we can add or remove information to be saved to the next time step.

We will now walk through each layer of the LSTM node and describe how it affects the cell state. There is a total of four layers in the LSTM node. Just like the cell state each of the layers is modeled as a matrix with its size determined by the number of neurons for an LSTM node. Each layer have the same input, namely the output of the previous LSTM node and the input at the current time step. The first layer is the sigmoid layer seen as the left most yellow box in Figure 3.4. This layer is referred to as the forget layer. This layer does is to update the cell state by removing previously known information that we now want to forget about. One way to visualise this is to think of the context of a room setting. As long as the

same people are in the room, we want to be able to refer to each person. However if one person leaves we want to remove this person from the context of the room, so we do not refer to a non-present person. This is what the forget layer handles. The formulas for computing the forget layer looks as follows.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Where σ is the sigmoid function, W_f is the weight matrix of the forget layer, $\cdot$ is the dot product, h_{t-1} is the output generated from the previous LSTM node, x_t is the input at time t to the network and b_f is the bias term for the forget layer.
The next two layers decide what information to add into the state. The first layer is a sigmoid layer, this layer decides which parts of the state we want to update. The other layer is the tanh layer. The tanh layer generates a new cell state that we want to add to the current cell state. The way these two layers cooperate is that the tanh layer generates a new cell state candidate. This candidate is then passed along to the sigmoid layer that acts as a gatekeeper which decides what parts of this new cell state that should be added to the current cell state. This procedure boils down to a way to insert new context into the cell state. To keep the analogy of the context being persons in a room, this mechanism allows us to add a new person in the room. The current state of the cell state will be passed along to the next time step as the input state. The current cell state will also be used to produce the output of the current time step. Below is the cell state update process expressed in formulas.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

In the above formulas i_t represent the update layer and $\tilde{C}_t$ is the cell state update candidate. The $\odot$ representation means pointwise multiplication. W_i and W_C is the weight matrix for the update layer and the cell state candidate respectively. b_i and b_C is the bias term which follows the same naming convention as the weight matrices. C_t is the new cell state, first multiplied with the matrix named the forget layer and then added with the matrix named the update layer.
The final layer is the output layer. In this layer we decide what we want to output. The output will be based on the updated cell state, but will be filtered by a sigmoid layer. The cell state is put through a tanh function, just to push the cell values within -1 and 1 and multiplied with the matrix named the sigmoid layer to produce the output. This output will also be used as input in the next time step of the LSTM layer for updating the cell state. We express the formulas for computing the ouput layer below.

$$o_t = \sigma(W_o \cdot W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

o_t is the output matrix which is multiplied with the tanh mapped cell state C_t to produce the current time step output h_t.

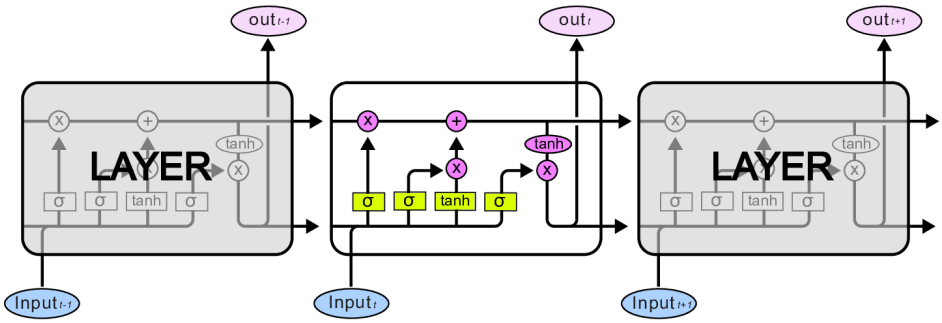

Figure 3.4: Illustration showing the structure of a LSTM node.

3.5 Sequence to sequence learning

Since out problem is essentially a mapping of one sequence to another sequence we need a way to express this for the network. A common and efficient approach is described by Sutskever et al. [41]. In their paper they take on the task of text to text translation from English to French and yielded great results. The setup they used were two deep LSTM networks. One for encoding the input sequence and another for decoding the output sequence. There are two benefits in splitting up the LSTMs, the number of model parameters increases at negligible computational cost and makes it natural to train the LSTM on multiple language pairs simultaneously [23].

3.6 Regularisation

When building neural networks and machine learning algorithms it is important to be aware of overfitting problems. Overfitting is when our model starts to learn features that are specific within the training set. That is, it does not learn the general rules that build up the output data from the input data or rather, it learn more rules than the general ones for the total data set. What this results in is that our training error decreases while our evaluation error increases. What we end up with is a model that performs worse on unknown data due to these training specific rules that the model has learnt. In our model we use two techniques to prevent this overfitting problem. The first model is called weight decay. The second one is called dropout. These two techniques are described in the subsections below.

3.6.1 Weight decay

The goal of all regularisation methods is to make the neural network generalise better over the data it tries to learn. It is a mechanism to decrease the potential of overfitting. The idea behind weight decay is rather simple and has been known for a long time [19]. One way to model a networks complexity is through the number of free parameters of a network, i.e. parameters like thresholds and weights. A

common approach is to minimize the number of free parameters while still keeping the training error small [37, 42, 8].
The weight decay method takes another approach to the problem where we make use of the free parameters of the network instead of minimizing the number of them. The weight decay method is simply an addition to the loss function of the network and can be described through the following equation.

$$E(w) = E_0(w) + \frac{1}{2} + \lambda \sum_i w_i^2, \tag{3.1}$$

in this equation E_0 is our chosen loss function and λ is a weight decay scalar that decide how much the weight decay factor will affect the error. If we have a very small λ value the weight decay will not help to regularize the network. On the other hand if λ is too large our error function will diminish and our network will just aim to keep the weight of the network at 0. In the equation w_i is a weight at index i in the network and the weight decay factor is basically a sum over the squared weights of the whole network. What the weight decay function add to the network performance is that it minimize any irrelevant components of the weight vector by choosing the smallest set of weights that keep the error rate low. When chosen well it can also remove some static noise from the targets [31].

3.6.2 Dropout

When training a neural network with non-linear nodes, as the LSTM network, the nodes learn to identify features in order to predict the correct output. In larger networks there are usually several settings of the weights connecting the nodes that can result in almost perfect prediction of the training data. We want to reduce the number of weights in the network that focus on few strong features of the training set in order to generalize [39]. One strong approach for this is dropout [40]. The underlying mechanism of dropout is to stochastically disable nodes in the network during training. By doing this we prevent units from co-adapting too much [7]. Co-adaption is when two neurons start to detect the same feature repeatedly. By reducing the number of neurons detecting the same feature we are able to learn more features in the training set and eventually perform better test results. Based on several benchmarks it has been shown that the introduction of dropout can give a big improvement on complex neural networks trained on a small training set [40]. When using dropout one must decide the *dropout probability* which express the probability for each node to be excluded when training.

3.7 Attention

The LSTM architecture that is used in this thesis is built on a classic encoder-decoder setting which essentially means that we take the source sentence and encode it into a fixed-length vector. To produce the prediction we then use this fixed-length vector in order to decode the prediction [12]. In research conducted by Bahdanau et al. it was found that this encoder fixed-length vector is a performance bottleneck when trying to improve the performance of this classic encoder-decoder model. The contribution

of Bahdanau's paper was a mechanism called an attention model. What this model actually does is to allow the decoder to make a search for parts in the raw source input that is relevant for predicting the output, without the need of expressing these segments explicitly. The new model creates a set of fixed-length vectors that the decoder can search through in order to find the positions with the most relevant information for each time step prediction. This results in a model that allows the network to keep translation performance even when the translated sentences become longer than the trained corpus [6].

3.8 Bucketing

When building encoder-decoder networks we must define the sequence length of both the encoder sequence and decoder sequence. In order to be able to train phrases of different lengths we either have to create a network for every sequence length and train independently or we have to make all phrases the same length. To make the sequences the same length one usually pad the end of the sequence with a predefined end segment, in our case this was just a vector of zeros. However if we were to pad a sequence built of 50 segments into a full-size 500 sequence the final sequence would be 90% padding segments. The problem of finding a middle way between one network for each sequence length and padding all sequences to a single sequence length can be solved by a method called *bucketing*. Each bucket is a pair of integers determining the length of the encoder sequence and decoder sequence. For each bucket a network is constructed to match the bucket length. Then each phrase is mapped to the closest bucket where the length of the sequence is strictly smaller than the size of the bucket. If necessary the phrase is then padded as previously described to fit the chosen bucket. By using this approach we can find a middle way between a small set of networks and keeping the paddings for each phrase low.

3.9 Data representation

The raw data source is, as described in the *data set* section, the raw audio files of a movie. One audio file in English and one in French. Reading these files yield a sequence of integers, the sampling points, representing the audio of the movie. Since we are trying to map sequences to sequences, why not use the raw sampled signal? There are two main reasons for choosing another representation. The first is that the input representation we chose will be processed as a whole. What this means is that if we would give the system a whole movie as one single input we would not be able to feed single sentences or phrases since the network would look at the input feed as a whole movie. The second reason is more concerned with the actual representation. Each integer in the sequence describes one sampling point. An audio file is usually sampled at around 44100Hz. Which means that for each second we get a sequence of 44100 integers. Even if we would split our movie into phrases or sentences we would still get a very long sequence to learn. Generally longer sequences is harder to learn than shorter [9, 41].
What we aim for is to map one vowel to each time step. A vowel is about 200ms

long so we want to describe every 200ms of our phrases in a rich way to our system. To get this rich description of each vowel we transform the raw signal into a single Fourier transform. The Fourier transform represents the signal in the frequency domain, rather than the time domain as the raw signal. This means that we get a clear look at what frequencies are important for a specific vowel.
The final data representation is in the *.npy* file format. This file format is the standard output of the well-known python library *numpy*, used for scientific computing [44]. The actual data in the .npy files are phrases represented in a filtered short-time Fourier transform mapped to polar coordinates. The reasoning and steps behind this representation will be explained in the following sections.

3.9.1 Raw movie track to phrases

Starting off we have several audio tracks from movies. For each movie we have two tracks, one in English and one in French. The first step is to extract each phrase said in the movies to its own audio file. So the result would be that each movie would have one set of small audio files for each language. To be able to extract these phrases we need to determine when someone start talking and when someone stop talking. We tried two approaches where the final approach was based on the subtitle of the movie.
In our first approach we tried to extract the phrases based on the audio correlation between the movies. The underlying reasoning behind this approach is that since the two movies have the same background noise there should be a large correlation between the signals when no one is talking. In the analog reasoning there should be a low correlation between the signals when someone is talking since it is two different languages. In theory this approach should work well. Two audio sequences with the same background sound and different foreground (voice) sound should yield a good correlation depending on the presence of the foreground sound. The first problem we encountered with this approach was that there was not necessarily a synchronization of the background sound between two audio tracks from a movie. The second problem was that the actual signal of the background sound did not match exactly, which meant that we did not get as good correlation as we needed. The first problem could have been handled by shifting one of the files to match the correlation as good as possible. However due to the second problem the best correlation was not the exact synchronization since the background signal differed.
Another approach we tried was independent component analysis, also known as ICA [22], which is a type of blind source separation [4]. In short what ICA tries to do is to separate a single source into several additive subcomponents by assuming that the subcomponents are statistically independent of each other. This approach did give results when assuming a three component source. However it was far from perfect and the subcomponent containing the voice was filled with artifacts.
Finally we tried the approach based on the subtitles belonging to the movie. An advantage of using this approach is that the actual signal is persevered, we only extract different parts from it based on the subtitles. Another reason for subtitles being a good choice for phrase extraction is that by using the subtitles to a movie as basis for extracting the speech is that although the dubbed phrase might be longer

than the original voice one often try to lip synchronize the dubbed voice to the image as well as possible. Which in turn means that the time spent for a phrase should not vary much in the general case. The variance could be dealt with by adding a small amount of extra time to the extraction window, assuming both languages start to speak at the same time. Although this approach yield decent phrases we found some minor changes that could be made to make them even better.

First off, a sentence or phrase can be longer than what is a suitable length of a subtitle. That is a single phrase can be split into several subtitles. If we split solely on each subtitle then longer phrases get cut in bad places like mid sentence or even worse, in the middle of a word. To reduce this we have an overlap threshold for merging consecutive subtitles.If the time difference between showing two consecutive subtitles, i.e. the time between the first get removed and the second start showing, is less than an overlap time the two subtitles are seen as part of the same phrase. In our experiments an overlap time of about 100ms was enough to catch the absolute majority of phrases. Worth to mention is that this did indeed create a new problem. In the case of a fast phased dialogue our split criteria sees the whole dialogue as a single phrase. Unfortunately these dialogues can be rather long. However they are very uncommon. Less than 3% of our extracted phrases were dialogues longer than 45 seconds. There are also dialogues that reside within the same subtitle. These phrases are obviously not possible to split into distinct phrases with our chosen approach.

There are two reasons to why we want as distinct splits between phrases as possible. The first one is for the mapping between vowels. In really long phrases the last vowels would have a hard time finding which vowels to translate into due to the attention of the earlier vowels. The other is the ability to keep the voice characteristics. In distinct phrases with only one person speaking it is easier for the system to extract the voice characteristics of the voice rather than if several persons talk in one phrase.

3.9.2 Filtering

During the project we understood that a filtering of the data would be necessary in order to shrink the data into a tighter representation that would help the network to converge faster. Our first approach was to use the audio source as given from the phrase split. With all the background noises and all the frequencies recorded by the movie. In theory this could possibly give us a more robust network where it identify the frequencies actually changing between the input data and output data. The problem is that just this step of identifying which frequencies are crucial for the translation is a time consuming problem in itself. When using all frequencies from our data sources with 44.1kHz sampling rate for the data representation we ended up with 8822 parameters to feed into the network and later to be predicted by the network. After doing our, rather simple, filtering we got 1402 parameters to feed into the network. This was done without loosing the characteristics of the voice. However the audio file did sound a bit more like a phone call than a regular audio recording. Following is the important steps to determine where and how to filter the data.

3.9.2.1 Bandpass filtering.

Since the bandpass filter is filtering out frequencies we need to move from the time domain into the frequency domain. That is, we want to transform our data so it is described in terms of amplitudes of frequencies rather than amplitudes over time. This transform is done through a Fourier transform [10]. In layman terms the Fourier transform describes each frequency as a sinusoidal curve where the complex part of the resulting transformation describe the phase offset from omega and the real part describe the amplitude of the sinusoidal curve. By taking the Fourier transform of the signal we can now remove the unwanted frequencies of our input signal in order to make the data representation smaller. In order to determine which curves we want to remove we need to identify which curves represent which frequencies. Once we have the frequency array and the Fourier transform array we can just enumerate through the two arrays and remove the sinusoidal curves that are outside our wanted frequency range.

3.9.2.2 Identifying voice frequencies.

In order to remove the correct frequencies we need to define within which frequencies the human voice usually reside. Based on the works of Titze et. al. [43] the voice frequencies usually range from 300Hz to 3400Hz. By doing this reduction from 0Hz - 44100Hz we were able to bring the number of data point in each segment to 1402. This 85% segment size reduction allowed us to increase our node size from 100 nodes per layer up to 800 nodes. The importance of this increase in the network size will be shown clearly in the *results and discussion* section. The filtered audio files sound similar to a phone call. A bit bulky but definitely still possible to recognize the characteristics of the voice. Especially with the added benefit of removing background noise that is far from the voice signal this transcoding was an important step for the project.

3.9.3 Fourier transform

The Fourier transform is used widely in audio processing. What the Fourier transform does is to move between the time domain and the frequency domain. An important feature of the Fourier transform is that it is invertible so that we can get back the original signal again. The Fourier transform is often used in filtering algorithms since we split up the data into frequencies so we can used algorithms such as band-pass to filter out sound we do not want. The fast Fourier transform (FFT) which is an algorithm that computes the discrete Fourier transform of a sequence is the common implementation of Fourier transform.

3.9.4 Short-time Fourier transform

The short-time Fourier transform (STFT) is a windowed Fourier transform that allows us to provide *time and frequency localization* simultaneously. This is interesting since the sound of a word or sentence is dependent on the order of the frequencies and not just which frequencies are used. The STFT segment the signal into narrow

segments. Then take the FFT of each segment. Thus each segment contain the spectral information of a separate time-slice of the signal. This results in a simultaneous time and frequency representation. As mentioned earlier it is important to be able to invert all operations to get the original signal again. We know that the FFT is inversible. Thus as long as we can keep the ordering of the time segments we can reproduce the original signal. The steps of constructing the STFT can be summed up as follows.

- Choose a window function of finite length.
- Place the window on top of the signal at $t = 0$.
- Truncate the signal using this window.
- Compute the FFT of the truncated signal and save the result.
- Incrementally slide the window to the right.
- Repeat until end of signal.

4

Model

In this section we go through step by step the process of creating the translation system. We start off by describing how we will represent the data in the network. The sections following the data representation goes through the steps needed to go from the source data to our final presentation. Starting with data extraction, followed by filtering process and finally the Fourier transform.

Following the data representation section is the *network architecture* section. This section is dedicated to describe how we built up the network used during the experiments.

4.1 Network architecture

As we have described how we pre-processed the data in order to get it to a suitable representation for our translation system it is now time to describe how we built the actual network. We start by describing the LSTM network setup and then explain the loss function used in the backpropagation during learning. Finally we describe the regularization and attention mechanisms added to yield a more complex system.

4.1.1 Loss function

Our goal with the loss function was to keep track of how far away we are from the correct value of each cell in the output sequence. However we want a single value to describe the loss. Therefore we implemented the loss function as the average of the squared error fo each cell in the whole output sequence. The following equation shows how it is computed.

$$E(\mathbf{O}, \mathbf{C}) = \frac{\sum_i^N \sum_j^M (O_{i,j} - C_{i,j})^2}{N \cdot M}$$

In the above equation O is the output sequence and C is the correct sequence. N is the length of the sequence and M is the length of each segment. $C_{i,j}$ is the cell at sequence step i and segment location j in the correct phrase. The whole phrase sequence can be seen as an NxM matrix. This loss function gives us a way to understand how well we are performing during training as long as we just know within what range the cell values reside.

4.1.2 Output projection

When we described the LSTM architecture we mentioned that we needed a loop function when predicting and that this loop function was basically an output projection. This output projection is also used at each time step of the sequence when going from decoder sequence to STFT sequence. The output projection is a simple linear projection

$$y = XW + \beta.$$

Where W is the weight matrix, X is the output sequence, β is the bias term, and y is the final STFT output sequence.

4.1.3 Regularization and attention

As introduced in the background we used weight decay and dropout for regularization. The dropout mechanism were put onto each input and output layer of the LSTM network as well as on the output projection.
The attention model we used for the thesis was based on the work of Vinyals et al [45] as described in the background. The mechanism is rather straight forward and no real parameters are needed to be set.

5
Data set

Since we could not find any open data set available online we had to make our own. As we stated in the goals section we defined four points to describe our desired data set. The same bullet points as in the goals section are repeated below.

- large set of phrases where each phrase is available in both English and French
- phrases of variable length
- a natural flow in the speech, as in day-to-day conversation
- a great variety of people talking with different voice characteristics

A data source that fit these points are dubbed movies. The French dub a lot of movies so there is a rich range of movies to choose from in order to get a variety in voice characteristics. Movies generally try to mimic everyday speech as well so we will get a rather realistic level of distinct pronunciation and flow. Depending on the genre of the movie the length of the phrases can be variable which makes the wide range of titles to choose from great.

5.1 Choosing the movies

As we seek diversity within our data set it is important to think about the genre distribution of the chosen movies. The movies we chose is mainly from the drama and action genre. In our experience a large part of the content of drama movies are made up of dialogues and as the name of the genre suggests it is often filled with different feelings. This makes the drama genre a good fit for our data set since we can expect to get a lot of phrases per movie. The action genre usually have less dialogues than drama movies but they have two interesting elements within the dialogues. The first is that they are usually shorter than dialogues in drama movies. This allows us to steer the mean of the phrase length distribution lower which will help training and evaluating the system. The second benefit is that it tend to be a lot of background noise during the dialogues. However we do filter away a lot of the unwanted frequencies but there still tend to be some background noise left. This background noise allows for testing the robustness of the model to be made. These tests are outside the scope of this thesis but can be interesting for future work.

5.2 Subtitles

As described in the data representation section the subtitles to a movie is an important part of our project. With the subtitles we define where to extract a phrase from the movies. We tried to use three different setups of subtitle files English, French,

and Swedish subtitles. Our initial approach were to use the subtitle file that were matching the audio language as we though they would match best in terms time to speak a phrase and how the subtitles were split. This approach did not work well since there were no guarantee that the two subtitles files matched. For example one subtitle file could have written down the song text for a song that is played in the movie while the other one did not. Therefore the problem of connecting two phrases to each other became even harder than before. We continued with looking at a single subtitles file but in a language not represented in the audio files, e.g. Swedish subtitles. This strategy solved the phrase matching problem but the overall quality were rather poor since the subtitle text needed for the phrases could vary greatly in length compared to the spoken phrase. But a single subtitles file solved the problem of matching phrases since we extracted the same times in each movie. We went back to the previous subtitles files and decided to only use the English subtitle for extraction. The reasoning behind this choice were based on availability and original source. There are far more subtitles available for English subtitles than French so we could tweak the synchronization easier. Since the original language of the movies were English and the dubbed movie try to mimic the English articulate which resulted in a good extraction from the English subtitles. There were a problem with the synchronization between the audio files but by using the overlapping method described in the data representation section and adding a small extra time frame in the beginning and end of the extraction window we managed to usually get all the speech.

5.3 Filtering subtitles

As previously mentioned there were a problem with some phrase extractions not containing any speech. To a large extent this is due to subtitles sometimes having describing text for background sounds as songs, explosions and sounds relevant to the current scene. These describing texts are usually not a large part of the subtitles except for hearing impaired subtitles. We did actively make the choice to not use the hearing impaired subtitles to minimize the amount of extracted phrases that did not contain any speech. But it still happen that some of these describing sounds get written out in the subtitle. Our approach to handle this were to remove the first subtitle in each file, since it is usually showing which movie it is and who have done the translation. Then we removed all phrases that were less than a second long. Usually these are just describing sounds but sometimes they may be exclamations of single words or names. We made the decision that these phrases are not of interest to us since names should be dubbed equally and exclamations does not contain any context by itself which is an important aspect of this project. However this did only take away a small part of the unwanted phrase extractions. We noticed that for some types of non speech subtitles there were a starting and ending character explaining what the subtitle was, for example a note when a song is played and the song text is written out. Identifying these characters and skip the extractions of phrases containing these characters could be a possible expansion to out data set. However we did not prioritize the analysis necessary to make such an extension satisfiable in this thesis.

5.4 Statistics

The final full data set that we built from this data were constructed from 16 movies and each movie generated approximately 700 phrases ranging from 3 seconds up to 50 seconds. This gave us a total of about 11000 phrases. The distribution of these phrases into the buckets we chose as 50, 100, 150, 250, 350, 500 can be seen in Figure 8.3. The number of phrases is clearly skewed toward the smaller buckets indicating that we have the majority of the phrases between 3 to 10 seconds and very few above 40 seconds. This distribution allows us to focus on translating shorter phrases and use the longer phrases for testing longer dependencies.
An important part when implementing our data pipeline for generating the data set were to be able to scale up fast when needed. As a result we could go from raw movie format to the final STFT representation in about ten minutes per movie with minimal manual attention.

6
Experiments

This project resulted in three types of experiments. The first experiment look at the performance of the network. The second experiment is closely tied to the translation performance goals stated in the introduction section of this thesis. Finally we conduct experiments to evaluate our created data set.

6.1 Network performance

As an experiment to see how our model learned we wanted to get a grasp of how the size of the network and the size of the data set determine how well we are able to translate. In order to examine this we made two experiments. In the first experiment we changed the size of the model, more specifically the number of nodes in each LSTM layer. The number of layers were 2 in these experiments and remained the same independent of the number of nodes in each layer. The different network sizes we trained were 10, 50, 100, 200, 400, 600, 700, 800. We used all phrases of the bucket with length 100 from the full data set. We chose size 100 since it was not the smallest bucket length and at the same time it was the bucket with the most number of phrases. The model did therefore not use bucketing, however we did use an attention model and the regularization techniques. Each model were trained for 20 hours and then we compared the training error and evaluation error between the different sizes of the network. The question we wanted to answer with this experiment is if there is a relation between the error of the network and the size of the network.
In the second network performance experiment we used a static network size, two layers with 800 nodes in each layer, the largest we could fit on our GPU. In this experiment we changed the number of training elements available for each model. The different data set sizes were 10, 50, 100, 200, 500, 1000, 2000, and finally all phrases from the original data set. We still only used the bucket with size 100. The reason for only running one bucket was mainly due to computational time of training all buckets. These models also used regularization techniques as well as an attention model. Each model were trained for 20 hours and then evaluated in terms of training and evaluation error for each network. Our aim here were to find out if there is a correlation between the number of elements available for training the model and the error of the model.
The reason for making these two experiments is to see if our model is learning but is heavily prone to underfitting. If the model is not underfitting this approach for translation is not working. On the other hand, if we can identify that the network

is learning but is underfitting the data we can conclude the result as a basis for making future attempts to scale up the network since the current network basically can not learn the large fine grained information embedded within each phrase.

6.2 Translation

The first experiment conducted was based on the whole phrase corpus of 11000 training phrases and 1100 test phrases of different length. The model were our most complex one with bucketing, and regularization in the form of weight decay and dropout. We also used attention on this model. The phrases were split into six buckets of size 50, 100, 150, 250, 350, and 500. The model were trained to translate from French to English. The model trained for 15 hours and were evaluated after every 500 phrases trained.
Based on the results of the previous experiment, which can be seen in the result section, we stripped the model of all extra features and made the data set much smaller. The new model only use one single bucket, the 100 bucket. It does not use any regularization techniques nor an attention model. The phrases used were from a single movie source which lead to 40 phrases for training and 4 phrases for evaluation. The evaluation were done with the same time step interval as the previous model and the result of this model can be seen in the result section.
In the translation experiments we qualitatively evaluate how well our models perform. We look at how easy it is to hear the voice in the generated translations. How well the generated voice sound in comparison to the actual output voice of a translated phrase and how large part of the input phrase the model manage to translate.

6.3 Data set

Since we have a goal to create a data set suitable for speech-to-speech tasks we need to evaluate how well it works for this purpose. First we will look at the size of the generated data set. Further we will evaluate the distribution of the phrase lengths. We evaluate how natural the speech is as well as how many different voice characteristics there exist. Finally we evaluate how well the phrase extraction mechanism work by statistically look at the data set in terms of faulty phrases like bad cuts or no voice element.

7

Implementation details

In this section we describe the hand-on tools and technologies used to implement our model. We also describe the parameter configurations used for the training and testing.

7.1 Tensorflow

To build these networks we have used an open source library released by Google [3]. The Tensorflow library contain functions to build and run neural networks as smooth and seamless as possible. Since it is well documented and have a good set of examples to build from, including a sequence to sequence example, it became a good choice for this project. Tensorflow is overall well built but do indeed still have some bugs due to its short lifetime. However the Tensorflow team is constantly updating and improving the library and its current state makes it a strong candidate for creating neural networks. One main advantage with Tensorflow is the ease of assigning the network computations to GPUs and as described in the background the use of GPU processing is necessary for networks of this scale.

7.1.1 LSTM

The long short-term memory network we use in our system is built up of two layers with 800 neurons in each layer at most. The inner workings of the LSTM network has been described in the background and the Tensorflow library provide function call *BasicLSTMLayer* to create the layer. Since our data representation is larger than the neuron size of the network we need to define the input dimension when creating the first LSTM layer. When defining this input dimension Tensorflow build an input projection in order to go from the data representation size to the LSTM size. When testing the network we need to provide a loop function. This loop function is used at output of each decoder step in order to predict the input to the next time step. The loop function used for this network is the same function as the output projection described below. When training we skip the loop function and the network instead insert the correct prediction as input for each next time step. This is done to allow the network to keep learning throughout the whole sequence even when the first outputs are bad.

7.1.2 Bucketing

In the model we map each phrase into its matching bucket. We used six different bucket sizes, 50, 100, 150, 250, 350, and 500. Tensorflow supply a function call to create a bucket model, which we built our bucket model on. The difference between our model and the supplied bucket model was that we wanted to keep the loss function separate from the model and thus we had to implement it without the loss function. The bucket modeling function creates a model matching the longest bucket. Then for the smaller buckets it slice the encoder and decoder length to match the smaller bucket. By doing this we get a model that make use of the same training between different bucket models for encoder and decoder steps that are shared between two buckets.

7.2 Regularization and attention

We used a dropout probability of 20%. Worth to know is that Tensorflow has defined the dropout probability as the keep probability, so to get a dropout probability of 20% one should set the keep probability as $1 - 0.2 = 0.8$.
The weightdecay were simply added as a new factor to the loss function

$$E(\mathbf{O},\ \mathbf{C}) = E(\mathbf{O},\ \mathbf{C}) + \sum_{W} |w|\lambda.$$

Where W is all weights in the whole network and w is a specific weight. An important part of the weight decay is the λ factor. This determines how much influence the weight decay has to the output error in relation to the loss function. We used a λ value of $1e^{-4}$ in order to keep it inferior to the actual loss function and rather just keep the weight low when possible
The one parameter we can set in Tensorflows implementation of the attention mechanism is the parameter determining how many encoder states we can look at simultaneously. We used the predefined single encoder state lookup which is the only we have seen in other experiments as well.

8

Results and discussion

This section present the results from the stated experiments and discuss the important elements of the results and the reasons behind the results. We start off by looking at the network performance experiments and then continue to the translation experiments, finally we discuss the created data set. Each subsection starts with presenting the results for each experiment and introduce the reader to the important features of the results. Each subsection end in a discussion about what the yielded results conclude.

8.1 Network performance

When running our network performance experiments we started with the network size experiment where we ran 8 different network sizes for 20 hours and then plotted the loss for each network as can be seen in Figure 8.1. What we see here is a clear decrease in the loss function depending on the size of the network where a larger network result in a smaller training and evaluation error.

The actual decrease in the loss function is rather small over the size increase of the network. However the decrease is consistent and happen between all size increases except size 400 and 600 where it actually did decrease, just not enough to be caught in the plot. What we can conclude from this experiment is that we would probably benefit from being able to increase the network size more in order to get better results.

The second network performance experiment were the one where we alternated the training size of the largest possible network, the one with 2 layers and 800 nodes. The resulting plot can be seen in Figure 8.2. This plot shows three interesting things. First of all it shows that when using a large training set we are able to generalize over the test set in a rather similar way as to the test set since the difference between the evaluation error and training error decrease as the training size increase. Further we see that when we use a very small set of training points we are able to yield very good results on the training data, however we perform very bad on the test set for obvious reasons. This tells us that the network is actually able to translate previously seen phrases as long as the training set is small. Finally we see that the increase in training error is rather fast in the early increments of the training size, which again indicates that we need a larger network in order to get better results.

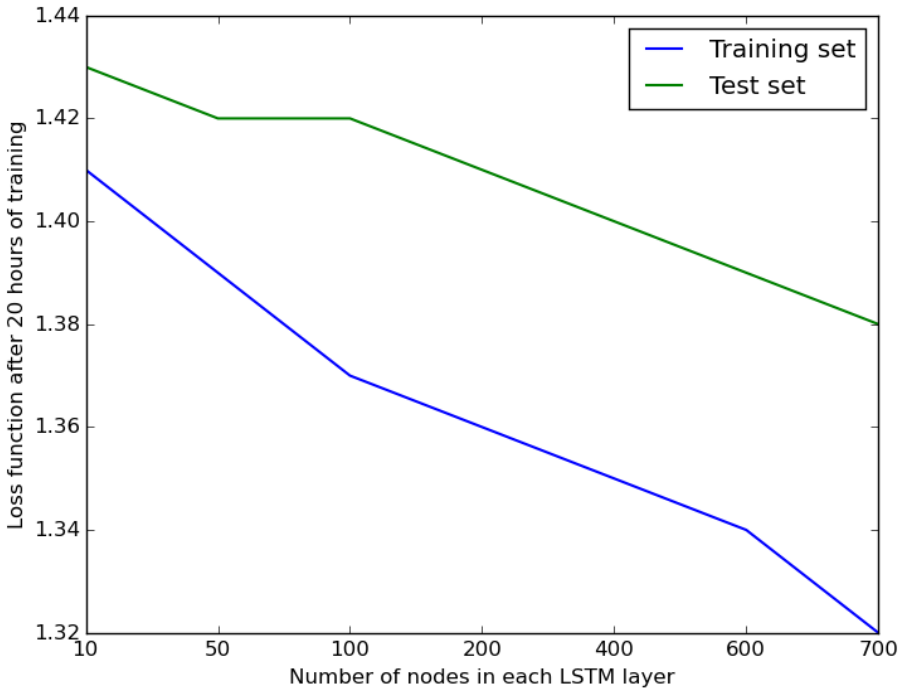

Figure 8.1: Graph showing how the loss function of the training and evaluation set change depending on the size of the trained network.

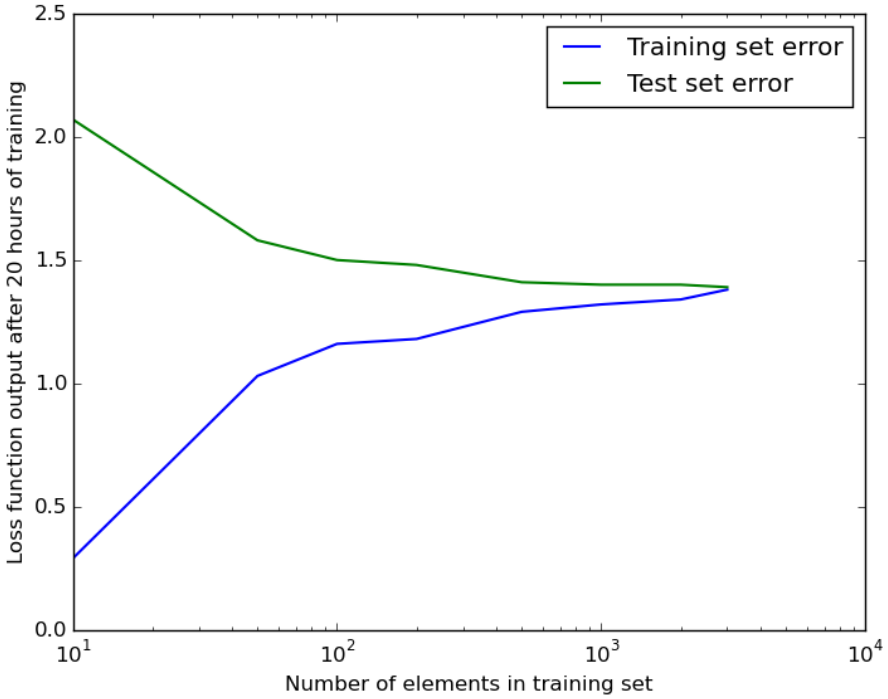

Figure 8.2: Graph showing how the loss function for the training and evaluation set change depending on the size of the training set.

8.2 Translation

Both networks used for the translation experiments were trained for 20 hours and each network trained a total of approximately 90000 phrases during that time. The complex network trained using all 11000 phrases could only output noise after these 20 hours. Although the regularization parts of the network should indeed yield a larger error from the network we thought it would perform better than just plain noise both for training data and evaluation data.
The results for the smaller network trained on 40 phrases were better. When allowing the network to correct the input for each time step the network were able to perfectly reconstruct the output phrase. When not assisting the input for each time step the network managed to keep the output with a clear voice and distinct characteristics of the speech for about 10 out of 100 segments of the input sequence before turning into noise. These first segments resulted in a very similar output voice and the words said were just as easily identified as in the actual output. One difference were that the background noise changed its pitch slightly although the characteristics of the background noise remained the same.
When evaluating the test data set on the small network we found some interesting elements. When only predicting one time step at a time the test set resulted in total noise. No human like voice element were present. However when allowing the network to predict all the time steps by itself it actually produced some human like voice output for about 15 out of 100 segments. We believe that the reason for the single time step prediction to perform worse than the full prediction is that the single time step keeps dragging the prediction away from the learned sweet spot of the network which in turn results in noise. Compared to the full prediction where it finds the closest sweet spot of the network and tries to predict from there but since the input sequence is far away from previously known inputs it results in human-like nonsense. The characteristics of this nonsense were tough to define, but it was clearly far away from the characteristics of the correct output. It was closer to the input phrases that the network had trained on in terms of speed and pitch. Another interesting feature were that the full prediction output managed to identify some simple and distinct elements of the background noise, like a car driving a way, when no voice were present.

8.3 Data set

As we previously stated the total size of the training set is about 11000 phrases. In terms of audio data sets our generated data set seem to be of medium size. However compared to the data sets used for text translation this set is very small. An approximation gives us that our data set consists of about 90000 words per language where data sets used for yielding good translation results reside at about a couple of million words in each language. However our results were not subject to lack of data in this case and our data pipe line is designed in such a way that we can scale up the data set with little effort when needed. The reason for wanting to get a data set in the millions of words is for when we have a model that can translate

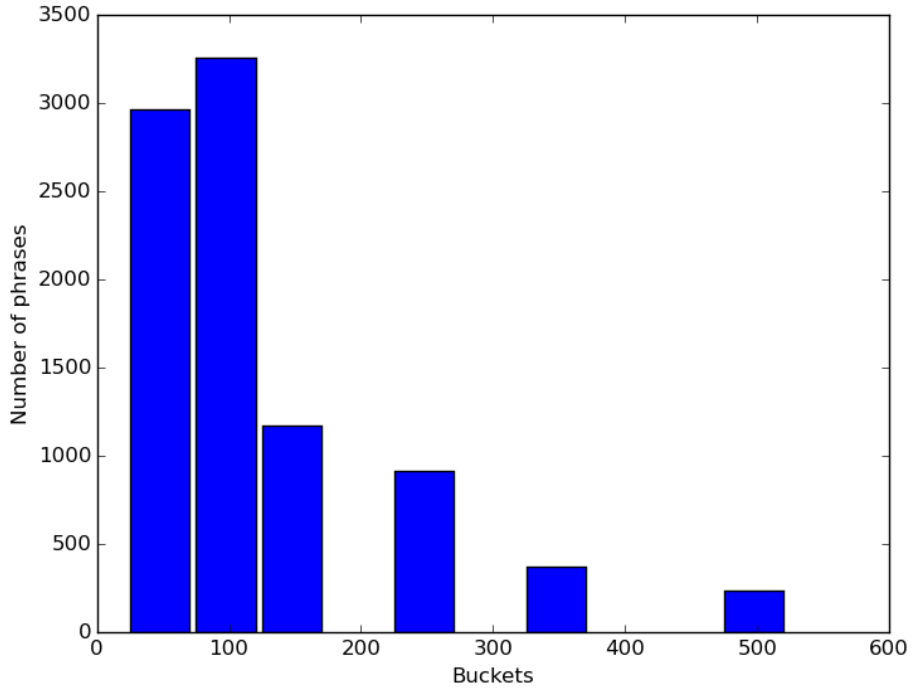

Figure 8.3: Distribution of phrases in data set over defined buckets.

more general and want to test it against a base. In such a scenario the systems need to have an equal amount of data to get a fair comparison.
Figure 8.3 clearly shows that we succeeded to get a distribution of the phrase lengths that have a mean around a phrase length of short to medium sized phrases of about 5 - 10 words which will make the evaluation of models easier due to the time dependency not being very large. We still have some very long phrases which are good for evaluating very difficult input data where several voice sources may be present.
Based on a randomly selected subset of 100 phrases from each language we can determine that about 5% do not contain any voice element, in those cases both the English and the French version of the phrase lack a voice element. We can also conclude that 7% of the sample set is cut in a way such that we lose some part of the voice element in the phrase. In the case of cutting the phrase badly it was always in the end of the phrase that the cut were bad and we could not see a strong correlation between a bad English cut and a bad French cut. Based on our approach for extracting phrases we get a $1:1$ relation between the English phrases and the French phrases. Our selection of movies from different genres also yield a wide variety of voice characteristics.
Each movie used for creating the data set had a large variety in both movie characters and emotions shown by these characters as well as some accents. This resulted in a data set with a large variety of voice characteristics as we aimed for.

9
Conclusion

Looking at the results of the network performance experiments combined with hearing the translation output from the trained network we can draw some conclusions. First of all, there is no doubt that the LSTM network architecture is complex enough to learn this type of sequences. Based on the error rate of training phrases with a small data set we are able to translate the seen phrases almost perfectly. The problem we find here is that the network is not large enough to handle a more general setting with much larger data set, that is the network is underfitting the data. We can also see that as the training set increases the test error moves closer to the training error. This indicates that there are similarities among the phrases that can be identified by the network if we are able to scale up.
We managed to create a data set that suited all of our needs and that had a very good distribution among the lengths of the phrases. However the time step added to the start and end of each phrase should be determined by a systematic search in order to reduce the bad cuts. Trying to eliminate the non voice phrases by identifying non voice subtitles could reduce the number of bad phrases in the data set. The pipeline setup for creating this data set also allow us to scale up the data set fast if need be.

10
Future work

As we conclude this thesis we have identified two potential extensions to this project that would be interesting to examine.

Network scaling
The first extension we would like to investigate would be how to scale up the network. As we trained the networks we put the whole network computation onto a single GPU. The next step would be to split up the LSTM layers so that each GPU do the computations of a single layer. This would allow us to scale up the nodes in each layer in order to persevere more information about the trained phrases. Another extension would be to put a regular feed-forward neural network onto the input and output of the LSTM network in order to decrease the input size and thus be able to increase the LSTM size. By using a pair of hidden layers in the feed-forward network and nonlinear activation functions in the nodes we could potentially transform the input into rather complex and small representations.

Data set remodeling
Although we find our created data set to have a lot of nice features there are possible changes to be made in order to try to get the current network to perform better. A first step would be to remove the overlapping feature that we used when extracting the phrases. This would yield a larger data set but with much smaller phrases, ranging from below one second to a maximum of $3 - 4$ seconds. This removes the possibility to test the networks ability to remember long term dependencies in languages but could help the network in terms of ability to translate since the sequence length to be translated would be shortened a lot. Another data set change that could help the network would be to use a stricter filtering policy. Currently we allow all frequencies between 300 and 3400. One could look at the potential of decreasing this range and even to remove parts within this range in order to tight up the segment dimension of the input to the network.
As mentioned in order to get a cleaner data set it would be necessary to clean the subtitles files from non voice subtitles and to make a systematic search for a good time step to add to the beginning and end of each phrase extraction.

Bibliography

[1] Skype translator. https://www.skype.com/en/features/skype-translator/. [Accessed: 2016-05-16].

[2] Skype voip software. https://www.skype.com/. [Accessed: 2016-05-16].

[3] Martín Abadi, Ashish Agarwal, and Paul Barham et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Ranjan Acharyya. *A New Approach for Blind Source Separation of Convolutive Sources - Wavelet Based Separation Using Shrinkage Function*. VDM Verlag Dr. Mueller e.K., 2008.

[5] Jonathan Allen, M. Sharon Hunnicutt, and Dennis Klatt. *From Text to Speech: The MI Talk system (Cambridge Studies in Speech Science and Communication)*. Cambridge University Press, 1987.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *The Computing Research Repository*, abs/1409.0473, 2014.

[7] Pierre Baldi and Peter J Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc., 2013.

[8] Eric B Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[10] Ronald N. Bracewell. *The Fourier Transform & Its Applications*. McGraw-Hill Science/Engineerin, 1999.

[11] Maureen Caudill. Neural nets primer, part vi. *AI Expert*, 4(2):61–67, 1989.

[12] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *The Computing Research Repository*, abs/1409.1259, 2014.

[13] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Mike Seltzer, Geoffrey Zweig, Xiaodong He, Julia Williams, et al. Recent advances in deep learning for speech research at microsoft. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8604–8608, 2013.

[14] Matthias Eck, Ian Lane, Ying Zhang, and Alex Waibel. Jibbigo: Speech-to-speech translation on mobile devices. In *IEEE Spoken Language Technology Workshop*, 2010.

[15] eWeek. Microsoft's skype translator provides on the fly translations. http://www.eweek.com/cloud/microsofts-skype-translator-provides-on-the-fly-translations.html, 2015. [Accessed: 2016-05-15].

[16] James Fung and Steve Mann. Using graphics devices in reverse: Gpu-based image processing and computer vision. In *IEEE International Conference on Multimedia and Expo*, pages 9–12, 2008.

[17] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649, 2013.

[18] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[19] Geoffrey E Hinton. Learning translation invariant recognition in a massively parallel networks. In *Parallel Architectures and Languages Europe*, volume 258, pages 1–13. Springer, 1987.

[20] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Master's thesis, Institut für Informatik, Technische Universität, München*, 1991.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[22] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[23] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Empirical Methods in Natural Language Processing*, volume 3, page 413, 2013.

[24] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Accessed: 2016-05-19].

[25] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[26] Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, et al. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 451–460. ACM, 2010.

[27] Chi-Ho Li, Minghui Li, Dongdong Zhang, Mu Li, Ming Zhou, and Yi Guan. A probabilistic approach to syntax-based reordering for statistical machine translation. In *Annual Meeting-association for Computational Linguistics*, volume 45, page 720, 2007.

[28] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

[29] Microsoft. Skype translator presentation. https://www.youtube.com/watch?v=rek3jjbYRLo. [Accessed: 2016-04-30].

[30] Microsoft. How technology can bridge language gaps. http://research.microsoft.com/en-us/research/stories/speech-to-speech.aspx, 2015. [Accessed: 2016-05-14].

[31] J Moody and S et al. Hanson. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1995.

[32] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, 2002.

[33] Y. Qian, Y. Fan, W. Hu, and F. K. Soong. On the training aspects of deep neural network (dnn) for parametric tts synthesis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3829–3833, May 2014.

[34] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. 1993.

[35] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[36] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[37] Daniel B Schwartz and Vijay K et al. Samalam. Exhaustive learning. *Neural Computation*, 2(3):374–385, 1990.

[38] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014.

[40] Nitish Srivastava and Geoffrey et al. Hinton. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014.

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *The Computing Research Repository*, abs/1409.3215, 2014.

[42] Naftali Tishby, Esther Levin, and Sara A Solla. Consistent inference of probabilities in layered networks: Predictions and generalizations. In *International Joint Conference on Neural Networks*, pages 403–409, 1989.

[43] Ingo R. Titze. *Principles of Voice Production*. Prentice Hall, 1994.

[44] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[45] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. *The Computing Research Repository*, abs/1412.7449, 2014.