Direct Speech to Speech Translation Using Machine Learning

Sireesh Limbu

Degree Project in Computational Science: Report

April 2020

Contents

Abstract

Nowadays, most of the speech to speech translation applications and services use a three step process. First step is the speech to text translation using speech recognition. This is followed by text to text language translation and finally the text is synthesized into speech. As the availability of data and computing power improved, each of these individual steps advanced over time. Although, the progress was significant, there was always error associated with the first translation step in terms of various factors such as tone recognition of the speech, accent etc. The error further propagated and quite often deteriorated as it went down the translation steps. This gave birth to an ongoing budding research in direct speech to speech translation without relying on text translations. This project is inspired from Google's 'Translatotron : An End-to-End Speech-to-Speech translation model'. In line with the 'Translatotron' model this thesis makes use of a simpler Sequence-to-Sequence (STS) encoder-decoder LSTM network using spectrograms as input to examine the possibility of direct language translations in audio form. Although the final results have inconsistencies and are not as efficient as the traditional speech to speech translation techniques which heavily rely on text translations, they serve as a promising platform for further research.

Keywords: Encoder-Decoder, Sequence to Sequence Modelling, LSTM Neural Networks

Acknowledgements

I would like to thank my supervisor Henrik Nässén for believing in me and for providing me the opportunity to work in this project. I would also like to express my gratitude to Prof Gustav Eje Henter for his informal yet extremely helpful guidance and my reviewer Prof Per Matsson.

1 Introduction

Although the translation services today provide a robust trained model with reasonable results and latency, the advent of globalisation demands for a better translation system. This serves as a breeding ground for a deeper research into the training of translation models and their architectures.

The objective of this project lies in translating language directly using voice characteristics of the source speaker. Hence, the key difference in this approach compared to the general machine translation techniques available today is the lack of an underlying text representation step during inference. Today most of the general translation services use the common 3 step process. First step is the speech recognition for transcribing speech to text [1]. Second step involves the text-to-text translation [2] followed by the final step of speech synthesis [3]. Although, these steps have individually advanced over the years, this project aims to develop a proof of concept to provide evidence supporting a unique translation system that might prove to be better and faster. Although it has to be noted that this task is extremely challenging for various reasons. Unlike Text-To-Text (TT) translation systems which require text pairs for end-to-end training, the Speech-To-Speech (STS) model training requires speech pairs that are more cumbersome to collect. In this project the model is trained to map speech spectrograms for both source and target speakers. To this end the uncertain alignment of spectrograms may lead to errors in source and target audio mapping.

Our aim is to probe into the possibility of a similar functional network structure but more fundamental and straight forward. Besides the difficulties mentioned above, there are technical nuances and code structures that are complicated to follow and simplify especially without a guide for non-theoretical matters.

1.1 Motivation

The need to address a speech based translation system roots from the paralinguistic features such as prosody, voice and emotion of the voice speaker that is ignored in the common Text-To-Text translation systems. Our motivation rests specifically on training a system that can ideally encapsulate the aforementioned features as it translates one language to another. Moreover, the cascaded 3 step machine translation systems carry the possibility of aggravating the errors encountered with each progressing step. In contrast, the Speech-To-Speech systems are advantageous over the cascaded TT systems as they follow a one step process which requires lesser computation power and have improved inference latency.

1.2 Problem Definition

This project seeks to investigate the feasibility of a language translation tool by emulating the 'Translatotron' [4] research conducted by Google in a smaller scale. As the research claims no usage of underlying text representations other than during the model training for the final translation, this thesis aims to achieve the same.

As we are dealing with data where order is important, we make use of the Long-Short-Term-Memory(LSTM) neural networks to design the model. Their improved performance over regular neural network for STS learning or order sensitive data such as spoken words and sentences have proven them to be a worthy building block for the translation algorithm [5].

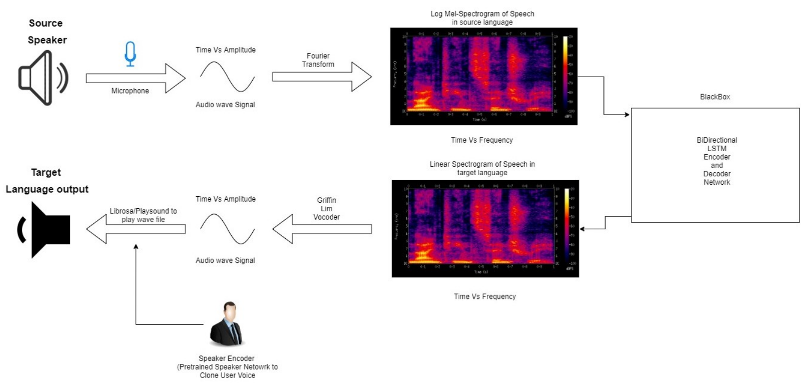

Source
Speaker
Microphone
Time Vs Amplitude
Audio wave Signal
Fourier Transform
Log Mel-Spectrogram of Speech in source language
Time Vs Frequency
BlackBox
BiDirectional LSTM Encoder and Decoder Network
Linear Spectrogram of Speech in target language
Time Vs Frequency
Griffin Lim Vocoder
Time Vs Amplitude
Audio wave Signal
Librosa/Playsound to play wave file
Target
Language output
Speaker Encoder (Pretrained Speaker Netowrk to Clone User Voice

Figure 1: The Translation Model Outline

We make use of the high level representations of the audio data for both target and source to compensate for the lack of pre-defined mapping for the purpose of training. The proposed model in itself is a modified version of Sequence-To-Sequence acoustic modelling for voice conversion [6] which outperformed in the Voice Conversion Challenge in 2018. Following is the overview of the model we use in this project.

1.3 Research Goals

The two substantial outcomes of this project that can be used as is for any further research are as follows:-

- Speech pairs generated for training the model using Google Text to Speech. As mentioned earlier, one of the reasons that makes this project challenging is the data collection. Furthermore, [7] also showed that training with synthetic data proved to be better than multitask training that has been carried out in the Translatotron model. Since, there are plenty of sources for text pairs in various languages we make use of Google Text-To-Speech free source API to generate our training dataset. We explain the data pre-processing step in detail later.
- The main translation model along with specific areas of future work that has been mentioned in this report can be used for studies in language translation using utterances.

1.4 Outline

This project report begins with the explanation of fundamental concepts based on which the model has been designed, in the background section. We then elaborate on the way the data has been processed in order to conform to the format and logic required by the model. This is followed by related work in the field of language translation and the services available as of today. The section also explains the models the services use and how it's different from the model we use. In the next section we dive deep into the model architecture by bringing together all the different components explained in the background section. Although we explain all the different sub-components used, it should be noted that only the final version of the model uses all the different sub-components explained in this section.

After having built the model in the next section we perform experiments with the processed data for each model version explaining the gradual improvement in the final output also explaining meanwhile the reason for choosing the modelling tools we have used. The results of the experiments are then discussed in the following section. Finally we end the report with the conclusion based on the results and our intended goals for this thesis project. We also provide the shortcomings and future work that can be implemented for further studies in this last section.

2 Related Work

This section provides insight into the services available for language translation. These services do not necessarily delve into speech synthesis while translating but use a similar encoder-decoder network as has been implemented in this project for translation purposes.

2.1 Google Translate

With over 500 million users and 100 billion words translated daily, Google Translate is the most convenient and smoothest online text translation service available since it's inception in 2006. Originally the network by default used Statistical Machine Translation(SMT) [8] which uses probability theory to translate word and phrase based translation trained on text pairs based out of UN and European parliament documents and transcripts. As SMT used predictive algorithms to come up with translated texts the grammar part could not be accomplished. Eventually the network was evolved to a Neural Machine Translation(NMT)[9] model system which could translate sentence based texts rather than just words. With just a few languages initially Google Translate now provides translation services for around 109 languages. As of now, the service also provides speech to speech translation using the 3 step translation process as mentioned earlier.

While translating, the network model searches for patterns amongst millions of documents to figure out the most logical word sequence in target language with considerable accuracy. The accuracy varies depending on the language and have been ridiculed by critics, but is still considered the most impressive translation model.

Furthermore, in the interest of this project we use Google Translate API which is a publicly available library that can be installed and called for translation in raw python code. For our project we have used this API to generate translated pairs using text pairs in source and target languages for training our model.

2.2 Moses

Moses [10] is a free source translation system that uses Statistical Machine Translation a similar Encoder Decoder network to ours. The network can train a model to translate any language pair. The model needs the collection training pairs to train after which to translate it looks for the best probable translation. The training words and phrases are word-aligned guided by heuristics to remove misalignment. The final output of the decoder is passed through the tuning process. In the tuning process, a bunch of statistical models are weighted against each other to come up with the best translation model based on various scores. Consequently, the word or phrase translations do not follow grammar. Furthermore, In terms of speech to speech translations, Moses as of Sep 2020 does not put up an end to end architecture.

2.3 Microsoft Translator

Microsoft Translator [11] is a cloud based translation service that can be either used individually for personal use or for enterprises and organizations. It has a cloud based REST API for speech translations that can be used for building translation support for websites or mobile apps that require language translation support. The default translation method for Microsoft Translator API is Neural Machine Translation. Originally known as Windows Live Translator - Bing Translator is a translation service provider that translates websites and texts online.

Using Microsoft translator's Statistical Machine Translation system, Skype translator provides an end to end speech to speech stand-alone translation service as a Mobile and Desktop application. Skype translator also supports instant translation for direct text messages into more than 70 languages.

2.4 Translatotron

As mentioned earlier Translatotron is the inspiration behind this thesis project. It is a translation service currently in beta phase funded by Google Research. The research was conducted for Spanish as the source and English as the target language The technical aspect in terms of raw code is still unavailable to public as of September 2020. The model is an attention based sequence to sequence neural network. The whole encoder decoder structure is trained to map speech spectrograms to target spectrograms using speech pairs called utterances. One of the important features of this model is the capability of the model to replicate the voice of the speaker in the translated language. This model has been trained on two different sets of Spanish to English datasets : Fisher Spanish to English Callhome corpus [12] and a large corpus of text pairs synthesiszed into speech using Google Translate API [13]. The resulting translated and source audio pairs can be found here [1]

Our goal is to build a network in the similar form but simpler, more understandable and study the possibility of a working algorithm. Although the speech synthesis part (post-decoder) in the original research has been modified from Tacotron 2[14] which is a very complex project in itself. We investigate to see if a small scale implementation of the model works. Translatotron also makes use of Voice Transfer after synthesizing speech audio from the spectrogram similar to Parrotron[15], which is a voice transfer neural network model again funded by Google Research. For the sake of keeping the project simple we exclude the voice conversion and auxiliary decoder part for this thesis. We delve into the model architecture further in section 4.

3 Background

This section provides a background to the reader on all the basic fundamentals required to build the model.

[1]https://google-research.github.io/lingvo-lab/translatotron/

3.1 Machine learning libraries and platforms

3.1.1 Python

Python[2] is one of the most popular high level programming language. Due to it's well organized inbuilt functions and helpful community Python is known to be very user friendly. The whole model including the data generation and cleaning has been built from scratch in python 3.0 using Jupyter Lab IDE.

3.1.2 Google Cloud and Colab

Google Colaboratory is an open source online platform for running Jupyter Notebook environment for Python. This platform provides access to powerful computing tools and file storage capability to process large amounts of data. It allows saving data analysis and simulations entirely in the cloud. The Jupyter Notebook currently provided by Google Colab uses Python Version - 3.6.9

3.1.3 Kaggle Kernel

Kaggle is the most popular online platform for various coding competitions specifically in Machine learning and AI. It is also a repository of various datasets provided by several organizations including research universities who outsource their deep learning problems to various Kaggle users in lieu of cash prizes. Kaggle provides one of the best free platforms backed up by a powerful NVidia Tesla P100 GPU with 2 CPU cores and around 16 GB of RAM. For the purpose of this project this is by far the best available free GPU source and has been extensively used.

3.1.4 Tensorflow

In this project we use the open source software library, frequently used for differentiable programming and machine learning applications such as building artificial neural networks, namely Tensorflow. We use the current Tensorflow version in Google Colab - 1.15. This version is compatible with the Tesla K80 GPU and CPU Intel(R) Xeon(R) CPU @ 2.30GHz that is provided with the Jupyter Notebook.

3.1.5 Keras

Keras is a robust and user-friendly open source library that can be implemented with a Tensorflow backend. Engineered for faster computations, Keras was designed specifically for building Neural Network architectures as part of a research project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) in Google. We use Keras version 2.2.5 in the project to initialize and implement models and methods for constructing the desired Artificial Neural Network (ANN).

3.2 Spectrograms

One of the most popular tool amongst linguists or anyone interested in researching audio signals, is the spectrogram. It represents a 2-dimensional plot with time in the x-axis and frequency on the y-axis along with the amplitude as a colored heat map or a scale of color saturation. It displays a spectrum of frequencies and how they vary with time

[2]https://www.python.org/

for a given audio file. The third dimension - colored heat map indicates the amplitude of a particular frequency at a particular point in time. As mentioned earlier we use spectrograms of both English and Swedish audio clips for training our model. Figure 2 and 3 are examples of mel-spectrograms for an English-Swedish audio pair.

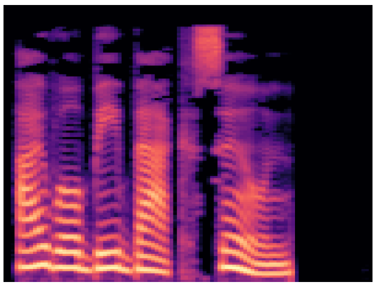


Figure 2: Spectrogram for English audio - "I will be back soon"

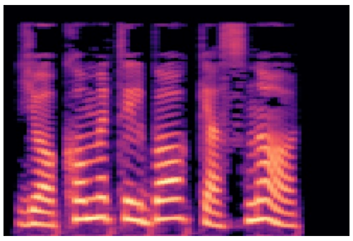


Figure 3: Spectrogram for Swedish audio - "Jag Kommer Tillbaka Snart"

Generation of Spectrograms before the advent of digital processing, were carried out either by approximating as a filterbank from a series of band pass filters or calculated using the fourier transform. Both these methods make use of time domain signals which is the case of our raw audio input files. This generation process uses Fast Fourier Transform (FFT) which is a digital process.

Fourier Transform

The audio signal that we input to the network is in the time domain. Although invisible to the naked eye in the time signal plots, that very signal is made up of various different frequency signals which is broken into it's constituent frequencies. These alternate representations are nothing but a combination of sinusoidal functions. The term 'Fourier' comes from the Fourier series named after the French Mathematican Joseph Fourier. The series breaks down any periodic function into sinusoidal functions. In our case the time domain signal is transformed into the frequency domain which is used in generating the spectrogram as can be seen in the figures 2 and 3.We use the librosa feature 'melspectrogram' for generating spectrogram magnitudes and map it to the mel scale. Mel scale is a result of non-linear transformations on the frequency scale. In simple terms this scale helps us identify a sound in a particular interval even if that interval for instance in Hz scale, would be inaudible. For example, difference between 500 and 1000Hz can be clearly differentiated but between 7500 and 8000 is not humanly possibly to identify individually.

This scale partitions the frequency (Hz) scale into bins and then transforms each bin into corresponding bin in Mel scale. Finally, in order to get the spectrogram reconstruction from the network output we use Griffin-Lim algorithm from the popular Librosa library for python.

Griffin-Lim

One of the popular ways of spectrogram reconstruction is the Griffin Lim algorithm. For the purpose of this project we have used an inbuilt Griffin Lim function from Librosa.

In our project we perform certain alterations to the spectrograms generated from the training audio samples. These modifications are to avoid the activation functions in the output layer from blowing up which has been explained in detail in the 'Data Representation' section. Now, due to the said modifications only the magnitude part of the output spectrogram remain pristine which is in the frequency domain. The phase of the amplitude signal is lost. This is required to get back the signal in time domain. The Griffin-Lim proposes to reconstruct this signal using Short-Time Fourier Transform (STFT). Although the output signal of the Griffin-Lim algorithm is of low quality due to lack of prior knowledge of the target signal, it serves the purpose of this project.

With reference to the project pipeline, Figure 1, the Griffin Lim algorithm is implemented on the output spectrogram of the main model architecture. This is the second last step of the whole pipeline which gives us back the estimated signal in time domain.

3.3 Neural Networks

Composed of densely inter-connected nodes a neural network is the most simplistic replica of a human brain. Consisting of roundabout 100 billion minute cells called 'neurons' with a number of connections coming off of it called dendrites, the human brain is the inspiration behind the branch of Artificial Intelligence known as 'Machine Learning'. In the most simplistic level, a comparison between a biological and artificial neuron can be seen in figure 4 .

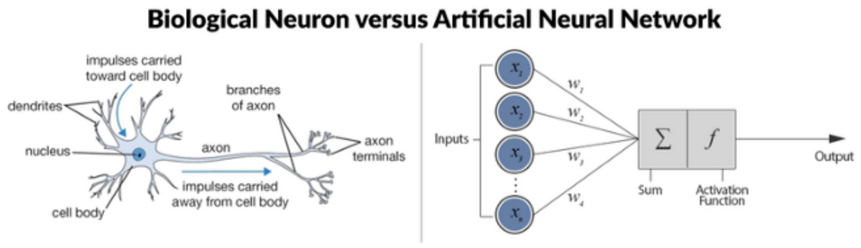


Figure 4: A basic neuron

As displayed in the figure 4 [3] above, the three basic parts of the neurons are : Input layer, hidden layer and the output layer. The input layer takes in the training data fed into the model, multiplied by the weights individually and summed together the resulting data goes through the hidden layer. After processing through the activation function we get the result at the output layer.

[3] Source : https://www.datacamp.com/community/tutorials/deep-learning-python

Activation function

An activation function is a key part of an artificial neuron as it decides how the output of the neuron should be in comparison to the input. One of the more common activation functions is called the *Sigmoid* function, which will is a part of LSTM unit as we will discuss later. This function will map the input to the range of (0,1) using equation 1.

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

Another activation function that is present in an LSTM unit is the hyperbolic tangent, *tanh* that maps the input to the range of (-1,1), governed by equation 2.

$$\varphi(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

Furthermore, the activation function used for the final output sequence of the network is **ReLU** stands for Rectified Linear unit which is the activation function we use in the output layer of the model network. Activation functions are responsible for reshaping the final summed weighted output of the node values of the final layer. In case of ReLU, the final output value is the value of the node input itself for positive values and zero for every other value. The function is a piecewise linear function with a simple modification. Following is the formula and graph of a ReLU function:-

$$y = x; x > 0$$

$$y = 0; x <= 0$$

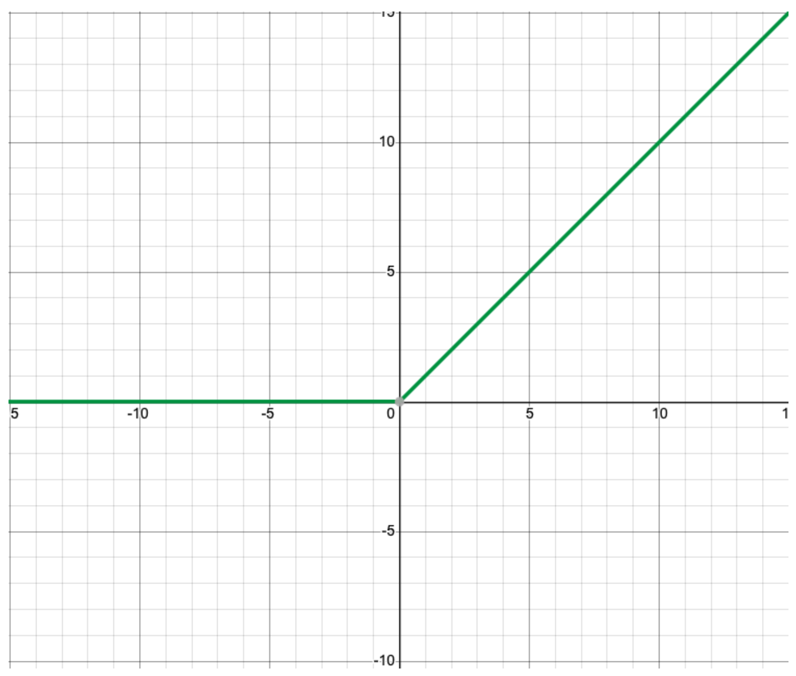


Figure 5: Rectified Linear Unit

3.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are specialized neural networks that contain loops in them which keeps moving the information in a cycle. This is beneficial as even humans when reading a word or a sentence for the first time, do not begin to think from ground up. Every impression or idea we get while reading is based on previous knowledge of words or sentences. Standard neural networks are unable to achieve this as they have no memory of previous data as they process the new data. Figure 5 [4] below describes a basic RNN where an input **Xt** goes through the network and an output **Ht** comes out.

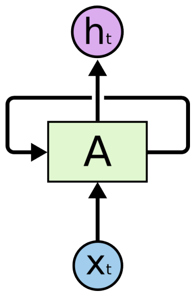


Figure 6: A basic RNN structure

In simpler terms if the loop is unrolled to an equivalent chain-like structure the network resembles a regular neural network akin to sequences and lists. This structure popularly has been used for speech recognition [1], language translation and other linguistic models. An RNN could be considered as a series of similar units through which message or input is passed through.

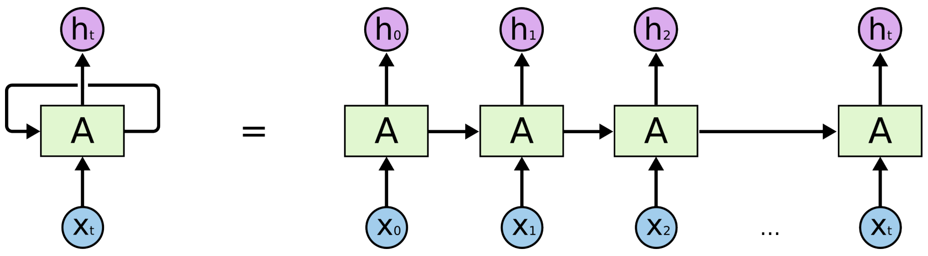


Figure 7: RNN structure

Figure 6[5] is the unrolled version of figure 5. Here the **Xt** where t is 1 to n where n is the number of timesteps in the given input sequence. As can be seen from the figure an RNN makes use of previous information to compute the future steps. This can be very useful in situations where information from the immediate previous time step input is to be considered for computing the next. But as this relevant information is found further back the more difficult it becomes for the Rectified network to map the required input to the next time step. This is where Long Short Term Memory (LSTM) networks come into play.

[4]https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[5]https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.3.2 LSTM Networks

Long Short Term Memory (LSTM) networks was again inspired by the way humans understand a sequence of words in a sentence. Simply put as we read a new sentence, in our minds we are backed up by the knowledge of everything we have ever read. We never read individual words as a separate unit but rather in a well connected series of words that it precedes. LSTM networks have the ability to use the further previous values of a sequence in a similar manner. Vanilla neural networks on the other hand do not posses this feature. For instance, in order to predict a scene in the middle of a movie, all the previous scenes in the plot would have to be taken into consideration. Whereas a regular neural network would consider each scene as a separate entity and process it accordingly to come up with a prediction. Naturally that will result in an error.

LSTMs as mentioned in the previous section were designed to avoid long term dependency issues faced by RNNs.

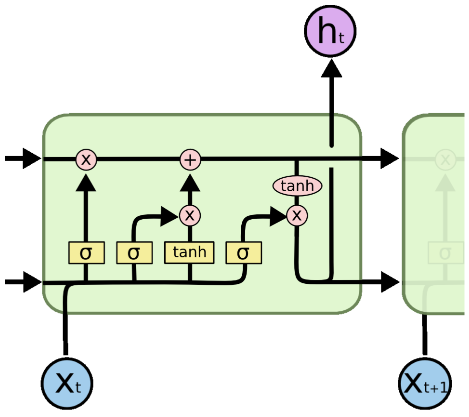
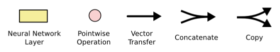


Figure 8: LSTM structure



The LSTM structure depicted in figure 7[6] explains the movement of information through the network. The horizontal line running on the top constitutes the cell state. This is a chain through which information can pass unchanged unless modified or added by the regulated structures called gates which is made up of a neural network with a sigmoid activation function followed by a pointwise multiplication operation. In total there are three such gates in an LSTM to forget, add or change information in the cell state.

The first step is to decide what to forget from the cell state chain. This is decided by the output of a sigmoid function which lies between 0 and 1 where '0' signifies completely forget the input already in the cell state chain and 1 means forget nothing. The second step is a combination of two sub steps which basically decide what is to be stored in the cell state. Firstly a 'sigmoid' layer decides which values are to changed, this layer is called 'input gate layer'. Secondly a 'tanh' layer comes up with potential values that could be used to add to the cell state chain. These steps are followed by the final

[6]https://colah.github.io/posts/2015-08-Understanding-LSTMs/

update implementation to the cell state. Then the final output is decided. This process again includes two small steps, first step is to execute a sigmoid layer that decides what parts of the cell state are to be sent in the output and the second step runs the cell state through a tanh layer to output the parts that were decided by the sigmoid layer.

As a modification over a uni-directional LSTM that goes from past to the future while taking the input vector, a new form of LSTM that inputs two forms of the same input, both past to future and future to past is known to have better results in speech recognition and translation [16]. These are known as BLSTMs or Bi-Directional LSTM networks which is discussed in the following section.

3.3.3 Bi-Directional LSTM Networks

In a research to improve over normal LSTMs [7], reversing a duplicate of the input sequence and then providing it with the original sequence side by side improved the performance of the model. Although not all domains see improvement, the initial popularity of Bi-Directional LSTMs (BLSTMs) was gained by speech recognition where the existence of a known context made the whole sentence more understandable or in our case translatable. In simple terms humans also are able to guess an otherwise meaningless sentence in the beginning unless a future context is already known. It has to be noted that the timesteps are conveyed one at a time, they are done so in both directions together in case of BLSTMs.

This helps BLSTMs understand context better while computing the output vector compared to unidirectional LSTMs. This feature has been extensively exploited to train models to convert sequences from one language or domain to another known as sequence to sequence learning.

3.4 Sequence to sequence learning

Introduced by Google in 2014, the encoder decoder network was designed to map a fixed length input 'X' to a fixed length output 'Y' where X and Y may not necessarily be equal. The paper on Sequence to Sequence architecture [17] was one of the first research to prove the possibility of end-to-end translation using deep neural networks. As described by Sutskever et al, they take on the task of text to text translation from English to French. The implementation of their LSTM based Encoder - Decoder setup gave remarkable results. The other notable result was the improvement in the translation by reversing the source input. This was associated with the introduction of various short term dependencies between the source and the target sentences.

For the purpose of our project we introduce the sequence to sequence neural network architecture similar to the voice conversion model that recreates a spoken utterance pair in another person's voice. We modify this model to make it simpler, understandable and more suitable to our needs. The sequence to sequence networks together with a stacked BLSTM acting as an 'encoder' and another stacked LSTM as a 'decoder' is a popular structure that has been implemented in the 'translatotron' research as well as this project. This system of networks is better known as Encoder Decoder Networks.

3.5 Encoder-Decoder Networks

The Encoder-Decoder network comprises of recurrent units wherein each unit accepts the input at a certain time step and propagates it further. This collected information from the input is further sent to the hidden layer. This hidden layer is a compressed representation of each row of training data which is finally decoded by the decoder. The Decoder is again a set of recurrent units that predicts the output at each time step.

The encoder decoder architecture is used as the standard method for automatic text-to-text machine translation and is the core part of Google's translation services available today. The major benefit of this structure lies in the ability to train models with variable input and output lengths.

Although this architecture is widely used for text-to-text translation models, it has also gained attention in the field of audio modelling applications. Following the 'Translatotron' paper, we begin with a basic implementation of an encoder decoder model for direct audio translations and improve it further.

Figure 7 explains a simple Encoder Decoder Network with three recurrent units in the encoder part and two in the decoder.

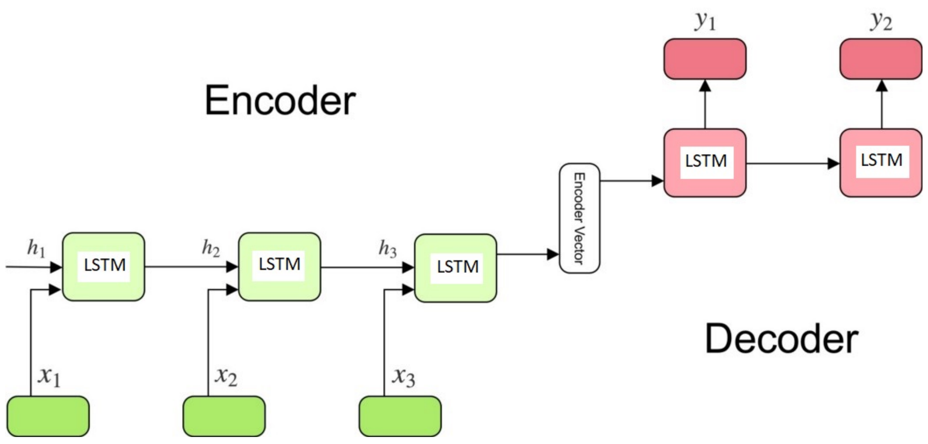


Figure 9: A basic Encoder Decoder Network using LSTM

The formula for the used for computing the hidden states from the encoder are as follows :-

$$h_t = f(W^{hh}h_{t-1} + W^{hx}x_t)$$

This is a simple multiplication of the weight vector to previous hidden state added to the input vector. As for the decoder, following is the formula used computing the hidden state:-

$$h_t = f(W^{hh}h_{t-1})$$

The final output if no specific activation function is mentioned is computed using the following formula:-

$$y_t = Softmax(W^s h_t)$$

This implies the output is nothing but the multiplication matrix of the weight matrix with the hidden vector of the current time step.

One of the most important features of this setup is the 'attention mechanism' that lies between the encoder and the decoder networks which is known to improve the efficiency of the whole mechanism [9]

3.6 Attention Mechanism

The classic encoder-decoder structure used in this project enables the model to encode the source utterance into a fixed length vector. This vector is used by the decoder to predict the output sequence. As the whole prediction process depends on this fixed vector, it is considered to be the bottleneck of the pipeline. The final hidden state of the encoder needs to encapsulate all the important features of the input sequence in order for the algorithm to work. Clearly, this was not good enough in quite a few cases, especially when the input sequence was long. The mis-alignment of the source and target inputs lead to a lot of errors in the resulting prediction. To fix this, Bahdanau in 2015, suggested the use of a context vector. This vector preserves the information from all the hidden states in the encoder cells and aligns them with the corresponding target outputs. This way the algorithm learns to 'attend to' specific parts of the input sequence while mapping with the target output and understand the relationship between them better. This mechanism became popular with Natural Language Processing (NLP) and came to be known as attention mechanism.

The Translatotron research bases the multi-training model on attention mechanism. This is implemented both for the phoneme recognition and the encoder decoder prediction training. Due to lack of a phoneme base, we do not implement a multi-training model. Nevertheless, we implement a similar attention mechanism.

4 Overview and Plan of Action

Up until now we have explained the basic tools and concepts used in the main model network that will be described in the following sections. In this section we discuss the goals of the model and the way we try to achieve it subsequently. We begin with a brief study of the proposed architecture of the Translatotron (refer section 2.4) model as shown in figure 10 below. We then try to come up with a simpler version of the same, carry out experiments and finally study the results

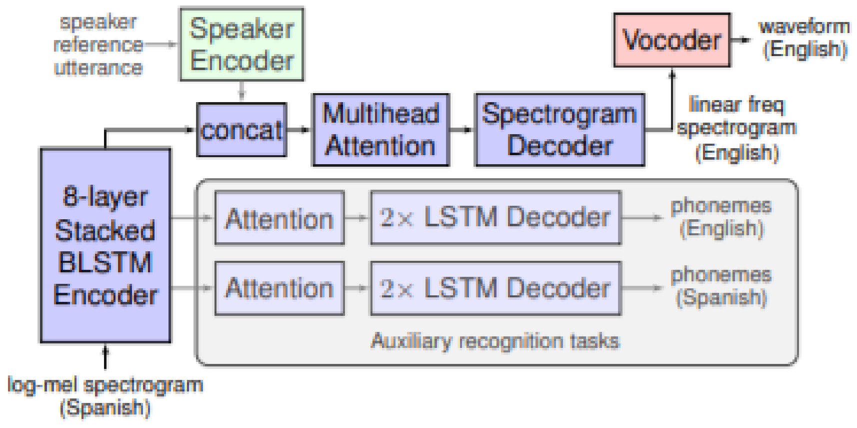


Figure 10: Model Architecture implemented in Google's Translatotron research

4.1 Translatotron Research

In this research, the main model aims to translate Spanish to English using log-mel spectrograms of the Spanish utterance pairs as input to the stacked BLSTM encoder. The boxes in dark purple constitute the main model network. The light purple box consisting of the 2 LSTM decoders with attention are the auxiliary tasks that has been added for better results which is proved later in the research. This makes the network a mutli-task trained model. Which means that the losses from these auxiliary tasks are also used along with the loss from the primary task from the whole network model. But the auxiliary losses are only used during the training phase to fortify the attention mechanism. They are not used during the final inference stage. It should be noted at this point that the exact type of loss function used has not been mentioned in the paper.

All the outputs from the encoder is concatenated and passed to the multi-head attention which in turn goes to the spectrogram decoder that computes a linear frequency spectrogram output. Finally this spectrogram output is sent to the Griffin Lim Vocoder which converts the target spectrogram into time domain waveform that can be heard. The cyan box on the top left containing the speaker encoder is a trained complex network that can imitate the voice of the speaker and maintain it's vocal characteristics. Clearly, this is out of the scope of this project as it also has not been built in this research.

4.2 Plan Of Action

As we begin to build the model from scratch we test different additions of tools and mechanisms in the model as go on improving it as per the research. In this section we propose an architecture similar to the one mentioned in section 4.2. With reference to the pipeline of this project as depicted in figure 1, in this section we explain the 'Blackbox' in further detail.

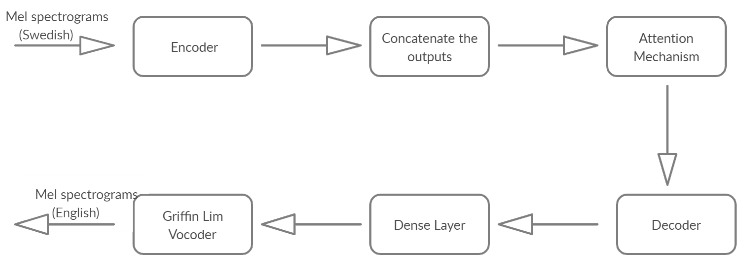


Figure 11: Blackbox of the project pipeline

As shown in figure 11 we intially build an encoder decoder setup. Both the encoder and decoder parts are built by stacking a few LSTMs and checking the result. We then replace this LSTM by a Bi-Directional LSTM and pass it through an attention mechanism. As we slowly add and implement different parts we try to create various models with growing improvements. The details of the parameters along with the number of LSTMs/BLSTMs are mentioned in the model flowchart in section 5.

5 Data Representation and Model Architecture

One of the challenging aspects of this project is the collection of training data. Compared to the training pairs required for text-to-text translation models, utterance pairs for this project are drastically difficult to procure. The Swedish language filter further narrows down our search to Swedish - English speech pairs rendering it impossible to find. Hence, the training utterance pairs were generated using Google Text-To-Speech (TTS) API calls on text pairs downloaded from Open Parallel Corpus - Opus website. These text pairs have been collected from various public sources such as parliamentary speeches, movie subtitles and book/novel translations.

For the simplicity of our model, we need short utterance pairs for which we divide the whole text paragraphs into separate sentence files and call the TTS python API individually to generate utterance pairs in both Swedish and English.

We use Spectrograms of short utterances in both English and Swedish to train our model. As the basic building block of our model are LSTMs we pre-process our data to the format that is required by an LSTM which is a 3-dimensional vector. The input shape is in the format - Number of samples, timesteps, features. Each mel-spectrogram of an utterance is a 2 dimensional vector with number of mel frequency bins in the y axis and time in the x-axis. We stack the spectrograms of all the data samples together for a 3 dimensional training dataset.

It is to be noted at this point, that we need to scale the data for the values to not be too far apart in magnitude which may blow up the activation function in the model resulting in 'NaN' values. For this purpose we use the logarithm function. Also, to avoid getting infinity for zero values we add a small buffer value in the range of 10ˆ-8. Furthermore, because the time axis has to be in the middle of the 3D matrix for the LSTM input format, we transpose the 2 dimensional spectrogram vector after applying the logarithmic function just before stacking.

While getting the audio back from the predicted mel-spectrogram we pass it through the exponential function before we use the Librosa feature 'inverse.mel_to_audio' to get the audio file which can be played using any python sound library.

Before we input the data into the main network model we 'mask' the sequences so that the input and output are of the same size, this is popularly known as masking.

Masking
Since majority of the calculations in neural networks are matrix computations, we need to ensure the input sequence in one language and the target sequence in another language are of the same length. The masking layer does exactly the same. In Keras platform, the masking layer is a class that can be invoked in the python code to mask the input sequence to any value. In our case, since we don't want the extra values to result in errors during the computations, we use zero values. For the purpose of the model training we use two data sources:

5.1 50 Languages website dataset

This website [7] provides 100 audio files in various languages including Swedish translated to English. Each audio file is about 3 minutes long containing phrases in Swedish spoken by a male and a female voice followed by the English translation in a male or

[7] https://www.50languages.com/phrasebook/en/sv/

female voice. The data has been prepared in such a way that each phrase is a separate utterance training data. The division of the original audio file into separate audio files has been executed using Pydub, a python library that can separate audio files using silence in between phrases.

5.2 Opus Parallel Text Corpus Website

The Opus website [8] provides various text sources and parallel translated text files for a lot of different languages including Swedish to English. For this project the 'Tatoeba' [18] Swedish to English translated texts have been used. For audio generation we invoke the Google Text-To-Speech API for both English and Swedish.

Figure 9 shows the input shape of an LSTM.

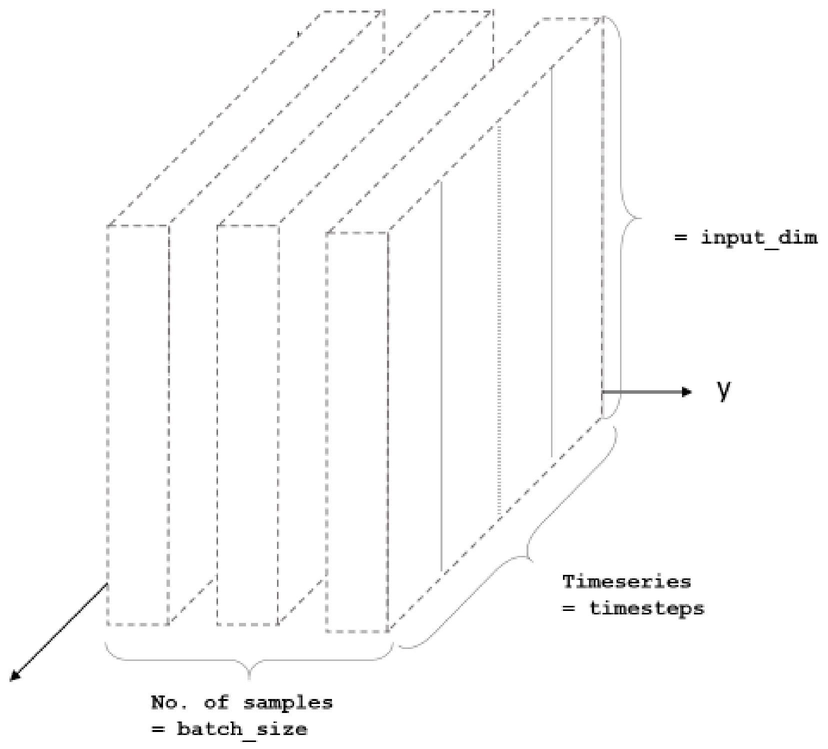


Figure 12: Input shape for LSTM

5.3 Model - Network Architecture

As mentioned earlier our model is a sequence to sequence encoder-decoder model. It is imperative to point out at this point that the project is a regression problem. Our main idea in a very simplistic form is to generate numbers from which we can get back spectrograms using the Griffin-Lim algorithm. These spectrograms ideally would be

[8]http://opus.nlpl.eu/Tatoeba.php

of the translated utterance. Using sound libraries these spectrograms can be converted back into audio.

The encoder decoder architecture is a two stage structure. The first part encodes the input data or sequence into a latent sequence or vector. This hidden vector is then decoded by the decoder for predicting the output sequence. This model architecture is specifically designed to tackle sequence to sequence problems. Further down we improve the model by implementing an attention model.

5.3.1 Encoder

The encoder part consists of 3 stacked Bi-directional LSTMs. The output sequences of these are stacked and sent further down to the Multi-Head Attention. We use a certain 'dropout rate' which in simple terms is the amount of neurons that have been ignored or dropped to optimize and figure out the exact neurons that actually help to improve the result.

5.3.2 Multi-Head Attention

The Multi-Head attention instead of calculating the attention once, it runs through the scaled dot-product attention several times in parallel. This is followed by concatenating the individual and independent outputs and linearly transforming them to fit the desired dimensions. In this case, this mechanism takes the concatenated output from the final layer of the Encoder BLSTMs, computes 4 different attention heads, concatenates the individual outputs from each, linearly transforms them to fit the input format of the first layer of the Decoder LSTM and passes it forward.

5.3.3 Decoder

The Decoder consists of 2 LSTMs with twice the amount of neurons used in the encoder as the BLSTMs convey a reversed duplicate of the input sequence down the pipeline alongwith the original input sequence. The idea of a decoder is to decode the hidden vector encoded by the encoder part and predict the output sequence. In our case input to the decoder is the aggregated output from the attention mechanism.

5.3.4 Dense output layer

The Dense layer is a simple well connected neural network. It is the final output layer which executes the following equation :-

$$output = activation(dot(input, kernel) + bias)$$

This takes the dot product between the input tensor and the weight kernel matrix which in our case is the output of the second LSTM of the decoder. Furthermore, in our case the optional bias is zero.

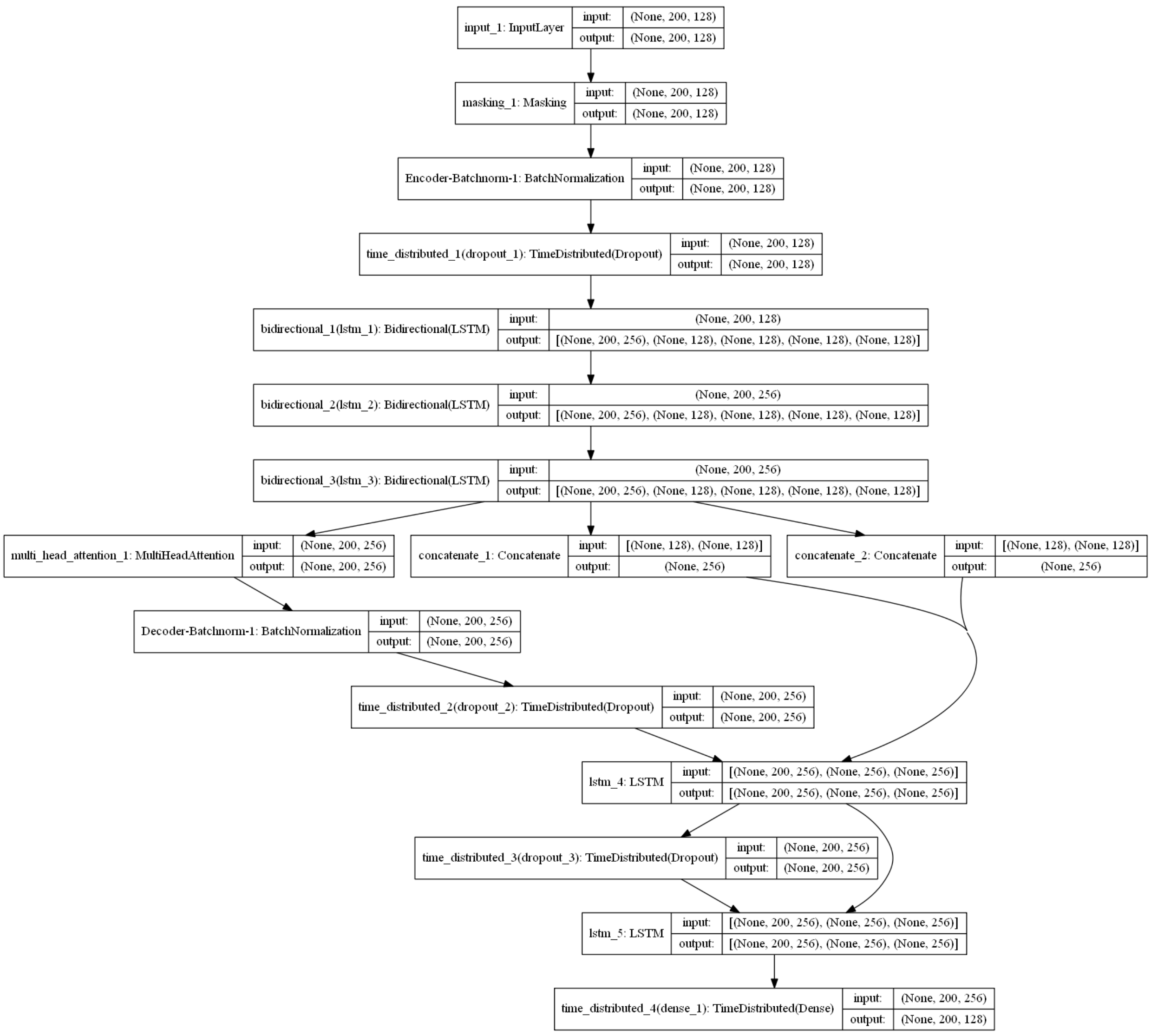
5.3.5 Loss function And Optimizer

The driving force for the iterations within an algorithm is the loss function. Simply put it tells the model how far the predictions are from the target values. Loss is used to update the gradients which in turn are used to update the weights within the neural network. For our model we use the most simple loss function available as a class in Keras used for regression problems, Mean Squared Error or MSE. The MSE is simply the average of the squared differences between the predicted and target values. The goal is to minimize this value by tweaking the weights. This is where optimizers come in.

The optimizer aims to associate the model and the loss function by updating the model in response to the result of the loss function. They are responsible for guiding the model to churn out as accurate results as possible. To serve our purpose we use the most common and powerful optimizer Adam.

5.3.6 Flowchart of the final model with parameters

Figure 10 below is the flowchart of the final improved model used for experiments and results.



input_1: InputLayer | input: (None, 200, 128) | output: (None, 200, 128)
masking_1: Masking | input: (None, 200, 128) | output: (None, 200, 128)
Encoder-Batchnorm-1: BatchNormalization | input: (None, 200, 128) | output: (None, 200, 128)
time_distributed_1(dropout_1): TimeDistributed(Dropout) | input: (None, 200, 128) | output: (None, 200, 128)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (None, 200, 128) | output: [(None, 200, 256), (None, 128), (None, 128), (None, 128), (None, 128)]
bidirectional_2(lstm_2): Bidirectional(LSTM) | input: (None, 200, 256) | output: [(None, 200, 256), (None, 128), (None, 128), (None, 128), (None, 128)]
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (None, 200, 256) | output: [(None, 200, 256), (None, 128), (None, 128), (None, 128), (None, 128)]
multi_head_attention_1: MultiHeadAttention | input: (None, 200, 256) | output: (None, 200, 256)
concatenate_1: Concatenate | input: [(None, 128), (None, 128)] | output: (None, 256)
concatenate_2: Concatenate | input: [(None, 128), (None, 128)] | output: (None, 256)
Decoder-Batchnorm-1: BatchNormalization | input: (None, 200, 256) | output: (None, 200, 256)
time_distributed_2(dropout_2): TimeDistributed(Dropout) | input: (None, 200, 256) | output: (None, 200, 256)
lstm_4: LSTM | input: [(None, 200, 256), (None, 256), (None, 256)] | output: [(None, 200, 256), (None, 256), (None, 256)]
time_distributed_3(dropout_3): TimeDistributed(Dropout) | input: (None, 200, 256) | output: (None, 200, 256)
lstm_5: LSTM | input: [(None, 200, 256), (None, 256), (None, 256)] | output: [(None, 200, 256), (None, 256), (None, 256)]
time_distributed_4(dense_1): TimeDistributed(Dense) | input: (None, 200, 256) | output: (None, 200, 128)

Figure 13: Flowchart of the final model

6 Experiments - Evolution of Network Pipeline

In this section the various models are described in the order of improvement with each addition or replacement with a better or new mechanism or tool. We use the Parametric method for defining our own model so as to suit our specific needs for the project.

6.1 Model - 1

This section describes the initial model as we try to build the whole pipeline from scratch. The structure of this initial model consists of a vanilla LSTM for both encoder and decoder parts. The output of the encoder goes through a repeat vector function so as to match the input format of the decoder LSTM. We use the 'ReLU' activation function and the 'Mean Squared Error' loss function since our problem is a regression problem. Figure 11 explains the parameters and flowchart for the first model

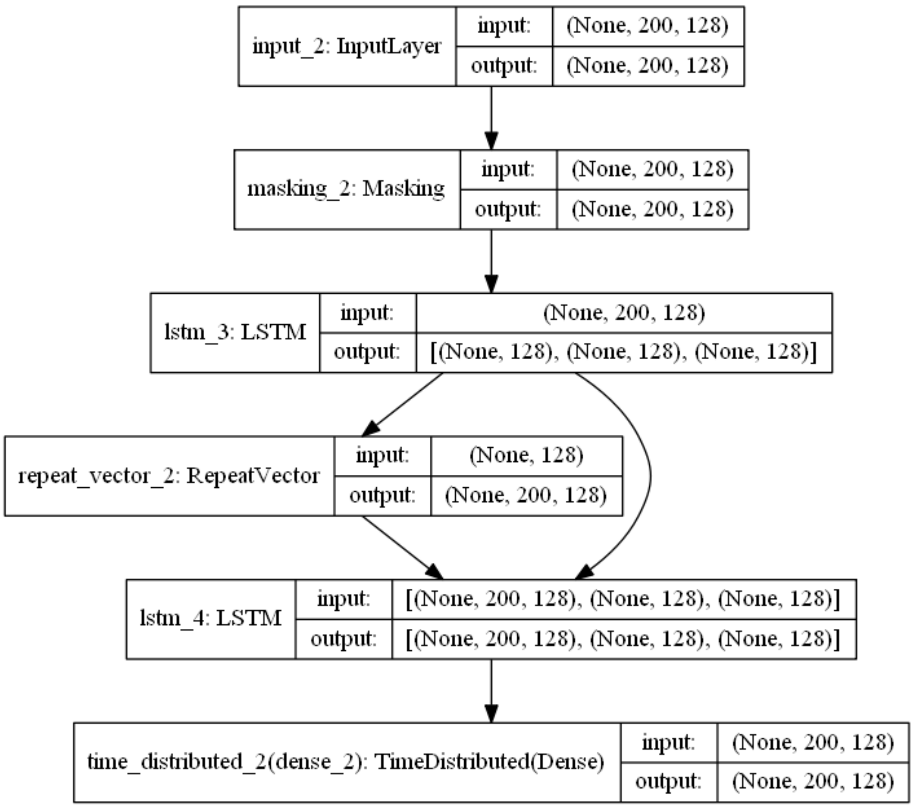


Figure 14: Flowchart of the final model

6.2 Model - 2

In this model we improve by replacing the LSTMs with Bi-Directional LSTMs. Known to improve sequence to sequence architectures, BLSTMs involve both the data from the past and the future together to develop better relationship between the source and target sequence.

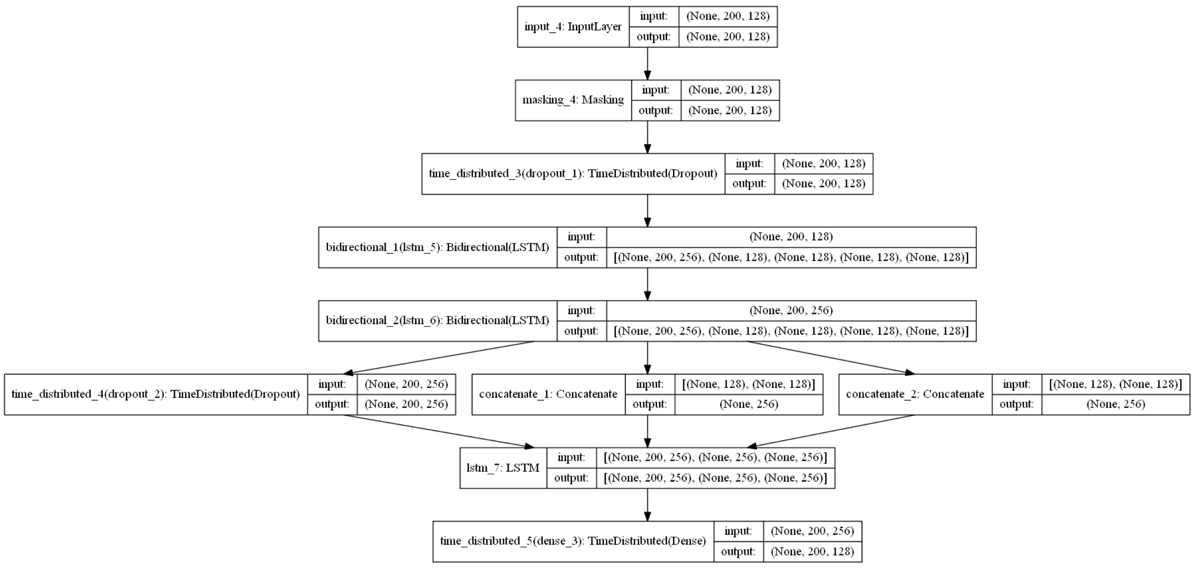

input_4: InputLayer
input: (None, 200, 128)
output: (None, 200, 128)
masking_4: Masking
input: (None, 200, 128)
output: (None, 200, 128)
time_distributed_3(dropout_1): TimeDistributed(Dropout)
input: (None, 200, 128)
output: (None, 200, 128)
bidirectional_1(lstm_5): Bidirectional(LSTM)
input: (None, 200, 128)
output: [(None, 200, 256), (None, 128), (None, 128), (None, 128), (None, 128)]
bidirectional_2(lstm_6): Bidirectional(LSTM)
input: (None, 200, 256)
output: [(None, 200, 256), (None, 128), (None, 128), (None, 128), (None, 128)]
time_distributed_4(dropout_2): TimeDistributed(Dropout)
input: (None, 200, 256)
output: (None, 200, 256)
concatenate_1: Concatenate
input: [(None, 128), (None, 128)]
output: (None, 256)
concatenate_2: Concatenate
input: [(None, 128), (None, 128)]
output: (None, 256)
lstm_7: LSTM
input: [(None, 200, 256), (None, 256), (None, 256)]
output: [(None, 200, 256), (None, 256), (None, 256)]
time_distributed_5(dense_3): TimeDistributed(Dense)
input: (None, 200, 256)
output: (None, 200, 128)

Figure 15: Flowchart of the final model

The model being and Encoder Decoder network, we keep both the 'Return Sequences' and 'Return State' as 'True'. We keep the loss function and activation function the same.

6.3 Model - 3

In the final model we add the multi-head attention mechanism between the encoder and decoder networks. As we see in the subsequent section the results of the final model show clear improvements from the other two models. Furthermore, to improve the decoder part we add another LSTM to the network. The flowchart of the the final model is figure 10 displayed in section 5.6.

7 Results And Discussion

The results of the experiments conducted on the three models explained in the above section have been broadly divided into two categories: The first one using the high quality human audio files taken from the '50 languages' website explained in the 'Data Representation' section. The second subsection describes the results procured using the 'Opus' parallel text corpus that have been converted to audio using Google Text-To-Speech API.

Furthermore, to see the improvement in the each of the progressing models we implement two ways of evaluation.

- First we use the method of physical hearing. Within this method we have two basic tests. These are described as follows:
  - **Test 1** : We check if the model can predict the translation of utterance file that was used to train the model in the first place
  - **Test 2** : We check if the model can predict the translation of Swedish utterance file after a part of the audio file has been edited by pasting the end part of another audio file. For instance, if audio file 1 says : 'Klockan är tolv' and audio file 2 : 'Klockan är elva', we edit the first audio file by removing the part of the audio signal that represents 'tolv' and replace it with 'elva' from the second file. We perform this editing using Audacity which is a free, open source audio editing software. This test is to check if the model has indeed learnt from the training data rather than memorizing.
- The second method of evaluation of models is by comparing the loss Vs epoch plots of each model keeping all the parameter values constant.

The plots shown in the subsequent sections are Loss Vs Epoch graphs for each model against the corresponding training data used. As mentioned in section 4.5, the loss function used for training the model is Mean Squared Error which as the name suggests is defined as the mean of the squared difference between the predicted and the actual value as shown below:

$$MeanSquaredError = mean((PredictedValue - ActualValue)^2)$$

Following are the parameter values used while training the models for both the training datasets:

- Timestep : 200
- Features : 128
- Latent Dimension of both the encoder and the decoder LSTMs : 128
- Dropout Rate : 0.2
- Batch Size : 100
- Epochs : 500
- Validation Split : 0.2 i.e. 20% of the whole training dataset will be considered for validation. This makes 400 samples for training and 100 samples for validation.

7.1 50 Languages website data

The result of the **first model** indicated the possibility of a translation tool using direct speech spectrogram mapping. After training for 6hrs in Kaggle Kernel the model can translate utterances that have been used for training. At this point due to lack of training samples, the model is overfitting and rather than learning. This means it has memorised the target language translation rather than learning. This model passes the first test successfully but Test 2 results in white noise.

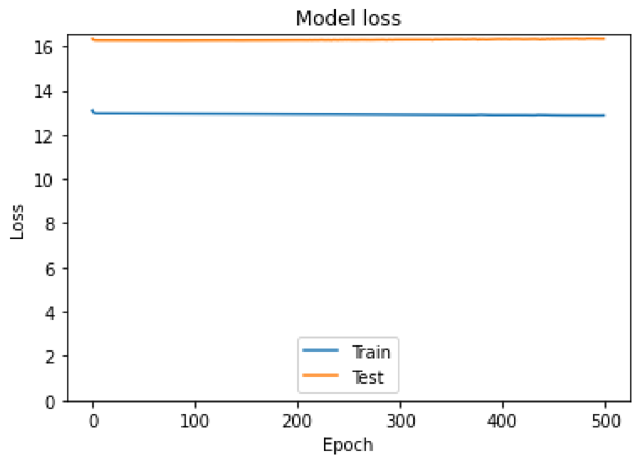


Figure 16: Loss Vs Epoch for Model II

As can be seen from the above plot, the validation loss is more than the training loss right from the beginning of the training. This can based on the aforementioned fact that the model is overfitting. Simply put, the model has memorised the mapping between the Swedish and the English utterances rather than learning. This makes it difficult for the model to generalize the translation when exposed to new data. Furthermore, with increasing epochs the overfitting increases as the model begins to memorise the training sample more and more, it begins to perform poorly with the new data.

For the **second model** the first test is successful. In the second test, the result although incomprehensible is not white noise but a mixture of human-like sounds. This can be accounted to the BLSTMs getting a little more information of the future of the input sequence and hence a better understanding of the target sequence mapping. But unfortunately not enough to exactly translate word by word within the utterance. Nevertheless, this can be concluded as an improvement over the first model

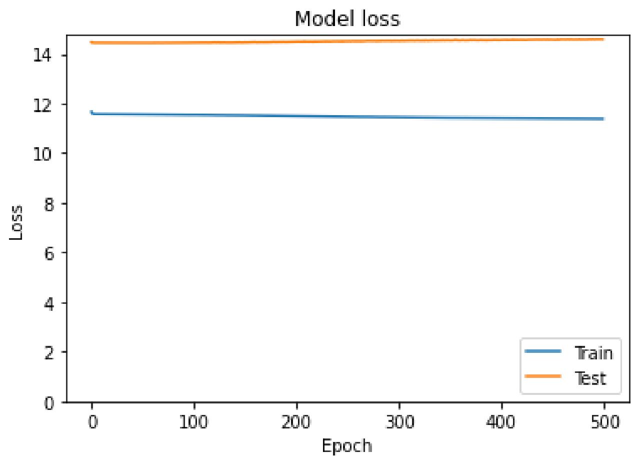


Figure 17: Loss Vs Epoch for Model II

We notice a similar plot as the first model. But, there seems to be a slight downward shift of loss for both train and validation plots. This indicates a furtherance from the first model.

The third and the final model with the attention mechanism overcomes this error by predicting the right translation for the editing part. It thus succeeds in both the tests. The aggregated result of all the four different attention heads help the model understand the relationship better between the target and source sequence mapping.

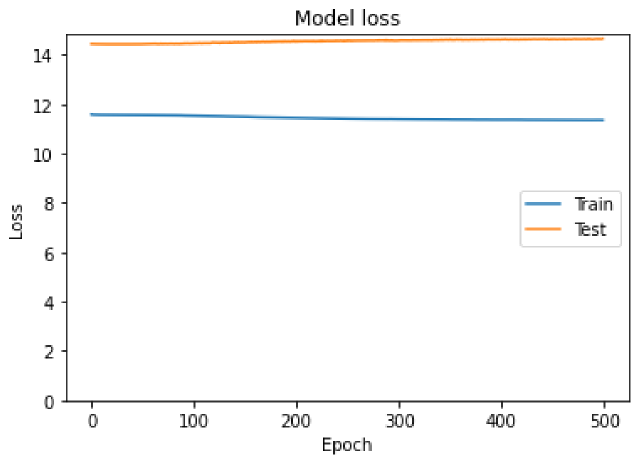
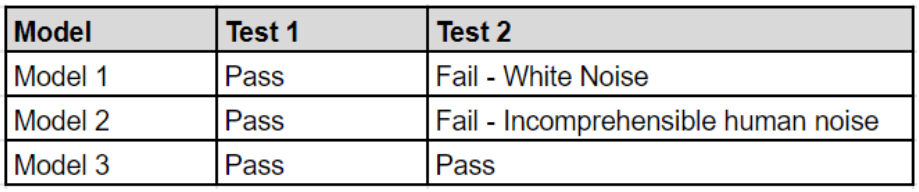


Figure 18: Loss Vs Epoch for Model III

There doesn't seem to be much change in the plots of model 2 and 3. The attention mechanism doesn't seem to be bringing a lot of change in loss compared to the previous model for this set of training data.

We summarise the result of the models according to their performance in different physical tests as follows:



Model	Test 1	Test 2
Model 1	Pass	Fail - White Noise
Model 2	Pass	Fail - Incomprehensible human noise
Model 3	Pass	Pass

Figure 19: Model Result Summary

To reiterate, model 3 passes the second test, which in simple terms means that it can predict an understandable English translation for a given sequence in Swedish from the training set even when the said source sequence was edited using an audio editing software. This might indicate that comparatively the model has overcome the overfitting to some extent and learnt the spectrogram mapping a little better than the other two models. Although it seems natural for the model to predict a value from a data that was used during it's training, in terms of sequential data this is not true as we study the results with the same models on the second set of training data.

It is to be noted again that the prediction sequences are of the utterances that have already been used while training of the model. The voice in the utterances are unique and limited for the model. As voice recognition is out of the scope of this project, we rely only on the limited data that has been procured for training.

7.2 Opus Parallel Corpus data

This section contains the results of the models when the synthesized Opus parallel corpus data was used for training. Although we use the same models, we find changes in the loss Vs Epoch plots due to changes made while pre-processing the data. The reason for making changes in the pre-processing for this particular data was guided by the error due to gradient explosion while using the ReLU activation function. This lead to the inability of the model to learn anymore after a certain epoch, reaching constant values of loss thereafter. For this very reason we replace the logarithmic scaling with normalization of the training data. Following is the simple formula used for normalization:

$$X' = (X - X_{min})/(X_{max} - X_{min})$$

For the **first model** we get the following Loss Vs Epoch plot:

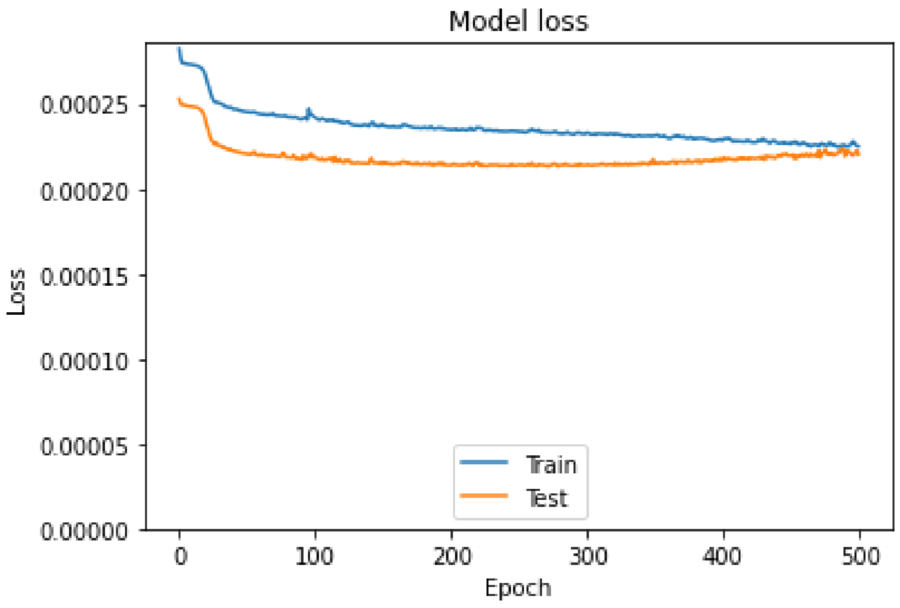


Figure 20: Model II

We immediately notice that the loss for the model has drastically decreased compared to the '50 languages training dataset'. This is because of the normalization of data which is localized between values 0 and 1. Furthermore, the inflection point i.e. the point where the validation and training plots meet is just around 500 epochs. Beyond this point, the model overfits.

For the **second model** we have the following Loss Vs Epoch plot:

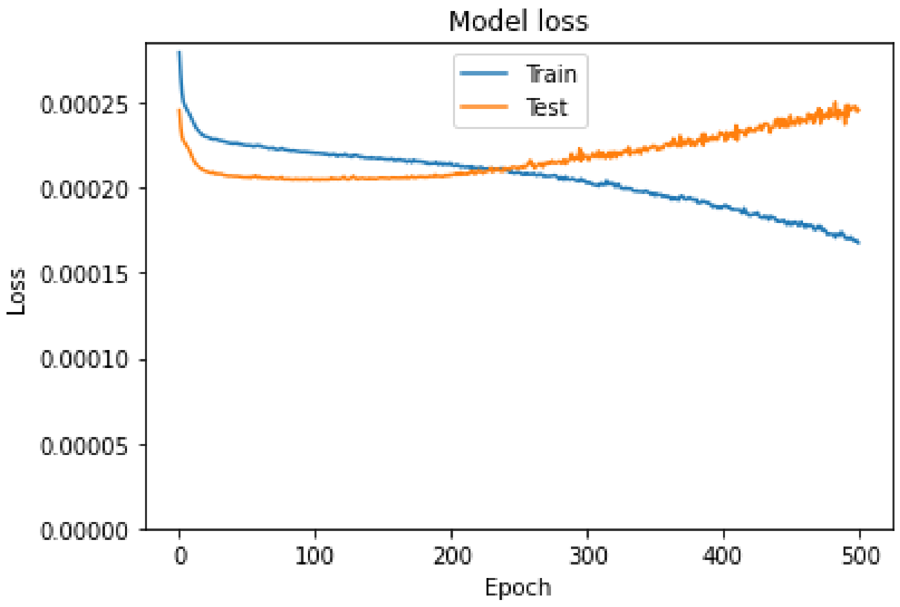


Figure 21: Model II

We notice that around 250 epochs we reach the inflection point after which epoch the model is overfitting. The solution to this overfitting would be increasing the number of training samples.

For the **third model** we get the following Loss Vs Epoch plot:

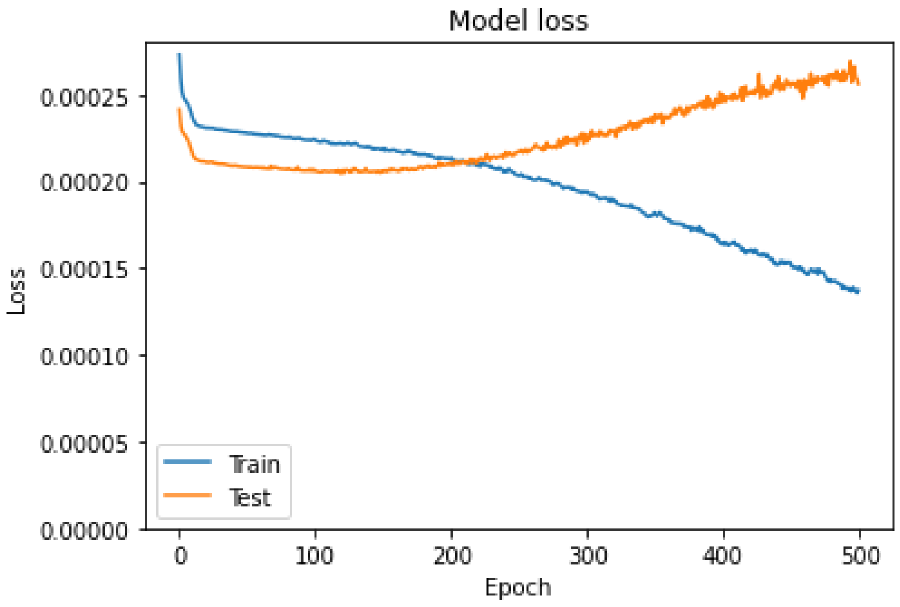


Figure 22: Model III

We notice a similar plot wherein the inflection was reached around 220 epochs. Beyond that point, the model again overfits. It should be noted here that the early model inflection with all other parameters constant indicates that this model indeed has improved over the previous two models.

For this dataset although the loss has drastically decreased compared to the previous dataset, the final output is still not comprehensible. All the three models fail both the physical hearing tests. To improve the results for this dataset we need to introduce complexity in the model network by increasing the BLSTM layers and further audio filters as per the 'Translatotron' research paper which at the time being are difficult to implement.

Lastly, in the predictions using Logarithmic scaling, the output translations have a trailing white noise which is nothing but random samples distributed at regular intervals with a zero mean and standard deviation of one. This white noise is absent in the case of normalized data. The parameter range of normalization is between 0 and 1 which probably causes the white noise to disappear.

8 Conclusion And Future Work

The results achieved by exploiting the complex nature of the encoder decoder network built up of the basic LSTM/BLSTM unit indicates the possibility of a direct translation algorithm even for a small scale architecture. With the right amount of training data and technical guidance concerning the bandfilter for the inputs in the decoder should create a vast difference in the results. Although time is not a metric we take into account in this project, replacing LSTMs with GRUs should decrease the computing time for the network. This will create a great deal of difference when processing large amounts of data.

As for future work, alongwith including the improvements mentioned above, a multi task training setup can be arranged. A list of phonemes both as an utterance and written text for both the source and target language would be required. As mentioned in the 'Translatotron' research, the auxiliary losses created a significant difference in the results.

References

[1] S. Benk, Y. Elmir, and A. Dennai, “A study on automatic speech recognition,” vol. 10, pp. 77–85, 08 2019.

[2] P. Brown, S. Della Pietra, V. Pietra, and R. Mercer, “The mathematics of statistical machine translation: Parameter estimation,” *Computational Linguistics*, vol. 19, pp. 263–311, 01 1993.

[3] A. W. Black, H. Zen, and K. Tokuda, “Statistical parametric speech synthesis,” in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP ’07*, vol. 4, pp. IV–1229–IV–1232, 2007.

[4] Y. Jia, R. J. Weiss, F. Biadsy, W. Macherey, M. Johnson, Z. Chen, and Y. Wu, “Direct speech-to-speech translation with a sequence-to-sequence model,” *CoRR*, vol. abs/1904.06037, 2019.

[5] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[6] J. Zhang, Z. Ling, L. Liu, Y. Jiang, and L. Dai, “Sequence-to-sequence acoustic modeling for voice conversion,” *CoRR*, vol. abs/1810.06865, 2018.

[7] Y. Jia, M. Johnson, W. Macherey, R. J. Weiss, Y. Cao, C. Chiu, N. Ari, S. Laurenzo, and Y. Wu, “Leveraging weakly supervised data to improve end-to-end speech-to-text translation,” *CoRR*, vol. abs/1811.02050, 2018.

[8] F. J. Och, C. Tillmann, and H. Ney, “Improved alignment models for statistical machine translation,” in *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[9] T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 1412–1421, Association for Computational Linguistics, Sept. 2015.

[10] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst, “Moses: Open source toolkit for statistical machine translation,” in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, (Prague, Czech Republic), pp. 177–180, Association for Computational Linguistics, June 2007.

[11] J. Guo, X. Tan, D. He, T. Qin, L. Xu, and T.-Y. Liu, “Non-autoregressive neural machine translation with enhanced decoder input,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, p. 3723–3730, 2019.

[12] A. L. D. K. C. C.-B. S. K. Matt Post, Gaurav Kumar†, “Improved speech-to-text translation with the fisher and callhome spanish–english speech translation corpus,” *Human Language Technology Center of Excellence, Johns Hopkins University † Center for Language and Speech Processing, Johns Hopkins University*, 2013.

[13] Y. Jia, M. Johnson, W. Macherey, R. J. Weiss, Y. Cao, C.-C. Chiu, N. Ari, S. Laurenzo, and Y. Wu, "Leveraging weakly supervised data to improve end-to-end speech-to-text translation," 2018.

[14] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," 2017.

[15] F. Biadsy, R. J. Weiss, P. J. Moreno, D. Kanevsky, and Y. Jia, "Parrotron: An end-to-end speech-to-speech conversion model and its applications to hearing-impaired speech and speech separation," 2019.

[16] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, K. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," *CoRR*, vol. abs/1712.01769, 2017.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.

[18] J. Tiedemann, "Parallel data, tools and interfaces in OPUS," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, (Istanbul, Turkey), pp. 2214–2218, European Language Resources Association (ELRA), May 2012.