Universiteit Leiden
The Netherlands

Computer Science

Bootstrapping Speech Synthesis Training

Stijn Boere

Supervisors:
Erwin M. Bakker & Michael S. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26-Sep-2022

Abstract

Training end-to-end speech synthesis models requires large amounts of data, which might not be readily available. Several methods exist to alleviate this problem but often require a resource rich environment of some sort. Our approach will consist of training the speech synthesis model on a small data set in addition to self-selected synthesized data obtained using a bootstrapping method. Using ASR on the synthesized data we obtain transcripts. By comparing these transcripts of the synthesized data to the input of Tacotron2, we can keep track of the correct and incorrect occurrences of each word. These words are then put into different data sets based on the percentage of correct occurrences and used to generate new sentences. Using these sentences in training will result in the iterative training and improvement of our speech synthesis model. The proposed method is evaluated in terms of both simulated mean-opinion-score and word-error-rate and compared to a low- and high-resource baseline. Our method achieves an improvement in simulated mean-opinion-score over both baselines, whereas an improvement in word-error-rate over the low-resource baseline is achieved. Furthermore, we show the possibility for our method to achieve an improvement in terms of word-error-rate over the high-resource baseline.

Contents

1 Introduction

With the ever increasing amount of uses for speech synthesis, the need for data used to train these speech synthesis models is also growing. Some specific uses range from helping the blind, deafened or vocally impaired to even educational applications [BS09]. More general applications allow for the usage of speech synthesis software in for example navigation software, or telephone services [GN07]. This shows how widely applicable speech synthesis is: from reading sentences out loud for those who are not able to read themselves, to helping a student on the pronunciation of words from different languages, it has become an essential part of many a person's life.

Speech generated by speech synthesis software is evaluated on two main qualities: naturalness and intelligibility [Tay09]. Naturalness describes how closely the generated speech resembles human speech based on sound, whereas intelligibility describes the ease of understanding the generated speech. Traditionally and by extent the most basic forms of speech synthesis consist of concatenating prerecorded fragments of speech such as phonemes in order to produce spoken words or sentences. Smaller prerecorded speech fragments result in a lower quality of the resulting speech but an increase in flexibility. This concatenation of smaller fragments often scores low on the naturalness given that this method does not take syllables and punctuation marks into account.

Fast forward to the current day and the most prominent models are deep learning based. A method within deep learning is the use of end-to-end models. These models excel at their ease of use: usage of these models can be done with minimal manual intervention and background knowledge of linguistics [NM21]. While these models can significantly outperform the concatenative models on both naturalness and intelligibility, they require vast amounts of training data to achieve these ideal results. Where concatenative models only require a minimal amount of data, say a few audio files containing all the phonemes of the specific language, these end-to-end models require large data sets containing upwards of 24 hours of spoken text. One can imagine that it takes a large amount of time and effort to collect such large amounts of data, let alone create the data set from scratch. This is why we asked the question whether it was possible to create a method for which a speech synthesizer can still thrive in cases where we do not have such a large data set available. Is it possible to bridge the gap between a speech synthesizer trained on a smaller data set and a speech synthesizer trained on a large data set?

Our aim is to create a bootstrapping scheme designed to handle these low resource scenarios, producing a speech synthesizer trained on a smaller data set with comparable results to a speech synthesizer trained on a larger data set. This paper studies such a bootstrapping scheme using Tacotron2, a speech synthesizer which we will train during the execution of our bootstrapping scheme and DeepSpeech, a pre-trained speech recognizer used to evaluate the

results at each inference of our speech synthesizer. Our speech synthesizer will be trained and retrained multiple times, each time producing its own data. This data is then evaluated by the speech recognizer and the results deemed worthy are added to the original training data set which in turn is used to retrain our speech synthesizer. The final result should be a trained speech synthesizer with similar performance to a traditionally trained TTS (text-to-speech) engine in addition to a augmented training data set.

The rest of the paper is organized as follows: Section 2 contains the related research as well as the contributions of this paper. Section 3 discusses fundamental terms needed to understand this paper while section 4 gives an overview of the data set used. In section 5 we evaluate the baselines used to measure the success of our method. Section 6 describes the method used to conduct the experiments, which are described in section 7. The results of these experiments are then reported in section 8 followed by a discussion of these results in section 9. Finally the paper concludes in sections 10 and 11 by discussing the research and future improvements.

2 Related research

This section describes relevant topics to both this paper and our research. These topics include relevant research done in the field of TTS, ASR, approaches taken in other projects and how they compare to ours, as well as a descriptions of our deliverables.

TTS

Prior work mostly uses the Tacotron2 speech synthesizer. As mentioned by Shen et al. [She+18], their implementation of Tacotron2 achieves superior results in terms of MOS to the competition with even comparable results to the ground truth audio. Another prominent model is Fastspeech2 [Ren+20] which seeks to improve the shortcomings of the Fastspeech model. While the main feature of Fastspeech is it being significantly faster at synthesizing than previous autoregressive models while still having comparable quality, it lacked in accuracy and contained very time consuming factors on other fronts. While Fastspeech2 achieves faster and more accurate results than Fastspeech, it still consists of a far more complicated pipeline than existing end-to-end TTS models.

Ge et al. [Ge+21] have researched a method to improve on the training time of TTS engines. This improvement can be achieved using their batching strategy of semi-sorting the samples. While this strategy can show up to 40% improvement in training time, it comes at the drawback of a reduction in randomness in the samples. The reduction in model performance due to the randomness of the batching strategy used might be negligible when compared to improvement in training time. In our case however, we have opted not to use this strategy given that we start in a low resource environment, thus needing all the performance we

can get while on top of that having a comparatively lower training time than models being trained on larger data sets.

ASR

Giollo et al. [Gio+21] have approached the bootstrapping method from the ASR (automatic-speech-recognizer) perspective showing promising results. Although their "post-ASR text-to-text mapping and synthetic data generated with TTS" does utilize multiple components, it clearly shows the benefit of using a TTS engine to generate synthetic data in their ASR bootstrap. By employing all methods they achieve an improvement of 46% in WER (word-error-rate) over the monolingual base. Other research by Fazel et al. also [Faz+21] utilizes synthetic data in their model. By using their combination of continual learning with their proposed multi-stage training, they achieved a relative improvement in WER of more than 65%, further demonstrating the potential that can be achieved using synthetic data.

Low-Resource TTS

Prior work investigated the effectiveness of data augmentation in low resource environments [Sha+20]. While these models mostly focus on improving the ASR whereas we focus on improving the TTS engine, they do show promising results in using synthesized data for training.

Several approaches have been taken in dealing with low resource environments. One such method investigates the effectiveness of training both ASR and TTS on data similar to the target data. Xu et al. [Xu+20] used data of rich-resource languages to pre-train both their ASR and TTS to benefit these models when learning on low-resource languages. Their LJSpeech model shows promising results for TTS with a more than 98% intelligibility rate and above 3.5 MOS (mean-opinion-score) all while using very little resources for their target language.

While these methods mostly rely on trans-lingual transfer learning to obtain their results, we take a different approach. The idea behind trans-lingual transfer learning is to transfer the results obtained by training the TTS on a rich-resource language to the low resource language. Since our goal is to obtain our results without an additional rich-resource language, we shift our focus to another approach, namely the bootstrapping method.

Contributions

This project delivers our generalized bootstrapping strategy as defined in the Method section of this paper, as well as the specific bootstrapping method we used as defined in the same section. In short, this method can be described as follows: The idea behind the

bootstrapping is to improve our speech synthesizer in iterations. We first train Tacotron2 on a smaller data set. After this we produce our own data, which will be evaluated by DeepSpeech. Individual words deemed fit by our selection (bootstrap) method are used to generate sentences. These sentences are added to the training data set and used to retrain Tacotron2. The desired result of each iteration is an increase in the training data set with an improved trained Tacotron2 model as a result.

3 Fundamentals

In this section we provide a quick summary of important subjects that are used in the rest of this paper. These subjects include mean-opinion-score, word-error-rate and automatic-speech-recognizers.

Mean-opinion-score

Since we need to evaluate whether or not our bootstrapping model was a success, we will compare it to a baseline. This evaluation is done by passing both our model as well as the baseline the same data after which a MOS will be taken of the results. The MOS can range from 1 through 5 with a score of 1 being the worst whereas a score of 5 represents the best possible quality. MOS originated from a poll conducted on real humans using audio (or video) files. These test subjects would judge these files on their quality using the numerical measure. The resulting averaged number assigned to the file is what we call the MOS. These days, rather than using test subjects, we use several algorithms which attempt to approximate the human experience given a file [SWH16]. While traditional MOS would still be optimal, conducting such an experiment would require too much work making it a non feasible solution. We thus opt to use simulated MOS (sMOS) as provided by Lo et al. [Lo+19] as an alternative to traditional MOS even if the results might not be optimal. Once again we use a scale from 1 through 5 representing the quality of the file.

Word-error-rate

The word error rate (WER) is another metric of quality. Rather than denote the audio quality of files, the WER is a metric of quality often used for the performance of automatic speech recognizers (ASR) [Par+08]. Since we use ASR to evaluate our TTS results, we also evaluate our success on WER. The WER can be calculated using the following formula:

$$WER = \frac{S + I + D}{N} \tag{1}$$

Here S represents the substitutions made by the ASR. A substitution occurs when a word is replaced such as the replacement of the word "mail" with "hail". I represents the insertions made by the ASR. An example of an insertion is the transcription of the word "dancing"

as the words "dance sing". D represents the deletions made, which occur when a word is simply omitted. An example is the transcription of the words "let it go" to the words "let go". Finally N represents the number of words spoken. In summary, above the line we have the number of mistakes, whereas under the line we have the total number of words.

ASR

In combination with WER, we use ASR as a tool to validate the TTS results. In addition to this validation, we also use ASR to select new training data for our bootstrapping method. We have looked at multiple ASR models ranging from Kaldi[Pov+11], an older open-source ASR written in C++, to JASPER [Li+19], an end-to-end model achieving near state-of-the-art results on LibriSpeech. The Kaldi model being an older model and the JASPER model being too heavy for the aperture we used eventually landed us on the DeepSpeech model [Han+14], a self-proclaimed simpler open-sourced end-to-end model with simpler meaning that the engine does not require server-class hardware to run.

4 LJSpeech Data Set

While we synthesize our own data, we do need to initially train our model on a proper data set. LJSpeech 1.1 as provided by keithito [kei] is such a data set containing 13100 audio clips combined with the transcripts. LJSpeech is the most used data set in training Tacotron2 and other end-to-end neural-network-based TTS models. The 13100 audio clips are read by a single speaker reading from 7 books. Each clip is provided with a transcript and varies in length between 1 and 10 seconds making the entire data set approximately 24 hours in length. The baselines are both trained entirely on the LJSpeech data set, whereas the other models are trained in conjunction with our bootstrapping method. One of the baselines is trained on the entire LJSpeech data set and is thus denoted as **100p**. The other baseline is trained on a randomly selected 50% of audio files in the LJSpeech data set (training set) and is thus denoted as **50p**. This same 50% of the data set is used for training the initial iteration of the other models allowing us to make a fair comparison on the effect of our bootstrap between the baseline and our models trained using the bootstrap. The resulting 50% of the LJSpeech data set (test set) is used to test our results after having trained the Tacotron2 model.

5 Baseline

In this section we describe the baseline model to which our bootstrapped models are compared.

The goal of this project is to find out whether or not bridging the gap between a text-to-speech engine trained on a smaller data set and a text-to-speech engine trained on a larger data set is possible. To validate the results of our previously discussed bootstrapping scheme we ensure that both the text-to-speech engine (Tacotron2) and the data set (LJSpeech) are the same for above cases. The differences lie in the methods we use to conduct our research. As we conduct several experiments, we define a baseline for each experiment. These baselines range from a Tacotron2 model trained on the entire LJSpeech data set to a Tacotron2 model trained on a part of this data set. One thing these baselines have in common is that they are purely trained on the LJSpeech data set, whereas the models which we compare to these baselines are trained on LJSpeech in addition to the data generated by our bootstrapping method.

Tacotron2 boasts a impressive MOS of 4.53 with a MOS of 4.58 being professionally recorded speech. As described by Shen et al. [She+18] "the system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to mel-scale spectrograms, followed by a modified WaveNet model acting as a vocoder to synthesize timedomain waveforms from those spectrograms". While we train all Tacotron2 models ourselves, we use a pre-trained WaveNet model given that our hardware did not possess the capabilities to train it ourselves.

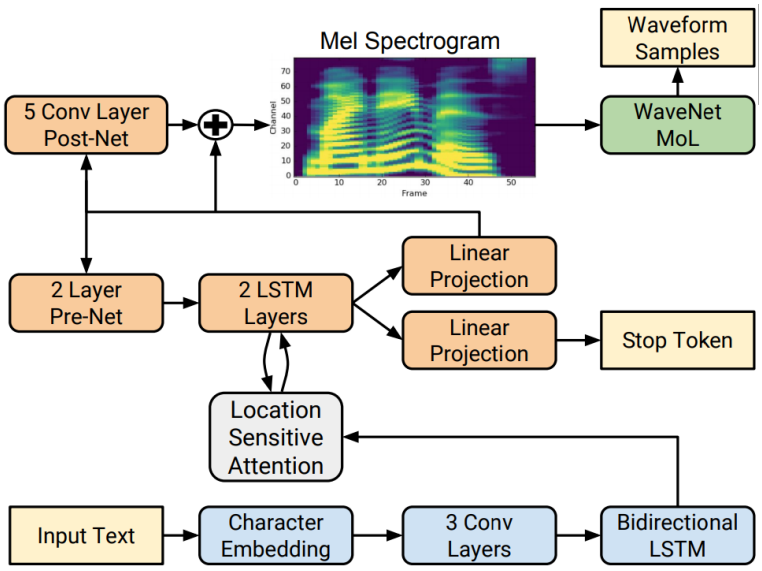

Figure 1: A graphical representation of the Tacotron2 architecture [She+18]

Above figure gives a global overview of the Tacotron2 architecture. Without going into too much detail, Tacotron2 consists of the combination of blocks which, given a input text,

produces a mel spectogram. This mel spectogram is then passed to the vocoder (WaveNet) which in turn produces a audio file from this mel spectogram.

6 Method

This section gives a general description of our bootstrapping scheme, as well as a description of the specific method we used for this research. A general iteration of our bootstrapping scheme is described by the following steps:

- Initialization: Split initial data set into Training Set Tr_i and Test Set Te consisting of audio + transcripts (a+t)
- Initialization: Train text-to-speech engine on the training data set Tr_i
- Start Iteration
- Data Selection: Synthesize a data set $Synth$ (a) using our trained model and the transcripts of test data set Te
- Data Selection: Generate transcripts of the synthesized data set $Synth$ using a speech recognizer
- Data Selection: Compare generated transcripts to the original transcripts of Te and write statistics to log file
- Data Synthesis: Using sentence generator, create new sentences using the words in the log file
- Data Synthesis: Synthesize the new sentences using the trained text-to-speech engine
- Data Synthesis: Add the newly generated files consisting of audio and transcripts to the training data set Tr_i
- Training: Continue training text-to-speech engine using Tr_{i+1}
- End Iteration

Pseudo-code

When using the specific models we used in our research, the described method can roughly be translated into the following pseudo-code:

Split LJSpeech into two parts: Training Set Tr_i and Test Set Te
Let *DeepSpeech* be the pretrained DeepSpeech model

Let *iterations* be the desired number of iterations
Let $logfile$ be the text file containing the statistics over the words
Let Sg be the sentence generator
Let M_0 be the initial Tacotron2 Model **(Initialization)**
Let M_i be the initial Tacotron2 Model trained with training set Tr_i
for i in iterations **do (Start Iteration)**
    $Synth = M_i(Te)$ **(Data Selection)**
    Convert the audio in $Synth$ to obtain the required sample rate
    $Text = DeepSpeech(Synth)$
    for *sentence* in $Text$ **do**
        for *word* in *sentence* **do**
            Check if *word* is in $Synth$[*sentence*]
            Write results to $logfile$
        end for
    end for
    for word w in $logfile$ **do (Data Synthesis)**
        if correct occurrences of $w \geq 80\%$ **then**
            Generate sentences using w and Sg
            CS_w = sentences generated using correct word w and Sg
            Add CS_w to data set CS
        else
            Generate sentences using w and Sg
            IS_w = sentences generated using incorrect word w and Sg
            Add IS_w to data set IS
        end if
        $SynthCS = M_i(CS)$
        $SynthIS = M_i(IS)$
        $Tr_{i+1} = Tr_i + SynthCS$ OR $Tr_{i+1} = Tr_i + SynthIS$
    end for
    Train M_i using Tr_{i+1} **(Training)**
    M_{i+1} = model M_i trained with training set Tr_{i+1}
end for(End Iteration)

In above pseudocode we have simplified several steps such as writing the results to the logfile as well as generating sentences. These steps are conducted as described in the above method. Finally in the last line we add either the 'correct' sentences or the 'incorrect' sentences to the new training data set. This further illustrates that both methods use the same starting point but part ways in the next iteration. Should the user choose to add the 'correct' sentences, we assume that the same choice shall be made in the coming iterations. The same holds for the 'incorrect' sentences.

The following image provides a graph displaying the specific bootstrapping scheme we used:

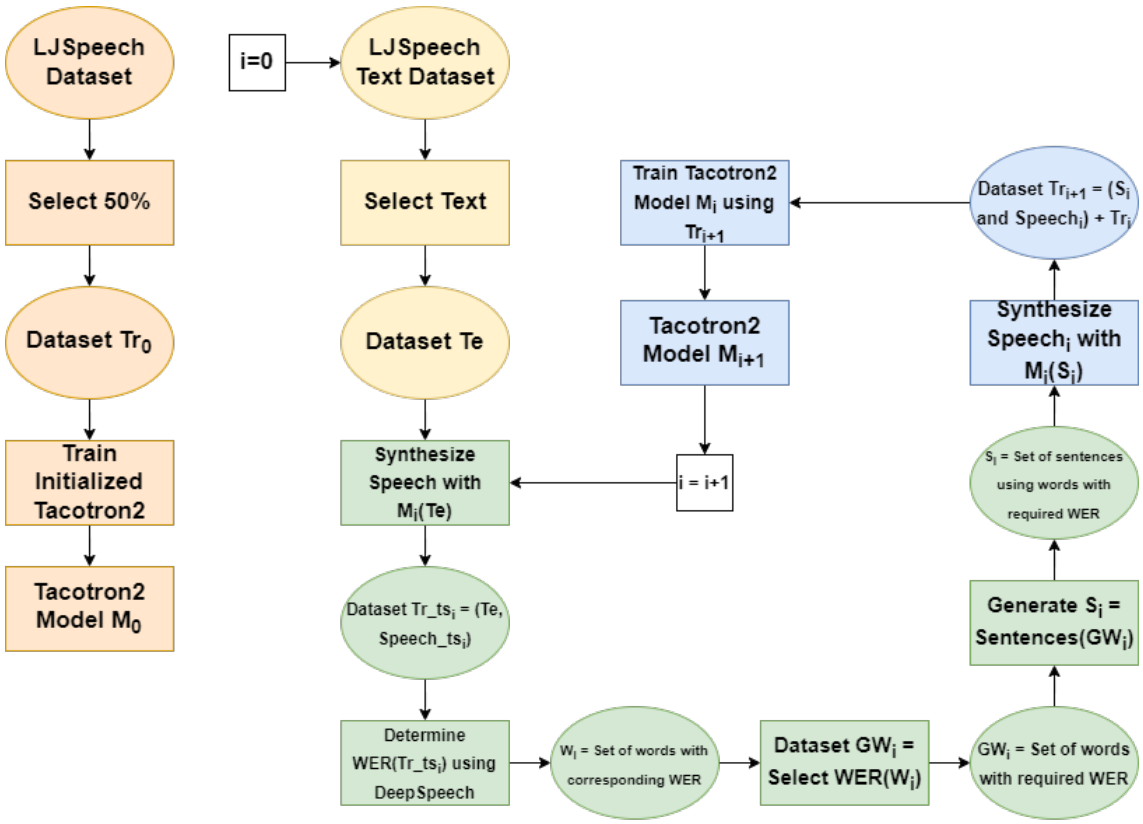

Figure 2: A graphical representation of our bootstrapping scheme

What follows is a short description of the generic key steps in our bootstrapping scheme, after which we will go into detail with respect to the bootstrapping scheme we used for our research.

Initialization: Data set

The data set is split into two parts, one part used for training the text-to-speech engine and one part used to create our own transcribed data using this trained model, called the training and test data sets respectively. We do this by simply keeping track of which files belong to which of the aforementioned groups and provide the text-to-speech engine with this list. Essential to this step is finding the right margin with which we will split the data set. If our training data set is too small, the resulting trained model of our text-to-speech engine will be inadequate, whereas a training set too large will invalidate our low resource

environment. The goal is thus finding the minimal size of our training data set with which we can obtain sufficient results.

Initialization: Text-To-Speech Engine

The obtained training data set is used to train our text-to-speech engine, generating checkpoints along the way. These checkpoints are trained models which will either be used to synthesize our own data set, or resume training. We provide such a checkpoint with the text from the test data set which we had set apart for creating our own transcribed data. The trained text-to-speech model uses this data to create audio files. Combining these audio files with the given input used to create these audio files results in the trained text-to-speech model synthesizing its own transcribed data set. This synthesized data set is then passed to our data evaluation model for evaluation.

Start Iteration: Evaluate Data

The first step in our data evaluation model is the pre-trained speech recognizer. This speech recognizer is used to generate the plain text representation, or transcript, of our synthesized audio files. The resulting transcripts are then compared to the original transcripts used to synthesize these audio files. A higher quality audio file corresponds to the degree to which the transcripts match. During the comparison of these transcripts we keep track of several interesting statistics. For each word in a transcript we will try to find a match in the original transcript. If this match is found, we update a counter in a log file which indicates the number of matches of that word. Similarly a counter for incorrect matches is updated once we do not find a match in the transcripts. Our log file containing the list of words as well as the counters indicating both the number of correct and incorrect matches is passed to the next step in our model.

Data Synthesis: Sentence Generation

For each word in the log file we can use the counters indicating the number of correct and incorrect matches to calculate the percentage of correct matches out of all the matches for said word. We can then use this percentage of correct matches by comparing it to a pre-defined margin. Words with a lower amount of matches than this margin are deemed insufficient whereas the other words are deemed sufficient. The result is a set of sufficient words as well as a set of insufficient word. The words are then passed to a sentence generator. The generated sentences obtained from this generator are passed to the trained text-to-speech model for synthesis. The result is once again a set of audio files with its corresponding transcripts.

Data Synthesis: Data Sets

The newly generated files are added to the training data set originally used to train the text-to-speech engine. The final result is two new data sets, one containing the original data set plus the sentences with the sufficient words and one containing the original data set plus the sentences with the insufficient words.

Our hope is that by providing insufficient words with a context of sufficient words, the speech synthesizer can achieve an improvement of the correctness of insufficient words. This holds both for insufficient words added to the sentences generated using sufficient words, as well as the sufficient words around which we generate sentences using some insufficient words.

End Iteration: Training

Using the now updated data set we can continue training our text-to-speech engine producing a bootstrapped model. Once training has been completed we can either repeat above steps for another iteration in our bootstrapping scheme, or validate our results by comparing it to a model trained for the same number of iterations on the original data set.

Now that we have defined our general method, we can elaborate on the specific method we used. This specific method can be divided into several subsections:

- Initialization: LJSpeech data set
- Initialization: Train Tacotron2
- Start Iteration
- Data Selection: Data set synthesis and conversion
- Data Selection: DeepSpeech
- Data Synthesis: Project Gutenburg
- Data Synthesis: Update data set
- Training: Retrain Tacotron2
- End Iteration

Above components run partially using the NVidia docker within a PyTorch container provided by NGC. A few scripts were written to ensure that the individual components can work in tandem.

Initialization: LJSpeech dataset

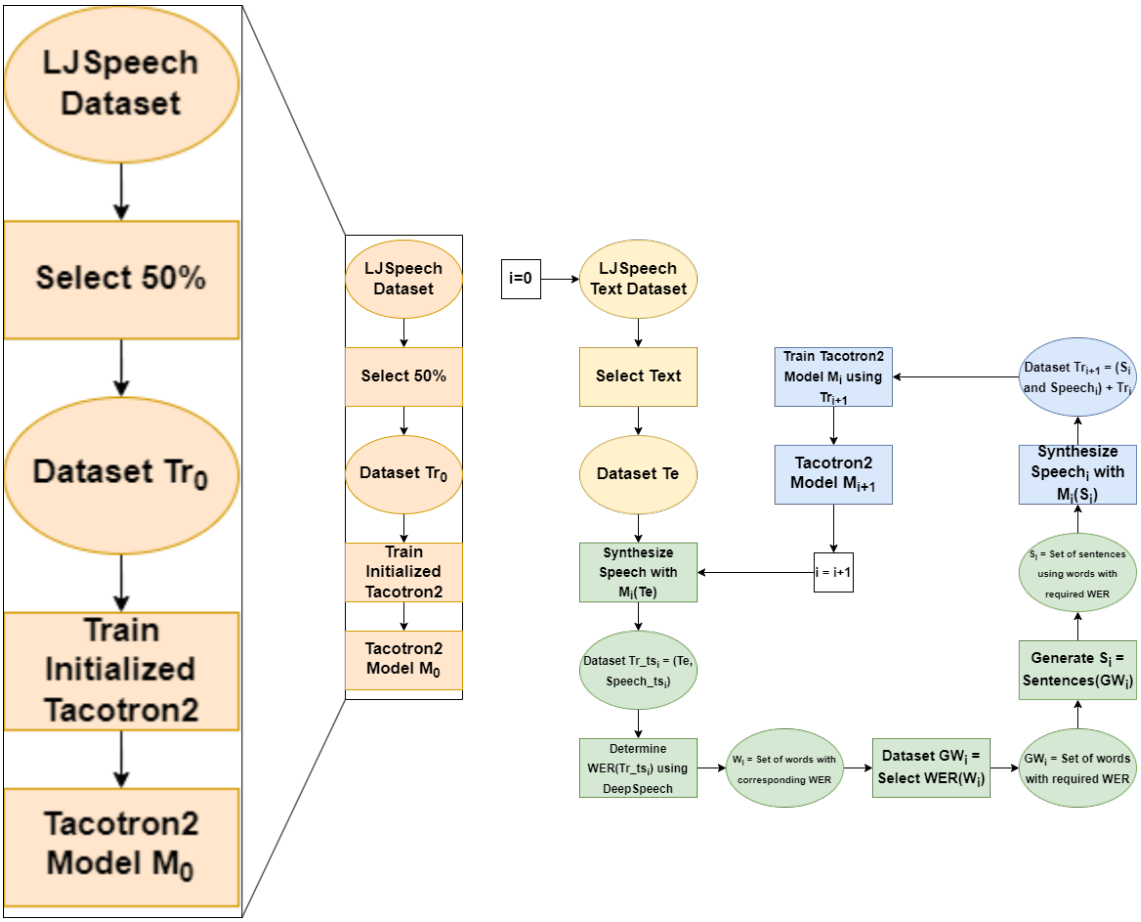

Figure 3: A graphical representation of the data set split

The LJSpeech dataset is obtained by simply executing a script included within the Tacotron2 directory [Tac] of the DeepLearningExamples as provided by NGC. The LJSpeech data set is first pre-processed by a script provided with the Tacotron2 model. This script converts the wave files contained in the data set into mel spectograms such that they can be used by the Tacotron2 model. We have written a script that takes the list of filenames contained in a file in the data set and simply creates two other files containing a split of the data set based on a margin given by the user. Rather than the Tacotron2 model taking the aforementioned list of all files contained in the data set, it will now take the list that we created and train on the files contained in said list. The remaining list of non-used audio files is used during inference as the input of our Tacotron2 model for data synthesis.

Initialization: Tacotron2

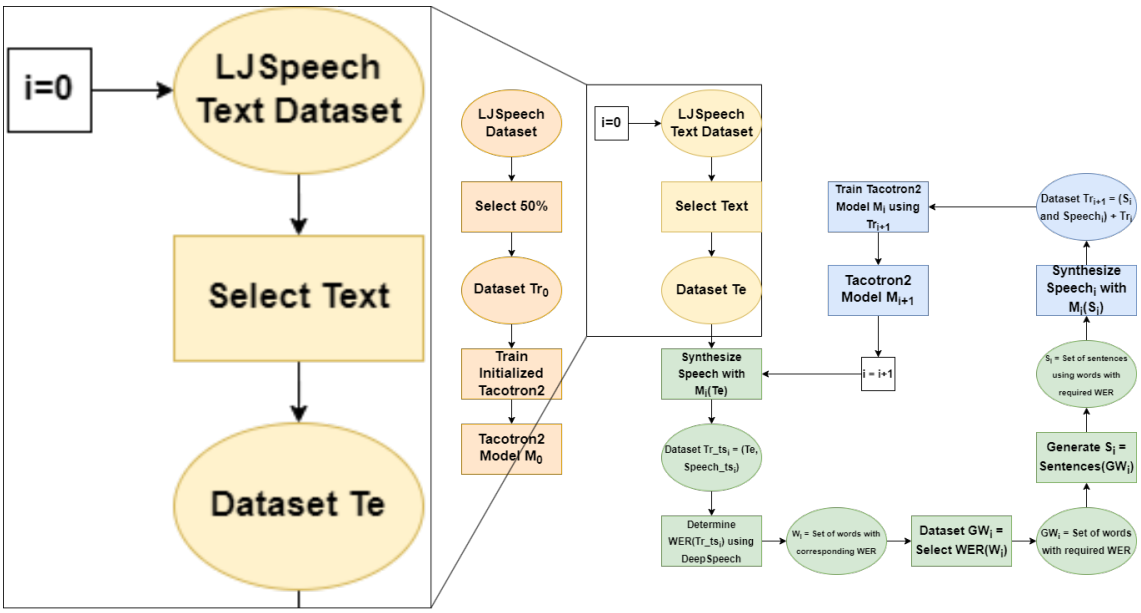

Figure 4: A graphical representation of the Tacotron2 training as well as the initial LJSpeech split

The Tacotron2 component is a combination of two neural network models:

- A modified Tacotron2 model
- A flow based WaveGlow model

Together these models form our text-to-speech system. In the context of our bootstrapping scheme we will only be training our Tacotron2 model whereas the WaveGlow model is a pre-trained model provided by NGC [Wav]. The combined model is trained and retrained on the LJSpeech dataset and uses this dataset to synthesize its own data using the split as defined in the previous section. The training of the baseline as previously defined is done by training the same model, but rather than using the split LJSpeech data set the entire data set is used. Our initial model is trained from scratch, but given a checkpoint we can continue training from a previous training session using the same or a different data set. The model is trained by iterating the entire data set through the model (one epoch) multiple times, where checkpoints are constructed at certain epoch intervals.

Start Iteration: Data set synthesis and conversion

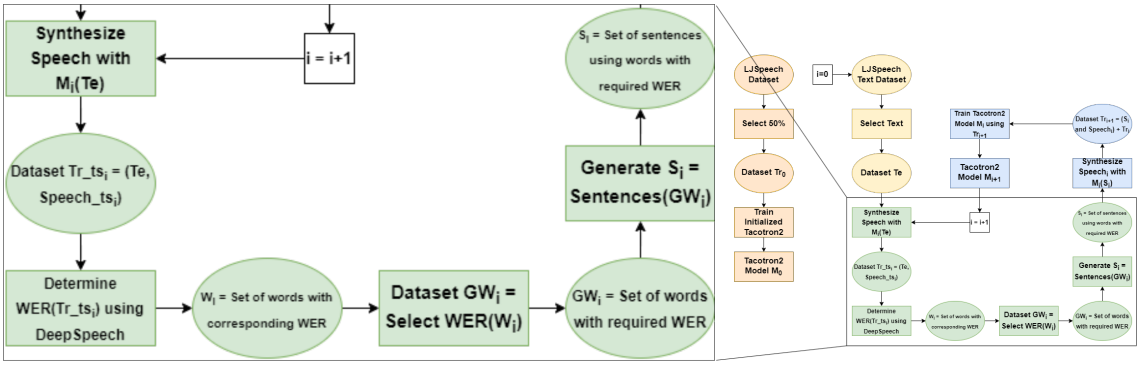

Figure 5: A graphical representation of the sentence generation

Once the model has finished training, we can use a generated checkpoint to run inference. Inference requires a file containing text to be provided and uses the trained model to generate an audio file corresponding to the text contained in the provided file. By providing the trained model with the file containing the non-used LJSpeech lines as mentioned above, we can create our own transcribed data set. This data set is then primed for evaluation by the DeepSpeech model.

Before we can use the transcribed audio files generated by the Tacotron2 model as input for DeepSpeech, we need to ensure these files adhere to the requirements set for input of DeepSpeech. One main feature the output of the Tacotron2 model and the input of the DeepSpeech model vary on is the sample rate. While both the Tacotron2 model and DeepSpeech can be retrained in such a way that both use the same sample rate, it is not an option when we use pre-trained models. Since both the WaveGlow component of the Tacotron2 model and DeepSpeech itself are pre-trained, we will thus have to convert the output from the Tacotron2 model to a format which can be used as input for DeepSpeech. To achieve these results we have written a script using the SoX (Sound eXchange) library [Sox]. The resulting data set is used as input for our DeepSpeech model.

Much like the Tacotron2 model, DeepSpeech also runs in its own virtual environment. Unlike Tacotron2 however, we will not be training DeepSpeech and will simply use a pre-trained model as described by Hannun et al. [Han+14]. With this model and the data set obtained from the previous step we can run inference. During the inference process, DeepSpeech generates a plain text representation corresponding to the audio file used as input.

A script is used which extracts the individual words in these plain text files based on white spaces. These words are then compared to the extracts of the provided transcript.

By keeping track of each word in a plain text log file we can update counters indicating a match or not a match for said word.

By calculating for each word whether or not it is sufficient as mentioned in section 6 Data Synthesis: Sentence Generation, we can use these words to generate sentences. This sentence generation is done by providing a script with the plain text transcripts of multiple books, as provided by Project Gutenberg. The "generation" part is in essence simply trying to find a matching sentence for our word in these transcripts.

Data Synthesis: Update data set and retraining Tacotron2

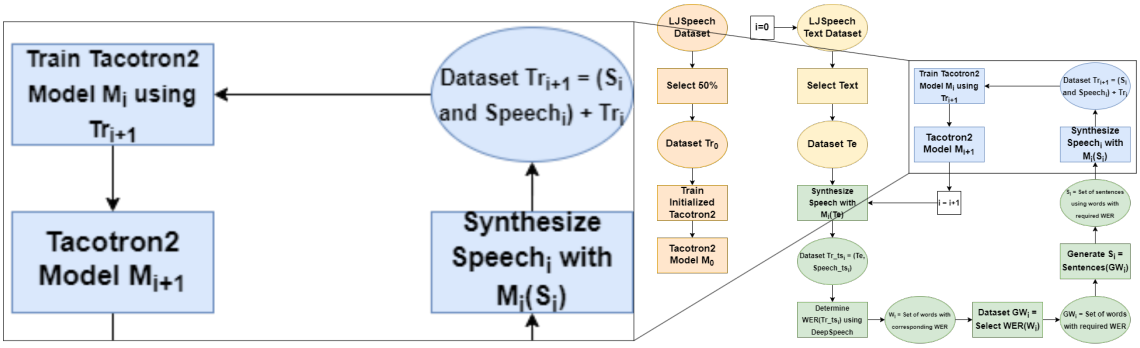

Figure 6: A graphical representation of the synthesis and addition of the generated sentences

The new sentences are passed to our trained Tacotron2 model for synthesis resulting in a newly synthesized data set. The newly obtained data set is added to the training data set. We do this by simply updating the list of files contained in the training data set and adding the corresponding audio files to the directory containing the entire data set. Having finished above steps we can retrain Tacotron2 and repeat the previous components of the bootstrapping scheme as needed. Do note that we will not train our baseline using the augmented data set, but simply continue training using the original training data set thus ensuring we always have a baseline trained on the same amount of epochs as our current bootstrap iteration.

We end our iteration by retraining the Tacotron2 model which we started our iteration with using the data sets obtained in above subsection. This results in newly trained Tacotron2 models which we can either start a new iteration with or compare to our baseline so that we may validate their results.

7 Experimental setup

We describe several experiments which we would like to conduct. These experiments focus on the effect of models trained using our bootstrapping scheme when compared to models trained in lower- and higher resource environments. Do note that these experiments are an extension of the main goal of this research: Is it possible to achieve similar if not better results with respect to speech synthesis by training on a lower resource data set using our bootstrapping scheme?

Several questions which fall under this umbrella question are as follows:

- Can our bootstrap achieve improvements to the baseline when trained on the same data set?
- Can we achieve similar results to Tacotron2 trained in a resource-rich environment by training using our bootstrapping scheme in a low-resource environment?

Comparison between models

For a fair comparison, all models will have to train for the same amount of epochs (one iteration of the dataset through our model) and the resulting trained models will be evaluated on the same data. This evaluation is done by taking the sMOS which produces a score based on the performance of the trained models on above data set. The average sMOS of a trained Tacotron2 model using our bootstrap is then compared to that of the corresponding baseline which reveals the effectiveness of the bootstrapping method on the resulting sMOS. The same comparisons are done using the WER of the models. The result is a comparison between the models on both sMOS and WER.

sMOS and WER of the baseline models

Since we want to test if our bootstrap has any effect at all, we define two baselines to compare it to. One baseline is trained on the the entire LJSpeech data set whereas the other is trained on half of the LJSpeech data set. This allows us to compare the effectiveness of our bootstrapping method by comparing the sMOS to the model trained in the low resource environment. By comparing it to the baseline trained on the entire data set we can see the difference between a model trained in a lower-resource environment and the same model trained in a higher resource environment.

Experiment 1

We would like to test the effectiveness of our bootstrapping scheme in a lower resource environment. This will be done by comparing the results of Tacotron2 trained on the

smaller data set to the results of Tacotron2 trained on the same data set as well as our bootstrapping scheme.

Experiment 2

To conduct this final experiment we first train Tacotron2 on the entire LJSpeech data set. Next we train Tacotron2 on the smaller data set as we have done in the previous experiments. Similarly we then use our bootstrapping scheme on the model trained on the smaller data set and compare the results to Tacotron2 trained on the entire data set.

Dataset

We opted for a 50-50 split on the LJSPeech data set with respect to training set Tr and test set Te. We came upon this number using a trial and error method: train the model on a split (25-75, 50-50, 75-25) and manually evaluate the audio produced by these models. Since the 25-75 split produced unintelligible audio, it was passed up for its better counterparts. The 50-50 model produced intelligible audio, although not as good as the 75-25 split. Since our goal was to bridge the gap between lower resource and higher resource environments, we chose the 50-50 split with it being the lower resource environment of the two while still producing intelligible audio.

Training

Before using our bootstrap, we trained the models on the 50-50 split for 200 epochs. LJSpeech consists of 13100 files resulting in our model being trained on 6550 files. We denote this model as **50p200** (50% of LJSpeech, 200 epochs). Our bootstrap then takes the 6550 unused LJSpeech files and uses the 50p200 model to synthesize the transcripts. We then walk through the steps as described in our method and use the 80% correct rate as a cutoff point for the words to decide which data set they will be included in. Much like the 50-50 split we chose the 80% through trial and error. Namely we looked at the respective sizes of the data sets created by this split. Through using the 80% cutoff, the number of files using the words above 80% was roughly 6000, while the number of files using the words below 80% was roughly 2000. We felt that adding the 6000 sentences created from the words above 80% to the original 6500 sentences would approximate the original LJSpeech data set size of roughly 13000 files and as such would make for an interesting comparison. The resulting data sets were then in according with our bootstrapping scheme used to retrain the 50p200 model for another 100 epochs. The resulting models we denote as **50p300h80** (50% of LJSpeech, 300 epochs, data set enhanced with words above the 80% correctness cutoff) and **50p300l80** (50% of LJSpeech, 300 epochs, data set enhanced with words below the 80% correctness cutoff).

8 Results

This section describes the results achieved after we conducted the experiments as described in the previous section as well as our baselines.

Baselines

Having defined the baselines as in the previous section, we obtain the following results:

Model	Trained on	Epochs	Avg. sMOS	WER
Tacotron2 Baseline 1	100% LJSpeech	300	**2.88**	**0.17**
Tacotron2 Baseline 2	50% LJSpeech	300	2.87	0.20

Table 1: Average sMOS and WER of the baseline models

Both models were evaluated on our evaluation data set. This evaluation was done by taking the sMOS of the 100 lines of the same text synthesized by the respective models. These scores are then used to calculate the average sMOS over these 100 lines which results in the average sMOS as displayed in Table 1. The same lines are then compared to the original lines from which we derive the WER also displayed in Table 1.

Experiment 1

As mentioned in section 7: Comparison between models, we evaluate all our models on the same data set of 100 lines. Experiment 1 as described in section 7 tests the effectiveness of our bootstrapping scheme by comparing the sMOS of both the 50p300h80 and the 50p300l80 models to that of the **50p300** model (or Tacotron2 Baseline 2 as described in our Baselines subsection). We report the following results:

Model	Trained on	Epochs	Avg. sMOS
50p300	50% LJSpeech = 6550 lines	300	2.87
50p300h80	50% LJSpeech + h80 = 12200 lines	300	**3.25**
50p300l80	50% LJSpeech + l80 = 9350 lines	300	3.19

Table 2: Average sMOS of the models on the test data set compared to the 50p300 baseline

We see that using our bootstrap, the 50p300h80 model achieves an improvement over the baseline of $3.25/2.87 * 100 = 113.24 \approx 13.24\%$ with respect to the sMOS. The 50p300l80 model achieves an improvement over the baseline of $3.19/2.87 * 100 = 111.14 \approx 11.14\%$. An interesting thing to note is the number of extra lines these models use. Should we weigh this number in the final result, we get the following numbers:
Here we see that per added line, the 50p300l80 model achieves a significant higher increase in sMOS than the 50p300h80 model with respect to the baseline.

Model	Trained on	Impr. per added line
50p300	50% LJSpeech = 6550 lines	0%
50p300h80	50% LJSpeech + h80 = 12200 lines	0.0023%
50p300l80	50% LJSpeech + l80 = 9350 lines	**0.0047%**

Table 3: Percentage improvement in sMOS of the models compared to the baseline

Experiment 2

Section 7 goes on to describes experiment 2. This experiments uses the **100p300** model (or Tacotron2 Baseline 1 as described in our Baseline section) as a baseline for our bootstrapped models 50p300h80 and 50p300l80. We report the following results:

Model	Trained on	Epochs	Avg. sMOS
100p300	100% LJSpeech = 13100 lines	300	2.88
50p300h80	50% LJSpeech + h80 = 12200 lines	300	**3.25**
50p300l80	50% LJSpeech + l80 = 9350 lines	300	3.19

Table 4: Average sMOS of the models on the test data set compared to the 100p300 baseline

Again we see that our models using the bootstrapping scheme achieve a notable improvement over the respective baseline in terms of sMOS.

WER

To further illustrate the importance of our bootstrapping method with respect to the sMOS, we would like to display all models as follows:

Model	Trained on	Epochs	Avg. sMOS
50p200	50% LJSpeech = 6550 lines	200	2.88
50p300	50% LJSpeech = 6550 lines	300	2.87
100p200	100% LJSpeech = 13100 lines	200	2.88
100p300	100% LJSpeech = 13100 lines	300	2.88
50p300h80	50% LJSpeech + h80 = 12200 lines	300	**3.25**
50p300l80	50% LJSpeech + l80 = 9350 lines	300	3.19

Table 5: Comparison of all models on average sMOS

In the above table we can see that for both the normal 50p models as well as the normal 100p models there is a decrease in average sMOS between the 200 and 300 epochs. The bootstrapped models on the other hand achieve a significant improvement in the 200 to 300 epochs range. While these results may seem promising, the sMOS has a major flaw. Upon further investigation we found that most of the audio files produced by the bootstrapped

models used to calculate the sMOS also had a higher WER. This was due to a high pitched note playing is the audio files rather than a spoken sentence. While these high pitched notes did achieve significant improvement in sMOS, it obviously does not align with the goal of our research. Taking WER into account, we report the following results:

Model	**Trained on**	**Epochs**	**WER**
50p200	50% LJSpeech = 6550 lines	200	0.22
50p300	50% LJSpeech = 6550 lines	300	0.20
100p200	100% LJSpeech = 13100 lines	200	**0.14**
100p300	100% LJSpeech = 13100 lines	300	0.17
50p300h80	50% LJSpeech + h80 = 12200 lines	300	0.72
50p300l80	50% LJSpeech + l80 = 9350 lines	300	0.39

Table 6: Comparison of all models on WER

Above table illustrates the shortcomings of just measuring on sMOS. We can clearly see that the baseline models average a reasonable accuracy of roughly $(1 - (0.22 + 0.20 + 0.14 + 0.17)/4) * 100 = 82\%$, whereas the 50p300h80 model has an accuracy of roughly $(1-0.72)*100 = 28\%$ and the 50p300l80 model an accuracy of roughly $(1-0.39)*100 = 61\%$. Given that the bootstrapped models produce faulty audio files as the result of a fault in the Tacotron2 architecture, we report the results of these models in terms of both sMOS and WER before and after filtering out these faulty lines.

Model	**Total correct**	**Total incorrect**	**sMOS**	**WER**
50p300h80b	366	946	**3.25**	0.72
50p300h80a	360	153	2.91	0.29
50p300l80b	794	518	3.19	0.39
50p300l80a	791	179	2.95	**0.18**

Table 7: Comparison of the bootstrapped models on both sMOS and WER before and after filtering out faulty lines

Above table shows the number of correct and incorrect words produced by the bootstrapped models, paired with the sMOS and WER of said models before (50p300h80b and 50p300l80b) and after (50p300h80a and 50p300l80a) the faulty lines were filtered out. While we do see a drop in sMOS, we see a significant increase in WER for both models after we filter out the faulty lines.
Above table summarizes the results of both the bootstrapped models as well as the baseline models after filtering out the faulty lines.

Model	sMOS	WER
50p300	2.87	0.20
100p300	2.88	**0.17**
50p300h80	2.91	0.29
50p300l80	**2.95**	0.18

Table 8: sMOS and WER of all models after filtering out the faulty lines

9 Discussion

In our experimental setup section we had described two experiments. Taking our experimental results into account, we provide some remarks regarding the success of these experiments:

Can our bootstrap achieve improvements to the baseline when trained on the same data set?
The success of our bootstrap was measured using two factors: sMOS and WER. These measurements were used on both the results with and without the faulty files, providing varying results. When we compare our bootstrapped models (50p300h80 and 50p300l80) to the 50p baseline, we can immediately see that both the bootstrapped models achieve an improvement in sMOS over the baseline with or without the faulty files. The WER however suffers greatly from these faulty files. The result is that both the bootstrapped models are far worse in terms of WER in comparison to the baseline. If we were to take out these faulty files, we can see that the 50p300l80 model achieves an improvement in WER over the baseline whereas the 50p300h80 model still performs worse than the baseline in terms of WER. Using these results we can conclude that in terms of sMOS, our bootstrapping method does achieve an improvement over the baseline when trained on the same data set. Furthermore we show that by filtering out the faulty files created as the result of a fault in the Tacotron2 architecture, our 50p300l80 model achieves an improvement over the baseline in terms of WER, giving our bootstrapping scheme an edge over this baseline in terms of both sMOS and WER.

Can we achieve similar results to Tacotron2 trained in a resource-rich environment by training using our bootstrapping scheme in a low-resource environment?
Using the results provided by the previous experiment, we compare our models to the 100p baseline. We can again see that our bootstrapped models achieved a higher sMOS than the baseline using both the scenario taking the faulty lines into account, as well as the scenario where we filtered out these lines. Neither bootstrapped model however achieved a better WER than the 100p baseline in both the filtered and unfiltered scenario. This on the other hand does not mean that it is impossible for our bootstrapping scheme to achieve similar results to the baseline. We only ran our scheme for one iteration, should we have ran our scheme for multiple iterations, the results might favor our bootstrapping scheme rather

than the traditional method. Using the results we have now however, we can conclude that our model trained in a low-resource environment once again achieves an improvement in terms of sMOS, but not fully bridges the gap in terms of WER when compared to the traditional method of training in a resource-rich environment.

A final remark is the difference between the 50p300h80 and the 50p300l80 models. The 50p300l80 model outperforms the 50p300h80 model both in terms of sMOS and WER, all while being trained on less lines than said model. These results indicate that our bootstrapping scheme benefits the most from a data set comprised of sentences using words with a correctness of lower than 80%.

10 Conclusion

In our research we tried to answer the following question: Is it possible to achieve similar if not better results with respect to speech synthesis by training on a lower resource data set using our bootstrapping scheme? To answer this question we created a bootstrapping scheme. Using our bootstrapped models we conducted several experiments.
These experiments yielded promising results on the sMOS front, whereas these results lacked on the correctness front. Correcting however for faulty lines alleviated this problem and showed that our bootstrapping scheme could achieve competitive results to the model trained on the entire data set in terms of WER. While the results of our bootstrapped models in terms of WER were not up to par with models trained in resource-rich environments, they were better than models trained on the same data set without our bootstrapping method with the 50p300l80 model showing the most promise.
Our research shows that using our bootstrapping scheme it is possible to achieve an improvement in sMOS over models trained on a larger data set, as well as a possibility to bridge the gap in WER between these models.

11 Future Research

We have several points we would have liked to have explored further. The first point being the use of a different speech synthesizer. Other than finding out the effect of our bootstrapping scheme on different speech synthesizers, we would have liked to find out if the faulty lines were a problem specific to Tacotron2 or whether or not other speech synthesizers suffered from the same problem. Another point we would have liked to explore was a change in the training method. In our research we opted to first train for 200 epochs after which we conducted our bootstrapping method followed by another 100 epochs of training. Since the learning process of a model differs at different epochs (early epochs for global factors, later epochs for more specific factors) it would have been interesting to see the effect of the initial training being longer/shorter, after which we run the bootstrap

followed by another training which can also be varied in length.

Another point to explore could be generating more sentences during the bootstrapping steps. In our research we opted to generate one sentence per word. Should the hardware allow it, then future research might choose to generate more sentences thus generating larger data sets per iteration. Our final point to explore, and by extend the most obvious one, would be to run our bootstrapping scheme for multiple iterations and compare these results to the baseline.

Judging by the results achieved by our research, future research could look into the effectiveness of our bootstrapping model to increase the quality of the audio. Namely shifting the focus to the 50p300l80 model, as this model has shown to provide competitive results in both terms of sMOS and WER when compared to the baseline trained on the entire data set.

References

[GN07] Bálint Gyires-Tóth and G Németh. "Speech Enabled GPS Based Navigation System in Hungarian for Blind People on Symbian Based Mobile Devices". In: (Jan. 2007).

[Par+08] Youngja Park, Siddharth Patwardhan, Karthik Visweswariah, and Stephen Gates. "An Empirical Analysis of Word Error Rate and Keyword Error Rate". In: Sept. 2008, pp. 2070–2073. DOI: 10.21437/Interspeech.2008-537.

[BS09] Felix Burkhardt and Joachim Stegmann. "Emotional Speech Synthesis: Applications, History and possible Future". In: *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2009*. Ed. by Rüdiger Hoffmann. TUD-press, Dresden, 2009, pp. 190–199. ISBN: 978-3-941298-31-6.

[Tay09] Paul Taylor. "Text-to-speech synthesis". In: Cambridge, UK ; New York : Cambridge University Press, 2009. Chap. 1.2.

[Pov+11] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Vesel. "The Kaldi speech recognition toolkit". In: *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding* (Jan. 2011).

[Han+14] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. *Deep Speech: Scaling up end-to-end speech recognition*. 2014. DOI: 10.48550/ARXIV.1412.5567. URL: https://arxiv.org/abs/1412.5567.

[SWH16] Robert Streijl, Stefan Winkler, and David Hands. “Mean opinion score (MOS) revisited: methods and applications, limitations and alternatives”. In: *Multimedia Systems* 22 (Mar. 2016), pp. 213–227. DOI: 10.1007/s00530-014-0446-1.

[She+18] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. “Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 4779–4783. DOI: 10.1109/ICASSP.2018.8461368.

[Li+19] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M. Cohen, Huyen Nguyen, and Ravi Teja Gadde. *Jasper: An End-to-End Convolutional Neural Acoustic Model*. 2019. DOI: 10.48550/ARXIV.1904.03288. URL: https://arxiv.org/abs/1904.03288.

[Lo+19] Chen-Chou Lo, Szu-Wei Fu, Wen-Chin Huang, Xin Wang, Junichi Yamagishi, Yu Tsao, and Hsin-Min Wang. “MOSNet: Deep Learning based Objective Assessment for Voice Conversion”. In: *Proc. Interspeech 2019*. 2019.

[Ren+20] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. “Fastspeech 2: Fast and high-quality end-to-end text to speech”. In: *arXiv preprint arXiv:2006.04558* (2020).

[Sha+20] Yash Sharma, Basil Abraham, Karan Taneja, and Preethi Jyothi. “Improving Low Resource Code-switched ASR using Augmented Code-switched TTS”. In: *INTERSPEECH*. 2020.

[Xu+20] Jin Xu, Xu Tan, Yi Ren, Tao Qin, Jian Li, Sheng Zhao, and Tie-Yan Liu. “LRSpeech: Extremely Low-Resource Speech Synthesis and Recognition”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 2802–2812. ISBN: 9781450379984. DOI: 10.1145/3394486.3403331. URL: https://doi.org/10.1145/3394486.3403331.

[Faz+21] Amin Fazel, Wei Yang, Yulan Liu, Roberto Barra-Chicote, Yixiong Meng, Roland Maas, and Jasha Droppo. “SynthASR: Unlocking Synthetic Data for Speech Recognition”. In: *Proc. Interspeech 2021*. 2021, pp. 896–900. DOI: 10.21437/Interspeech.2021-1882.

[Ge+21] Zhenhao Ge, Lakshmish Kaushik, Masanori Omote, and Saket Kumar. “Speed up Training with Variable Length Inputs by Efficient Batching Strategies”. In: *Proc. Interspeech 2021*. 2021, pp. 156–160. DOI: 10.21437/Interspeech.2021-2100.

[Gio+21] Manuel Giollo, Deniz Gunceler, Yulan Liu, and Daniel Willett. "Bootstrap an End-to-End ASR System by Multilingual Training, Transfer Learning, Text-to-Text Mapping and Synthetic Audio". In: *Proc. Interspeech 2021*. 2021, pp. 2416–2420. DOI: 10.21437/Interspeech.2021-198.

[NM21] Owais Nazir and Aruna Malik. "Deep Learning End to End Speech Synthesis: A Review". In: *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*. 2021, pp. 66–71. DOI: 10.1109/ICSCCC51823.2021.9478125.

[kei] keithito. *LJSpeech 1.1 data set*. URL: https://keithito.com/LJ-Speech-Dataset/. (accessed: 17.08.2022).

[Sox] Webpage Sox. *Sound eXchange documentation*. URL: http://sox.sourceforge.net/SoX/Resampling. (accessed: 26.Jul.2021).

[Tac] Webpage Tacotron2. *Tacotron2 and Waveglow 2.0 for PyTorch*. URL: https://ngc.nvidia.com/catalog/resources/nvidia:tacotron_2_and_waveglow_for_pytorch. (accessed: 22.06.2022).

[Wav] Webpage Waveglow. *WaveGlow AMP checkpoint, 14000 epochs*. URL: https://ngc.nvidia.com/catalog/models/nvidia:waveglow256pyt_fp16. (accessed: 22.06.2022).