End-to-End Text to Speech Synthesis

CS 229 Project Report, Autumn 2018

Xiao Wang
xiao1105@stanford.edu

Yahan Yang
yangy96@stanford.edu

Ye Li
liye5@stanford.edu

Abstract

While traditional text to speech synthesis research usually separates text to speech process into multiple steps, such as encoder, decoder, and wave synthesizer, this process now could be constructed together to perform an end-to-end synthesis model. In our project, we applied word/phoneme mapping, signal filter and machine learning techniques (support vector regression (SVR), simple neural network, and Seq-2-seq with attention model) to transform text to speech. As a result, our synthesis system could successfully generate a wav file by inputting a single text. Moreover, the seq-2-seq model gained the highest MOS(Mean opinion score) of 2.5, which is determined by a group of listeners, higher than other two baseline model.

I. Introduction

Audio signal processing and speech synthesis is a complicated process which includes text normalization tool, encoder, decoder, and wave synthesis. One of the most useful applications is generating speech from text. With rapid development of deep learning, researchers invent many end-to-end algorithms for real life problems, which leads more innovative methods in solving speech synthesis problem. There are several developed models which focus on speech synthesis:Tacotron from Google[1], Char2Wav[3], Deep Voice[4] from Baidu, etc. Inspired by those models, our project targets in generating speech from text using an end-to-end speech synthesis system. In particular, we want to generate a wav file with a single text input. There are three main parts in the system: preprocess from text to spectrogram numpy, training models (SVR, neural network, seq-2-seq with attention), postprocess from spectrogram to wav file.

II. Related Work

There were some past research in end-to-end text to speech synthesis.

A. Char2Wav

Char2Wav project from Montreal Institute of Learning Algorithms proposed a new end-to-end model which included two components: a reader and a neural vocoder. The reader is composed of a bidirectional recurrent neural network (RNN) encoder and a recurrent neural network with attention decoder. Neural vocoder is a sample RNN to generate waveform file. The model learns to generate speech from text and phoneme.

B. Tacotron from Google

Tacotron takes characters as input and raw spectrogram as output without knowing features as well as phoneme-level alignment. Its model is consisted of encoder based on CHBG module, an RNN based attention decoder, and a post-processing module. They proposed a CHBG module consisted of 1D convolutional layer, highway network and bidirectional GRU to extract. After learning, the output goes into Griffin-Lim synthesizer to generate waveform.

C. Deep Voice from Baidu

Deep Voice 3 project from Baidu presented an innovative a fully-convolutional architecture that includes encoder, decoder with attention block and converter to transform text to speech. The mechanism of the architecture firstly interprets textual feature into vocoder parameters and then input those features into different neural vocoders (WaveNet/WORLD/Griffin-Lim) for waveform synthesis.

III. Preprocess and post-process of datasets

We used LJ-speech, a public domain speech dataset consisting of 13,100 short audio clips of a single speaker reading passages from 7

non-fiction books, as our raw dataset. Before and after utilizing our machine learning models, we did several steps of preprocessing and post-processing of the dataset to optimize the learning process.

A. Input text

Our input text raw data are sequences of words. Using the CMU dictionary, input words are randomly mapping to either phonemes or alphabet to increase naturalness in generating speech, which turns the input data to a 2D numpy array. First dimension of the array is number of input texts, and second dimension of the array is the number mapped by dictionary, represents either phoneme or alphabet.

B. Input audio

Input audio files are treated as testing labels to train our model. To digitize audio waveforms, the audio file will first pass a FIR filter, which is an efficient tool to derive discrete frequency response of the audio. This process also help smooth noise in wav file. Then, the output of the FIR filter will secondly be passed into Short time Fourier transform(STFT) structure, so that the sinusoidal frequency and phase content of local sections of a signal as it changes over time can be determined.

After two filters, the training label turns into a 3D numpy array, with first dimension of the array is number of inputs, second dimension of time steps which is depending on the time duration of the audio, and third dimension of the amplitude of spectrogram, with a fixed length of 1025.

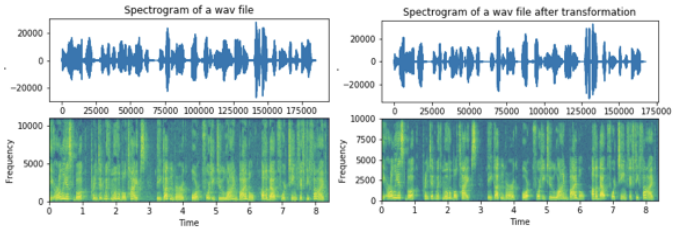

Figure 1. Spectrogram before/after FIR

C. Zero paddling

Due to the inconsistent lengths across 200 sentences, we padded zeros to increase the length of all 200 sentences to length of the longest sentence. With the same process, we padded matrices of shorter wav-files with rows of zeros, so that the entire set of output arrays share the same dimension. However, due to the randomness of phoneme mapping, our program could not save the numpy in advance, which means the whole preprocess of input text is necessary for each training.

D. Postprocess of output

After the system generates output spectrogram matrix from prediction, we utilize inverse STFT and inverse FIR filter to transfer spectrogram back to audio and save the wave into byte with the method provided in scipy library.

IV. METHODS

We built SVR model and simple neural network model as two baseline models, as well as a complex seq-2-seq model with attention as advanced one.

A. SVR model

The first method we used is a support vector regression (SVR) models.

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Firstly, the program generates a SVR for each timestep, so the total number of SVR in our model equals to the number of time step after we preprocess data. In each time step, our training data requires the model to map multi-input to multi-output, so we have a support vector multi-regressor for training each input text matrix and an array of spectrogram output (length 1025). For each time step, the program takes the corresponding SVR model and the input data to make prediction, and then concatenate prediction together as final output. In our experiments, we tried a linear kernel and a polynomial kernel (utilized in many natural language processing models) for our SVM models.

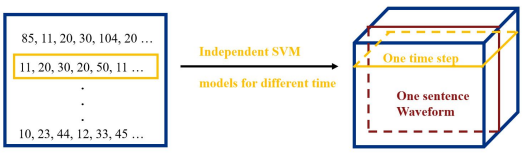

Figure 2. SVR model

One problem for SVR model is that the training time is too long, since the model works

with many separate models and a number of multi-regressors, which takes a long time to run. To improve the problem, we attempted another variation of SVR model: using incremental learning rather than retraining model every time with great amount of spectrogram data. The algorithm we chose from sklearn tool was online passive-aggressive algorithm[5], which also uses hinge loss in its optimization formula. The training process accelerated efficiently with the algorithm, but the result wav file sounds worse than that from polynomial SVR since this algorithm is more close to linear kernel SVR. To achieve better mean opinion score result, we still chose regular polynomial kernel SVR.

B. Simple neural network model

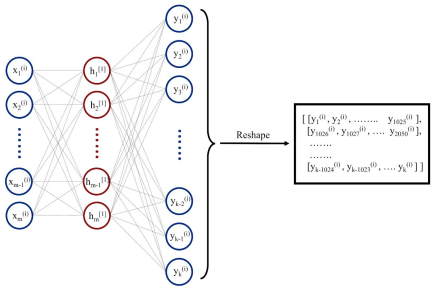

Figure 3. Simple Neural Network Model

A simple neural network is a model that consists layers of neuron and connections among them with weights and biases. It consists three sections: input layer, hidden layer(s), and output layer. Hidden layers are human-determined shapes that can analyze the input data in different perspective, and after all the hidden layers, the output layer uses an activation function to squish hidden results to a limited range of values(typically, 0 to 1). In our model, we used sigmoid function as our activation function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

In this particular problem, we reshaped the training labels to 1D arrays in order to match the neural network structure, and after the model was validated, we reshaped the prediction vectors back to matrices for the post-processing process. The input layer is the word embedding vector encoded from original sentences, and the hidden layer is fully connected with same number of neurons as the input layer. The output layer is a stretched 1D vector of the spectrogram. We set mini batch to be 10, and trained 100 samples each time.

C. Seq-2-seq model with attention

The third method we used is a seq2seq model with one embedding layer and one LSTM(Long short term memory) layer (latent dimension is 512) as encoder and one LSTM layer as decoder. Input data was first put into embedding layer to align the dimension with output wave matrix. And then output of embedding layer was feed into LSTM encoder model. The hidden state and cell state of LSTM model were shared with decoder model.

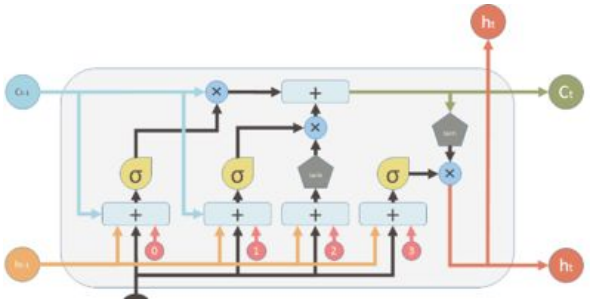

Figure 4. LSTM model diagram

LSTM is a type of recurrent neural network(RNN) contains memory cell, input gate, output gate and forget gate. When the input vector goes into the LSTM cell, it will be temporarily stored in the gates, so that when future inputs come in, the model can adjust the output by those temporarily stored values to prevent gradient vanishing or exploding.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

With attention in LSTM for text translation, it searches all the information from the text and calculate its attention weight to determine the relevance of each word. In each step, encoder LTSM keeps its output of the input sequence and

train the input with attention. Thus, the items in output depend on the corresponding items in input sequence with attention.The attention information is stored in attention vector as shown below.

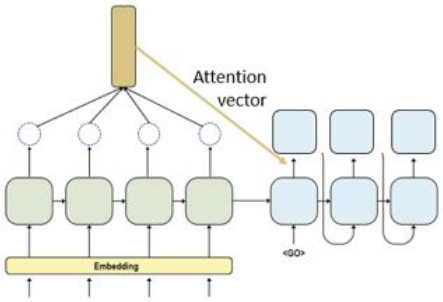

Figure 5. Seq2Seq with attention diagram

During the training process, we train the model with two inputs, text input and audio input, the output is audio wave one time step later than audio input. In the validation process, the encoder model was feed with entire sentence. Then internal states of LSTM were shared with decoder LSTM. Decoder LSTM used the first time step of wave matrix to generate the whole wave matrix of the sentence.

V. Experiments & Results

A.SVR model

The wave file generated from our SVR model mainly consisted of disjoint words, so that it does not sound like consistent human speech.

B.Simple Neural Network

The wave file generated from this simple neural network does not sounds like consistent speech. It was just a random combination of phonemes and words. Although through the training process, the loss function can be minimized down to the magnitude of 10e-4, when comes to the dev/test data, the result sounds not quite reasonable.

C.Seq-2-seq with attention

The wave file generated from this model sounds like human speech. However, inside the wave file, several words were repeated many times, which cannot be counted a complete sentence. Even though after training, the loss function can be minimized down to the magnitude of 10e-3, when comes to the dev/test data, the result sounds not quite reasonable.

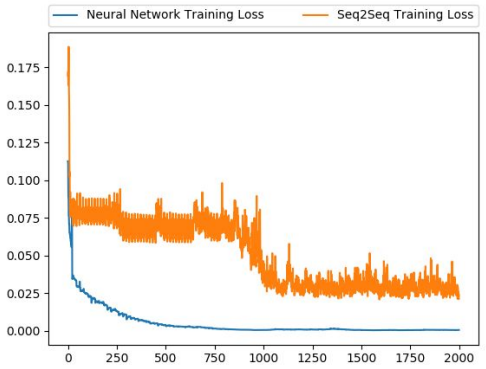

Figure 6. Training loss after 250 iterations

In our experiment, we trained the model with up to 200 input data. To do prediction, the system firstly read a single input text and preprocess data. Our training data for the experiment is from LJ-speech database, and our test text is an input string. After the postprocess procedure of the model prediction, the system stores the generated bytes in a wav file. The training and test accuracy were not included because this regression problem involves timestep operation and comparison of each number in the matrix was not that meaningful. Then, we asked a group of people to listen the wav file and to determine the quality score of our result based on consistency and naturalness. The mean score standard listed in *Table 1* is adopted from Tacotron project[1]. After asking 10 Stanford students, we obtained the mean score of for these three models, respectively. Detailed mean scores rated by 10 students are shown in *Table 2*. We put our generated sample wav files in the web (https://www.xiaowang.me/cs229).

Label	Excellent	Good	Fair	Poor	Bad
Rating	5	4	3	2	1

Table1: Mean Opinion Score Standard

Model	SVR	NN	Seq-2-Seq
Mean	1	1.7	2.5

Table 2: Mean opinion score result

According to MOS comparison, seq-2-seq model with attention gives best score among three models, which is in accordance with our expectation. Nevertheless, this still much lower than 3.82, the score of Tacotron project.

VI. Discussion

According to the result of experiments, all three training methods did not work well. We concluded that this may because of the following reasons:

A. Complexity of problem

Our simple neural network model was only a 4-layer structure and our seq-2-seq was 7-layer. Compared with Tacotron, a ten-layer based RNN, our models are not complicated enough to train our text input numpy with spectrogram numpy sufficiently because end-to-end model did not include any features of audio. In our problem, the output dimension is way higher than the input dimension, so that with the limited knowledge we got from our input text, we need to predict a matrix containing much more information that we cannot see from the input. Due to this fact, it is kind of hard for us to choose an appropriate model and complete the end-to-end process using simple machine learning algorithms.

B. Computation constraints

Another problem is the training size for our models: our training was up to 200 sentences. For our system, the preprocess part demanded a great amount of memory and training parts took long time since the model attempted to map each input numpy with a huge 2D output numpy. This raises up the issue that since we need to sweep all the data we loaded and find the longest sentence to complete zero-padding, those zeros are not efficient data and may take up too much memory in our dataset. However, if we sweep the entire LJ-speech dataset and use the same length every time, it will contradict with the CMU dictionary mapping, because we randomly mapped words to either phonemes or alphabets, which turns out that each time the input encoded may not be same, and that randomness will definitely improve our prediction quality. Thus, 200 sentences were not enough for this system to learn and to fit randomness. The size of training data for other existed models were much bigger than ours. For instance, Tacotron uses 10,000 sentences in their training. Therefore, all those factors were the possible error source of our training models.

VII. Conclusion and Future Work

The above results have proven that models like NN, SVM, simple Seq2Seq don't work quite well for such complex problem. Conclusively, human speech synthesis is an interesting but difficult task and we definitely need more complicated model for this kind of problem. For future work, developing a deeper model is the preliminary improvement. Another possible enhancement is ensembling different models. Moreover, as discussed above, larger size of training is also necessary for a more efficient speech synthesis system.

VIII. Contributions

Xiao: Constructed, trained and tested Seq-2-Seq with attention model. Built data preprocess procedure. Ran Tacotron project and gathered result. Documents/Posters. Mean score survey.
Yahan: Constructed, trained and tested SVM model. Documents/Posters. Mean score survey.
Ye: Constructed, trained and tested the simple neural network model. Documents/Posters. Mean score survey.

IX. Source code link

The source code of this project can be accessed below:
https://drive.google.com/file/d/12RY9A2w_QQje8DopqZ5LlFacCcSf1b-T/view?usp=sharing

References

[1] Wang, Yuxuan, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang et al. "Tacotron: Towards end-to-end speech synthesis." *arXiv preprint arXiv:1703.10135* (2017).

[2] Perraudin, Nathanaël, Peter Balazs, and Peter L. Søndergaard. "A fast Griffin-Lim algorithm." In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2013 IEEE Workshop on*, pp. 1-4. IEEE, 2013.

[3] Sotelo, Jose, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. "Char2wav: End-to-end speech synthesis." (2017).

[4] Ping, Wei, Kainan Peng, Andrew Gibiansky, Sercan Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. "Deep voice 3: Scaling text-to-speech with convolutional sequence learning." (2018)

[5] Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer. "Online Passive-Aggressive Algorithms." (2006)

[6] Github Repo: https://github.com/keithito/tacotron