Speech Synthesis

Noah Brackenbury and Alex Mathson

CS 322, Fall 2015

Problem

Many people throughout the world have, by some means or another, lost their ability to produce speech from their vocal chords. While conversing through text is often enough, speech allows us to project to a larger audience and articulate our thoughts better. Speech can be synthesized through text, however, using analysis of the sounds in our natural languages and analysis of desired text. We created a program that synthesizes speech out of basic English text input.

Training

This project didn't require any actual training, but we did have to make recordings of all the Arpabet phones. To make these recordings, we looked at a table of Arpanet phonemes and had Noah reproduce them to the best of his ability while recording with Audacity. After several recordings were made, the most faithful sounding repetition of the Arpanet phoneme was selected and saved as a .wav file in our project directory. The recordings of these phones were later pieced together into a longer sound file intended to mimic a word or sentence. Converting from English text to Arpabet was another obstacle we encountered while working on this project.

Carnegie Mellon University's Pronunciation Dictionary was useful for this part of the project, as it contained pairings of most English words to their sequence of Arpabet phones. Using making use of the CMU pronunciation dictionary only required a simple parser to read through and construct a Python dictionary out of the English-Arpabet pairs in the CMU Pronunciation Dictionary. After this dictionary object was created, it was pickled for later use by our text-to-speech program.

Methods

The first thing we did in this problem is address the smoothing of phones in each word. We found that the best solution we could create was to use frame data to cut the head and tail off of each phone in a word. The algorithm that we used preserves the head of a phone at the beginning and tail at the end of a word, but jams the middle phones against one another to blend the sound.

English words that appear in our CMU dictionary are automatically converted into Arpabet to be synthesized into speech. Any non-words were instead passed through a series of regular expressions to arrive at an approximation of how it could be pronounced in English.

Additional Challenges

While working on this project, we ran into issues with pairs of English words that had different pronunciations but the same spellings. In the CMU Pronunciation Dictionary, these pairs of words are differentiated by the addition of numbers to the ends of the second and third words with the same spelling as another word. This method allowed multiple words with the same spelling to be included as keys in the CMU Pronunciation Dictionary, but it sometimes made it difficult for us to access the right pronunciation in the dictionary when a written word had multiple possible pronunciations. Our only strategy when dealing with these types of words involved deleting less frequently used words that were accessed instead of a more frequently used word from the CMU Pronunciation Dictionary and recreating our dictionary with CMU_parser.py.

Results

The results of this program are hard to quantify, because they depend on a user's understanding of the sound produced. However, it is fair to say that this program produces recognizable speech, especially when the text is also read along with the vocals. Another way to measure the capability of this program is how accurately it produces correct speech from unknown words. Using regular expressions, we were able to create a reasonably good representation of the unknown text in Arpabet, and thus, in speech.

Improvements

This program can be improved in two major ways. First of all, the smoothing of phones could be less choppy. Using a Fourier transform on the end of one phone on the beginning of the next, and averaging the values for those windows could produce a smoother transition between phones, which would allow the program to sounds better. Secondly, this program could be applied to any topic. Synthesizing speech from the text on a website or app would be a lot more useful than a simple speech synthesis program. We could also have the program train itself on a user's recorded phones to make a personalized speech synthesizer for each user. A more elegant solution to the challenges we ran into with words with the same spelling could have been developed, but this would have likely required the incorporation of some sort of parser into our project. Finally, a more effective strategy for dealing with non-words could have been implemented in the form of a more comprehensive list of English pronunciation rules or a model trained from the CMU Pronunciation Dictionary.