ABSTRACT

With the techniques developed for habitual disorders such as speech disorders, it would be useful to have a device with you to constantly monitor your behavior and give signals to indicate if the proper technique is being used or not.

In the case of stammering, such a device can be used to assist the speaker, by processing each word spoken, analyzing the features of the speech data, and outputting a signal or feed back to tell the user if the proper technique is being used or not.

Techniques such as Long Lengthening speaking, a technique developed by speech pathologist Partha Bagchi, is a well-known technique and **when used properly**, can enable a stammerer to speak fluently in *most* situations. More about the technique and how it works mentioned later. Given that the proper practices and guidelines are being employed by the speaker, the speech is nearly fluent, with minor blockages

Furthermore, after prolonged practice using this device, the technique becomes habituated/involuntary, and substitutes the faulty speech mechanism with the fluent one, at which point the device is not needed anymore.

GOALS

Develop and app on Android and iOS which can be used by stutterer to monitor the speech and ensure that the user is speaking fluently.

METHODS: Deep learning network using ADAM Optimization in TensorFlow framework in Python, and app development using Swift 4 and Android Studio.

Why do we need a device if the technique already guarantees fluency?

The only issue with employing such practice in real situations, is that:

1. people often forgot to use it,

2. are too embarrassed to speak differently

3. or are distracted due to stress or over consciousness

Furthermore, in stressful conditions, it becomes hard to focus on the technique, and users unknowingly end up using the wrong/partial technique without knowing.

Such a device will ensure that every word is spoken perfectly and will constantly remind the user to be conscious of it. After continuous practice, the speed of talking can be increased and the device is no longer needed.

TECHNIQUE:

EXPECTED RESULTS:

To have a trained deep learning model, which takes a each word spoken in real time (Wav -> numpy array), and outputs 0/1 (0 for wrong use, 1 for proper use), constantly giving feedback to the user.

To test this device with different people and finetune the hyper parameters. If the neural network does not over fit the data, and employs regularization, and has been trained with a data set of a large enough size, it will most work with multiple individuals, regardless of gender/ frequency of speech.

PLANS: (in Python environment)

As mentioned earlier, the input to the tf graph consists of a numpy array of values, each array corresponds to a word in the sentence.

Therefore to prepare the data we must convert an audio sentence in wav format —> numpy list of data , where number of rows = 1000 (number of nodes in the input layer) or number of samples in the audio word data.

DEPENDANCIES

- numpy
- scipy
- matplotlib
- pydub
- os
- wave
- pyaudio

Steps to prepare chunks of numpy "word data": *(All necessary python files and directories are found in project folder)*

Preparing the training data set, cleaning it for training.

1a. **get_word.py**: starts recording and stores sentences in wav format in "output.wav"

1b. **get_words.py**: takes "output.wav" and splits it into audio chunks corresponding to each word and stores in the folder all_chunks.

1c. **import_words.py**: takes all the audio wav chunks in "all_chunks" directory and converts them into numpy array, for further processing.

1d. **import_words.py:** contains func convertToMp3() to convert each wav chunk to mp3 and store in mp3_chunks. Also contains plotAll() to plot each numpy audio chunk.

Cleaning/processing of numpy audio chunk data.

1e. **clean_data.py:** soundData contains numpy list of values for all words, and is roughly 22000 samples in size. To remove redundant data, reduceDensity() is used to reduce it to roughly 1000 samples in size.

1f. **clean_data.py:** trimmer() is use to cut the start and end edges of the numpy word chunk, to make sure all chunks are of equal sample size.

1g. **clean_data.py:** weightedAverage() is used to extrapolate the noisy oscillating sound data for easier processing for tf model. (or smoothen the data)

1h. **envelope_data.py:** (Ongoing) uses RC filter to further smoothen the data.

2. Preparing The Tensorflow model (Ongoing)

2a. **Input layer:** 1000 nodes

2b. **Hidden layers:** 3 hidden layers *(with batch norm and dropout during training)*

2c. **Output layer:** 1 node (0/1) *(hardmax classifier to round off to 0 or 1)*

3. Tuning the Hyper parameters of model to improve accuracy of dev/test set

4. Developing App

5. Testing app and fine tuning.

6. Publishing and Documentation.

INFERENCES:

http://www.stammeringcurecentre.com/