Speech2Braille: A novel method to assist the hearing-impaired

Jacky Zhao, St. George's School

1. Background

Over 460 million people in the world have disabling hearing loss which can make day to day communication and life very difficult[1]. In particular, the deaf are not able to respond to auditory information like warnings and alarms as well as suffering from mental health issues which is serious problem. Current solutions that address these problems have pitfalls themselves that prevent them from being accessible. Lip Reading and American Sign Language are incredibly dependent on visual cues and take a long time to learn how to interpret them accurately.[2] Other, more technical solutions, such as hearing aids and cochlear implants, are prevented from being accessible because of their high purchase and maintenance costs and risky surgical procedures.[3][4]

2. Objective

The purpose of this project is to implement an embedded device that allows those who are hearing-impaired to have access to auditory information without visual distraction.

3. Proposition

For the hearing-impaired, the best way to receive information without visual interaction is through touch, or haptic technology. A novel device could be developed to transcribe speech to tactile messages to be recognized. To allow for ease of use and portability, the whole solution is designed to be compact enough to fit in an embedded device, eventually the size of a smartwatch. Through further research and development, it became clear that several technologies (speech recognition, haptic feedback device) needed to be integrated together to create a functioning solution.

4. Implementation

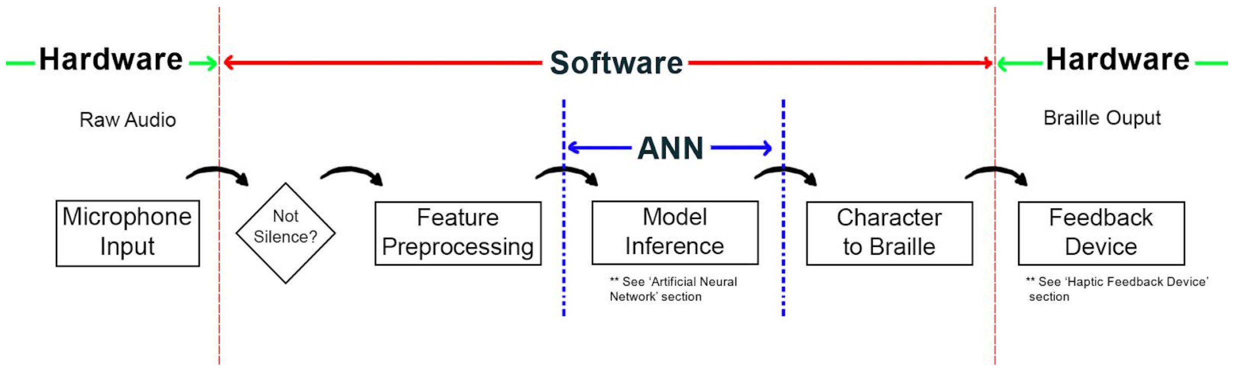


Fig 1. The end-to-end data flow from sound input to haptic feedback device.

The top-level functions that need to be implemented are:

1) Microphone input to audio file
2) Audio file to character sequence
3) Character Sequence to haptic feedback

4.1 Sound recording/Pre-processing

In a real environment, the target speech is always mixed with background noise. A noise removal technique has to be implemented to get large enough signal to noise ratio[5]. Taking into consideration aspects such as logarithmic frequency perception, an operation is performed to find the mel-frequency cepstral coefficients(MFCCs) of each sound. MFCCs take into account that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. which manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope[6]. This allows the program to represent complex speech with a simple array.

4.2 ANN-Based Speech recognition

For computers, recognizing speech is much more complex than simply identifying patterns and sounds[7]. Speech is extremely variable--different people speak in different ways. ANNs would allow us to tackle a lot of these problems by mimicking a biological neural network, allowing it to 'learn' how to deal with issues like variation in accents, pauses, tone, and volume[8].

4.2.1 Key Elements of implementing ANN

- **Code Framework**: Tensorflow
- **Platform**: Laptop with i7-6700HQ processor, 16G of RAM, and a NVIDIA 970M GPU

- **OS**: Ubuntu 16.04 x64
- **Datasets**: LibriSpeech Corpus, contains 100 hours of speech and is free to use[9].

4.2.2 Development of application programs

The programs coded to setup and train ANN models are as follows:

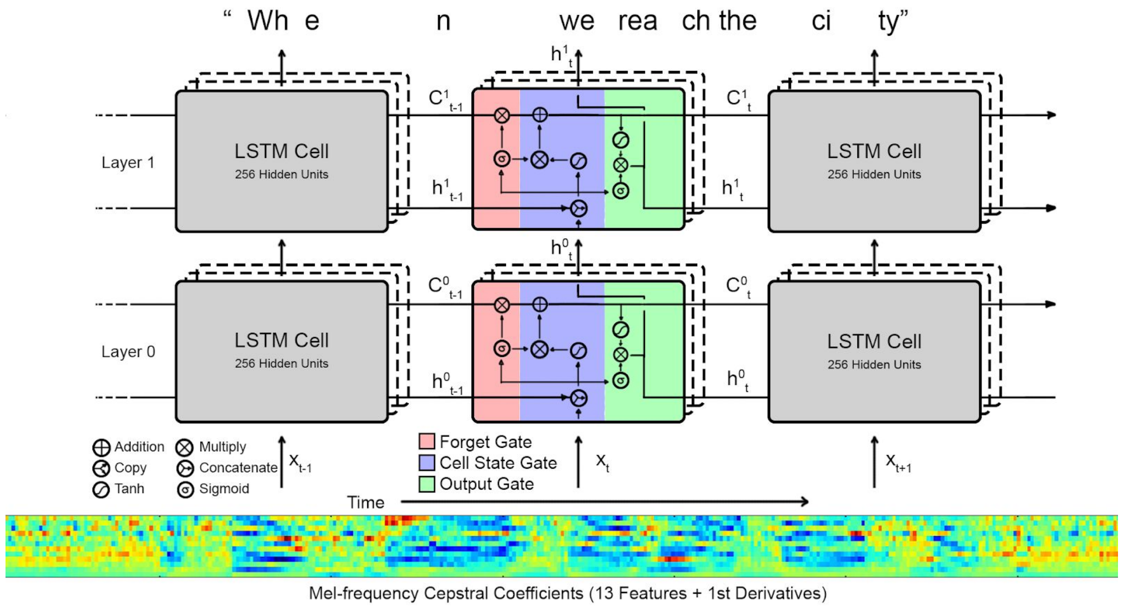
Name	**Description**
tf_model.py	Python script containing the network architecture construction and network training loop Has dataset loading and preprocessing functions
denoise.sh	Bash script that removes background noise from audio via sound profiling
braille_util.py	Python script that converts character sequences to a [2,3] array of on/offs. Also manages GPIO interfacing
gpiostates.py	Python script that manages the pilot light and state locking
init.sh	Bash script that serves as a master function that spawns necessary child processes to process data and gets an output
load_meta.py	Python script that loads ANN from saved model '.meta' and runs preprocessed data through the loaded model

4.2.3 Model Architecture Evaluation: After intensive research and evaluation, only the following key components are chosen to limit the complexity and size of the ANN for embedded applications

Feature	**Used**	**Reason**
LSTM - ANN cell body	Yes	The LSTM serves as the 'brain' of the ANN, processing all of the inputs. They are a subset of cells called RNNs that have 'temporal memory,' which allow them to excel at tasks like sequence labelling and speech recognition[10]. Having 2 layers and 256 hidden units allowed the LSTM to achieve an acceptable level of abstraction and accuracy without sacrificing too much run time.
CTC- Output and Cost	Yes	CTC serves as an output function and cost function for the ANN. Instead of a vector of possibilities, it outputs a sequence of labels expressed as probability distributions. CTCs are widely used in training LSTMs and RNNs, where they excel in cases like speech recognition, where timing is variable[11].

RMSProp – Optimizer	Yes	The optimizer in a neural network is responsible for updating the parameters in the ANN depending on the output of the cost function. It is based on the Adagrad optimizer, which adapts its learning rate based on how frequent parameters are being updated. RMSProp is different in which it tries to tackle Adagrad's radically diminishing learning rates[11]. An initial learning rate of 1e-4 and γ, which controls the rate of decay, was set to be 0.9.
Greedy Decoder - CTC decoder	Yes	Because CTC models its outputs on a sequence of probability distributions of all possible labels, I implemented the Greedy Decoder, which only searches for the most probable path. Compared to the Beam Search algorithm, this provides a substantial boost in runtime at the cost of a small loss in accuracy[13].
Batch Normalization	No	Added instabilities in the training process and crashed too often.
Regularization	No	Layer Normalization, Dropout, and L2 Regularization didn't improve performance as network never overfit the dataset.
Bi-directionality	No	The computational cost was too high as it requires too much memory.
FCLayers	No	The computational cost was too high when relative performance is taken into account
Different Optimizers	No	RMSProp, after extensive experimentation, worked better than other available ones like ADAM, ADAGRAD, ADADELTA, Momentum, and Stochastic Gradient Descent

4.2.4 ANN Data Flow Design and Implementation



There are 5 main steps that data go through when running data through an ANN.

1) Pre-processing

Compute 13 MFCCs, replacing first MFCC with RMSE, and 1st Derivatives into a numpy array of size [timesteps, 26]

2) Running the ANN

a) Run each timestep and concatenate it with the previous timestep's output
b) Forget Gate - Decide how much of the previous cell state to keep.
c) Cell Update Gate - Update the current cell state.
d) Output Gate - Get cell output.

3) CTC Decoder

Find the most likely set of sequence labels using CTC Greedy Decoder and return it as a character sequence

4) Compute CTC loss

Compares target sequence and predicted sequence and outputs cost

5) Optimize gradients

Update weights through the RMSProp optimizer based on the computed CTC Loss

5. Character Sequence to Haptic Feedback

Braille code was chosen to present haptic feedback devices due to it being small, easy to learn, and easy to implement. As a proof of concept, a 2x3 solenoid array was used to display tactile Braille. The character sequences are encoded into Braille using a short Python script, then sent to the GPIO pins to drive 6 solenoids. Additional Darlington transistors were also used to provide required voltage and current to drive the solenoids.

5.1 Embedded platform selection

As a proof of concept, the commercial Raspberry Pi 2B+ is used as the embedded platform for the following reasons:

- Linux software stack allows for easy transfer of pre-trained Tensorflow models
- Multiple GPIO ports for driving solenoids

6. Data and Results

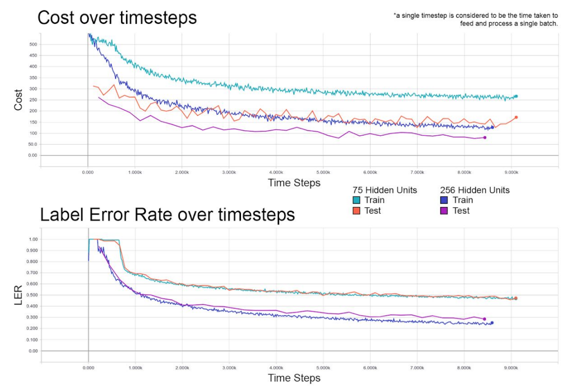
6.1 ANN Performance Metrics

Two different metrics were used to measure the performance of the ANN over the training period.

1) Cost - Defined as the negative natural log of the probability to correct classification of a label in every possible search path[11].
2) Label Error Rate - defined as the total Levenshtein distance over total characters of a batch[6][11].

The ANN achieved an accuracy of 74.77% on the training set and 71.50% on the test set after training for approximately 37 epochs or 44 hours and 10 minutes. This is comparable to Google's Speech Recognition API in 2013, which claims to have achieved an accuracy of 77.00%[14].

6.2 ANN Trends



Over the training period, there were a few notable observations.

1) The model underfits the data, showing high bias and low variance. The complexity of the ANN wasn't enough to capture the level of abstraction.
2) Because noise wasn't added to the test set, we can infer that the network becomes more robust to noise as training goes on as the performance of the train and test set converges over time.
3) The improvements to both the LER and cost functions decrease exponentially. Due to lack of available time and resources, the network training was stopped before overfitting.

7. Conclusion

A prototype speech to Braille embedded device has been successfully created by integrating and innovatively fusing:

- Various pre-processing techniques such as MFCC and de-noising
- ANN-based speech recognition on the Tensorflow framework
- Self-built solenoid-driven haptic feedback system displaying braille
- Embedded Linux based Software and Hardware environment

The ANN worked well for the time spent training and the complexity of the network. Accuracy could be improved in real-world applications by adding more random noise to the training process[15]. The hardware aspect served to be a functional prototype and demonstrated essentially everything that the end model needed to. Overall, this project was successful in accomplishing my original goal of creating an alternative method of receiving communication for the deaf that is independent of visual cues.

8. Appendix

8.1 Acronyms

ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long-short Term Memory Cell
CTC	Connectionist Temporal Classification
LER	Label Error Rate
MFCC	Mel-frequency Cepstral Coefficient
GPIO	General Purpose Input / Output
RMSProp	Root Mean Squared Propagation
RMSE	Root Mean Squared Energy

8.2 References

1. WHO. (n.d.). "*Deafness and Hearing Loss*." World Health Organization, World Health Organization, www.who.int/mediacentre/factsheets/fs300/en/.
2. Bauman, Niel. (2011). "*Speechreading (Lip-Reading).*" Center for Hearing Loss Help, hearinglosshelp.com/blog/speechreading-lip-reading/.
3. *"Cochlear Implants."* (2017). American Academy of Otolaryngology-Head and Neck Surgery www.entnet.org/content/cochlearimplants.
4. Center for Devices and Radiological Health. (2017)"*Cochlear Implants - Benefits and Risks of Cochlear Implants.*" U S Food and Drug Administration Home Page, Center for Devices and Radiological Health, www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/ImplantsandProsthetics/CochlearImplants/ucm062843.htm.
5. Lecun, Y., Bottou, L., Orr, G. B., & Müller, K. (1998). *Efficient BackProp. Lecture Notes in Computer Science Neural Networks: Tricks of the Trade*, 9-50. doi:10.1007/3-540-49430-8_2

6. Leeuwen, D.A., & Niedek, T.V. (2016). *Phonetic Classification in TensorFlow.*
7. Forsberg, Markus. (2003). *Why is speech recognition difficult.*
8. Sak, Haşim & Senior, Andrew & Rao, Kanishka & Beaufays, Françoise. (2015). *Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition.*
9. Povey, D. (n.d.). *LibriSpeech ASR corpus*. Dataset, www.openslr.org/12/
10. Panzner, M., & Cimiano, P. (2016). *Comparing Hidden Markov Models and Long Short Term Memory Neural Networks for Learning Action Representations. Lecture Notes in Computer Science Machine Learning, Optimization, and Big Data,* 94-105. doi:10.1007/978-3-319-51469-7_8
11. Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). *Connectionist temporal classification. Proceedings of the 23rd International Conference on Machine Learning* - ICML 06. doi:10.1145/1143844.1143891
12. Bohouta, Gamal & Këpuska, Veton. (2017). *Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx).* Int. Journal of Engineering Research and Application. 2248-9622. 20-24. doi: 10.9790/9622-0703022024.
13. Geiger, J.T., Rigoll, G., Schuller, B.W., Weninger, F., & Zhang, Z. (2014). *Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling.*

8.3 Bibliography

N., & A., M. *Neural Networks and Deep Learning*. (1970, January 01), Web-book http://neuralnetworksanddeeplearning.com/

Deng, L., & Liu, Y. (2017). Deep learning in natural language processing. Singapore: Springer Singapore.

Sak, Haşim & Senior, Andrew & Rao, Kanishka & Beaufays, Françoise. (2015). *Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition.*

Gurevych, I., & Reimers, N. (2017). *Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks*. CoRR, abs/1707.06799.

Hope, T., Lieder, I., & Resheff, Y. S. (2017). *Learning TensorFlow: A guide to building deep learning systems. Sebastopol: OReilly Media.*