SPEECH RECOGNITION SYSTEM

Isaac Mosebrook

University of Rochester Department of Audio and Music Engineering

ABSTRACT

One of the most prominent applications of audio signal processing in the modern era is speech recognition. When devices have the ability to perform tasks based on a user's speech, it greatly increases both the convenience to the user and the number of possible functions. For example, speech recognition could allow a user to make a memo using their voice on a device that previously had no manual keyboard input. For this project, the goal is to create a speech recognition system that can be interfaced with an embedded device.

***Index Terms*—** Speech Recognition, MFFC, ANN, Keyword Detection

1. INTRODUCTION

To implement the above application, we first have to construct a data acquisition system to record audio data that we can analyze. Once we have the input signal, we can split it into frames and analyze its features. These features will be the input to an artificial neural network that will output the spoken command as a text word. If the spoken word matches one of our commands, we can make a real world change. For example, if the spoken word is "on", we can turn on an LED.

2. DATA ACQUISITION

In order to analyze speech on any device, we first must construct a physical system that allows us to capture real world audio data. To do this, a microphone can be attached to the system. In this case I will use a MEMS microphone because they are the most common and powerful microphone type used in embedded speech recognition applications.

To turn the analog electrical signal from the microphone into a digital signal that a computer can read, we must send the microphone as an input to an analog-to-digital converter (ADC). For this project, I have chosen the Arduino Due as the microcontroller platform and a digital MEMS microphone which outputs a PDM signal. This allows us to connect the microphone directly to the Due. Additionally, the Due has a microcontroller which is where the signal processing will occur.

Instead of writing code to be run on the Due, we will instead use Matlab's built-in support package. This allows us to write Matlab code and process the data on a computer while retaining the ability to have analog inputs and outputs from the Due [6]. In this case, we will attach a single microphone to the Due which can be read into Matlab. We can also access the Due's built in LED to give the user visual feedback to their command.

Figure 1: Hardware Setup

3. FEATURE EXTRACTION

3.1. Windowing

Now that our physical system is in place, we are ready to perform signal processing on our speech. The very first step is to segment the incoming time-domain signal into frames [1]. We can do this by taking only the first M samples where M is the window length. We will perform our processing on these samples before looking at the next window. Additionally, we window these samples using a hamming window to reduce the noise generated by the multiplication of the window and the samples.

$$w(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right), \quad \alpha = 0.54,\ \beta = 1 - \alpha = 0.46$$

3.2. Fast Fourier Transform

Now we have our windowed time domain signal, but this representation is highly specific and will be difficult to compare to other speakers. To generalize the signal more, we can move it to the frequency domain by taking the

Fourier Transform of the windowed signal. Doing this for each audio frame will result in a spectrogram, visualized as follows:

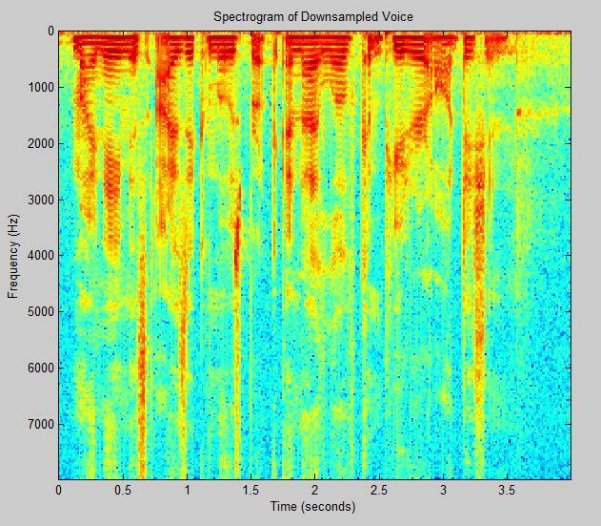

Figure 2: Visualization of Spectrogram

3.3. Mel-Frequency Filter Bank

Now we have reached a representation that is acceptably general, but we can still do better. This spectrogram represents all frequencies equally. However, humans do not hear all frequencies equally. The first step is to shift the frequencies in our spectrogram to a mel scale, which is defined as follows [1]:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) = 1127 \ln\left(1 + \frac{f}{700}\right)$$

This gives us our mel filter bank. Also note that the center frequency of each filter is where the previous filter reached zero. We can also use the bandwidth of the filter to determine the height of the filter by (2 / bandwidth). Each individual filter is multiplied by the spectrogram and the results are summed together for the final mel-frequency representation [1].

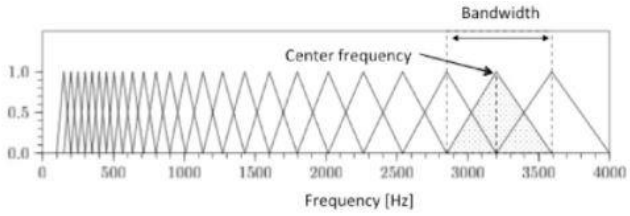

Figure 3: Mel Filter Bank

3.4. Logarithm

Beyond hearing frequencies at different intensities, humans also hear sound logarithmically. We can model this very simply by taking the logarithm of the value at each mel-frequency bin.

3.5. Discrete Cosine Transform

One last modification we can make is to take our signal to the cepstrum domain instead of the frequency domain. This domain will help us become even more general and more accurately observe pitch, which will help identify phonemes. We can do this by taking the inverse discrete cosine transform. The resulting vector is known as the Mel-Frequency Cepstral Coefficients (MFCC). This feature set is very good for representing phonemes [2].

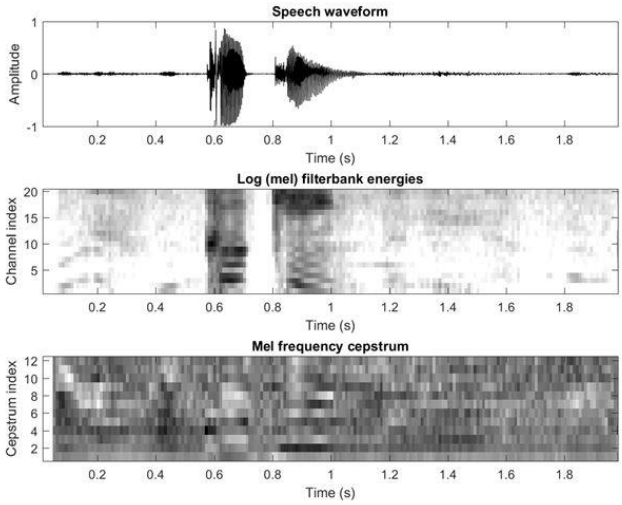

Figure 4: Visualization of MFCC

4. ARTIFICIAL NEURAL NETWORK

Now that the acoustic signal has been represented as a small feature set, we can use that as an input to our neural network. A neural network is a set of interconnected nodes that approximate the way a human brain thinks. The architecture is as follows:

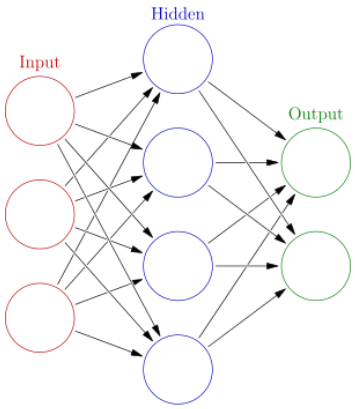

Figure 4: Visualization of Neural Network

Each column is considered to be a layer and each circle is a node. In this case, we want each node in the input to be given one value of a MFCC coefficient. At each node in all the following layers, the value is computed as a weighted sum of every node from the previous layer [5].

$$f(x) = Wx + b$$

Also note that after the sum at each node is completed, the output is biased and put into a sigmoid function to prevent values from getting too small or too large that they can no longer be read by the computer [4].

$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Each output node represents one of the possible words that the user could have said as well as one node for silence (or no spoken words). The final system can be visualized as follows:

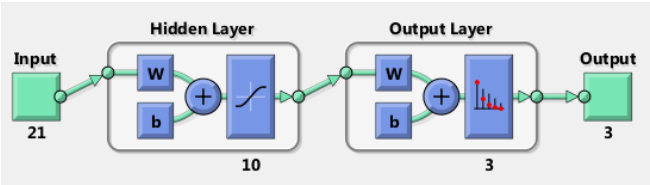

Figure 5: Diagram of Implemented Network

4.1. Training the Network

In order for the neural network to produce good results, it must be trained. This is accomplished by feeding it a known input. Based on the error between the experimental result and the known result, all the weights and biases can be adjusted to produce a lesser error for future tests.

The method by which this backpropagation is performed is called gradient descent optimization. Once the output of the known input is generated, the error between the resulting output and our desired output can be calculated [3].

$$E(y, y') = \frac{1}{2}\|y - y'\|^2$$

These errors in the output are used to find the errors for the previous layer, and so forth. This creates a gradient that can be optimized by a new set of weights and biases.

For this project, I did not create the code to train the network manually, but instead used Matlab's Neural Network Toolbox. Training a neural network is very computationally expensive and can take many hours to do. Therefore it is smarter to use a well-optimized library that can produce a great set of weights and biases.

Additionally, to create a network that produces great results, you need to train it with a set high quality recordings. For my training data, I utilized Google's speech command database, which has over 2000 recordings for different signal word commands such as "on" or "off" [7].

4.2. Preparing the Data

The first step to train this data was formatting it properly. Matlab's Neural Network Tool requires two matrices, one for the sample inputs and one for the expected outputs of those inputs. To do this, I looped through every file in the database. At each file, I obtained its first 21 MFC coefficients and stored them in the input matrix. At the same time, I also placed a 1 in the target matrix for the actual word that was spoken in that recording. In this case, the target matrix had dimensionality of three by the number of samples. The three dimension represented silence, on, or off as the guess for the spoken word.

Now that the data is properly formatted, we can train the neural network. Matlab allows us to easily train our network by selecting the inputs, targets, and training algorithm from a dropdown menu. Additionally, Matlab automatically divides the samples into training, validation, and testing groups. In this case, I always specified 70%, 15%, and 15% respectively. The network was trained using gradient descent backpropagation.

5. IMPLEMENATION

Now that the network has been constructed, we can begin to use it on real-time input. First we read our analog microphone input from the Due into a buffer until we reach a window size of 16000 (or one second of audio). This is preferred because the network was trained on 1 second long recordings at a sampling rate of 16000 Hz. Once we obtain our window, we find the MFC coefficients. This resulting

vector is forward propagated through the network using the formula described in section 4. We then can search our output vector for the location of the 1. The index of the 1 tells us which word has been spoken. If the word is found to be "on", we then tell the Due to set the pin connected to the LED high. If the word is "off", the pin is set low. If silence is detected, nothing happens. This process is repeated indefinitely.

6. RESULTS

The success of this system relies solely on how well-trained that neural network is. Therefore, different networks were attempted with the following varieties of parameters (confusion matrices available in Appendix A):

Parameter	System 1	System 2	System 3	System 4
Number of Samples	3406	406	3406	406
Number of MFFC	21	101	21	101
Training Method	Gradient Backpropagation	Gradient Backpropagation	Bayesian Regularization	Bayesian Regularization
Success Rate	75.6%	71.4%	89.3%	97.6%

From these results it is very clear that the most important factor in training the network is the training algorithm. Gradient backpropagation is the simplest method, so it makes sense that a more robust method like Bayesian Regularization would generate a more successful network. As far as the number of coefficients, it seems to generally not be that important, but different values should certainly be tested.

Overall, the system responds fairly well to real speaker recordings not in the dataset. The biggest issue with this project is that silence is frequently determined to be a word. The reason for this is that there were only 6 recordings of background noise. When using over one thousand samples for the words, background noise was removed as an option entirely during training. This should be a simple fix by obtaining equal amounts of background noise recordings for training.

7. FUTURE WORK

The type of Neural Network that I used in this project is the most simplistic version possible. There are many types of layers and modifications that can improve the performance of the network [5]. For example, a recurrent neural network would have some memory of past inputs to make better decisions on time sensitive matters, like speech. Also, you can add more hidden layers or combine multiple neural networks for different results.

One major obstacle to the performance of the system is the fidelity of the input recordings. There are several methods to improve the quality. Note that the goal here is make the speech recording as close to that of an anechoic chamber as possible. Two methods I would like to implement in the future are beamforming with a microphone array and active noise cancellation.

Another area of improvement for this project is expanding the functionality of the product. Currently, the only tasks the device can perform are turning the LED on or off. A good set of expanded tasks to start with may be speaker control with volume control, song selection, and some kind of audio enhancement DSP.

8. CONCLUSIONS

This paper outlined the steps to create a speech recognition system. It starts by converting an acoustic speech system into a digital signal via a MEMS microphone. Then the signal is windowed and its MFCC values are obtained. This feature set is used as the input to an artificial neural network. The net outputs a command or silence, and the system responds accordingly.

Voice recognition is an important tool that will become more and more common in modern devices. Existing products like this project such as the Amazon Echo or Google Home prove to be successful and will continue to grow in both performance and functionality.

9. REFERENCES

[1] Maiolo, Antonio. "Speech Recognition Wiki." *SR Wiki*, 2015, recognize-speech.com/.

[2] F. Zheng, G. Zhang, Z. Song, "*Comparison of Different Implementations of MFCC*", Journal of Computer Science & Technology, vol. 16, pp. 582–589, 2001.

[3] Rumelhart, D. E., Learning representations by back-propagating errors. *Cognitive modeling*. 1988

[4] Murphy A. Implementing Speech Recognition with Artificial Neural Networks. 2014.

[5] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury, "*Deep Neural Networks for Acoustic Modeling in Speech Recognition*". IEEE Signal Processing Magazine: 2012.

[6] Jonel Jozef B. Catapang1, Rionel B. Caldo*Implementation of Speech Recognition using MFCC for Plant Watering and Lighting System.* LPU-Laguna Journal of Engineering and Computer Studies Vol. 3 No.3, 2016

[7] Google. *Speech Commands Database*. https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.01.tar.gz

10. APPENDIX A

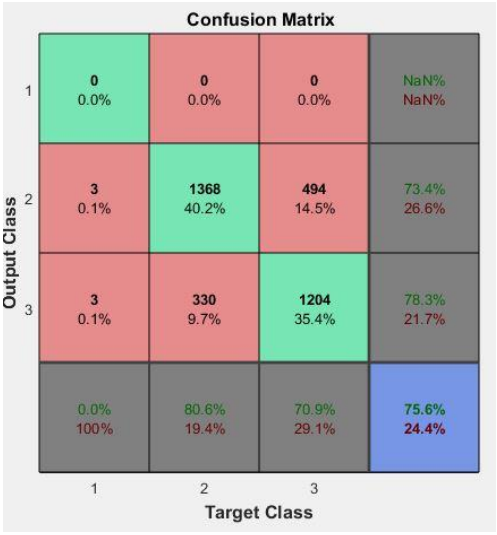

Figure 6: System 1 Confusion Matrix

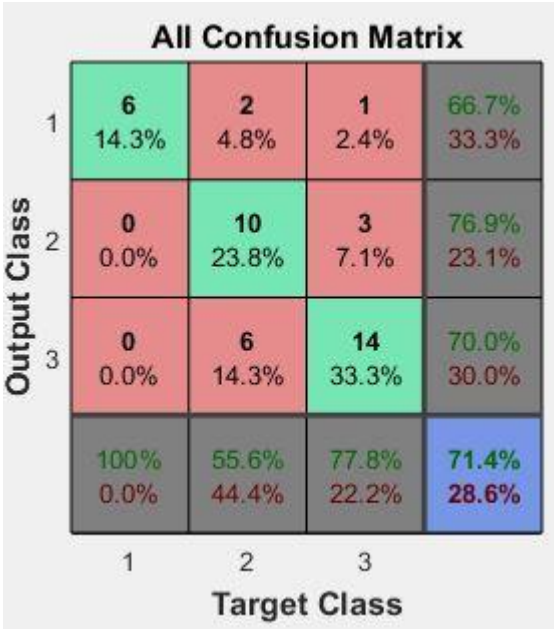

Figure 7: System 2 Confusion Matrix

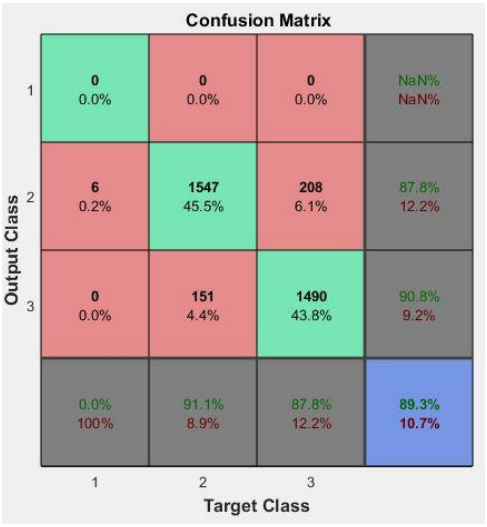

Figure 8: System 3 Confusion Matrix

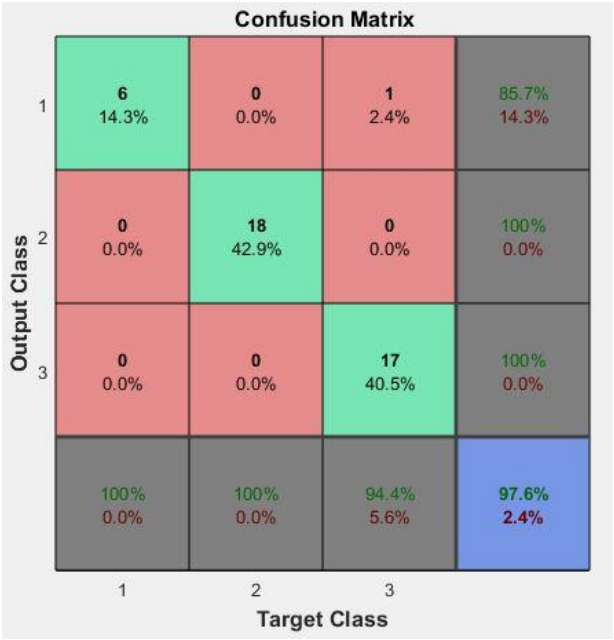

Figure 9: System 4 Confusion Matrix