Real Time Filler Word Detection in Conversational Speech

John Herrick

May 13, 2019

Abstract

Filler words such as “uh” and “um” are frequently used in everyday conversational speech. These fillers are undesirable and often times go unnoticed by the speaker themselves. The first step in reducing the amount of filler words said is identifying them as they are spoken. In this paper we present an implementation of a real time filler word detection system. This system can be used as a public speaking assistant to help people increase awareness of spoken filler words with the intent of eliminating them over time.
1 Introduction

Most speech is not scripted; it is spontaneous and arises in everyday conversation. As a result, it is far from perfect. It doesn’t necessarily make grammatical sense and sometimes the sentences aren’t well formed. One big difference between spoken speech and written words is the presence of filler words—words like “um” and “uh.” These words carry no semantic meaning and clutter the sentence. Ideas could be communicated more succinctly and effectively without filler words. It is very difficult to completely eliminate all filler words from our daily conversations because they’re so ingrained in how we speak. Many people don’t even realize they are using them or to what extent they are present.

There’s a public speaking game that exists to try and help with this problem of lack of awareness; it relies on audio cues. In a group setting if someone hears someone else use a filler word, they snap their fingers, clap, or make some other sort of auditory noise to let the speaker know. The filler word that would normally go unnoticed is immediately brought to the speaker’s attention. Playing the game causes everyone in the group to increase their attentiveness, especially the speaker themselves. This game works because it increases self awareness while speaking and this self awareness is critical for improving public speaking skills. One downside to this game is you can’t always have a group of people that can snap their fingers for you any time you say a filler word. Additionally, for those who fear public speaking, being under such scrutiny in that setting is a very stressful experience. For these reasons we decided to try and automate this public speaking game by giving the job of the audience to a computer. The computer takes in a stream of audio from the mic and makes an auditory cue such as a beeping sound any time a filler word is detected to increase the speakers self awareness.

Filler words are used when the thinking process cannot keep up with the speaking process and when the brain is still processing what to say [3]. The detection of filler words has additional use in speech dialogue systems because while filler words don’t serve semantic functions, they serve important other functions. One such function is the communicative function where a speaker uses filler words to keep the speaking turn while gaining time to think about the next spoken phrase. A listener often respects these filler words and does not interrupt the speaker. A dialogue system could use this information by inferring that the identity of the current speaker will not change immediately after a filler word is spoken. Another function of filler words is expressing cognitive functions. Instead of a speaker saying “hold on, let me think just one moment,” they can say “uh” and produce the same effect with less words. According to Goto et al., these filler words are “unconsciously used to express mental states such as diffidence, anxiety, hesitation, and humility as well as to express different thinking states, such as retrieving information from memory and seeking an expression appropriate for a listener” [3]. A dialogue system can therefore detect these filler words and use them as an insight into the speaker’s emotional and mental state.
1.1 Goals

Our goal for this system is to design and implement a program that will take in a stream of audio and recognize the filler words “um” and “uh” in real time. Upon recognizing these filler words the program will play a beeping sound to alert the user the filler words were picked up. Our tool is intended to be used as a public speaking assistant so the environment is assumed to be similar to a presentation setting with a singular speaker and minimal background noise. We decided that in order to run in “real time” filler words should be recognized at most a second after they are said. Finally our program should run on an average quality microphone such as a computer integrated microphone or a phone microphone so that the program could be widely used as a public speaking assistant on common hardware. Our technique is novel because this is the first time filler word detection has been attempted using a Wake Word system. We hope to determine whether this technique is more effective for this task.

2 Background

There are other technologies present in filler word detection, one of the most common being ASR. Below we discuss the history of ASR techniques and propose a new way to think of the filler word problem.

2.1 Automatic Speech Recognition

ASR exists in many commercialized forms such as Google Assistant, Apple’s Siri, and Microsoft’s Cortana. ASR enables the recognition and translation of spoken language into text. Before the deep learning revolution, speech recognition was done using Hidden Markov Models [1]. Since speech signals can be interpreted as a piecewise or short-time stationary signal, speech can be approximated as a stationary process. From this approximation, we can use a Hidden Markov Model to produce an output vector that gives a likelihood that a small window of audio contains a certain phoneme. By joining these outputs across time the sequence of phonemes and words can be extracted. The most likely utterance used to create the resulting sounds is discovered using the Viterbi algorithm, a dynamic programming algorithm commonly used to find the most likely sequence of hidden states in Markov models.

After deep learning took off neural networks have been the state of the art for ASR. LSTM (Long short-term memory) networks along with Recurrent Neural Networks (RNNs) have been very successful in speech recognition. Since 2014 there has been a large volume of research in “end to end” ASR. Alex Graves of Google DeepMind and Javdeep Jaitly from the University of Toronto created a Connectionist Temporal Classification (CTC) based system that consisted of RNNs with a CTC layer [11]. CTC is a relatively new objective function that attempts to predict the sequence of
phonemes without dealing with alignment. The basic idea behind the method is instead of feeding in alignments as input, only the labels need to be fed in and the model can try all possible alignments and learn from that. More recently, Chan et al. and Bahdanau et al. simultaneously introduced attention-based models [12] [13]. Attention based models pay special “attention” to different parts of the signals and do not have conditional-independence assumptions that CTC networks do. Additionally attention based models don’t need to be deployed with a language model. A language model is a probability distribution over a sequence of words and can be used to distinguish words and phrases that sound similar. For example, “The stuffy nose” sounds very similar to “The stuff he knows” and a language model can help determine which combination is more likely. Language models however are often very large and introduce a large amount of overhead. Because attention based models do not require a language model, they can be deployed onto applications with limited memory such as mobile devices. Attention based models are state of the art and have outperformed CTC models.

There are some commercial speech APIs by companies such as Google and Microsoft that have speech to text capabilities. Additionally, speech to text technologies are built into our mobile devices. One interesting thing about these speech to text models is that they ignore filler words. If you are dictating into your notes app and you say a filler word, it’s not represented in the text stream that comes out. Because of the fact that these models are trained to ignore filler words, using these APIs in an attempt to detect them does not work. It is likely that when these models were trained that even if there were filler words in the audio input, the desired text output did not have them. It would be possible to train a speech to text model including these filler words but there might be a better way to solve this filler word detection problem. Because the main focus is on the filler words themselves, the time and compute power to transcribe the rest of the words is not needed. This brings us to the topic of wake word systems.

2.2 Wake Word Systems

A wake word system is one like Alexa where you say “Hey Alexa” and the system wakes up. Cortana and Siri are other examples. A wake word system is fed a constant stream of audio and listens for a specific keyword or phrase to wake up. Over 39 million Americans have some sort of smart speaker and the rapid commercialization of this technology has led to a large amount of research on the topic. Amazon and Microsoft among others have pioneered new breakthroughs in wake word systems and recent papers “Time-Delayed Bottleneck Highway Networks Using A DFT Feature For Keyword Spotting” by Guo et al. [10] and “Monophone-based Background Modeling for Two-Stage On-device Wake Word Detection” by Wu et al. [9]. These two papers propose a new network architecture and noise reduction technique respectively. These commercial systems maintain high accuracy even while operating in noisy environments with other human speakers or music playing. Because these systems only really care about a specific wake word being said, we think they are a better solution that ASR
techniques for our particular problem.

3 Related Work

3.1 Matthew Lease et al.

Matthew Lease et al. presented the paper “Recognizing Disfluencies in Conversational Speech” which looks at different kinds of disfluencies in everyday speech [2]. These include repairs, filler words, and self-interruption points. Repairs are corrections made on the fly during a sentence such as “...let’s meet up Monday uh I mean Tuesday...” where a mistake was made in speech and is soon corrected. A self-interruption point is the point in which speech becomes disfluent due to the presence of a filler word or a repair. Lease et al. used a Tree Adjoining Grammar (TAG)-based model of speech repairs with a maximum-entropy reranker to determine where the repairs take place. They built this system previously and identified it had trouble identifying filler word because they usually occurred right before or after the repair itself. To address this issue they augmented their system with a set of manually constructed deterministic rules to detect fillers. Their model takes a transcribed sentence as input and constructs applies the TAG channel model after the sentence is tokenized. The TAG channel model defines a stochastic mapping of source sentences into observed sentences. A source sentence may contain self-repairs and fillers but in the observed sentence they are not present. They then identify candidates for possible repairs and refine these candidates with a syntactic language model and maximum entropy reranker. Finally deterministic filler rules are applied to identify the interruption points as well as filler words. This model works well for transcribed audio but in the real time setting it is forced to rely on automatic speech recognition (ASR). ASR would be the first step to process audio into a textual input for the model and errors in transcription would propagate through the entire pipeline. Additionally their model processes sentences one at a time and the runtime is of order \( O(n^6) \) which is not suited for real time analysis. Their paper did not talk about real time analysis so we’re not sure how much latency would exist if their system was forced to process a real time audio stream. In the end they had an overall error of about 19 percent for filler word recognition.

3.2 Masataka Goto et al.

Masataka Goto et al. presented a paper “A Real-Time Filled Pause Detection System for Spontaneous Speech Recognition” [3]. This paper describes a method to automatically detect filled pauses which are another name for filler words. The paper looks for continuous voiced sound of an unvaried phoneme to classify filler words and focuses on spoken Japanese. Standard Japanese filler words such as “ee”, “maa”, “eto”, and
“ano” fit the description of an unvaried phoneme. In English the two most frequent filler words “um” and “uh” also fit this description. To detect an unvaried phoneme they look at the F0 (fundamental frequency) of a speaker’s voice over time as well as spectral envelope deformation. An unvaried phoneme has an almost constant F0 as well as an almost constant spectral envelope. The spectral envelope is estimated using local information on the harmonic structure of the F0 score. Goto et al. showed that their system had a real time recall rate of 84.9 percent and a precision rate of 91.5 percent for detecting filler words on a Japanese spontaneous speech corpus. The two most common Japanese filler words are “eto” and “ano” which each have two syllables. In English the most common filler words “um” and “uh” have only a single syllable which makes recognition more difficult. They did not try their method on an English corpus but we imagine the accuracy would be lower because of the fewer syllables of the English filler words.

3.3 Gabrea and O’Shaughnessy

M. Gabrea and D. OShaughnessy presented a paper “Detection of Filled Pauses in Spontaneous Conversational Speech” which attempts to detect filler words using F0, zero crossing, energy, duration, and spectral estimation directly from speech signals in addition to an expert based system [4]. They identified filled pauses as a long and steady vowel with a low F0 relative to the average F0 for the speaker during the full conversation as well as a spectrum of a steady central vowel, often bordered by silence. They performed their experiments on the SWITCHBOARD dataset which is a corpus containing 2430 conversations averaging 6 minutes in length. The corpus contains over 240 hours of speech and is spoken by over 500 speakers across genders and races. They had three metrics to determine how well the model worked. The first is recall (RC) which is disfluencies detected / total disfluencies. The second is false alarms (FA) which is disfluencies / non-disfluencies. Lastly they had accuracy (AC) which is correct classifications / all data points. They presented a table of results with different recall rates as well as different false activations and accuracies. They defined two different cases where they used slightly different spectrums in classification.

<table>
<thead>
<tr>
<th>Case 1</th>
<th>RC</th>
<th>FA</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>71.43</td>
<td>1.44</td>
<td>98.07</td>
</tr>
<tr>
<td></td>
<td>85.71</td>
<td>1.83</td>
<td>97.81</td>
</tr>
<tr>
<td></td>
<td>92.86</td>
<td>2.36</td>
<td>97.43</td>
</tr>
<tr>
<td></td>
<td>92.86</td>
<td>3.67</td>
<td>96.27</td>
</tr>
</tbody>
</table>
As it can be seen from their results, there’s a close relationship between the recall rate and the false alarm rate. Increasing the recall rate for filler words comes at the cost of increasing the false alarm rate.

4 Methodology

Our methodology differs from previous methods by treating the problem as an end to end Wake Word problem as opposed to a feature based ASR problem. We also set forth a definition of what is acceptable for a “real time” which is left unclear in the previous literature.

4.1 Model

Our model architecture is based off an open source wake word listener called Mycroft Precise by the company Mycroft. Mycroft’s motto is “AI for everyone” and their open source wake word architecture design worked well for longer wake words and we thought it could be successful in the case of shorter filler words as well [7]. At its core our model is a single layer Recurrent Neural Network with twenty Gated Recurrent Units (GRUs). The output of the RNN is passed through a sigmoid which gives a prediction on whether a given segment of audio contains a filler word. To separate the audio into segments, we use a sliding window approach on a buffer of the most recent audio. As audio is read through the input stream it is added to a buffer. From the buffer we have a sliding window 1500 milliseconds wide that breaks up the audio into 1500 millisecond segments. The sliding windows heavily overlap and each sliding window shifts 50 milliseconds from one segment to the next. The oldest 50 milliseconds of the buffer are then discarded after each segment is taken.

4.2 Feature Extraction

After segmenting the audio into a 1500 millisecond window, we perform feature extraction using Mel-Frequency Cepstral Coefficients (MFCCs). MFCCs are commonly used in automatic speech recognition and can be used to extract features that focus more on the linguistic content instead of other extraneous sounds that may be picked up [14]. MFCCs are focused around the concept of the Mel scale.
4.2.1 Mel Scale

The Mel scale is a representation of how humans perceive the spacing of different frequencies. The mel scale was derived through experiments on human perception of sound by Stevens, Volkman, and Newman in 1937 [8]. Imagine listening to a sound wave at 100Hz and then imagine listening to another one at 200Hz. That difference you perceive between the two is the perceptual distance between 100 and 200Hz. If you were to repeat this with a sound wave at 1,000Hz and 1,100Hz, the actual distance between the two sound waves is the same but the perceived distance is smaller. In general as the sound waves increase in frequency, our ability to discern how far they are apart diminishes. Humans are better at perceiving differences in lower frequencies than higher frequencies. The mel scale normalizes these differences such that sound waves of 100mels and 200mels sound the same distance apart as two sound waves of 1,000 mels and 1,100 mels. Below is a plot of pitch in the mel scale versus pitch in the Hertz scale from the Wikipedia page on Mel scale.

Feature Extraction

Going back to Mel-Frequency Cepstral Coefficients, they are computed by first taking the fourier transform of a window of audio. The powers of the spectrum are then mapped on to the mel scale using triangular overlapping windows. We then take the log of the powers at each of the mel frequencies and lastly take the discrete cosine transform of the mel log powers as if they were signals. The MFCCs are the amplitudes of the resulting spectrum. The steps sound complicated but they attempt to simulate how human hearing works. The human ear has an organ called the cochlea which vibrates based on the frequency of incoming sound. Different parts of the cochlea vibrate depending on the frequency of sound being heard and that information is transmitted to the brain so doing the fourier transform of the audio breaks the audio into frequencies much in the same way. This is mapped onto the mel scale because this is how humans interpret pitches relative to each other and we take the log of
the powers because perceive volume on a logarithmic scale. In order to double the perceived volume of a sound, there needs to be around 8 times more energy put into the sound. Overall the MFCC produces very useful features and after generating these features we feed them into the 20 unit GRU and the final output is generated through a sigmoid. The model architecture is summarized in the figure below.

Starting at the bottom left, we have chunks which is how we quantize the audio from realtime samples into larger and more manageable pieces. Each chunk is 1024 samples of audio and we sample the audio at 16,000 samples per second. These chunks go into a buffer, along with the leftover audio from the previous chunks. A sliding window takes a 1.5 second long section of this buffer to be used for MFCC feature extraction. The window hops 50 milliseconds each iteration so hop_samples represents 50 milliseconds of audio. MFCC features are extracted according to the methodology above where they are then passed to the GRUs which finally feed into a sigmoid.
5 Dataset

For our implementation we used the Buckeye Speech Corpus which is a corpus of conversational speech published by Ohio State University. The corpus contains recordings of 40 people, each interviewed for around one hour. Of the 40 people, 20 are old, 20 are young, 20 are male, and 20 are female. All of the speakers are caucasian and long time residents of Columbus Ohio. The participants were told they were participating in a focus group on local issues and were only afterwards told the true purpose of the recordings.

The corpus contains around 300,000 words and the audio recordings are phonetically transcribed with the citation forms as well as the actual pronunciations of words. The phones spoken by the participants are also stored by their time stamp along with the word itself. The acoustic signal is clear and digitally recorded in a quiet room with no noise, recorded by a microphone close to the interviewee.

6 Training, Results, and Analysis

We went through many variations of how to extract the training data from the speech corpus. The training process was performed on all of these variations and we will discuss each variation below.

6.1 Validation

To determine how well a model works, we keep track of the recall rate and false positive rate. Recall represents what percentage of existing filler words our model recognized. False positives represent how often our model will classify a word as a filler word when it is not a filler word.

\[
Recall = \frac{\text{filler words detected}}{\text{total filler words}}
\]

\[
False \text{ Positive Rate} = \frac{\text{false positives}}{\text{total nonfiller words}}
\]

To gather these metrics we run our model on a stream of audio and record the timestamps where the model claims there is a filler word. We then take these timestamps and check the labeled transcriptions and check if there was a filler word said within one second of the model recognizing a filler word. If the model makes many predictions in quick succession (within one second) we cluster them as one prediction. To implement this, once the model recognizes a filler word, all recognitions are suppressed for the next second. This helps to reduce duplicate positives since speakers usually don’t use filler words multiple times in a single second. If there was a filler word in the transcription
within one second of the timestamp we recognized a filler word, we mark the word as recognized and if there was not we mark it as a false positive. Any filler words not recognized are false negatives and anything that is not a filler word that did not cause an activation of the model is a true negative. There is a close relationship between the number of true positives and false positives. By increasing the sensitivity of the model we will pick up more of the filler words and increase our recall rate but at the same time we’ll have more incorrect predictions as well which increases the false positive rate.

6.2 Training and Results

To train the models themselves we perform batch gradient descent with backpropagation using TensorFlow. The training examples are fed to the model along with the boolean label of whether or not the training example contains a filler word. To generate the training examples themselves we tried a few different methods.

6.2.1 Perfectly Aligned Word Segmentation

Initially we scanned through the buckeye corpus and segmented out every word exactly based on the labeled timestamps. Any words notated as “uh” or “um” were grouped as filler words while everything else was grouped as not a filler word. This means that the audio windows were perfectly aligned with the word boundaries such that there was no padding before or after the word. If a filler word starts at 1,000 milliseconds and ended at 1,150 milliseconds we take the 150 millisecond window as a new audio clip and that is a training example. Because filler words comprise a very small percentage of the Buckeye corpus, this generates around 6,000 filler word examples and 290,000 non-filler word examples. All of these examples were used for training data. After training the model for 3,000 iterations and analyzing at the resulting model, the false positive rate was over 20 percent and far too high. After some investigation we discovered the major issue was silence. The model would often flag long periods of silence as filler words. We concluded this was likely due to the fact that the training examples had no padding so the model never trained on silence itself.

6.2.2 Perfectly Aligned Word Segmentation with Silence

In a second iteration the training examples included silence and each labeled instance of silence in the dataset was segmented out and grouped in the non-filler word category. After retraining, the model performed much better on periods of sustained silence which greatly reduced the false positive rate but the filler word recall was low at around
only 35 percent.

6.2.3 Centered Word Segmentation with Context

In a third iteration we added more context to the training examples. Instead of just segmenting out the words at their strict timestamps, we segmented out 500 milliseconds before and after the filler word. This proved to be much more successful and raised the recall rate to around 48 percent. This increased the average length of training examples and gave context to the spoken words. This recall rate is still short of satisfactory however and we set to improve it further. In each time step the model outputs a probability that the window contains a filler word and when detecting filler words the probability output looks similar to a tight Gaussian curve with a brief spike in probability. This spike occurs when the filler word is in the center of the window of audio being considered. Instead of a brief spike in probability, it would be better if there was a higher probability for the full second after a filler word was said. With this in mind we tried another method of generating training examples. Below is a table summarizing the results of methods up to this point

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>False Positive</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aligned words only</td>
<td>30.82</td>
<td>23.53</td>
<td>75.73</td>
</tr>
<tr>
<td>Aligned words + silence</td>
<td>34.87</td>
<td>.08</td>
<td>98.86</td>
</tr>
<tr>
<td>Aligned words + silence + context</td>
<td>47.49</td>
<td>.13</td>
<td>99.02</td>
</tr>
</tbody>
</table>

6.2.4 Non-centered Word Segmentation with Context

Instead of just producing training examples with words centered in their segment of audio, we varied it such that the filler word could be anywhere inside the audio segment. We attempted to gather our training data in a way that would most resemble real time use. Real time use looks at successive 1500 millisecond segments and so for our training we used a 1500 millisecond sliding window over the training data. We slide the window 50 milliseconds for each new example. Because the windows are heavily overlapping this generates a large amount of training data. We trained this model on around 200,000 windows containing filler words and 400,000 windows containing non-filler words. We also hypothesized that since filler words are only 2 percent of the Buckeye corpus, if we increased the ratio of filler words to non-filler words in training then we could achieve a higher recall rate. With this new training method we actually generated 10 million examples of non-filler word windows but we only took a random subset of 400,000. Each segment was labeled as containing a filler word if the filler word was completely enclosed by the interval. Training on this dataset produced the most positive results and significantly increased recall rate. Depending on how sensitive we set our prediction threshold, we get different results. With a relatively low threshold
we get a recall rate of 84 percent so 84 percent of all filler words said are recognized. This gives us a false positive rate of 3 percent. If we set our prediction threshold to require more confidence, we get a recall rate of 78 percent with a false positive rate of 1.7 percent.

Below is a table summarizing the datapoints with different recall values

<table>
<thead>
<tr>
<th>Recall</th>
<th>False Positive</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>71.00</td>
<td>1.84</td>
<td>98.45</td>
</tr>
<tr>
<td>83.42</td>
<td>2.82</td>
<td>96.95</td>
</tr>
<tr>
<td>90.27</td>
<td>4.59</td>
<td>95.32</td>
</tr>
<tr>
<td>92.78</td>
<td>5.52</td>
<td>94.43</td>
</tr>
</tbody>
</table>

6.3 Analysis

Our model can be run on prerecorded audio snippets or accept a real time audio stream from the microphone. In the real time case, the program outputs a bar that represents the confidence that the most recently processed chunk is a filler word. If the model is confident enough that a filler word was said, it also makes a beeping sound to alert the user.

Our model catches the majority of filler words from people. From other testing it seems to struggle with thick accents and clapping and laughter sometimes flags as filler words. We unfortunately could not find a baseline in previous literature using the
Buckeye Corpus to compare our model directly against. M. Gabrea and D. OShaughnessy discussed in the related work section had a recall rate of 85.71 and a false positive rate of 1.83 on the SWITCHBOARD dataset which we were not able to obtain for this project. Masataka Goto et al. had a recall rate of 84.9 percent on the Japanese dataset and instead of reporting a false positive rate the reported a precision of 91.5 percent. This precision translates to a false positive rate of less than 1 percent. In terms of recall our model is similar but we have a higher false positive rate. This shows that our system is competitive with respect to previous literature. We theorize one factor explaining the very low false positive rate on the Japanese dataset is that the most common Japanese filler words are two syllables which makes them easier to detect.

7 Future Work and Analysis

As it currently stands our model has only been trained on “uh” and “um”. If this system is going to be used as an actual public speaking assistant it would be helpful to have a larger variety of filler words and to allow the user to choose some subset of them. Since we have a large dataset to work with already we could identify other types of filler words and train my model on those as well. With filler words like “so” and “like” however the task becomes more complicated because these words can be used in the filler word sense but also in normal speech. When you say “The candle was bright like the sun” you are using the word “like” but not as a filler word. Contrast that with “The candle was, like, bright”. In order to differentiate the two uses it is necessary to look at the semantic parse of the sentence. This is extremely difficult to do in real time and it would require ASR to process the words in the sentence up to that point. Sometimes it’s not even clear if a word is a filler or not until a few words after it was said. This would make real time detection much less real time. Either way our current model would not support these seemingly ambiguous uses and it would have to transition to a model blended with some sort of ASR.

Another thing we would want to do is make the model more robust towards noise. Currently the model was trained on data without any noise and as a result is not as robust when it comes to noise. Our feature extraction method from the audio using MFCCs is somewhat resistant to noise but there is room for improvement. We’ve looked a bit into this and to solve it we would add a new layer in between the live audio stream and the input to the model. This layer would take the audio stream and denoise it, getting rid of background noise while keeping the vocal characteristics. There is a technique called spectral elimination that has proven to work for this and getting that hooked up would likely entail adding a new RNN. With the new noise filter we could artificially noise our training data and pass it through both layers to train with noise and hopefully make our model robust.
One paper on this topic used a Japanese data set on conversational speech and while we did not use that dataset during this project it would be interesting to experiment with it. We noticed during my testing that our model performed best on people without any accents which makes sense since my dataset did not contain people with accents and in order to make it more robust it would be helpful to look for an additional dataset with more diverse speakers. Mixing languages in our training dataset would probably help a lot to fix that.

Lastly it would be beneficial to incorporate a video feed in addition to the audio stream to increase the quality of filler word recognition. A recent paper “Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation” by Ariel Ephrat et al. demonstrated that by combining a video stream with the audio stream allowed for state of the art audio segmentation. Even if two speakers were present in the video and speaking at the same time, their system would differentiate between the two and create two separate audio streams, devoid of background noise as well.

8 Conclusion

We have described our method for creating a real time filler word detection system by extracting features from an audio stream using Mel-Frequency cepstral coefficients and then using those features in an RNN to classify each audio window as either containing a filler word or not. We have more clearly defined what it means to run in real time and what is acceptable for such a system. We produced a recall rate similar to previous literature albeit with a slightly higher false positive rate. Lastly we discussed future work that could increase the accuracy of the system even further. From our tests it seems like Wake Word Detection techniques are in the same ballpark as ASR techniques. While ASR techniques can gain more information about the sentence as a whole, if the only goal is to recognize the filler words themselves then a Wake Word System can perform the same task with a simpler model and less computational power.

References


[12] Chan, William; Jaitly, Navdeep; Le, Quoc; Vinyals, Oriol *Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition*. ICASSP (2016)
