Automated Transcript Generation for Video Conferences

Nikolas Lee and Jia Wern Yong; {nikolas, jayyong}@stanford.edu

December 2017

1 Motivation

With the rise of Deep Learning, speech Recognition has shown great results in recent years. As people who use video conferencing software on daily basis, we aim to demonstrate the viability of generating meeting transcripts using speech recognition combined with a preprocessing step of identifying unique speakers in the meeting. If successful, participants in the meeting can focus on engaging fully in the conversation without spending effort on note taking.

2 Problem set up

Given that our application focuses on transcribing conversations in meetings, we expect the audio data at test time to be relatively clean with minimal background noise (with the exception of minimal static from poor internet connections etc.). For a meeting comprising of N speakers, we will collect N sets of audio for inference. Each audio sample will be a mixture of mic and system audio where the system audio captures what the other people are saying.

Our problem can be split into two main tasks. The first is a preprocessing step to identify every unique speaker present in the conference. The second is performing speech recognition on the preprocessed audio.

3 Unique Speaker Identification

3.1 Method

A clear way for us to approach this is to apply Independent Component Analysis (ICA). In ICA, given some data s in R $\in$ n that is generated by n independent sources, we have x = As, where A is an unknown square matrix called the mixing matrix. Our goal is to find $W = A^{-1}$, where W is known as the unmixing matrix. This way, we have s = Wx, and we will be able to recover our sources. In our case, we would be able to identify what each speaker said.

3.2 Preliminary Experiments

For the first part of the project, we needed a way for us to first split our sound data into the individual speakers, before putting the data through our speech recognition model. To make our training environment as realistic as possible, we need to ensure that our training dataset is as realistic as possible.

In order to help us to quickly perform ICA, we employed the FastICA library from sklearn decomposition[4]. For our first experiment, we wanted to test out the functionality, accuracy and correctness of the code that we have. We first got sound data from three separate individuals, where each individual is saying a sentence or two. We then mixed each sound data up with a 'fixed' mixing matrix A as a simulation. Below are the results that we got:

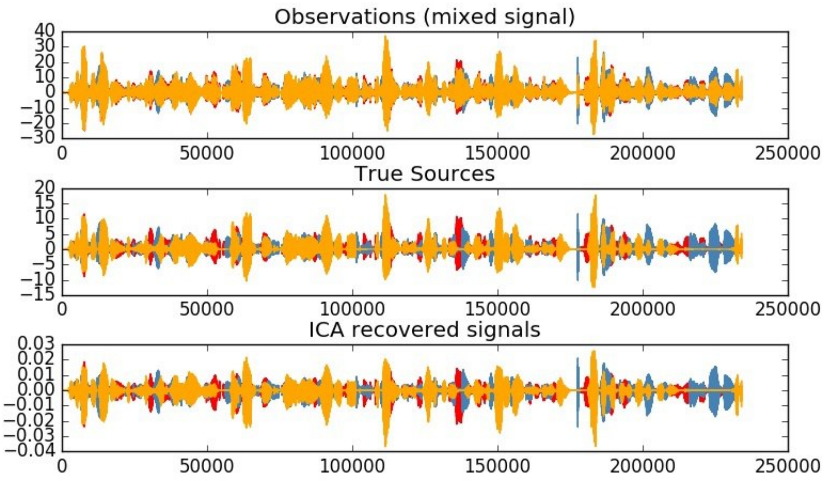

Figure 1: Sound signals for mixed signal, true sources and ICA signals

The graphs above show an arbitrary signal unit plotted against the corresponding frames. The three different colored graphs correspond to the sound wave of each of our three different sources. As seen, although the general shapes of the ICA recovered signals are not entirely the same as the true sources, the recovered signals closely resemble the true sources. We have also verified that our corresponding output sound files are audible.

So far, our data set consists of sound files of different individuals. We then simply performed ICA on sound files where there are multiple people/sources speaking. We plotted a similar graph. Below are the results that we obtained from using a data set with two people having a conversation.

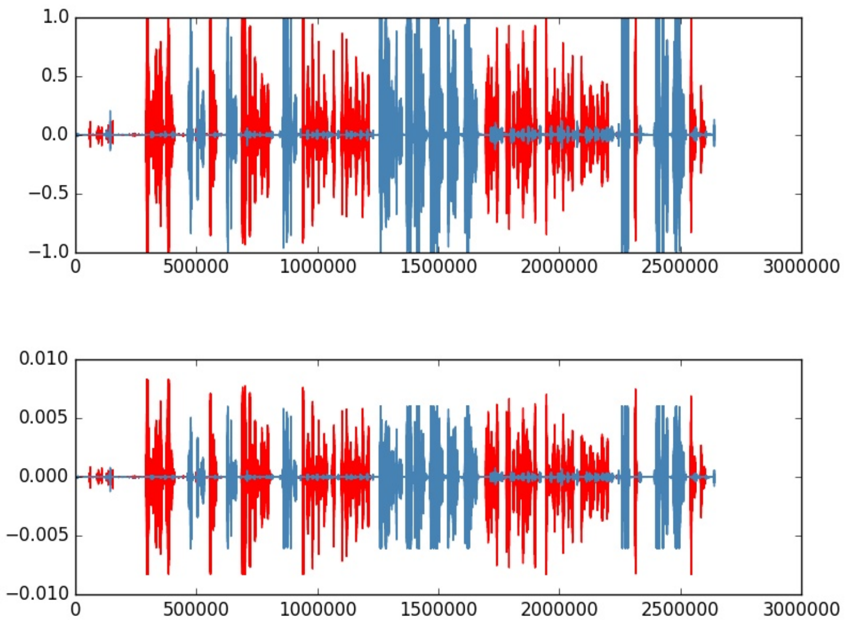

Figure 2: Top: Sound signals for mixed signal, Bottom: Sound signals obtained after separation

3.3 Discussion

While we were working on our project, we realized that we would need to create and train on our own data sets. The reason for this is because we needed to gather data where there are multiple people speaking at the same time to simulate a video conference setting. What we did was that we recorded our own conversations. As such, it was difficult for us to create a baseline for the ICA algorithm that we have. One way that we thought that we could potentially have created a baseline is by taking the output mixing matrix A and performing error analysis on that. However, because we created our own data set, we do not actually know what the true mixing matrix is supposed to be. Hence, looking at the mixing matrix A would not work.

Besides, it is also difficult for us to quantify the correctness of the ICA algorithm. So far, we usually had to 'check' for the correctness of the algorithm by listening to the output sound files. This was the best that we could have done so far given our limitations.

4 Speech Recognition

4.1 Method

A sound wave can be represented a N dimensional vector where N is the number of times the wave is sampled - sampling rate x time. The numbers are the height of the wave at the point being sampled. While getting a sound wave represented as an array of numbers is a step in the right direction.

Two problems immediately surface. Firstly, the vector in its raw form is very large and difficult to work with computationally. Secondly and more importantly, the length of the vector can vary.

To mitigate the first problem, we looked for feature extraction methods commonly used in signal processing. Two methods that caught our attention are 1) Calculating Mel Frequency Cepstral Coefficients 2) Using a Fast Fourier Transform and deriving a spectrogram.

The second problem is slightly more challenging. One way of working around this is to use models capable of taking in a sequence of inputs. Through our research we have identified the Hidden Markov Model and the Recurrent Neural Network which falls under deep learning. Between the two, the Hidden Markov Model is more traditional and has since been overtaken by the Recurrent Neural Network. While pitting the recurrent model against the hidden markov model as a baseline would be a natural approach, we decided to focus our efforts into learning how to implement the recurrent neural network, given that we did not know either of these models.

4.2 Preliminary Experiments

However, before diving into the realm of deep learning, we decided to use the techniques that we have learnt to observe first hand the feasibility of using machine learning to detect patterns in sound waves. Since we had access to the speech commands data set[3], whereby audio files of single words were nicely cut into the same length, we could use the first feature extraction method of calculating the MFCCs and trying to run a model on the features extracted from the MFCCs without worrying about unequal vector lengths.

After using MFCCs as a feature extraction step, we used a Gaussian Naive Bayes Classifier due to its computational ease. A Gaussian Naive Bayes is different from a standard Naive Bayes Classifier in that it assumes that its features are drawn from a Gaussian distribution instead of a Binomial/Multinomial Distribution, this allows it to be used on a data set with continuous features.

From our preliminary results. We obtained an accuracy of about 0.42. While this was disappointing, it was higher than simply guessing at random over 30 classes. We hence suspected that the model might have learnt something from the data although it cannot discriminate to the desired accuracy. To investigate further, we plotted a confusion matrix which would inform us where the bulk of the errors were:

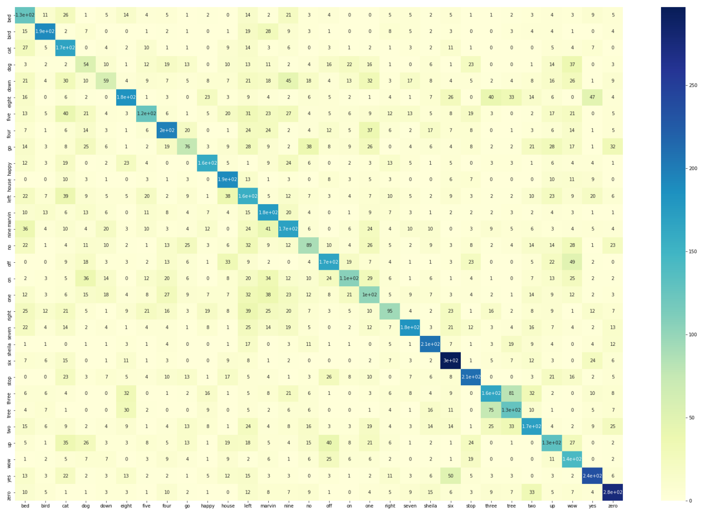
Figure 3: Confusion Matrix for the Speech Commands Data set

From the confusion matrix, we can see that the diagonal is distinctly more heavily coloured than the other cells, this means that the correct classes still hold the majority of the model's predictions.

While we could try improving the results of this experiments, we quickly realized that even a high performing word classification model would struggle for out application. This is because extra, non-trivial, processing would have to be done to handle these challenges: 1) The inputs and output are not constant. 2) The ratio of lengths between the input and output is not constant. 3) It is extremely difficult to specify how the elements in the input should correspond to elements of the output.

4.3 Connectionist Temporal Classification

Given that the ICA part of the pipeline was working as we hoped, all that was left was a good speech recognition system. In our research, we came across a paper by Baidu Research[1] which proposed an approach involving recurrent neural networks. Such a system would require a lot of data, much more than what could be found open source. Yet, we decided that it was better to use an algorithm that could scale with lots of data and computation and surpass non deep learning methods as we intend to continue improving the performance even after the course.

In Connectionist Temporal Classification (CTC), a sequence of audio is mapped to a sequence of characters. This is done without knowledge of how the input is aligned with the output in terms of length. This is convenient as it is difficult to specify how the alignment should be. For example, specifying a certain window of the sound wave to align with a character will not work for speakers with different rates of speech. CTC circumvents the alignment problem by simply assigning a token to every input step and collapsing the string of tokens. For a given X, it returns an output distribution over all possible Y's, which is used during inference to predict a likely output. To train the model, we maximize the conditional likelihood of the data with gradient descent.

4.3.1 CTC Algorithm

As mentioned earlier, CTC does not require knowledge of how the output is aligned to the input, instead it has its own way of producing an intermediate alignment which it later collapses into an output.

Forming the intermediate alignment is done by assigning a token to each input step. The tokens can be any character a,b,..,z,blank. The blank token accounts for periods of time where there is silence and hence no corresponding output. To collapse the alignment, we remove any repeated characters not separated by a blank token. This allows for repeated characters. For example, the algorithm might assign the sequence "HHHEE-LL-L-OO" to the input, where the "-" character represents the blank token. This sequence is then collapsed to "HELLO". The characters are chosen based on a distribution given by the network so using this per time-step output we can compute the probabilities of different alignment sequences.

To derive a loss function, we have a probability associated with each predicted alignment sequence. Many of these alignments might collapse into the same output so the algorithm marginalizes over all the alignment sequences and get a distribution over the outputs. However, marginalizing over the alignments is very expensive if performed naively. Dynamic programming allows the computation on the loss to be performed much faster by merging alignments which have collapsed to the same output at the same step.

During inference, we need to solve for the output that maximizes the conditional likelihood for the given input, this involves a modified beam search.

For our application, we used Mozilla's Deep Speech library[2] which implements the CTC algorithm.

4.4 Pre-trained Model

While we tried training a model with the open source data we had, we found that the Deep Speech project had a platform for getting labelled data contributed by the open source community. Furthermore, they had a pre-trained model using that data. We then used the weights learned in the pre-trained model for our application.

4.5 Results

While speech data is readily available on the internet, the data encountered in our application is a bit different. Mic and system audio are mixed together and passed through the ICA algorithm, this introduces a bit of noise and (soft) dialogue that is not intended to be picked up on.

To test our entire pipeline, we would have to create our own test data representative of the application environment. We made ten recordings of ourselves reading out a dialogue and averaged the WER across the recordings. we ended up with a poor WER of about 0.73 for the pre-trained model. We then proceeded to perform some error analysis and found that the top commercial speech recognition system from Google got around 0.48 WER.

We also found that the recordings that got poorer word error rates were due to the the recordings not being synced accurately, this resulted in poor performance from the ICA part of the pipeline and the input to the speech recognition being more noisy.

5 Next Steps

While speech recognition might perform well enough for most applications today such as voice assistants, a lot of work still has to be put in to turn automated transcript generation into a product. This is because the results are scrutinized by a human as compared to to a machine which can carry out a task by just approximating what was said. A full transcript generated from even a 30 minute meeting might be ridden with errors using the best speech recognition engines.

By choosing to use deep learning for the speech recognition system, we found ourselves getting poor results for the amount of data we got. However, we hope to come up with better ways of obtaining labelled speech data and allowing the model to get better performance.

Also, in collecting the test data, we encountered difficulty in synchronizing the start times for the recordings. We plan to build an application that handles this for us.

6 Contributions

For this project, Jia Wern Yong worked on the section on Unique Speaker Identification. He performed experiments on using the FastICA algorithm to simulate the separation of different sound waves from different sources. Nikolas Lee did conducted preliminary experiments and implemented the speech recognition portion.

References

[1] G. Alex. *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks.* ICML, 2006.

[2] Mozilla. *Mozilla Deep Speech.* https://github.com/mozilla/DeepSpeech.

[3] Google Research. *Speech Commands Data Set.* https://research.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[4] Scikit-Learn. *Blind source separation using FastICA*. http://scikit-learn.org/stable/auto$_{e}xamples/decomposition/plot_{i}ca_{b}lind_{s}ource_{s}eparation.html$.