Reconstruction of human voice for impersonation.

Final report

Amritha Raghunath (amrithar@stanford.edu)
Gunaa Arumugam Veerapandian (avgunaa@stanford.edu)
Vignesh Ganapathi Subramanian (vigansub@stanford.edu)

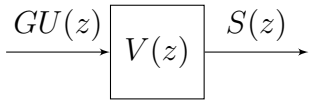
18 November, 2013

Introduction

The project we are doing deals with the idea of reconstructing of human voice. As we had mentioned in the project proposal, the aim is to learn the voice of an existing voice, and then to be able to convert any given sound input, into that particular voice. This idea consists of 2 voices, one of which is the source voice, and the second being the target voice. The target voice is the voice in which form we try to observe the required input. The source voice is the voice which contains the information that we need to have reconstructed.

Our implementation consists of three major stages: filter analysis, voice defiltering and voice conversion. The broad outline of each of these methods is as follows. In the first stage, we find using Machine Learning techniques such as minimizing the mean squared error, the components unique to a human voice, and this is what we call the human voice filter. In the second stage, we use speech signal processing techniques such as the Z-transform, to obtain the speech content from the given speech signal, by defiltering the unique voice content of the particular human voice. in the third and final stage, we now pass this defiltered voice into the human voice filter of the target voice, and obtain the final speech in the target voice.

A rough idea of this is given below in the form of a block diagram.



In the above block diagram, the filter $v[n]$ refers to the discrete time filter which models the human voice, and the filter $Gu[n]$ refers to the discrete time input which saves the words and other sounds in speech in some form. The output $s[n]$ consists of the exact speech samples recorded by us. Now, by some basic discrete time filter analysis ideas, we know that $s[n] = Gu[n] * v[n]$. This is the basis behind our three stages.

Linear regression techniques on a z-transform implementation.

In this section, we talk about a popular technique for voice conversion, which uses plain linear regression as a tool. Here, the filter is a plain filter employing the idea of a $z-$transform, and we use linear regression to determine the coefficients of this filter. This method has been explained in detail.

Filter Analysis

In this stage, we compute the feature values obtained by minimizing the mean-squared error. This is done as follows. We try to minimize the minmum mean-squared error of $s[n] - \sum_{i=1}^{k} s[n-i]a_i$, since minimizing this would leave us with the value of $Gu[n]$, which consists of the speech content in the speech signal. So, we try to minimize the value of $\sum_{n=1}^{N}(s[n+k] - \sum_{i=1}^{k} a_i s[n+k-1-i])^2$. This is done by the standard machine learning technique of getting the vector $a = (X^T X)^{-1}(X^T y)$, where X and y are obtained from trying to optimize the above summation. So, in this part, we obtain the values of a_i, which are our features, and the human voice filter is ready.

Voice Defiltering

In this stage, we first convert the various signals to their Z-transforms. In the z-domain, we get $GU(z)V(z) = S(z)$. From the previous section, we know that $V(z) = \frac{1}{A(z)}$. Therefore, we have $GU(z) = A(z)S(z)$. So, that gives us $Gu[n] = s[n] * a[n]$, and therefore, we are able to obtain the defiltered speech, which contains words and other related information.

Voice Conversion

This is the final stage of our project. Here, we use the fact that the defiltered speech of various people speaking the same stuff must be identical. That gives us $GU(z) = A_1(z)S_1(z) = A_2(z)S_2(z)$. Since we know $A_1(z), A_2(z)$ and $S_1(z)$, we can obtain $S_2(z)$, which effectively models the speech content in the target voice.

We have been able to obtain a dictionary of around 100 input voice samples. This dictionary has been replicated to compare with output observations. The linear regression output currently is extremely noisy, and we are trying to better it, or obtain a different machine learning algorithm which helps us minimize the mean-squared error all the same.

We implemented this approach, and observed that the output we obtain from this method is not only very noisy, but also has a strong influence of the source speaker on the target voice. One implication of this result is that the filter is not really a linear one. Also, we do not consider the effect of causality on the output voice. We try to implement these ideas in the following approach.

Auto-regressive techniques for Voice Conversion

This technique implements ideas of auto-regression on stationary time-frames. The auto-regression is also customized to the properties of the time-frame we consider. This is explained in detail below.

Stage 1 - Dynamic Time Warping

Dynamic Time Warping is an algorithm for measuring similarity between two temporal sequences which may vary in time or speed. In our problem, we use this algorithm to resize both sound sequences to the same size. When we are given the source and target training data, we use this technique to resize the size of the source and target data frames, to a uniform value, which is used for further processing purposes. Henceforth any usage of the words source and target sound frames would refer to the dynamically time warped versions of the frames.

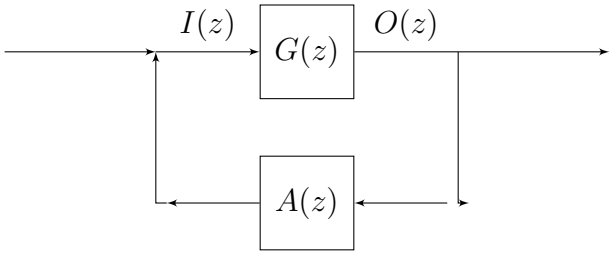
Stage 2 - $k-$means clustering

In our technique, we use the fact that sound samples are stationary for relatively small time frames. This is justified by the fact that for small time frames, of the order of 10ms, the sound does not really vary much. Each of the frames would have auto-regressive techniques performed on them, as would be explained in the next subsection. We try to cluster each of these frames into one of many cluster sets. This clustering is done by the $k-$means clustering algorithm. This is an unsupervised algorithm, on the dimension which is the number of frames. It is to be noted that this clustering process would take a fair bit of time, which would cause our training time to shoot up, but the advantage is that the training time would be amortized over our use of the machine for testing, and so it does not really matter to spend time on training.

The initial idea here was to use clustering based on Gaussian mixture models. This would have been extremely time efficient, but it turns out that the source speech is not normally distributed, and this sort of a model did not really provide us with an output matching expectations.

Stage 3 - Auto-Regression for time-frame

We use the auto-regression technique on stationary frames. The auto-regression technique proposes that output samples are dependent on a few previous output and input samples (p and q samples to be more specific). This uses a feedback from output to determine the future output samples. The following block diagram models the auto-regression in the $z-$domain.



The input is given by the input sound sequence $i[n]$, and the output is given by the output sound sequence $o[n]$. The above diagram can further be written in equation form as follows: $o[n] = \sum_{i=1}^{p} a_i o[n-i] + \sum_{j=1}^{q} g_j i[n-j]$. This is the auto-regressive equation we use for training and output generation on a stationary sound frame. In the above block diagram, the $z-$transform is the input transform filter is given by $G(z) = \sum_{i=0}^{q} g_i z^{-i}$. Similarly, the $z-$transform of the feedback transform filter is given by $A(z) = \sum_{i=1}^{p} a_i z^{-i}$. The effective transform filter from the input voice to the output voice is given by $\frac{G(z)}{1+A(z)}$. Our aim is to find these coefficients A_i, G_j for $1 \leq i \leq p, 0 \leq j \leq q$. This is planned to be accomplished by using linear regression on each of these time frames.

Stage 4 - Training phase

The training phase consists of two stages - clustering and obtaining coefficients. Once we are get the input samples of source voice $i[n]$ and target voice $o[n]$, the first stage requires us to cluster the input sound frames into a fixed number of clusters. This is done because if we do not cluster the input sound frames into a fixed number of clusters, the coefficients we obtain for each sound frame would be different, and storing all the coefficients would become an issue. Clustering is helpful because we need to store coefficients for only the various clusters, and not for every frame irrespective of its properties.

In this first stage, we do the clustering by using the plain $k-$ clusters algorithm. Each of the input vectors is a vector of size 200 (which keeps the time frame of length lesser than 10ms, thereby justifying our assumption of stationarity), and so the clustering algorithm is of dimension 200. This is time consuming, but justified in the second section where we talk about amortizing the time spent on training clustering over the various testing data.

The final stage of the training phase is to find the auto-regressive coefficients of each of the clusters. Each of the auto-regressive equations of each cluster (across time-frames) are put together in the form of a linear equation $X\theta = y$, where θ refers to the auto-regressive coefficients. This can be obtained by a least-squares approximation using the formula $(X^T X)^{-1} X^T y$, which is the solution to the linear regression equation.

This phase is crucial because we need enough data samples to obtain a tractable output. If the number of data samples are low, the output might have a noisy quality. The main idea here is the two-pronged approach to machine learning by initially applying a $k-$means clustering followed by linear regression to obtain the actual mime output. It is also to be noted that we do not influence the output in any other manner since the $k-$means clustering is an unsupervised learning method.

Stage 5 - Testing phase

The testing phase, similar to the training phase, consists of two stages. Once the source speaker's voice sample is obtained, it is first split into stationary time samples, as in the training phase. These stationary time samples, in the first stage of our testing phase, are detected to be part of a cluster, among the set of clusters obtained in the training phase. The cluster obtained is then further used to obtain the output in the next stage.

The second stage consists of predicting the output frame given the cluster the input frame belongs to. Once the cluster has been obtained, we pull out the coefficients corresponding to the cluster, and use it to linearly generate the samples which mimic the output.

Here, to obtain the first few samples, we pad the output vector with the previous output samples (this doesn't really matter since we are anyway considering stationarity to be present, and using p extra previous samples from the output shouldn't typically affect our output). This ends our testing phase, and the obtained output across frames can be concatenated to give the required output.

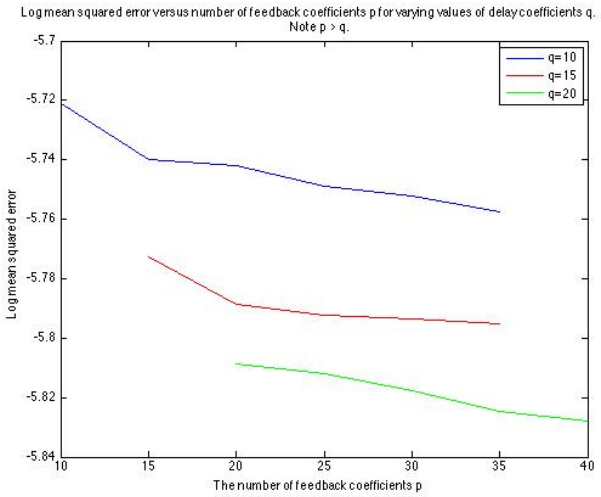
Analysis

The outputs obtained from the various techniques we employ have to be compared. The output obtained from the first method, which uses plain linear regression clearly has a very poor voice quality, which has a strong influence of the voice tone of the source speaker. Therefore, we do not provide any measures of comparing that with the actual speech of the target speaker.

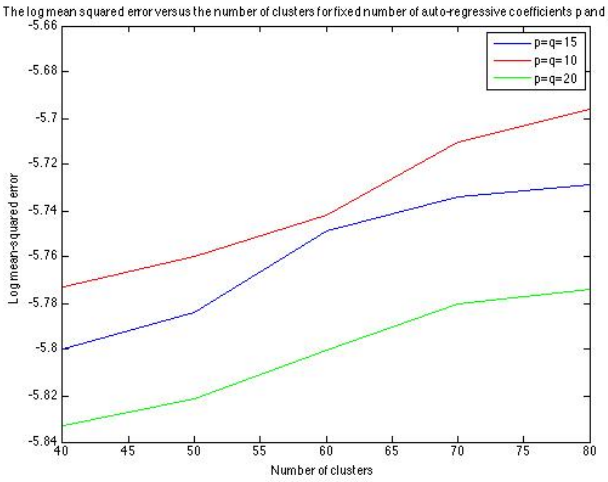
The output obtained from the auto-regression technique can be compared to the actual speech of the target speaker. This can be done as follows. We employ the log mean squared error of the actual speech to the target speaker as an error metric for measuring how well (or badly) our algorithm performs. This is given by $log(\sum_{i=1}^{N} \frac{(o[i]-\hat{o}[i])^2}{N})$, where $\hat{o}[n]$ is an estimate of our target speaker obtained through the algorithm, and N refers to the total number of voice samples of the output signal. This is expected to be as low as possible, and preferably lesser than 0, for a favorable output.

The two main features we use in our algorithm are p and q, the number of output and input delay elements affecting the output sample. In our problem, we try varying the values of p and q and observe how that affects the output observed, and the error metric.

The error metric, which is the log mean squared error, is varied with three parameters primarily, those being the number of clusters N, the auto-regressive components p and the delay components q. Figure 1a plots the log mean squared error varying with the number of delay coefficients q. As q increases, the log mean squared error reduces, and this makes intuitive sense, as we have more data to predict the future output samples, and that explains the reduction in error. Figure 1b shows the variation of log mean squared error with the number of clusters. The increase in error with number of clusters might at first seem unintuitive, but it is true because we seem to be overcorrecting for the number of phonemes. The ideal cluster value would be around 45, where we have all the phenomes



(a) Log mean squared error versus the number of feedback coefficients p for varying values of delay coefficients q.



(b) The log mean squared error versus the number of clusters for fixed number of auto-regressive coefficients p & q

Figure 1: Results of Analysis

covered for sure by the sample data, and we have also made up for any missed data in the 40 necessary phonemes.

Future work to be done

The machine learning algorithm has to be smoothened, so as to observe lesser noise in the output. This algorithm can be extremely potent if cleaned up so as to reduce the noise content in the speech. Another factor that has to be looked into is to better the voice quality. The current output sounds more like a growl in the voice of the target speaker, while it is aimed to get the target speaker's voice to be mimed as speaking properly.

The work could also be used eventually to break into voice recognition systems, and for that kind of an implementation, it is necessary that the output voice quality and timbre match identically that of the target speaker. For that to happen, we need to drastically tighten the algorithm up, so as to obtain a voice conversion system close to perfection.

References

[1] Ye, H. and S. Young (2003). "Perceptually Weighted Linear Transformations for Voice Conversion". Eurospeech 2003, Geneva.

[2] Ganvit,Y Lokhandwala, MA and Bhatt, NS (2012). "Implementation and Overall Performance Evaluation of Voice Morphing based on PSOLA Algorithm", International Journal of Advanced Engineering Technology