GPU-Friendly Local Regression for Voice Conversion

Taylor Berg-Kirkpatrick **Dan Klein**
Computer Science Division
University of California, Berkeley
{tberg,klein}@cs.berkeley.edu

Abstract

Voice conversion is the task of transforming a source speaker's voice so that it sounds like a target speaker's voice. We present a GPU-friendly local regression model for voice conversion that is capable of converting speech in real-time and achieves state-of-the-art accuracy on this task. Our model uses a new approximation for computing local regression coefficients that is explicitly designed to preserve memory locality. As a result, our inference procedure is amenable to efficient implementation on the GPU. Our approach is more than 10X faster than a highly optimized CPU-based implementation, and is able to convert speech 2.7X faster than real-time.

1 Introduction

Voice conversion is the task of transforming an utterance from a source speaker's voice into a target speaker's voice. The primary setup in recent work has been to learn this transformation from a parallel corpus consisting of recordings of the same sequence of sentences read by both source and target speakers (Stylianou et al., 1998). The converted speech is evaluated by how well its spectral properties match those of the target voice.

While various models have been proposed (Stylianou et al., 1998; Toda et al., 2007; Toda et al., 2005), the most accurate ones are non-parametric because the mapping between two voices' spectra can be highly non-linear (Helander et al., 2012; Popa et al., 2012). Unfortunately, while non-parametric methods are accurate, they are also slow – current non-parametric approaches to voice conversion are too compute-intensive for the real-time speed required by many voice conversion applications. In this paper, we begin with the state-of-the-art local linear regression (LLR) model used by by Popa et al. (2012) for voice conversion, and present a new GPU-based inference approach that greatly accelerates it, to much faster than real-time.

LLR, in principle, requires each new model prediction to be a function of the entire set of training examples. In practice, LLR depends most strongly on nearby points, so a standard CPU implementation will skip distant points, with limited loss of accuracy. A GPU cannot exploit sparsity in the same way (scan and skip) without suffering from memory bottlenecks, but even a GPU will be relatively slow if all training points are included in each computation. Our primary algorithmic change is to make use of a new sparsity structure that allows the GPU to skip major sections of the training data while still using dense memory access patterns on the points it does process. In experiments, this inference technique is more than 10X faster than a highly-optimized CPU-based implementation, operates almost three times faster than real-time, and is only slightly less accurate than the CPU-based method.

2 Background and Model

Most representations of speech that are useful for speech processing break the acoustic signal into separate components that represent the sound source (the lungs and vocal folds) and sound filter (the vocal tract) portions of the vocal apparatus. Work on voice conversion is generally focused on transforming the representation of the vocal tract. We follow this approach and learn a transformation of a mel-cepstral representation of the acoustic signal (Kawahara, 2006).

We treat the task as a multiple regression problem. In order to produce the transformed signal, we break the source signal into a sequence of frames, each of which is a 24-dimensional vector of mel-

cepstral coefficients. We denote a single frame of input mel-cepstral coefficients as x. In order to produce the transformed signal, we simply predict frame-by-frame. Specifically, for a frame x of the input we predict a transformed frame $\hat{y}$ as the mode of density $p(y|x)$, which we estimate from training data. A naive approach would be to use a linear model:

$$y = Ax + \epsilon$$

Here, ϵ is a Gaussian noise term, and the model is parameterized by the transformation matrix, A. For now, we assume training data are already frame-aligned (see Section 4). Let y_i be frame i of the target signal in the training data. Similarly, let x_i be the corresponding frame of the source signal in the training data. Thus, using this linear model, we would estimate the transformation as:

$$\hat{A} = \underset{A}{\operatorname{argmin}} \sum_i \|y_i - Ax_i\|^2$$

This very simple approach works, to some extent, but, because it cannot capture important non-linear relationships between x and y, it is far from state-of-the-art. A more popular approach is to use a Gaussian mixture model (GMM) to jointly generate both source and target cepstral features (Stylianou et al., 1998; Toda et al., 2007; Toda et al., 2005). This approach essentially learns different linear transformations for different regions of the input space, capturing some non-linearity. However, the GMM introduces a new problem: the posterior over the latent clusters learned by the GMM can be highly peaked (Popa et al., 2012) and as a result distortion is introduced by discontinuities at cluster boundaries. Thus, we adopt neither the simple linear approach nor the GMM. We instead the state-of-the-art fully non-parametric approach introduced by Popa et al. (2012). This method, described in the next section, learns transformations that capture non-linearity but vary smoothly as the input changes.

2.1 Local Regression

Like Popa et al. (2012), we use local linear regression (LLR) (Cleveland, 1979) to estimate $p(y|x)$. LLR is a non-parametric method that estimates $p(y|x)$ as a linear transformation that varies slowly with the input x. Specifically, $p(y|x)$ is estimated as follows:

$$y = A(x) \cdot x + \epsilon$$
$$\hat{A}(x) = \underset{A}{\operatorname{argmin}} \sum_i \left[w(x, x_i) \cdot \|y_i - Ax_i\|^2 \right]$$

The transformation $\hat{A}$ is a function of x, and is computed by solving a weighted least squares problem that depends on x. w is a kernel function that measures similarity between the current input, x, and each of the source training frames, x_i. We use a Gaussian kernel:

$$w(x, x_i) = \exp\left(\frac{-\|x - x_i\|^2}{2\sigma^2}\right)$$

Intuitively, for each input frame x we solve a separate least squares regression where each training datum is weighted by its similarity to the input. As the input varies, so will the weighting, and thus so will the linear transformation.

3 Inference

Exact inference using LLR is too computationally expensive for most applications since it means solving a least squares problem over the entire training set for each input frame x. A common approach is to define a neighborhood function that, for each input frame x, selects K training frames x_i that are most relevant to x. Then, $\hat{A}(x)$ is computed by only solving the least squares problem over this neighborhood. This approach can work well in practice since the support for each local least squares problem is relatively sparse. By choosing the right neighborhood function, work can be skipped without substantially impacting learning.

3.1 Inference on the CPU

The standard approach when using a CPU is to let the neighborhood function pick out the indices of the K training frames x_i that maximize $w(x, x_i)$. We let this particular neighborhood function be called $g(x)$, depicted in Figure 1. Using this approach, for input frame x, $\hat{A}(x)$ has the following closed form expression:

$$\hat{A}(x) = H(x)^\top W(x) G(x) \Big(G(x)^\top W(x) G(x) \Big)^{-1}$$

$G(x)$ is a matrix formed by appending the training source vectors in the neighborhood of x. More specifically, $G(x)$ is a matrix that has the vectors $x_i^\top$ s.t. $i \in g(x)$ as its rows. Similarly, $H(x)$ is a matrix that has the vectors $y_i^\top$ s.t. $i \in g(x)$ as its rows (in the corresponding order). $W(x)$ is a matrix that has the weights $w(x, x_i)$ s.t. $i \in g(x)$ along its diagonal (also in the corresponding order).

This approach is much faster than the exhaustive method, but at typical audio sampling frequencies, inferring the transformation of the signal for an entire sentence can take over 30 seconds on a modern CPU, which is too slow for real-time conversion. In order to transform the signal for a new sentence, $\hat{A}(x)$ must be computed for each frame. This means that for each frame x, the distance to all training source frames must be computed, neighborhood matrices $G(x)$ and $H(x)$ must be formed, followed by several matrix multiplies, an LU decomposition, and a triangular solve operation.

3.2 Inference on the GPU

The computation of $\hat{A}(x)$ for a block of multiple input frames can be done in parallel if a fixed amount of lag is tolerated in the conversion process. The parallel computation of large number of small dense matrix operations (multiply, LU decomposition, triangular solve) is a perfect fit for implementation a GPU, which can achieve vastly more throughput than modern CPUs can. However, using the CPU's neighborhood structure on the GPU has a crippling bottleneck. The extraction of the neighborhood matrices $G(x)$ and $H(x)$ from the training data requires a large number of memory accesses that are effectively random. The indices of the closest K training source vectors to an input x are generally non-contiguous. As a result, the K vectors in each of the neighborhood matrices must be copied with separate memory accesses, and since random access time on modern GPUs is very slow, extraction becomes a bottleneck. In initial experiments, we found that when this approach is implemented on a GPU it is even slower than the CPU-based implementation.

In contrast, memory bandwidth is extremely high on modern GPUs. Thus, if it were possible to order the training vectors x_i and y_i in GPU memory such that neighborhood matrices $G(x)$ and $H(x)$ were composed of contiguous blocks of training vectors, extraction on the GPU could be made very efficient. Unfortunately, with the current definition of the neighborhood function, g, such an ordering does not exist. Therefore, we define a new GPU-friendly neighborhood function, $\tilde{g}$, for which an ordering that permits contiguous extraction does exist.

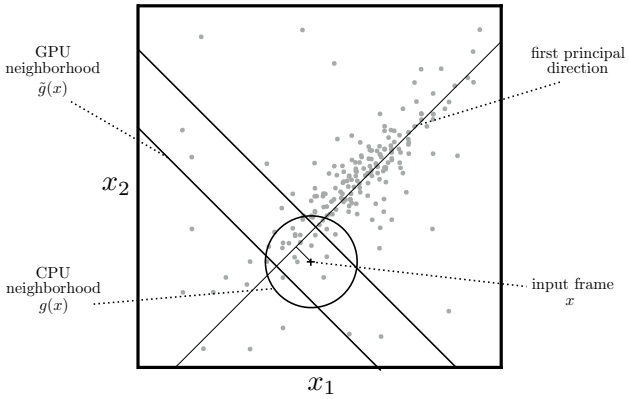

Figure 1: Depiction of standard neighborhood function $g(x)$ used for local regression on the CPU and surrogate neighborhood function $\tilde{g}(x)$ used for inference on the GPU, plotted for two-dimensional input data.

Let $u(x)$ be the projection of source vector x onto the first principal component resulting from running PCA on the source side of the training data. Now, we define $\tilde{g}$ as follows: $\tilde{g}(x)$ is the set of K indices for which $|u(x) - u(x_i)|$ is the smallest, or, in other words, the set of training indices with source projections closest to the projection of the input frame. By ordering the training data by their projection onto the first principal component, we can ensure that $G(x)$ and $H(x)$ are contiguous in memory.

The hope is that this approach yields substantial speedups on the GPU without negatively impacting the learned transformation (see Section 4). Figure 1 depicts the difference between the CPU neighborhood function g and the GPU function $\tilde{g}$. Intuitively, when most of the variance in the training data occurs along the first principal direction, the CPU and GPU neighborhood functions may be similar since the distance between projections is a good proxy for distance in the original space. The weighting function, w, is still computed in the original space, so distant training vectors that are inadvertently included in the neighborhood will be severely down-weighted. The potential pitfall is that for a fixed neighborhood size, $\tilde{g}$ may be less efficient at collecting training points that are relevant to the input.

System	Scottish Male to US Female			US Female to Scottish Male		
	CD	Time	Frac. RT	CD	Time	Frac. RT
GMM	7.05	1.1	3.7X	7.01	0.7	3.9X
CPU LLR	5.99	12.2	0.3X	6.51	8.75	0.3X
GPU LLR	5.98	1.4	2.7X	6.55	0.9	2.9X

Table 1: Voice conversion results for the GMM baseline system, CPU-based local linear regression baseline system, and the GPU-based local linear regression method. The cepstral distortion (CD), average inference time per sentence in seconds (Time), and fraction of real-time (Frac. RT) are shown. Smaller cepstral distortion corresponds to more accurate transformations and fractions of real-time that exceed one imply faster than real-time operation.

4 Experiments

We run a series of experiments to determine whether our GPU-based inference technique offers speeds-ups and at what cost to accuracy.

Baselines We compare our LLR-based conversion system that performs inference on the GPU (using the GPU-friendly neighborhood function) with two different baseline systems. The first baseline system also uses LLR, but performs inference on the CPU using the standard neighborhood function. The second baseline is the GMM model of Toda et al. (2007), which is known to be fast and is widely used in practice. The size, K, of both CPU and GPU neighborhoods was set on a development data to the smallest value that did not show degraded performance compared to exact local regression.

Implementation We implemented our GPU-based LLR technique using the CUDA API (Nickolls et al., 2008), and the CUBLAS API which contains bindings for GPU BLAS routines. We ran the system using an NVIDIA Tesla K40c GPU. We built a multi-threaded implementation of CPU-based inference for local regression using calls to CPU BLAS routines, and ran this system on a 4.4GHz 4-core Intel CPU.

Data We train and test on a portion of the CMU Arctic database. The training data consists of 70 sentences spoken by both a US female speaker and a Scottish male speaker. The testing data consists of 20 sentences spoken by the same two speakers. We give results for converting in both directions, from the female voice to the male voice, and from the male voice to the female voice.

Frame Alignment Since the source and target speakers speak at slightly different rates, our training data consist of different numbers of frames for each training sentence. We use dynamic time warping to induce the frame alignment. Specifically, we find the minimum cost monotonic alignment from source frames into target frames where the cost of each alignment edge is the L2 distance between the corresponding vectors. We use a distortion limit of 2, and a linear distortion cost.

Analysis and Synthesis We use the CMU implementation of the STRAIGHT analysis and synthesis methods introduced by Kawahara (2006). This is the same method used many state-of-the-art voice conversion systems, included our GMM baseline of Toda et al. (2007). We transform the top 24 cepstral coefficients using our system, but process the power coefficient and fundamental frequency separately, using simple transformations for the latter two components.

Evaluation In order to evaluate the accuracy of our model we measure the cepstral distortion between the predicted ceptstral frames $\hat{y}$ and the actual cepstral frames for the target voice y. The cepstral distortion is calculated as follows:

$$\text{distortion}(\hat{y}, y) \propto \|\hat{y} - y\|$$

We using dynamic time warping to align the predicted frame sequence to the target frame sequence.

4.1 Results

The results of our experiments are displayed in Table 1. For the Scottish male to US female and the US female to Scottish male transformations the systems that use local regression outperform the parametric

GMM baseline in terms of average cepstral distortion. The GMM baseline, is however, the fastest of the compared systems. For both experiments, it is able to produce transformations substantially faster than real-time. The CPU-based local regression baseline achieves the best overall cepstral distortion, but is also the slowest method. In both experiments, it operates at 0.3X real-time speed. The GPU-based local regression method performs only slightly worse overall than the exact method in terms of cepstral distortion, yet in both experiments it operates substantially faster than real-time, nearly as fast as the parametric baseline.

5 Conclusion

We have demonstrated a method for substantially speeding-up inference using a non-parametric estimator for spectral voice conversion. Related approaches may prove useful for making non-parametric estimators more efficient in other areas of speech and language processing.

Acknowledgements

This work was partially supported by BBN under DARPA contract HR0011-12-C-0014 and by an NSF fellowship for the first author. Thanks to the anonymous reviewers for their insightful comments. We further gratefully acknowledge a hardware donation by NVIDIA Corporation.

References

William S Cleveland. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836.

E. Helander, H. Silen, T. Virtanen, and M. Gabbouj. 2012. Voice conversion using dynamic kernel partial least squares regression. *IEEE transactions on audio, speech, and language processing*, 20(3):806–817.

H. Kawahara. 2006. Straight, exploitation of the other aspect of vocoder: Perceptually isomorphic decomposition of speech sounds. *Acoustical science and technology*, 27(6):349–353.

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable parallel programming with cuda. *Queue*, 6(2):40–53.

Victor Popa, Hanna Silen, Jani Nurminen, and Moncef Gabbouj. 2012. Local linear transformation for voice conversion. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4517–4520. IEEE.

Yannis Stylianou, Olivier Cappé, and Eric Moulines. 1998. Continuous probabilistic transform for voice conversion. *Speech and Audio Processing, IEEE Transactions on*, 6(2):131–142.

Tomoki Toda, Alan W Black, and Keiichi Tokuda. 2005. Spectral conversion based on maximum likelihood estimation considering global variance of converted parameter. In *ICASSP (1)*, pages 9–12.

T. Toda, A.W. Black, and K. Tokuda. 2007. Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(8):2222–2235.