Applying a Recurrent Neural Network using Connectionist Temporal Classification to Automatic Recognition of Lyrics in Singing

Maneesh Apte, Matthew Chen, Teddy Morris-Knower, and Shalom Rottman-Yang
Stanford University
Stanford, CA 94305
[mapte05, mchenja, ted107mk, jaronry]@stanford.edu

Abstract

We apply techniques originally developed for Automatic Speech Recognition (ASR) towards the more challenging task of Automatic Recognition of Lyrics in Singing (ARLS). Our system is based on a deep bidirectional recurrent neural network architecture and the Connectionist Temporal Classification objective function. Due to a lack of suitable song datasets for training data, we train the system on speech utterances transformed to more closely resemble singing. Testing the system on monophonic singing (i.e., isolated vocals) shows that it successfully understands words from the training set and can apply them during testing, but it requires a significant amount of training data and time in order to achieve superior performance.

1 Introduction

Commercially produced music has become more widely available in recent years, and contemporary streaming services have catalogs containing tens of millions of songs (Digital Trends, 2017). Unfortunately, algorithms for music information retrieval lack the performance needed to maximize the functionality of such well-stocked catalogs. For example, querying a catalog for the set of songs containing the terms “Rick” and “Astley” is excessively slow in the absence of orthographic transcriptions for each of the catalog’s songs. Such transcriptions are manually produced or risk lacking sufficient quality.

However, information retrieval algorithms achieve exceptionally high performance when applied to text. As such, an ARLS system that produced sufficiently accurate orthographic transcriptions would facilitate music-specific implementations of algorithms like WAND (Broder et al., 2003), which are used to quickly query collections of text but do not work with songs that lack orthographic transcriptions.

Unfortunately, the literature suggests that ARLS systems require considerably more development before they achieve sufficient accuracy for the above-mentioned algorithms. While modern ASR achieves a word accuracy of 95.1% (VentureBeat, 2017), Mesaros and Virtanen (2010) note that their ARLS system is limited to an accuracy of 12.4% when performing word recognition on monophonic music. More recently, Kawai et al. (2016) report that their system achieves a word accuracy of 32.9% on (Japanese) monophonic music.

In this paper, we present a system which produces orthographic transcriptions from monophonic singing. This system is based on a deep bidirectional recurrent neural network architecture and is trained using the Connectionist Temporal Classification objective function, facilitating removal of a common source of human error from the transcription process.

The rest of this paper is organized as follows: In §2, we review the literature which guided our system design. In §3, we describe our processes for obtaining, evaluating, and transforming the datasets used to train our system. In §4, we present the details of our approach, and in §5, we analyze its performance. We conclude the paper in §6.

2 Related Work

The model we present in the next section draws inspiration from prior work on end-to-end speech recognition and on techniques for training ARLS systems.
2.1 End-to-End Speech Recognition

Previous ARLS systems have been built using the same foundational concepts employed by ASR systems; for example, Mesaros and Virtanen (2010) build their best-performing ARLS system by coupling a phonetic hidden Markov model (HMM) recognizer and a phoneme bigram language model. As such, this sort of ARLS system design suffers the same drawbacks associated with the corresponding ASR system design. Namely, such systems are laborious to train, as the data - originally in the form of orthographic transcriptions - must be converted to and from the phonetic transcriptions used within the model. This conversion process requires a pronunciation dictionary, which is written by humans and serves as a source of reduced performance (Graves and Jaitly, 2014).

While speech is relatively easy to phonetically transcribe, the artistic nature of music means non-standard pronunciations occur exceptionally frequently; Table 1 shows an example of such a pronunciation. Such pronunciations are more difficult to transcribe and serve as a source of error. As such, eliminating the aforementioned conversion process is critical for ARLS systems.

<table>
<thead>
<tr>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lyrics with the birds I’ll share</td>
</tr>
<tr>
<td>Speech W IH DH DH AH B ER D Z AY L SH EH R</td>
</tr>
<tr>
<td>Song W IH DH AH B ER D Z AY SH EH</td>
</tr>
</tbody>
</table>

Table 1: Phonetic transcriptions of Scar Tissue, spoken and sung. (Red Hot Chili Peppers, 1999b)

The recurrent neural network design introduced by Graves and Jaitly facilitates end-to-end speech recognition, obviating the need for orthographic-phonetic conversion of the data in an ASR system. As an added benefit, this design requires no language model; the neural network learns the correct spelling of words directly from the labeled training dataset. These benefits require use of the Connectionist Temporal Classification (CTC) objective function, which maximizes the log probability of a completely correct transcription for a given sequence. As a result of the recurrent neural network (RNN) design and the choice of objective function, much more data is needed to adequately train the system than is used in traditional ASR implementations.

By employing a derivative system design, we also remove the need for the orthographic-phonetic conversion in ARLS. However, datasets of song utterances are not sufficiently large to properly train this type of ARLS system. The consequent discrepancy between our results and those of Graves and Jaitly are detailed more thoroughly in the following sections.

2.2 Methods for training ARLS Systems

ASR systems based on recurrent neural networks require especially large training datasets to properly tune model parameters for maximal performance. Graves and Jaitly (2014) note that training their RNN-CTC system on 14 hours of speech utterances results in a word accuracy of 25.8% in the absence of a language model. For comparison, their DNN-HMM baseline system had a word accuracy of 43.9% in the same scenario. However, training on 81 hours of speech utterances enabled the RNN-CTC system to achieve a word accuracy of 69.9%, compared with 48.9% for the baseline system.

As the ARLS task is more computationally difficult than the ASR task, our system requires more training data than Graves and Jaitly’s in order to achieve similar performance. Unfortunately, orthographically transcribed song datasets of that magnitude are not publicly available. While datasets of songs with lyrics do exist, systems for automatically aligning lyrics with vocals exhibit line-level errors on the order of seconds (Mesaros and Virtanen, 2010), which is too significant to ensure that a lyric-aligned song dataset would be sufficiently accurate for RNN training without manual inspection - an infeasible proposition on either a 14 or 81 hour dataset.

As such, we reproduced the procedure introduced by Mesaros and Virtanen and used large speech datasets to train our system, although smaller datasets of monophonic music were used for testing. (The music datasets were manually collected and orthographically transcribed.) In addition, we performed signal processing techniques to make these speech utterances more closely resemble song utterances. These techniques are detailed in §3.

3 Datasets

The datasets of speech utterances used for training our ARLS system were CMU Arctic (Kominek et al., 2003) and CSR-I (WSJ0) Sennheiser (Garofalo et al., 1993). CMU Arctic consists of 1,132 read sentences, with each sentence phonetically...
balanced and repeated 6 times by different speakers. (All sentences are in US English; the speakers are 2 American men, 2 American women, 1 Canadian man and 1 Scottish man.) In total, CMU Arctic contains about 4 hours of speech utterances. Meanwhile, WSJ0 consists of 35,487 read sentences mainly consisting of text isolated from the Wall Street Journal, performed by various speakers including journalists. These sum to 14 hours of speech utterances.

These datasets were used in their original forms for training, and were additionally perturbed to yield a greater variety of training samples that would more closely resemble the singing used for testing the model. One perturbation method, which we termed noising, involved adding random noise directly to the array of audio signals used to encode the speech utterances with the 16-bit LPCM WAV format. This was done by finding the mean and standard deviation of these audio signals, creating a Gaussian distribution from those parameters, and perturbing each value in the array by a sample taken from that distribution multiplied by a constant. The final effect was that pseudorandom noise was added to the audio file, and when the utterance was played white noise could be heard in the background. This process replicated the artifacts from the vocal isolation process that were present in the singing utterances, which also sounded like white noise in the background when we played them. The results of this process can be viewed objectively; note that Figure 3 appears to more closely resemble Figure 2 than Figure 1 does.

The second perturbation method, which we termed warping, made use of an intuition realized from observation of the frequency histograms for read speech (Figure 1) and isolated vocals.
(Figure 2). The frequency distribution for isolated vocals approximated a Gaussian distribution, while that of speech had a very significant mode and three less significant modes (not including silence), suggesting that read speech is limited to only a few frequencies while singing uses a considerably wider vocal range. Based on this intuition, warping involves splitting the speech samples into ten parts of equal amplitude, and multiplying the pitch of each sample by a random variable with $\mu = 1, \sigma = 0.2$. This process increases the frequency variance of the original read speech utterances, as can be viewed in Figure 4, and these utterances sounded more like song when they were played.

Regardless of whether the original sound data is speech or song: unaltered or perturbed; or intended for training, validation, or testing, we extract 13 mel-frequency cepstral coefficients (MFCC) features from the data and use them to represent the original sound. These 13 features, along with labels of orthographic transcriptions, are what our models accept as input.

To test our ARLS system, we use 6 monophonic songs (i.e., with isolated vocals) that we had orthographically transcribed by hand. The songs were located with the aid of an internet community dedicated to extracting isolated vocals from popular songs (r/IsolatedVocals, 2017). Our dataset was collected by downloading these songs, scraping their lyrics from an associated site (All The Lyrics, 2017), and then manually segmenting the songs and lyrics into slices with length 5-7 seconds. This process yielded 175 orthographically transcribed monophonic songs which totaled to 17.5 minutes of singing utterances.

4 Model Design and Implementation

Our model is derivative of that introduced by Graves and Jaitly (2014). Their model is a significant departure from past ASR systems, which were largely based on HMMs. Instead, Graves and Jaitly describe an end-to-end neural network model where much of the ASR pipeline is replaced by a recurrent neural network with a Connectionist Temporal Classification loss function. Additionally, beam search decoding is used to get a predicted output of text from the given inputs at each time step. As such, our model has these features as well, but extensions are made in an effort to decrease training time and optimize performance.

Our model is implemented using standard TensorFlow libraries.

4.1 Neural Network Architecture

More concretely, for the neural network part of our model we use multi-layered bidirectional GRU RNNs. There are several motivations behind this decision. First, RNNs have empirically been successfully used to predict on temporal data. Speech falls into this category, as speech is inherently dependent upon time. We used GRU cells instead of the vanilla RNN cell to help solve the vanishing gradient problem that occurs when backpropagating through each time-step of the RNN. (Graves and Jaitly use LSTM cells, but Andrew Maas recommended we use GRU cells instead.) Pertinent equations detailing the GRU cells are found in equations below:

Vanilla RNN Cell Equation:

$$h_t = f(W^h h_{t-1} + W^x x_t)$$

GRU Cell Equations:

$$\tilde{z}^{(t)} = \sigma(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)})$$
$$\tilde{r}^{(t)} = \sigma(W^{(r)}x^{(t)} + U^{(r)}h^{(t-1)})$$
$$\tilde{h}^{(t)} = \tanh(\tilde{r}^{(t)} \cdot Uh^{(t-1)} + Wx^{(t)})$$
$$h^{(t)} = (1 - z^{(t)}) \cdot h^{(t)} + z^{(t)} \tilde{h}^{(t-1)}$$

As can be seen from the equations above, GRU cells output the hidden state $h^{(t)}$ via a combination of the past hidden state $h^{(t-1)}$ and the “gates” described by $z^{(t)}$ and $\tilde{h}^{(t)}$. The gates are designed to have memory across time-steps and make it easier for the model to make predictions based on long-term dependencies.

The hidden state output is then passed into an affine layer that gives logits at each time-step. This is described by the equation below:

$$\hat{y} = W^{(S)} f(h_t) + b$$

Furthermore, since our model is able to ingest the input speech or song data in its entirety and then output the predicted text, we can model the data with both a forward context and backward context via a bidirectional RNN. In theory, this allows our model to understand the nuances of long-range contexts of the training data better, as well as to use all surrounding context of a segment of speech, in order to give better predictions. The
specific equations that govern this are similar to the above equations for any RNN cell, except that there are forward and backward outputs that are concatenated together along with a bias. This is detailed in Figure 5.

![Bidirectional RNN Architecture](image)

Figure 5: Bidirectional RNN Architecture

Finally, we stacked multiple RNNs on top of each other, creating a deep RNN structure. The exact number of stacked layers and the size of the hidden layers was a hyperparameter we tuned (see Experiments section). The output hidden sequence of the lower RNN layer provides the input for the next RNN layer and so forth, creating a deep RNN architecture. This culminates in the top layer outputting its hidden state into an affine transformation. The motivation for doing this is to allow the model to have higher levels of representation for the input data and hopefully better learning.

4.2 CTC Loss

Connectionist Temporal Classification is a loss function that allows RNNs to be trained in an end-to-end fashion without having a prior set alignment between the input data and output targets. Given an output transcription $\mathbf{y}^*$, the defining loss equation is below:

$$CTC(x) = -\log(Pr(\mathbf{y}^* | x))$$

However, the lack of alignment in speech means finding $Pr(\mathbf{y}^* | x)$ must be handled with a task specific approach.

As described in depth by Graves and Jaitly (2014), the output layer for the CTC function contains units for each transcription label; in our case this was the full character set, in addition to a ‘blank.’ The blank allows the model to emit null character predictions which give it flexibility in deciding when to emit an actual transcription label. If the input sequence $x$ is of length $T$, the output vectors $y_t$ (this is an output vector, or logits, from the affine transformation defined above at time-step $t$ of the RNN) are normalized with softmax and represent the probability of emitting the character or blank at time $t$. For the sake of implementation, we modeled characters as indexes $k$ corresponding to a unique character in the output vocabulary. Thus, the overall probability distribution at each time-step of the RNN is modeled by:

$$Pr(k, t | x) = \frac{\exp(y^k_t)}{\sum_{k'} \exp(y^{k'}_t)}$$

where $y^k_t$ is an element of the logits vector $y_t$. With this defined, we realize that we must now consider all possible alignments of the same sequence. More specifically, because of the introduction of the blank emission, for a length $T$ output sequence $\alpha$, we must remove all blanks (since they do not correspond to any real character) and collapse all repeated characters. This process, denoted by the operator $\beta$, yields our desired output transcription $\mathbf{y}^*$. However, this means that there are multiple sequences that can lead to the same $\mathbf{y}^*$, and thus to find the probability of an output transcription $\mathbf{y}^*$, we must sum across the probabilities of all possible sequences, or alignments, that make $\mathbf{y}^*$. This is given by the equation below.

$$Pr(\mathbf{y}^* | x) = \sum_{\alpha \in \beta^{-1}(\mathbf{y}^*)} Pr(\alpha | x)$$

The quantity above yields our desired probability for output transcription $\mathbf{y}^*$ for CTC loss.

4.3 Decoding

Decoding the network refers to the process by which the model finds the most probable output transcription $\mathbf{y}$ given an input $\mathbf{x}$. In mathematical terms, this means we are trying to find the value of $\arg\max_{\mathbf{y}} Pr(\mathbf{y} | \mathbf{x})$. We use beam search on our
model to find the best possible output \( y \), using a beam width of 100 and not allowing repeated characters to be merged (this was a flag we set on the TensorFlow module). Beam search allows us to gather output sequences from the inputted logits from the RNN.

5 Experiments

Initial experiments confirm the validity of our model. We fit the model to a set of five samples from the WSJ0 corpus and achieved training loss of 1.0 within 4 hours. Separately, we also overfit the model on distinct subsets of CMU Arctic consisting of both original and perturbed data; these experiments completed roughly an order of magnitude faster.

We ran further experiments to determine the best performing dataset, tune the model hyperparameters, and evaluate methods for preventing overfitting. Models were trained on four datasets: CMU Arctic (unaltered, noised, and warped) and WSJ0 (unaltered). To tune hyperparameters, models were trained while independently varying the L2 lambda from 1e-5 to 1e-2, the learning rate from 1e-5 to 1e-3, the hidden layer size from 100 to 500, and the number of hidden layers from 1 to 5. We additionally trained models varying the activation functions for the GRU cells between ReLU and tanh. To combat overfitting, we also trained models varying the aforementioned hyperparameters and with dropout applied between the timesteps and layers of the RNN.

In total, we trained 48 unique models. Models were trained for an average of 16 hours on 20 different Google Compute Virtual Machines running Ubuntu 16.04. In conjunction with the time needed to test a trained model, these experiments took roughly 100 hours of real-world time.

5.1 Evaluation

Models were evaluated during training by computing the loss and character error rate (CER) on a validation set distinct from and one-tenth the size of the training set. (CER is used as a metric because it correlates well with WER and is more precise on the small samples used for training.) We used these results to choose optimal hyperparameters and model architecture.

Final model performance was evaluated by computing the loss and CER achieved when running the trained model on the test dataset of labeled monophonic singing. For comparison, we also ran Google Cloud Speech on the same test dataset of labeled monophonic singing.

On the WSJ0 dataset, the runtime was too large to achieve meaningful performance. Therefore, we evaluated our model’s performance on the smaller CMU Arctic dataset, both unaltered as well as augmented with perturbed speech utterances. Our experiments using dropout were similarly too time-intensive to allow the model to train long enough for proper evaluation.

Our best performing models from our experiments on the CMU Arctic datasets achieved the following results:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Train CER</th>
<th>Val CER</th>
<th>Test CER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.4</td>
<td>0.7</td>
<td>0.801</td>
</tr>
<tr>
<td>Noised</td>
<td>0.44</td>
<td>0.7</td>
<td>0.819</td>
</tr>
<tr>
<td>Warped</td>
<td>0.49</td>
<td>0.7</td>
<td>0.814</td>
</tr>
</tbody>
</table>

![Figure 6: Graph of CER over time for model trained on the original CMU Arctic dataset. The blue line is training CER and the orange line is validation CER.](image)

![Figure 7: Graph of CER over time for model trained on the CMU Arctic dataset with noising. The blue line is training CER and the purple line is validation CER.](image)
Figure 8: Graph of CER over time for model trained on the CMU Arctic dataset with warping. The blue line is training CER and the orange line is validation CER.

As a benchmark, we found that the Google Cloud Speech recognition system achieved a WER of 0.61 and a CER of 0.48 on our test set of monophonic singing. Table 2 compares some transcriptions of the test set by our model and Google Cloud Speech:

| Correct lyrics: | day was gonna come when I was gonna mourn ya |
| Google Speech: | I was going to call him when I was on ammonia |
| Our model       | yo                                             |
| Correct lyrics: | I know we can make it if we take it slow       |
| Google Speech: | play we can make it if we take it slow         |
| Our model       | both r r t o o o t                              |

Table 2: Orthographic transcriptions of Dani California. (Red Hot Chili Peppers, 1999a)

5.2 Discussion

Our experiments show that this system would not produce sufficiently accurate orthographic transcriptions if applied in isolation to monophonic music. Additionally, they show that this inferior performance is likely due to a lack of training time and insufficient training dataset size.

More specifically, as mentioned in §2, Graves and Jaitly were able to achieve a word accuracy of 25.8% after training on WSJ0, the 14-hour speech dataset; however, our model requires 4 hours to train on a tiny fraction of that dataset. While we also began training on WSJ1, the same 81-hour speech dataset that allowed Graves and Jaitly to achieve a word accuracy of 69.9%, it quickly became obvious that our model would require an extreme amount of training time before producing sufficient performance.

As an alternative, we trained on CMU Arctic and its perturbed versions, but the widening gap between training and validation CER in Figures 6, 7, and 8 suggest that our model was overfitting these smaller datasets. To combat this, we tried adding dropout, but it multiplied the training time by an unreasonable factor given our time and resource limitations.

Nevertheless, there is a key takeaway from the model results displayed in Table 2. The prediction of English words such as “both” demonstrate that our model, which has no language model or dictionary incorporated, learns full words from the training set. We cannot determine whether our model would be capable of recognizing words outside of the training set (e.g. slang words) because no experiments reached the point of predicting full words on test data.

For comparison, the Google Cloud Speech recognizer predicted only valid English words. As a result, it is inaccurate when sung words are pronounced unconventionally, as in the line “day was gonna come when I was gonna mourn ya” from Dani California.

Concerning the augmented data, our preliminary results show that there is no improvement in the test CER between the unaltered speech data and either the noised or the warped data, meaning that our data augmentation did not increase performance. While these results suggest that our methods for data augmentation were ineffective, there may be more sophisticated signal processing methods for altering speech that would cause it to better resemble singing and which would improve the performance of our system.

Aside from augmenting spoken data, a large training set of isolated vocals (at least 4 hours, the size of CMU Arctic) could very well facilitate a large improvement in the accuracy of lyrical transcription. Results from training our model on a larger monophonic singing dataset would distinguish whether better data (i.e., a larger dataset of isolated vocals), or more advanced or complex models are needed to see improvements in test performance. If we were able to achieve better results by training with even a few hours of isolated vo-
6 Conclusion

This was a particularly exciting project because it encompassed a novel application of cutting-edge deep learning techniques for Automatic Recognition of Lyrics in Singing. We found the nuances of the deep RNN architecture interesting to implement and significantly challenging to train optimally. Even though our results were not com-
parable to the Google Cloud Speech benchmark, we still believe there are promising results to be had.

There are a variety of future steps that could improve upon this work. This includes experiments with longer runtimes, the addition of a language model, retraining the model with expected transcription loss minimization (a technique used by Graves and Jaitly), additional data augmentation techniques, and vocal isolation from polyphonic music.

Streaming services continue to revolutionize the music industry, and tools for understanding music automatically must be developed. We hope this work informs future research done in the subject area of Automatic Recognition of Lyrics in Singing.

Source Code

The code for our most successful model design can be found at GitHub at the following link: raw.githubusercontent.com/mapte05/2Pac2text/test-rnn/model.py

Acknowledgments

We would like to thank Lindsey Kostas and Amani Peddada, also in CS224S, for help in converting the WSJ dataset into a usable format.

We would also like to thank our professor Andrew Maas for his support and guidance throughout this project.

We are grateful for the credits that allowed us to access Google Compute Cloud with a limited budget.

References


Red Hot Chili Peppers. 1999b. Scar tissue.
