LAS for Dysarthric Speech Recognition

Jeremy Tate Campbell
Department of Computer Science
Stanford University
jcamp12@stanford.edu

Abstract

People with dysarthria have motor disorders that often prevent them from efficiently using commercial speech recognition systems. In this project, we train a speech recognition system on dysarthric speech using a Listen-Attend-Spell model, which uses a pyramidal bidirectional LSTM and a beam search decoder to predict phonetic transcriptions. The network produces phoneme error rates higher than state-of-the-art models on a test set of speakers with low speech intelligibility.

1 Introduction

People with speech disorders regularly struggle to interact with automatic speech recognition (ASR) systems. This is caused by a variety of reasons, the primary one being the scarcity of in-domain data due to the extensive ethical and medical permission requirements for data collection. Other reasons include abnormal tendencies in speech patterns and an inadequate breadth of research and advocacy on the subject. People with speech disorders also rely more heavily on these systems, as they often have motor disorders that hinder their ability to use any handheld device. Our project focuses on people with dysarthria, a general motor speech disorder due to weakness in the face, lips, tongue, or throat. Some signs of dysarthria include slurring, stuttering, mumbling, hoarseness, and irregular or inconsistent volume and pace of speech, many of which are troublesome for ASR systems.

It was therefore our goal in this project to build a speech recognition system tailored towards people with dysarthria by training on dysarthric speech. Because of the inconsistency of acoustic cues in their speech patterns, this project chooses to combine a bidirectional LSTM (BLSTM), in hopes that its memory cells could handle the inconsistent temporal behavior, with a beam search decoder, in hopes that learning phonetic relationships can improve accuracy. These components form a state-of-the-art Listen-Attend-Spell (LAS) model (1). We chose to break down words by their phonemes using the NLTK's CMU Pronouncing Dictionary (11); this helped to compensate for not having access to the profusion of data needed for end-to-end ASR, and it allowed us to assess performance with phoneme-level granularity.

In building our speech recognition system, our algorithm took in an audio (WAV) file of a single word spoken by a person with dysarthria. It was then sent to the LAS network to output the phonetic transcription of the word. We have also used the same dataset for a concurrent project in CS 230 to accomplish the same task. The CS 230 project uses a deeper bidirectional LSTM (DBLSTM) as an encoder with a CTC decoder. This puts much more emphasis on the encoding portion of the network and decodes the phonemes *independently* of each other, whereas the LAS decoder uses information from previous phonemes to make predictions for subsequent ones.

2 Related work

Most of the related work in dysarthric speech recognition involves finding ways to handle the irregularity of acoustic cues. Early attempts at modeling this involved using GMM, HMM, and

CS229: Machine Learning, Spring 2019, Stanford University, CA.

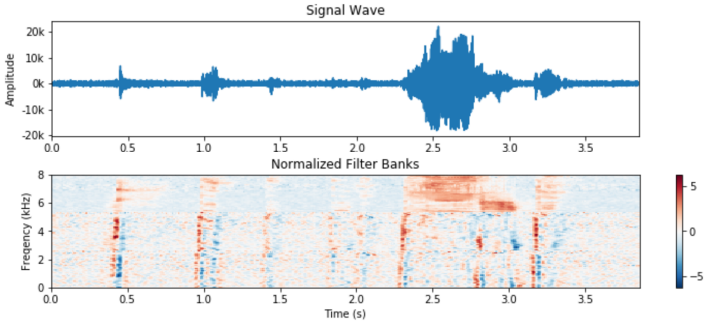
SVM-based recognition (2), where it was found that HMMs have robustness against large-scale word-length fluctuations, while SVMs have robustness against deletion of consonants. Attempts to exploit prior articulatory knowledge of dysarthric speech in the presence of limited data has been researched (3), and incorporating that information into a dynamic Bayes network showed mild improvements in phoneme recognition. Our project does not attempt to use prior speech knowledge. While these results are noteworthy, they don't achieve the accuracy deeper networks can.

Kim et al. used a Kullback-Leibler divergence-based HMM on phoneme posterior probabilities outputted from a deep neural network (4), which is similar in concept to what the LAS does. They found word error rate (WER) improvements over conventional models despite only a limited supply of data. Transfer learning has also been applied on an ASR system trained on non-dysarthric speech to include additional hidden layers trained on dysarthric speech (5), a shrewd idea given the large discrepancy of data between the two sources. This multi-staged deep neural network produced a WER improvement compared to both of its parts. While our LAS uses training data from dysarthric and non-dysarthric speakers, we group them together rather than staging the training process.

More state-of-the-art models have involved CNNs and RNNs. Convolutive bottleneck feature extraction (6) was able to mitigate unstable speaking styles in dysarthric speakers and decreased WER compared to conventional MFCC features. RNNs similar to ours (7) were employed with mild success in capturing temporal characteristics, achieving a WER of 13% on the well-known TORGO database for dysarthric speech. One particularly clever model used a convolutional LSTM, which benefits both from the local feature extraction of the CNN and the temporal modeling of the LSTM (8). This network achieved a phoneme error rate (PER) of 35%, lower than either their CNN or LSTM did alone. Finally, Joy et al. took a deep dive into hyperparameter tuning on the TORGO database, and they saw great improvements in WER on similar previous architectures (9).

3 Dataset and Features

The dataset we used was from the UA-Speech Database, a fundamental resource for ASR development for people with neuromotor disabilities (10). All audio was recorded using an eight-microphone array with preamplifiers attached. Video recordings were also available, though not used in this project. The full dataset consisted of single, isolated words spoken by 15 speakers with varying degrees of dysarthria and 13 speakers without dysarthria to use as a control. During recording sessions, subjects were seated comfortably near a computer screen and read the words off of a PowerPoint slide. Each subject was asked to recite a total of 765 words, split over the course of three recording sessions (about 255 words per session). Subjects were encouraged to take breaks whenever needed. The words in the dataset consist of digits ("zero," "one," etc.), the International Radio Alphabet ("alpha," "bravo," etc.), computer commands ("space," "enter," etc.), common words ("the," "of," etc.), and uncommon words ("naturalization," "frugality," etc.). The uncommon words were chosen using a greedy algorithm on digitized children's novels in order to maximize uncommon phonemes.



ALPHABET

a0 aa1 aa2 ae0 ae1 ae2 ah0 ah1 ao1 ao2 aw 1 aw2 ay0 ay1 ay2 b ch d dh eh0 eh1 eh2 er 0 er1 er2 ey0 ey1 ey2 f g hh ih0 ih1 ih2 iy0 i y1 iy2 jh k l m n ng ow0 ow1 ow2 oy1 oy2 p r s sh t th uh1 uh2 uw0 uw1 uw2 v w y z zh

command	k ah0 m ae1 n d
pajamas	p ah0 jh aa1 m ah0 z
observation	aa2 b z er0 v ey1 sh ah0 n

Figure 1: (Left) Raw audio signal for the word "paste" and its corresponding filter bank. (Right) Full alphabet of phonemes with word-to-phoneme example translations.

We split the data by speaker, with only two of the subjects with dysarthria and one control subject being left out of the training data. The remaining data from these three speakers were split in half randomly and distributed to the dev and test sets. This resulted in about an 84-8-8% train-dev-test split. Before being fed into the LAS model, we first pre-processed the audio data and the text transcriptions.

Normalized filter banks for the audio data were calculated by applying pre-emphasis filters, Fourier transforms, triangle filters, and normalization for time windows of 10ms. An example of an audio signal and its corresponding normalized filter bank for the word "paste" are shown in Figure 1. Notice the small, recurring amplitude jumps in the audio signal, representing a stutter on the first phoneme. Each column of the filter bank was a vector of 123 features and represents one input x_i into the LAS. Transcriptions were broken down into phonemes using the alphabet shown in Figure 1. Several examples of word-to-phoneme translations are also given.

4 Methods

The LAS model can be broken down into two major components, a "listener" and a "speller." The listener's job is to take in the pre-processed audio data and encode it into a hidden representation $\mathbf{a}$. The speller uses this representation to learn where in $\mathbf{a}$ we "attend" to for each phoneme step, and uses this to predict the phonemes. The full model architecture, equations, and an example LSTM cell are given in Figure 2.

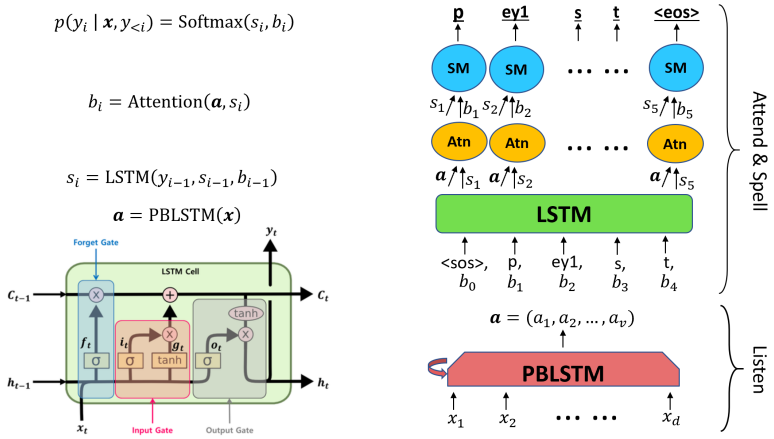


Figure 2: The full LAS architecture for the example word "paste" (right), along with an example LSTM cell and the model equations (left).

The listener uses a pyramidal bidirectional LSTM to convert the 10ms normalized filter bank vectors x_1, x_2, ... x_d into a shorter representation $\mathbf{a} = a_1, a_2, \ldots a_v$. "Bidirectional" indicates that for each LSTM cell, there is both a "forward cell" and a "backward cell." This allows the RNN at each time step to learn from the audio both before it and after it. The activations of each were concatenated and sent to the next layer of the network. "Pyramidal" specifies that in each successive layer of the LSTM, adjacent activations are concatenated with each other and used as input into the next layer, thereby reducing the number of time steps by a factor of two for each layer. This is tremendously useful in reducing computational complexity.

Figure 2 also shows the computational graph of a single, generic LSTM forward cell. The previous activation h_{t-1} and the current input x_t are combined and sent through sigmoid activations to obtain weights of a forget gate, input gate, and output gate. The forget gate and input gate are used to weight how much the new memory value c_t should rely on the old memory value c_{t-1} or the proposed new memory value g_t. The output gate determines how much weight we put into the $\tanh$-activated new memory value c_t to obtain the output activation h_t.

The speller portion of the model has three parts: an attender, an LSTM, and a softmax layer. The attender's job is to figure out which parts of $\mathbf{a}$ should be focused on at each phoneme step. More specifically, it finds b_i by taking a weighted sum of each component of $\mathbf{a}$. These weights should be higher at the time steps corresponding to when the phoneme is being pronounced. The weights are found by inputting $\mathbf{a}$ and s_i into a simple 1-layer neural network with a $\tanh$ activation and a softmax output. The LSTM part of the speller takes as input the attention b from the previous time step, as well as the *ground truth* phoneme label from the previous time step. Taking in the ground truth label allows the speller to predict phonemes based on previous ones. Finally, we use the b_i

from the attender and the s_i from the LSTM as inputs into the last softmax layer, which will give us probabilities for each possible phoneme. The sequence starts by sending a <sos> token into the LSTM, and terminates when the <eos> token is encountered in the output (the <eos> token is added to the end of each phoneme transcription). Note that the speller will have (N + 1) steps, where N is the number of phonemes (not counting <eos>) in the word.

We train the model using the average cross-entropy over each phoneme. If the ground truth word has N phonemes and we have a total of K phonemes in the alphabet, this is given by

$$\text{loss } = -\frac{1}{N+1} \sum_{i=1}^{N+1} \sum_{k=1}^{K+1} y_k^{(i)} \log \hat{y}_k^{(i)} \tag{1}$$

where $\hat{y}_k^{(i)}$ and $y_k^{(i)}$ represent the predicted probability and the ground truth label (1 or 0), respectively, of the i-th phoneme in the word being the k-th phoneme in the alphabet. We add one to each N and K for the possibility of the <eos> token. This equation is then averaged over all words in the batch.

During prediction time, we use a beam search decoder, which works by specifying a beam size B and predicting each phoneme one-by-one, while only keeping track of the B most likely sequences. For instance, if we set B to 16, at each step we will have the 16 most likely sequences up to that point. We then try to add every phoneme in the alphabet to each of the 16 sequences, and again, only keep the 16 most likely sequences. The process terminates when all 16 sequences have predicted an <eos> token. More specifically, at the k-th step, we try to maximize the conditional probability given in equation 2. The outputs from the softmax layer represent the RHS of equation 2.

$$p(y^{(1)},\ y^{(2)},\ \ldots,\ y^{(k)}|\mathbf{x}) = p(y^{(1)}|\mathbf{x})\ p(y^{(2)}|\mathbf{x},\ y^{(1)})\ \ldots\ p(y^{(k)}|\mathbf{x},\ y^{(1)},\ y^{(2)},\ \ldots,\ y^{(k-1)}) \tag{2}$$

5 Results

We settled on a mini-batch size of 64 even though this batch size produced erratic training loss jumps. We'd have liked to increase it, but higher batch sizes were too memory expensive for our machine to handle. A learning rate of 0.001 was initially chosen based on other successful models in the literature, but an exponential decay of 0.1 was added after seeing the training loss continue to have large oscillations later into training. The full model did 30 epochs through the training data and we used a beam size of 16 in an attempt to balance accuracy with computational expenses. We trained three models and their results are given in Figure 3. The metric we used for our results was the phoneme error rate (PER), which gives the rate at which the true phonemes in the target word did not occur in the predicted phonemes.

Model	Train Set PER (%)	Dev Set PER (%)
Listener: 128 HU, 2 layers, keep_prob = 0.5 Speller: 128 HU, 2 layers, keep_prob = 0.5	48.66	62.56
Listener: 256 HU, 3 layers, keep_prob = 0.7 Speller: 256 HU, 3 layers, keep_prob = 0.7	41.19	64.13
Listener: 256 HU, 3 layers, keep_prob = 0.5 Speller: 128 HU, 2 layers, keep_prob = 0.5	44.20	60.02

Figure 3: PER on the train and dev sets.

Our parameter search attempted to optimize the number of layers and hidden units in the listener and speller's LSTMs, though this proved to be quite difficult given how expensive it is to train an ASR system. We started with 128 hidden units and 2 layers for each with a dropout rate of 0.5, but felt that we weren't adequately fitting to the training set. We then increased the complexity of each and decreased regularization, but we saw the dev set error actually increased. Our error analysis on this showed that many of the incorrect phonemes that were predicted were because the phonemes before it were also incorrect, indicating the speller was overfitting to previous phonemes. Our final model decreased the complexity of the speller and increased the regularization, as shown in the last row of Figure 3. This resulted in a *final test set PER of 60.15%*. This PER is slightly higher than expected, but it is worth noting that the three speakers in our test set had low speech intelligibility scores for dysarthric speakers, so the task is quite challenging. For reference, the Google Cloud speech recognition toolkit had a word error rate of 51% on the dysarthric speech and 14% on the control speech.

Figure 4 gives a heatmap of the test set PERs for the 30 most common phonemes spoken. The subjects are labeled on the y-axis by their gender and an identifying number, with the control speaker on the

bottom labeled with a prefix "C." Their aggregate totals are summarized in the phoneme labeled "all." Despite being trained largely on dysarthric speech, the control results still were substantially better. We also can notice how successful each phoneme was; for instance, "v" did very poorly on all subjects while "w" did very well.

Figure 4: PERs for the 30 most common phonemes broken down by speaker.

Figure 5 analyzes phoneme performance further by listing the best and worst three phonemes by PER for the control speaker, the dysarthric speakers, and the difference between the two. In general, common phonemes tended to be recognized at higher rates. In the CS 230 project, we found that phonemes that were strenuous on speech muscles like the "m" or "er0" (as in her) sounds performed worse on the dysarthric speakers relative to the control. However, this wasn't the case with the LAS model because phoneme predictions are heavily dependent on the previous predictions, whereas the DBLSTM model assumed conditional independence between predictions. This dependence led to phoneme performance in the LAS model having much more randomness.

	Control			Dysarthric			Difference		
	Phoneme	Example	PER (%)	Phoneme	Example	PER (%)	Phoneme	Example	PER (%)
Best	t	paste – p ey1 s *t*	31.25	w	with – *w* ih1 dh	44.58	l	l – *l* ay1 n	1.71
	s	so – *s* ow1	33.17	ah0	the – dh *ah0*	44.99	p	paste – *p* ey1 s t	4.28
	w	with – *w* ih1 dh	33.50	k	can – *k* ae1 n	50.35	ah0	the – dh *ah0*	6.82
	Phoneme	Example	PER (%)	Phoneme	Example	PER (%)	Phoneme	Example	PER (%)
Worst	l	l – *l* ay1 n	62.69	ey1	way – w *ey1*	77.51	eh1	then – dh *eh1* n	28.15
	p	paste – *p* ey1 s t	63.24	Ih0	in – *ih0* n	77.96	ey1	way – w *ey1*	28.94
	v	of – ah1 *v*	68.69	V	of – ah1 *v*	85.89	ih0	in – *ih0* n	31.96

Figure 5: The top three best and worst phonemes by PER from the control subject, the dysarthric subjects, and the difference between those two PERs.

The DBLSTM model for the CS 230 project actually managed a test set PER of 44.68%, about 16% better than the LAS model. Strangely, though, the word error rates between the two only differed by 5%. The reasoning behind this is that when the LAS model correctly predicted the first few phonemes, the accuracy of the remaining phonemes in the word greatly improved; if the first few were incorrect, this drastically hurt future predictions. We conclude that the dataset used does not have the requisite number of different phoneme combinations to prevent from overfitting to the limited number of phoneme combinations. Perhaps a speller with even less complexity would have also helped.

6 Conclusion

This project trained a speech recognition model on dysarthric speech using the listen-attend-spell approach. The PER of the final model on the test set was 60.15%, which is inferior to other state-of-the-art models (usually about 35%), but our test speakers had comparatively lower speech intelligibility. We found that a deeply-layered speller overfit the limited number of unique words in the dataset, and we likely require more words to properly train the LAS model since it relies so heavily on learning the conditional dependence between phonemes. Future work can include using prior knowledge of phoneme relationships to make predictions. Transfer learning using non-dysarthric speech also seems like a promising idea.

Acknowledgments

We'd like to thank Professor Mark Hasegawa-Johnson of the University of Illinois for kindly allowing us access to the UA-Speech database he helped to create.

Code

Code used for the project can be on Github with the link: https://github.com/jtcampbellsoup/SR-dysarthria. Be sure to look over the README first.

References

[1] Chan, William, et al. "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition." 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2016.

[2] Hasegawa-Johnson, Mark, et al. "HMM-based and SVM-based recognition of the speech of talkers with spastic dysarthria." 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings. Vol. 3. IEEE, 2006.

[3] Rudzicz, Frank. "Articulatory knowledge in the recognition of dysarthric speech." IEEE Transactions on Audio, Speech, and Language Processing 19.4 (2010): 947-960.

[4] Kim, Myung Jong, Jun Wang, and Hoirin Kim. "Dysarthric Speech Recognition Using Kullback-Leibler Divergence-Based Hidden Markov Model." INTERSPEECH. 2016.

[5] Yilmaz, Emre, et al. "Multi-stage DNN training for automatic recognition of dysarthric speech." (2017).

[6] Nakashika, Toru, et al. "Dysarthric speech recognition using a convolutive bottleneck network." 2014 12th International Conference on Signal Processing (ICSP). IEEE, 2014.

[7] Espana-Bonet, Cristina, and José AR Fonollosa. "Automatic speech recognition with deep neural networks for impaired speech." International Conference on Advances in Speech and Language Technologies for Iberian Languages. Springer, Cham, 2016.

[8] Kim, Myungjong, et al. "Dysarthric Speech Recognition Using Convolutional LSTM Neural Network." Proc. Interspeech 2018 (2018): 2948-2952.

[9] Joy, Neethu Mariam, and S. Umesh. "Improving acoustic models in torgo dysarthric speech database." IEEE Transactions on Neural Systems and Rehabilitation Engineering 26.3 (2018): 637-645.

[10] Kim, Heejin, et al. "Dysarthric speech database for universal access research." Ninth Annual Conference of the International Speech Communication Association. 2008.

[11] Natural Language Toolkit.
`https://www.nltk.org/`

[12] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] Nabu End-to-End ASR.
`https://github.com/vrenkens/nabu`

[14] 07-3 LSTM, GRU.
`https://excelsior-cjh.tistory.com/185`