CS230

# Dysarthric Speech Recognition Using a Deep Bidirectional LSTM

**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 deep bidirectional LSTM encoder with a CTC decoder. The network produces phoneme error rates slightly higher than other state-of-the-art models on a test set of speakers with low speech intelligibility.

## 1 Introduction

While speech recognition is a well-developed field, people with speech disorders still regularly struggle to interact with everyday speech recognition 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, difficulties adjusting algorithms for specific disabilities, and an inadequate breadth of research and advocacy on the subject. Furthermore, people with speech disorders tend to rely more heavily on these systems, as they often have motor disorders that hinder their ability to use a mouse, keyboard, or handheld device. Our project focuses on people with dysarthria, a general motor speech disorder due to weakness in the face, lips, tongue, or throat. Dysarthria can be caused by neurological injury, and some signs of dysarthric speech include slurring, stuttering, mumbling, hoarseness, breathiness, and irregular or inconsistent volume and pace of speech. These irregularities are naturally troublesome for automatic speech recognition (ASR) systems.

It was therefore our goal in this project to build a speech recognition system tailored towards people with dysarthria by training the model on dysarthric speech. Because of the inconsistency of acoustic cues in their speech patterns (it was quite common in our dataset to hear long pauses in the middle of words), we chose to use a deep bidirectional LSTM (DBLSTM) in hopes that its memory cells could handle the inconsistent temporal behavior. While major breakthroughs have been made in end-to-end deep learning in speech recognition, we instead broke down words by their phonemes using the NLTK's CMU Pronouncing Dictionary (10); 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 file of one word spoken by a person with dysarthria. It was then sent through the DBLSTM, which outputted the phonetic transcription of the word. The audio data we used was in the form of WAV files, each of which contained a single, one-word utterance of various words. We have also used the same dataset for a concurrent project in CS 229 to accomplish the same task. This paper will focus on solving the problem by applying a DBLSTM. The CS 229 project attempts to solve the problem using a listen-attend-spell (LAS) approach. This approach uses an attention model with a beam search decoder to

spell out the phonemes and it therefore relies more on phoneme relationships and their conditional probabilities to make predictions.

## 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 SVM-based recognition (1), 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 (2), 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. Naturally, the exploration of deeper models has been growing recently, with our DBLSTM being one of them.

Kim et al. (2016) used a Kullback-Leibler divergence-based HMM on phoneme posterior probabilities outputted from a deep neural network (3), which is similar in organization to what our encoder-decoder DBLSTM 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 (4), 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 DBLSTM uses training data from both dysarthric and non-dysarthric speakers, we group the training data together rather than staging the training process.

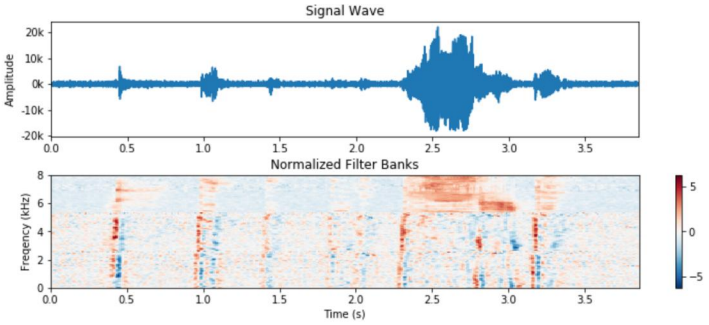
More state-of-the art models have involved CNNs and RNNs, with convolutional networks being proven to successfully extract local acoustic features using convolution and pooling layers. Convolutive bottleneck feature extraction (5) was able to mitigate unstable speaking styles in dysarthric speakers and decreased WER compared to conventional MFCC features. RNNs similar to ours (6) were employed with mild success in capturing temporal characteristics, achieving a WER of 13% on the well-known TORGO database for dysarthric speech, though this network did not include GRU or LSTM cells. 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 (7). This network achieved a phoneme error rate (PER) of 35%, lower than either their CNN or LSTM did alone. Finally, Joy et al. (2018) took a deep dive into hyperparameter tuning on the TORGO database, and they saw great improvements in WER on similar previous architectures (8).

## 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 (9). 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 while a helper advanced the slides for them. 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.

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 DBLSTM 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 into the LSTM. Transcriptions were broken down into phonemes using the alphabet shown in Figure 1. Several examples of word-to-phoneme translations are also given.




**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.

## 4 Methods

The DBLSTM we used functions as an encoder-decoder system, where the pre-processed audio data first runs through an encoder neural network to give an encoding of some hidden representation. The decoder then uses this representation as an input into another neural network that outputs the class labels for our phonemes. Figure 2 provides the high-level structure of the full network, as well as the logic within a single LSTM unit.

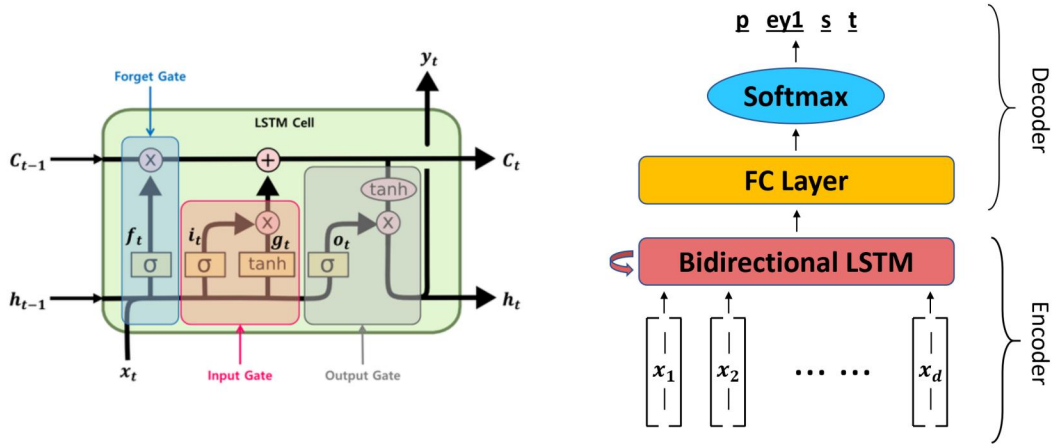



Figure 2: (Left) A single LSTM cell in the forward pass of the network. (Right) The full encoder-decoder architecture.

This project focused on optimizing the DBLSTM, which constitutes the encoder portion of the network. The 10ms normalized filter bank vectors $x_1$, $x_2$, ... $x_d$ were fed into the LSTM, as shown in Figure 2. Because the network was bidirectional, each LSTM unit had both a "forward cell" and a "backward cell." The forward cells propagated the activations $h_t$ forward; the backward cells propagated them back to earlier time steps. The activations of each were then concatenated and used as inputs into the next layer of the network. Both the number of hidden layers and number of hidden units in each activation were hyperparameters we attempted to optimize. Batch normalization and dropout were applied to each activation.

Figure 2 also shows the computational graph of a single 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 decoder takes the output of the DBLSTM and uses one fully-connected layer to one softmax layer for each time step. The output of the softmax layer gives probabilities for each possible phoneme. We then use Connectionist Temporal Classification (CTC) to give us a sequence of phonemes. CTC is a network output and loss function that is commonly used in sequence models because it is robust to varying sequence lengths (in our case, number of phonemes in a word and number of time steps). The CTC output finds the most likely character at every time step, condenses all characters that aren't separated by a "blank" label, then removes the "blank" labels. Figure 3 gives an example of a CTC output.

**p p p _ ey1 ey1 _ _ _ s t → p ey1 s t**

Figure 3: A CTC output of a valid representation of the word "paste."

The CTC loss function operates with similar logic, but instead tries to minimize the negative log-likelihood of the true label conditioned on the inputs. So for the true word "paste," we try to maximize the likelihood of its correct phonemes (given in Figure 3) being outputted based on the probabilities of the softmax outputs. For one mini-batch B, this is formalized in equation 1.

$$\text{loss} = \sum_{(x,y)\in B} -\log\ p(y|x) \tag{1}$$

## 5 Results

We tried a few different batch sizes on the first two epochs of the training data and settled on a batch size of 256 after seeing that it brought down the training loss the quickest. We used a learning rate of 0.001 with no decay based on a few other models in the literature and after seeing a sufficient learning curve during our training. The full model did 50 epochs through the training data. Using these hyperparameters, we trained three models and their results are given in Figure 4. 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 | Dev Set PER (%) |
|---|---|
| 128 hidden units, 3 layers, keep_prob = 0.5 | 48.12 |
| 128 hidden units, 4 layers, keep_prob = 0.5 | 47.03 |
| 256 hidden units, 5 layers, keep_prob = 0.7 | 44.50 |

Figure 4: The phoneme error rates on the dev set for varying hyperparameters.

Our first model had 128 hidden units and 3 layers, but because the training and dev sets had similar PERs, we tried to increase the complexity of the model. Despite this, the second model faced the same issue of not fitting the training set enough. Our last model further increased complexity by adding an extra hidden layer and extra hidden units, while also decreasing regularization by decreasing the dropout rate. Our final model did significantly better, lowering the PER down to 44.50% for speakers with dysarthria. While that may seem high for a speech recognition task, it is worthwhile to note that the three speakers in our test set had low intelligibility speech 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.

We ran this final model on the test set and found a *final PER of 44.68%*. This was very similar to the dev set score since each set was drawn from the same distribution. Figure 5 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." We can see that the model did very well on the control speaker despite half of the training data consisting of dysarthric speech. Speaker M14 had the lowest PER of the dysarthric speakers. We also can notice how successful each phoneme was; for instance, "v" did very poorly on all subjects while "ah0" did very well.

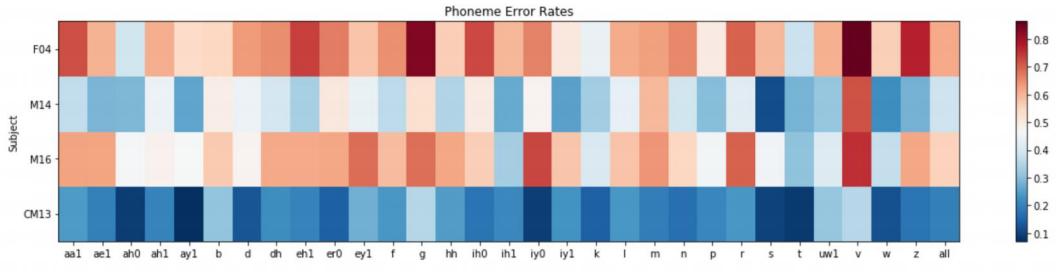



Figure 5: PERs for the 30 most common phonemes broken down by speaker.

Figure 6 analyzes phoneme performance further by listing the best and worst five phonemes by PER for the control speaker and the dysarthric speakers. Relative phoneme performance was fairly similar for the dysarthric versus the control speech, with common phonemes generally having lower PERs. Phonemes that have another quite similar to it were harder to label. For example, the phonemes "r" (as in their) and "er0" (as in her) are very difficult to distinguish. The last column in Figure 6 takes the difference of the PERs by phoneme for the two sets of speakers. Some of the phonemes that did much worse on the dysarthric speakers compared to the control were ones that can be challenging on speech muscles, such as the "m" and hard "er0" sounds.

| | Control | | | Dysarthric | | | Difference | | |
|---|---|---|---|---|---|---|---|---|---|
| | Phoneme | Example | PER (%) | Phoneme | Example | PER (%) | Phoneme | Example | PER (%) |
| Best | ay1 | line – l *ay1* n | 7.03 | t | paste – p ey1 s *t* | 32.17 | uw1 | to – t *uw1* | 13.34 |
| | t | paste – p ey1 s *t* | 8.63 | ah0 | the – dh *ah0* | 37.75 | ih1 | it – *ih1* t | 19.23 |
| | ah0 | the – dh *ah0* | 9.08 | s | so – *s* ow1 | 38.50 | p | paste – *p* ey1 s t | 21.01 |
| | iy0 | many – m eh1 n *iy0* | 9.25 | w | with – *w* ih1 dh | 38.65 | iy1 | he – hh *iy1* | 21.24 |
| | s | so – *s* ow1 | 10.08 | k | can – *k* ae1 n | 38.94 | b | tab – t ae1 *b* | 22.69 |
| Worst | ey1 | paste – p *ey1* s t | 27.75 | r | their – dh eh1 *r* | 60.30 | v | of – ah1 *v* | 42.35 |
| | b | tab – t ae1 *b* | 30.92 | m | some – s ah1 *m* | 62.35 | ih0 | in – *ih0* n | 42.40 |
| | uw1 | to – t *uw1* | 31.37 | iy0 | many – m eh1 n *iy0* | 62.72 | m | many – *m* eh1 n iy0 | 42.96 |
| | g | go – *g* ow1 | 35.55 | g | go – *g* ow1 | 68.52 | er0 | her – hh *er0* | 45.77 |
| | v | of – ah1 *v* | 35.65 | v | of – ah1 *v* | 78.00 | iy0 | many – m eh1 n *iy0* | 53.46 |

Figure 6: The top five best and worst phonemes by PER from the control subject, the dysarthric subjects, and the difference between those two PERs.

## 6 Conclusion

This project trained a speech recognition model on dysarthric speech using a deep bidirectional LSTM encoder and a CTC decoder. The PER of the final model on the test set was 44.68%, which is inferior to other state-of-the-art models (usually about 35%), but our test speakers had lower speech intelligibility than in other common dysarthric speech databases. We found that a deeper network of five layers fit the training data sufficiently when models of less complexity couldn't do so. We would like to have done a more thorough search of the hyperparameter space. Training a speech recognition model is computationally expensive, and with more time or computational resources, more combinations of hyperparameters could have been tested. Future work can include using prior knowledge of phoneme relationships to make predictions, or perhaps using a beam search decoder, which would use conditional probabilities between phonemes rather than assuming independence at each time step, like the CTC decoder did. 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.

## References

[1] 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.

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

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

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

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

[6] 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.

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

[8] 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.

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

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

[11] 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.

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

[13] 07-3 LSTM, GRU.
`https://excelsior-cjh.tistory.com/185`