# Deep Learning based Speech-to-Text Framework

Sidharth Vishnu Bhakth
MS in Data Science
Indiana University Bloomington
svbhakth@iu.edu

Vijayalaxmi B Maigur
MS in Data Science
Indiana University Bloomington
vbmaigur@iu.edu

## 1. INTRODUCTION

Deep Neural Networks and end-to-end models have significantly changed the landscape of speech recognition and language modeling. They have eliminated the need for hand-engineering features and allow the model to learn feature representations from clean/noisy speech signals. With large enough datasets, these networks can achieve state-of-the-art performance in speech recognition tasks and have offered significant improvement over the traditional approach of hand-engineering of features which are commonly used in automatic speech recognition (ASR) [1] [4] systems. They are commonly used in hybrid DNN-HMM speech recognition systems for acoustic modeling, pronunciation models that map words to phonemes, and language models that map speech to text. One of the biggest challenges in speech recognition is that it contains a wide range of variability in speech and acoustics. Thus, a good speech recognition model should be able to generalize and perform robustly even with varied acoustics. In traditional automatic speech recognition (ASR) systems, numerous complex components like feature extraction, pronunciation models, speaker adaption, acoustic models, etc. are modeled separately. Building and tuning these individual models are very hard, especially for a new language.

Recent work in this domain attempts to unify the components into a single end-to-end pipeline capable of taking in audio input and directly output text transcriptions. Two of the most popular end-to-end models in use today are Deep Speech by Baidu [2] and Listen Attend and Spell (LAS) by Google [13]. Both Deep Speech and LAS, are recurrent neural network (RNN) based architectures with different approaches to modeling speech recognition. Deep Speech uses the Connectionist Temporal Classification (CTC) loss function to predict the speech transcript. LAS uses a sequence-to-sequence network architecture for predictions.

Our approach is to use a hybrid deep neural network model with two components: A Convolutional Neural Network (CNN) architecture to learn the relevant features from the audio signal data and a Recurrent Neural Network (RNN) architecture to take the features as the input and output characters. We aim to create an iteration of the state-of-the-art Deep Speech 2 model [14] and modify certain architectural choices in the model to create an improved version. This approach also includes writing optimized code to parallelize model training on a GPU cluster. This allows us to train our model on larger audio datasets and thus offering generalizability.

## 2. RELATED WORK

This work is inspired by the recent advances made in end-to-end models. Advancements in deep learning have played a significant role in dramatically improving the performance of automatic speech recognition systems. Two of the most efficient end-to-end architecture for speech recognition are Deep Speech 2 by Baidu [2] and Listen Attend and Spell by Google. [13]

Convolution Neural Networks (CNN) have proven to be very effective at learning latent feature representations from images. Recent research shows that CNNs can work well on audio signals as well by feeding in a spectrogram of an audio signal as input. [3] A spectrogram can be thought of as an image-like representation of an audio signal. Even though there are variations in audio signals due to changes in acoustics and noise, the spectrograms will have similar features corresponding to a certain syllable/word. CNN's can effectively extract these latent features from the spectrogram. Recurrent Neural Networks [10] are highly effective for modeling sequential data with a temporal structure. After extracting the latent feature representations from CNNs, RNNs can learn the temporal pattern in the audio sequence to generate the text transcriptions.

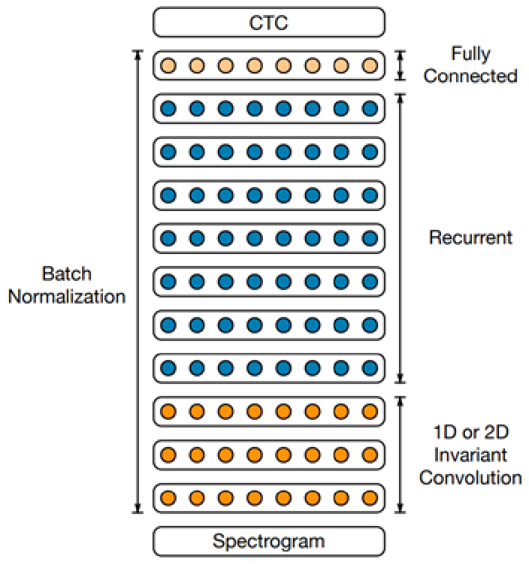


**Figure 1:** Architecture of the Deep Speech 2 model

Deep Speech 2 is an end-to-end deep learning speech recognition model, trained on very large amounts of data, and has achieved near human performance in speech transcription. Deep Speech 2 has a hybrid neural network architecture – a 1D convolutional layer that takes a speech spectrogram as input and extracts features from the data and a recurrent neural network-based architecture to model the sequence. Convolutional Neural Networks (CNN) are great at extracting latent feature representations from unstructured data and recurrent neural networks are very efficient at learning patterns in sequential data. A fully connected network with Connectionist Temporal Classification (CTC) loss function is used as the decoder for the speech to text transcription.

## 3. Architecture

Through this project, we aim to improve upon the DeepSpeech2 architecture by introducing certain key architectural changes. We add skip connections to the CNN layers and instead of using bidirectional RNNs, we modify them to bidirectional gated recurrent unit layers.

- # of residual CNN layers: 3
- # of bidirectional GRU layers: 5
- # of fully connected layers: 2

### 3.1 Residual CNNs

The CNNs used in the original Deep Speech 2 model speech convert audio signals to feature maps. CNN's have the advantage of being able to exploit local correlations of speech signals in time and frequency dimensions. Instead of the vanilla CNNs used in the Deep Speech 2 model, we plan to use Residual CNN layers. [4] Residual or Skip connections help achieve a 'flatter' loss surface as shown in Figure 2 and are hence easier to optimize.

Residual networks are composed of several such stacked residual connection blocks that contain direct links between the lower layer outputs and higher layer inputs. There is considerable empirical evidence suggesting that with residual networks we can use deeper layers with good accuracy gains. A deeper network would be better able to capture the nuances in human speech signals.

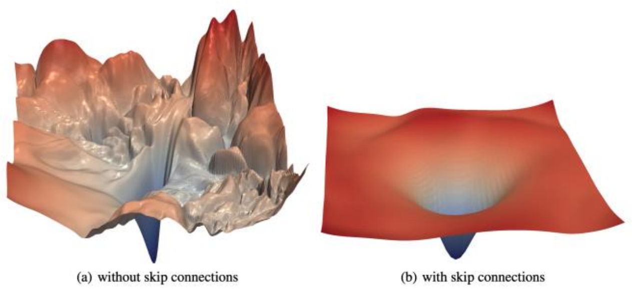


**Figure 2:** The loss surfaces of a DNN with/without skip connections

### 3.2 Gated Recurrent Unit (GRU)

The Deep Speech 2 model uses bidirectional recurrent layers which model both the forward in time and back in time directions to make a prediction for each frame using context from the previous and future frames. We plan to improve upon this architecture by using use a gated recurrent architecture like Gated Recurrent Unit (GRU) networks since they can learn the state over more time steps and hence learn long term dependencies in a sequence. Recent research in the areas of speech recognition and natural language processing shows that gated recurrent networks outperform vanilla RNNs. However, they are more computationally expensive to train.

### 3.3 Connectionist Temporal Classification (CTC) Loss

To explain this, we will have to first analyze the input and output formats. We have a dataset of audio clips and corresponding transcripts/sentence. The most important thing is we do not know how the characters in the sentence align to the audio data. This makes it hard for us to calculate the loss. Also, due to the varying rate of speech loss calculation becomes harder. CTC [5] is a way to get around this problem of not know the mapping between input and output. For a given input, we would train our model to maximize the probability it assigns to the right character. Conditional probability is used to achieve this.

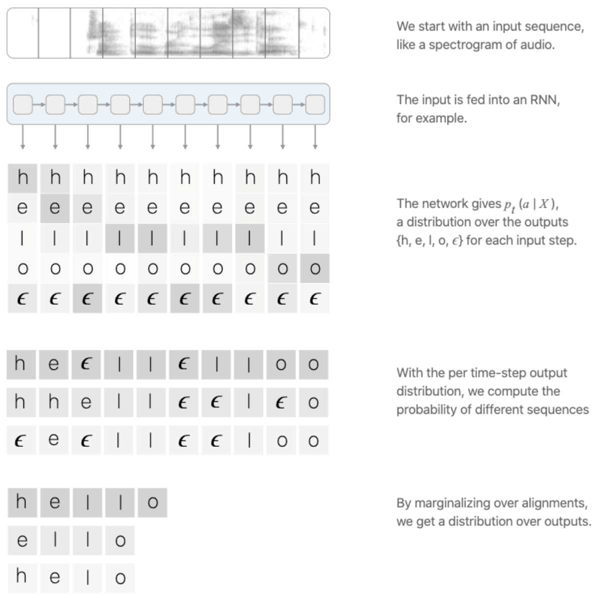

We start with an input sequence, like a spectrogram of audio.
The input is fed into an RNN, for example.
The network gives $p_t(a \mid X)$, a distribution over the outputs {h, e, l, o, $\epsilon$} for each input step.
With the per time-step output distribution, we compute the probability of different sequences
By marginalizing over alignments, we get a distribution over outputs.

**Figure 3**: Connectionist Temporal Classification (CTC) Loss

## 4. IMPLEMENTATION

The algorithm is implemented in pytorch [9] framework and torchaudio and torch distributed libraries are used as a part of implementation. AdamW is used as the optimizer for this architecture with one cycle learning rate scheduler. Connectionist Temporal Classification loss is the loss function.

### 4.1 SpecAugment

The Deep Speech 2 model was trained on ~12,000 hours of audio data most of which belongs to the internal Baidu corpora. Since we have access only to open-source corpora, we must resort to techniques to artificially increase the diversity of our dataset and thus increase the dataset size. For speech recognition, the standard data augmentation techniques include changing the pitch, adding noise and reverb, etc. Instead, we plan to use the Spectrogram Augmentation [11] (SpecAugment) approach which works by masking out contiguous blocks of time steps and randomly chosen frequency channels as shown in Figure 4. This significantly improves the ability of the model to generalize and is especially helpful in our case where the data is scarce and there is a risk of overfitting.

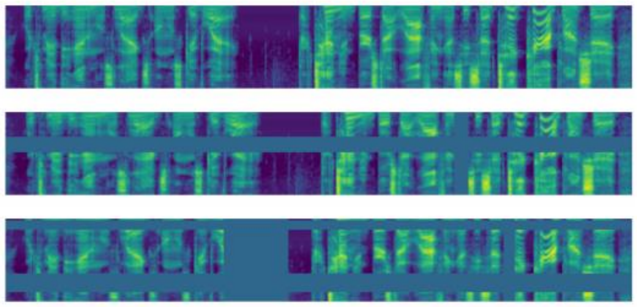


**Figure 4:** SpecAugment applied to the base input. From top to bottom, the spectrogram of the original input signal, with frequency masking applied, with frequency and time masking applied.

### 4.2 AdamW and One Cycle Learning Rate Scheduler

Adam is a widely used optimizer that helps the model converge more quickly to a local minimum. Although it performs great, it is not known for generalizing as good as Stochastic Gradient Descent. The implementation of Adam implicitly binds weight decay and learning rate together. This means that when optimizing the learning rate, weight decay also must be optimized. The AdamW [6] optimizer decouples the weight decay from learning rate optimization. This substantially improves model generalization. In one cycle learning rate scheduler [7], we can train the model faster, while achieving generalizability by starting with a low learning rate and gradually increase to a maximum and then decrease to the same point where you started. Neural Networks with AdamW optimizer using one cycle learning rate scheduler are trained quickly and achieving good generalization

### 4.4 PyTorch distributed

In our use case, because the dataset is too big to be fit into the memory, we could only do stochastic gradient descent for batches. We have 110K data points in the training dataset, and every time we could only use 10 data points to calculate the estimate of the gradients, otherwise, our GPU may stop working due to insufficient memory.

The shortcoming of stochastic gradient descent is that the estimate of the gradients might not accurately represent the true gradients of using the full dataset. Therefore, it may take much longer to converge. A natural way to have a more accurate estimate of the gradients is to use larger batch sizes or even use the full dataset. To allow this, the gradients of small batches were calculated on each GPU, the final estimate of the gradients is the weighted average of the gradients calculated from all the small batches.

## 5. EXPERIMENTS

### 5.1 Hardware

For model development, we initially started using Google Colaboratory which we found insufficient for our use. Later, we migrated development to the Juliet servers on IU Futuresystems and made use of GPU nodes available on the Romeo nodes. We had access to six NVIDIA Tesla K80 GPUs on the Romeo node. The configuration of the distributed environment is listed below:

- GPU: NVIDIA Tesla K80
- CUDA version: 11.0

### 5.2 Software

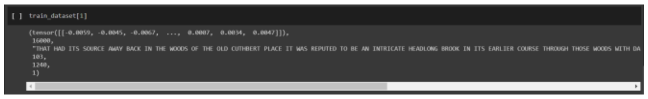
The entire model development was done using the PyTorch deep learning library. [9] PyTorch is an open-source deep learning library based on the Torch library and is primarily developed and maintained by Facebook's AI Research lab. PyTorch combines ease of use with a NumPy like syntax with powerful features like automatic differentiation and multi-GPU support for distributed computing.

For distributing training over a GPU cluster, we have used the PyTorch distributed package included in the PyTorch library. [12] PyTorch distributed implements distributed synchronous SGD by making use of the Distributed Data Parallel (DDP) parallel computing paradigm and replicates the model on every worker (GPU). Every model replica is fed with a different set of input samples. DDP synchronizes each model replica and overlaps it with the gradient computations to speed up training.

### 5.3 Dataset

The dataset we are using is the LibriSpeech corpus which is derived from audiobooks that are part of the open source LibriVox project and contains 1000 hours of speech sampled at 16 kHz. [1] The data consists of automatically aligned and segmented English read speech from audiobooks with the corresponding book text and is suitable for training speech recognition systems. The corpus is freely available and is easy to access using the torchaudio package in the PyTorch library. For initial development, we used a subset of the clean English training set which consists of about

100 hours of transcribed audio data. Each sample of the dataset contains the waveform, sample rate of audio, the utterance/label, and more metadata on the sample. For later development, we have used the 360 hours long train dataset which is about 23 GB in size. The data has 110K audio samples and training can only be done in small minibatches as the entire dataset is very large to fit into the memory of a GPU.



```
[ ] train_dataset[1]

    (tensor([[-0.0059, -0.0045, -0.0067,  ...,  0.0007,  0.0034,  0.0047]]),
     16000,
     "THAT HAD ITS SOURCE AWAY BACK IN THE WOODS OF THE OLD CUTHBERT PLACE IT WAS REPUTED TO BE AN INTRICATE HEADLONG BROOK IN ITS EARLIER COURSE THROUGH THOSE WOODS WITH DA
     103,
     1240,
     1)
```

**Figure 5**: Snapshot of a single training example

### 5.4 Pre-processing

The first step in the data pre-processing pipeline is to transform the raw audio signals into Mel Spectrograms. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies through time. A Mel spectrogram converts the frequencies to a non-linear scale such that the frequencies appear equally distant to the listener. So, essentially a spectrogram is a 'picture' of a sound from which features can be extracted.

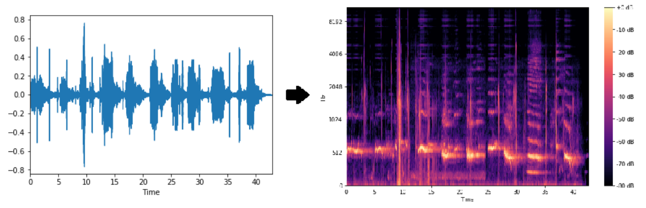


**Figure 6**: Transforming an audio signal to a Mel spectrogram

We have created a preprocessing script for transforming the audio signals to Mel spectrograms and applying the frequency and time masking transformations for augmenting the data.

| Dataset | Size (GB) | Run time (s) |
|---|---|---|
| train-clean-100 | 6.3 | 345.55 |
| train-clean-360 | 23 | 1044.47 |

### 5.5 Hyperparameters

For every minibatch on a worker node, the following configurations were chosen to compute gradients for every epoch during training and validation.

- Epochs: 20
- Learning rate: 0.0005
- Batch Size: 10

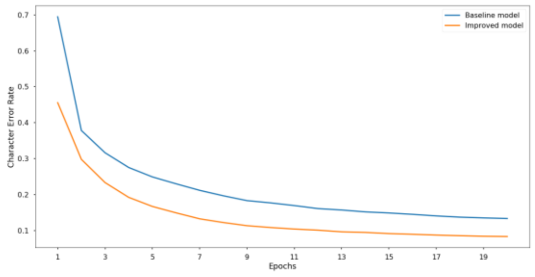
### 5.6 Results

For comparing model performance, we aim to use two metrics that are commonly used in speech recognition systems. Both metrics are calculated as Levenshtein distances which can be defined as the minimal quantity of insertions, deletions, and substitutions of words/characters for conversion of a hypothesis to a reference.

1. Word Error Rate (WER) – WER is the Levenshtein distance on a word level between the predicted word sequence and the actual word sequence.
2. Character Error Rate (CER) - CER is the Levenshtein distance on a character level between the predicted word sequence and the actual word sequence.

Word Error Rate is a robust measure of performance of a speech recognition model and is considered as the industry standard in evaluating ASR systems.

For evaluation of our model, we measure the average error rate on a test minibatch of size 10 after training for 20 epochs.



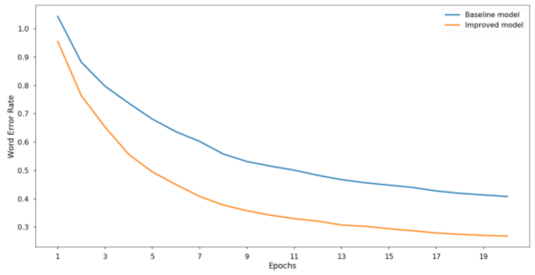


**Figure 7**: Comparing average Character Error Rate (CER) and average Word Error Rate (WER) for both models

| Model | Average CER | Average WER |
|---|---|---|
| DeepSpeech2 | 0.17 | 0.45 |
| Improved DeepSpeech2 | 0.1 | 0.26 |

The improved model achieves an average word error rate of 0.26 on a test minibatch of 10 audio signals after training for 20 epochs, a 42% improvement on the baseline model performance.

For speeding up model training we parallelized training over 6 NVIDIA Tesla K80 GPUs. The number of parameters and train time for both the models is tabulated below:

| Model | # of parameters | Train time (hrs) |
|---|---|---|
| DeepSpeech2 | 9M+ | 18 |
| Improved DeepSpeech2 | 23M+ | 28 |

## 6. CONCLUSION

### 6.1 Summary

At the end of this project, we were able to build a robust speech recognition model that is trained on large amounts of spoken English audio data. Our model achieves an average word error rate of 0.26 on a test minibatch of 10 audio signals after training for 20 epochs. We have used techniques like SpecAugment so that the model generalizes well and does not overfit on training data. As evidenced by the performance on the unseen test dataset, we believe that the model can achieve comparable performance on any English Speech-to-Text corpus.

### 6.2 Findings

As hypothesized, the architectural changes we made improved the performance of the speech recognition model although the additional number of parameters increases the training time. However, the training time does not follow a linear relationship with the number of parameters. The training time for the improved model with more than double the number of parameters as the baseline model is less than twice the time it took to train the baseline model.

For future studies, it would be worthwhile to explore an attention transformer based speech recognition model and compare the performance with the current model we have developed.

### 6.3 Challenges

The greatest learning through this project was getting to understand distributed training of deep learning models and how to implement the same in the PyTorch library. The learning curve was steep and the official documentation available on PyTorch distributed is insufficient. So, we had to refer to independent blogs and refer to the PyTorch forums for clarification on many implementation details. PyTorch Distributed DataParallel is incompatible with iPython notebooks, where our initial development was carried out. Spawning multiple processes can happen only using a command line interface. So, we had to migrate our codebase from iPython notebooks to python files.

## 7. ACKNOWLEDGMENTS

We would like to wholeheartedly thank Prof. Judy Fox for her support and guidance throughout this project. Prof. Fox always made herself available for feedback meetings and gave us directions on how to steer the project to successful completion. We would also like to thank Selehettin Akkas, the associate instructor for the course. Selahettin provided us with timely help in providing us with access to the Romeo GPU nodes without which it would have been difficult to complete this project successfully.

## 8. REFERENCES

[1] Gillick, D., S. Wegmann, and L. Gillick. "2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)." (2012).

[2] Amodei, D et al; Deep Speech 2: End-to-End Speech Recognition in English and Mandarin; arXiv preprint arXiv:1512.02595, 2015

[3] Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences." arXiv preprint arXiv:1404.2188 (2014).

[4] Wang, Yisen, et al. "Residual convolutional CTC networks for automatic speech recognition." arXiv preprint arXiv:1702.07793 (2017).

[5] Graves, Alex, et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." Proceedings of the 23rd international conference on Machine learning. 2006.

[6] Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017).

[7] Smith, Leslie N., and Nicholay Topin. "Super-convergence: Very fast training of neural networks using large learning rates." Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications. Vol. 11006. International Society for Optics and Photonics, 2019.

[8] A. Hannun; Sequence Modeling with CTC, Distill, 2017

[9] https://pytorch.org/

[10] A. Graves, A. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 10.1109/ICASSP.2013.6638947.

[11] Park, Daniel S., et al. "SpecAugment: A simple data augmentation method for automatic speech recognition." arXiv preprint arXiv:1904.08779 (2019).

[12] https://pytorch.org/tutorials/intermediate/dist_tuto.html

[13] Chan, William, et al. "Listen, attend and spell." arXiv preprint arXiv:1508.01211 (2015).

[14] Hannun, Awni, et al. "Deep speech: Scaling up end-to-end speech recognition." arXiv preprint arXiv:1412.5567 (2014).