DS 4400 Final Project Write-up

Neil Patel

Abstract

DeepSpeech is a paper that details an architecture for doing automatic speech recognition that was, at the time of its release, significantly better performing than other leading speech recognition models [1]. At the same time, it was much simpler than many of those other models, relying on a straightforward recurrent deep learning network. In this project, I attempted to replicate the model used by the team for DeepSpeech in Pytorch, and using a smaller dataset and less computing power, replicate the results they achieved. I was unable to fully replicate the state-of-the-art results, but I was able to produce a model that partially learns, using the DeepSpeech Architecture.

Introduction

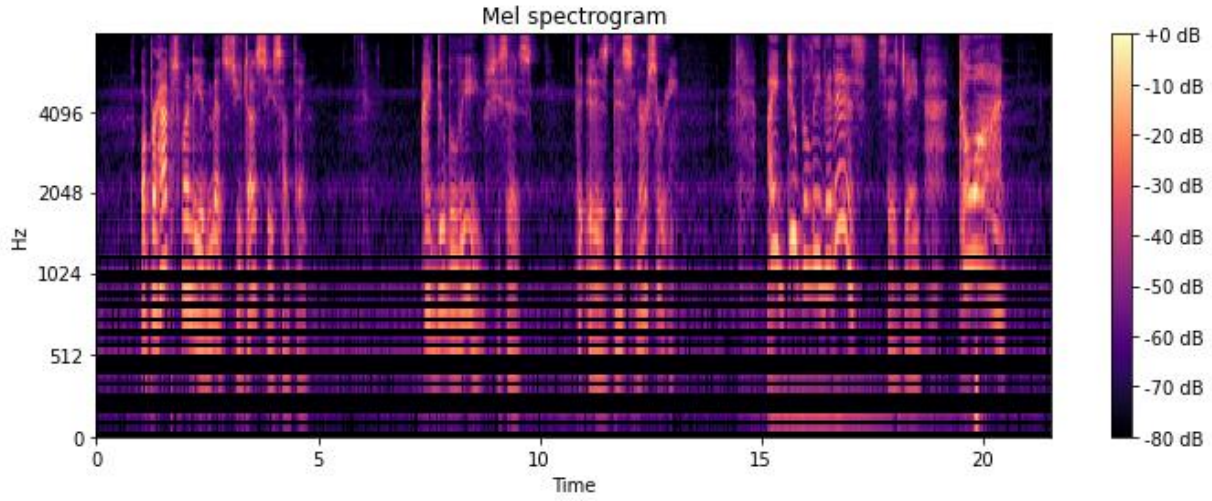
Automatic Speech Recognition, or the task of transcribing spoken language into text, is a task that was often considered to be very difficult. The state-of-the-art models at the time that DeepSpeech was written often contained many complex hand-designed components. DeepSpeech's model, in comparison, was much simpler, consisting of a Bi-directional Recurrent layer, surrounded by fully connected layers. It also achieved very impressive results, with a Word Error Rate of below 10% on clean datasets, and around ~20% on noisier datasets, which was below the best error rates from completing models by Apple and Google at the time.

For my implementation of the speech recognition model, I would obviously have access to much less computing power, and additionally the amount of data that I could feasibly train on was also limited by the time and computing resources I had. I decided to use the LibriSpeech Corpus as my dataset, as it was the most commonly used freely available dataset [2]. This dataset consists of read English Speech compiled from audiobooks. There is a smaller training set consisting of 100 hours of clean speech, which I used due to the entire dataset being too large to feasibly download and work with. This dataset also ended up being fairly large, with the total size of the test and train sets being around 7 GB.

Experimental Setup

Data Processing

The first step in the process was to transform the data into a form that the model could use. In the original paper, the speech data was input as a spectrogram to the model. The spectrogram format allows you to view which frequencies are present in the signal over time. The original data files were in a waveform format, so they had to be transformed into a Mel spectrogram, which converts the frequencies to a new 'mel' scale which help emphasize frequencies humans can hear [3]. Each speech fragment is converted to a mel spectrogram when it is input into the model using Torchaudio's built in MelSpectrogram transformation. Each sentence also has a corresponding English text transcription to use as a label. This label is converted to a integer array, by mapping each letter to a number, e.g. a to 1, b to 2.

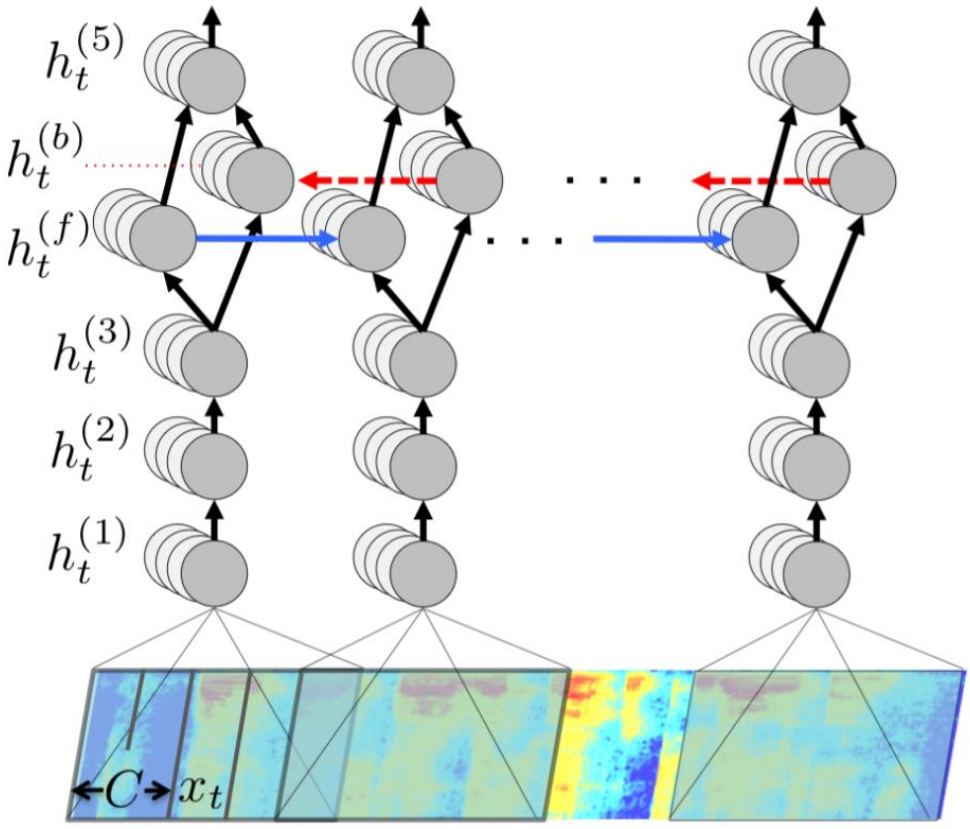


An example Mel Spectrogram from the dataset.

Model Details

My implementation of the DeepSpeech model follows closely to the original paper's implementation. The model has 5 layers of hidden units, and the first three are fully connected linear layers, with a clipped ReLU activation function, and a dropout of 10%. The clipped ReLU has a maximum value of 20. The fourth layer is the bi-directional recurrent layer, and consists of a forward recurrent layer, and a backward recurrent layer. In the paper, they state that the reasoning behind using a simpler RNN architecture vs an LSTM is that each LSTM cell requires computing and storing multiple responses at each step, which can become a computational bottleneck. By using the simpler RNN model, you can speed up the computation time required

drastically. The input from the linear layers is passed into both recurrent layers, and the outputs are concatenated together to produce the total output. This output from the RNN is then passed through one more linear layer, with another clipped ReLU, to produce the final output of the model. The output has a size of 29, to account for the probability of the frame being each of the 26 letters of the alphabet, a space, an apostrophe, or a ‘blank’ character which is used for the loss. A softmax is taken over this output to find out which letter the model predicts for that frame, and the output is passed along with the label for that sentence into the loss function, which is a CTCLoss. This loss is then backpropogated through the model to update the weights and biases.

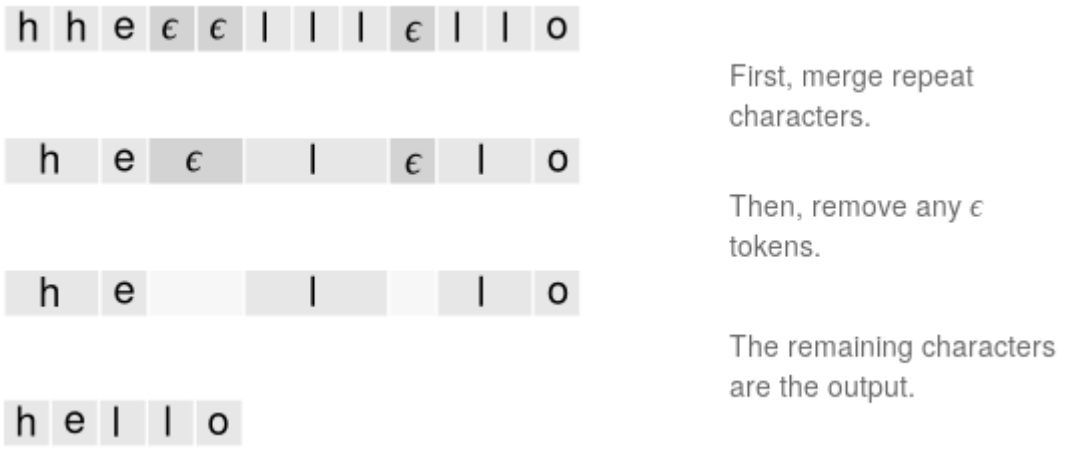


The DeepSpeech Model Architecture.

CTCLoss

The particular loss used for this model is the CTC, or Connectionist Temporal Classification, loss. This loss function works by calculating the most likely probabilities at each time frame, and combined repeated values together into one value. This works for speech recognition because, in a sound wave, there may be multiple frames that all account for one

letter. By combining all these frames together to the one letter they represent, the CTC loss is able to compare sequences of different lengths together.



How CTCLoss produces predicted words and sentences.

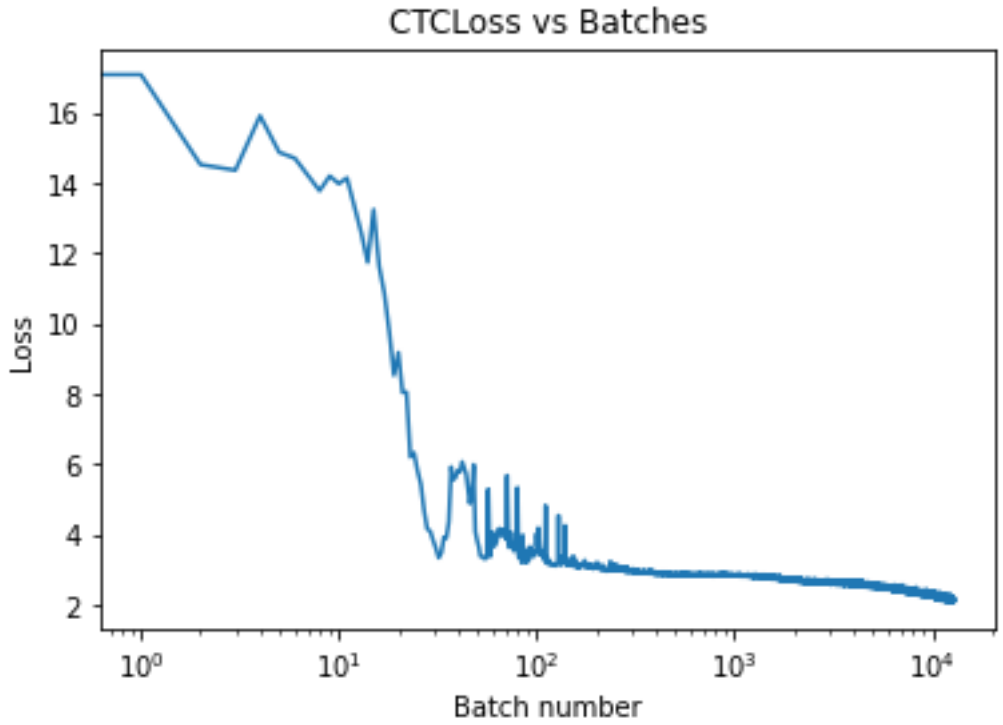
Training

When training the model, I tried various combinations of hyperparameters. The final set of parameters I ended up using was a learning rate of .0003, with a batch size of 20, for 9 epochs. Higher batch size values would be faster, but would occasionally cause the system to run out of memory and error. I used the same optimizer used in the original paper, Nesterov's Accelerated gradient, with a momentum value of .99.

Results/Discussion

Due to the fact that I did not have nearly as much training data or time to run the model as the researchers, I was unfortunately not able to replicate the state-of-the-art results. The main metric utilized in automatic speech recognition is word error rate, or character error rate. When running my model, however, the word error rate never got to a point where I would actually be able to match a significant number of words. However, at the point when the loss was minimized, I was able to match similar sounding phrases as the labeled ones. While my word error rate was still 100%, when examined, many of the sounds were close to being replicated. The output was generated through a 'greedy' algorithm, which applied a softmax to the output to produce the most likely sound at each time frame, and removed the repeated ones. While the outputs are mostly gibberish, phonemes from the input can be seen, and so I believe that with

more data or a more optimized model, it would be able to learn these inputs. However, if I run the model for more than 9 or 10 epochs, the training loss starts to increase again, and when I tested the model after 20 epochs, the output was much worse and did not resemble the labels at all.



The loss graphed over 9 epochs, with the total number of batches run through on the x-axis.

Label Sentence	Model Output
marie sighed	oe san
one hardly likes to throw suspicion where there are no proofs	wan co at toos stasanten co
mother dear father do you hear me	mae te fote e en
she sat down in a rocking chair and clasping her hands in her lap rocked slowly back and forth i'm sorry said beth	sis a o ete oe sat acos the an ta e lo to oc so bote foa a sot sat fat

Some examples of labels from the test set, with the corresponding model outputs. While no actual words were translated, many sounds and phonemes can be seen.

Since I can see some of the features of the input being detected in the model output, I believe that with more training and data, this model would have the capacity to learn to match words.

Conclusions/Future Work

While I was unable to replicate the state-of-the-art results achieved by the research team behind DeepSpeech, I was able to replicate their architecture and setup in Pytorch, and was able to have the model begin to understand speech. There is a lot of future work that can be done, whether it be continuing to tune parameters and expanding the model to try to optimize what can be learned with this dataset, or adding the full LibriSpeech Corpus, given more computing power, to train the model on a more extensive set of data.

Some changes that could be made could also be changing the optimizer used, as well as modifying the learning rate more over time. The model currently slowly converged to a low point, but then after this jumped up again. Maybe modifications to the learning rate and optimizer could help the model find a more optimal minima and solution. Additionally, expanding the size of the model, and the batch size, would allow more data to be trained on faster, however my current limitations are with the amount of memory available on Colab, which quickly runs out at higher batch sizes. Additionally, modifications to the input, such as random jittering and frequency masking, are done in the paper, and could perhaps improve the results with my model.

Works Cited

1) Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G. et. al (2014). Deep Speech: Scaling up end-to-end speech recognition. Retrieved from https://arxiv.org/abs/1412.5567
2) LibriSpeech ASR corpus. (n.d.). Retrieved December 15, 2020, from http://www.openslr.org/12
3) Roberts, L. (2020, March 14). Understanding the Mel Spectrogram. Retrieved December 15, 2020, from https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53