Comparison of Voice-to-Text Neural Networks

Introduction

Speech in the form of words, phrases, and sentences with some grammatical rules is the primary and most important form of communication. Inspired by the widespread use of Alexa, Siri, and Google Home, we wanted to build our own small speech recognition system. But building an automatic speech recognition system is a very complex problem. Big giant companies like Google, Apple, and Amazon have spent years of development time, build deep models with million hours of data on supercomputers. So we decided to use the already existing opensource framework to further our work. We came across an end-to-end speech to text recognition framework DeepSpeech2.

DeepSpeech 2 is an end-to-end pipeline which leverages High Performance Computing and deep neural network to recognize the speech. The model built is very robust which allows handling a variety of speech including noisy environment, accent, and different languages. The framework is primarily built on an4, librispeech, Ted-Lium database.

The rest of the report will follow the following structure. A description will describe my contribution in the project which is the refactoring of the codebase and applying transfer learning approaches, results will provide an analysis of the transfer learning approach and conclusion.

Description

As this project involved using the already existing codebase, it was important to understand how does the current code works. At the very beginning, I tried to run the code and generate the output. The codebase was structured in three segments mainly, data loader and preprocessing, model building, and transcription. The data segment has scripts to download

data from an4, voxforge, librispeech, common voice, and tedlium. These scripts download the data from the URLs mentioned in the file. The audio files are in .wav format and the associated transcription file is saved in .txt files. There is a manifest file that saves the path of the .wav and txt files in the comma-separated fashion. The audio files were read using a scipy.io.wavfile package. All audio files were considered to be sampled at 16000 sample rate. All the signals were normalized by 32767 which is a max value for 16 bit PCM. The amplitudes (loudness) of the sound waves are not very informative as they only talk about the loudness of audio recording. Hence we need to transfer the signal to the frequency domain to understand it better. The frequency-domain will show what all different frequencies are present in the audio signal. For this conversion, we used a short-time Fourier transformation using librosa package. The function was applied on a window size of 0.2 which generated 320 fft points and the slide rate of the window function was 160 points. Hence there was an overlap of 50% between the two windows. This function returns a spectrogram from which magnitude and phase are separated out. Below is the snippet of the code:

```
D = librosa.stft(y, n_fft=n_fft, hop_length=hop_length,
                 win_length=win_length, window=self.window)

# magphase = separate a spectorgram into magnitude and phase, see more at
# https://librosa.github.io/librosa/generated/librosa.core.magphase.html
# spect is n x m (161x641)
# phase is n x m (161x641)
spect, phase = librosa.magphase(D)
```

Model building:

The spectrogram signal is passed through the convolution 2D layer of 32 channels. The idea behind using the convolution layer is to capture the important information and features from the spectrogram image. The 1st convolution layer which has an input of 1 channel is traced using the kernel of size 41X11. The spectrogram image is padded with 20X5. Every layer is batch normalized to optimize the process. Hard tan h function is used as an activation layer. The next convolution layer reduces the size of the kernel by half (21,11). The reduced

spectrogram image of 32 channels is passed through the recurrent neural network of LSTM. The 5 layered bi-directional LSTM has input units of 1312 (32*41 feature space) and 1024 hidden units. RNN layer is further connected with the fully connected linear layer 29 neurons. The output of the neurons is passed through a softmax layer to get a class probability score. Stochastic Gradient Descent is used as a training algorithm. And the evaluation criteria used is CTC loss. CTC loss is a connectionist temporal classification that works well on the aligned data. CTC is alignment-free and works by summing over the probability of all possible alignments between the input and the label. Figure 1 shows a network architecture diagram of the librispeech pre-trained models.

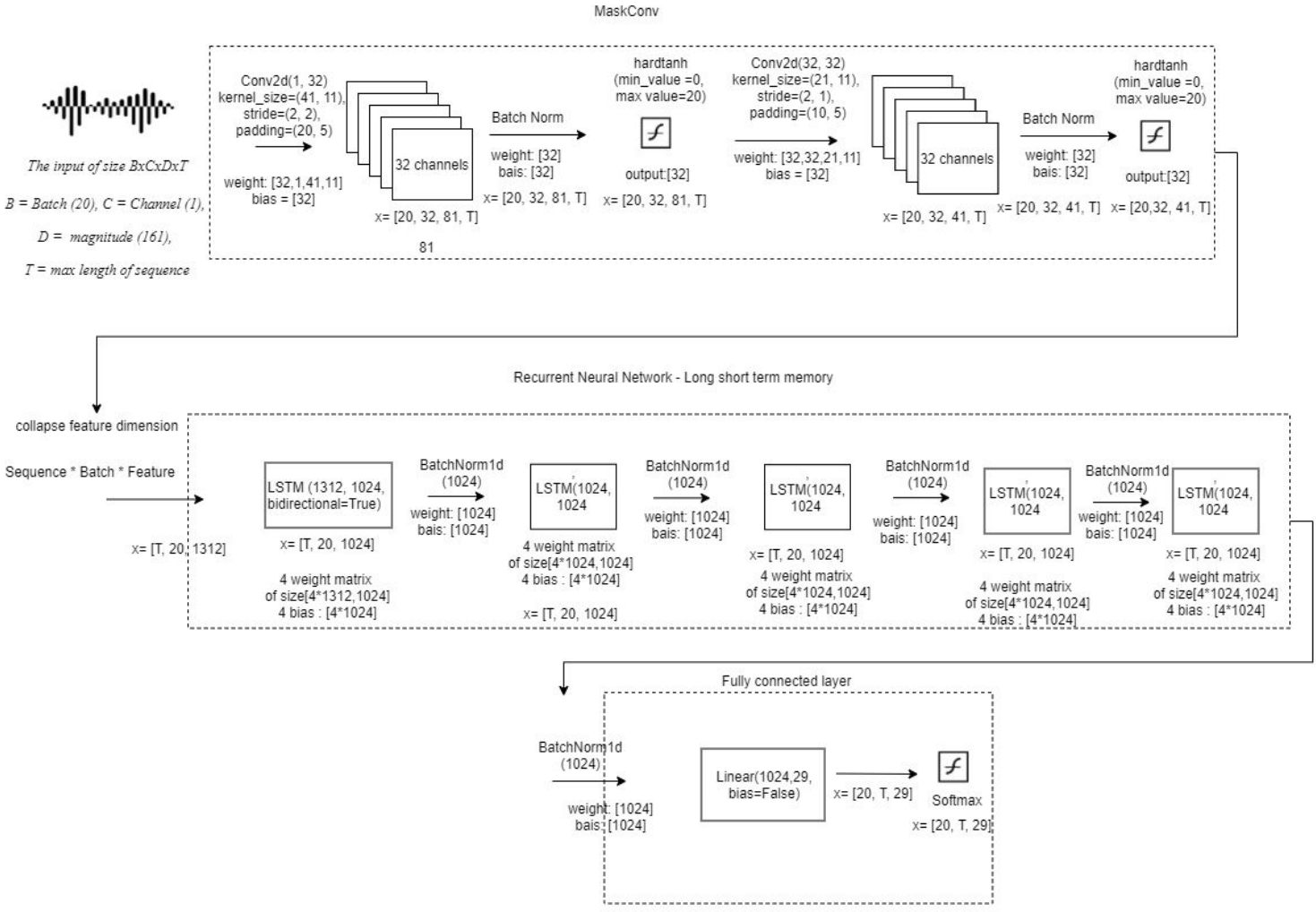

Figure 1: Network Architecture diagram for librispeech model

The output of the network has a shape of [batch size, max sequence length in a batch, 29 labels]. The label size is 29 as it includes 26 letters, space, comma, and blank.

Transcribe:

The probability matrix is further transcribed to using a greedy decoder approach. The greedy search selects most likely at each step in the output sequence. The whole probability matrix of the output labels is traversed and these values are converted into sentences or words.

Figure 2 shows the original flow of the codebase. The code either starts from train.py file or transcribe.py file

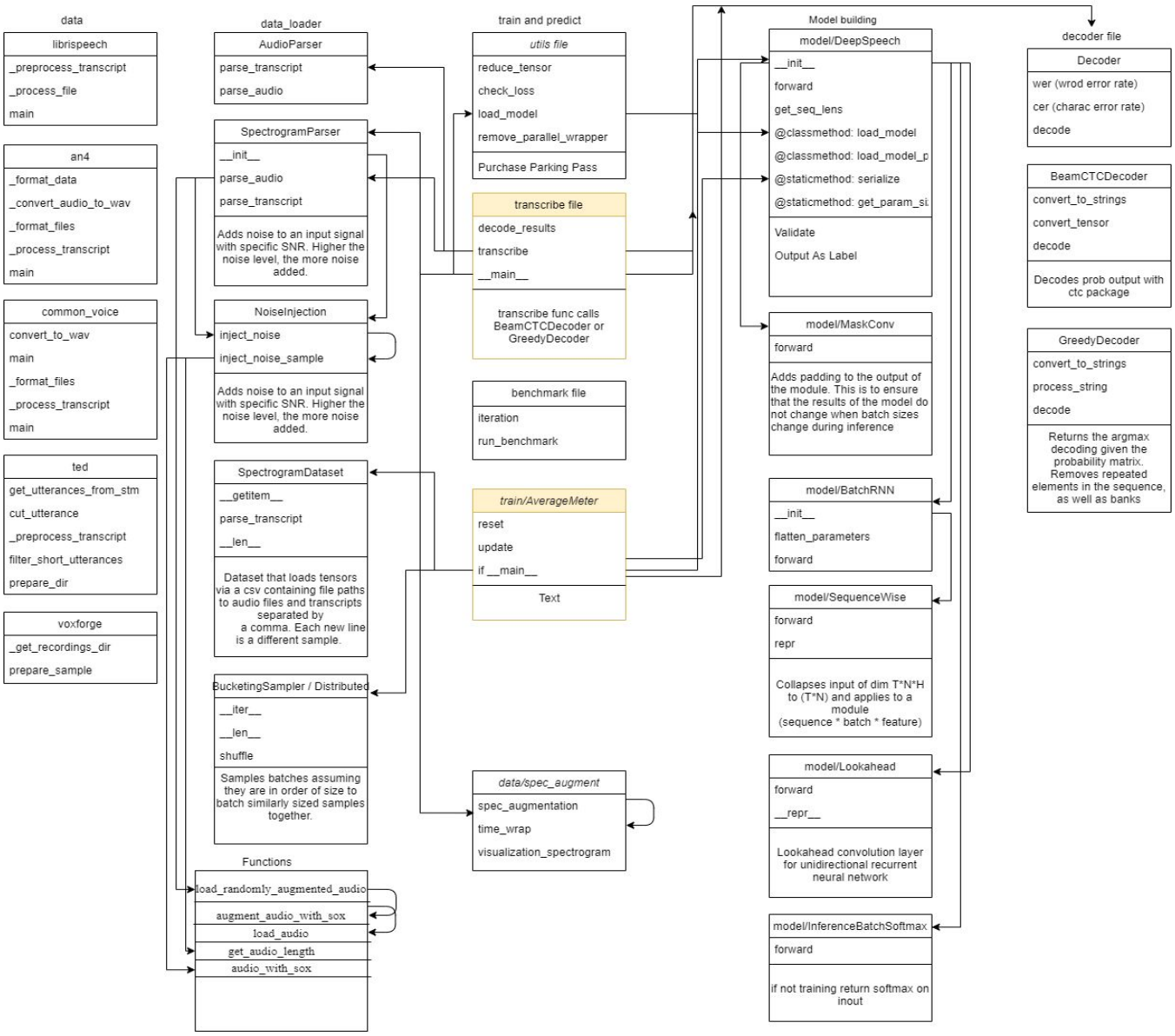

Figure 2: The code flow diagram of Deep Speech 2 codebase

Transfer learning approach

Initially, as our first experiment, we decided to test the generalizability of the Deep Speech 2 model on the unknown dataset. So I tested 500 randomly sampled voxforge files wit the pre-trained models and compared it against Google API. The accuracy metric used to compare these models are word error rate and character error rate. WER is the number of errors divided by the total words. And the number of errors is the sum of the substitutions, insertions, and deletions that occur in a sequence of recognized words. Character error rate does the same calculation character-wise. As the librispeech model had the least error rate I further fine tuned the pre-trained model to compete with the Google API.

For the transfer learning approach, I used different parameter settings to further fine-tune the models. I started with the librispeech model and tuned hyper-parameters such as the hidden size of RNN 500 and RNN type as Gated Recurrent Unit (GRU). GRU is just like LSTM with a forget gate but has fewer parameters than LSTM as it lacks an output gate. Synchronous Stochastic Gradient Descent (SGD) was used as a training algorithm with a learning rate of 3e-4 and a momentum of 0.9. In order to optimize the learning process and not to miss the global minimum, I adjusted the learning rate by 1.1 after every iteration. This method of starting at a relatively high learning rate and gradually lowering it is called learning rate annealing. The model training starts with the loading of the weights and bias of the pre-trained model. The data generator script generates 20 audio files in every batch. This number was decided after checking optimal time performance and the CTC loss. The data generator returns input, targets, input_percentages, and target sizes. The input percentage is calculated as the length of the input sequence divided by the maximum length of the sequence in that batch. These inputs are then moved to the GPU to perform model building exercise. The % input sizes are further used to mask the maximum length of the sequence in the corresponding batch. The input is passed through 5 layers of 2D Conv and 5 layers of GRU and fully connected linear layer. The output of the fully connected linear layer is of shape batch size, max length of the sequence, 29 labels. The probability matrix generated at the end of the model is further passed into decode to

generate the sequence. The accuracy metric used is CTC loss, WER, and CER for every epoch. The model is saved after epoch and overwritten by the better model in the further epochs. These models and optimizers are loaded Nvidia apex AMP function. AMP is automatic mixed precision. AMP allows users to easily experiment with different pure and mixed precision levels. 70 epochs were used for the first iteration in the fine tuning process which took around 71secs. As the results were not satisfactory I changed the RNN type to bi-directional LSTM and ran an iteration with the same settings as before. The results were still not satisfactory hence I increased the hidden size of RNN to 1000 and keeping the rest of the settings the same. The results were still not up to the mark so I decided to increase the training dataset. We further increased the training dataset to ~7500 audio files from voxforge. An iteration was run with the same hyperparameter setting. But as the data size had increased, the execution time for each epoch was ~ 5 hours. Hence I ran only 3 epoch.

Results

The results of the first experiment, testing the generalizability of the Deep Speech 2 pre-trained models are summarized in the below table. The pre-trained models were tested with the randomly sampled 500 voxforge audio files, as this dataset is entirely unknown for the models. The models were compared using the evaluation metric as word error rate and character error rate. Within the pre-trained models, the An4 data set had the maximum WER and CER of 10.05% and 30.80%, followed by the Tedlium model with a WER and CER of 4.86% and 9.23%. The LibriSpeech model had the lowest WER and CER of 2.27% and 4.28%, respectively. This could be due to the similarities between the training and testing datasets, as the Voxforge data and LibriSpeech data are both audiobook corpora. Moreover, the LibriSpeech dataset's size is relatively higher than the An4 and Tedlium datasets by hundreds of hours. Unsurprisingly Google has the least error rate of 1.61% and 3.73% when compared to the pre-trained models. The pre-trained models took ~4 mins of evaluation time as these files were evaluated on GPU. Whereas this was doubled in the case of google API as this is an external interface call, the backend processing and communication time may lead to a delay response time. Overall as the

librispeech pre-trained model had a better performance on the voxforge dataset when compared to the other pre-trained models, we further fine tuned this model to compete with the Google API error rate.

Model	WER(%)	CER(%)	Eval time
An4 - Pretrained	10.05	30.80	~4 mins
LibriSpeech - Pretrained	2.27	4.28	~4 mins
TED-LIUM - Pretrained	4.86	9.23	~4 mins
Google Speech-to-Text	1.61	3.73	~8 mins

Table - Word Error Rate and Character Error Rate on pre-trained models

Starting with the librispeech pre-trained models, the first iteration was executed using RNN type as GRU and specified hyperparameters of 5 hidden layers and 500 neurons in each recurrent hidden layer. The model was trained for 70 epochs with a batch size of 20 audio files. The WER and CER were 3.09% and 6.09%, which had increased from the initial baseline of 2.27% and 4.28%. We ran the second iteration by changing the RNN type to bi-directional LSTM and keeping the hidden layer, hidden size, number of epochs, and batch size the same as before. In this case, the error rate increased to 3.15% and 6.35%. Hence we decided to increase the hidden layer size to 1000 neurons, keeping the rest of the parameters the same. Although the error was reduced in comparison to the last iteration, the results were not satisfactory as we were aiming to compete with Google API score. Hence to get better performance we increased the training size to ~7.5k voxforge audio files. The training with the huge dataset took exceptionally long, ~ 5 hours per epoch. Hence the results were evaluated after 3 epochs (~15 hours of execution time). The CER (3%) was better than the Google API (3.73%). Even in the case of WER, there was only a 0.05% difference between our highly-tuned model and Google API. This shows that the pre-trained model, when trained with data similar to our evaluation

set, performs at par with the Google API. The below table summarizes the WER, CER, and evaluation time for the fine tuned models.

Model	WER(%)	CER(%)	Eval time
LibriSpeech - Pretrained	2.27	4.28	~4 mins
LibriSpeech - Pretrained -Transfer Learning (GRU)	3.09	6.09	~4 mins
LibriSpeech - Pretrained -Transfer Learning (LSTM)	3.15	6.35	~4 mins
LibriSpeech - Pretrained -Transfer Learning (LSTM+1000)	3.14	6.31	~4 mins
LibriSpeech - Pretrained -Transfer Learning (LSTM+1000)	1.66	3.00	~4 mins
Google Speech-to-Text	1.61	3.73	~8 mins

Table - Word Error Rate and Character Error Rate on the fine tuned models

Summary and conclusions

In conclusion, I learned that Deep Speech 2 pre-trained models that are specific to the particular data set are not very robust compared with the Google API. The Librispeech model is generalized the best out of the pre-trained Deep Speech 2 models, likely due to the larger amount of training data available as well as due to its similarities to the Voxforge dataset. Further in the transfer learning exercise, I learned that it is possible to train the open-source pre-trained models with the data in interest. But this did not work well when trained with the smaller dataset. It worked well when trained with larger dataset however this does not mean the model generalizability is increased. There was a limitation of the computation power, as I

was training the model only on 1 GPU. The paper suggests the use of multiple GPUs to improve performance time.

The biggest part of this project was to understand the scalability, coding standards, and methodology of the production code. As shown in the code flow diagram, the code base is very complex with lots of pieces and specific tuning purposes. I first learned to decode the original codebase and strip out the parts which were needed for the model training exercise. I also learned about how to integrate different neural network components together to get better accuracy models.

The models can be further trained on a larger corpus of the dataset, probably all dataset- an4, voxforge, commonvoice, librispeech together. This will make the model robust and generalizable. Also adding data augmentation techniques like adding noise to the audio signal, trimming the voice, changing the sampling rate can help to improve the model results. It would be also good to explore training visualization packages like tensorboard or visdom to closely monitor and better understand the training process.

Code calculation

As most of the code was downloaded from the sources, my additions to the code base include code commenting, removing parser arguments, and defining the parse arguments as a class.
% modified lines in the files:
Train.py = 448 - 65/448+ 77 *100 = 72.95%
Model.py = 323-0/323+20 = 94.16%

References

Baidu Silicon Valley AI Lab, B. S. (2015). Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. 28.

SeanNaren (2017-2020). https://github.com/SeanNaren/deepspeech.pytorch

Chaudhary, Kartik. "Understanding Audio Data, Fourier Transform, FFT, Spectrogram and Speech Recognition." *Medium*, Towards Data Science, 10 Mar. 2020, towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520.

Dossman, Christopher. "A Data Lake's Worth of Audio Datasets." *Medium*, Towards Data Science, 14 Nov. 2018, towardsdatascience.com/a-data-lakes-worth-of-audio-datasets-b45b88cd4ad.

Panayotov, Vassil, et al. "Librispeech: An ASR Corpus Based on Public Domain Audio Books." *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, doi:10.1109/icassp.2015.7178964.

Scheidl, Harald. "An Intuitive Explanation of Connectionist Temporal Classification." *Medium*, Towards Data Science, 4 Dec. 2018, towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c.