Comparison of Voice-to-Text Neural Networks

DATS 6203 : Final Project

Anamay Agnihotri · Tanaya Kavathekar · Alex Cohen

Outline

- Introduction

- Defining the Problem

- Data Sources and Collection

- Understanding the Networks

- Experiment Methodology

- Preliminary Results

- Final Results and Contributions

- Conclusion and Next Steps

Translating audio to text is a complex problem

- Audio data is extremely complex, relying on the translation from physical speech to electrical signals to encoded ones and zeros computers can interpret

- Variability in speech and acoustics, attributable to different languages, pronunciations and accents, along with lack of extensively labelled training data, make generalizable systems difficult.

- Speech recognition systems have become commonplace, with implementations like Siri, Alexa, Amazon's Echo, and Google Home becoming a part of people's everyday lives.

- Each implementation presumably has its own proprietary model architectures, which have been worked on for years and trained over 1000s of hours of data.

Problem Statement

Can open source Voice-to-Text neural architectures, either through pretrained models or transfer learning, compete with the proprietary, broadly developed translation models of tech giants like Google?

Data Sources

DeepSpeech 2

- An end-to-end, automatic speech recognition pipeline developed by researchers at **Baidu**, an AI organization.

- The DeepSpeech2 used here follows the **pytorch** implementation of github user **SeanNaren,** an ML research engineer from the UK.

- The repository has three pretrained models based on the following data sources:

    - **An4**: an alphanumeric database of census responses compiled by Carnegie Mellon University

    - **LibriSpeech**: A large-scale corpus of 1000 hours of English audio-book recordings

    - **TED-LIUM**: a collection of 1495 recorded and transcribed TED talks

- **Voxforge** data, a cleaned and accented English audiobook dataset from LibriVox , was used for the model evaluation and all transfer learning purposes.

Data Manipulation

- All of the data came in the form of wav files (similar to an MP3 file), yet need to be converted to an input format the networks can understand

- Data transformations involved conversion using a Fourier Transformation and decomposition into magnitude and phase elements (done using the popular audio package LibRosa) to create a Spectrogram, which would be used as input to the DeepSpeech model
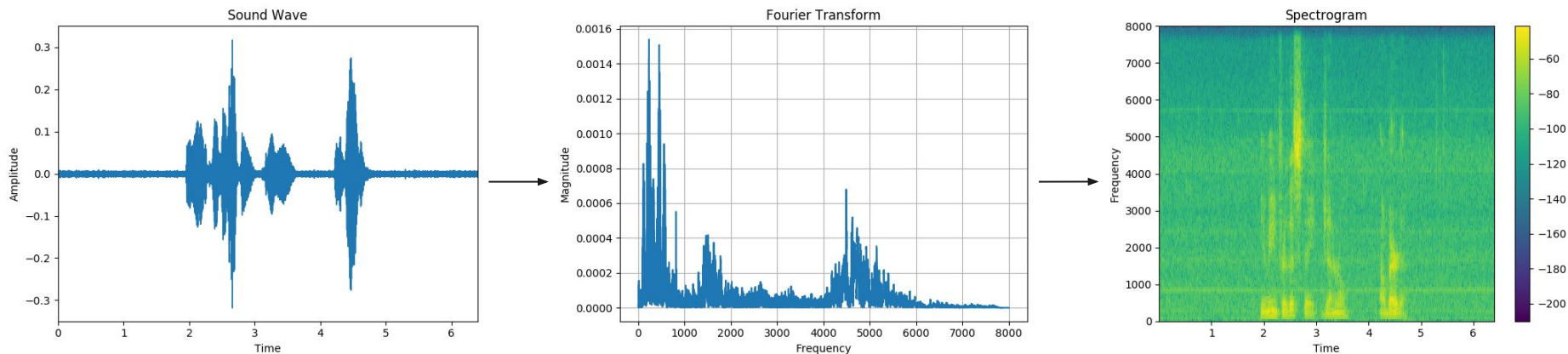
Text: 'YOU MEAN FOR THIS STATE GENERAL ALBERTA'

Understanding the DeepSpeech Network

- Inputs: Spectrograms of power normalized audio clips

- Outputs: graphemes (units of language). {a,b..z, apostrophes, spaces, blank}

- Hidden activation : clipped ReLU/hardtanh $\longrightarrow$ σ(x) = min(max(x,0),20)

- Subsampling by striding in the convolutional layers - not good for english models.

- Output layer uses softmax activation, computing the probability distribution for each grapheme per time step.
$$p(\ell_t = k|x) = \frac{\exp(w_k^L \cdot h_t^{L-1})}{\sum_j \exp(w_j^L \cdot h_t^{L-1})}$$

- 2D convolutions are used to convolve along (normalize) both the frequency and time axes

- Batch normalization improves speed of convergence without loss in generalization performance.
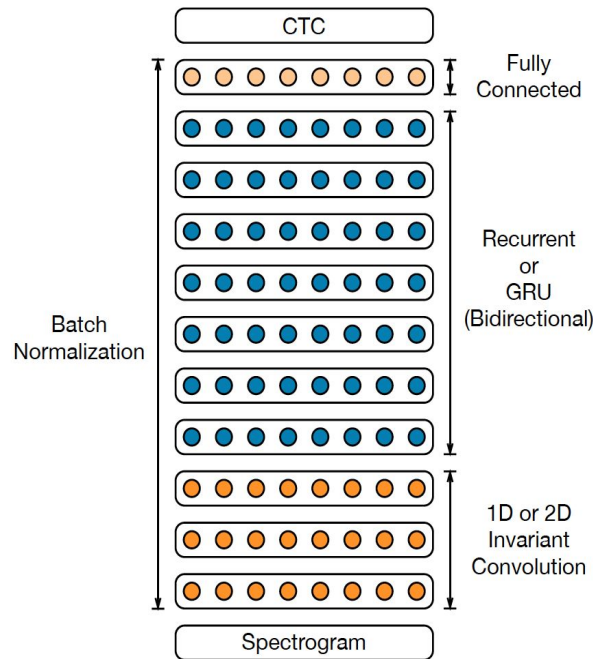
Understanding the DeepSpeech Network (Continued)

- Loss function: Connectionist Temporal Classification (CTC) loss

$$\mathcal{L}(x, y; \theta) = -\log \sum_{\ell \in \text{Align}(x,y)} \prod_t^T p_{\text{ctc}}(\ell_t | x; \theta).$$

- CTC loss is used in problems where having aligned data is an issue.

- Probability of each possible text alignment (path) is the product of the predicted probability of each character in that path.

- This alignment probability is summed up for all possible alignments of the label text. The negative logarithm of this summation is defined as the loss function, which is back-propagated.

- Optimizer: Synchronous SGD (reproducible, deterministic)

Google's Speech-to-Text API

- Google Speech-to-Text enables developers to convert audio to text by applying powerful neural network model.

- Recognizes more than 120 languages and can process real-time streaming or prerecorded audio.

- Ideal for quick prototyping for complex speech tasks like separating multiple speakers, detecting speaker language and transcribing over multiple channels (audio sources).

- Enable the API and obtain a private key (credentials) for a given service/billing account.

- Each API request returns multiple transcriptions, sorted by their confidence.

- Easy to use and reliable REST API.

Experiment

- Use the pre-trained models (an4, LibriSpeech, and TED-LIUM) to transcribe 500 test audio files from the Voxforge dataset.

- Use different parameter tunings (transfer learning) to train the pre-trained models on other Voxforge files and transcribe the same 500 test audio files.

- Use the Google Cloud Speech-to-Text API to transcribe the same 500 test audio files.

- Calculate the Character Error Rate (CER) and Word Error Rate (WER) for all models to evaluate performance

Preliminary Results

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

Transfer learning approaches

Trained a librispeech pre-trained model on Voxforge dataset with different parameter settings:

- Default settings: batch size =20, lr = 3e-4, learning-anneal = 1.1, momentum = 0.9
- Using 500 random sampled files with more than 1 sec and less than 15 secs audio file
  - hidden layers =5, hidden size of RNNs = 500, RNN type = GRU
  - hidden layers =5, hidden size of RNNs = 500, RNN type = LSTM
  - hidden layers =5, hidden size of RNNs = 1000, RNN type = LSTM
- Using 7636 sampled with more than 1 sec and less than 15 secs audio file
  - hidden layers =5, hidden size of RNNs = 1000, RNN type = LSTM

Final Results

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

Key contributions

- Strip and reformat the original source code (rewriting arguments, re-defining classes, etc.)

- Simplified code base to meet project requirements and increase ease of use/understanding

  - Updated with significant commenting to explain most existing lines of code

- Developed model evaluation comparison scripts, rewriting target metric code

- Conducted informed parameter tuning and transfer learning to fit the context of the Voxforge dataset, generating four new transfer-learning models

- Developed Google API transcription code

Conclusion and Next Steps

- The generalized Google Speech-to-text model outperforms the pre-trained DeepSpeech2 models.

- Our customized transfer learning approaches with the bi-directional LSTMs performed **marginally** better the Google API service on our test set, however may not be as generalizable.

- Next Step: Creating a model trained on all three datasets simultaneously to increase variety of audio data

- Next Step: Using data augmentation techniques in transfer learning models to increase robustness and generalizability.

- Next Step: Parallelization of GPU computing using CUDA could scale the transfer learning to larger datasets.

- Next Step: Incorporate training visualization that could better guide the transfer learning tunings.

Thank you!

Appendix

data

librispeech
- _preprocess_transcript
- _process_file
- main

an4
- _format_data
- _convert_audio_to_wav
- _format_files
- _process_transcript
- main

common_voice
- convert_to_wav
- main
- _format_files
- _process_transcript
- main

ted
- get_utterances_from_stm
- cut_utterance
- _preprocess_transcript
- filter_short_utterances
- prepare_dir

voxforge
- _get_recordings_dir
- prepare_sample

data_loader

AudioParser
- parse_transcript
- parse_audio

SpectrogramParser
- __init__
- parse_audio
- parse_transcript

Adds noise to an input signal with specific SNR. Higher the noise level, the more noise added.

NoiseInjection
- inject_noise
- inject_noise_sample

Adds noise to an input signal with specific SNR. Higher the noise level, the more noise added.

SpectrogramDataset
- __getitem__
- parse_transcript
- __len__

Dataset that loads tensors via a csv containing file paths to audio files and transcripts separated by a comma. Each new line is a different sample.

BucketingSampler / Distributed
- __iter__
- __len__
- shuffle

Samples batches assuming they are in order of size to batch similarly sized samples together.

Functions
- load_randomly_augmented_audio
- augment_audio_with_sox
- load_audio
- get_audio_length
- audio_with_sox

train and predict

utils file
- reduce_tensor
- check_loss
- load_model
- remove_parallel_wrapper
- Purchase Parking Pass

transcribe file
- decode_results
- transcribe
- __main__

transcribe func calls BeamCTCDecoder or GreedyDecoder

benchmark file
- iteration
- run_benchmark

train/AverageMeter
- reset
- update
- if __main__

Text

data/spec_augment
- spec_augmentation
- time_wrap
- visualization_spectrogram

Model building

model/DeepSpeech
- __init__
- forward
- get_seq_lens
- @classmethod: load_model
- @classmethod: load_model_p
- @staticmethod: serialize
- @staticmethod: get_param_si.
- Validate
- Output As Label

model/MaskConv
- forward

Adds padding to the output of the module. This is to ensure that the results of the model do not change when batch sizes change during inference

model/BatchRNN
- __init__
- flatten_parameters
- forward

model/SequenceWise
- forward
- repr

Collapses input of dim T*N*H to (T*N) and applies to a module (sequence * batch * feature)

model/Lookahead
- forward
- __repr__

Lookahead convolution layer for unidirectional recurrent neural network

model/InferenceBatchSoftmax
- forward

if not training return softmax on inout

decoder file

Decoder
- wer (wrod error rate)
- cer (charac error rate)
- decode

BeamCTCDecoder
- convert_to_strings
- convert_tensor
- decode

Decodes prob output with ctc package

GreedyDecoder
- convert_to_strings
- process_string
- decode

Returns the argmax decoding given the probability matrix. Removes repeated elements in the sequence, as well as banks