Automatic Speech Recognition with Sequence to Sequence Transformers

Name: Ilan Motiei
TAU Email: ilanmotiei@mail.tau.ac.il
Email: ilanmotiei1@gmail.com
ID: 212649701

Abstract

Similarly to previous works of Google's 'Magenta' team that used a seq2seq transformer from the NLP domain for the music transcription task [1] [2] - I have re-implemented the work for the ASR task.

Problem Description

Developing an ASR system using an 'off-the-shelf' seq2seq transformer from the NLP domain. An advantage of using an encoder-decoder transformer which auto-regressively generates the tokens, is that the model is less prone to spelling errors, and when trained extensively, may not need an additional language model on-top to perform well.

Method

Dataset:

'LibriSpeech' which contains:

- For training: 960h of spoken english audio along with transcriptions.
- For validating: 20h " ".
- For testing: 20h " ".

Model:

The network used is the T5 seq2seq transformer model. Specifically, I used the T5-small version, which has 60M parameters.

[1] SEQUENCE-TO-SEQUENCE PIANO TRANSCRIPTION WITH TRANSFORMERS
[2] MT3: MULTI-TASK MULTITRACK MUSIC TRANSCRIPTION

A log-mel-spectrogram is given to the model as input, with 512 mel-bins to match the embedding dimension of the original T5 model (which was trained on NLP tasks).

Because the space complexity of the model is quadratic in the input length, the input length in the original T5 is limited to 512, and as mentioned - so it is here.

Training:

In order to train the model, in each epoch, we iterate over all the files, and for each batch of files, we crop a random slice of 5 seconds of audio from each one, (which results in spectrograms with 500 time-steps), along with the corresponding transcription (which is determined by forced alignment with the Montreal Forced Aligner tool).
We enter the 500 spectrogram feature vectors into the T5 model, and the model is trained to minimize the cross-entropy loss between its prediction and the ground truth prediction.
As usual - teacher forcing is used when training.

Inference:

At inference time, we extract the spectrogram of the whole audio file to be transcribed, divide it into chunks of 512 (model's max input size), insert each to the model, which auto-regressively generates a list of the characters (tokens) spoken in each audio chunk, using beam search with a beam size of 4. In the end we concatenate the transcriptions of the model from all the chunks together for getting the final transcription.

Additional details:

Spectrograms were generated using torchaudio with a window size of 400 (25ms), and a hop length (stride) of 160 (10ms).

SpecAugment method was when training.
I.e. each spectrogram had a 50% chance of being augmented, in the following way:

- Between 1 to 80 of the frequencies (bins) in the spectrogram are masked (the amount is chosen randomly at each time and uniformly).
- Between 0% to 5% of the time vectors in the spectrogram are being masked (again, the amount is chosen randomly and uniformly).

Code Explanation

Framework: Pytorch.

My code (Link):

The repository also contains instructions on how to evaluate the model on LibriSpeech's test-sets, and for plotting the learning curves of the training process.

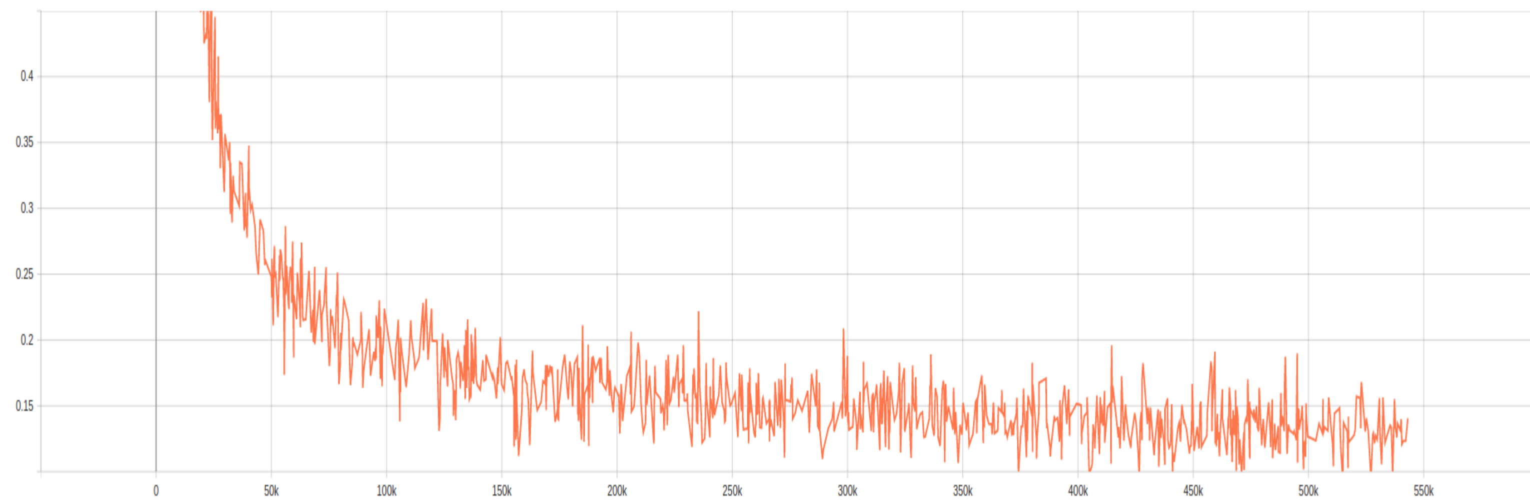
- dataset.py - contains utils for loading the data of LibriSpeech.
- dataset_stats.py - contains utils for measuring some statistics about the dataset.
- model.py - contains the model. For convenience the model is not a torch.nn.Module object, but pytorch_lightning.LightningModule object, which already contains utilities for fast training, checkpointing, and validating of the model.
- params.py - configurations for the experiments.
- preprocess.py - utils for combining between the alignments of the Montreal Forced Aligner to the LibriSpeech dataset.
- main.py - contains the training code. Running this file, (with no additional parameters), will start the training.
- eval.py - script for measuring model's performance on every subset of LibriSpeech after training.

Existing code:

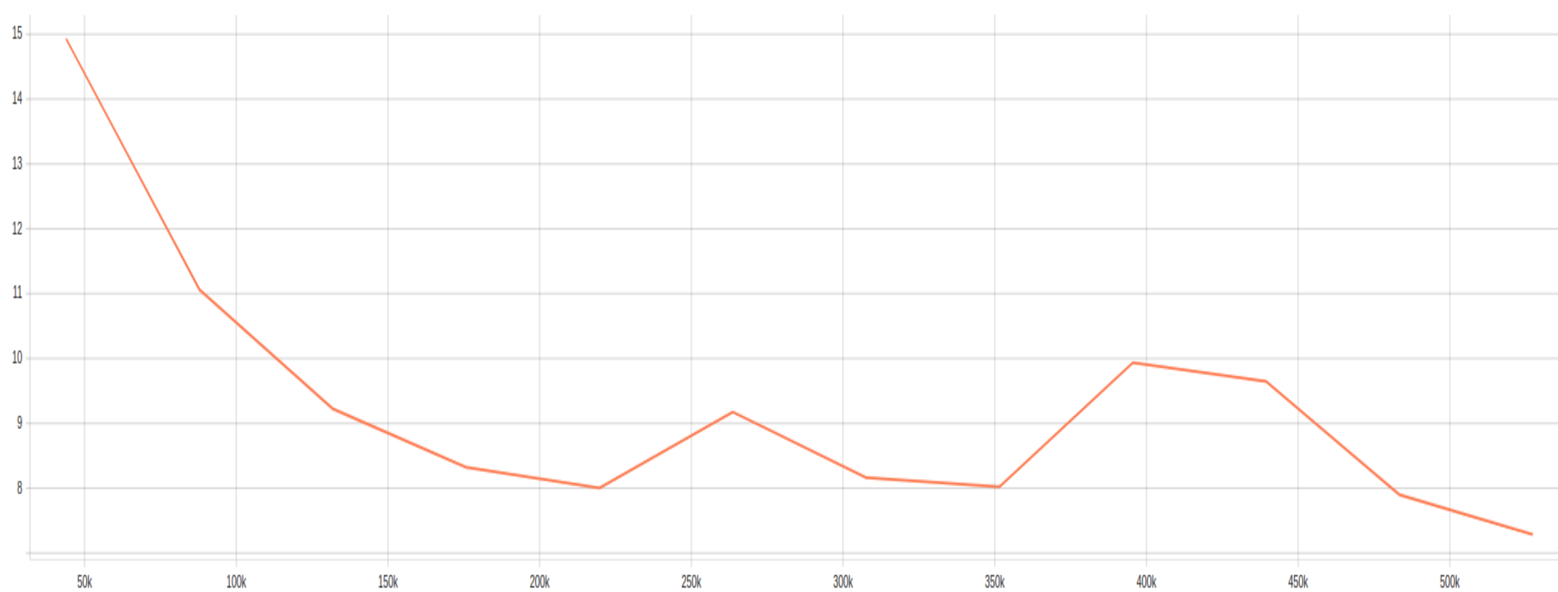
The alignments were taken from this repo (specifically 'condensed format' alignments were taken).

Results

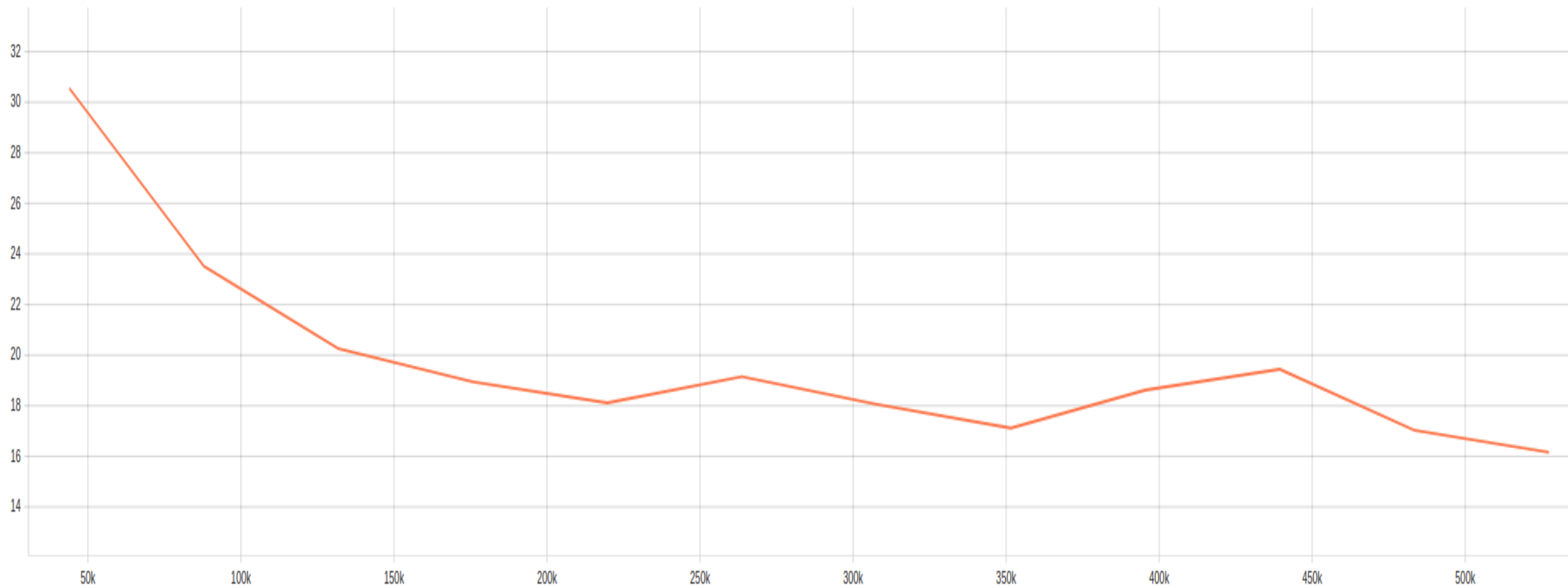
Training Loss



Validation CER



Validation WER



Validation set used: ‘dev-clean’.

My model’s final results:

Set / Metric	CER	WER
DEV-CLEAN	7.39	16.29
DEV-OTHER	15.06	28.95
TEST-CLEAN	7.06	15.75
TEST-OTHER	17.02	32.25

The model was trained on a single Nvidia RTX 3090 for ~550k steps (1.5 days). Although the model actually performs well, comparing it to the current SOTA system, which performance is presented in the next page, the results are much worse.

SOTA: w2v-BERT XXL

Set / Metric	CER	WER
DEV-CLEAN	-	-
DEV-OTHER	-	-
TEST-CLEAN	-	1.4
TEST-OTHER	-	2.5

Further Research

Based on the graphs that I presented, I assume that by training the model more extensively, we can achieve much better results, as the WER and CER performance were in a descending trend.
Also, the checkpoint of Magenta's MT3 work wasn't in the ".msgpack" format that is regularly used in the Flax framework, but it was given as a directory, so I couldn't find how to use it.
Future research may consider starting from that massively pre-trained checkpoint, which can encode audio much better than the original T5 checkpoint, and train the model for much more steps.

Conclusions

Using an off-the-shelf NLP seq2seq model, I showed how to build an ASR system. Although the system may be applicable as a 'toy' one, the results are worse than those of the SOTA, but probably can be improved a lot using more compute power or an available pre-trained checkpoint of the MT3 model.