Automatic Speech Recognition:

Connectionist Temporal Classification With Prefix Beam Search Decoding via Language Models

Benjamin Geyer

bgeyer3@masonlive.gmu.edu
G00997436

Volgenau School of Engineering
George Mason university
November 15, 2020

I. INTRODUCTION

The medium in which most people are familiar with Artificial Intelligence is through a smart voice assistant such as Alexa, Siri, and Google Home. These devices use automatic speech recognition to parse and understand what someone is saying. The applications of speech recognition are endless and can improve many people's daily lives by augmenting and simplifying their interface with the internet and the world around them.

II. BACKGROUND

Automatic speech recognition (ASR) is a subfield of sequence-to-sequence (Seq2Seq) learning in which spoken audio is translated into text using various statistical and machine learning algorithms. Historically, this was done using a combination of Hidden Markov Models (HMM) and then a deep neural network (DNN) to transform an audio signal into probabilities of phonemes using acoustic, lexical, and language models. There are many issues with this multi-part style approach including slower development due to first having to build the earlier stages (HMM) before being able to train the later stages (DNN). Additionally, language models are not easily transferable between languages preventing the model from generalizing well.

ASR using an end-to-end model allows the model to learn encoding, decoding, and language model details during training. This requires more data to train on of course, but that is becoming less and less of an issue as larger datasets continue to be generated. End-to-end ASR has multiple general steps including encoding the input audio into some latent-space vector representation using Fourier Transforms and a learned encoding, a middle "hyper-layer" consisting of connections between slices/frames of the input to build non-linear relationships, and then a decoder to transform the latent-space outputs into phoneme, word, or even sentence predictions with corresponding confidence levels.

III. THE PROBLEM

The input does not have a 1 to 1 mapping from input to labeled output. For any given input, the model needs some assigned output to compare the model's output against to check. However, the model might not know the correct word or syllable at any given time. To combat this, you can decide to assign a continuous softmax probability assigned to each possible output (character / phoneme) at every slice of input so that the model can continuously build posterior probabilities for future outputs based on prior probabilities of outputs through recurrence. The result of this is a string of repeated outputs in a row corresponding to a single character / phoneme. Decoding this string of predicted outputs is the subject of Connectionist Temporal Classification (CTC) first introduced by Graves et. al in 2006 [1]. CTC iterates over each output combination for each successive input and builds a probability distribution over each reduced output string based on the number of possible paths to generate a given string based on the output probabilities from the network.

IV. METHODOLOGY

Greedy or Best Search Decoding is the simplest decoding algorithm because it just chooses the most likely character at each timestep. This then results in an output sequence much longer than the transcript because it has a length of $sampling\,frequency * input\,length$. However, it is common for Greedy and Best Search Decoders to also eliminate all repeating characters to generate a much smaller output sequence. However, this can also introduce error by eliminating repeating characters in English words. An example of this would be the sequence 'fffffeeeeeellllll' decoding to 'fel' instead of 'feel' or 'fell' as was more likely the correct output.

Beam Search Decoding is a method of CTC Decoding where the most likely consecutive characters are iteratively concatenated into possible labelings/transcripts. However, unlike Greedy decoding, Beam Search has a beam-width k that stores some number k "beams" that are the most likely labelings/transcripts leading to the current timestep and their associated probabilities. As the timesteps progress, new potential beams are created extending existing beams and the k most likely are culled after each one has been evaluated.

Prefix Beam Search Decoding builds on this by applying an extra factor where the probability of each prefix/possible word is taken into account when new characters are added that aren't spaces [9]. This means that a learned Language Model is used to output a probability of a possible prefix being in the text. This LM-factor is applied with a weight of alpha, a hyperparameter, ranging from 0 to 1 as the LM-factor is raised to the power of alpha.

V. PREFIX BEAM SEARCH

The algorithm I have implemented is Prefix Beam Search Decoding using a learned Language Model (LM) to punish predicted prefixes/words that are not common in the training corpus. This algorithm was originally proposed by Graves et al. in 2006 but the exact algorithm steps I followed are detailed more clearly in Stanford researchers Hannun et al.'s 2014 paper *First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs* which aims to build a large learned vocabulary using the transcriptions of the training audio [1][2][14]. The paper demonstrates some state-of-the-art results in terms of Character and Word Error Rates (%CER, %WER) in the range of 5-35% which means that for a given target sequence, characters were correct up to 95% of the time for certain models shown in Table 1.

TABLE I
%CER AND %WER FOR EACH LANGUAGE MODEL

Model	%CER	%WER
No LM	10.0	35.8
Dictionary LM	8.5	24.4
Bigram LM	5.7	14.1

Prefix Beam Search is an algorithm that combines with Connectionist Temporal Classification (CTC) to predict a

sequence of outputs from a sequence of inputs that don't have a corresponding output alignment. What this means is that in ASR and OCR, for any transcription, there is no alignment for where in the input each output character/word is located in the audio or image. This presents a challenge during training because for a predicted network output at a timestep/frame in the sound input, there is no corresponding ground truth expected output. This also applies in OCR because rather than having each pixel map to an output character, there is a known transcription for a given handwriting sample image. CTC bridges the gap between a "continuous" sequence such as an audio file/image and a fixed-length output sequence by finding the probability of each time-alignment of input audio to output characters. One key detail about CTC is that it is applied differently during training than afterwards during the inference stage.

During the training of the model, network weights must be updated according to a loss function, which in this case is CTC. CTC loss is calculated by taking the sum of negative log-probabilities of all possible alignments of the transcribed text [2]. The more confident the model is on correct/close alignment predictions, the better. CTC Decoding comes into effect once training is complete and the model needs to transform the table of probabilities from the network into the most likely sequence of characters resulting in a predicted transcript of the input file.

VI. DATASET

Two of the most common applications of Connectionist Temporal Classification (CTC) Beam Search are Automatic Speech Recognition (ASR) and Optical Character Recognition (OCR). Hannun et al. focus on ASR but use the 81 hour Wall Street Journal (WSJ) news article dictation corpus which costs 2500 to purchase access [2]. Graves et al. 2006 used the TIMIT Acoustic-Phonetic Continuous Speech Corpus which is also 250 to access but has phonemes as the basis of the transcriptions rather than characters. Due to these limitations, I had to turn to the LibriSpeech 1000-hour transcribed English spoken audiobook corpus with subsections of different size and clarity (train-clean-100 ... train-other-500) [12]. I used a subset of the train-clean-100 dataset rather than including data from the "other" category for training to reduce the effect of badly labeled data on the comparison of decoding algorithms. Having better transcriptions to train on helps the language model have a larger improvement by truly learning the language and mapping audio to text.

The dataset contains '.flac' files containing short sentences spoken by people from various different audio books. For each folder of audio files, there is also a '.txt' file containing the manually-transcribed text being spoken and the audio filename for each file in that directory. There is also more information such as books, chapters, speaker ID's but the only data I considered for my implementation was simple the audio and text transcriptions. Each '.flac' file has a sampling rate of 44.1kHz and the longest file is almost 30 seconds long with many being under 5 seconds.

VII. DATA PREPROCESSING

The first step I did was to split up the transcription files into individual audio, text pairs with the same filenames (without the extension). Then, for each audio file, I take the Short Time Fourier Transform with a moving window to sample the audio and transform it into its frequency wave.

Then, each character of a transcription is vectorized according to an alphabet consisting of the lowercase ascii character as well as a few characters such as space, end of sequence, and unknown. These aid the algorithm in determining probabilities of sequences based on the language model probabilities.

The waveform spectrogram and vectorized transcript are fed to a neural network and passed through a series of layers augmenting and transforming them into a probable sequence of characters and their probabilities.

VIII. EXPERIMENTS

A. Comparison of Decoding Algorithms

To evaluate the performance of my implementation, I first trained a relatively simple neural network consisting of a Conv-1D embedding layer, a Bidirectional-LSTM layer, and a Dense layer which outputs the log-likelihood of each character output. This output is then saved and passed into a decoder which outputs the predicted character string representing the spoken words in the audio file. To compare my implementation to other state-of-the-art implementations of Beam Search, I passed the network output activations to multiple other beam search decoders including the built-in Tensorflow CTC Beam Search Decoder and a Beam Search Decoder by Harald Scheild [7] [10]. The network was trained for up to 200 epochs but was found to be heavily overfitting after only 75, so for testing, only the output from the trained model at 50 and 75 epochs was used.

B. Language Model

To try and improve my algorithm's performance, I also included a pre-learned Language Model (LM) to increase the difference between scores that were valid English words and those that weren't. I obtained this language model from the same Librispeech dataset source (http://www.openslr.org/11). These language models consist of log-likelihoods of many thousands of 1, 2, 3, and even 4-gram word sequences that occur in the Librispeech dataset. I used the 3-gram language model pruned to 3e-7 which means that any probability for an n-gram below 3e-7 was not included. This can cause errors known as out-of-vocabularies (OOV) of probability 0 which is why the LM output is used as a weighted combination with the neural network output weighted with a hyperparameter.

C. Pruning

One difference in my algorithm is from a proposed change by Moritz et al. which explains that for any timestep, the majority of characters have a quite low predicted probability and hence, shouldn't be considered for a new beam [8]. Only considering new characters to add that are above a certain prune threshold such as 0.001 allows for the model to only

iterate through characters that the network thinks have at least some chance of being correct for a given timestep.

D. Deep Speech 2

I used modified an online library (Automatic-Speech-Recognition) to allow me to test my implementation on the Mozilla DeepSpeech2 model that is much larger and complex with Convolutional, BatchNorm, Bi-LSTM, Dropout, and TimeDistributedDense layers. This model has nearly 60 Million trainable parameters and took quite a long time to train even on my GTX-1070 graphics card. To do this, I modified the library's pipeline to allow for CTC decoding after the neural network output as well as adding the integration of a language model.

E. End to End Data Processing Pipeline

I also built the entire dataprocessing and learning pipeline myself under end_to_end/ to control every aspect of the algorithm myself. However, I had some issues with my model architecture at the end so I can't fully test everything using it. However, it takes the fresh unzipped dataset from openASL.org and preprocesses it entirely and attempts to train the neural network and evaluate the prefix beam search on it.

F. Mel Scale Spectrograms

One last experiment I tried to perform was comparing regular Short Time Fourier Transforms to the Mel Scale which scales audio based on pitch so that it is more equal compared to how humans differentiate pitches. The difference between high-pitched sound is mostly indistinguishable to humans and as a result, most of the space on a regular Spectrogram is useless for humans. This means that theoretically, the sounds we use to communicate should be more uniformly distributed on a Mel Spectrogram compared to a regular Spectrogram. However, I didn't get this working properly.

IX. RESULTS

I was able to develop the prefix beam search decoding algorithm to work with and without a language model but I was unable to fully reproduce the results of the various papers in terms of average CER and WER over an entire dataset or using a Language Model due to issues with preprocessing and generating the Language Model. However, I was able to calculate the %CER for a single sample transcript as you can see in Table 2.

TABLE II
%CER FOR EACH DECODING ALGORITHM

Algorithm	50 Epoch %CER	75 Epoch %CER
Prefix Beam Search	29.2	0.0
Tensorflow Beam Search	37.1	2.2
Harald Scheidl Beam Search	38.2	2.2
Greedy Decoding	66.3	49.6

Every beam search decoding algorithm had roughly similar results on the simple neural network with my Prefix Beam Search doing the best with 29.2% CER after 50 epochs and 0% CER after 75 epochs. This is where the network was starting to overfit and training was stopped. Comparing my prefix beam search decoder to Tensorflow's and Harald Scheidl's, you can see that Prefix Beam Search performs better than standard Beam Search even with the best optimizations done by Tensorflow.

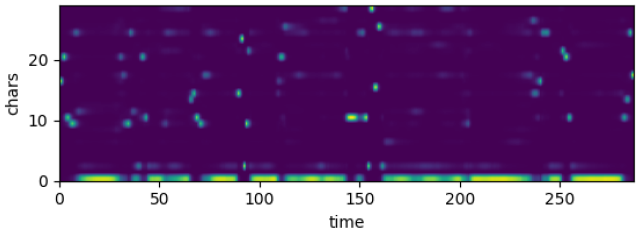


Fig. 1. Confidence scores for each character for each timestep given from the neural network after training for 50 epochs.

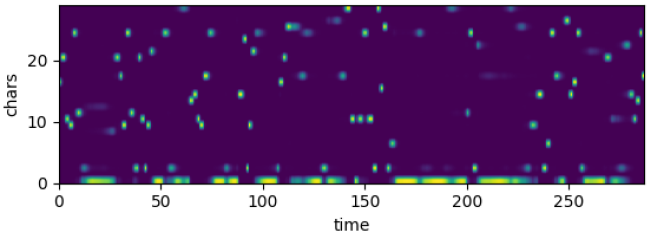


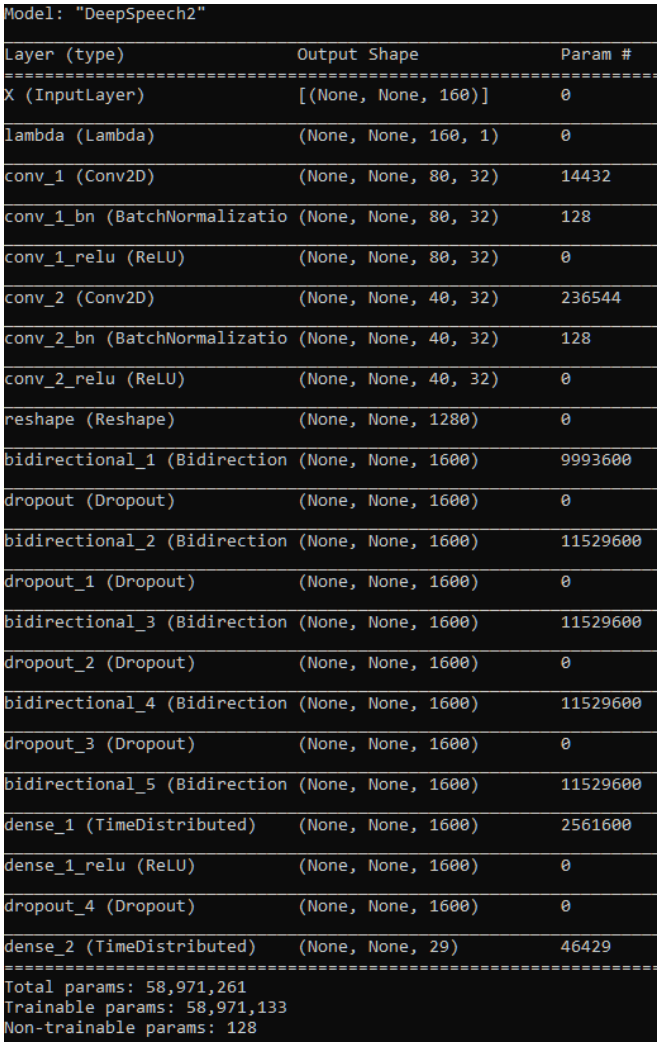
Fig. 2. Confidence scores for each character for each timestep given from the neural network after training for 75 epochs.

Figure 1. and 2. show the input matrix given to the decoding algorithms after 0 and 75 epochs. Each bright pixel corresponds to a highly predicted character for a given time. The row of bight pixels along the bottom corresponds to the blank/unknown character usually between two letters or words. Notice that after 75 epochs, each character prediction has a much higher confidence and is brighter than after only 50 epochs. Each decoder algorithm is able to achieve a much lower error rate when the character predictions are much more likely (brighter) and there is less noise within words.

Using Automatic-Speech-Recognition and DeepSpeech2's model, after 250 epochs, I was able to achieve some rough results of 48.6% CER for my prefix algorithm with a language model, 54.8% CER without a language model, and their Greedy Search Decoder achieved 65.2% CER. This training took a very long time and I strongly believe that the CER's would have become much better than my other models as their losses were still decreasing as the network was still slowly learning.

TABLE III
%CER AND %WER FOR EACH DECODER ON DEEPSPEECH2

Model	%CER	%WER
Greedy	65.2	88.6
Prefix No LM	54.8	74.0
Prefix w/ LM	48.6	67.2



```
Model: "DeepSpeech2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
X (InputLayer)               [(None, None, 160)]       0
_________________________________________________________________
lambda (Lambda)              (None, None, 160, 1)      0
_________________________________________________________________
conv_1 (Conv2D)              (None, None, 80, 32)      14432
_________________________________________________________________
conv_1_bn (BatchNormalizatio (None, None, 80, 32)      128
_________________________________________________________________
conv_1_relu (ReLU)           (None, None, 80, 32)      0
_________________________________________________________________
conv_2 (Conv2D)              (None, None, 40, 32)      236544
_________________________________________________________________
conv_2_bn (BatchNormalizatio (None, None, 40, 32)      128
_________________________________________________________________
conv_2_relu (ReLU)           (None, None, 40, 32)      0
_________________________________________________________________
reshape (Reshape)            (None, None, 1280)        0
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 1600)        9993600
_________________________________________________________________
dropout (Dropout)            (None, None, 1600)        0
_________________________________________________________________
bidirectional_2 (Bidirection (None, None, 1600)        11529600
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 1600)        0
_________________________________________________________________
bidirectional_3 (Bidirection (None, None, 1600)        11529600
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 1600)        0
_________________________________________________________________
bidirectional_4 (Bidirection (None, None, 1600)        11529600
_________________________________________________________________
dropout_3 (Dropout)          (None, None, 1600)        0
_________________________________________________________________
bidirectional_5 (Bidirection (None, None, 1600)        11529600
_________________________________________________________________
dense_1 (TimeDistributed)    (None, None, 1600)        2561600
_________________________________________________________________
dense_1_relu (ReLU)          (None, None, 1600)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, None, 1600)        0
_________________________________________________________________
dense_2 (TimeDistributed)    (None, None, 29)          46429
=================================================================
Total params: 58,971,261
Trainable params: 58,971,133
Non-trainable params: 128
```

Fig. 3. Mozilla's Deep Speech 2 Architecture trained for 250 epochs.

This demonstrates that even though the network hadn't fully optimized yet as the smaller network did, the presence of a language model helped increase the accuracy of the predictions in terms of CER% and WER% which means the experiment was a success. I wanted to test this more with different parameters but this one run took many hours to complete and as a result, I did not repeat the experiment on a large scale with varying hyperparameters. In hindsight, I should've saved the model checkpoints at various points to resume training.

X. ISSUES

I had a number of issues working on this project, mostly dealing with preprocessing the data into a useful format. For example, I was creating 16,000 spectrograms for each second of audio of each audio file and passing it to the network. Even the smallest of the datasets meant for meaningful training (train-clean-100) was on the order of 10's of gigabytes when uncompressed. This meant that as I processed each input without saving, I would be wasting many hours reprocessing the same files over and over. To solve this, I eventually figured out to pickle the preprocessed files and save them all in a big '.pkl' file with a '.tsv' corresponding transcript file.

XI. ONLINE LIBRARY

To address some my issues, I looked online for libraries that assist in preprocessing audio files, generating language files from those audio files, as well as managing the trained model performance. At first, I found a library called Fairseq which is a sequence to sequence modelling library built by Facebook in PyTorch that has a lot of useful data preprocessing functionality [3, 6]. I tried it out and eventually, I got their pre-setup model to train on LibriSpeech train-clean-100 preprocessed data with a language model it generated using Google's sentencepiece library that was fed all of the transcripts of train-clean-100. However, when I tried to implement my prefix beam search decoder in their pipeline to make used of the stored preprocessed data and learned language mode, I ran into a slew of issues due to PyTorch having many unsupported features in Windows that were a little beyond my expertise to fix. However, I was able to save checkpoints of the Fairseq model achieving a validation loss of approximately 5.0% WER after training for 26 epochs on train-clean-100.

After working on Fairseq for a while, I found another library (Automatic-Speech-Recognition) which is a python library that bundles together multiple other frameworks such as DeepSpeech and Seq2Seq but uses Tensorflow/Keras rather than PyTorch. I found this much easier to use and work with based on my other model being Tensorflow and that I had no issues with Windows. I was able to get this library to work and achieve nearly 26% CER. The library has utilities for preprocessing the audio files, generating multiple types of LM's based on the corpus, and tracking/storing the model performance metrics as well as activations. This allows me to run the more easily train and run the model only once as opposed to every time I want to evaluate a decoder. However, I stopped working with this library because PyTorch was causing my many issues so I just wanted to move on with implementing it all myself with tensorflow.

XII. SOURCE CODE

Under Automatic-Speech-Recognition, I have code to test my algorithm on the DeepSpeech2 model using a '.arpa' language model stored in under bens_evaluation.py. This code has a simple single test audio file and transcription to "train" on for convenience. I added my decoder to their pipeline under decoding.py.

Under end_to_end, I have code which I wrote that preprocesses the librispeech dataset into '.wav' and '.txt' files and passes their spectrograms to a Deep Learning model under CustomModel.py. I spent a lot of time on this trying to get it to work end to end. I am 99% there but I am having an issue with my model architecture and getting tensorflow's CTC loss to work with the data. I originally was working on the entire dev-clean and test-clean datasets but they take quite a long time and space. Included I have some sample

'.wav' and '.txt' files I kept as development sets to quickly develop on.

Lastly, I have my original implementation comparing several decoding algorithms with the same exact small model outputs under compare_decoders with my prefix implementation under prefix_beam_search.py. I have a README file as well containing some information for running the code in this directory as well as using a conda environment to get some of the packages installed quickly if you want.

I used many libraries including mostly Tensorflow 2.3, numpy, pandas, librosa and pydub for audio decoding, editdistance (CER and WER calculations), matplotlib, and tqdm for progressbars during training. I planned to use Tensorboard for visualizations while training my end to end code but I never got the model to properly work for decoding. I also tried using PyTorch for FacebookAI's Fairseq but had issues so I stopped.

XIII. REFERENCES

[1] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in ICML, Pittsburgh, USA, 2006.

[2] A. Y. Hannun, A. L. Maas, D. Jurafsky and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns", arXiv preprint arXiv:1408.2873, 2014.

[3] C. Wang, Y. Tang, X. Ma, A. Wu, D. Okhonko, and J. Pino, "fairseq s2t: Fast speech-to-text modeling with fairseq," ArXiv, 2020.

[4] Garofolo, John S., et al. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1. Web Download. Philadelphia: Linguistic Data Consortium, 1993.

[5] H. Scheidl, S. Fiel and R. Sablatnig, "Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm," 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), Niagara Falls, NY, 2018, pp. 253-258, doi: 10.1109/ICFHR-2018.2018.00052.

[6] M.Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, M. Auli, "FAIRSEQ: A Fast, Extensible Toolkit for Sequence Modeling," Proceedings of NAACL-HLT 2019: Demonstrations, ArXiv, 2019.

[7] M. Abadi, A. Agarwal, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2020. Software available from tensorflow.org.

[8] N. Moritz, T. Hori and J. L. Roux, "Streaming End-to-End Speech Recognition with Joint CTC-Attention Based Models," 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), SG, Singapore, 2019, pp. 936-943, doi: 10.1109/ASRU46091.2019.9003920.

[9] R. Collobert, A. Hannun, and G. Synnaeve, "Word-level speech recognition with a letter to word encoder," arXiv preprint arXiv:1906.04323, 2019.

[10] https://github.com/githubharald/CTCDecoder

[11] Synnaeve, G., Xu, Q., Kahn, J., et al., "End-to-end ASR: from Supervised to Semi-Supervised Learning with Modern Architectures," 2019, arXiv:1911.08460

[12] V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, 2015, pp. 5206-5210, doi: 10.1109/ICASSP.2015.7178964.

[13] "LibriSpeech language models, vocabulary and G2P models," http://www.openslr.org/11/, accessed: 2020-11-15.

[14] Linguistic Data Consortium, and NIST Multimodal Information Group. CSR-II (WSJ1) Complete LDC94S13A. Web Download. Philadelphia: Linguistic Data Consortium, 1994.