LSTM: Speech to Text

Blake Prall, Kainoa Jim

A. Abstract

In this project, we present a Speech-to-Text model employing a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs). Trained on the "train-clean-100" subset of the LibriSpeech dataset, our model transforms audio waveforms into spectrograms using MelSpectrogram transformation. To enhance robustness, we apply data augmentation techniques such as frequency masking and time masking. The model's architecture captures both local frequency patterns and temporal dynamics, crucial for accurate speech recognition. We employ the AdamW optimizer and utilize the Connectionist Temporal Classification (CTC) loss function to train our model, addressing challenges associated with variable-length sequences. Evaluation metrics include Character Error Rate (CER) and Word Error Rate (WER). Despite challenges, our model showcases promising performance in automatic speech recognition.

B. Introduction

Automatic Speech Recognition (ASR) plays a pivotal role in various applications, from transcription services to voice-activated systems. Our project aims to develop an effective Speech-to-Text model by leveraging the power of deep learning techniques. Trained on the LibriSpeech dataset, our model incorporates both CNNs and LSTMs, harnessing their abilities to capture intricate patterns in frequency and time. The choice of the "train-clean-100" subset ensures a focus on clean speech, laying the groundwork for future extensions to handle real-world audio conditions.

Data preprocessing involves transforming audio waveforms into spectrograms, providing a visual representation highlighting frequency changes over time. Augmentation techniques such as frequency masking and time masking contribute to the model's robustness by simulating variations in real-world audio conditions during training.

The heart of our model lies in the combination of CNNs and LSTMs. CNN layers excel at feature extraction from spectrograms, identifying local patterns critical for speech understanding. Meanwhile, bidirectional LSTMs capture the temporal dynamics, processing data in both forward and reverse directions to capture context effectively. The LSTM's memory mechanism and gate operations allow it to handle long-range dependencies, making it ideal for sequential data like audio recordings.

The training process involves the AdamW optimizer, adapting learning rates individually for each parameter, and the CTC loss function, addressing the challenge of aligning variable-length sequences. We evaluate our model using metrics such as CER and WER, considering both in-sample and out-of-sample error rates. Challenges, lessons learned, and potential improvements are discussed in the subsequent sections, providing a comprehensive exploration of the complexities and possibilities in ASR.

C. Methods

C.1. Data

To train our model, we used the subset "train-clean-100" of the LibriSpeech dataset, a dataset comprising of approximately 100 hours of English speech derived from audiobooks, featuring a diverse set of speakers and accents. It is important to note that the dataset does not have non-target attributes like background noise or speaker demographics, focusing solely on clean speech. As a result, we would expect worse results when tested on audio containing such attributes.

This data underwent several pre-processing steps to convert it into a suitable format for our model. Firstly, the audio waveforms were transformed into spectrograms using MelSpectrogram transformation. This step converts audio into a visual representation that highlights frequency changes over time, rather than magnitude changes. We then performed some data augmentation tasks. We augmented our data in two ways.

The first way we augmented our data was through frequency masking, which involves intentionally setting certain frequency bands to zero in the spectrogram of an audio signal. This technique helps deep learning models for tasks like speech recognition become more robust by simulating variations in real-world audio conditions. By exposing the model to masked frequency ranges during training, it enhances its ability to generalize across different acoustic environments. The other method of data augmentation that we used was time masking.

Time Masking is a data augmentation technique for sequential data, like time-series or speech. It involves randomly masking consecutive time steps in a sequence, and replacing them with neutral values. This helps improve the model's robustness by encouraging it to generalize better to variations in the temporal structure of the data.

C.2. Model

Our model combined Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs). The CNN layers are used for feature extraction from the spectrograms. They help in identifying local patterns in frequency and time. The LSTM Layers capture the temporal dynamics in speech. Being bidirectional, they pro-

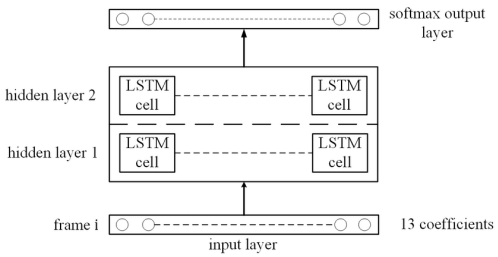

Figure 1. A visualization of the LSTM Neural Network

cess data in both forward and reverse directions, capturing context more effectively. LSTMs are specifically designed to capture long-range dependencies in data, making them ideal for sequential forms of data, such as audio recordings. These networks possess a memory mechanism that selectively retains and forgets information from past inputs, crucial for recognizing context and patterns in speech [3]. They adapt to the variability present in spoken language, including changes in speaking rate, accents, and background noise. LSTMs can also learn features from raw audio data, eliminating the need for manual feature engineering. Furthermore, they excel at capturing both low-level acoustic features and high-level linguistic information, resulting in state-of-the-art performance on Automatic Speech Recognition(ASR) benchmarks.

LSTMs operate by maintaining a crucial component known as the cell state, which acts as a long-term memory storage, enabling them to capture and remember information over extended time intervals. These networks employ three essential gates: the forget gate, input gate, and output gate 1. The forget gate decides what to retain or discard from the previous cell state, the input gate determines what new information to add, and the output gate controls which cell state information is used to generate the current output [2]. LSTMs process input sequences step by step, updating the cell state and producing outputs at each time step. Through a process called back-propagation through time, LSTMs are trained to optimize their weights and biases, allowing them to learn and adapt to specific data patterns, making them highly effective for tasks involving sequential data, such as natural language processing [4].

C.3. Hyperparameters

We experimented with different numbers of CNN and LSTM layers to find a balance between model complexity and performance. One issue we had trouble with was finding an optimal number of layers. We had limited computational power, so our model complexity couldn't be immense. However, we also had trouble getting our model to converge when settling for a simpler model, so we had to find a middle ground. For the learning rate, we started with a high learning rate of 0.1. We then adjusted it through testing and settled for 0.01 after testing. Any lower of a learning rate we ran into issues with time performance, so 0.01 was a compromise for both performance and time. For regularization, we used a dropout of 20 percent for both our CNN layers and LSTM layers, to prevent overfitting. Preventing overfitting also contributed to limitations of our model's complexity.

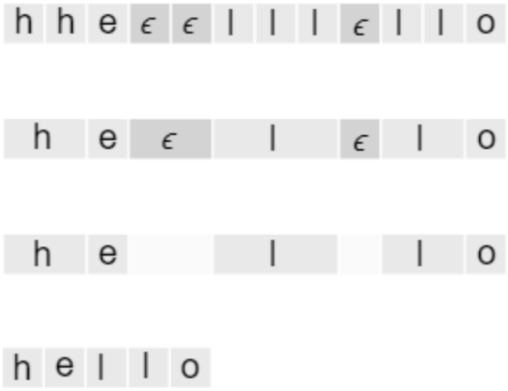

Figure 2. A visualization of how CTC works

C.4. Training

For training our ASR model, we utilized the AdamW optimizer and the Connectionist Temporal Classification (CTC) loss function.

We chose the AdamW optimizer for its effectiveness in training deep neural networks. AdamW is an extension of the popular Adam optimizer, but with the addition of weight decay, which helps prevent overfitting by penalizing large weights. This regularization term encourages the model to generalize better to unseen data. The AdamW optimizer adapts the learning rates individually for each parameter, making it well-suited for training complex models like ours with a combination of CNN and LSTM layers.

The CTC loss function is a key component in training ASR models. ASR deals with sequences of variable length, where aligning input audio frames directly to target transcriptions can be challenging due to different speaking rates. CTC addresses this problem by allowing the model to learn alignments between input frames and output symbols indirectly [1] (see Figure 2).

In our case, the CTC loss is applied to the output of the final classifier in the ASR model. During training, it measures the difference between the predicted sequence and the ground truth, considering all possible alignments. This allows the model to learn to emit the correct sequence of characters, regardless of the exact timing of each character within the input sequence.

The CTC loss function is defined as follows:

$$\mathcal{L}_{\text{CTC}}(\mathbf{X}, \mathbf{Y}) = -\log \sum_{u} \mathbf{F}[T, u] \tag{1}$$

where $\mathbf{X}$ is the input sequence, $\mathbf{Y}$ is the output sequence, and $\mathbf{F}$ is the forward probability matrix. The blank symbol is denoted as blank.

Overall, the combination of AdamW optimizer and CTC loss provides a robust framework for training our ASR model on the specified LibriSpeech dataset. Adjusting hyperparameters and monitoring training progress are essential steps in achieving optimal performance.

C.5. Evaluation

To evaluate the performance of our binary word classification model, we used an approach that considers both in-sample and out-of-sample error rates. In-sample error rate assesses the model's accuracy on the training data, gauging its ability to learn from the provided dataset. Out-of-sample error rate, on the other hand, measures the model's performance on unseen data, ensuring its generalization capabilities to new inputs. The two main evaluation metrics we used were Character Error Rate (CER) and Word Error Rate (WER).

Initially, we planned on only using WER, but found that doing so wouldn't show the extent of our model's success. For a word to be correct, every character in the word would have to be correct, so incorporating CER gave us a better idea of the model's performance. To do this, we used the Levenshtein distance, which is a metric that checks the minimum number of character edits needed to change one word into another. This was an essential metric for both evaluating the effectiveness of our model as well as training it.

We checked our results against existing speech recognition models to compare our model's efficiency against real-world applications. The dataset we used was already in a train/test split, which helped to test our model's out-of-sample error. It is difficult to establish a baseline for a model like this, as a simple model would randomly guess characters and words, leading to a high CER and WER. A random predictor would likely yield a CER higher than 95 percent and a WER close to 100 percent.

D. Results and Discussion

In our results, as illustrated in Fig. 3, the lines depicting WER and CER exhibit strikingly similar slopes throughout the training and testing phases. This observation signifies that the two error rates decrease at comparable rates, suggesting a consistent relationship between word-level and character-level accuracy improvements.

We achieved an average WER of 0.5102 and an average CER of 0.173798 on our test subset. Although our results fall short of benchmark values set by high-level automatic

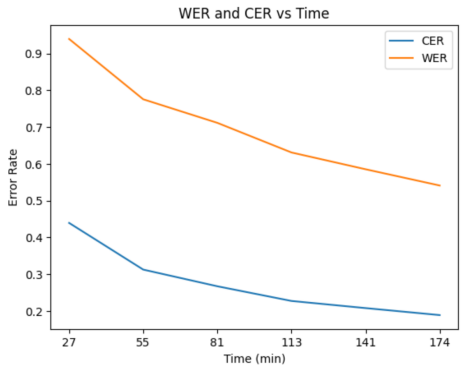

Figure 3. A graph showing our results of WER and CER throughout training and testing

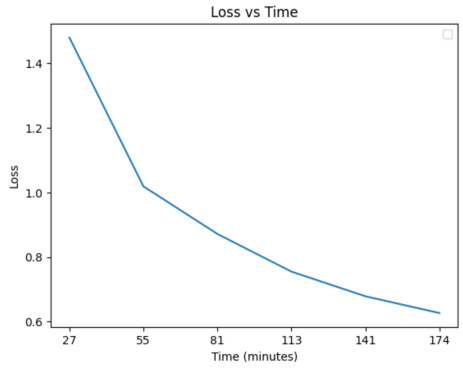

Figure 4. A graph showing loss throughout training

speech recognition models deployed by tech giants such as Amazon and Google, reaching Word Error Rates as low as 0.1, we acknowledge the computational constraints we faced. Our model's performance, considering these limitations, is satisfactory.

For a classification task like ASR, our model outperforms a baseline model such as random prediction, demonstrating its effectiveness in providing relatively accurate text transcriptions. The achieved average CER of 0.17 aligns with our primary objective. However, efficiency remains an area for improvement, as training the model required nearly 3 hours for 6 epochs on our dataset. Notably, reducing the model's complexity resulted in a significant performance dip.

The graphs in Fig. 4 depict the loss throughout training. Loss, WER, and CER were all still decreasing at the end of our training epochs, suggesting the potential for further performance improvement with additional training.

The trade-off between approximation and generalization, crucial in ASR tasks, was addressed through the combination of CNN and LSTM layers. While CNN layers excel at approximating local features from spectrograms, LSTMs

handle generalization by capturing long-term dependencies in time-series data. The number of layers in each part of the model played a significant role in this trade-off. Adjusting these parameters allowed us to strike a balance between model complexity, approximation, and generalization, contributing to the overall performance of our ASR model.

E. Conclusions

We had lofty goals when we created the proposal for this project. We wanted to create a script that could take an mp3 file as input and output accurate text transcriptions. We underestimated the computational power and time it would take to optimize our model based on the clean audio data set, so we did not develop our program to accept user inputted files. Furthermore, we trained our model on clean audio files with very limited noise, so testing on real-world sounds would result in high error rates. The factor that turned out to be most important in determining the results was the complexity of the model. Initially, with just an LSTM layer, we struggled to get our model to converge, and saw extremely high error rates. By increasing our model's complexity and combining the LSTM with a CNN improved our model's performance drastically. We are happy with how we approached this project, but we should have allotted more time to allow the model to train. There are also a few other things we could do to improve.

Some possible extensions for this project include expanding the training datasets to include more noise, but also possibly include different languages. We would also recommend developing a user interface for a practical deployment of our program. This project has been a valuable exercise in understanding and implementing speech recognition technology, but also in gaining experience in real-world applications of machine learning. While we achieved our primary goal of developing a functional speech-to-text model, this project revealed the complexities and challenges in this field, and the possibilities to go beyond what we accomplished. The experience gained provides a solid foundation for further exploration and innovation in speech recognition technology as well as future projects in machine learning.

References

[1] Awni Hannun. Sequence modeling with ctc. *Distill*, 2017. https://distill.pub/2017/ctc. 2

[2] Takaaki Hori, Jaejin Cho, and Shinji Watanabe. End-to-end speech recognition with word-based rnn language models, 08 2018. 2

[3] Kanishka Rao, Hasim Sak, and Rohit Prabhavalkar. Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer. pages 193–199, 12 2017. 2

[4] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. Fast and accurate recurrent neural network acoustic models for speech recognition. pages 1468–1472, 09 2015. 2