Connectionist Temporal Classification for Automatic Speech Recognition

Chibuzor John Amadi[502623]

[1] University of Pavia, Pavia
[2] University of Milano-Biccoca, Milan
[3] University of Milano-Statale, Milan

Abstract. Exploring classification and transformation of audio data typically involves the simple architecture of an encoder-only transformer. The input sequence is an audio waveform, and the encoder processes this waveform into hidden states; however, there is an alignment problem. Alignment here refers to a correspondence between the input audio data and the resulting characters in the output text. In this project, we will experiment with the use of *Connectionist Temporal Classification* (CTC) to deal with possible alignment problems on an *Automatic Speech Recognition* (ASR) task. Our architecture will take the input of audio data in raw waveform, which will be introduced first to a *Convolutional Neural Network* (CNN) for feature extraction and to produce hidden states. These hidden states will be passed to a transformer encoder that produces embeddings for each of these hidden states. Finally, the embeddings are passed to the CTC model, which carries out the final classification.

Keywords: Alignment · CNN · CTC · Automatic Speech Recognition

1 Introduction

Automatic Speech Recognition (ASR) [1] is a revolutionary technology in machine learning and deep learning that enables machines to understand and process human speech. In ASR, machines are able to process the audio signal, real human speech, and convert it into text. A very simple example of this technology in recent times is Siri by Apple and Alexa by Amazon.

Modern ASR systems leverage advanced techniques such as CNNs and CTC to significantly improve accuracy and efficiency. CNNs are pivotal in these systems as they downsample the input audio signal to produce a sequence of hidden states. For instance, in the Wave2Vec model [2], initial CNN layers process the input to produce a sequence of 50 hidden states, each representing about 25ms of audio with a slight overlap of 5ms between consecutive embeddings. This downsampling is crucial as it transforms the high-dimensional audio data into a more manageable form, capturing essential features while reducing computational complexity.

Following the CNN layers, the transformer encoder in Wave2Vec [2] receives these hidden states and encodes them into embeddings. These embeddings, typically of dimensionality 768, are then passed through a linear layer to map them into possible tokens of a small vocabulary, usually around 32 items or less. This process results in a tensor of shape (50, 32), where each token corresponds to approximately 20ms of audio, allowing the system to predict characters effectively.

However, predicting characters at such a fine temporal resolution inevitably leads to duplicate predictions for characters pronounced over more than 20ms. This is where the CTC algorithm [3] plays a crucial role. CTC introduces two special characters, the "blank" (_) and the space (|), to facilitate the alignment and classification of words. By allowing the model to output these special characters, CTC helps manage the variable length of spoken input and the corresponding text, ensuring accurate transcription despite the temporal variability of speech. In the next section, we will properly dissect the different models and the methods used to carry out this task.An evaluation of the model as well as an insight into the dataset will be in Section 3. Section 4 of this paper will be an analysis comparing the results of various experiments on variations of the model's execution.

2 Model Architecture

2.1 CONVOLUTIONAL NEURAL NETWORK

The model begins with the raw audio signal as input, typically a waveform representing human speech. This audio signal we can denote as X, is processed first by the Convolutional Neural Network (CNN) to extract meaningful features. The input X has a shape of **(N,1,L)**, where **N** is the batch size, 1 represents the number of channels and L is the length of the audio signal in samples. The CNN applies several convolutional layers, each using filters to slide over the input data and perform multiple computations. This operation will produce some feature maps **H**that capture essential features of the audio signal. Each convolutional layer is followed by a *ReLU* activation function, introducing non-linearity and helping the model learn complex patterns. Additionally, the convolutional layers use strides greater than one, which downsample the input sequence, reducing its length while preserving critical information. The result is a sequence of feature maps **H** with reduced temporal resolution, making the data more manageable for subsequent stages.

2.2 TRANSFORMER ENCODER

Following the CNN, the sequence of feature maps**H** undergoes a linear transformation to match the input dimensionality required by the transformer encoder. This transformation involves a linear layer that performs a weighted sum of the input features, projecting them into a higher-dimensional space represented by linear H $_{\text{linear}}$. The transformer encoder employs a self-attention mechanism [4] to process this sequence, allowing each position in the input to consider all other positions. This mechanism captures long-range dependencies and contextual relationships within the audio data. Mathematically, the self-attention operation computes the relevance of each input position to every other position, enhancing the model's ability to focus on important parts of the sequence. The transformer encoder consists of multiple layers, each comprising a self-attention mechanism and a feed-forward neural network. As the input sequence passes through these layers, it undergoes normalization and refinement, resulting in a highly informative representation transformer H $_{\text{transformer}}$.

2.3 CONNECTIONIST TEMPORAL CLASSIFICATION

The final stage of the ASR model involves mapping the encoded representations transformer H $_{\text{transformer}}$ to vocabulary tokens. This is achieved by passing the encoded features through a linear layer, producing logits Z that represent the un-normalized probabilities of each token at each time step. To ensure proper alignment between the predicted sequences and the target transcriptions, the model employs the Connectionist Temporal Classification (CTC) loss function. CTC is designed to handle variable-length sequences and introduces special blank tokens to allow for flexible alignment. By integrating CNN feature extraction, transformer encoding, and CTC alignment, the ctc (pre-trained) object model effectively handles variations in speech, delivering accurate and efficient ASR performance

3 Evaluation

The dataset used in evaluating this model was a 3-second audio recording gotten from an online tutorial on PyTorch's audio library[5]. The sound used was of what sounds like a man saying, "I had that curiosity beside me at that moment."

This audio is passed through the model as explained in the section above, and the CTC loss computes the probability of the correct transcription by summing over all possible alignments, enabling the model to learn the most likely mapping between the audio input and the text output. During inference, CTC decoding is used to generate the final transcription. This decoding process involves selecting the most likely sequence of tokens, often using beam search or greedy decoding, and collapsing repeated tokens while removing blanks to produce the final text.

3.1 CTC Decoding

Beam Search Algorithm The Beam Search algorithm is implemented using the *ctc_decoder*[6] from the *torchaudio*[7] library. This algorithm explores multiple possible sequences at each time step and keeps the most promising sequences (beams) based on their probabilities. The beam search[8] is initialized with parameters to control how many sequences we want and the number of sequences to consider at each time step. For each time step, the decoder expands each sequence in the beam by appending every possible token from the probability distribution. It calculates the probabilities of the new sequences by multiplying the sequence probability with the token probability at the current time step. After expanding the sequences, it keeps only the top N beams based on their probabilities. After processing all time steps, the decoder selects the sequence with the highest probability as the final output sequence.It removes consecutive duplicates and blanks to form the final transcription.

Greedy Algorithm The Greedy Algorithm[9] is a simpler and faster decoding method that selects the most probable token at each time step without considering alternative sequences. For each time step, it selects the token with the highest probability by finding the index of the maximum value in the probability distribution for that time step. It removes consecutive duplicate tokens and blanks from the sequence of selected tokens. It translates the remaining indices into characters using the provided labels to form the final transcription.

4 Results

Experiments details Experiments on the greedy algorithm and the beam search algorithm were carried out to improve its accuracy and computational speed. For the beam search algorithm, the beam width is a critical parameter in beam search. A larger beam width allows the algorithm to explore more sequences, which can lead to better accuracy but at the cost of increased computational complexity. The increase in the beam width in this project resulted in the same scores and accuracy which is expected considering it is a small dataset but it was 3 seconds slower. The greedy algorithm was the faster among both. Experiment on the greedy algorithm introduced dynamic thresholding. Dynamic thresholding[10] is a technique used to filter out low-probability tokens more effectively during the decoding process. With the greedy algorithm on CTC decoding, it helps to improve the accuracy of the predicted sequence by setting a minimum probability threshold. Only tokens with probabilities above this threshold are considered, reducing the influence of less likely predictions.

5 Conclusion

In this project, the use of Convolutional Neural Networks (CNNs), Transformer Encoders, and Connectionist Temporal Classification (CTC) for Automatic Speech Recognition (ASR) was explored. We experimented with both the Beam Search and Greedy decoding algorithms. Beam Search provided higher accuracy by exploring multiple sequences, but was computationally intensive. The Greedy Algorithm was faster, selecting the most probable token at each step, but potentially less accurate.

To improve the Greedy Algorithm, we introduced dynamic thresholding to filter out low-probability tokens, enhancing accuracy. The results demonstrated that while Beam Search achieved better accuracy, the Greedy Algorithm with dynamic thresholding offered a good balance between speed and performance, making it suitable for real-time applications.

6 References

1 Graves, A., Fernández, S., Gomez, F., Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks.

2 Baevski, A., Zhou, Y., Mohamed, A., Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. In Advances in Neural Information Processing Systems

3 Hannun, A. (2017). Sequence Modeling with CTC. Distill. https://distill.pub/2017/ctc/

4 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need.

5 PyTorch Audio Tutorial. (n.d.). Retrieved from https://pytorch.org/audio/stable/tutorials.html

6 Watanabe, S., Hori, T., Karita, S., Ogawa, A., Chen, Z., Hayashi, T., ... Watanabe, T. (2018). ESPnet: End-to-End Speech Processing Toolkit. Interspeech 2018.

7 Luo, Y., Chen, Z., Yu, S., Yoshioka, T. (2019). Deep Clustering and Conventional Networks for Music Source Separation: Stronger Together.

8 Graves, A., Mohamed, A. R., Hinton, G. (2013). Speech recognition with deep recurrent neural networks.

9 Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.

10 Povey, D., Khudanpur, S. (2005). Boosted MMI for model and feature-space discriminative training. In 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (Vol. 1, pp. I-97). IEEE.