Towards CTC-based Speech Recognition with Needle

Kelton Zhang
zhuoran2@andrew.cmu.edu

Qingzheng Wang
qingzhew@andrew.cmu.edu

November 5, 2024

1 Project Scope

Automatic Speech Recognition (ASR) is the technique of converting spoken language into text, enabling applications such as voice-controlled systems, transcription services, and accessibility tools. Nowadays, such systems are often built with training techniques like Connectionist Temporal Classification (CTC) [1], and encoders based on Transformer [2] and its variants like Conformer [3]. While most deep learning frameworks and toolkits support the techniques, this project aims to show how to implement them from scratch using low-level numerical operations based on our course's deep learning framework, Needle. Specifically, this project will implement key features for an ASR system, including the Transformer [2] encoder for high-level representation extraction, CTC [1] loss for alignment-free training, and beam search decoding for selecting the most probable time-synchronous output during inference. In addition, we will develop the necessary dataset processing and training procedures to facilitate effective training and validation of the ASR system using the Needle framework.

2 Proposed Design

We aim to expand Needle with ASR capabilities. Specifically, we plan to build upon homework 4 extra's transformer implementation and develop CTC loss for alignment-free training and beam search for decoding.

2.1 Transformer Encoder

Given audio input features, a Transformer [2] encoder is used to extract embedding features for subsequent training. The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$.

The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a (multi-head) self-attention mechanism, and the second is a simple, position-wise fully connected

feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization . That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{model} = 512$.

The attention mechanism maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. It can be implemented in a number of ways. We will start with the simplest dot-product attention where the input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, optionally divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values. The actual implementation will compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as: Attention(Q, K, V) = softmax($\frac{QK^T}{\sqrt{d_k}}$)V. Another more complex attention that could give better performance is multi-head attention where instead of performing a single attention function with d_{model} dimensional keys, values, and queries, linearly project the queries, keys, and values h times with different, learned linear projections to d_k, d_k, d_v dimensions, respectively. On each of these projected versions of queries, keys, and values we then perform the attention function in parallel, yielding d_v dimensional output values. These are concatenated and once again projected, resulting in the final values. The process can be formulated as MultiHead(Q, K, V) = Concat($head_1, ..., head_h$)W^O where $head_i$ = Attention(QW_i^Q, KW_i^K, VW_i^V) and the projections are parameter matrices $W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}$ and $W^O \in R^{hd_v \times d_{model}}$.

2.2 Training with CTC Loss

CTC loss is specifically designed for sequence-to-sequence tasks where the alignment between the input and output sequences is unknown. Given the input speech features $X = (x_0, \cdots, x_t, \cdots, x_{T-1}) \in \mathbb{R}^{D \times T}$, where D is the dimension of the speech features and t indexes the time steps of the input sequence with the length of the T, the target sequence token dictionary could be represented as $S = \{-, S_1, \cdots S_k, \cdots, S_{K-1}\}$ where $-$ represents blank and K is the size of the token dictionary including the blank. We first pass the input features through the encoder and obtain the probabilities of all tokens $Y = (y_0, \cdots, y_t, \cdots, y_{T-1}) \in \mathbb{R}^{K \times T}$. For each target sequence s, we construct a compressed graph

$$G(S) = \{-, S_0, -, \cdots, -, S_r, -, \cdots, -, S_{L-1}, -\}, \tag{1}$$

where $S_r \in S$ is the token that occurs in the target sequence with length L. Thus, the time-aligned token sequence is $s = \{s_0, \cdots, s_t, \cdots, s_{T-1}\}$, where $s_t \in G(S)$ is the token at time step t.

Then, we utilized the forward-backward algorithm to compute the probability $P(s_t = S_r, s|X)$ of the alignment between the target token and the input feature sequence on the compressed graph $G(S)$. The forward pass involves calculating the forward probability $\alpha(t, r) = P(s_t = S_r, s_0, \cdots, s_{t-1}|X)$,

which is calculated by the recursion

$$\alpha(t,r) = y_t^{S_r} \sum_{q:S_q \in prev(S_r)} \alpha(t-1,q), \tag{2}$$

where $prev(S_r)$ includes all the previous tokens with a path to the current token S_r and q indexes these previous tokens. Similarly, the backward pass computes the backward probability $\beta(t,r) = P(s_t = S_r, s_{t+1}, \cdots, s_{T-1}|X)$. The backward recursion is

$$\beta(t,r) = y_t^{S_r} \sum_{q:S_q \in succ(S_r)} \beta(t+1,q), \tag{3}$$

where $succ(S_r)$ includes all the successive tokens with a path to the current token S_r and q indexes these successive tokens. Then, we will compute the posterior possibility

$$P(s_t = S_r|s,X) = \frac{\alpha(t,r)\beta(t,r)}{y_t^{S_r} \sum_{r' \in K} \alpha(t,r')\beta(t,r')}, \tag{4}$$

which is typically represented as $\gamma(t,r)$.

Next, we will compute the KL divergence

$$Div = -\sum_t \sum_{S_r \in G(s)} P(s_t = S_r|s,X) \log Y(t, s_t = S_r) \tag{5}$$

as the training loss, *CTC loss*. For the backpropagation, the derivative of the divergence with respect to any particular output y_t^l could be computed by

$$\frac{dDiv}{dy_t^{S_k}} = -\frac{\sum_{r:S_r=S_k} \gamma(t,r)}{y_t^{S_k}}, \tag{6}$$

where r indexes the token S_r in the compressed graph $G(S)$ and k index the token S_k in the full token dictionary S.

In this project, we will implement these operations in Needle to facilitate effective alignment-free training for ASR.

2.3 Beam Search Decoding

At inference time, the Transformer encoder outputs will go through a linear layer with softmax to produce character probabilities for each timestep of the output. With these character probabilities, we will use beam search to generate the more probable sequence. Comparing to greedy search that chooses the most probable character one at a time, beam search aims to generate "best N" characters with beam width N being a hyperparameter. We will use a dynamic programming approach to implement beam search efficiently.

3 Timeline

Stage 1 (Nov 4 – Nov 14): Design and prototype
At this first stage, we will flesh out the design of the CTC-based ASR system with Needle and start prototyping. More concretely, we will work on implementing the Transformer [2] encoder starting from homework 4 extra and integrating it into our project codebase. We will also prepare for the required check-in on November 15. By that time, we should have a clear idea of how the different components work together.

Stage 2 (Nov 15 – Nov 30): Implement end-to-end ASR and test.
In this stage, we will push on building CTC loss and beam search decoding, testing the loose ends for the different components, and getting them to work together. With a functional ASR system, we will test its performance based on Librispeech ASR corpus [4]. If time allows, we will test with different models and train with more data to optimize the accuracy of ASR.

Stage 3 (Dec 1 – Dec 11): Present and report.
Finally, we will present a final video and report at the end of the semester to summarize the project's features and experimental outcomes.

References

[1] Fernández S. Gomez F. Schmidhuber J. Graves, A. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[2] A Vaswani. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[3] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[4] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.