Global Normalization for Streaming Speech Recognition in a Modular Framework

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce the Globally Normalized Autoregressive Transducer (GNAT) for addressing the label bias problem in streaming speech recognition. Our solution admits a tractable exact computation of the denominator for the sequence-level normalization. Through theoretical and empirical results, we demonstrate that by switching to a globally normalized model, the word error rate gap between streaming and non-streaming speech-recognition models can be greatly reduced (by more than 50% on the Librispeech dataset). This model is developed in a modular framework which encompasses all the common neural speech recognition models. The modularity of this framework enables controlled comparison of modelling choices and creation of new models.

1 Introduction

Deep neural network models have been tremendously successful in the field of automatic speech recognition (ASR). Several different models have been proposed over the years: cross-entropy (CE) models with a deep feed-forward architecture [19], connectionist temporal classification (CTC) models [14] with recurrent architectures such as long short-term memory (LSTM) [20], and more recently sequence-to-sequence (Seq2Seq) models like listen, attend and spell (LAS) [8], recurrent neural network transducer (RNN-T) [13], and hybrid autoregressive transducer (HAT) [40]. When configured in non-streaming mode, these neural ASR models have reached state-of-the-art word error rate (WER) on many tasks. However, the WER significantly drops when they are operating in streaming mode. In this paper, we argue that one main cause of such WER gap is that all the existing models are constrained to be locally normalized which makes them susceptible to *label bias* problem [36, 2, 24, 4]. To address this problem, we introduce new category of globally normalized models called *Globally Normalized Autoregressive Transducer* (GNAT). Our contributions are:

(1) **Addressing the label bias problem in streaming ASR through global normalization** that significantly closes more than 50% of the WER gap between streaming and non-streaming ASR.
(2) **Efficient, accelerator-friendly algorithms for the exact computation of the global normalization** under the finite context assumption.[1]
(3) **A modular framework for neural ASR** which encompasses all the common models (CE, CTC, LAS, RNN-T, HAT), allowing creation of new ones, and extension to their globally normalized counterparts.

[1]The implementation is included in the supplementary material of this paper.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

2 Streaming Speech Recognition

For an *input feature sequence* $\mathbf{x} = x_1 \dots x_T$, usually represented as a sequence of real valued feature vectors (such as log mel), and a finite output alphabet Σ, we wish to predict the corresponding *output label sequence* $\mathbf{y} = y_1 \dots y_U,\ y_i \in \Sigma$. We call each element x_i in $\mathbf{x}$ a *frame*, and each element y_i in $\mathbf{y}$ an *output label*. Common ASR models do not directly predict $\mathbf{y}$, but rather a *alignment label sequence* $\mathbf{z} = z_1 \dots z_V, z_i \in \Sigma \cup \Delta$. Δ is a finite alphabet of control labels, such as the blank label in CTC or RNN-T, or the end-of-sequence label in LAS. There is a deterministic mapping $y(\mathbf{z}) : (\Sigma \cup \Delta)^* \to \Sigma^*$ for obtaining $\mathbf{y}$ from $\mathbf{z}$ (e.g. in RNN-T, we simply remove all the blank labels from $\mathbf{z}$). An ASR model can then be broken down in two tasks: (1) assigning a score $\prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ to each alignment sequence $\mathbf{z}$, where $\omega(z_i|\mathbf{x}, \mathbf{z}_{<i}) > 0$ is the *alignment score* of predicting a single alignment label; (2) finding (usually approximately) $\arg\max_{\mathbf{y}} \sum_{\mathbf{z}|y(\mathbf{z})=\mathbf{y}} \prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$.

A non-streaming ASR model's alignment score $\omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ has access to the entire $\mathbf{x}$ for any i. In contrast, a streaming model's alignment score takes the form $\omega(z_i|\mathbf{x}_{<t(i)}, \mathbf{z}_{<i})$: it only has access a prefix $\mathbf{x}_{<t(i)}$ of the input feature sequence, where $t(i)$ is the frame to which z_i is *aligned*. Streaming models can be seen as special case of non-streaming models with respect to the alignment scores.

All the common neural ASR models use a locally normalized alignment score which satisfies the constraint $\sum_{z \in \Sigma \cup \Delta} \omega(z_i = z|\mathbf{x}, \mathbf{z}_{<i}) = 1$. This is achieved by applying the softmax function to the last layer activations. The local normalization constraint makes $\omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ easily interpretable as a conditional probability distribution $P_\omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$, and thus $\prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ easily interpretable as $P_\omega(\mathbf{z}|\mathbf{x})$. With samples from the true data distribution $P(\mathbf{x}, \mathbf{y})$, the modeling parameters are optimized by minimizing the negative log-conditional-likelihood loss $E_{P(\mathbf{x},\mathbf{y})}[-\log P_\omega(\mathbf{y}|\mathbf{x})] = E_{P(\mathbf{x},\mathbf{y})}[-\log \sum_{\mathbf{z}|y(\mathbf{z})=\mathbf{y}} P_\omega(\mathbf{z}|\mathbf{x})]$.

2.1 Label Bias in Streaming ASR

For a non-streaming, locally normalized model, the negative log-conditional-likelihood loss is minimized *in the limit* (assuming enough model capacity) by setting $\omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ to the true conditional probability $P(z_i|\mathbf{x}, \mathbf{z}_{<i})$, leading to $\prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ being equal to the true posterior $P(\mathbf{z}|\mathbf{x})$.

For a streaming model, $\mathbf{x}$ is replaced by $\mathbf{x}_{<t(i)}$ in the alignment score (e.g. by using a unidirectional encoder). Here the negative log-conditional-likelihood loss is minimized *in the limit* by setting $\omega(z_i|\mathbf{x}_{<t(i)}, \mathbf{z}_{<i})$ to $P(z_i|\mathbf{x}_{<t(i)}, \mathbf{z}_{<i})$. As a result, the product $\prod_i \omega(z_i|\mathbf{x}_{<t(i)}, \mathbf{z}_{<i}) = \prod_i P(z_i|\mathbf{x}_{<t(i)}, \mathbf{z}_{<i})$ is in general not equal to $P(\mathbf{z}|\mathbf{x})$ anymore. In other words, using a streaming locally normalized model means that the estimated alignment sequence posterior is the product of some locally normalized alignment scores which depend only on partial input $\mathbf{x}_{<t(i)}$, and as a result can no longer accurately represent the true conditional distribution. This will bias the model towards predictions with low-entropy estimated posterior probabilities at each decoding step. This degrades the model ability to revise previous decisions, a phenomenon called label bias [24, 2].

2.2 Global Normalization

Traditionally, *globally normalized models* such as conditional random fields [24] are used to address the label bias problem. This paper seeks to apply global normalization to modern neural architectures that are more similar to CTC, RNN-T, or LAS, rather than traditional linear models with the purpose of addressing label bias problem for streaming ASR models.

A globally normalized model does not constrain the alignment score $\omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ to be locally normalized; it only requires it to be any non-negative score, as long as the *denominator* $Z(\mathbf{x}) = \sum_{\mathbf{z}} \prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})$ is finite. The finite denominator allows us to interpret $\frac{\prod_i \omega(z_i|\mathbf{x}, \mathbf{z}_{<i})}{Z(\mathbf{x})}$ as a conditional probability distribution $P_\omega(\mathbf{z}|\mathbf{x})$. Minimum negative log-conditional-likelihood training can be more expensive for globally normalized models due to the need to compute $Z(\mathbf{x})$ and the corresponding gradients. However with our proposed modular framework, globally normalized model training can be made practical with careful modelling choices on modern hardware.

Any locally normalized model is by definition also a globally normalized, whose $Z(\mathbf{x}) = 1$. While the reverse is not true in general, Appendix E shows that under non-streaming settings with a mild condition, locally and globally normalized models actually express the same class of conditional

distributions $P_\omega(\mathbf{z}|\mathbf{x})$. Based on this observation, we argue that under non-streaming settings, with adequately powerful neural architectures, maximum log-conditional-likelihood training should yield behaviorly similar locally or globally normalized models, and thus similar WERs in testing. Results from [37, 17] and our own experiments in Section 6 validate this.

3 A Modular Framework for Neural ASR

In this section, we introduce a modular framework for neural ASR, using the weighted finite state automaton (WFSA) formalism to calculate the conditional probabilities via alignment scores. The modular framework clearly expresses the modelling choices enabling practical globlally normalized model training and inference. We use the WFSA formalism as the language for describing our framework because of its succinctness and precision. Our algorithms are implemented from scratch to run on TPUs, without using existing toolkits such as OpenFst [1], Kaldi [33] or K2 [21].

3.1 Preliminaries

We begin with an introduction to the relevant concepts and notations.

A *semiring* $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ consists of a set $\mathbb{K}$ together with an associative and commutative operation $\oplus$ and an associative operation $\otimes$, with respective identities $\bar{0}$ and $\bar{1}$, such that $\otimes$ distributes over $\oplus$, and $\bar{0} \otimes x = x \otimes \bar{0} = \bar{0}$. The *real* semiring $(\mathbb{R}_+, +, \times, 0, 1)$ is used when the weights represent probabilities. The *log* semiring $(\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$, isomorphic to the real semiring via the negative-log mapping, is often used in practice for numerical stability.[2] The *tropical* semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ is often used in shortest-path applications.

A *weighted finite-state automaton* (WFSA) $A = (\Sigma, Q, i, F, \rho, E)$ over a semiring $\mathbb{K}$ is specified by a finite alphabet Σ, a finite set of states Q, an initial state $i \in Q$, a set of final states $F \subseteq Q$, a final state weight assignment $\rho : F \to \mathbb{K}$, and a finite set of transitions $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ (ϵ denotes the empty label sequence). Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $o[e]$ its label, and $\omega[e]$ its weight. A *path* $\pi = e_1 \dots e_k$ is a sequence of consecutive transitions $e_i \in E$: $n[e_{i-1}] = p[e_i]$, $i = 2, \dots k$. The functions n, p, and ω on transitions can be extended to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$ and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $\omega[\pi] = \omega[e_1] \otimes \dots \otimes \omega[e_k]$). An *unweighted finite-state automaton* (FSA) $A = (\Sigma, Q, i, F, E)$ is simply a WFSA whose transitions and final states are all weighted by $\bar{1}$.

$\Pi(Q_1, Q_2)$ is the set of all paths from a subset $Q_1 \subseteq Q$ to a subset $Q_2 \subseteq Q$. $\Pi(Q_1, \mathbf{y}, Q_2)$ is the subset of all paths of $\Pi(Q_1, Q_2)$ with label sequence $\mathbf{y} = y_1 \dots y_U$, $y_i \in \Sigma$. A path in $\Pi(\{i\}, F)$ is said to be accepting or *successful*. The weight associated by A to any label sequence $\mathbf{y}$ is given by $A(\mathbf{y}) = \bigoplus_{\pi \in \Pi(\{i\}, \mathbf{y}, F)} \omega[\pi] \otimes \rho(n[\pi])$. The *weight* of A is the $\oplus$-sum of weights of all accepting paths $W(A) = \bigoplus_{\pi \in \Pi(\{i\}, F)} \omega[\pi] \otimes \rho(n[\pi])$. For a semiring $\mathbb{K}$ where $\otimes$ is also commutative, the *intersection* (or Hadamard product) of two WFSA A_1 and A_2 is defined as: $(A_1 \cap A_2)(\mathbf{y}) = A_1(\mathbf{y}) \otimes A_2(\mathbf{y})$. [28] gives an algorithm to compute the intersection. We can view $\mathbf{y}$ as a WFSA that accepts only $\mathbf{y}$ with weight $\bar{1}$, then $A(\mathbf{y}) = W(A \cap \mathbf{y})$.

3.2 Probabilistic Modeling and Inference on Acyclic Recognition Lattices

For any feature sequence $\mathbf{x} = x_1 \dots x_T$, we use a model with trainable parameters θ to induce a *recognition lattice* WFSA $A_{\theta,\mathbf{x}} = (\Sigma, Q_{\theta,\mathbf{x}}, i_{\theta,\mathbf{x}}, F_{\theta,\mathbf{x}}, \rho_{\theta,\mathbf{x}}, E_{\theta,\mathbf{x}})$. For a label sequence $\mathbf{y} = y_1 \dots y_U$, the weight associated with it, $A_{\theta,\mathbf{x}}(\mathbf{y})$, is interpreted under the log semiring as the unnormalized negative log conditional probability $P_\theta(\mathbf{y} \mid \mathbf{x}) = \frac{\exp(-A_{\theta,\mathbf{x}}(\mathbf{y}))}{\exp(-W(A_{\theta,\mathbf{x}}))} = \frac{\exp(-W(A_{\theta,\mathbf{x}} \cap \mathbf{y}))}{\exp(-W(A_{\theta,\mathbf{x}}))}$.

The recognition lattice $A_{\theta,\mathbf{x}}$ is designed to be acyclic, and therefore the weight of the automata in both the numerator and denominator above can be efficiently computed by visiting the states of the corresponding WFSA in topological order [27]. See Appendix C for our accelerator friendly version of this algorithm. We can thus train an ASR model by minimizing the negative log-conditional-likelihood on the training corpus $\mathcal{D}$, and choosing $\theta^\star = \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}}[-\log(P_\theta(\mathbf{y} \mid \mathbf{x}))] = \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}}[W(A_{\theta,\mathbf{x}} \cap \mathbf{y}) - W(A_{\theta,\mathbf{x}})]$.

[2] $a \oplus_{\log} b = -\log\left(e^{-a} + e^{-b}\right)$

In general, there can be more than one path in $A_{\theta,\mathbf{x}}$ that accepts the same $\mathbf{y}$. During inference, finding the optimal $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P_\theta(\mathbf{y}|\mathbf{x})$ requires running the potentially expensive WFSA disambiguation algorithm [30] on $A_{\theta,\mathbf{x}}$. As a cheaper approximation, we instead look for the shortest path $\hat{\pi}$ in $A_{\theta,\mathbf{x}}$ under the tropical semiring, and use the corresponding label sequence as the prediction, again using the standard shortest path algorithm for an acyclic WFSA [27].

3.3 Inducing the WFSA

Our framework decomposes the sequence prediction task in ASR into three components, each playing a specific role in inducing the recognition lattice $A_{\theta,\mathbf{x}}$.

- The *context dependency* FSA $C = (\Sigma, Q_C, i_C, F_C, E_C)$ is an ϵ-free, unweighted FSA, whose states encode the history of the label sequence produced so far. C is fixed for a given GNAT model, independent of input $\mathbf{x}$.
- The *alignment lattice* FSA $L_T = (\Sigma, Q_T, i_T, F_T, E_T)$ is an acyclic, unweighted FSA, whose states encode the alignment between input frames $\mathbf{x}$ and output labels $\mathbf{y}$. L_T depends on only the length T of input $\mathbf{x}$.
- The *weight* function $\omega_{\theta,\mathbf{x}} : Q_T \times Q_C \times (\Sigma \cup \epsilon) \to \mathbb{K}$. $\omega_{\theta,\mathbf{x}}$ is the only component that contains trainable parameters and requires full access to $\mathbf{x}$. This function defines the transition weights in the recognition lattice $A_{\theta,\mathbf{x}}$.

We will discuss how one can define these components in detail in the next section. With $(C, L_T, \omega_{\theta,\mathbf{x}})$ given, the recognition lattice $A_{\theta,\mathbf{x}}$ is defined as follows:

$$
\begin{aligned}
Q_{\theta,\mathbf{x}} &= Q_T \times Q_C \\
i_{\theta,\mathbf{x}} &= (i_T, i_C) \\
F_{\theta,\mathbf{x}} &= F_T \times F_C \\
E_{A_{\theta,\mathbf{x}}} &= \Big\{ \big((q_a, q_c), y, \omega_{\theta,\mathbf{x}}(q_a, q_c, y), (q'_a, q'_c)\big) \mid \\
&\qquad y \in \Sigma,\ (q_a, y, q'_a) \in E_T,\ (q_c, y, q'_c) \in E_C \Big\} \\
&\qquad \cup \Big\{ \big((q_a, q_c), \epsilon, \omega_{\theta,\mathbf{x}}(q_a, q_c, \epsilon), (q'_a, q_c)\big) \mid \\
&\qquad (q_a, \epsilon, q'_a) \in E_T,\ q_c \in Q_C \Big\} \\
\rho_{A_{\theta,\mathbf{x}}}(q) &= \bar{1},\ \forall q \in F_{\theta,\mathbf{x}}
\end{aligned}
$$

In other words, the topology (states and unweighted transitions) of the recognition lattice $A_{\theta,\mathbf{x}}$ is the same as the FSA intersection $L_T \cap C$; and the transition weights are defined using $\omega_{\theta,\mathbf{x}}$. The ϵ-freeness of C and the acyclicity of L_T implies that the recognition lattice $A_{\theta,\mathbf{x}}$ is also acyclic.

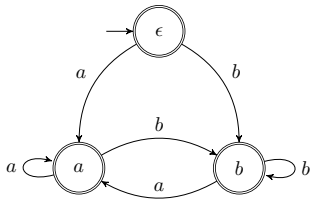
4 Components of a GNAT Model

In this section, we define *globally normalized autoregressive transducer* (GNAT) through the framework above, by specifying each model component.

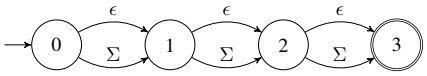
4.1 Context Dependency

For the experiments later in this paper, GNAT uses an n-gram context-dependency defined by $C_n = (\Sigma, Q_n, i_n, F_n, E_n)$, where $Q_n = \Sigma^{\leq n-1}$ corresponds to a label history of length up to $n-1$. The initial state $i_n = \epsilon$ is the empty label sequence. The transitions $E_n \subseteq Q_n \times \Sigma \times Q_n$ correspond to truncated concatenation: $E_n = \big\{(q, y, q') \mid q \in Q, y \in \Sigma)\big\}$ where q' is the suffix of qy with length at most $n-1$. For example, when $n = 3$, the transition (ab, c, bc) goes from state ab to state bc with label c. All states are final: $F_n = Q_n$. See Figure 1a for the FSA C_2 when $\Sigma = \{a, b\}$. The intersection $L_T \cap C_n$ is easy to compute thanks to the absence of ϵ-transitions in C_n. Appendix C demonstrates how C_n can be efficiently intersected during the shortest distance computation.

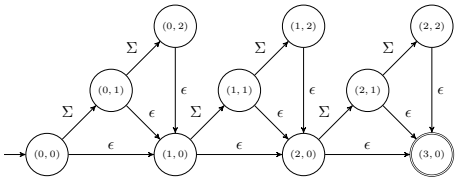
Although not studied in this paper, our modular framework makes it easy to switch to a more sophisticated context dependency, such as clustered histories often used for context-dependent phone models, or a variable context length as used in n-gram language models [29].



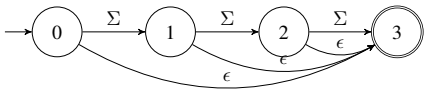
(a) n-gram context-dependency automaton C_2 for $\Sigma = \{a, b\}$.



(b) Frame dependent alignment lattice with $T = 3$.



(c) k-constrained label and frame dependent alignment lattice with $T = 3, k = 2$.



(d) Label dependent alignment lattice with $l(T) = 3$.

Figure 1: Examples of GNAT components

4.2 Alignment Lattices

Given the feature sequence length T, the alignment lattice FSA L_T defines all the possible alignments between the feature sequence and allowed label sequences. Since the feature sequence length T usually differs from the label sequence length U, many different alignments between the feature sequence and a label sequence can be defined. The states in an alignment lattice FSA encode how the next label or ϵ-transition corresponds to some position in the feature sequence.

We can choose different structures for L_T by encoding one or both of the positions in the feature sequence and the label sequence. A simple example is the *frame dependent alignment* similar to [14], where each frame is aligned to at most one label:

$$\begin{aligned} Q_T &= \{0, \dots, T\} \\ i_T &= 0 \\ F_T &= \{T\} \\ E_T &= \big\{(t-1, y, t) \mid y \in \Sigma \cup \{\epsilon\},\ 1 \le t \le T\big\} \end{aligned}$$

Here any state $t < T$ represents a position in the feature sequence. We start by aligning to the initial frame of the feature sequence, and repeatedly shift to the next frame for every subsequent label or ϵ-transition until all frames have been visited. Figure 1b depicts a frame dependent alignment lattice.

To allow a label sequence longer than the feature sequence, we can use the *k-constrained label and frame dependent alignment* similar to [13]:

$$\begin{aligned} Q_T &= \big\{(t, n) \mid 0 \le t \le T-1,\ 0 \le n \le k\big\} \cup \big\{(T, 0)\big\} \\ i_T &= (0, 0) \\ F_T &= \big\{(T, 0)\big\} \\ E_T &= \Big\{\big((t, n-1), y, (t, n)\big) \mid \\ &\qquad y \in \Sigma,\ 0 \le t \le T-1,\ 1 \le n \le k\Big\} \\ &\qquad \cup \Big\{\big((t-1, k), \epsilon, (t, 0)\big) \mid 1 \le t \le T\Big\} \end{aligned}$$

Here, up to k consecutive label transitions can align to any single frame. An ϵ-transition is then taken to explicitly shift the alignment to the next frame. The number of labels aligned to one frame is constrained by a constant k solely in order to impose acyclicity. Figure 1c depicts a k-constrained label and frame dependent alignment lattice.

Some models may only depend on the position in the label sequence, similar to [8]. In this case we can bound the length of the label sequence by some function $l(T)$, and use the following *label*

dependent alignment:

$$
\begin{aligned}
Q_T &= \{0, \ldots, l(T)\} \\
i_T &= 0 \\
F_T &= \{l(T)\} \\
E_T &= \{(u-1, y, u) \mid y \in \Sigma,\ 1 \leq u \leq l(T)\} \\
&\quad \cup \{(u, \epsilon, l(T)) \mid 0 \leq u \leq l(T) - 1\}
\end{aligned}
$$

Here, each label can be seen as aligning to the entire feature sequence, and the ϵ-transition serves as an explicit termination of the label sequence. Figure 1d depicts a label dependent alignment lattice.

4.3 Weight functions

The weight function $\omega_{\theta,\mathbf{x}}$ translates trainable parameters into transition weights for states in the recognition lattice $A_{\theta,\mathbf{x}}$. The choice of the weight function depends on the choice of the context and alignment lattice FSA, especially the alignment lattice where the meaning of a state directly affects how the weight function can access $\mathbf{x}$.

The experiments discussed in this paper use frame dependent and k-constrained label and frame dependent alignment lattices. In these two types of alignment lattices, a non-final state q_a in Q_T contains a position $\tau(q_a)$ in the feature sequence (the state itself in the case of frame dependent alignment lattices; the first value in the state in the case of label and frame dependent alignment lattices). Weight functions can thus be defined in three steps,

1. Feed $\mathbf{x}$ into an encoder, such as unidirectional or bidirectional RNN, or a self-attention encoder to obtain the sequence of hidden units $\mathbf{h}$ of dimension D. In the experiments we compared streaming vs non-streaming encoders.
2. Map a single frame of hidden units $\mathbf{h}[t]$ and context state q_c to a $(|\Sigma| + 1)$-dimensional vector, corresponding to the unnormalized transition weights for $y \in \Sigma \cup \{\epsilon\}$.
3. Optionally locally normalize transition weights across $y \in \Sigma \cup \{\epsilon\}$ given (q_a, q_c).

For step 2, we experiment with the following concrete modelling choices with varying degree of parameter sharing,

Per-state linear projection (*unshared*) For every context state q_c, we obtain a $D \times (|\Sigma| + 1)$ projection matrix W_{q_c} and a $(|\Sigma| + 1)$-dim bias vector b_{q_c} from θ, and define for $y \in \Sigma \cup \{\epsilon\}$:

$$\omega_{\theta,\mathbf{x}}(q_a, q_c, y) = \big(W_{q_c} \cdot \mathbf{h}[\tau(q_a)] + b_{q_c}\big)[y]$$

Shared linear projection with per-state embedding (*shared-emb*) We obtain from θ (a) for every context state q_c a D-dimensional state embedding E_{q_c}, (b) independent of context states a $D \times (|\Sigma|+1)$ projection matrix W and a $(|\Sigma| + 1)$-dim bias vector b , and define for $y \in \Sigma \cup \{\epsilon\}$:

$$\omega_{\theta,\mathbf{x}}(q_a, q_c, y) = \big(W \cdot \tanh(\mathbf{h}[\tau(q_a)] + E_{q_c}) + b\big)[y]$$

Shared linear projection with RNN state embedding (*shared-rnn*) Similar to *shared linear projection with per-state embedding* but E_{q_c} is obtained from running an RNN (e.g. LSTM) on the n-gram label sequence represented by q_c.

5 Discussion

A Modular Framework All the existing locally normalized models can be explained within the modular framework presented in Section 3 with a particular choice of context size, alignment lattice and of course with constraining the weights to be locally normalized using softmax function. Appendix B presents how CE, CTC, LAS, RNNT and HAT models can be expressed within this framework. This allows controlled comparison of different components as well as creating new models by mixing different modeling choices. Like locally and globally models in general, when the weights are locally normalized, the denominator of models defined in our framework is one.

Finally, our framework is very different from traditional uses of finite state machines via a cascade of weighted finite state transducer compositions. The separation of the weight function from the

automaton topology allows an arbitrarily complex, non-linear weight function to model the dependency among alignment states, context states, and output label, which is impossible with composition cascades. The idea of weight function has been proposed before (e.g. [38]), although as we shall see in experiments, the exact architecture of a weight function plays a crucial role in model accuracy.

Related Globally Normalized Models There is a rich literature on the applications of globally normalized models [5, 26, 35, 24], as well as theoretical studies on the label bias problem [24, 2, 12]. In ASR, there is a lot of research on applying globally normalized models [3, 7, 6, 16, 25, 45, 18]. Among these, MMI [3, 7] is the most relevant to our work. In MMI, a sequence level score is factorized into an acoustic model (AM) score and a language model (LM) score. The denominator is usually approximated over a pruned lattice. Recently [34] introduced lattice-free MMI, which replaces the word level LM with a 4-gram phone LM during training. Both the GNAT and MMI are globally normalized. The GNAT model differs from MMI in several ways: (1) Unlike MMI, GNAT does not require an external LM, nor does it impose any constraint on how the sequence level scores are defined. (2) In MMI training, the LM is kept frozen while the AM parameters are optimized. In GNAT all the model parameters are trained together. (3) GNAT's recognition lattice encodes the same set of weighted alignment paths during training and inference, therefore the denominator computation in GNAT is exactly matched between training and inference. This does not hold for the standard MMI, because by using different LMs between training and inference, the denominator computation in training time only serves as an approximation of the denominator during inference.

Implementation-wise, GNAT trains from scratch without any need for initialization or special regularization techniques as used in the lattice-free version of MMI [34]. In addition we were able to train GNAT models with accelerators without any techniques discussed in [34].

The concept of global normalization has also been visited with deep neural networks [17, 10, 42, 44]. These models can be seen as special cases of MMI based on aspects such as how the sequence level scores are factorized, their use of fixed LM for training, and the mismatching training and inference LMs. Thus the comparison between MMI and GNAT applies here as well. Their use of WFST composition cascades results in weaker modelling power compared to our weight function formulation, and their use of non-streaming encoders combined with their model formulation makes globally normalized models equally expressive to the locally normalized counterparts (Appendix E).

Challenges The main challenge with the GNAT model is its scalability to a larger number of label contexts. At each training step, the model requires $2 \times |Q_C| \times |\Sigma|$ multiplication and summation. For n-gram context dependency, $|Q_c| = (|\Sigma|^{n+1} - 1)/(|\Sigma| - 1)$, thus the computation scale exponentially by value of n. However as shown in Appendix D, due to the particular structure of this space the practical computation and memory cost benchmarks do not scale exponentially with n. We also note that large value of n might also not be necessary: The HAT model [40] reports that a Seq2Seq model with a label history of just the two previous phonemes performs on par with a similar model with a full history trained on very large voice-search corpus. Similar observations are reported in studies with grapheme and wordpiece units [43, 11]. Due to the data sparsity there might not be enough training to fully represent a n-gram space, so increasing the value of n might not necessarily lead to performance improvement. One way of dealing with large number of states is to use standard pruning techniques to keep only some of the most common states in the training data.

6 Experiments

Data We use the full 960-hour Librispeech corpus [32] for experiments. The input features are 80-dim. log Mel extracted from a 25 ms window of the speech signal with a 10 ms shift. The SpecAugment library with baseline recipe parameters were used [15]. The ground truth transcription is used without any processing and tokenized by the 28 graphemes that appear in the training data.

Architecture Attention-based architectures allows using the same parameterization for streaming and non-streaming models, thus for all the experiments we used 12-layer Conformer encoders [15] with model dimension 512, followed by a linear layer with output dimension 640. The Conformer parameters are set such that the only difference between streaming and non-streaming models is the right context: at each time frame t, the streaming models only access the left context (feature frames from 1 to t), while the non-streaming models can see the entire acoustic feature sequence. To enforce the consistency of the encoder architecture between streaming and non-streaming modes, we removed all the sub-architecture which behaved differently between these two modes. Specifically, we

context dep.	alignment lattice	weight fn streaming	WER [%] clean	WER [%] other
0-gram	frame	no	4.0	10.0
		yes	7.1	16.0
	label frame	no	6.7	10.2
		yes	8.8	14.5
1-gram	frame	no	2.8	6.0
		yes	4.9	10.0
	label frame	no	2.5	5.6
		yes	5.1	10.3
2-gram	frame	no	2.5	5.3
		yes	4.9	9.7
	label frame	no	2.5	5.3
		yes	5.0	9.8
RNN-T	frame	no	2.5	5.3
		yes	5.1	9.8
	label frame	no	2.5	5.5
		yes	5.0	9.8

(a) Locally normalized baselines with different context-dependency and alignment lattice. All models used a shared-rnn weight function with or without a streaming encoder. (sum-path decoding)

context dep.	weight function streaming	weight function normalization	WER [%] clean	WER [%] other
1-gram	no	local	3.4	8.7
		global	3.3	8.4
	yes	local	7.0	17.4
		global	5.5	14.0
2-gram	no	local	2.8	6.7
		global	2.8	6.7
	yes	local	4.9	11.0
		global	3.8	9.5

(b) Weight function parameters: normalization and streaming. (max-path decoding)

weight function type	weight function normalization	WER [%] clean	WER [%] other
unshared	local	4.9	10.7
	global	4.2	10.6
shared-emb	local	5.4	13.1
	global	4.1	9.9
shared-rnn	local	4.9	11.0
	global	3.8	9.5

(c) Comparison of different weight function types (max-path decoding).

Table 1: Experiment results

removed the convolution sub-sampling layer, and also forced the stacking layers to only stack within the left context. The baseline experiments use a shared-rnn weight function defined in section 4.3. A single layer LSTM is used with 640 cells. The experiments with the unshared weight function use a linear layer of size $(|Q_C| \times |\Sigma|) \times 640$ to project the encoder activation at each time frame into the transition weights of the recognition lattice. In our experiments, $|\Sigma| = 32$. For the n-gram context dependency, $|Q_c| = (|\Sigma|^{n+1} - 1)/(|\Sigma| - 1)$. The experiments with the shared-emb weight function use an embedding table of size $|Q_c| \times 128$.

Training All models are trained on 8×8 TPUs with a batch size 2048. The training examples with more than 1961 feature frames or more than 384 labels are filtered out. We used Adam optimizer [22] ($\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^-9$) with the Transformer learning rate schedule [41] (10k warm-up steps and peak learning rate $0.05/\sqrt{512}$). We applied the same regularization techniques and the training hyperparameters used in the baseline recipe of [15].

Evaluation We report WER results on standard Librispeech test sets: test_clean and test_other. The WER is either computed with *sum-path* algorithm or *max-path* algorithm. The sum-path algorithm merges the alignment hypothesis corresponding to the same label sequence prefix after removal of epsilons. In ideal decoding, sum-path should result in the most likely output label sequence. The max-path algorithm computes the highest scoring path using algorithms in Appendix C.

Baselines The RNN-T baselines are presented in the rows corresponding to the "*RNN-T*" context dependency in Table 1a. Unlike the other rows in Table 1a, the decoder RNN in RNN-T in principle encodes an unlimited output history as context. The standard RNN-T model effectively uses the label frame dependent alignment lattice with an infinite k. The rows corresponding to the frame dependent alignment lattice represent a slightly modified RNN-T model, where each frame is aligned to exactly one output (either a blank, or a lexical label). The WER difference between non-streaming RNN-T baselines in Table 1a and [15] are mainly due to our modifications to the Conformer encoder for a controlled comparison against streaming models.

Choice of the context dependency Table 1a compares effect of n-gram context dependency for $n = 0, 1, 2$ and baseline RNN-T models. The general observation is that increasing n leads to better performance quality independent of the other choices of the modeling parameters. However, the model with 2-gram context dependency already performs on par with RNN-T baseline. The 1-gram context dependency perform almost on par as baseline on clean test set while still lagging on the other test set. This is consistent with the earlier observations in [40, 43, 11].

Choice of the alignment lattice The comparison of different alignment lattices in Table 1a suggests that this choice does not significantly contribute to the model performance. While there is

a performance gap for 0-gram context dependency, we do not think there is a principal argument in favor of frame dependent alignment lattice. We speculate that this is more due to the choice of optimization parameters. However, the choice of lattice type can have some side effects. For example as k in k-constrained label frame dependent alignment lattice increases, the model has more ability to delay its prediction to the end of the signal. This implicit lookahead can translate into performance gains particularly for unidirectional models. By limiting this quantity to 1, we observed that performance on clean and other sets degrades by 34.9% and 36.7%, respectively.

Choice of the weight function normalization Table 1b examines the effect of weight normalization on non-streaming and streaming models. Here we present models with 1-gram and 2-gram context dependency with a frame dependent alignment lattice. Note that for 0-gram context dependency with a frame dependent alignment lattice it is easy to show that locally normalized and globally normalized models are equivalent. For non-streaming models, the normalization seems to not have an impact on the performance quality neither for 1-gram nor for 2-gram context dependency experiments. This is validates our argument in the end of Section 2.2 that locally and globally normalized models express the same class of conditional distributions under this particular setting. On the other hand, streaming models significantly benefit from global normalization: For clean test set, the globally normalized model outperforms the locally normalized model by about 21% relative WER for 1-gram context dependency and by about 20% relative gain for 2-gram context dependency.

The globally normalized model with 2-gram context dependency also beat the baseline streaming RNN-T model in Table 1a and performs significantly closer to the non-streaming RNN-T baseline. The equivalent streaming RNN-T model performs 5.1% on test clean and the non-streaming model performs 2.5% on same test set. The globally normalized model decoded with max-path algorithm performs 3.8% on same test set. The globally normalized model effectively closed almost 50% of the performance gap between streaming and non-streaming models.

The reported performance for the globally normalized models is from max-path decoding, while the baselines benefit from sum-path decoding. Comparing the locally normalized models' WER from max-path decoding in Table 1b and their counterparts in Table 1a, it is clear that sum-path decoding leads to an extra WER gain. This gain is more significant on test_other. So we expect the globally model performs even better when decoded with sum-path.

The standard sum-path algorithms use several heuristics particularly for path merging and pruning. While similar merging techniques can be applied to the globally normalized models, the pruning heuristics require several adjustments. This is particularly due to the nature of the globally normalized models where the transition weights are not constrained and can take any value, unlike locally normalized models where the transition weights are constrained to be positive number between 0 and 1 and sum to 1 for all the weights leaving the same state in recognition lattice. We plan to explore approaches to improve globally normalized model decoding in the future.

Choice of the weight function architecture Finally Table 1c compares different choices of the architectures for a streaming model with 2-gram context dependency and frame dependent alignment lattice. While the unshared and shared-rnn architectures are very different in terms of parameter sharing among states, both perform well, though the shared-rnn architecture performs slightly better. The shared-emb architecture performs significantly worst than shared-rnn architecture. Note that the shared-rnn model is able to learn common structures across states in the context dependency while shared-emb does not have such capability.

7 Conclusion

The GNAT model was proposed and evaluated with the focus on the label bias problem and its impact on the performance gap between streaming and non-streaming locally normalized ASR. The finite context property of this model allows exact computation of the sequence level normalization which makes this model differ from existing globally normalized models. Furthermore, the same property allows accelerator friendly training and inference. We showed that the streaming models with globally normalized criteria can significantly close the gap between streaming and non-streaming models by more than 50%. Finally, the modular framework introduced in this paper to explain the GNAT model encompasses all the common neural speech recognition models. This enables fair and accurate comparison of different models via controlled modelling choices and creation of new ASR models.

References

[1] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata*, pages 11–23. Springer, 2007.

[2] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[3] Lalit Bahl, Peter Brown, Peter De Souza, and Robert Mercer. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 49–52. IEEE, 1986.

[4] Léon Bottou. *Une approche théorique de l'apprentissage connexionniste et applications à la reconnaissance de la parole*. PhD thesis, Paris 11, 1991.

[5] Léon Bottou, Yoshua Bengio, and Yann Le Cun. Global training of document processing systems using graph transformer networks. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 489–494. IEEE, 1997.

[6] John S Bridle and L Dodd. An alphanet approach to optimising input transformations for continuous speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, pages 277–280. IEEE Computer Society, 1991.

[7] Peter F Brown. The acoustic-modeling problem in automatic speech recognition. Technical report, Carnegie-Mellon University, Pittsburgh, PA, Department of Computer Science, 1987.

[8] William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

[9] Zhiyi Chi. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160, 1999.

[10] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193*, 2016.

[11] Mohammadreza Ghodsi, Xiaofeng Liu, James Apfel, Rodrigo Cabrera, and Eugene Weinstein. Rnn-transducer with stateless prediction network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7049–7053. IEEE, 2020.

[12] Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. An empirical investigation of global and local normalization for recurrent neural sequence models using a continuous relaxation to beam search. *arXiv preprint arXiv:1904.06834*, 2019.

[13] Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

[14] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 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.

[15] 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.

[16] Asela Gunawardana, Milind Mahajan, Alex Acero, and John C Platt. Hidden conditional random fields for phone classification. In *Ninth European Conference on Speech Communication and Technology*. Citeseer, 2005.

[17] Awni Hannun, Vineel Pratap, Jacob Kahn, and Wei-Ning Hsu. Differentiable weighted finite-state transducers. *arXiv preprint arXiv:2010.01003*, 2020.

[18] Yasser Hifny and Steve Renals. Speech recognition using augmented conditional random fields. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(2):354–365, 2009.

[19] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] k2-fsa authors. k2-fsa.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[24] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[25] Martin Ian Layton. *Augmented statistical models for classifying sequence data*. PhD thesis, University of Cambridge, 2007.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[28] Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

[29] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In Jacob Benesty, M. Sondhi, and Yiteng Huang, editors, *Handbook of Speech Processing*, chapter 28, pages 559–582. Springer, 2008.

[30] Mehryar Mohri and Michael D Riley. On the disambiguation of weighted automata. In *International Conference on Implementation and Application of Automata*, pages 263–278. Springer, 2015.

[31] Nelson Morgan and Herve Bourlard. Continuous speech recognition using multilayer perceptrons with hidden markov models. In *International conference on acoustics, speech, and signal processing*, pages 413–416. IEEE, 1990.

[32] 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.

[33] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

[34] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech*, pages 2751–2755, 2016.

[35] Ronald Rosenfeld. A whole sentence maximum entropy language model. In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 230–237. IEEE, 1997.

[36] Noah A Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491, 2007.

[37] Hao Tang, Liang Lu, Lingpeng Kong, Kevin Gimpel, Karen Livescu, Chris Dyer, Noah A Smith, and Steve Renals. End-to-end neural segmental models for speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1254–1264, 2017.

[38] Hao Tang, Weiran Wang, Kevin Gimpel, and Karen Livescu. Discriminative segmental cascades for feature-rich phone recognition. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 561–568. IEEE, 2015.

[39] Ehsan Variani, Tom Bagby, Erik McDermott, and Michiel Bacchiani. End-to-end training of acoustic models for large vocabulary continuous speech recognition with tensorflow. 2017.

[40] Ehsan Variani, David Rybach, Cyril Allauzen, and Michael Riley. Hybrid autoregressive transducer (hat). In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6139–6143. IEEE, 2020.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[42] Hongyu Xiang and Zhijian Ou. Crf-based single-stage acoustic modeling with ctc topology. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5676–5680. IEEE, 2019.

[43] Qian Zhang, Han Lu, Hasim Sak, Anshuman Tripathi, Erik McDermott, Stephen Koo, and Shankar Kumar. Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7829–7833. IEEE, 2020.

[44] Huahuan Zheng, Wenjie Peng, Zhijian Ou, and Jinsong Zhang. Advancing ctc-crf based end-to-end speech recognition with wordpieces and conformers. *arXiv preprint arXiv:2107.03007*, 2021.

[45] Geoffrey Zweig and Patrick Nguyen. A segmental crf approach to large vocabulary continuous speech recognition. In *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 152–157. IEEE, 2009.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A An overview example

To illustrate different components of the GNAT model, here we present a toy example of designing a speech recognition for finite alphabet $\Sigma = \{a, b\}$. Given an input feature sequence $\mathbf{x} = x_1 \dots x_T$, we wish to predict the corresponding output label sequence $\mathbf{y} = y_1 \dots y_U,\ y_i \in \Sigma$. Our objective is to create a conditional probabilistic model $P(\mathbf{y}|\mathbf{x})$ which assigns the highest probability to the correct label sequences for any given feature sequence. We construct a GNAT model with the following components:

- context-dependency: 2-gram
- alignment-lattice: frame dependent
- weight function: per-state linear projection, streaming

Next we elaborate details of each of these modules and how they are integrated to create the final space the recognition lattice $A_{\theta,\mathbf{x}}$ and the probabilistic model $P(\mathbf{y}|\mathbf{x})$ as described in Section 3.

A.1 Context Dependency FSA

Figure 2 presents the 2-gram context dependency C_3. The set of states for this space are the initial state, 1-gram states and 2-gram states:

$$Q_C = \{i, a, b, aa, ab, ba, bb\}$$

With the lexicographic order, these states are indexed as follow:

state	state index
i	0
a	1
b	2
aa	3
ab	4
ba	5
bb	6

For this particular FSA, the transitions space is

$$E_C = \{(q, y, q') \mid q \in Q, y \in \Sigma)\}$$

where q' is the suffix of qy with length at most 2. All 14 transitions of this space are listed in the following table:

from state	label	to state
i	a	a
i	b	b
a	a	aa
a	b	ab
b	a	ba
b	b	bb
aa	a	aa
aa	b	ab
ab	a	ba
ab	b	bb
ba	a	aa
ba	b	bb
bb	a	ba
bb	b	bb

A.2 Alignment Lattice FSA

Figure 1b depicts a frame dependent alignment lattice L_4 for four frames feature sequence $\mathbf{x}$. The states of this space are:

$$Q_T = \{0, 1, 2, 3, 4\}$$

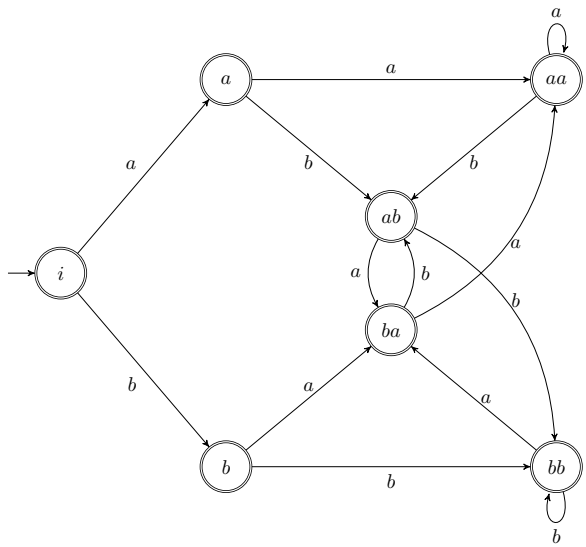


Figure 2: 2-gram context-dependency automaton C_3 for $\Sigma = \{a, b\}$.

where 0 is the initial state and 4 is the final state. Every path starting from the initial state in this automaton corresponds to one possible alignment sequence of the input feature sequence. The example FSA in Figure 1b encodes $3^4 = 81$ possible alignment sequences. An alignment path $\epsilon a \epsilon b$ corresponds to the following sequence of transitions in L_4:

$$(0, \epsilon, 1), (1, a, 2), (2, \epsilon, 3), (3, b, 4)$$

A.3 Weight Function

The weight function $\omega_{\theta,\mathbf{x}} : Q_T \times Q_C \times (\Sigma \cup \epsilon) \to \mathbb{K}$. $\omega_{\theta,\mathbf{x}}$ is the only trainable component of the GNAT model which assigns a weight to every transition of the recognition lattice $A_{\theta,\mathbf{x}}$. We first feed $\mathbf{x} = x_1, ..., x_4$ into an encoder to obtain hidden activations $\mathbf{h} = h_1, ..., h_4$ of dimension D. The encoder can be any neural architecture such as DNNs, CNNs, RNNs or Transformers. Since we are interested in streaming weight function for this example, we need to make sure the encoder is also streaming. This means h_t can only depends on $x_{1:t-1}$. Finally we define a $D \times 3$ matrix W_{q_c} and a 3-dim bias vector b_{q_c} for any $q_c \in Q_c$. The weight function is then defined as:

$$\omega_{\theta,x_{1:t}}(q_a = t, q_c, y) = W_{q_c}[y, :] \cdot h_t + b_{q_c}[y]$$

where $W_{q_c}[y, :]$ is the row of W_{q_c} corresponding to label $y \in \Sigma \cup \{\epsilon\}$. The total number of trainable parameters is $7 \times D \times 3 + 7 \times 3$ parameters plus the number of parameters of the encoder function.

A.4 The recognition lattice $A_{\theta,\mathbf{x}}$

Given the context dependency FSA C_3, the frame dependent alignment lattice L_4 and the weight function $\omega_{\theta,\mathbf{x}}$, we are ready to derive the recognition lattice $A_{\theta,\mathbf{x}}$. The state space has 5×7 states:

$$Q_{\theta,\mathbf{x}} = \big\{(t, q_c) \,|\, 0 \leq t \leq 4, q_c \in Q_c\big\}$$

The transitions in this space is specified by the state it is originated from, (t, q_c), the label $y \in \Sigma \cup \{\epsilon\}$, weight $\omega_{\theta,\mathbf{x}}(t, q_c, y)$ and the state the transition is ended to $(t + 1, q'_c)$. For the alignment sequence of our example, $\epsilon a \epsilon b$, the transitions are:

from	label	weight	to
$(0, i)$	ϵ	$\omega_{\theta,x_1}(q_a = 0, q_c = i, y = \epsilon) = W_0[0, :] \cdot h_1 + b_0[0]$	$(1, \epsilon)$
$(1, i)$	a	$\omega_{\theta,x_{1:2}}(q_a = 1, q_c = i, y = a) = W_0[1, :] \cdot h_2 + b_0[1]$	$(2, a)$
$(2, a)$	ϵ	$\omega_{\theta,x_{1:3}}(q_a = 2, q_c = a, y = \epsilon) = W_1[0, :] \cdot h_3 + b_1[0]$	$(3, a)$
$(3, a)$	b	$\omega_{\theta,x_{1:4}}(q_a = 3, q_c = a, y = b) = W_1[2, :] \cdot h_4 + b_1[2]$	$(4, ab)$

The product of the above weights is the score that the GNAT model assigns to the features sequence $\mathbf{x}$ and alignment sequence $\epsilon a \epsilon b$:

$$\text{score}(\mathbf{x}, \epsilon a \epsilon b) = \omega_{\theta,x_1}(0, i, \epsilon)\omega_{\theta,x_{1:2}}(1, i, a)\omega_{\theta,x_{1:3}}(2, a, \epsilon)\omega_{\theta,x_{1:4}}(3, a, b)$$

Note that for simplicity we use real semiring for all the score calculations in this example.

The GNAT model formulates the posterior probability $P(\mathbf{y}|\mathbf{x})$ by ratio of two quantities:

- numerator: sum of all the score$(\mathbf{x}, \tilde{\mathbf{y}})$ where $\tilde{\mathbf{y}}$ is an alignment between $\mathbf{x}$ and $\mathbf{y}$. For example if $\mathbf{y} = ab$, there are only 6 possible alignments: $ab\epsilon\epsilon$, $a\epsilon b\epsilon$, $a\epsilon\epsilon b$, $\epsilon ab\epsilon$, $\epsilon a\epsilon b$, $\epsilon\epsilon ab$
- denominator: sum of all the score$(\mathbf{x}, \tilde{\mathbf{y}})$ where $\tilde{\mathbf{y}}$ can be any $(\Sigma+1)^T = 3^4 = 81$ sequences.

Since the numerator computation is a special case of the denominator, we only present the denominator calculation. To follow the computation presented in Section C, we first define forward variable α_t which is a 7-dim real-valued vector where $\alpha_t[j]$ is the total alignment scores reaching to state index j (corresponding to the state indices in Q_c) at time t:

$$\alpha_t = \begin{array}{ccccccc} (t,i) & (t,a) & (t,b) & (t,aa), & (t,ab) & (t,ba) & (t,bb) \\ (\alpha_t[0] & \alpha_t[1] & \alpha_t[2] & \alpha_t[3] & \alpha_t[4] & \alpha_t[5] & \alpha_t[6]\,) \end{array}$$

the initial state α_0 is a 1-hot vector with $\alpha_0[j] = 1$ iff $j = 0$, the initial state. At every time frame t, the transition weight matrix $\Omega_{t,\Sigma}$ is defined for all the transitions in $E_{A_{\theta,\mathbf{x}}}$ where label is an element of Σ. This matrix is a structured matrix with only $|\Sigma| = 2$ non-zero elements per row:

$$\Omega_{t,\Sigma} = \begin{array}{c} \\ (t-1,i) \\ (t-1,a) \\ (t-1,b) \\ (t-1,aa) \\ (t-1,ab) \\ (t-1,ba) \\ (t-1,bb) \end{array} \begin{array}{c} \begin{array}{ccccccc} (t,i) & (t,a) & (t,b) & (t,aa), & (t,ab) & (t,ba) & (t,bb) \end{array} \\ \begin{pmatrix} 0 & \omega(t,i,a) & \omega(t,i,b) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \omega(t,a,a) & \omega(t,a,b) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega(t,b,a) & \omega(t,b,b) \\ 0 & 0 & 0 & \omega(t,aa,a) & \omega(t,aa,b) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega(t,ab,a) & \omega(t,ab,b) \\ 0 & 0 & 0 & \omega(t,ba,a) & \omega(t,ba,b) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega(t,bb,a) & \omega(t,bb,b) \end{pmatrix} \end{array}$$

Similarly we denote $\Omega_{t,\epsilon}$ to be the transition weight matrix for all the transitions in $E_{A_{\theta,\mathbf{x}}}$ where label is ϵ. This matrix is a diagonal matrix corresponding to the weights of the self loops:

$$\Omega_{t,\epsilon} = \begin{array}{c} \\ (t-1,i) \\ (t-1,a) \\ (t-1,b) \\ (t-1,aa) \\ (t-1,ab) \\ (t-1,ba) \\ (t-1,bb) \end{array} \begin{array}{c} \begin{array}{ccccccc} (t,i) & (t,a) & (t,b) & (t,aa), & (t,ab) & (t,ba) & (t,bb) \end{array} \\ \begin{pmatrix} \omega(t,i,\epsilon) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \omega(t,a,\epsilon) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \omega(t,b,\epsilon) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \omega(t,aa,\epsilon) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \omega(t,ab,\epsilon) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega(t,ba,\epsilon) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \omega(t,bb,\epsilon) \end{pmatrix} \end{array}$$

For our model, the forward variable α_t can be calculated given α_{t-1} and the above weights matrices as:

$$\alpha_t = \alpha'_{t-1}(\Omega_{t,\Sigma} + \Omega_{t,\epsilon})$$

since every transition at time $t+1$ is either an ϵ transition or a non-ϵ transition. Here α'_{t-1} is the transpose of forward variable α_{t-1}.

Given the above iterative equation, the forward variable at time 4 is equal to:

$$\alpha_4 = \alpha_0(\Omega_{1,\Sigma} + \Omega_{1,\epsilon})(\Omega_{2,\Sigma} + \Omega_{2,\epsilon})(\Omega_{3,\Sigma} + \Omega_{3,\epsilon})(\Omega_{4,\Sigma} + \Omega_{4,\epsilon})$$

and the denominator of the GNAT model is equal to $\sum_{j=0}^{6} \alpha_4[j]$. Replacing the real semiring with tropical semiring in above calculation will allow us to find the most likely alignment sequence.

B A Modular Framework

In this section we demonstrates how the existing and common neural speech recognition models can be expressed within our proposed framework.

B.1 Cross-entropy with Alignments

The conventional cross-entropy models with feed-forward neural architectures [31] define the conditional probability of label sequence $\mathbf{y}$ given feature sequence $\mathbf{x}$ by:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} P_\theta(y_t|x_t)$$

where probability factors $P_\theta(y_t|x_t)$ are derived by some neural architecture parameterized by θ:

$$P_\theta(y = y_t|x_t) = \frac{\exp(W[y_t, :] \cdot h_t + b[y_t])}{\sum_{y \in \Sigma} \exp(W[y, :] \cdot h_t + b[y])}$$

where h_t is the encoder activation of dimension D at time frame t, W is a weight matrix of shape $|\Sigma| \times D$ and b is a $|Sigma|$-dim bias vector.

The equivalent GNAT model is configured as follow:

- context dependency: 0-gram C_1
- alignment lattice: frame dependent without ϵ transitions
- weight function:
  - $\omega_{\theta,\mathbf{x}}(q_a = t, q_c = i, y = y_t) \triangleq P_\theta(y = y_t|x_t)$
  - locally normalized
  - streaming

here i is initial state of C_1.

The more advanced cross-entropy models use recurrent architectures or transformers as encoder [39]. The only difference between the GNAT equivalent of these models and above configuration is that whether the encoder is streaming or not.

B.2 Listen, Attend and Spell (LAS)

This model formulates the posterior probability by directly applying chain rule (Eq1 in [8]):

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{l} P_\theta(y_l|\mathbf{x}, y_{<l})$$

the posterior factors are defined as (Eq6-Eq8 of [8]):

$$P_\theta(y_l|\mathbf{x}, y_{<l}) = \text{CharacterDistribution}(s_l, c_l)$$

where

$$\begin{aligned} \mathbf{h} &= \text{Listen}(\mathbf{x}) \\ s_l &= \text{RNN}(s_{l-1}, y_{l-1}, c_{l-1}) \\ c_l &= \text{AttentionContext}(s_l, \mathbf{h}) \end{aligned}$$

here Listen is a bidirectional encoder function, AttentionContext is the attention network (Eq9-Eq11 of [8]).

The equivalent GNAT model is configured as follow:

- context dependency: M-gram where M is the length of longest label sequence in the training data. Note that M-gram context dependency is equivalent of the tree space truncated at depth M.

- alignment lattice: label dependent since the probability factorizes only on label sequence.
- weight function:
  - $\omega_{\theta,\mathbf{x}}(q_a = l, q_c = q, y = y_l) \triangleq P_\theta(y = y_l|\mathbf{x}, y_{<l})$
  - locally normalized
  - non-streaming

B.3 Recurrent Neural Transducer

The RNNT model formulate the posterior probability as marginalization of alignment sequences (Eq1 in the RNNT paper [13]):

$$P_\theta(\mathbf{y}|\mathbf{x}) = \sum_{\tilde{\mathbf{y}} \in B^{-1}(\mathbf{y})} P_\theta(\tilde{\mathbf{y}}|\mathbf{x})$$

where $\tilde{\mathbf{y}} = \tilde{y}_1, \cdots, \tilde{y}_{T+L}$ is an alignment sequence, $\tilde{y}_i \in \Sigma \cup \{\epsilon\}$, T is the number of acoustic frames and L is the number of labels. The function $B(\tilde{\mathbf{y}}) = \mathbf{y}$ removes the epsilons from the alignment sequence. The alignment posterior is factorized along the alignment path as:

$$P_\theta(\tilde{\mathbf{y}}|\mathbf{x}) = \prod_{j=1}^{T+L} P_\theta(\tilde{y}_j|\mathbf{x}, \tilde{y}_{<j})$$

and finally RNNT make the following assumption:

$$P_\theta(\tilde{y}_j|\mathbf{x}, \tilde{y}_{<j}) = P_\theta(\tilde{y}_j|\mathbf{x}, B(\tilde{y}_{<j}) = y_{<u})$$

which means if the prefix of two alignments be equal after epsilon removal, the model assigns same expansion probability for the next alignment position. The inner terms in the above equation is defined (Eq12-Eq15 of the RNNT paper):

$$P_\theta(\tilde{y}_j|\mathbf{x}, B(\tilde{y}_{<j}) = y_{<u}) = \frac{\exp(W[\tilde{y}_j, :] \cdot (h_{j-u} + g_u) + b[\tilde{y}_j])}{\sum_{y \in \Sigma \cup \{\epsilon\}} \exp(W[y, :] \cdot (h_{j-u} + g_u) + b[y])}$$

where h_{j-u} is the encoder activation at time frame $j - u$ (referred to as the transcription network in [13]) and g_u is the output of the prediction network which is a simple stack of RNNs.

The equivalent GNAT model is configured as follow:

- context dependency: M-gram where M is the maximum value of $T + L$ in the training data set.
- alignment lattice: k-constrained label and frame dependent with k being the label sequence length.
- weight function:
  - $\omega_{\theta,\mathbf{x}}(q_a = (t, u), q_c = q, y = \tilde{y}_{t+u+1}) \triangleq P_\theta(\tilde{y}_{t+u+1}|\mathbf{x}, B(\tilde{y}_{<t+u+1}) = y_{<u})$
  - locally normalized
  - non-streaming

While the original definition of the RNNT model is based on non-streaming encoder (transcription network), this model is widely used for streaming applications by using a streaming encoder. This is in contradiction of the forward-backward derivations in the original paper which explicitly assumes dependency on the whole sequence for any position of alignment sequence (Eq17 of [13])

Similar to RNNT, the hybrid autoregressive transducer (HAT) [40] model can be also configured in the GNAT framework with the same parametrization as RNNT. The only difference is the weight function. The HAT model defines different probabilities for label transitions and epsilon transitions (duration model in [40]):

$$P_\theta(\tilde{y}_j|\mathbf{x}, B(\tilde{y}_{<j}) = y_{<u}) = \begin{cases} b_{j-u,u} & \tilde{y}_j = \epsilon \\ (1 - b_{j-u,u})\, P_\theta\left(y_{u+1}|X, B(\tilde{y}_{<j}) = y_{<u}\right) & \tilde{y}_j \in \Sigma \end{cases} \tag{1}$$

where $b_{t,u}$ is a sigmoid function defined in Eq6 of [40].

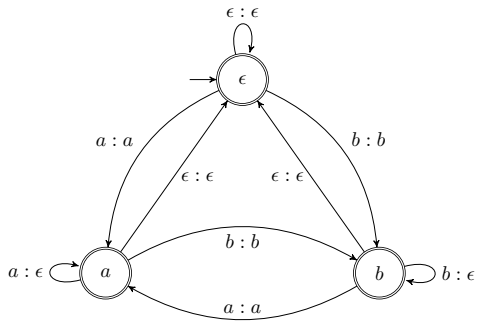


Figure 3: An unweighted FST for CTC style label deduplication with $\Sigma = \{a, b\}$.

B.4 Supporting CTC Style Label Deduplication

The standard CTC model [14] is very similar to a GNAT model using a frame dependent alignment lattice and a 0-gram context dependency. One key difference is that CTC introduces a deduplication process when turning its model output sequence to a label sequence. Each model output of a CTC model is either a lexical label from Σ, or the special ϵ (blank) label. To obtain the label sequence, two steps are applied on the model output in order,

1. Maximal consecutive repeated non-ϵ labels are merged into one (e.g. turning $abbc$ into abc, or $abb\epsilon b$ into $ab\epsilon b$);
2. All the ϵ labels are removed.

As a comparison, paths on the alignment lattice of the GNAT models in the main paper is equivalent to the model outputs in CTC, whereas the ϵ free label sequence seen by the context dependency is equivalent to the label sequence in CTC. To support the deduplication of repeated non-ϵ labels, we need to introduce a finite state transducer into our series of finite state machine compositions. Similar to a finite state automaton, a *weighted finite-state transducer* (WFST) $T = (\Sigma, Q, i, F, \rho, E)$ over a semiring $\mathbb{K}$ is specified by a finite alphabet Σ, a finite set of states Q, an initial state $i \in Q$, a set of final states $F \subseteq Q$, a final state weight assignment $\rho : F \to \mathbb{K}$, and a finite set of transitions E.[3] The meaning of Σ, Q, i, F, and ρ are identical to those of a WFSA. The set of transitions E is instead a subset of $Q \times (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$, i.e. containing a pair of input/output labels instead of just one. A WFSA can be viewed as a WFST with identical input/output labels on each arc, and similar to WFSA intersection, a series of WFST can be composed into a single WFST. We refer the readers to [28] for a full description of WFST and the composition algorithm. Figure 3 is an example unweighted FST, when composed with another input FSA or FST, performs the CTC style label deduplication. More generally, the unweighted label deduplication transducer D of vocabulary Σ consists of

- States $Q_D = \Sigma \cup \{\epsilon\}$, $i_D = \epsilon$, $F_D = Q_D$
- Transitions $E = \{(p, x, x, x) | p, x \in Q_D\} \cup \{(x, x, \epsilon, x) | x \in \Sigma\}$

Given a context dependency FSA C, an alignment lattice FSA L_T, and the weight function $\omega_{\theta,\mathbf{x}}$, as defined in Section 3.3, a GNAT model with CTC style label deduplication induces a WFST $T_{\theta,\mathbf{x}}$ as follows,

[3] Here we make the simplification that the input and output vocabularies are identical, i.e. Σ.

$$
\begin{aligned}
Q_{\theta,\mathbf{x}} &= Q_T \times Q_D \times Q_C \\
i_{\theta,\mathbf{x}} &= (i_T, i_D, i_C) \\
F_{\theta,\mathbf{x}} &= F_T \times F_D \times F_C \\
E_{T_{\theta,\mathbf{x}}} =& \Big\{ \big((q_a, q_d, q_c), y, y, \omega_{\theta,\mathbf{x}}(q_a, q_c, y), (q'_a, q'_d, q'_c)\big) \mid \\
& \quad y \in \Sigma,\ (q_a, y, q'_a) \in E_T,\ (q_d, y, y, q'_d) \in E_D,\ (q_c, y, q'_c) \in E_C \Big\} \\
& \cup \Big\{ \big((q_a, q_d, q_c), \epsilon, \epsilon, \omega_{\theta,\mathbf{x}}(q_a, q_c, \epsilon), (q'_a, q'_d, q_c)\big) \mid \\
& \quad (q_a, \epsilon, q'_a) \in E_T,\ (q_d, \epsilon, \epsilon, q'_d) \in E_D,\ q_c \in Q_C \Big\} \\
& \cup \Big\{ \big((q_a, q_d, q_c), y, \epsilon, \omega_{\theta,\mathbf{x}}(q_a, q_c, y), (q'_a, q'_d, q_c)\big) \mid \\
& \quad y \in \Sigma,\ (q_a, y, q'_a) \in E_T,\ (q_d, y, \epsilon, q'_d) \in E_D \Big\} \\
\rho_{T_{\theta,\mathbf{x}}}(q) &= \bar{1},\ \forall q \in F_{\theta,\mathbf{x}}
\end{aligned}
$$

In other words, the topology of $T_{\theta,\mathbf{x}}$ is the same as the following cascade of FST compositions,

1. $D \cdot C$ treating output ϵ labels in D as empty (i.e. standard FST composition).
2. $L_T \cdot (D \cdot C)$ treating ϵ transitions in L_T and input ϵ labels in $(D \cdot C)$ as regular labels.

and the transition weights are defined using $\omega_{\theta,\mathbf{x}}$ just like the GNAT models in the main paper.

When implemented naively, CTC style label deduplication causes a $|Q_D| = (V + 1)$ blow up in $|Q_{\theta,\mathbf{x}}|$. However, by inferring about states in Q_D from states in Q_C, we can greatly reduce the number of states needed. For each state $x \in \Sigma$ in Q_D, we know the last non-ϵ label observed when reaching state x must be label x. Similarly, for context dependencies we care about (n-gram and string), there is a unique label $x(q_c)$ for all incoming arcs of each non-start state q_c (start states do not have any incoming arcs in these context dependencies). Thus, the states in $Q_{\theta,\mathbf{x}}$ that are reachable from the start must match one of the following patterns,

- $(q_a, \epsilon, q_c),\ \forall q_a \in Q_A, q_c \in Q_X$
- $(q_a, x(q_c), q_c),\ \forall q_a \in Q_A, q_c \in Q_X \setminus \{i_X\}$

This means the actual number of states we shall visit in computing the shortest distance is only $2|Q_A||Q_C|$.

C Accelerator-Friendly Computation

The standard shortest distance/path algorithm for acyclic WFSA [27] can be used for training (computing $W(A)$ for some acyclic A) and inference of a GNAT model. To compute $W(A)$ for an acyclic WFSA A, we maintain the following forward weight α_q for each state q in Q_A:

$$
\alpha_q = \begin{cases} \bar{1} & \text{if } q = i_A, \\ \bigoplus_{(p,y,w,q) \in E_A} \alpha_p \otimes w & \text{else.} \end{cases}
$$

The weight of A is then $W(A) = \bigoplus_{q \in F_A} \alpha_q \otimes \rho_A(q)$. The recurrence in the definition of α_q can be computed by visiting states in Q_A in a topological order.

To make better use of the compute power of modern accelerator hardware, we observe the following properties of the C or L_T presented so far that enable us to use a more vectorized variant of the shortest distance algorithm in Figure 4:

- From any topological ordering on Q_T, we can derive a topological ordering on $Q_{\theta,\mathbf{x}}$.
- The n-gram context dependency FSA C_n is deterministic, namely leaving any state there is no more than 1 transition for any label $y \in \Sigma$, and there is no ϵ-transition.
- For all three types of alignment lattices, for any non-final state $q \in Q_T \setminus F_T$, there is a unique next state $\mathrm{succ}(q)$ for transitions leaving q consuming any label $y \in \Sigma$.

{Initialize the length $|Q_C|$ forward weight vectors $\bar{\alpha}_{q_a}$}
for all $q_a \in Q_T$ **do**
  $\bar{\alpha}_{q_a} \leftarrow [\bar{0}, \ldots, \bar{0}]$
end for
$\bar{\alpha}_{i_T}[i_C] \leftarrow \bar{1}$
{Compute $\bar{\alpha}_{q_a}$ for $q_a \neq i_T$}
for all $q_a \in Q_T$ in topological order **do**
  {Ω is a $[|Q_C|, |\Sigma| + 1]$ matrix}
  $\Omega \leftarrow \bar{\omega}_{\theta,\mathbf{x}}\big(q_a, Q_C, \Sigma \cup \{\epsilon\}\big)$
  if q_a has outgoing label transitions to $q'_a = \mathrm{succ}(q_a)$ **then**
    $\bar{\alpha}_{q'_a} \leftarrow \bar{\alpha}_{q'_a} \bar{\oplus}\, \mathrm{next}_C\big(\bar{\alpha}_{q_a}, \Omega[:, \Sigma]\big)$
  end if
  for all q'_a such that $(q_a, \epsilon, q'_a) \in E_T$ **do**
    $\bar{\alpha}_{q'_a} \leftarrow \bar{\alpha}_{q'_a} \bar{\oplus}\big(\bar{\alpha}_{q_a} \bar{\otimes} \Omega[:, \epsilon]\big)$
  end for
end for
return $\bigoplus_{q_a \in F_T, q_c \in F_C} \bar{\alpha}_{q_a}[q_c]$

Figure 4: The vectorized shortest distance algorithm for $A_{\theta,\mathbf{x}}$. We denote $\bar{\oplus}$, $\bar{\otimes}$, and $\bar{\omega}_{\theta,\mathbf{x}}$ the vectorized versions of the corresponding operations.

{Inputs: $\bar{\alpha}_{q_a}$ and $\Omega[:, \Sigma]$}
if $n = 1$ **then**
  {Q_{C_n} contains only i_{C_n}}
  return $\bar{\alpha}_{q_a} \bar{\otimes} \bigoplus_{y \in \Sigma} \Omega[i_{C_n}, y]$
end if
{Initialize length $|Q_{C_n}|$ vector $\bar{\alpha}$}
$\bar{\alpha} \leftarrow [\bar{0}, \ldots, \bar{0}]$
{States in Q_{C_n} are numbered from 0 to $|Q_{C_n}| - 1 = \sum_{i=1}^{n-1} |\Sigma|^i$ following the lexicographic order}
$l \leftarrow 0$
for $i = 0$ **to** $n - 2$ **do**
  $h \leftarrow l + |\Sigma|^i$
  $\bar{\alpha}[l \cdot |\Sigma| + 1 : h \cdot |\Sigma| + 1] \leftarrow \mathrm{flatten}(\Omega[l : h, \Sigma])$
  $l \leftarrow h$
end for
for $i = 0$ **to** $|\Sigma| - 1$ **do**
  $\bar{\alpha}[l :] \leftarrow \bar{\alpha}[l :] \bar{\oplus}\, \mathrm{flatten}(\Omega[l + i \cdot |\Sigma|^{n-2} : l + (i + 1) \cdot |\Sigma|^{n-2}, \Sigma])$
end for
return $\bar{\alpha}$

Figure 5: Specialized implementation of next_{C_n}. The $\mathrm{flatten}$ function flattens a matrix into a vector by joining the rows.

Center to an efficient implementation of the algorithm in Figure 4 is the function next_C. This function receives as input the current forward weight vector $\bar{\alpha}_{q_a}$ for states (q_a, q_c), $\forall q_c \in Q_C$, and the transition weights for leaving these states via label transitions, and returns the forward weights going to states (q'_a, q'_c) by taking the (q_c, y, q'_c) transitions for $y \in \Sigma$. In other words, $\mathrm{next}_C[q'_c] = \bigoplus_{(q_c, y, q'_c) \in E_C} \bar{\alpha}_{q_a}[q_c] \otimes \Omega[q_c, y]$. The n-gram context dependency C_n allows a particularly simple and efficient implementation of next_{C_n}, as outlined in Figure 5. The key observation is that when we number the states in Q_C following the lexicographic order, the $|\Sigma|$ transitions leaving the same q_c lead to states in a consecutive range $[\sigma(q_c y_0), \ldots, \sigma(q_c y_{|\Sigma|-1})]$, where $[y_0, \ldots, y_{|\Sigma|-1}]$ are the lexicographically sorted labels of Σ, and $\sigma(s)$ is the suffix of label sequence s of length up to $n - 1$.

During training, we also need to compute the shortest distance $D(A_{\theta,\mathbf{x}} \cap \mathbf{y})$. We note the algorithm in Figure 4 can also be used for this purpose since $(L_T \cap C) \cap \mathbf{y} = L_T \cap (C \cap \mathbf{y})$, and we simply need to substitute C with $C \cap \mathbf{y}$ in the algorithm.

D Memory and Computation Time Benchmarks

The memory and computation benchmark of our implementation for the GNAT model is presented in Table 2. We present benchmarks for training and inference for different configurations of the GNAT model:

- Context dependency: 0-gram, 1-gram and 2-gram
- Alignment lattice: frame dependent, 1-constrained label and frame dependent
- Weight functions: Per-state linear projection (unshared), Shared linear projection with per-state embedding (shared-emb), Shared linear projection with RNN state embedding (shared-rnn)

context dependency	alignment lattice	weight function		memory [M]		time [sec]	
		type	normalization	train	decode	train	decode
0-gram	frame	unshared	local	126.47	64.97	0.15	0.02
			global	124.58	65.19	0.14	0.02
		shared-emb	local	124.20	65.12	0.20	0.02
			global	124.64	65.19	0.18	0.02
		shared-rnn	local	124.43	65.16	0.20	0.02
			global	124.88	65.29	0.18	0.02
	label frame	unshared	local	174.62	65.00	0.16	0.02
			global	172.21	65.20	0.18	0.04
		shared-emb	local	172.32	65.15	0.20	0.02
			global	172.25	65.17	0.22	0.03
		shared-rnn	local	172.55	65.19	0.20	0.02
			global	172.49	65.27	0.22	0.04
1-gram	frame	unshared	local	144.04	64.95	0.17	0.044
			global	146.23	65.51	0.19	0.05
		shared-emb	local	156.02	70.42	0.22	0.05
			global	158.12	70.52	0.22	0.05
		shared-rnn	local	157.04	70.67	0.23	0.05
			global	159.15	70.76	0.23	0.05
	label frame	unshared	local	192.19	64.98	0.18	0.04
			global	192.70	65.19	0.27	0.07
		shared-emb	local	204.13	70.45	0.23	0.05
			global	204.65	70.14	0.29	0.07
		shared-rnn	local	205.16	70.69	0.23	0.05
			global	205.68	70.38	0.30	0.07
2-gram	frame	unshared	local	306.94	187.21	0.23	0.07
			global	513.58	195.36	1.55	0.41
		shared-emb	local	156.05	70.42	0.23	0.05
			global	181.42	73.49	1.16	0.23
		shared-rnn	local	174.22	73.04	0.23	0.05
			global	199.62	76.11	1.16	0.23
	label frame	unshared	local	320.98	187.24	0.24	0.07
			global	428.40	187.90	3.79	0.94
		shared-emb	local	204.17	70.45	0.24	0.05
			global	210.61	71.42	2.63	0.48
		shared-rnn	local	222.33	73.07	0.24	0.05
			global	229.17	74.04	2.64	0.49

Table 2: Memory and computation benchmarks of the GNAT model for different configurations.

For each configuration, the memory usage footprint is presented in terms of MB and total computation time is presented in terms of number of seconds. The benchmarks do not include the memory and computation footprint of the encoder activations. The training benchmarks are corresponding to the calculation of the GNAT criterion as well as all the backward gradient calculation up to the encoder

activations. The evaluation benchmarks only contain the forward pass memory and compute footprint to find the most likely hypothesis.

All the memory and computation benchmarks are evaluated for an input batch of 32 examples each with 1024 number of frames. Each frame is a 512-dim vector corresponding to the encoder activations. Each example in the input batch are assumed to have at most 256 labels. The alphabet size is set to 32.

The main observations are:

- The larger context dependency lead to more memory and compute footprint. This is expected since the computation complexity is directly related to the context dependency state size. However, interestingly, the memory and computation values do not scale exponentially by value of n in n-gram context dependency (as a result by number of states in the context dependency).
- label frame dependent alignment lattice generally leads to higher memory usage and computation time compare to the frame dependent alignment lattice. This is expected since the label frame dependent consist of alignment paths of length $1024 + 256 = 1270$, corresponding to the sum of number of frames and number of labels.
- The per-state linear projection weight function requires more memory and has longer compute time compare to the shared weights function which is expected by design. Both shared weight functions are performing on-par of each other in terms of memory and compute.
- The global normalization requires more memory and time and the difference is more significant for context dependency FSAs with more number of states (2-gram versus 1-gram).

E Equivalence of Certain Locally and Globally Normalized Models under Non-Streaming Settings

As widely discussed in literature (e.g. Section 4.6.12 of [23], [24, 2]), under streaming settings, a locally normalized model introduces an independence assumption between an output z_i and future inputs $\mathbf{x}_{\geq t(i)}$, when future outputs $\mathbf{z}_{>i}$ are not yet known. On the contrary, a globally normalized model using the same score factorization does not introduce the same independence assumption. This is the key reason to a locally normalized model's inability to express certain conditional distributions and thus prone to the label bias problem.

However, under non-streaming settings, the entire input sequence $\mathbf{x}$ is available for predicting each z_i. The aforementioned independence assumption is thus no longer present for locally normalized models. We shall show that under a mild condition regarding how an alignment score $\omega(z_i|\mathbf{z}_{<i}, \mathbf{x})$ makes use of information from the past outputs $\mathbf{z}_{<i}$, for every non-streaming globally normalized model, there is a non-streaming locally normalized model that defines the same conditional distribution for the output sequence.

For an alignment score $\omega(z_i|\mathbf{z}_{<i}, \mathbf{x})$, for each output step i, define the *past output index set* $\delta_i \subseteq \{1, 2, \ldots, i-1\}$ to consist of the indices of past outputs on which $\omega(z_i|\mathbf{z}_{<i}, \mathbf{x})$ actually depend. For example, for a GNAT model using a full n-gram dependency with context size 2, $\delta_i = \{i-1, i-2\}$, i.e. the alignment score only depends on the output of the previous two outputs. On the other hand, an RNN-T model with unlimited history has $\delta_i = \{1, 2, \ldots, i-1\}$. When comparing locally and globally normalized models, it is important to ensure the same past output index sets are used. Otherwise the comparison is not fair since the parameterization of these models will differ.

Given the sequence of past output index sets $[\delta_1, \delta_2, \ldots]$, the alignment score function $\omega(z_i|\mathbf{z}_{<i}, \mathbf{x})$ can also be written as $\omega(z_i|\mathbf{z}_{\delta_i}, \mathbf{x})$, with $\mathbf{z}_{\delta_i} = \{z_j | j \in \delta_i\}$. The sequence of past output index sets *grows monotonically* if for all i, $\delta_i \cap \{1, 2, \ldots, i-2\} \subseteq \delta_{i-1}$. In other words, each δ_i can be constructed from δ_{i-1} by first removing some elements then optionally adding $i-1$. Importantly if some $j < i$ is not in δ_i, then j is not in any subsequent δ_k for any $k \geq i$. Notably, all existing work on neural globally normalized ASR models including this paper itself use score functions where the sequence of past output index sets grows monotonically [17, 10, 42, 44].

When the sequence of past output index sets grows monotonically, globally normalized models and locally normalized ones are in fact equivalent in terms of the conditional distributions that they can express. Since any locally normalized model is by definition globally normalized, we only need to show the other direction.

Theorem E.1. *For any non-streaming globally normalized model with alignment score $\omega_G(z_i|\mathbf{z}_{\delta_i}, \mathbf{x})$, if the sequence of past index sets $\{\delta_1, \delta_2, \ldots\}$ grows monotonically, then there exists a non-streaming locally normalized model with alignment score of the same sequence of index sets $\omega_L(z_i|\mathbf{z}_{\delta_i}, \mathbf{x})$, such that $P_{\omega_G}(\mathbf{z}|\mathbf{x}) = P_{\omega_L}(\mathbf{z}|\mathbf{x})$.*

This theorem can be seen a result of such a globally normalized model's corresponding Markov network representation being chordal (see Theorem 4.13 of [23]). In an effort to be self-contained , below we instead present a direct proof that does not rely on any results on graphical models, using essentially the same technique as [9, 36].

Proof. For any prefix $\mathbf{z}_{\leq i}$, the sum of the scores of all possible suffices continuing $\mathbf{z}_{\leq i}$ is commonly known as the *backward score*. More specifically, in our case, the backward score at position i is

$$\beta_i = \sum_{\mathbf{z}'|\mathbf{z}'_{<i+1}=\mathbf{z}_{<i+1}} \prod_{j=i}^{|\mathbf{z}'|} \omega_G(z'_j|\mathbf{z}'_{\delta_j}, \mathbf{x})$$

It is easy to see that β_i only depends $\mathbf{x}$ and z_j for $j \in \{i\} \cup \delta_i$. Thus $\beta_i(z_i, \mathbf{z}_{\delta_i}, \mathbf{x})$ is a well-defined function. β_i is upper bounded by the denominator $Z(\mathbf{x})$ and thus finite. $\omega_G(z_j|\mathbf{z}_\delta, \mathbf{x}) > 0$ implies $\beta_i > 0$. We can thus define the following alignment score which is a locally normalized model,

$$\omega_L(z_i|\mathbf{z}_{\delta_i}, \mathbf{x}) = \frac{\beta_i(z_i, \mathbf{z}_{\delta_i}, \mathbf{x})}{\sum_{z'_i} \beta_i(z'_i, \mathbf{z}_{\delta_i}, \mathbf{x})}$$

Then

$$\begin{aligned}
P_{\omega_G}(z_i|\mathbf{z}_{<i}, \mathbf{x}) =& \frac{\sum_{\mathbf{z}'|\mathbf{z}'_{<i+1}=\mathbf{z}_{<i+1}} \prod_{j=1}^{|\mathbf{z}'|} \omega_G(z'_j|\mathbf{z}'_{\delta_j}, \mathbf{x})}{\sum_{\mathbf{z}'|\mathbf{z}'_{<i}=\mathbf{z}_{<i}} \prod_{j=1}^{|\mathbf{z}'|} \omega_G(z'_j|\mathbf{z}'_{\delta_j}, \mathbf{x})} \\
=& \frac{\left(\prod_{j=1}^{i-1} \omega_G(z_j|\mathbf{z}_{\delta_j}, \mathbf{x})\right) \left(\sum_{\mathbf{z}'|\mathbf{z}'_{<i+1}=\mathbf{z}_{<i+1}} \omega_G(z_i|\mathbf{z}_{\delta_i}, \mathbf{x}) \prod_{j=i+1}^{|\mathbf{z}'|} \omega_G(z'_j|\mathbf{z}'_{\delta_j}, \mathbf{x})\right)}{\left(\prod_{j=1}^{i-1} \omega_G(z_j|\mathbf{z}_{\delta_j}, \mathbf{x})\right) \left(\sum_{\mathbf{z}'|\mathbf{z}'_{<i}=\mathbf{z}_{<i}} \prod_{j=i}^{|\mathbf{z}'|} \omega_G(z'_j|\mathbf{z}'_{\delta_j}, \mathbf{x})\right)} \\
=& \frac{\beta_i(z_i, \mathbf{z}_{\delta_i}, \mathbf{x})}{\sum_{z'_i} \beta_i(z'_i, \mathbf{z}_{\delta_i}, \mathbf{x})} \\
=& \omega_L(z_i|\mathbf{z}_{\delta_i}, \mathbf{x}) \\
=& P_{\omega_L}(z_i|\mathbf{z}_{<i}, \mathbf{x})
\end{aligned}$$

Therefore, $P_{\omega_G}(\mathbf{z}|\mathbf{x}) = P_{\omega_L}(\mathbf{z}|\mathbf{x})$. □