Star Temporal Classification: Sequence Modeling with Partially Labeled Data

Vineel Pratap
Meta AI
vineelkpratap@meta.com

Awni Hannun
Zoom AI
awni.hannun@zoom.us

Gabriel Synnaeve
Meta AI
gab@meta.com

Ronan Collobert*
Meta AI
collobert@apple.com

Abstract

We develop an algorithm which can learn from partially labeled and unsegmented sequential data. Most sequential loss functions, such as Connectionist Temporal Classification (CTC), break down when many labels are missing. We address this problem with Star Temporal Classification (STC) which uses a special *star* token to allow alignments which include all possible tokens whenever a token could be missing. We express STC as the composition of weighted finite-state transducers (WFSTs) and use GTN (a framework for automatic differentiation with WFSTs) to compute gradients. We perform extensive experiments on automatic speech recognition. These experiments show that STC can close the performance gap with supervised baseline to about 1% WER when up to 70% of the labels are missing. We also perform experiments in handwriting recognition to show that our method easily applies to other sequence classification tasks.

1 Introduction

Applications of machine learning in settings with little or no labeled data are growing rapidly. Much prior work in temporal classification focuses on semi-supervised or self-supervised learning [6, 37]. Instead, we focus on weakly supervised learning, in which the learner receives incomplete labels for the examples in the data set. Examples include multi-instance learning [1, 44], and partial-label learning [18, 9, 25]. In our case, we assume that each example in the training set is partially labeled, as in Figure 1: neither the number of missing words nor their positions in the label sequence are known in advance.

Input:

Original Label: seen from an airplane the island looks like a big spider

Partial Label: from airplane the a spider

Figure 1: An example of speech with a complete and a partial label.

Partial labeling problem could arise in many practical scenarios. In semi-supervised learning or transfer learning of sequence labeling tasks, we can only consider high confidence tokens in the label

*Currently at Apple.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

sequence and discard rest of the tokens. These labels with incomplete tokens can be treated as partial labels. When transcribing a dataset, we can randomly label parts of the dataset and use them as partial labels, at a much lower cost than transcribing the full dataset. We can also consider semi-automatic ways to label a dataset. Restoring damaged texts [2, 3] where some of the writing is illegible could be considered as a partial labeling problem. We also often face partial alignment problem [5] in practice where beginning and ending tokens are missing. This is a sub-problem of the more general version of partial labeling problem that we solve in this work.

To solve this weak supervision problem we develop the Star Temporal Classification (STC) loss function. The STC algorithm explicitly accounts for the possibility of missing labels in the output sequence at any position, through a special *star* token. For the example of Figure 1, the STC label sequence is `* from * airplane * the * a * spider *` (where `*` can be anything but the following label token). We show how to simply implement STC in the weighted finite-state transducer (WFST) framework. Moreover, through the use of automatic differentiation with WFSTs, gradients of the loss function with respect to the inputs are computed automatically, allowing STC to be used with standard neural network training pipelines.

For practical applications, a naive implementation of STC using WFSTs is intractably slow. We propose several optimizations to improve the efficiency of the STC loss. The *star* token we use results in much smaller graphs since it combines many arcs into one. The *star* token also enables us to substantially reduce the amount of memory transfers between the GPU, on which the network is computed, and the CPU, on which the STC algorithm is evaluated. These optimizations result in only a minor increase in model training time when using the STC loss over a CTC baseline.

The STC loss can be applied to most sequence transduction tasks including speech recognition, handwriting recognition, machine translation, and action recognition from videos. Here we demonstrate the effectiveness of STC on both automatic speech recognition and offline handwriting recognition. In speech recognition, with up to 70% of the labels missing, STC is able to achieve similar (about 1% WER difference) performance as supervised baseline . Similarly in handwriting recognition STC can achieve low character error rates with up to 50% of the labels missing. In both cases STC yields dramatic gains over a baseline system which does not explicitly handle missing labels.

To summarize, the main contributions of this work are:

- We introduce STC, a novel temporal classification algorithm which explicitly handles an arbitrary number of missing labels with an indeterminate location.
- We simplify the implementation of STC by using WFSTs and automatic differentiation.
- We describe key optimizations for STC enabling it to be used with minimal increase in training time.
- We show in practical speech and handwriting recognition tasks that STC yields low word and character error rates with up to 70% of the labels missing.

2 Related Work

Many generalizations and variations of CTC have been proposed. Graph-based temporal classification (GTC) [32] generalizes CTC to allow for an n-best list of possible labels. [40] also extends CTC for using with for multiple labels. [23] propose selfless-CTC which disallows self-loops for non-blank output tokens. Unlike these alternatives, STC allows for an arbitrary number of missing tokens anywhere in the label for a given example. A recent work, concurrent with this work, proposes a wild card version of CTC [5]. However, in that work the wild cards, or missing tokens, can only be at the beginning or end of the label.

Extended Connectionist Temporal Classification (ECTC) [17] was developed for weakly supervised video action labeling. However, the weak supervision in this case refers to the lack of a segmentation, which is the common paradigm for CTC. The extensions to CTC are intended to handle the much higher ratio of input video frames to output action labels. [11] develop a sequence level loss function which can learn from noisy labels. They extend the auto segmentation criterion [7] to include a noise model which explicitly handles up to one missing label between tokens in the output. Unlike that work, STC extends the more commonly used CTC loss and allows for any number of missing tokens.

The idea of expressing CTC as the composition of WFSTs is well known [27, 23, 41]. However, recently developed WFST-based frameworks which support automatic differentiation [15, 19] make the development of variations of the CTC loss much simpler.

WFSTs have been modified by to take into account of noisy transcripts [16, 31, 4] by composing the WFST graphs of word sequences. However, in this work the WFST composition is between significantly larger WFST graphs (label sequence and neural network model output sequence). Also, the composition is performed during the training of the neural network which makes it a more challenging engineering problem.

3 Method

3.1 Problem Description

We consider the problem of temporal classification [20], which predicts an output sequence $\mathbf{y} = [y_1, ..., y_U] \in \mathcal{A}^{1\times U}$, where $\mathcal{A}$ is a fixed alphabet of possible output tokens, from an unsegmented input sequence $\mathbf{x} = [x_1, ..., x_T] \in \mathbb{R}^{d\times T}$, where each x_i is a d-dimensional feature vector. We also assume that the length of the output U is less than or equal to that of the input, T. A partially labeled output sequence $\tilde{\mathbf{y}}$ of length $\tilde{U}$, such that ($\tilde{U} \leq U$), is formed by removing zero or more elements from the true output sequence $\mathbf{y}$. Given a training set $\mathcal{D} \triangleq \{\mathbf{x}^i, \tilde{\mathbf{y}}^i : i = 1, \ldots, N\}$ consisting of input sequences $\mathbf{x}^i = [x_1, \ldots, x_{T^i}]$ and partially labeled output sequences $\tilde{\mathbf{y}}^i = [y_1, \ldots, y_{\tilde{U}^i}]$ our goal is to learn temporal models which can predict true output sequences $\mathbf{y}^i$. We consider this a weakly supervised temporal classification problem as the learning algorithm only has access to a subset of the true output labels during training.

3.2 Connectionist Temporal Classification (CTC)

Connectionist Temporal Classification (CTC) [13] is a widely used loss function for training sequence models where the alignment of input sequence with the target labels is not known in advance.

Consider an input sequence $\mathbf{x} = [x_1, \ldots, x_T]$ and target sequence $\mathbf{y} = [y_1, \ldots, y_U] \in \mathcal{A}^{1\times U}$, where $\mathcal{A}$ is a finite alphabet of possible output tokens and $U \leq T$. The CTC loss uses a special *blank* token, $\langle b \rangle$, to represents frames which do not correspond to an output token. An alignment between the input $\mathbf{x}$ and output $\mathbf{y}$ in CTC is represented by a sequence of length T; $\boldsymbol{\pi} = [\pi_1, \pi_2, ..., \pi_T]$ where $\pi_t \in \mathcal{A} \cup \{\textit{blank}\}$. The CTC collapse function $\mathcal{B}$ maps an alignment to an output, $\mathcal{B}(\pi) = \mathbf{y}$. The function $\mathcal{B}$ removes all but one of any consecutively repeated tokens and then removes *blank* tokens e.g., (a, b, *blank*, *blank*, b, b, *blank*, a) $\mapsto$ (a, b, b, a). Given the input $\mathbf{x}$, each output is independent of all other outputs, hence the probability of an alignment $P(\boldsymbol{\pi}|\mathbf{x})$ is:

$$P(\boldsymbol{\pi}|\mathbf{x}) = \prod_{i=1}^{T} P(\pi_t|\mathbf{x}). \tag{1}$$

There can be many possible alignments $\boldsymbol{\pi}$ for a given $\mathbf{x}$ and $\mathbf{y}$ pair. The CTC loss computes the negative log probability of $\mathbf{y}$ given $\mathbf{x}$ by marginalizing over all possible alignments:

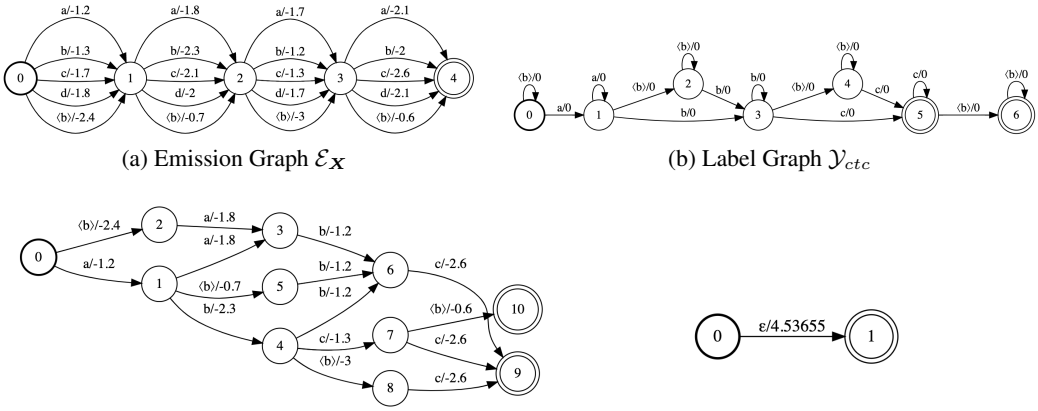
$$L_{CTC} = -\log P(\mathbf{y}|\mathbf{x}) = -\log \sum_{\boldsymbol{\pi}\in\mathcal{B}^{-1}(\mathbf{y})} P(\boldsymbol{\pi}|\mathbf{x}). \tag{2}$$

The sum over all alignments can be computed efficiently with dynamic programming as described by [13].

The CTC loss can be computed purely with WFST[2] operations as described in Figure 2. The composition of the emission graph (Figure 2a) which has arc weights weights corresponding to $\log P(\pi_t|\mathbf{x})$ and the label graph (Figure 2b) is shown in Figure 2c. Each path from a start to a final state in Figure 2c is a valid CTC alignment. The negation of the forward score of the composed graph gives the CTC loss. These operations are differentiable and gradients can easily be propagated backward from the loss to a neural network to train the model.

Constructing the CTC loss from simpler WFST graphs also simplifies the implementation of variations of CTC. For example, the variations of [32] only require changes to the label graph $\mathcal{Y}_{ctc}$. However,

[2] A brief introduction to WFSTs can be found in the Appendix A



(a) Emission Graph $\mathcal{E}_{\boldsymbol{X}}$

(b) Label Graph $\mathcal{Y}_{ctc}$

(c) Alignment Graph $\mathcal{A}_{ctc} = \mathcal{Y}_{ctc} \circ \mathcal{E}_{\boldsymbol{X}}$

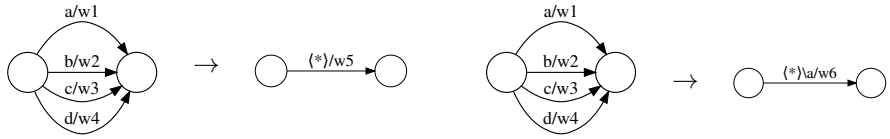
(d) Loss Graph $\mathcal{L}_{ctc} = -Fwd(\mathcal{A}_{ctc})$

Figure 2: The sequence of steps involved in computing CTC loss using WFSTs. The arc label "p/w" is a shorthand notation for "p:p/w". (a) is the emissions graph constructed from the log probabilities over the alphabet $\mathcal{A} = \{a, b, c, d\}$ and blank symbol $\langle b \rangle$. The CTC label graph corresponding to the target sequence (a, b, c) is shown in (b). In (c), we compose label graph with emission graph to get all valid paths which collapse to the target sequence. Finally, in (d) we sum the probabilities of all of the valid paths (in log-space) and negate the result to yield the CTC loss.

when efficiency is a primary concern, custom GPU kernels can be faster than a generic implementation with WFSTs.

3.3 Star Temporal Classification (STC)

The label graph of CTC, $\mathcal{L}_{ctc}$ (Figure 2b), constructed from partial labels does not allow for the true target as a possibility. The STC algorithm addresses this problem by allowing for zero or more tokens from the alphabet between any two tokens in the partial label. Like CTC, an alignment between the input $\mathbf{x}$ and output $\mathbf{y}$ in STC is represented by a sequence of length T; $\boldsymbol{\pi} = [\pi_1, \pi_2, ..., \pi_T]$ where $\pi_t \in \mathcal{A} \cup \{blank\}$ and $\mathbf{y}$ is a partial label of $\mathcal{B}'(\pi)$. Unlike CTC, the STC collapse function $\mathcal{B}'(\pi)$ only removes blank tokens. In other words, we do not allow self-loops on non *blank* tokens in the STC label graph. This enables us to use a token insertion penalty as discussed in Section 3.3.1.



(a) Collapsing to $\langle * \rangle$ token
w5 = logsumexp(w1,w2,w3,w4)

(b) Collapsing to $\langle * \rangle \backslash a$ token
w6 = logsumexp(w2,w3,w4)

Figure 3: Example showing the collapsing of tokens in alphabet $\mathcal{A} = \{a, b, c, d\}$ to star token, $\langle * \rangle$

To make the computation of the STC loss efficient, STC uses a special *star* token, $\langle * \rangle$, which represents every token in the alphabet. It can be used to collapse the arcs between any two nodes in the WFST graph as shown in Figure 3a. We also define $\langle * \rangle \backslash t = \{y : y \in \mathcal{A}; y \neq t\}$ which is the relative complement of t in $\mathcal{A}$. We use the token, $\langle * \rangle \backslash t$, in the label graph of STC, $\mathcal{L}_{stc}$, to avoid counting the same alignment multiple times as dicsussed in Section 3.3.2 The STC label graph, $\mathcal{L}_{stc}$, for the partial label (a, b, c) is shown in Figure 4. The graph allows any alignment with zero or more tokens in between any two tokens of the given partial label sequence.

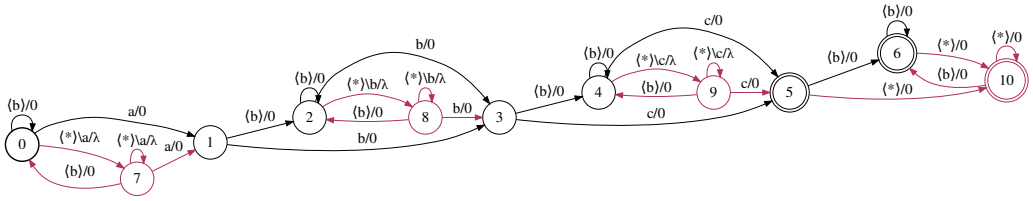


Figure 4: STC Label Graph, $\mathcal{Y}_{stc}$ for the output sequence (a, b, c). In regex terms, this corresponds to "[^a]$*$ a [^b]$*$ b [^c]$*$ c .$*$ ". $\langle * \rangle$, $\langle b \rangle$ refer to *star*, *blank* tokens and $\langle * \rangle \backslash a$ is the relative complement of a in $\langle * \rangle$. $\lambda (\leq 0)$ corresponds to token insertion penalty which has a regularization effect. The arc transitions marked in red show the changes from a selfless-CTC label graph.

We also manually add the $\langle * \rangle$ and $\langle * \rangle \backslash t$ tokens to the emission graph $\mathcal{E}_{\boldsymbol{X}}$ with weights corresponding to their log-probabilities for each timestep $\mathbf{t}$ given by:

$$
\begin{aligned}
P(\langle * \rangle | \mathbf{x}, \mathbf{t}) &= \sum_{y \in \mathcal{A}} P(y | \mathbf{x}, \mathbf{t}) = 1 - P(\langle b \rangle | \mathbf{x}, \mathbf{t}) \\
P(\langle * \rangle \backslash t | \mathbf{x}, \mathbf{t}) &= \sum_{y \in \mathcal{A}; y \neq t} P(y | \mathbf{x}, \mathbf{t}).
\end{aligned} \tag{3}
$$

3.3.1 Token Insertion Penalty, λ

We noticed that model training with STC does not work if we train directly with the STC label graph, even when 10% of the labels are missing. This is because the new paths introduced in STC from CTC (marked in red in Figure 4) would allow the model to produce a lot of possible output sequences for any given example and can confuse the model, especially during the early stages of training. To circumvent this problem, we use a parameter called token insertion penalty, λ to add a penalty when using lot of new tokens. We define STC loss in the following way

$$
\text{STC Loss} = -\log(p^{k_1} P_{path_1} + p^{k_2} P_{path_2} + ... + p^{k_N} P_{path_N})
$$

where P_{path_i} is the probability of an alignment path i, $p \in (0, 1]$ and k_i is the number of additional star tokens for path i. This down weight paths which include many additional tokens. The token insertion penalty is in log-space and is defined as $\lambda = \log(p)$.

We use a exponential decay scheme to gradually reduce the penalty as the model starts training.

$$
\begin{aligned}
p_t &= p_{max} + (p_0 - p_{max}) exp(-t/\tau) \\
\lambda_t &= ln(p_t)
\end{aligned} \tag{4}
$$

where p_0, p_{max}, τ are hyperparameters and λ_t denotes the token insertion penalty used in STC for training step t. For choosing value of τ, it is useful think in terms of half-life $t_{1/2} = \tau ln(2)$, which is the number of time steps taken for p_t to reach $(p_0 + p_{max})/2$

3.3.2 On the importance of $\langle * \rangle \backslash t$ token

The "$\langle * \rangle \backslash t$" token which matches any token except the token, t is useful to not count the same alignment path multiple times while computing the STC loss. We illustrate it through the two possible STC label graphs from Figure 5. Consider that the partial label has just one word "the" and number of time steps, T = 3. In Figure 5a, the alignment path {"the", "the", "cat"} is counted multiple times when we use start token $\langle * \rangle$ directly $0 \rightarrow 1 \rightarrow 2 \rightarrow 4$ and $0 \rightarrow 2 \rightarrow 4 \rightarrow 4$. However, if we use $\langle * \rangle \backslash the$ token as shown in Figure 5b , we do not have this issue of accepting the path "the the cat" multiple times.

It should also be noted that the "$\langle * \rangle \backslash t$" token is only present after the token t but not before the token t in the label graph of STC from Figure 4. So, the label graph would still accept all possible labels sequences for the given partial label.

3.3.3 Implementation Details

With the recent advent of frameworks such as GTN[15], K2[19] which support autograd on WFSTs, we focus on the engineering details used by us to use these frameworks and train large-scale recognition systems below.

We use the CPU-based WFST algorithms from GTN [15] to implement the STC criterion. Multiple threads are used to compute STC loss in parallel for all of the examples in a batch. Since the neural network model is run on GPU, the emissions needed by STC must be copied to the CPU, and the STC gradients must be copied back to GPU. To reduce the amount of data transfer between the CPU and the GPU, we transfer values corresponding only to the tokens present in the partial labels and the *star* tokens. This works since the gradients corresponding to all other tokens are zero.

Figure 6 shows an overview of the STC training pipeline. The input sequence is typically passed through a neural network on GPU to produce a frame wise distribution (in log-space) over $\mathcal{A} \cup \{blank\}$. The output number of frames depend on the size of the input sequence, as well as the amount of padding and the stride of the neural network model architecture. We then compute the log-probability of star token, $\langle * \rangle$ and $\langle * \rangle \backslash t$ tokens $\forall t \in \mathbf{y}$ using Equation 3. We construct the emission graph $\mathcal{E}_{\boldsymbol{X}}$ with weights as log-probabilities and the label graph $\mathcal{L}_{stc}$ on CPU. Using the WFST operations shown in Figure 2, we compute STC loss and gradients of loss with respect to arc weights (log probability output) of $\mathcal{E}_{\boldsymbol{X}}$. These gradients are copied back to GPU and the gradients of parameters in neural network are computed using backpropagation. Then the model can be trained using standard gradient descent methods.

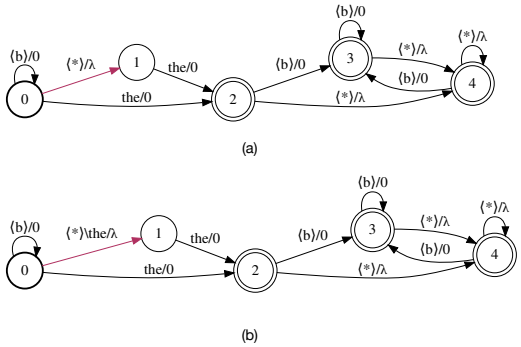


Figure 5: Examples to illustrate the importance of $\langle * \rangle \backslash t$ token

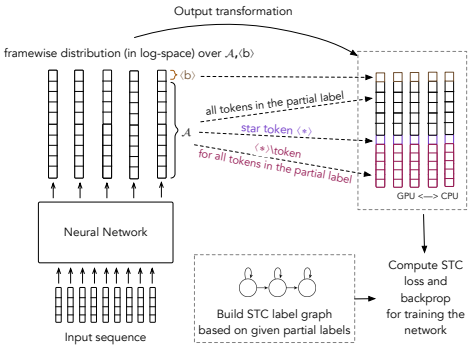


Figure 6: The STC Training Pipeline

4 Experimental Setup

4.1 Automatic Speech Recognition (ASR)

We use LibriSpeech [33] dataset, containing 960 hours of training audio with paired transcriptions for our speech recognition experiments. The standard LibriSpeech validation sets (*dev-clean* and *dev-other*) are used to tune all hyperparameters, as well as to select the best models. We report final word error rate (WER) performance on the test sets (*test-clean* and *test-other*).

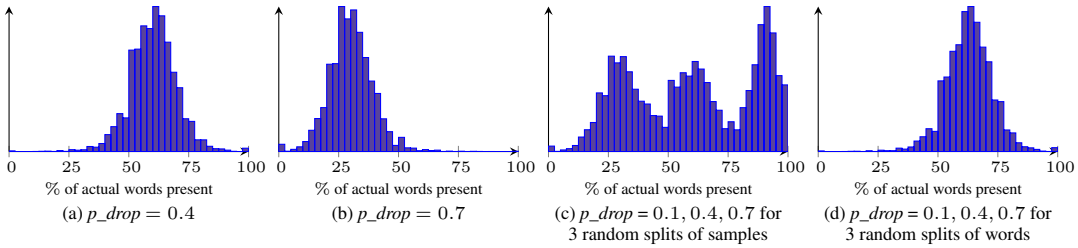
We use the top 50K words (sorted by occurance frequency) from the official language model (LM) training data provided with LibriSpeech as the alphabet for training the models. These 50K words cover 99.04% of all the word occurances in the LM training data. The model architectures and the STC loss are implemented with the ASR application [35] of the flashlight[3] machine-learning framework and the C++ API of GTN.

4.1.1 Label Generation for Weakly Supervised Setup

As LibriSpeech dataset consists of fully labeled data, we need to drop the labels manually to simulate the weakly supervised setup. We quantify this using a parameter $p_drop \in [0, 1]$, which denotes the

[3] `https://github.com/flashlight/flashlight`

probability of dropping a word from the transcript label. It can be seen that higher value of p_drop corresponds to higher number of missing words in the transcript and vice-versa. Figures 7a-b show the histogram of percentage of the labeled words present in each training sample in LibriSpeech for different values of p_drop which closely follows a normal distribution. Also, from the definition of p_drop, it naturally follows that $p_drop = 0$ corresponds to supervised training setup while $p_drop = 1$ corresponds to unsupervised training.



(a) $p_drop = 0.4$ (b) $p_drop = 0.7$ (c) $p_drop = 0.1, 0.4, 0.7$ for 3 random splits of samples (d) $p_drop = 0.1, 0.4, 0.7$ for 3 random splits of words

Figure 7: Histograms of percentage of transcript words retained in each training sample in LibriSpeech for different ways of generating weakly supervised data using p_drop.

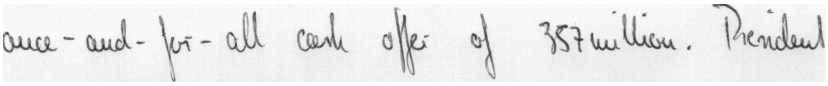
To make sure our method is robust to different types of partial labels, we also test our method on two other ways of generating the data :
1. Randomly split the training set into p parts and use a different p_drop for each split (Figure 7c)
2. Randomly split the set of all words in training vocab into p parts, use a different p_drop for each split (Figure 7d)

For all the new partially labeled datasets created, we prune samples which have empty transcriptions before using them for training.

4.1.2 Pseudo-labeling

To further achieve better word error rate (WER), we generate pseudo labels (PLs) on the training set using the word-based model trained with STC and use these PLs to train a letter-based model, via a regular CTC approach. The PLs are generated using rescoring as described in Section B.1.2 on the training set using the hyperparameters optimized on validation set.

4.2 Handwriting Recognition (HWR)



True Label : once-and-for-all|cash|offer|of|357million|.|President
***pDrop* = 0.1 :** oc-andforallcah|offer|o|357million|.|Preident
***pDrop* = 0.3 :** once-nd-fralcasr|of|357milon|Presidnt
***pDrop* = 0.7 :** nc-llaf|7mlioPreit

Figure 8: A training example from IAM along with weakly supervised labels generated for different p_drop values.

We test our approach on IAM Handwriting database [26], which is a widely used benchmark for handwriting recognition. The dataset contains 78 different characters and a white-space symbol, |. We use Aachen data splits[4] to divide the dataset into three subsets: 6,482 lines for training, 976 lines for validation and 2,915 lines for testing.

To create weakly supervised labels, we use the same methodology using p_drop as described in 4.1.1. However, we drop the labels at character-level instead of word-level. This is because it is efficient to

[4] https://www.openslr.org/56

Table 1: WER comparison on LibriSpeech test sets for various partial label generation settings. For each setting, we report the results for training with CTC (control experiment), STC on the partially labeled data and training with CTC using PLs generated from the trained STC model. Whenever possible, for each of these experiments, we report WER with a greedy decoding and no LM (top row), with 5-gram LM beam-search decoding (middle row) and with additional second-pass rescoring by Transformer LM (below row). We also include state-of-the-art results on LibriSpeech using wordpieces, letters and words as output tokens in the top section of the table.

WEAK LABEL GEN. STRATEGY	METHOD	CRITERION	MODEL STRIDE/ PARAMETERS	OUTPUT TOKENS	LM	TEST WER CLEAN/OTHER
	CONFORMER [14]	RNN-T	4/119M	1K WORD PIECES	- LSTM	2.1/4.3 1.9/3.9
	TRANSFORMER [36]	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	2.7/6.1 2.4/5.3 2.1/4.5
	TRANSFORMER [8]	SEQ2SEQ	8/$\sim$ 300M	WORDS	- WORD 4-GRAM	2.9/6.7 3.0/6.3
P_DROP = 0.1	TRANSFORMER	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	6.2/11.0 4.9/8.8 4.5/8.1
	TRANSFORMER	STC	8/70M	WORDS	- WORD 5-GRAM TR. RESCORING	4.1/9.0 4.1/8.7 3.3/6.9
	+ PSEUDO LABELING	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	3.1/6.3 2.9/5.6 2.7/5.2
P_DROP = 0.4	TRANSFORMER	CTC	3/270M	LETTERS	-	48.2/56.9
	TRANSFORMER	STC	8/70M	WORDS	- WORD 5-GRAM TR. RESCORING	4.5/10.2 4.6/9.9 3.6/7.7
	+ PSEUDO LABELING	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	3.3/6.6 3.1/5.9 2.9/5.4
P_DROP = 0.7	TRANSFORMER	CTC	3/270M	LETTERS	-	100/100
	TRANSFORMER	STC	8/70M	WORDS	- WORD 5-GRAM TR. RESCORING	6.9/15.1 7.0/14.8 5.1/11.1
	+ PSEUDO LABELING	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	3.9/7.9 3.5/7.0 3.1/6.2
SPLIT ALL SAMPLES INTO 3 PARTS RANDOMLY; ASSIGN P_DROP= 0.1.0.4,0.7 FOR THE SPLITS	TRANSFORMER	CTC	3/270M	LETTERS	-	58.6/63.9
	TRANSFORMER	STC	8/70M	WORDS	- WORD 5-GRAM TR. RESCORING	4.9/11.1 5.1/11.0 3.7/8.2
	+ PSEUDO LABELING	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	3.3/6.7 3.0/5.9 2.8/5.4
SPLIT ALL WORDS INTO 3 PARTS RANDOMLY; ASSIGN P_DROP= 0.1.0.4,0.7 FOR THE SPLITS	TRANSFORMER	CTC	3/270M	LETTERS	-	45.6/49.2
	TRANSFORMER	STC	8/70M	WORDS	- WORD 5-GRAM TR. RESCORING	5.8/13.0 6.6/12.2 4.1/8.7
	+ PSEUDO LABELING	CTC	3/270M	LETTERS	- WORD 5-GRAM TR. RESCORING	3.4/6.9 3.1/5.9 2.9/5.6

express the STC label graph with "$\langle * \rangle$" tokens when we use the same type of targets which are being dropped. In ASR, the targets are dropped at word level, hence we choose words and in HWR we use character targets. Figure 8 shows a training example from IAM along with weakly supervised labels generated for different p_drop values.

Additional implementation and training details (models, tokens, optimization, other hyperparameters and settings) for the ASR and HWR experiments are in the Appendix B.

P_DROP	METHOD	DEV CER	TEST CER
	TRANSFORMER [21]		4.7
	CNN+CTC [43]		4.9
	LSTM W/ATTN [28]	3.2	5.5
	CNN + CTC (OUR BASELINE)	3.7	5.4
0.1	CNN + CTC	5.4	7.7
	CNN + STC	5.0	7.2
0.3	CNN + CTC	8.7	11.6
	CNN + STC	5.6	8.1
0.5	CNN + CTC	48.2	53.6
	CNN + STC	10.0	13.5
0.7	CNN + CTC	77.3	78.5
	CNN + STC	22.7	26.7

Table 2: CER performance comparison of STC and CTC with greedy decoding on IAM dataset for different p_drop values. A comparison with selected works using the "same" training set is shown at the top.

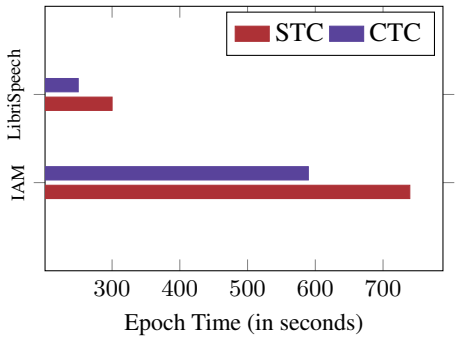


Figure 9: Epoch time performance of STC, CTC on LibriSpeech and IAM datasets for the models we have used.

5 Results

5.1 Automatic Speech Recognition

In Table 1, we compare the WER performance of STC models for a fixed value of $p_drop = 0.1, 0.4, 0.7$ and also when p_drop is dependent on the sample or the word. We also compare these results with the models trained directly with CTC on the partial labels. We can see that STC performs better than CTC in all the settings. We observe that performing beam search decoding and an additional second-pass rescoring with a Transformer LM can reduce the WER of the models significantly.

We also see that pseudo labeling the training set using STC model trained on partial labels and then training a letter-based CTC model can further improve WER performance. We can get competitive results compared with the fully supervised results.

5.2 Handwriting Recognition

In Table 2, we compare the CER performance of CTC and STC for various values of p_drop with greedy decoding. We can see that using STC clearly gives better performance over CTC trained models for $p_drop = 0.1, 0.3, 0.5, 0.7$. The performance gap between the supervised baseline and models trained with partial labels using STC is larger compared to ASR experiments. This is because we did not use LM decoding or pseudo-labeling steps which should help in reducing CER.

Hyperparameters used for token insertion penalty, λ to train STC models for ASR, HWR experiments are in Appendix D.

5.3 Runtime Performance of STC

In Figure 9, we compare the epoch time of STC, CTC models on LibriSpeech and IAM datasets. For a fair comparison, we use the same model that is used to report STC results for each dataset. The experiments on LibriSpeech, IAM are run on 32, 8 Nvidia 32GB V100 GPUs and uses C++, Python APIs of GTN respectively.

We were able to run an epoch on LibriSpeech in about 300 seconds for the STC model and in about 250 seconds for the CTC model. Optimizing the data transfer between CPU and GPU by only moving the tokens which are present in the current training sample (Section 3.3.3) is a crucial step in achieving good performance. Without this optimization, it would take about 5400 seconds per epoch on LibriSpeech as the alphabet size, $|\mathcal{A}|$ of 50000 is very high.

For IAM experiments, we did not perform the data transfer optimization for STC as $|\mathcal{A}|$ is only 79. It can be see that the epoch time of STC is about 25% more compared to CTC.

6 Conclusion

In a variety of sequence labeling tasks (ASR, HWR), we show that STC enables training models with partially labeled data and can give strong performance. Weakly supervised data can be collected in semi-automatic ways and can alleviate labeling of full training data, thus reducing production costs. We also show that using WFSTs with a differentiable framework is flexible and powerful tool for researchers to solve a completely new set of problems, like the example we have shown in this paper. STC requires additional hyperparameter tuning and these vary by p_drop. The label graph can be easily modified to include the cases where the missing label's position or the number of missing tokens is known in advance. Also, STC can be specialized to allow a particular range of missing tokens (e.g. between 2 and 4 characters). A direct potential extension is to study the application of STC to noisy labels.

References

[1] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artif. Intell.*, 201:81–105, 2013.

[2] Yannis Assael, Thea Sommerschield, and Jonathan Prag. Restoring ancient text using deep learning: a case study on Greek epigraphy. In *Empirical Methods in Natural Language Processing*, pages 6369–6376, 2019.

[3] Yannis Assael, Thea Sommerschield, Brendan Shillingford, Mahyar Bordbar, John Pavlopoulos, Marita Chatzipanagiotou, Ion Androutsopoulos, Jonathan Prag, and Nando de Freitas. Restoring and attributing ancient texts using deep neural networks. *Nature*, 2022. doi: 10.1038/s41586-022-04448-z.

[4] Peter Bell and Steve Renals. A system for automatic alignment of broadcast media captions using weighted finite-state transducers. pages 675–680, 12 2015. doi: 10.1109/ASRU.2015.7404861.

[5] Xingyu Cai, Jiahong Yuan, Yuchen Bian, Guangxu Xun, Jiaji Huang, and Kenneth Church. W-CTC: a connectionist temporal classification loss with wild cards. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0RqDp8FCW5Z`.

[6] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

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

[8] Ronan Collobert, Awni Hannun, and Gabriel Synnaeve. Word-level speech recognition with a letter to word encoder. In *International Conference on Machine Learning*, pages 2100–2110. PMLR, 2020.

[9] Timothee Cour, Ben Sapp, and Ben Taskar. Learning from partial labels. *J. Mach. Learn. Res.*, 12(null): 1501–1536, jul 2011. ISSN 1532-4435.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL `http://jmlr.org/papers/v12/duchi11a.html`.

[11] Adrien Dufraux, Emmanuel Vincent, Awni Hannun, Armelle Brun, and Matthijs Douze. Lead2gold: Towards exploiting the full potential of noisy transcriptions for speech recognition. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 78–85. IEEE, 2019.

[12] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2019.

[13] 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.

[14] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Proc. Interspeech 2020*, pages 5036–5040, 2020. doi: 10.21437/Interspeech.2020-3015.

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

[16] Timothy J. Hazen. Automatic alignment and error correction of human generated transcripts for long speech recordings. In *INTERSPEECH*, 2006.

[17] De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Connectionist temporal modeling for weakly supervised action labeling. In *European Conference on Computer Vision*, pages 137–153. Springer, 2016.

[18] Rong Jin and Zoubin Ghahramani. Learning with multiple labels. In *NIPS*, 2002.

[19] k2-fsa. `https://github.com/k2-fsa/k2`, 2021.

[20] Mohammed Waleed Kadous. Temporal classification: extending the classification paradigm to multivariate time series. 2002.

[21] Lei Kang, Pau Riba, Marçal Rusiñol, Alicia Fornés, and Mauricio Villegas. Pay attention to what you read: Non-recurrent handwritten text-line recognition, 2020.

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

[23] Aleksandr Laptev, Somshubra Majumdar, and Boris Ginsburg. CTC variations through new wfst topologies. *arXiv preprint arXiv:2110.03098*, 2021.

[24] Tatiana Likhomanenko, Qiantong Xu, Vineel Pratap, Paden Tomasello, Jacob Kahn, Gilad Avidov, Ronan Collobert, and Gabriel Synnaeve. Rethinking Evaluation in ASR: Are Our Models Robust Enough? In *Proc. Interspeech 2021*, pages 311–315, 2021. doi: 10.21437/Interspeech.2021-1758.

[25] Liping Liu and Thomas Dietterich. Learnability of the superset label learning problem. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1629–1637, Bejing, China, 22–24 Jun 2014. PMLR.

[26] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.

[27] Yajie Miao, Mohammad Gowayyed, and Florian Metze. Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 167–174. IEEE, 2015.

[28] Johannes Michael, Roger Labahn, Tobias Grüning, and Jochen Zöllner. Evaluating sequence-to-sequence models for handwritten text recognition. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1286–1293. IEEE, 2019.

[29] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

[30] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In *Springer handbook of speech processing*, pages 559–584. Springer, 2008.

[31] Pedro Moreno and Christopher Alberti. A factor automaton approach for the forced alignment of long speech recordings. pages 4869–4872, 04 2009. doi: 10.1109/ICASSP.2009.4960722.

[32] Niko Moritz, Takaaki Hori, and Jonathan Le Roux. Semi-supervised speech recognition via graph-based temporal classification. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6548–6552. IEEE, 2021.

[33] 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.

[34] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *Interspeech 2019*, Sep 2019. doi: 10.21437/interspeech.2019-2680. URL `http://dx.doi.org/10.21437/Interspeech.2019-2680`.

[35] Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve, Vitaliy Liptchinsky, and Ronan Collobert. Wav2letter++: A fast open-source speech recognition system. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019. doi: 10.1109/icassp.2019.8683535. URL `http://dx.doi.org/10.1109/ICASSP.2019.8683535`.

[36] Vineel Pratap, Qiantong Xu, Tatiana Likhomanenko, Gabriel Synnaeve, and Ronan Collobert. Word order does not matter for speech recognition. *arXiv preprint arXiv:2110.05994*, 2021.

[37] Gabriel Synnaeve, Qiantong Xu, Jacob Kahn, Tatiana Likhomanenko, Edouard Grave, Vineel Pratap, Anuroop Sriram, Vitaliy Liptchinsky, and Ronan Collobert. End-to-end asr: from supervised to semi-supervised learning with modern architectures, 2020.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Enrique Vidal, Franck Thollard, Colin De La Higuera, Francisco Casacuberta, and Rafael C Carrasco. Probabilistic finite-state machines-part i and ii. *IEEE transactions on pattern analysis and machine intelligence*, 27(7), 2005.

[40] Curtis Wigington, Brian Price, and Scott Cohen. Multi-label connectionist temporal classification. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 979–986. IEEE, 2019.

[41] 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.

[42] Mohamed Yousef and Tom E. Bishop. Origaminet: Weakly-supervised, segmentation-free, one-step, full page textrecognition by learning to unfold. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[43] Mohamed Yousef, Khaled F Hussain, and Usama S Mohammed. Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. *Pattern Recognition*, 108:107482, 2020.

[44] Zhi-Hua Zhou, Min-Ling Zhang, Sheng-Jun Huang, and Yu-Feng Li. Multi-instance multi-label learning. *Artificial Intelligence*, 176(1):2291–2320, 2012.

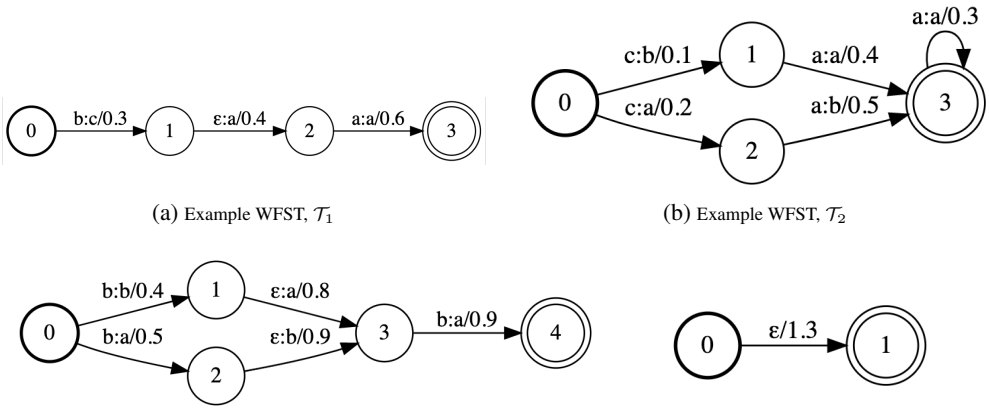
Checklist

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? [Yes]
   (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 Appendix: Weighted Finite-State Transducers

A weighted finite-state transducer (WFST) is a generalization of a finite-state automaton (FSA) [29, 30, 39] where each transition has an input label from an alphabet Σ an output label from an alphabet Δ and scalar weight w. Figure 10a shows an example WFST with nodes representing states and arcs representing transitions. A path from an initial to a final state encodes a mapping from an input sequence $\mathbf{i} \in \Sigma^*$ to an output sequence $\mathbf{o} \in \Delta^*$ and a corresponding score. While WFST operations can be performed in any semiring, in this work we only use the log semiring.



(a) Example WFST, $\mathcal{T}_1$ (b) Example WFST, $\mathcal{T}_2$

(c) Composition, $\mathcal{T}_1 \circ \mathcal{T}_2$ (d) Forward Score, $Fwd(\mathcal{T}_1)$

Figure 10: Examples of WFSTs and their operations (in log semiring). The arc label "p:r/w" denotes an input label p, an output label r and weight w. Special symbol, ϵ allows to make a transition without consuming an input label or without producing an output label. Start states are represented by bold circles and final states by concentric circles.

In this work we use composition and shortest distance operations on WFSTs. The composition operation combines WFSTs from different modalities. Given two WFSTs $\mathcal{T}_1$ and $\mathcal{T}_2$, if $\mathcal{T}_1$ transduces $\mathbf{a}$ to $\mathbf{b}$ with weight w_1 and $\mathcal{T}_2$ transduces $\mathbf{b}$ to $\mathbf{c}$ with weight w_2, then their composition $\mathcal{T}_1 \circ \mathcal{T}_2$ transduces $\mathbf{a}$ to $\mathbf{c}$ with weight $w_1 + w_2$. The forward score operation is the shortest distance from a start state to a final state in the log semiring. Given a transducer $\mathcal{T}_1$, the forward score is the log-sum-exp of the scores of all paths from any start state to any final state. The example output graphs from composition and forward score operations are shown in Figure 10c and Figure 10d respectively.

A.1 Autograd with WFSTs

Most operations on WFST are differentiable with respect to the arc weights of the input graphs. This allows WFSTs to be used dynamically to train neural networks. Frameworks like GTN [15] and k2 [19] implement automatic differentiation with WFSTs. For the purpose of this work, we use the GTN framework.

B Appendix: Additional Implementation Details

B.1 Automatic Speech Recognition

We keep the original 16kHz sampling rate and compute log-mel filterbanks with 80 coefficients for a 25ms sliding window, strided by 10ms. All features are normalized to have zero mean and unit variance per input sequence before feeding them into the acoustic model. We use SpecAugment [34] as the data augmentation method for all ASR experiments.

B.1.1 Model architecture

The acoustic model (AM) architecture is composed of a convolutional frontend (1-D convolution with kernel-width 15 and stride 8 followed by GLU activation) followed by 36 $\times$ 4- heads Transformer blocks [38] with relative positional embedding. The self-attention dimension is 384 and the feed-forward network (FFN) dimension is 3072 in each Transformer block. The output of the final Transformer block if followed by a Linear layer with output dimension of 580 and a letter-to-word encoder (see Appendix C) to the output classes (word vocabulary + *blank*). For all Transformer layers, we use dropout on the self-attention and on the FFN, and layer drop [12], dropping entire layers at the FFN level. We apply LogSoftmax operation on each output frame to produce a probability distribution (in log-space) over output classes. The model consists of 70 million parameters and we use 32 $\times$ Nvidia 32GB V100 GPUs for training.

B.1.2 Decoding

Beam-Search decoder: In our experiments, we use a beam-search decoder following [8] which leverages a n-gram language model to decrease the word error rate. The beam-search decoder outputs a transcription $\hat{\mathbf{y}}$ that maximizes the following objective

$$\log P(\hat{\mathbf{y}}|\mathbf{x}) + \alpha \log P_{LM}(\hat{\mathbf{y}}) + \beta|\hat{\mathbf{y}}| \tag{5}$$

where $\log P_{LM}(.)$ is the log-likelihood of the language model, α is the weight of the language model, β is a word insertion weight and $|\hat{\mathbf{y}}|$ is the transcription length in words. The hyperparameters α and β are optimized on the validation set.

We use 5-gram LM trained on the official LM training data provided with LibriSpeech for beam-search decoding.

Rescoring: To further decrease the word error rate, we use the top-K hypothesis from beam search decoding and perform rescoring [37] with a Transformer LM to reorder the hypotheses according to the following score:

$$\log P(\hat{\mathbf{y}}|\mathbf{x}) + \alpha \log P_{LM'}(\hat{\mathbf{y}}) + \beta|\hat{\mathbf{y}}| \tag{6}$$

where $\log P_{LM'}(.)$ is the log-likelihood of the Transformer LM, α, β are the hyperparameters as described above and $|\hat{\mathbf{y}}|$ is the transcription length in characters. We use the pre-trained Transformer LM on LibriSpeech from [24] which has a perplexity of 50 on dev-other transcripts. In this work, top 512 hypothesis from beam search decoding are used as candidates for rescoring.

B.1.3 Optimizer

For training the word-based STC models, we use the Adagrad optimizer [10] with a learning rate warmup scheme that increases linearly from 0 to 0.02 in 16000 training steps for all the experiments. We halve the learning rate initially after 400 epochs and every 200 epochs after that. All models are trained with dynamic batching with a batch size of 240 audio sec per GPU. We use SpecAugment [34] with two frequency masks, and ten time masks with maximum time mask ratio of p = 0.1, the maximum frequency bands masked by one frequency mask is 30, and the maximum frames masked by the time mask is 50; time warping is not used. SpecAugment is turned on only after 32000 training steps are finished. Dropout and layer dropout values of 0.05 is used in the AM.

B.1.4 Pseudolabeling

For training the letter-based CTC models, we use the same Transformer-based encoder consisting of 270M parameters from [37] for the AM: the encoder of our acoustic models is composed of a convolutional frontend (1-D convolution with kernel-width 7 and stride 3 followed by GLU activation) followed by 36 4-heads Transformer blocks [38]. The self-attention dimension is 768 and the feed-forward network (FFN) dimension is 3072 in each Transformer block. We uses 64 $\times$ Nvidia 32GB V100 GPUs for training. The output of the encoder is followed by a linear layer to the output classes. We use dropout after the convolution layer. For all Transformer layers, we use dropout of 0.2 on the self-attention and on the FFN, and layer drop [12] value of 0.2. Token set for all acoustic models consists of 26 English alphabet letters, augmented with the apostrophe and a word boundary token. To speed up training, we also use mixed-precision training. We use the same SpecAugment and learning rate schemes as discussed above for STC models. We decay learning rate by a factor of 2 each time the WER reaches a plateau on the validation sets.

B.2 Handwriting Recognition

For training the handwriting recognition models, we have used the open-source code[5] based on a prior work from [42] and adapted it for our use case. It uses depthwise separable convolutions as the main computational

[5] `https://github.com/IntuitionMachines/OrigamiNet`

block and the model consists of about 39 million parameters. We use 8 $\times$ Nvidia V100 32GB GPUs for training the models.

It should also be noted that we were unable to find an open source implementation of Yousef et al. [43] and our baseline is our best attempt to reproduce their results.

We use Adam Optimizer [22] with an initial learning rate of 0.02 and run training using a batchsize of 8 per GPU for all the experiments. All image are scaled to a maximum width, height of 600, 32 pixels respectively before feeding to the neural network. We use exponential learning rate decay schedule with a gamma factor of $10^{-1/90000} \approx 0.99997$. We use random projection transformations as the augmentation method while training.

C Appendix: A simple letter to word encoder

Since, we are working on direct-to-word models for ASR, a major hurdle would be to transcribe words not found in the training vocabulary. Recently, [8] propose an end-to-end model which outputs words directly, yet is still able to dynamically modify the lexicon with words not seen at training time. They use a convnet based letter to word embedding encoder which is jointly trained with the acoustic model and is able to accommodate new words at testing time.

In this work, we propose a simpler system for letter to word encoder which can be embedded into the acoustic model and thus avoiding the needs for a separate network. Consider $\mathcal{A}_L$, $\mathcal{A}_W$ are the alphabets for letters, words respectively and l_{max} is the maximum length of a word. We assume $\mathcal{A}_L$ also contains two special letters - c_{blank} which produces *blank* token required for CTC/STC models and c_{pad} which is used to pad all words, *blank* token to the same maximum word length l_{max}. We let the acoustic model produce a $(|\mathcal{A}_L| \times l_{max})$ dimensional vector for each time frame.

$$\underbrace{\begin{bmatrix} 1&0&0&0&0&0&0&0&0&1&0&0&0&0&1 \\ 0&0&1&0&0&1&0&0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0&0&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0&0&1&0&0&0&0&1 \end{bmatrix}}_{\boldsymbol{E}} \times \begin{bmatrix} 0.9 \\ -0.4 \\ 0.2 \\ 1.8 \\ -0.1 \\ -0.4 \\ 1.2 \\ -1.3 \\ 0.1 \\ 1.2 \\ 2.1 \\ -0.8 \\ -1.4 \\ 0 \\ 0.1 \end{bmatrix} \begin{matrix} \text{a} \\ \text{b} \\ \text{c} \\ c_{blank} \\ c_{pad} \\ \text{a} \\ \text{b} \\ \text{c} \\ c_{blank} \\ c_{pad} \\ \text{a} \\ \text{b} \\ \text{c} \\ c_{blank} \\ c_{pad} \end{matrix} = \begin{bmatrix} 2.3 \\ -1.0 \\ -0.3 \\ 3.1 \end{bmatrix} \begin{matrix} \text{a} \\ \text{cab} \\ \text{ca} \\ blank \end{matrix}$$

Figure 11: Letter to word encoder in action. Matrix $\boldsymbol{E}$ converts the letter scores over each timeframe to a score over words and *blank*. $\mathcal{A}_L = \{a, b, c, c_{blank}, c_{pad}\}$, $l_{max} = 3$, $\mathcal{A}_W = \{\text{a}, \text{cab}, \text{ca}\}$.

Since CTC/STC expect scores for $\mathcal{A}_W$ and *blank* for each time frame, we carefully construct a matrix $\boldsymbol{E}$ of 1s and 0s which converts a vector of size $(|\mathcal{A}_L| \times l_{max})$ to $(|\mathcal{A}_W| + 1)$ as shown in Figure 11 . Since we are using c_{pad} token for padding, we can always assume every words and *blank* token is a sequence of l_{max} letters. Each row in $\boldsymbol{E}$ is constructed by concatenating the one-hot representation of each letter (including c_{pad}) and it produces the score for a word or *blank*. We use $\mathcal{A}_L = \{a - z, ', c_{blank}, c_{pad}\}$ and $l_{max} = 20$ for all our experiments on LibriSpeech using words as the output tokens.

Table 3: WER comparison on LibriSpeech dev and test sets

METHOD	CRITERION	MODEL STRIDE/ PARAMETERS	OUTPUT TOKENS	LM	DEV WER CLEAN/OTHER	TEST WER CLEAN/OTHER
TRANSFORMER [8]	CTC	8/$\sim$ 300M	WORDS	-	2.9/7.5	3.2/7.5
				WORDS 4-GRAM	2.6/6.6	2.9/6.7
	SEQ2SEQ	8/$\sim$ 300M	WORDS	-	2.7/6.5	2.9/6.7
				WORD 4-GRAM	2.5/6.0	3.0/6.3
TRANSFORMER (USES OUR SIMPLE LETTER TO WORD ENCODER)	CTC	8/70M	WORDS	-	2.3/6.8	2.8/7.0
				WORD 5-GRAM	2.2/6.3	2.7/6.5

From Table 3, we can see that we are able to match the performance of the word based models from [8] using a much smaller acoustic model. Also, our system is simpler as we do not have a separate network for letter-to-word encoder.

D Appendix: Hyperparameters for token insertion penalty, λ

In Table 4, we report the hyperparameters used for training STC models on LibriSpeech (Table 1) and IAM (Table 2). The best performing values are found by running a grid search over the hyperparameters and measuring performance on the development set. It can be seen that with higher p_drop values, a higher token insertion penalty values are usually preferred.

Table 4: Best performing hyperparameters used for token insertion penalty, λ while training STC models for LibriSpeech (left) and IAM (right).

WEAK LABEL GEN. STRATEGY	p_0	p_{max}	$t_{1/2}$
P_DROP=0.1	0.1	0.3	8000
P_DROP=0.4	0.4	0.7	8000
P_DROP=0.7	0.5	0.9	8000
SPLIT ALL SAMPLES INTO 3 PARTS RANDOMLY; ASSIGN P_DROP=0.1,0.4,0.7 FOR THE SPLITS	0.3	0.6	8000
SPLIT ALL WORDS INTO 3 PARTS RANDOMLY; ASSIGN P_DROP=0.1,0.4,0.7 FOR THE SPLITS	0.5	0.7	8000

P_DROP	p_0	p_{max}	$t_{1/2}$
0.1	0.5	0.8	10000
0.3	0.5	0.9	10000
0.5	0.5	0.9	10000
0.7	0.7	0.9	10000

D.1 Sensitivity of the model performance when setting token insertion penalty, λ

In Table 5, we report the effect of hyper parameter choices on the model performance for IAM dataset for $p_drop = 0.1$ and $p_drop = 0.7$. We see that the model performance is more sensitive with respect to hyperparameter choices for $p_drop = 0.7$ compared to $p_drop = 0.1$.

Table 5: Effect of hyper parameter choices on the model performance for IAM dataset. $p_drop = 0.1$ (left) , $p_drop = 0.7$ (right).

$p_drop = 0.1$

p_0	p_{max}	$t_{1/2}$	DEV CER
0.5	0.8	10000	5.0
0.3	0.6	10000	5.0
0.5	0.9	10000	5.0
0.4	0.8	10000	5.1
0.6	0.8	10000	5.1
0.5	0.8	5000	5.2
0.2	0.5	10000	5.3
0.2	0.4	10000	5.3
0.2	0.4	30000	5.4
0.2	0.6	10000	5.4
0.1	0.3	10000	5.6
0.2	0.4	50000	5.6

$p_drop = 0.7$

p_0	p_{max}	$t_{1/2}$	DEV CER
0.7	0.9	10000	22.7
0.65	0.9	10000	23.6
0.65	0.95	10000	24.9
0.6	0.95	10000	26.1
0.5	0.9	10000	27.7
0.8	0.9	10000	30.3
0.5	0.95	10000	30.5
0.7	0.8	50000	32.9
0.6	0.9	10000	33.9
0.7	0.8	10000	34.6
0.7	0.95	10000	35.9
0.3	0.9	10000	41.0
0.5	0.9	10000	42.7
0.7	0.75	10000	46.7
0.8	0.95	10000	48.3