Efficient Sequence Transduction by Jointly Predicting Tokens and Durations

Hainan Xu [1] **Fei Jia** [1] **Somshubra Majumdar** [1] **He Huang** [1] **Shinji Watanabe** [2] **Boris Ginsburg** [1]

Abstract

This paper introduces a novel *Token-and-Duration Transducer* (TDT) architecture for sequence-to-sequence tasks. TDT extends conventional RNN-Transducer architectures by jointly predicting both a token and its duration, i.e. the number of input frames covered by the emitted token. This is achieved by using a joint network with two outputs which are independently normalized to generate distributions over tokens and durations. During inference, TDT models can skip input frames guided by the predicted duration output, which makes them significantly faster than conventional Transducers which process the encoder output frame by frame. TDT models achieve both better accuracy and significantly faster inference than conventional Transducers on different sequence transduction tasks. TDT models for Speech Recognition achieve better accuracy and up to 2.82X faster inference than conventional Transducers. TDT models for Speech Translation achieve an absolute gain of over 1 BLEU on the MUST-C test compared with conventional Transducers, and its inference is 2.27X faster. In Speech Intent Classification and Slot Filling tasks, TDT models improve the intent accuracy by up to over 1% (absolute) over conventional Transducers, while running up to 1.28X faster. Our implementation of the TDT model will be open-sourced with the NeMo (https://github.com/NVIDIA/NeMo) toolkit.

1. Introduction

Over the past years, automatic speech recognition (ASR) models have undergone shifts from conventional hybrid models (Jelinek, 1998; Woodland et al., 1994; Povey et al., 2011) to end-to-end ASR models, including attention-based encoder and decoder (AED) models (Chorowski et al., 2015; Chan et al., 2016), Connectionist Temporal Classification (CTC) (Graves et al., 2006), and Transducers (Graves, 2012). Those models are commonly used in academia and industry, and there exist open-source toolkits with efficient implementation for those methods, including ESPNet (Watanabe et al., 2018), NeMo (Kuchaiev et al., 2019), Espresso (Wang et al., 2019), SpeechBrain (Ravanelli et al., 2021) etc.

This paper focuses on Transducer models. There have been a significant number of works that improve different aspects of the original Transducer (Graves, 2012). For example, the original LSTM encoder of transducer models has been replaced with Transformers (Tian et al., 2019; Yeh et al., 2019; Zhang et al., 2020), Contextnet (Han et al., 2020) and Conformers (Gulati et al., 2020). Decoders for transducers are well-investigated as well, e.g. (Ghodsi et al., 2020) used stateless decoders instead LSTM decoders; (Shrivastava et al., 2021) proposed *Echo State Networks* and showed that a decoder with random parameters could perform as well as a well-trained decoder. The loss function of Transducers has also been an active research area. FastEmit (Yu et al., 2021) introduces biases in the gradient update of the transducer loss to reduce latency. Multi-blank Transducers (Xu et al., 2022) introduce a generalized Transducer architecture and loss function with *big blank* symbols that cover multiple frames of the input.

RNN-Ts have achieved impressive accuracy in speech tasks, but the auto-regressive decoding makes their inference computationally costly. To alleviate this issue, we propose a new Transducer architecture that jointly predicts a token and its duration. The predicted token duration can direct the model decoding algorithm to skip frames during inference. We call it a TDT (Token-and-Duration Transducer) model. The primary contributions of this paper are:

1. A novel Token-and-Duration Transducer (TDT) architecture that jointly predicts a token and its duration.
2. An extension of the forward-backward algorithm to derive the analytical solution of the gradients of the TDT model. We also derive gradients of pre-softmax logits for the token prediction inspired by *Transducer function-merging* (Li et al., 2019).
3. TDT models achieve better accuracy and significant

[1]NVIDIA, USA [2]Carnegie Mellon University, PA, USA. Correspondence to: Hainan Xu <hainanx@nvidia.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

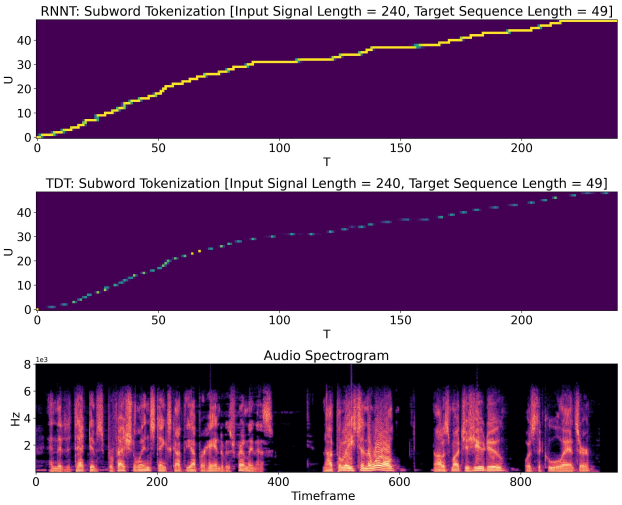

Figure 1. From top to bottom: alignments generated with conventional RNNT, TDT models with config [0-8], and the corresponding spectrogram. Each unit in the T axis of the alignment corresponds to 4 frames in the spectrogram due to subsampling. Note, TDT model learns to skip frames. Long skips are not frequently used in the audio where speech is present, but for the 4 relatively silent segments in the audio, where conventional RNN-T's alignment shows mostly horizontal lines, the TDT model uses long durations to skip the majority of frames.

inference speed-up compared to original RNN-Ts for 3 different tasks – speech recognition, speech translation, and spoken language understanding.

4. TDT-based ASR models are more robust to noise than conventional RNN-Ts models, and they don't suffer from the performance degradation for speech corresponding to the text with repeated tokens.

Our TDT model implementation will be open-sourced with NVIDIA's NeMo [1] toolkit.

2. Background: Transducers

An RNN-Transducer[2] (Graves, 2012) consists of an encoder, a decoder (or a prediction network), and a joint network (or a joiner). The encoder and decoder extract higher-level representations of the acoustic and text and feed the output to the joint network, which generates a probability distribution over the vocabulary. The vocabulary includes a special blank symbol Ø; a text sequence could be augmented by adding an arbitrary number of blanks between any adjacent tokens. During training, we maximize the log-probability $\log P_{\text{RNNT}}(y|x)$ for an audio utterance x with corresponding text y, which requires summing over all possible ways to augment the text sequence to match the audio:

$$\begin{aligned}\mathcal{L}_{\text{RNNT}}(y|x) &= \log P_{\text{RNNT}}(y|x) \\ &= \log \sum_{\pi:B^{-1}(\pi)=y} P_{\text{frame-level}}(\pi|x),\end{aligned} \tag{1}$$

where π represents an augmented sequence (including Ø), $B(.)$ is the operation to augment a sequence by adding blanks, and B^{-1} is the inverse of the operation B, which removes all the blanks in the sequence.

Computing $P_{\text{RNNT}}(y|x)$ using its definition is intractable since it needs to sum over exponentially many possible augmented sequences. In practice, the probability can be efficiently computed with the forward variables $\alpha(t,u)$ or backward variables $\beta(t,u)$, which are calculated recursively:

$$\begin{aligned}\alpha(t,u) &= \alpha(t-1,u)P(\text{Ø}|t-1,u) \\ &\quad + \alpha(t,u-1)P(y_u|t,u-1)\end{aligned} \tag{2}$$

$$\begin{aligned}\beta(t,u) &= \beta(t+1,u)P(\text{Ø}|t,u) \\ &\quad + \beta(t,u+1)P(y_{u+1}|t,u).\end{aligned} \tag{3}$$

with recursion base conditions $\alpha(1,0)=1$ and $\beta(T,U) = P(\text{Ø}|T,U)$. In order to make this recursion well-defined, we require that both $\alpha(t,u)$ and $\beta(t,u)$ are zero outside domain $1 \le t \le T$ and $0 \le u \le U$. With those quantities defined, $P_{\text{RNNT}}(y|x)$ could be computed with either the α or β efficiently:

$$P_{\text{RNNT}}(y|x) = \alpha(T,U)P(\text{Ø}|T,U) = \beta(1,0).$$

Then we could compute the loss function as,

$$\mathcal{L}_{\text{RNNT}}(y|x) = \log P_{\text{RNNT}}(y|x). \tag{4}$$

3. Token-and-Duration Transducers

A Token-and-Duration Transducer (TDT) differs from conventional transducers in that it predicts the token duration of the current emission. Namely, the TDT joiner generates two sets of output, one for the output token, and the other for the duration of the token (see Fig. 2). [3] Let us first

[1] https://github.com/NVIDIA/NeMo/. Pull request https://github.com/NVIDIA/NeMo/pull/6536

[2] When originally proposed, an RNN-Transducer uses a recurrent network as the encoder hence the name *RNN-T*; nowadays Transducers usually adopt more sophisticated networks involving self-attention in their encoder. In this paper, we use the words *RNN-T* and *Transducer* interchangeably to represent any encoder-decoder-joiner model that uses the Transducer loss.

[3] In our implementation, the two outputs are disjoint sub-vectors of the joiner output. For example, let's take vocabulary size (voc) of 1024 (including Ø) that supports durations {0,1,2,3,4} (a total of 5 durations). The last layer of the joiner maps the hidden activation to a tensor `joiner_out` of size 1024 + 5 = 1029. Then `joiner_out[:1024]` and `joiner_out[1024:]` are independently normalized to generate the two sets of distributions.

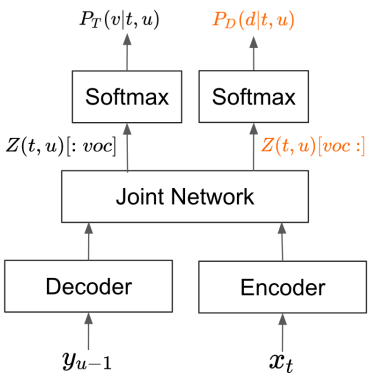

Figure 2. Architecture of a TDT model, which contains an encoder, a decoder, and a joint network. The TDT joint network emits two sets of output, one for the output token $Z(t,u)$[:voc], and the other for the duration of the token $Z(t,u)$[voc:]. The two distributions are jointly trained during model training.

define a joint probability $P(v, d|t, u)$ as the probability of generating token v (v could either be a text token or $\emptyset$), with duration d at location (t, u). We assume that token and durations are conditionally independent:

$$P(v, d|t, u) = P_T(v|t, u)P_D(d|t, u) \tag{5}$$

where $P_T(.)$ and $P_D(.)$ correspond to the token distribution and duration distribution, respectively. Next, we can compute the forward variables $\alpha(t, u)$:

$$\begin{aligned}\alpha(t, u) = &\sum_{d\in\mathcal{D}\setminus\{0\}} \alpha(t-d, u)P(\emptyset, d|t-d, u) \\ &+ \sum_{d\in\mathcal{D}} \alpha(t-d, u-1)P(y_u, d|t-d, u-1)\end{aligned} \tag{6}$$

with the same base condition $\alpha(1, 0) = 1$ as that of the conventional Transducer. Note, this Equation differs from 2 in that, for both non-blank and blank emissions, we need to sum over durations in $\mathcal{D}$ to consider all possible contributions from states that can reach (t, u), weighted by the corresponding duration probabilities.[4] Readers are encouraged to compare those Equations with the transition arcs in Figure 3 to see the connections. The total output probability

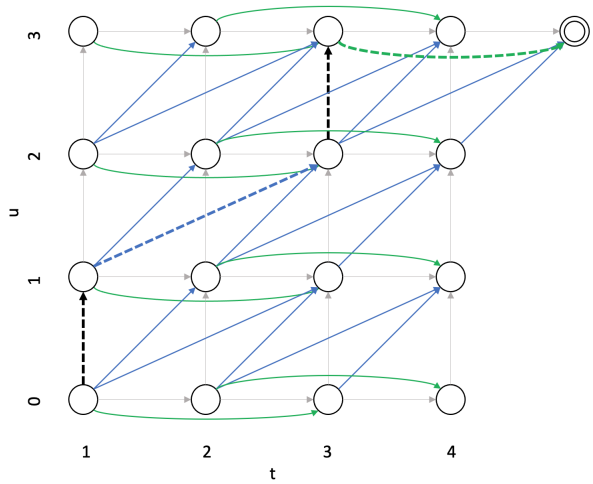

Figure 3. Output probability lattice of TDT model with supported durations {0,1,2}. We follow the convention in (Graves, 2012) making t start with 1 and u with 0. The probability of observing the first u output labels in the first t frames is represented by node (t, u). Dotted arrows constitute a complete path in the lattice.

$P_{\text{TDT}}(y|x)$ is computed through α at the terminal node: [5]

$$P_{\text{TDT}}(y|x) = \alpha(T + 1, U) \tag{7}$$

The backward variables (β) are computed as,

$$\begin{aligned}\beta(t, u) = &\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d, u)P(\emptyset, d|t, u) \\ &+ \sum_{d\in\mathcal{D}} \beta(t+d, u+1)P(y_{u+1}, d|t, u)\end{aligned} \tag{8}$$

with the base condition $\beta(T + 1, U) = 1$. The probability of the whole sequence is $P_{\text{TDT}}(y|x) = \beta(1, 0)$.

With those quantities defined, we define TDT loss as

$$\mathcal{L}_{\text{TDT}} = -\log P_{\text{TDT}}(y|x) \tag{9}$$

3.1. TDT Gradient Computation

We derive an analytical solution for the gradient of the TDT loss, since automatic differentiation for transducer loss is highly inefficient. [6] The gradient of the TDT loss $\mathcal{L}$ has two parts. The first part is the gradient with respect to the token probabilities $P_T(v|t, u)$:

$$\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P_T(v|t, u)} = -\frac{\alpha(t, u)b(v, t, u)}{P_{\text{TDT}}(y|x)} \tag{10}$$

[4]We disallow blank emission with duration 0, thus the sum is over $\mathcal{D} \setminus \{0\}$ for the blank emission. This makes the model not strictly probabilistic unless we renormalize the duration probabilities excluding duration = 0 for blank emissions computation. Although in practice we find that this does not matter, since duration=0 is in general rarely predicted according to Figure 4, and this design makes the derivation of gradients much easier.

[5]This equation, and the base condition of β are slightly different from the common definition used for conventional RNNT, although they are equivalent to the standard definition. For TDT though, this notation will make the boundary case much easier. In the recursion, we have $\beta(T + 1, u) = \alpha(T + 1, u) = 0, \forall u \neq U$.

[6]The detailed derivation for all gradients is in Appendix A.

where $\alpha(t, u)$ are defined in Equation 6 and $b(v, t, u)$ is define as:

$$b(v,t,u) = \begin{cases} \sum_{d \in \mathcal{D}} \beta(t+d, u+1) P_D(d|t,u), & v = y_{u+1} \\ \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d, u) P_D(d|t,u), & v = \emptyset \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Note, $b(v, t, u)$ can be interpreted as a weighted sum of β's that are reachable from (t, u), where the weights are from the duration probabilities.

The second part is the gradient with respect to the duration probabilities $P_D(d|t, u)$:

$$\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P_D(d|t,u)} = -\frac{\alpha(t,u) c(d,t,u)}{P(y|x)} \tag{12}$$

where

$$c(d,t,u) = \begin{cases} \beta(t, u+1) P_T(y_{u+1}|t,u), & d = 0 \\ \beta(t+d, u+1) P_T(y_{u+1}|t,u) & \\ \quad + \beta(t+d, u) P_T(\emptyset|t,u), & d > 0. \end{cases} \tag{13}$$

3.2. Gradient with Transducer Function Merging

The $P_T(v|t, u)$ terms in the Transducer loss are usually computed with a softmax function. Thus the gradients of the TDT loss have to go through the gradient of the softmax function to be passed to the previous layers, which could be costly. We use *Transducer function merging* proposed in (Li et al., 2019) to directly compute the gradient of the Transducer loss with respect to the *pre-softmax* logits ($h^v(t, u)$):

$$\frac{\partial \mathcal{L}_{\text{TDT}}(y|x)}{\partial h^v(t,u)} = \frac{P_T(v|t,u)\alpha(t,u)\Big[\beta(t,u) - b(v,t,u)\Big]}{P_{\text{TDT}}(y|x)} \tag{14}$$

where $b(v, t, u)$ is defined in Eq. 11. Note we apply function merging only to the *token logits*, not *duration logits* since the latter usually has very small dimensions, and the negligible efficiency improvements do not outweigh the added complexity in implementation.

3.3. Logits Under-normalization

We adopt the *logit under-normalization* method from (Xu et al., 2022) during the training of TDT models, in order to encourage longer durations. In our TDT implementations, we compute $P_T(v|t, u)$ in the log domain in order to have better numerical stability. The log probabilities $\log P_T(v|t, u)$ are computed from the logits $h^v(t, u)$ corresponding to token v:

$$\log P_T(v|t,u) = \text{log_softmax}_{v'}(h^{v'}(t,u)). \tag{15}$$

Algorithm 1 Greedy Inference of Conventional Transducer

1: **input:** acoustic input x
2: enc = encoder(x)
3: hyp = []
4: t = 0
5: **while** $t <$ len(enc) **do**
6: dec = decoder(hyp)
7: joined = joint(enc[t], dec)
8: idx = argmax(joined)
9: **if** token is not blank **then**
10: hyp.append(idx2token[idx])
11: **else**
12: t += 1
13: **end if**
14: **end while**
15: **return** hyp

The TDT model uses the pseudo "probability" $P'_T(v|t, u)$ in its forward and backward computation, which *under-normalize* the logits in the following way:

$$\log P'_T(v|t,u) = \text{log_softmax}_{v'}(h^{v'}(t,u)) - \sigma. \tag{16}$$

The under-normalization is only used in training, which encourages TDT models to prioritize emissions of any token (blank or non-blank) with longer durations. The gradients that incorporate the logit under-normalization method are shown in Eq. 17,

$$\frac{\partial \mathcal{L}_{\text{TDT}}(y|x)}{\partial h^v(t,u)} = \frac{P_T(v|t,u)\alpha(t,u)\Big[\beta(t,u) - \frac{b(v,t,u)}{\exp(\sigma)}\Big]}{\exp\Big[\mathcal{L}_{\text{TDT}}(y|x)\Big]} \tag{17}$$

where $b(v, t, u)$ are defined in Eq. 11.[7] Note that Eq. 17 is similar to Eq. 14, with the only difference being for TDT, the $b(v, t, u)$ term is scaled by $\frac{1}{\exp(\sigma)}$.

3.4. TDT Inference

We compare the inference algorithms of conventional Transducer models (Algorithm 1) and TDT models (Algorithm 2), which fully utilize the duration output. Note, that for TDT, an additional distribution over durations is computed from the joiner (line 9). This duration can increment t by more than one (line 13), compared with line 12 of conventional Transducer algorithm, where t could only be incremented by 1 at a time, and this only happens for blank emissions. This is the key place that makes TDT inference faster.

[7] $P_T(.)$ in Eq. 17, represents "real" probability, while the loss function $\mathcal{L}$ is computed with pseudo-probabilities.

Algorithm 2 Greedy Inference of TDT Models

input: acoustic input x
enc = encoder(x)
hyp = []
$t = 0$
while $t <$ len(enc) **do**
  dec = decoder(hyp)
  joined = joint(enc[t], dec)
  idx = argmax(joined[:vocab_size])
  duration_idx = argmax(joined[vocab_size:])
  if token is not blank **then**
    hyp.append(idx2token[idx])
  end if
  t += duration_idx2duration[duration_idx]
end while
return hyp

4. Experiments

We evaluate our model in three different tasks: speech recognition, speech translation, and spoken language understanding. We use the NeMo (Kuchaiev et al., 2019) toolkit for all experiments. Unless specified otherwise, we use *Conformer-Large* for all tasks. [8] For acoustic feature extraction, we use audio frames of 10 ms and window sizes of 25 ms. Our model has a conformer encoder with 17 layers with num-heads = 8, and relative position embeddings. The hidden dimension of all the conformer layers is set to 512, and for the feed-forward layers in the conformer, an expansion factor of 4 is used. The convolution layers use a kernel size of 31. At the beginning of the encoder, convolution-based subs-ampling is performed with subsampling rate 4. All models have around 120M parameters, The exact number of parameters may vary, depending on the size of the sub-word vocabulary and durations used with TDT models. We use different subword-based tokenizers for different models, which will be described in their respective sections. Unless specified otherwise, logit under-normalization is used during training with $\sigma = 0.05$. For all experiments, we train our models for no more than 200 epochs, and run checkpoint-averaging performed on 5 checkpoints with the best performance on validation data, to generate the model for evaluation. We run non-batched greedy search inference [9] for all evaluations reported in this Section. TDT Batched inference is discussed in Section 5.2. No external LM is used in any of our experiments.

[8] `examples/asr/conf/conformer/conformer_transducer_bpe.yaml` in `https://github.com/NVIDIA/NeMo`

[9] Beam search for TDT models is highly complex since the search space spans both token and duration dimensions. That being said, it is possible to come up with different pruning methods to step up the TDT beam search, which will be our future work.

TDT config	WER(%)	time(s)	rel. speed-up
RNNT	2.14	256	-
0-2	2.35	175	1.46X
0-4	2.17	129	1.98X
0-6	2.14	119	2.15X
0-8	2.11	117	2.19X

Table 1. English ASR, Librispeech test-clean. TDT vs RNNT: WER, decoding time, and relative speed-up against the RNNT.

TDT config	WER(%)	time(s)	rel. speed-up
RNNT	5.11	244	-
0-2	5.50	171	1.43X
0-4	5.06	128	1.91X
0-6	5.05	118	2.07X
0-8	5.16	115	2.12X

Table 2. English ASR, Librispeech test-other. TDT vs RNNT: WER, decoding time, and relative speed-up against the RNNT.

4.1. Speech Recognition

We evaluate TDT for English, German, and Spanish ASR. All ASR models uses Conformer-Large encoder with stateless decoders (Ghodsi et al., 2020), which concatenates the embeddings of the last 2 history words as the decoder output. All models use Byte-Pair Encoding (BPE) (Sennrich et al., 2015) as the text representation with vocabulary size = 1024. Fast-emit (Yu et al., 2021) regularization is used in all our models, with $\lambda = 0.01$.

For each language, the baseline is the conventional Transducer model. We test TDT models with different $\mathcal{D}$ configurations. We choose consecutive integers as our configurations, and use a shorthand notation to represent durations from 0 to the maximum value, e.g. "0-4" means $\mathcal{D} = \{0, 1, 2, 3, 4\}$. All models have been trained only on public datasets to make the experiments reproducible.

4.1.1. English ASR

Our English ASR models are trained on the Librispeech (Panayotov et al., 2015) set with 960 hours of speech. Speed perturbation with factors (0.9, 1.0, 1.1) is performed to augment the dataset. TDT models achieve similar accuracy compared to the baseline (RNNT). TDT models are also significantly faster in inference, up to 2.19X and 2.12X with config 0-8 for test-clean and test-other, respectively (see Tables 1 and 2).

4.1.2. Spanish ASR

Our Spanish models are trained on combination of Mozilla Common Voice (MCV) (Ardila et al., 2019), Multilingual

TDT config	WER(%)	time(s)	rel. speed-up
RNNT	19.84	47	-
0-2	17.95	33	1.42X
0-4	18.57	26	1.81X
0-6	18.06	24	1.96X
0-8	18.73	24	1.96X

Table 3. Spanish ASR, on CallHome dataset. TDT vs RNNT: WER, decoding time, and relative speed-up against RNNT.

TDT config	WER(%)	time(s)	rel. speed-up
RNNT	3.99	558	-
0-2	4.10	352	1.59X
0-4	3.93	232	2.41X
0-6	4.00	207	2.70X
0-8	3.95	198	2.82X

Table 4. German ASR on MLS set. TDT vs RNNT: WER, decoding time, and relative speed-up against RNNT.

Librispeech (MLS) (Pratap et al., 2020), Voxpopuli (Wang et al., 2021a), and Fisher (LDC2010S01) dataset with 1340 hours in total. We evaluate our model on the Spanish Callhome (LDC96S35) test set. We see consistent WER improvement with TDT models compared to RNNT, with up to almost 2 absolute WER points for 0-2 TDT, and over 1 absolute WER point for our fastest model with configuration = 0-8 (See Table 3). TDT models are much faster than RNNT, with maximum speed up factor of 1.96 for 0-6 and 0-8 TDT configurations.

4.1.3. German ASR

The German ASR was trained on MCV, MLS, and Voxpopuli datasets, with a total of around 2000 hours. Models are evaluated on MLS test set. TDT models have accuracy similar or better than RNNTs (Table 4). We also observe 2.82X speed up on German MLS test for TDT 0-8 configuration, which is higher than on other datasets.[10]

4.2. Speech Translation

We evaluate TDT models on English-to-German Speech Translation. For baseline, we directly applies a Conformer Transducer model on speech translation datasets, without any changes to the model. To the best of our knowledge, at the time of writing this paper, there are no reported results with such models, while the closest are from (Xue et al., 2022), where the authors added attention pooling to the joint network in the Transducer model in order to better model reordering in translations. We train our models on a combination of MUST-C V2 (Cattoni et al., 2021), CoVoST V2 (Wang et al., 2021b), ST-TED (Niehues et al., 2018), Europarl-ST (Iranzo-Sánchez et al., 2020), as well as English audio data from CommonVoice v6 and VoxPopuli v2 with German text generated with an NMT model trained on WMT21 (Farhad et al., 2021) data. Tokenization of the text uses the YouTokenToMe [11] tokenizer, with a 16k vocabulary size. All models are trained from scratch using the aforementioned training set. Table 5 shows our results on the MUST-C V2 Test dataset. For reference, we also include results of the best publicly available model at the time of writing from (Indurthi et al., 2021).[12] Note, the models are not directly comparable with our RNNT and TDT models, since they are trained on different datasets. We see that while baseline RNNT gives a decent result of a BLEU score of 23.21, TDT models consistently improve that with up to 1.26 BLEU score points, and the inference is up to 2.27X faster (see Table 5). TDT models demonstrate a stronger modeling capacity over conventional RNNTs.

Model	BLEU (%)	time(s)	speed-up
(Indurthi et al., 2021)	28.88	N/A	N/A
RNNT	23.21	218	-
TDT 0-2	24.03	143	1.52X
TDT 0-4	24.15	106	2.06X
TDT 0-8	24.47	96	2.27X

Table 5. Speech Translation, MUST-C V2 Test dataset. TDT vs RNNT: BLEU score, inference time, and relative inference speed-up of different speech translation models.

4.3. Spoken Language Understanding

In this section, we apply TDT models to spoken language understanding (SLU), specifically the *Speech Intent Classification and Slot Filling* (SICSF) task, which takes audio as input to detect user intents and extract the corresponding lexical fillers for detected entity slots (Bastianelli et al., 2020). An intent is composed of a scenario type and an action type, while slots and fillers are represented by key-value pairs. The ground-truth intents and slots of input are organized as a Python dictionary, represented as a Python string. The SICSF task is to predict this text based on the input audio. Experiments are conducted using the SLURP (Bastianelli et al., 2020) dataset, where intent accuracy and SLURP-F1

[10] The large speed-up is related to the fact that MLS dataset has longer text: for other datasets (Librispeech, Spanish Callhome), an utterance contains on average between 20 and 40 subword tokens, but MLS has 68. While the encoder is easily parallelized, the decoding is autoregressive and it has to be performed sequentially. Therefore the model spends more time in the decoding search, so we see a larger speed up.

[11] https://github.com/VKCOM/YouTokenToMe

[12] https://paperswithcode.com/sota/speech-to-text-translation-on-must-c-en-de.

Model	#params (M)	intent acc.	SLURP F1	rel. speedup
SpeechBrain	96	87.7	76.19	N/A
ESPnet-SLU	109	86.52	76.91	N/A
RNNT	119	88.53	79.41	-
TDT 0-2	119	87.12	79.43	1.17x
TDT 0-4	119	89.85	80.03	1.17x
TDT 0-6	119	89.28	80.61	1.28x
TDT 0-8	119	90.07	79.90	1.28x

Table 6. Speech intent classification and slot filling on SLURP dataset. TDT vs RNNT: Relative speed-up against RNNT.

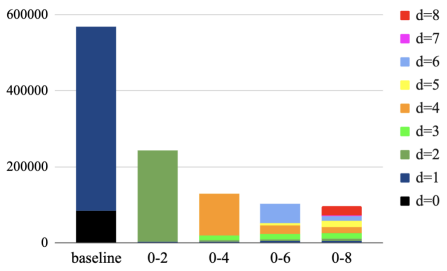

Figure 4. Duration distribution during inference on Librispeech test-other. The model is trained with 4X subsampling. The y-axis shows the number of emissions of different types of durations during the inference of Librispeech test-other datasets.

are used as the evaluation metric.

Our baseline model is a Conformer Transducer model, initialized from our pretrained ASR model [13]. We also include results from two state-of-the-art models from ESPNet-SLU (Arora et al., 2022) and SpeechBrain (Wang et al., 2021c) for comparison. Both ESPNet-SLU and SpeechBrain use HuBERT (Hsu et al., 2021) encoders pretrained on LibriLight-60k (Kahn et al., 2020), while ESPNet-SLU further finetunes the encoder on LibriSpeech before training on SLURP. For our TDTs, we use the same duration configurations that contain maximum durations 2, 4, 6, and 8. Different from our ASR and ST experiments that use $\sigma = 0.05$, here we use $\sigma = 0.02$ for all experiments since we found using $\sigma = 0.05$ may destabilize training for SLURP[14].

The results are shown in Table 6. While the RNNT baseline already has better performance than ESPNet-SLU and SpeechBrain baselines, TDT models with [0-4], [0-6], and [0-8] configurations achieve even better accuracy which makes the new state-of-the-art in the SICSF. In addition, the TDT [0-8] model is 1.28X faster in inference than RNNT.[15] The results demonstrate not only the effectiveness but also the efficiency of TDT algorithm applied in SICSF tasks.

We notice relatively smaller speed-up factors with TDT in this task. This could be explained by the much lower average audio-to-token-length ratio for SICSF tasks. For example, in ASR tasks, the typical ratio between audio length to text length is around 7:1, and the ratio is around 0.89:1 for the SLURP testset, with average text sequence being longer than audio sequence[16]. Nevertheless, we see a significant speed up, which also shows that TDT models can bring improvement even for scenarios where the audio sequence is longer than the text sequences.

5. Discussion

5.1. TDT Emission Analysis

In this section, we investigate the output distribution of TDT models using Librispeech test-other dataset. First, we collect statistics on the duration predicted during decoding, with different TDT configurations (Fig. 4). For the baseline RNN-T, we treat blank and non-blank symbol emissions as having durations 1 and 0, respectively, since blank advances the t by one and non-blank does not. TDT models with configs [0-2] and [0-4] fully utilize longer durations during inference, with almost all of the durations predicted for the [0-2] model, and around 90% of durations for the [0-4] model have the maximum duration. For [0-6] and [0-8] models, the frequencies of predicted long durations are reduced. This is expected since our analysis shows the average ratio of audio length to text length for Librispeech test-other is 5.5:1, smaller than 6. Hence, it is not possible to always emit such long durations.

Next, we collect the frequency of blank emissions vs non-blank emissions (Fig. 5). We can see that as the model incorporates longer durations, fewer and fewer blank emissions are produced with TDT models, while the number of non-blank emissions remains unchanged. TDT models with durations [0-6] and [0-8] have very few blank emissions, indicating that TDT models are close to the theoretical lower bound in terms of the least number of decoding steps.

[13] https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/stt_en_conformer_transducer_large

[14] This is caused by a much smaller ratio between the audio and text length of SLU datasets: the average ratio of audio to and text is 0.89:1 for SLURP, compared to around 5.5:1 for ASR for example. Since on average audio is shorter than text, setting σ too high, which encourages large duration outputs, will hurt training. A smaller σ alleviates the issue.

[15] SLURP has on average shorter audio than text . This means larger durations occur much less for SLURP, resulting in smaller speed-ups compared to ASR and ST.

[16] Text sequence is computed as the number of subword tokens, and audio sequence is computed as the number of 40ms frames due to 10ms per audio frame during feature extraction, with 4X subsampling performed by the encoder.

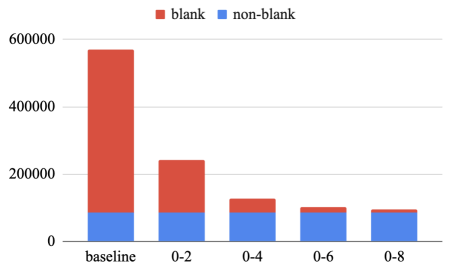

Figure 5. Emission counts of blanks VS non-blanks in Librispeech test-other. The model is trained with 4X subsampling. The y-axis shows the number of emissions of either a blank symbol (red) or a non-blank symbol (blue) during the inference of Librispeech test-other datasets.

5.2. TDT Batched Inference

The main difficulty with batched inference for TDT models is that utterances in the batch may have different duration outputs that denote the number of frames that should be skipped. As a result, it is difficult to fully parallelize the computation for the same batch. One can make the whole batch skip the same number of frames, by selecting the minimum of predicted durations, e.g. if batch-size=4, and predicted durations are $\{3, 4, 3, 6\}$, we advance the whole batch by 3 frames. However, we found this method results in significantly increased insertion errors with the same tokens repeated multiple times. This is not surprising since we skip fewer frames than the model indicates, and the model would not be ready to emit the next token, but can only emit previously emitted tokens instead. To solve this issue, we propose a modification to the training loss of our models, by combining TDT loss with conventional transducer loss L_{TDT} with a sampling probability ω:

$$\mathcal{L}_{\text{sampled}} = \begin{cases} \mathcal{L}_{\text{Transducer}}, & \text{with probability } \omega \\ \mathcal{L}_{\text{TDT}}, & \text{with probability } 1-\omega \end{cases} \tag{18}$$

Note, the conventional transducer loss $\mathcal{L}_{\text{Transducer}}$ is computed on the token logits only, and the duration logits will not take part in the computation nor get updated. We found that the sampled loss solves the aforementioned performance degradation issue. Table 7 shows the ASR performance and inference speed of TDT models when training with $\omega = 0.1$, and running inference with batch=4. We even see slightly improved ASR accuracy, as well as inference speed-up with batched inference with TDT. [17]

[17] The speed-up for batched inference is slightly smaller than for non-batched case because 1. the overhead related to padding for batched computation and 2. all utterances in the batch advance by the minimum of predicted durations which increases the number of decoding steps.

TDT config	clean	other	total time	rel speed-up
RNNT	2.13	5.11	274	-
0-2	2.10	4.94	182	1.51X
0-4	2.15	5.04	151	1.81X
0-6	2.10	4.91	146	1.88X
0-8	2.13	5.03	159	1.79X

Table 7. Batched inference for TDT ASR models, trained with loss sampling $\omega = 0.1$. WER (%) on Librispeech test-clean and test-other. Batch-size=4. When different utterances in the same batch predict different durations, we take the minimum of those predictions and advance all utterances in the batch by that amount.

5.3. TDT Robustness to noise

In this section, we compare the noise robustness for TDT and RNNT ASR models. For this, we run inference on Librispeech test-clean augmented with noise in different signal-noise-ratios (SNRs). For each utterance, we randomly select a noise sample from MUSAN (Snyder et al., 2015) and Freesound [18]. The noise sample is sub-segmented if it's longer than the utterance, or repeated if it's shorter than the utterance. The utterance samples are augmented with noise samples in 0, 5, 10, 15, and 20 SNRs. We report the WER and inference time of conventional Transducers and TDT models with configuration 0-8. We found TDT models perform much better in noisy conditions than conventional Transducers, both in terms of accuracy and speed (Fig. 6). While RNN-T and TDT models achieve similar WERs for clean speech, TDT models gradually outperform RNNT as more noise is added. The inference time for TDT is practically the same for all SNRs. More details of those experiments are in Appendix E.

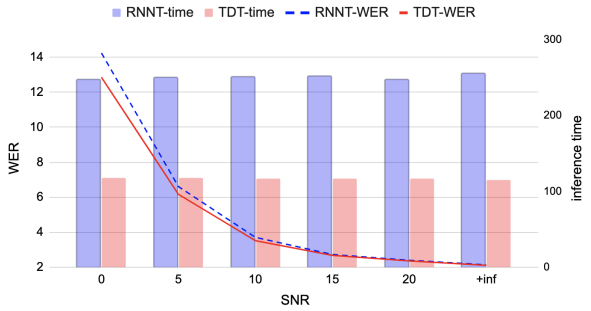

Figure 6. TDT vs RNNT ASR on noisy speech. WER(%) for Librispeech test-clean with noise added at different SNRs. WER on the original test-clean is shown at SNR = +inf. While TDT and RNNT achieve similar WER at low noise conditions, TDT is more robust to noise.

[18] https://freesound.org/

model	WER%
RNNT-LSTM	59.95
RNNT-stateless	64.62
TDT [0-2]	12.59
TDT [0-4]	9.35
TDT [0-6]	6.12
TDT [0-8]	5.78

Table 8. WERs with different Transducer models on TTS generated dataset with repeated digits.

5.4. TDT Robustness with respect to repeated tokens

We notice that RNN-T model performance significantly degrades when the text sequence has repetitions of the same (subword) tokens, for example:

- Ground truth: *seven seven seven nine nine nine eight eight eight*
- RNNT w/ LSTM decoder result: *seven seven eight eight*
- RNNT w/ stateless decoder result: *seven nine eight*

We find TDT models are significantly more robust than RNN-Ts for such cases. We use NeMo TTS to generate 100 audios containing random digits repeating 3 - 5 times and run ASR with different models with results in Table 8. We see that while the conventional RNN-Ts achieve very bad word error rates (all of them with error rates more than 50%, regardless of the type of decoder used), TDT models are able to achieve significantly lower error rates, and this effect is more prominent for TDT models with longer durations. With TDT models with durations 0-8, we achieve a word error rate of 5.78%, which is more than 10X error rate reduction compared to conventional Transducers.

This set of experiments also shows that TDT models do not suffer from the potential issue of accumulation of errors induced by consecutive duration predictions that might be inaccurate. More details on the analysis of repeated tokens and our experiments can be found in Appendix F.

5.5. TDT Comparison with Multi-blank Transducers

Multi-blank Transducer (MBT) (Xu et al., 2022) introduces *big blank* symbols that cover multiple input frames. During inference, when a big blank is emitted, the MBT model skips frames according to the duration of the emitted blank symbol. Compared to multi-blank Transducers which skip frames only with certain blank symbols, TDT models allow frame-skipping for both non-blank and blank symbols, which means potentially larger speed-up factors. We compare MBT and TDT in Table 9. We see that while both MBT and TDT models give comparable WERs, larger inference speedup factors are seen with TDT models when using the same max-duration configs.

model	max-duration	WER	time	rel. speed up
RNNT	-	5.11	244	-
MBT	2	5.15	208	1.17X
TDT	2	5.50	171	1.43X
MBT	4	5.05	161	1.52X
TDT	4	5.06	128	1.91X
MBT	8	5.18	139	1.76X
TDT	8	5.16	115	2.12X

Table 9. Inference and accuracy comparison between 3 type of ASR models: RNNT, multi-blank RNNT (MBT), and TDT. Greedy WER (%) and the total decoding time of the Librispeech test-other with batch = 1. Relative speed-up is measured against the RNNT. For MBT max-duration=4 means MBT model with big-blank-durations=[2,3,4] in addition to the conventional blank. For TDT models, max-duration=4 means model with durations [0,1,2,3,4].

6. Conclusion

In this paper, we propose Token-and-Duration Transducers, which extend conventional Transducer models by adding explicit duration modeling. We present detailed derivations of the extended forward-backward algorithm used for TDT models, as well as the close-form solutions for TDT model training. We show that TDT models are superior to conventional Transducers across multiple sequence tasks, including speech recognition, speech translation, and spoken language understanding. In all those tasks, we see better or similar performances with TDT models than conventional Transducers, while TDT models run inference significantly faster, with up to 2.82X speed up. TDT is also more noise-robust, and robust to token repetition than conventional RNN-Ts. Our TDT implementation will be open-sourced in NVIDIA's NeMo `https://github.com/NVIDIA/NeMo` toolkit.

For future work, we will work on other ways to improve the computational efficiency and accuracy of TDT models, as well as algorithms and implementation for efficient beam-search with TDT models.

References

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. Common voice: A massively-multilingual speech corpus. *arXiv:1912.06670*, 2019.

Arora, S., Dalmia, S., Denisov, P., Chang, X., Ueda, Y.,

Peng, Y., Zhang, Y., Kumar, S., Ganesan, K., Yan, B., et al. ESPnet-SLU: Advancing spoken language understanding through ESPnet. In *ICASSP*, 2022.

Bastianelli, E., Vanzo, A., Swietojanski, P., and Rieser, V. SLURP: A Spoken Language Understanding Resource Package. In *EMNLP*, 2020.

Cattoni, R., Di Gangi, M. A., Bentivogli, L., Negri, M., and Turchi, M. Must-c: A multilingual corpus for end-to-end speech translation. *Computer Speech & Language*, 66: 101155, 2021.

Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. Listen, Attend and Spell. In *ICASSP*, 2016.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. Attention-based models for speech recognition. *NeurIPS*, 28, 2015.

Farhad, A., Arkady, A., Magdalena, B., Ondřej, B., Rajen, C., Vishrav, C., Costa-jussa, M. R., Cristina, E.-B., Angela, F., Christian, F., et al. Findings of the 2021 conference on machine translation (WMT21). In *Conf. on Machine Translation*, 2021.

Ghodsi, M., Liu, X., Apfel, J., Cabrera, R., and Weinstein, E. RNN-Transducer with stateless prediction network. In *ICASSP*, 2020.

Graves, A. Sequence transduction with recurrent neural networks. In *ICML*, 2012.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., et al. Conformer: Convolution-augmented transformer for speech recognition. In *Interspeech*, 2020.

Han, W., Zhang, Z., Zhang, Y., Yu, J., Chiu, C.-C., Qin, J., Gulati, A., Pang, R., and Wu, Y. Contextnet: Improving convolutional neural networks for automatic speech recognition with global context. In *Interspeech*, 2020.

Hsu, W.-N., Bolte, B., Tsai, Y.-H. H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.

Indurthi, S., Zaidi, M. A., Lakumarapu, N. K., Lee, B., Han, H., Ahn, S., Kim, S., Kim, C., and Hwang, I. Task aware multi-task learning for speech to text tasks. In *ICASSP*, 2021.

Iranzo-Sánchez, J., Silvestre-Cerda, J. A., Jorge, J., Roselló, N., Giménez, A., Sanchis, A., Civera, J., and Juan, A. Europarl-st: A multilingual corpus for speech translation of parliamentary debates. In *ICASSP*, 2020.

Jelinek, F. *Statistical methods for speech recognition*. MIT press, 1998.

Kahn, J., Rivière, M., Zheng, W., Kharitonov, E., Xu, Q., Mazaré, P.-E., Karadayi, J., Liptchinsky, V., Collobert, R., Fuegen, C., et al. Libri-light: A benchmark for ASR with limited or no supervision. In *ICASSP*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kuchaiev, O., Li, J., Nguyen, H., Hrinchuk, O., Leary, R., Ginsburg, B., Kriman, S., Beliaev, S., Lavrukhin, V., et al. Nemo: a toolkit for building ai applications using neural modules. In *NeurIPS Workshop on Systems for ML*, 2019.

Li, J., Zhao, R., Hu, H., and Gong, Y. Improving RNN transducer modeling for end-to-end speech recognition. In *ASRU*, 2019.

Niehues, J., Cattoni, R., Stüker, S., Cettolo, M., Turchi, M., and Federico, M. The IWSLT 2018 evaluation campaign. In *IWSLT*, 2018.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: an ASR corpus based on public domain audio books. In *ICASSP*, 2015.

Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeuRIPS*, 2019.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. The Kaldi speech recognition toolkit. In *ASRU*, 2011.

Pratap, V., Xu, Q., Sriram, A., Synnaeve, G., and Collobert, R. MLS: A large-scale multilingual dataset for speech research. In *Interspeech*, 2020.

Ravanelli, M., Parcollet, T., Plantinga, P., Rouhe, A., Cornell, S., Lugosch, L., Subakan, C., Dawalatabad, N., Heba, A., Zhong, J., et al. SpeechBrain: A general-purpose speech toolkit. In *Interspeech*, 2021.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *Proc. of the 54th Annual Meeting of the ACL*, 2015.

Shrivastava, H., Garg, A., Cao, Y., Zhang, Y., and Sainath, T. Echo state speech recognition. In *ICASSP*, 2021.

Snyder, D., Chen, G., and Povey, D. Musan: A music, speech, and noise corpus. *arXiv:1510.08484*, 2015.

Tian, Z., Yi, J., Tao, J., Bai, Y., and Wen, Z. Self-attention transducers for end-to-end speech recognition. In *Interspeech*, 2019.

Wang, C., Riviere, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., Williamson, M., Pino, J., and Dupoux, E. VoxPopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation. In *Proc. of the 59th Annual Meeting of the ACL and the 11th International Joint Conf. on NLP*, 2021a.

Wang, C., Wu, A., and Pino, J. CoVoST 2 and massively multilingual speech-to-text translation. In *Interspeech*, 2021b.

Wang, Y., Chen, T., Xu, H., Ding, S., Lv, H., Shao, Y., Peng, N., Xie, L., Watanabe, S., and Khudanpur, S. Espresso: A fast end-to-end neural speech recognition toolkit. In *ASRU*, 2019.

Wang, Y., Boumadane, A., and Heba, A. A fine-tuned Wav2vec 2.0/HuBERT benchmark for speech emotion recognition, speaker verification and spoken language understanding. *arXiv:2111.02735*, 2021c.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Enrique Yalta Soplin, N., Heymann, J., Wiesner, M., Chen, N., Renduchintala, A., and Ochiai, T. ESPnet: End-to-end speech processing toolkit. In *Interspeech*, 2018.

Woodland, P. C., Odell, J. J., Valtchev, V., and Young, S. J. Large vocabulary continuous speech recognition using HTK. In *ICASSP*, 1994.

Xu, H., Jia, F., Majumdar, S., Watanabe, S., and Ginsburg, B. Multi-blank transducers for speech recognition. *arXiv:2211.03541*, 2022.

Xue, J., Wang, P., Li, J., Post, M., and Gaur, Y. Large-scale streaming end-to-end speech translation with neural transducers. *arXiv preprint arXiv:2204.05352*, 2022.

Yeh, C.-F., Mahadeokar, J., Kalgaonkar, K., Wang, Y., Le, D., Jain, M., Schubert, K., Fuegen, C., and Seltzer, M. L. Transformer-transducer: End-to-end speech recognition with self-attention. *arXiv:1910.12977*, 2019.

Yu, J., Chiu, C.-C., Li, B., Chang, S.-y., Sainath, T. N., He, Y., Narayanan, A., Han, W., Gulati, A., Wu, Y., et al. FastEmit: Low-latency streaming ASR with sequence-level emission regularization. In *ICASSP*, 2021.

Zhang, Q., Lu, H., Sak, H., Tripathi, A., McDermott, E., Koo, S., and Kumar, S. Transformer Transducer: A streamable speech recognition model with transformer encoders and RNN-T loss. In *ICASSP*, 2020.

A. Derivations of TDT gradients with respect to probabilities

A.1. Background: gradients of RNN-Transducer

The gradient of the conventional Transducer loss function with respect to $P(v|t,u)$ has been derived in (Graves, 2012). The total probability $p(y|x)$ equals to the sum of $\alpha(t,u)\beta(t,u)$ over any top-left to bottom-right diagonal through the probability lattice, i.e. $\forall n : 1 \leq n \leq U+T$

$$P_{\text{RNNT}}(y|x) = \sum_{(t,u):t+u=n} \alpha(t,u)\beta(t,u) \tag{19}$$

We plug $\beta(t,u) = \beta(t+1,u)p(\emptyset|t,u) + \beta(t,u+1)p(y_u|t,u)$, in the equation above and get:

$$P_{\text{RNNT}}(y|x) = \sum_{(t,u):t+u=n} \alpha(t,u)\Big[\beta(t+1,u)p(\emptyset|t,u) + \beta(t,u+1)p(y_{u+1}|t,u)\Big] \tag{20}$$

Now let us take the partial derivative of $p(y|x)$ over individual $p(v|t,u)$:

$$\frac{\partial P_{\text{RNNT}}(y|x)}{\partial P(v|t,u)} = \alpha(t,u)\begin{cases}\beta(t,u+1), & v = y_{u+1}\\ \beta(t+1,u), & v = \emptyset\\ 0, & \text{otherwise.}\end{cases} \tag{21}$$

Since $\mathcal{L}_{\text{RNNT}} = -\log P_{\text{RNNT}}(y|x)$, we have

$$\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(v|t,u)} = \frac{d\mathcal{L}_{\text{RNNT}}}{dP_{\text{RNNT}}(y|x)}\frac{\partial P_{\text{RNNT}}(y|x)}{\partial P(v|t,u)} = -\frac{\alpha(t,u)}{P_{\text{RNNT}}(y|x)}\begin{cases}\beta(t,u+1), & v = y_{u+1}\\ \beta(t+1,u), & v = \emptyset\\ 0, & \text{otherwise.}\end{cases} \tag{22}$$

A.2. Gradients of TDT models

Derivation of gradients for TDT follows similar steps as gradients for conventional Transducers. But first note, that Eq. 19 does not hold true for TDT: since frame-skipping is not possible for a RNNT, for any n, the diagonal $(t,u) : t+u$ contains a set of nodes that *all* possible paths in the lattice must go through. But for TDT, frame-skipping is allowed, so we also need to consider paths that *jump over* the diagonal. The correct formulation for TDT is that $\forall n : 1 \leq n \leq U+T$:

$$\begin{aligned}P_{\text{TDT}}(y|x) = &\sum_{(t,u):t+u=n} \alpha(t,u)\beta(t,u) + \sum_{(t,u,t'):t+u<n,t'+u>n,(t'-t)\in\mathcal{D}} \alpha(t,u)\beta(t',u)P(\emptyset,t'-t|t,u)\\ &+ \sum_{(t,u,t'):t+u<n,t'+u+1>n,(t'-t)\in\mathcal{D}} \alpha(t,u)\beta(t',u+1)P(y_{u+1},t'-t|t,u)\end{aligned} \tag{23}$$

Eq. 23 has extra terms comparing to Eq. 19, which correspond to transitions that go across the specified diagonal. Starting from Eq. 23, we follow similar steps as conventional Transducers. To get the gradients of the token probabilities, we replace $\beta(t,u)$ according to Eq. 8, and take the partial derivative of token probabilities. The last two terms do not have contributions to the partial derivative, and we obtain:

$$\frac{\partial P_{\text{TDT}}(y|x)}{\partial P_T(v|t,u)} = \alpha(t,u)\begin{cases}\sum_{d\in\mathcal{D}}\beta(t+d,u+1)P_D(d|t,u), & v = y_{u+1}\\ \sum_{d\in\mathcal{D}\setminus\{0\}}\beta(t+d,u)P_D(d|t,u), & v = \emptyset\\ 0, & \text{otherwise.}\end{cases} \tag{24}$$

Then taking $\mathcal{L}_{\text{TDT}} = -\log P_{\text{TDT}}(y|x)$, we have:

$$\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P_T(v|t,u)} = -\frac{\alpha(t,u)}{P_{\text{TDT}}(y|x)}\begin{cases}\sum_{d\in\mathcal{D}}\beta(t+d,u+1)P_D(d|t,u), & v = y_{u+1}\\ \sum_{d\in\mathcal{D}\setminus\{0\}}\beta(t+d,u)P_D(d|t,u), & v = \emptyset\\ 0, & \text{otherwise.}\end{cases} \tag{25}$$

Similarly, starting with Eq. 23 and Eq. 8, we take the partial derivatives of the duration probabilities:

$$\frac{\partial P_{\text{TDT}}(y|x)}{\partial P_D(d|t,u)} = \alpha(t,u) \begin{cases} \beta(t,u+1)P_T(y_{u+1}|t,u), & d=0 \\ \beta(t+d,u+1)P_T(y_{u+1}|t,u) + \beta(t+d,u)P_T(\emptyset|t,u), & d>0. \end{cases} \tag{26}$$

Since $\mathcal{L}_{\text{TDT}} = -\log P_{\text{TDT}}(y|x)$, we get:

$$\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P_D(d|t,u)} = -\frac{\alpha(t,u)}{P_{\text{TDT}}(y|x)} \begin{cases} \beta(t,u+1)P_T(y_{u+1}|t,u), & d=0 \\ \beta(t+d,u+1)P_T(y_{u+1}|t,u) + \beta(t+d,u)P_T(\emptyset|t,u), & d>0. \end{cases} \tag{27}$$

B. Derivations of TDT gradients with respect to pre-softmax logits

B.1. Background: Softmax function merging for conventional Transducers

As noted previously, $h^v_{t,u}$ are the pre-softmax logits joint network for the token prediction. (Li et al., 2019) proposed to compute a closed-form solution of Transducer loss of $h^v_{t,u}$ by the following steps.

First, we apply the chain rule to represent the gradients to pre-softmax logits as follows, [19]

$$\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h^v_{t,u}} = \sum_{v' \in \mathcal{V}} \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(v'|t,u)} \frac{\partial P(v'|t,u)}{\partial h^v_{t,u}} \tag{28}$$

where $\mathcal{V}$ represents a set of all vocabulary including the $\emptyset$ token. In the summation, $\frac{\partial \mathcal{L}}{\partial P(v|t,u)}$ can be calculated via Eq. 22. Although this summation is over all elements in $\mathcal{V}$, only two of them are non-zero: the one where $v' = \emptyset$ and where $v' = y_{u+1}$. Therefore, we could simplify Eq. 28 as,

$$\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h^v_{t,u}} = \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(\emptyset|t,u)} \frac{\partial P(\emptyset|t,u)}{\partial h^v_{t,u}} + \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h^v_{t,u}} \tag{29}$$

Next, consider the second part of each term in the summation, $\frac{\partial P(.|t,u)}{\partial h^v_{t,u}}$. The gradients of the softmax $y = \text{softmax}(x)$ are:

$$\frac{\partial y_i}{\partial x_j} = \begin{cases} -y_i y_j, i \neq j \\ y_i(1-y_i), i = j \end{cases} \tag{30}$$

Now we could simplify Eq. 29 based on different v. When $v = \emptyset$, we have

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h^{\emptyset}_{t,u}} =& \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(\emptyset|t,u)} \frac{\partial P(\emptyset|t,u)}{\partial h^{\emptyset}_{t,u}} + \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h^{\emptyset}_{t,u}} \\ =& -\frac{\alpha(t,u)\beta(t+1,u)}{P_{\text{RNNT}}(y|x)} P(\emptyset|t,u)(1-P(\emptyset|t,u)) + \frac{\alpha(t,u)\beta(t,u+1)}{P_{\text{RNNT}}(y|x)} P(y_{u+1}|t,u)P(\emptyset|t,u) \\ =& \frac{P(\emptyset|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \Big[\underbrace{\beta(t+1,u)P(\emptyset|t,u) + \beta(t,u+1)P(y_{u+1}|t,u)}_{\beta(t,u)} - \beta(t+1,u) \Big] \\ =& \frac{P(\emptyset|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \Big[\beta(t,u) - \beta(t+1,u) \Big] \end{aligned} \tag{31}$$

[19] In the original paper (Li et al., 2019), although their final result is correct, there seems to be a small issue in their Eq. 12, where the authors did not include the summation.

Similarly, when $v = y_{u+1}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h_{t,u}^{y_{u+1}}} =& \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(\emptyset|t,u)} \frac{\partial P(\emptyset|t,u)}{\partial h_{t,u}^{y_{u+1}}} + \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h_{t,u}^{y_{u+1}}} \\
=& \frac{\alpha(t,u)\beta(t+1,u)}{P_{\text{RNNT}}(y|x)} P(\emptyset|t,u)P(y_{u+1}|t,u) - \frac{\alpha(t,u)\beta(t,u+1)}{P_{\text{RNNT}}(y|x)} P(y_{u+1}|t,u)(1 - P(y_{u+1}|t,u)) \\
=& \frac{P(y_{u+1}|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \Big[\underbrace{\beta(t+1,u)P(\emptyset|t,u) + \beta(t,u+1)P(y_{u+1}|t,u)}_{\beta(t,u)} - \beta(t,u+1) \Big] \\
=& \frac{P(y_{u+1}|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \Big[\beta(t,u) - \beta(t,u+1) \Big]
\end{aligned} \tag{32}$$

Lastly, when $v \neq \emptyset$ and $v \neq y_{u+1}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h_{t,u}^{v}} =& \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(\emptyset|t,u)} \frac{\partial P(\emptyset|t,u)}{\partial h_{t,u}^{v}} + \frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h_{t,u}^{v}} \\
=& \frac{\alpha(t,u)\beta(t+1,u)}{P_{\text{RNNT}}(y|x)} P(\emptyset|t,u)P(v|t,u) + \frac{\alpha(t,u)\beta(t,u+1)}{P_{\text{RNNT}}(y|x)} P(y_{u+1}|t,u)P(v|t,u) \\
=& \frac{P(v|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \Big[\underbrace{\beta(t+1,u)P(\emptyset|t,u) + \beta(t,u+1)P(y_{u+1}|t,u)}_{\beta(t,u)} \Big] \\
=& \frac{P(v|t,u)\alpha(t,u)\beta(t,u)}{P_{\text{RNNT}}(y|x)}
\end{aligned} \tag{33}$$

Combining Eq. 31, 32, 33, we get the gradients of Transducer loss over pre-softmax logits, shown in Eq. 34:

$$\frac{\partial \mathcal{L}_{\text{RNNT}}}{\partial h_{t,u}^{v}} = \frac{P(v|t,u)\alpha(t,u)}{P_{\text{RNNT}}(y|x)} \left[\beta(t,u) - \begin{cases} \beta(t,u+1), & v = y_{u+1} \\ \beta(t+1,u), & v = \emptyset \\ 0, & \text{otherwise} \end{cases} \right] \tag{34}$$

B.2. Softmax function merging for TDT

The derivation of pre-softmax logit gradients for TDT loss is slightly more complex than the conventional Transducer but follows similar steps. First we follow the chain rule by writing the gradient as the summation of terms, and then listing only the non-zero terms in the sum,

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h_{t,u}^{v}} =& \sum_{v \in \mathcal{V}} \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(v|t,u)} \frac{\partial P(v|t,u)}{\partial h_{t,u}^{v}} \\
=& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(\emptyset|t,u)} \frac{\partial P(\emptyset|t,u)}{\partial h_{t,u}^{v}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h_{t,u}^{v}}
\end{aligned} \tag{35}$$

Now we simplify this depending on different v.

B.2.1. The case when $v = \varnothing$

$$
\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{\varnothing}_{t,u}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(\varnothing|t,u)} \frac{\partial P(\varnothing|t,u)}{\partial h^{\varnothing}_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h^{\varnothing}_{t,u}} \\
=& - \frac{\alpha(t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(\varnothing|t,u)(1 - P(\varnothing|t,u)) \\
&+ \frac{\alpha(t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(\varnothing|t,u) P(y_{u+1}|t,u) \\
=& \frac{P(\varnothing|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)} \\
&\left[(P(\varnothing|t,u) - 1) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \right]
\end{aligned} \tag{36}
$$

The term in the bracket could be further simplified as,

$$
\begin{aligned}
& (P(\varnothing|t,u) - 1) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \\
=& - \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) \\
&+ P(\varnothing|t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \\
=& - \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + \underbrace{\left[\sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P(\varnothing, d|t,u) + \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P(y_{u+1}, d|t,u) \right]}_{\beta(t,u)} \\
=& \beta(t,u) - \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)
\end{aligned} \tag{37}
$$

Apply the results of 37 to 35, we get

$$
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{\varnothing}_{t,u}} = \frac{P(\varnothing|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)} \left[\beta(t,u) - \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) \right] \tag{38}
$$

B.2.2. The case when $v = y_{u+1}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{y_{u+1}}_{t,u}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(\varnothing|t,u)} \frac{\partial P(\varnothing|t,u)}{\partial h^{y_{u+1}}_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(y_{u+1}|t,u)} \frac{\partial P(y_{u+1}|t,u)}{\partial h^{y_{u+1}}_{t,u}} \\
=& \frac{\alpha(t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(\varnothing|t,u) P(y_{u+1}|t,u) \\
&+ \frac{\alpha(t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(y_{u+1}|t,u)(P(y_{u+1}|t,u) - 1) \\
=& \frac{P(y_{u+1}|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)} \\
&\left[P(\varnothing|t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + (P(y_{u+1}|t,u) - 1) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \right]
\end{aligned} \tag{39}
$$

The term in the bracket could be further simplified as,

$$
\begin{aligned}
& P(\emptyset|t,u) \sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u) + (P(y_{u+1}|t,u)-1)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) \\
=& -\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) + P(\emptyset|t,u)\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u) \\
& + P(y_{u+1}|t,u)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) \\
=& -\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) + \underbrace{\left[\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P(\emptyset,d|t,u) + \sum_{d\in\mathcal{D}} \beta(t+d,u+1)P(y_{u+1},d|t,u)\right]}_{\beta(t,u)} \\
=& \beta(t,u) - \sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u)
\end{aligned}
\tag{40}
$$

Apply the results of 40 to 35, we get

$$
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h_{t,u}^{y_{u+1}}} = \frac{P(y_{u+1}|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)}\left[\beta(t,u) - \sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u)\right] \tag{41}
$$

B.2.3. THE CASE WHEN $v \neq y_{u+1}$ AND $v \neq \emptyset$

$$
\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h_{t,u}^{v}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(\emptyset|t,u)}\frac{\partial P(\emptyset|t,u)}{\partial h_{t,u}^{v}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P(y_{u+1}|t,u)}\frac{\partial P(y_{u+1}|t,u)}{\partial h_{t,u}^{v}} \\
=& \frac{\alpha(t,u)\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(\emptyset|t,u)P(y_{u+1}|t,u) \\
& + \frac{\alpha(t,u)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u)}{P_{\text{TDT}}(y|x)} P(y_{u+1}|t,u)P(y_{u+1}|t,u) \\
=& \frac{P(y_{u+1}|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)} \\
& \left[P(\emptyset|t,u)\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u) + P(y_{u+1}|t,u)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u)\right]
\end{aligned}
\tag{42}
$$

The term in the bracket could be further simplified as,

$$
\begin{aligned}
& P(\emptyset|t,u)\sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u) + P(y_{u+1}|t,u)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) \\
=& \sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u) + P(y_{u+1}|t,u)\sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u) \\
=& \sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P(\emptyset,d|t,u) + \sum_{d\in\mathcal{D}} \beta(t+d,u+1)P(y_{u+1},d|t,u) \\
=& \beta(t,u)
\end{aligned}
\tag{43}
$$

Apply the results of 43 to 35, we get

$$
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h_{t,u}^{v}} = \frac{P(v|t,u)\alpha(t,u)\beta(t,u)}{P_{\text{TDT}}(y|x)} \tag{44}
$$

Combining the results from Equations 38, 41, 44, we have

$$
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h_{t,u}^{v}} = \frac{P(v|t,u)\alpha(t,u)}{P_{\text{TDT}}(y|x)}\left[\beta(t,u) - \begin{cases} \sum_{d\in\mathcal{D}\setminus\{0\}} \beta(t+d,u)P_D(d|t,u), & v=\emptyset \\ \sum_{d\in\mathcal{D}} \beta(t+d,u+1)P_D(d|t,u), & v=y_{u+1} \\ 0, & \text{otherwise} \end{cases}\right] \tag{45}
$$

C. Derivation of TDT Gradients with Logit Under-normalization

In this section we derive the gradients of TDT with logit under-normalization introduced in Section 3.3). Let's denote the pseudo "probability" acquired with under-normalization, using the "under-softmax" operation of $y' = \frac{\text{softmax}(x)}{\exp(\sigma)}$ as $P'(v|t,u) = \frac{P(v|t,u)}{\exp(\sigma)}$. Let's first work out the gradients of the "under-softmax" operation. We assume $y = \text{softmax}(x)$ and thus $y' \exp(\sigma) = y$, then

$$\begin{aligned}
\frac{\partial y'_i}{\partial x_j} = \frac{\partial y'_i}{\partial y_i}\frac{\partial y_i}{\partial x_j} &= \frac{1}{\exp(\sigma)} \begin{cases} y_i(1-y_i), & i = j \\ -y_i y_j & i \neq j \end{cases} \\
&= \frac{1}{\exp(\sigma)} \begin{cases} \exp(\sigma) y'_i (1 - \exp(\sigma) y'_i), & i = j \\ -y'_i y'_j \exp^2(\sigma) & i \neq j \end{cases} \\
&= \begin{cases} y'_i (1 - \exp(\sigma) y'_i), & i = j \\ -y'_i y'_j \exp(\sigma) & i \neq j \end{cases}
\end{aligned} \tag{46}$$

Next apply the chain rule:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^v_{t,u}} &= \sum_{v \in \mathcal{V}} \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(v|t,u)} \frac{\partial P'(v|t,u)}{\partial h^v_{t,u}} \\
&= \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(\emptyset|t,u)} \frac{\partial P'(\emptyset|t,u)}{\partial h^v_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(y_{u+1}|t,u)} \frac{\partial P'(y_{u+1}|t,u)}{\partial h^v_{t,u}}
\end{aligned} \tag{47}$$

We would like to emphasize that under-normalization is only applied to token logits, not duration logits. Now, let's use $P'(y|x)$ to denote the pseudo "probability" of the sequence, computed with $P'(v|t,u)$ throughout the forward-backward algorithm. Now we further simplify the terms.

C.1. When v is neither $\emptyset$ nor y_{u+1}

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^v_{t,u}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(\emptyset|t,u)} \frac{\partial P'(\emptyset|t,u)}{\partial h^v_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(y_{u+1}|t,u)} \frac{\partial P'(y_{u+1}|t,u)}{\partial h^v_{t,u}} \\
=& \frac{\alpha(t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)}{P'(y|x)} P'(\emptyset|t,u) P'(v|t,u) \exp(\sigma) \\
&+ \frac{\alpha(t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}{P'(y|x)} P'(y_{u+1}|t,u) P'(v|t,u) \exp(\sigma) \\
=& \frac{\exp(\sigma) P'(v|t,u) \alpha(t,u)}{P'(y|x)} \\
&\underbrace{\left[P'(\emptyset|t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P'(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \right]}_{\beta(t,u)} \\
=& \frac{\exp(\sigma) P'(v|t,u) \alpha(t,u) \beta(t,u)}{P'(y|x)} \\
=& \frac{P(v|t,u) \alpha(t,u) \beta(t,u)}{P'(y|x)}
\end{aligned} \tag{48}$$

C.2. When $v = \emptyset$

$$
\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{\emptyset}_{t,u}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(\emptyset|t,u)} \frac{\partial P'(\emptyset|t,u)}{\partial h^{\emptyset}_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(y_{u+1}|t,u)} \frac{\partial P'(y_{u+1}|t,u)}{\partial h^{\emptyset}_{t,u}} \\
=& \frac{\alpha(t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)}{P'(y|x)} P'(\emptyset|t,u)(P'(\emptyset|t,u)\exp(\sigma) - 1) \\
&+ \frac{\alpha(t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}{P'(y|x)} P'(y_{u+1}|t,u) P'(\emptyset|t,u)\exp(\sigma) \\
=& \frac{\exp(\sigma) P'(v|t,u)\alpha(t,u)}{P'(y|x)} \Bigg[-\frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) \\
&\underbrace{+ P'(\emptyset|t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P'(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}_{\beta(t,u)} \Bigg] \\
=& \frac{\exp(\sigma) P'(v|t,u)\alpha(t,u)}{P'(y|x)} \left[\beta(t,u) - \frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) \right] \\
=& \frac{P(v|t,u)\alpha(t,u)}{P'(y|x)} \left[\beta(t,u) - \frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) \right]
\end{aligned} \tag{49}
$$

C.3. When $v = y_{u+1}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{y_{u+1}}_{t,u}} =& \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(\emptyset|t,u)} \frac{\partial P'(\emptyset|t,u)}{\partial h^{y_{u+1}}_{t,u}} + \frac{\partial \mathcal{L}_{\text{TDT}}}{\partial P'(y_{u+1}|t,u)} \frac{\partial P'(y_{u+1}|t,u)}{\partial h^{y_{u+1}}_{t,u}} \\
=& \frac{\alpha(t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u)}{P'(y|x)} P'(y_{u+1}|t,u) P'(\emptyset|t,u)\exp(\sigma) \\
&+ \frac{\alpha(t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}{P'(y|x)} P'(y_{u+1}|t,u)(P'(y_{u+1}|t,u)\exp(\sigma) - 1) \\
=& \frac{\exp(\sigma) P'(v|t,u)\alpha(t,u)}{P'(y|x)} \Bigg[-\frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \\
&\underbrace{+ P'(\emptyset|t,u) \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u) + P'(y_{u+1}|t,u) \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u)}_{\beta(t,u)} \Bigg] \\
=& \frac{\exp(\sigma) P'(v|t,u)\alpha(t,u)}{P'(y|x)} \left[\beta(t,u) - \frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \right] \\
=& \frac{P(v|t,u)\alpha(t,u)}{P'(y|x)} \left[\beta(t,u) - \frac{1}{\exp(\sigma)} \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u) \right]
\end{aligned} \tag{50}
$$

Combining Eq. 48, 49 and 50, we have the TDT gradients with under-normalization as,

$$
\frac{\partial \mathcal{L}_{\text{TDT}}}{\partial h^{v}_{t,u}} = \frac{P(v|t,u)\alpha(t,u)}{P'(y|x)} \left[\beta(t,u) - \frac{1}{\exp(\sigma)} \begin{cases} \sum_{d \in \mathcal{D}} \beta(t+d,u+1) P_D(d|t,u), & v = y_{u+1} \\ \sum_{d \in \mathcal{D} \setminus \{0\}} \beta(t+d,u) P_D(d|t,u), & v = \emptyset \\ 0, & \text{otherwise} \end{cases} \right] \tag{51}
$$

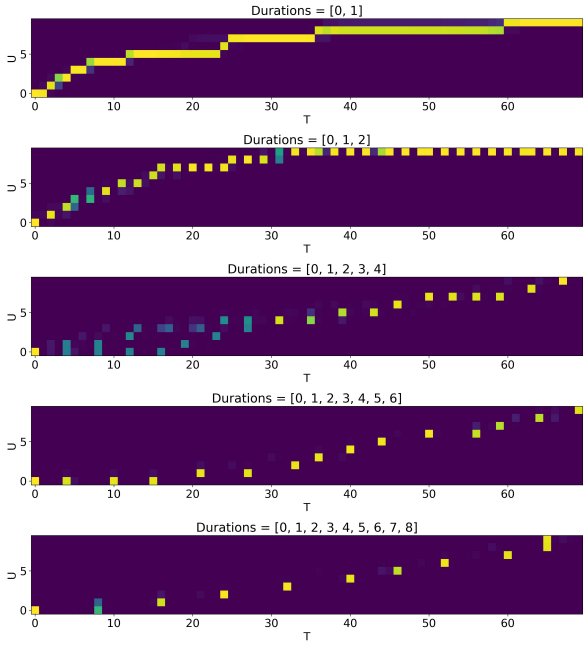

Figure 7. Alignment ($P(\mathcal{A}_{t,u}|x)$ as a function of the duration ($\mathcal{D}$) supported by the TDT model. Simulated joint (J_S) trained with a larger number of duration tokens (N_d) possess alignments with correspondingly longer gaps between time steps ($t \in T$) for each token prediction ($u \in U$)

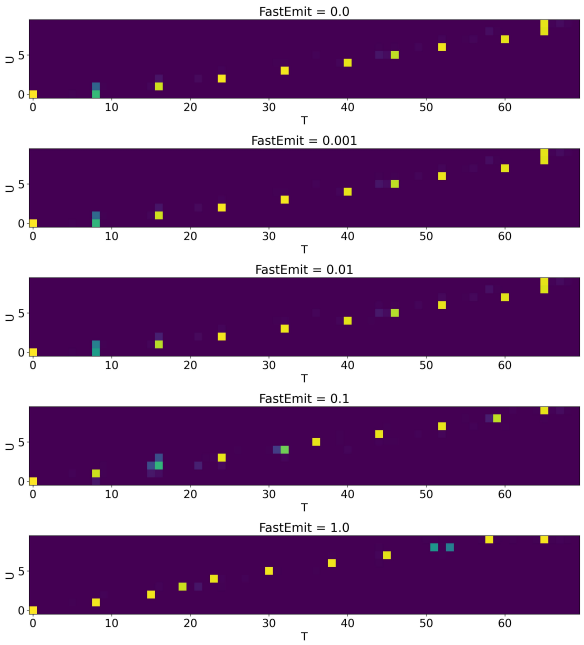

Figure 8. Alignment ($P(\mathcal{A}_{t,u}|x)$ as a function of the FastEmit regularization strength (λ). Simulated joint (J_S) trained with varying λ possess alignments with a correspondingly shorter delay between each token prediction ($u \in U$) as compared to the baseline of $\lambda = 0$

D. Analysis of Token-and-Duration Transducer Alignments

TDT models are capable of learning the alignment $P(\mathcal{A}_{t,u}|x)$ between an input sequence (S) and the corresponding target sequence (S'). This quantity represents the total probability mass that goes through the state (t, u) in the lattice. We use the definition of alignment $P(\mathcal{A}_{t,u}|x)$ from (Yu et al., 2021), computed as,

$$P(\mathcal{A}_{t,u}|x) = \alpha(t, u)\beta(t, u) \tag{52}$$

In the following section, we construct a set of force-alignment experiments to determine the effect of input-output sequence interactions and hyper-parameter choices. Unless stated otherwise, all following experiments are with a simulated joint tensor ($J_S \in R^{T \times U \times (V+1+N_d)}$) for some input sequence (S) sampled from the normal distribution ($N(0, 1)$) and a target sequence ($S \in \mathbb{Z}^U$) generated from the discrete uniform distribution ($U\{1, \ldots, V\}$), where N_d refers the the the number of duration tokens and V is the size of the vocabulary of the token set. We use the PyTorch (Paszke et al., 2019) to optimize J_S using S' as the target sequence. We minimize he TDT loss defined in Eq. 9 using the Adam optimizer (Kingma & Ba, 2015) with a fixed learning rate of 0.1 for 100 update steps. Once J_S is optimized, we compute $P(\mathcal{A}_{t,u}|x)$ and plot the $T \times U$ alignment matrix. We use a fixed random seed so as to reproduce the same J_S and S' when T, U and V are kept constant and chose a fixed value of $T = 70$, $U = 10$, $V = 5$ (chosen simply to speed up convergence), number of duration tokens ($N_d = 8$), $\sigma = 0.05$, $\omega = 0.0$ and fastemit $\lambda = 0.0$ for the experiments unless explicitly mentioned.

D.1. Effect of Durations on TDT Alignments

In a TDT model, one head emits the token while another predicts the duration of the token (say $\mathcal{D} \in \{0, 1, \ldots, (N_d) - 1\}$) where N_d is the number of duration tokens. In the following set of experiments, we attempt to optimize J_S while modifying only the duration set. Given that $T >> U$, and $\sigma > 0$, we expect the alignment to intuitively contain longer-duration tokens as N_d becomes larger.

In Fig. 7, we see that the learned alignment precisely matches our expectations. As the N_d grows larger, the model selects longer durations, significantly reducing the number of decoding steps required. A natural effect of selecting longer tokens is that token emissions are significantly delayed compared to the baseline of $\mathcal{D} \in \{0, 1\}$ which can be considered an

approximation of conventional Transducer alignment with single duration step per token emitted.

Note, that longer duration tokens are enabled by the large difference in the input and target sequence lengths ($T = 70$; $U = 10$), reducing to a $\frac{T}{U} = 7 : 1$ ratio. This ratio of sequence lengths is slightly larger than the observed ratio of acoustic sequence length versus the corresponding sub-word encoded target text tokens in Librispeech with a sufficiently large sub-word encoding vocabulary, close to $\frac{T_{\text{LS}}}{U_{\text{LS}}} \approx 5.5 : 1$. We expect that target token encoding schemes such as character-based encoding will diminish this ratio of sequence lengths to approximately $\frac{T}{U} \approx 2 : 1$, thereby preventing long-duration tokens from being emitted frequently.

D.2. Effect of FastEmit on Alignments

One reduce the delay of token emission using FastEmit method proposed in (Yu et al., 2021). As can be seen in Section D.1, when the number of duration tokens (N_d) is large, the emission of tokens ($u \in U$) is delayed significantly which may hinder latency-sensitive applications. In the following section, we discuss the utilization of FastEmit (Yu et al., 2021) as a strong regularization scheme in order to prevent the model from deferring token emission to such a degree. FastEmit introduces a hyperparameter λ and scales the gradients to token probabilities by $1 + \lambda$ and keeping the gradients to blank probabilities unchanged. We attempt to simply train the same simulated joint (J_S) optimized with different strengths of the λ scaling factor for FastEmit.

In Fig. 8, we find that the effect of FastEmit regularization strength constant (λ) has a substantial effect on reducing the delay between token emissions. It must be noted that $\lambda > 1e$-2 is not realistically applicable when training on non-synthetic data, as the strength of the regularization term will cause the model to diverge, but in this simulated setting, it has been done in order to explicitly show the effect of FastEmit on token emission delay.

An important observation is that when comparing the alignment of Figure 7 (1st row) and Figure 8 (5th row), the first $5 \sim 6$ token emissions occur rapidly with the duration set $N_d = 1$ but are delayed by a significant number of steps for $N_d = 8$. FastEmit does improve the token emission of the first few tokens, however since each token presents a duration of roughly 8 timesteps, the overall latency is significantly higher. This can be tackled by carefully modifying the strength of σ along with λ, so as to discourage long tokens while encouraging faster emission of tokens if latency is a concern.

D.3. Effect of Input Sequence length on Duration Prediction

In Section D.1, we note that the large ratio of input sequence to target sequence $\frac{T}{U}$ was important to the emission of tokens with long durations, and that with a smaller ratio, the model would emit shorter duration tokens, even if it supported a large duration set ($N_d = 8$). In the following section, we attempt to analyze such a scenario, using different input sequence lengths ($T = \{20, 40, 70, 100\}$) while maintaining the target sequence length of $U = 10$. This enables us to analyze the effect of modifying the ratio $\frac{T}{U}$. Note that as a result of changing T, J_S is effectively a different sampled Joint tensor (represented as J_{S_T}), however, the target sequence U remains the same.

In Fig. 9, we observe the alignments as we modify T. Of particular note is that when the ratio $\frac{T}{U}$ is small, tokens with short duration are preferred, performing nearly all token emissions without significant delay, and only towards the end does it emit a $d \in \{6, 8\}$ duration token. On the other hand, when the ratio $\frac{T}{U}$ is large, tokens with long duration are preferred (with a large number of tokens having duration $d = 8$), and with a significant delay of token emission (note that FastEmit has not been enabled here). In Figure 10, we see that increasing the FastEmit strength (λ) provides a corresponding decrease in token latency, even for large differences in sequence lengths.

E. Robustness to Noise

We measure the noise robustness of TDT models, by running inference on Librispeech test-clean augmented with noise samples in different signal-noise-ratios (SNRs). For each utterance, we randomly select a noise sample from MUSAN (Snyder et al., 2015) and Freesound [20]. The noise sample is sub-segmented if it's longer than the utterance, or repeated if it's shorter than the utterance. The utterance samples are augmented with noise samples in 0, 5, 10, 15, and 20 SNRs. We report the WER and inference time of conventional Transducers and TDT models with configuration 0-8. The results are shown in Figure 6. As we can see, while Transducer and TDT models achieve very similar WERs in high SNR scenarios, as more noise is added, TDT models gradually outperform Transducers with larger and larger margins. We also see that despite the

[20] https://freesound.org/

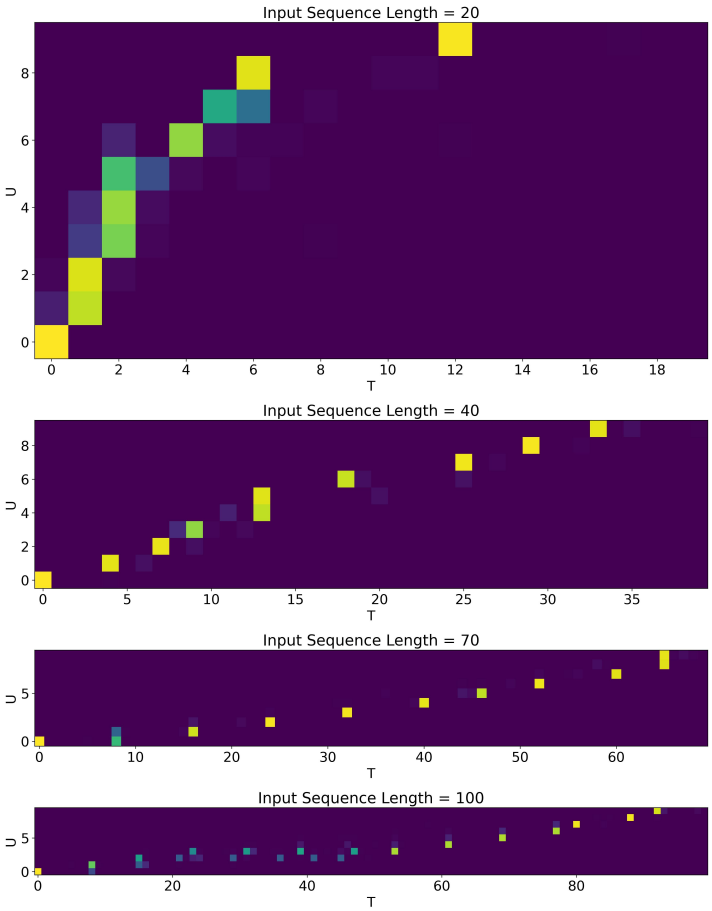

Figure 9. Alignment ($P(\mathcal{A}_{t,u}|x)$ as a function of the input sequence length (T). For each T, the corresponding simulated joint (J_{S_T}) is trained. The alignments for each J_{S_T} show that as T grows, thereby causing a larger ratio of $\frac{T}{U}$, the model begins to emit tokens with longer duration, as well as increased delay in token emission.

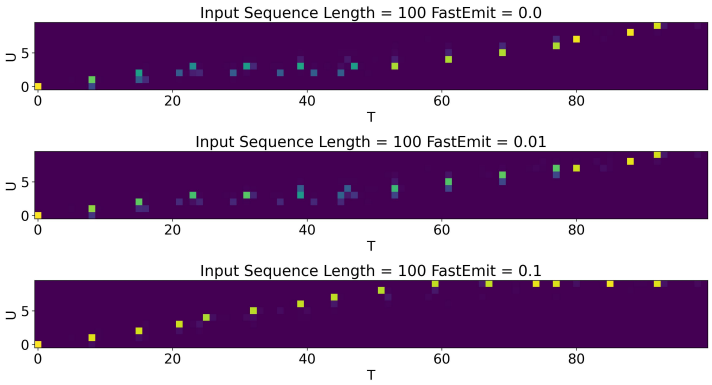

Figure 10. Alignment ($P(\mathcal{A}_{t,u}|x)$ as a function of the input sequence length (T), with FastEmit weight (λ) as regularization. For given $T >> U$, the corresponding simulated joint (J_{S_T}) is trained. The alignments for each J_{S_T} show that as T grows, thereby causing a larger ratio of $\frac{T}{U}$, the model begins to emit tokens with longer duration, as well as increased delay in token emission. By modifying λ, we encourage reduction in token emission delay.

SNR changes, the inference time for TDT only has minimal increases. This shows that TDT models have the capacity to perform much better in noisy conditions than conventional Transducers, in terms of both accuracy and speed aspects.

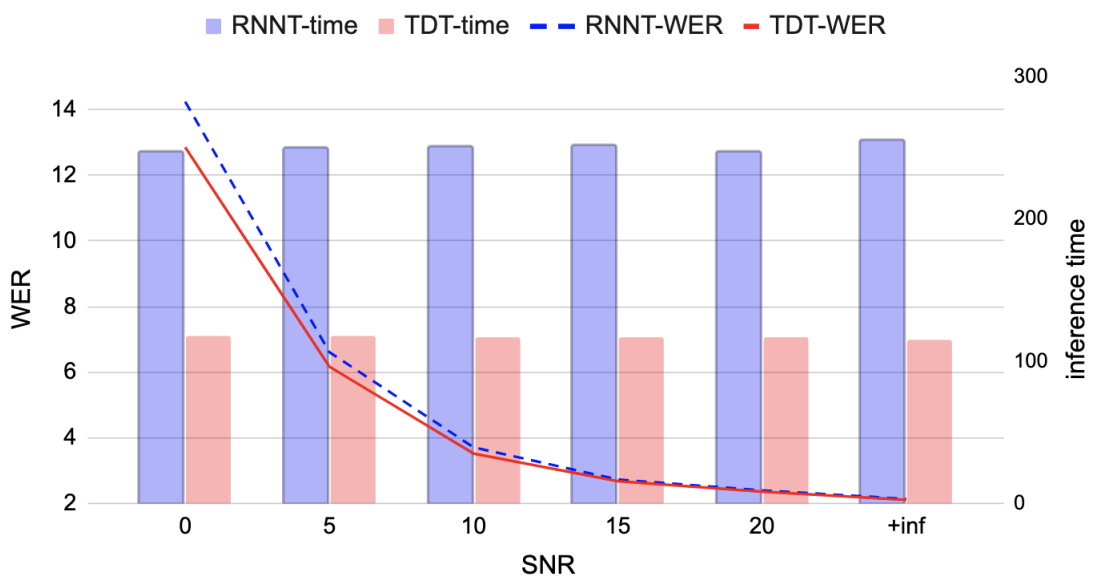

Figure 11. Comparison of TDT VS RNNT on Librispeech test-clean with noise added at different SNRs. TDT model uses 0-8 configuration. The original test-clean dataset is listed as having SNR = +inf. We see while TDT and RNNT achieve similar WER at low noise conditions, TDT is more robust to noise in low SNR settings. We also notice that the inference time stays constant for all TDT models, and this shows having additional noise in the audio does not change how the model emits long durations in inference. This Figure is the same as Figure 6, placed here for easy access.

F. Robustness to Token Repetitions

We notice that RNN-T models often suffer serious performance degradation when the text sequence has repetitions of the same tokens (repetition on the subword level to be exact if using subword tokenizations). Our investigation shows that more training data will not solve this issue, and this is an intrinsic issue of RNN-Ts.

Let's demonstrate the issue with an example here: suppose we have audio with text sequence *two two two two two two two five*. Those words are frequent enough that they are all part of the BPE vocabulary. Let's assume in the audio, frames 0 to frame 40 correspond to all the *two*s, and *five* starts at frame 41; let's also assume we are at audio frame 30, and the model just emitted 5 *two*s during decoding. At this time, the decoder state was updated by feeding in 5 *two*s. At this point, there are two possibilities,

Option 1. If the model emits another *two* between frame 30 and 40, say at frame t, this means,

$$\text{two} = \text{argmax}\big(\text{join}(\text{enc}[t], \text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two}}_{\text{5 twos}})\big) \tag{53}$$

where join, enc, dec represent the computation of joiner, encoder, and decoder of the RNN-T model; for convenience, we assume the argmax operation directly returns the word from the distribution generated by the joiner. dec(a, b, c, d, ...) represents the final output of the decoder, after we sequentially pass a, b, c, d, ... as the decoder input. Since an LSTM decoder[21] rarely has memory beyond 3-4 words, we have

$$\text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two}}_{\text{5 twos}}) \approx \text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two, two}}_{\text{6 twos}}) \tag{54}$$

We would like to point out that having a large number of repetitions isn't the necessary condition for this to happen; sometimes this happens with just two repetitions of the same token. Since *two* is a non-blank emission, t will not get incremented, and the next decoding step operates on the same enc(t). Therefore, when we compute the output distribution of

[21] Or in the case of stateless decoders, if the context size is less than 5, then it should be strictly equal.

the next decoding step, it's likely that,

$$\text{join}(\text{enc}[t], \text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two, two}}_{\text{6 twos}}) \approx \text{join}(\text{enc}[t], \text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two}}_{\text{5 twos}}) \tag{55}$$

Note, since joiner usually has a non-linearity in its computation, this does not strictly follow; although based on what we observed this is usually the case. The equations above are not meant to be rigorous proof but only serve to explain the issue.

Therefore in this next decoding step, it is likely that,

$$\text{two} = \text{argmax}\big(\text{join}(\text{enc}[t], \text{dec}(<\text{bos}>, \underbrace{\text{two, two, two, two, two, two}}_{\text{6 twos}})\big) \tag{56}$$

i.e. the model emits *two* for a second time at frame t. This will likely keep happening for 3 *two*s, 4 *two*s, etc, causing an infinite loop and won't terminate unless some *max-symbol-per-decoding-step* is implemented in the decoding algorithm. In this case, we will end up having a lot of insertion errors in the output in the form of the same token repeating too many times.

Option 2. if the model keeps emitting all blanks until somewhere after frame 41, and then it emits a *five*. Then we would have deletion errors in the decoding output.

TDT is less prone to such repetition issues because the duration output of the model makes it not likely to stay on the same frame at different decoding steps (refer back to Fig. 4, there are very rare cases when duration 0 is emitted). Due to a lack of datasets specifically made with text repetitions, we use NeMo-TTS to generate 100 utterances, which randomly pick three digits from 1 to 9, and repeat each digit 3 - 5 times. We run ASR with different models on this dataset and results are reported in Table 10. We see that TDT models are much more robust than RNN-Ts with repeated speech.

model	WER%
RNNT-LSTM	59.95
RNNT-stateless	64.62
TDT [0-2]	12.59
TDT [0-4]	9.35
TDT [0-6]	6.12
TDT [0-8]	5.78

Table 10. WERs with different Transducer models on TTS generated dataset with repeated digits. This table is the same as Table 8 placed here for easy access.