Final Project: RNN-Transducer-based Losses for Speech Recognition on Noisy Targets

Vladimir Bataev

Contents

List of Figures

List of Tables

1 Project Concept and Motivation

1.1 Overview and Template

In our project, we will train an automatic speech recognition (ASR) system on noisy targets. We start with the template "CM3015 Machine Learning and Neural Networks, Theme 1: Deep Learning on a public dataset," which describes the task of choosing a publicly available dataset and training a deep learning model on it. So, we will work with a neural network-based end-to-end ASR system, using LibriSpeech [2] dataset, a popular academic benchmark. We limit our task to RNN-Transducer [3] systems, which are widely used in production and provide state-of-the-art quality [4] in most cases.

We are going beyond the standard task and focusing our research on making RNN-Transducer systems robust to noisy targets: unlike well-curated datasets, in the industry, the training data contains different errors due to the unreliability of the transcription sources or the inability to transcribe noisy speech accurately. To solve the problem of training on the noisy data, we will analyze the impact of different types of errors in training data on the quality of the RNN-Transducer system and explore different loss modifications to overcome the problem. We will construct the artificial training data by mutating correct transcripts from the LibriSpeech [2] training part, similar to the approaches used in the related work, and try to achieve the best possible quality on the development and test data standard for LibriSpeech.

1.2 Motivation

Training ASR systems usually requires a large amount of well-transcribed audio-text paired data. The process of dataset preparation often calls for filtering out "noisy" transcriptions using some pretrained model, which results in a lower amount of available data for training. It is hard to obtain large, well-transcribed datasets for many languages and scenarios. On the other hand, data with limited transcript quality is widely available. Developing new approaches for working with weakly supervised setups can be beneficial in the following ways:

- (1) making it easier to use non-well-curated datasets;

- (2) improving the quality of ASR models without tricky filtering pipelines for raw data, making an approach fully "end-to-end," and potentially getting benefits from more non-filtered data;

- (3) improving ASR quality for low-resource languages when the data is extremely limited;

- (4) transfer learning scenarios when another (imperfect) model transcribes the data.

1.3 Related Work: Systems

Most of the modern ASR systems use mel filter bank features extracted from the speech signal [3] and learn to map a sequence of feature vectors to the correct sequence of the units derived from the text (e.g., characters, subword units, words, or phonemes). There are three dominating types of end-to-end ASR systems [4]: Connectionist Temporal Classification (CTC) [5], RNN-Transducer (RNN-T) [3] and attention-based encoder-decoder

(AED) [6]. CTC and RNN-Transducer rely on an explicit latent monotonic alignment between the audio and corresponding transcript [7]. CTC is the most straightforward non-autoregressive system, which predicts each text unit independently. RNN-Transducer was introduced as a solution to fix the wholly conditional independence assumption of CTC and consists of 3 parts: an Encoder, which produces the representation of input features non-autoregressively; autoregressive Prediction network; and a Joint network, which combines their output and produces the final prediction [3]. Encoder-Decoder systems with Attention [6] implicitly learn the alignment between the audio and text via the attention mechanism. We are primarily interested in RNN-Transducers since such systems are suitable for streaming by design, widely used in production, and power most of the state-of-the-art monolingual models [7]. Initially, recurrent neural networks (RNNs) were used as an autoregressive prediction (thus, RNN-T is named after RNNs), but other non-recurrent architectures can also be used, e.g., a simple stateless network whose output depends on a fixed number of previous outputs [8].

1.4 Related Work: Models

In ASR, the dominating architecture for the encoder is a Conformer [9, 10, 4], which originates from a transformer block architecture [11] augmented with convolutional modules [9]. The important part is that the original Conformer encoder subsamples the sequence of input features (derived from the audio signal) 4 times, and recently, multiple architectures were proposed to subsample the input features 8 times to provide better speed without performance drop, e.g., Fast Conformer [12]. We will use the Fast Conformer architecture as a basic architecture for our model, focusing on loss modifications rather than changing the architecture.

1.5 Related Work: Losses

CTC is the loss that can naturally be represented [13] with weighted finite state transducers (WFSTs). Due to the existence of the libraries that allow to construct differentiable WFSTs and use them to train deep learning systems, e.g., k2 [14], different modifications of CTC loss were proposed to solve the problem of different errors in training data. Particularly, W-CTC (CTC with Wild Cards) [15] allows missing text at the start and the end of an utterance in transcription, Star Temporal Classification (STC) [16] allows missing labels anywhere, Bypass Temporal Classification (BTC) [17] solves the problem of insertions and substitutions. Recently proposed Omni-temporal Classification (OTC) [18] is a generalized loss that combines all the previous work and proposes a CTC loss modification that is robust to any type of errors in transcripts.

RNN-Transducer robustness to the errors in training data is still an unsolved problem. The recent work about a graph-based framework for RNN-Transducer [1] proposes a generalized solution to develop different loss modifications based on WFSTs and also proposes a W-Transducer loss that can deal with missing transcripts at the start and end of the transcription (similar to W-CTC [15]). Moreover, an autoregressive prediction network can require modifications since its input in training time is a ground truth transcription and can be sensitive to incorrect text. We are planning to increase the complexity of the project gradually, starting from the "under-transcribed" case, when the training texts can contain missing words (deletions) similar to STC [16]. Then, we will explore more complex cases with insertions and substitutions, finally providing a general combined solution.

2 Literature Review

2.1 Introduction

We start our review with the work on the CTC criterion and its modifications. Since both CTC and RNN-T take into account all possible alignments between features extracted from audio and text units, thus it is possible to apply some techniques to both of them. Then, we discuss the RNN-T loss and the relevant work to improve its robustness to errors, along with the structure of the Transducer models and differences between CTC and RNN-T that prevent direct application of CTC-based techniques.

2.2 CTC and Modifications

2.2.1 Connectionist Temporal Classification (CTC) loss

The Connectionist Temporal Classification (CTC) was originally introduced in [5] as a replacement to a "classical" ASR pipeline based on hidden Markov Models (HMMs) and is historically the first so-called "end-to-end" ASR system. The original work proposes a solution for the automatic learning for the alignment between the target units (phonemes in this work, using the TIMIT [19] dataset) and the representation extracted by the neural network from the audio signal. The system utilizes RNNs (particularly bidirectional Long Short-Term Memory (BLSTMs) networks [20]) as its backbone. Mel-Frequency Cepstrum Coefficients are extracted from the input audio every 10ms, which forms the input for the neural network. The output of the network has a softmax activation and is interpreted for each label as a probability of observing the label at the current time frame [5], as in the classification task for each frame. The vocabulary is augmented with a special $\langle blank \rangle$ symbol, and the algorithm considers all possible variants of transcriptions that map to the original text units after removing duplicated predictions and the $\langle blank \rangle$ symbol. The loss is a minus log probability of all possible correct alignments (defined by the rule described above) given the audio features. Thus, training maximizes the log probability for the possible alignments without requiring a forced alignment, which was used in classical HMM-based systems. The paper also proposes an efficient forward-backward algorithm to calculate the CTC loss and gradients and shows the system's efficiency in predicting phonemic transcriptions of utterances.

The paper about CTC loss [5] forms a basis for our research since we plan to apply techniques to modify the possible alignments in the CTC framework to the RNN-Transducer system. The work shows that it is not necessary to have one perfect alignment as a target, but once we construct a mapping between the ground truth target and the inner latent alignment (in the paper - sequences with repeated labels and additional $\langle blank \rangle$ symbols, that are removed in decoding), we can use a full-sum training (taking into account all possible alignments), and the system can effectively solve the ASR task.

2.2.2 Wild-card CTC (W-CTC)

Wild-card CTC (W-CTC) was proposed in [15] to solve the problem when the utterance is partially transcribed, and the transcription can be missing at the start, at the end, or on both sides. The work introduces a simple but efficient modification of CTC criterion computation, using a special "*" (star) symbol that can be prepended to each transcription, and means that at the start of the transcription, any possible sequence (of any size,

including the empty sequence) of symbols can be missing. This allows a simple modification of the dynamic programming algorithm for CTC computation introduced in [5] to handle the alignments not only when the ground truth is entirely correct but also start and end the alignments between text units and features extracted from audio at any part of the audio, but with the assumption, that the part of the audio fully matches the transcription. The authors also use a TIMIT [19] dataset, randomly masking a portion of the start/end of the utterance transcriptions, thus showing the effectiveness of the approach (compared to the original CTC) on partially transcribing ASR data in a synthetic setup. Additionally, the authors validate the effectiveness of W-CTC on Optical Character Recognition (OCR) and Continuous Sign Language Recognition (CSLR) tasks.

The paper is interesting as the first work which introduces a synthetic setup with under-transcribed data to study and develop training criteria for ASR robust to corrupted targets. The proposed W-CTC criterion uses a larger possible amount of alignments but still converges and surpasses CTC even when a small portion of the data is corrupted. Also, the paper shows the application of the techniques to OCR and CSLR tasks, which shows the potential impact of modifications of losses used for ASR on areas outside the speech recognition field.

2.2.3 Star Temporal Classification (STC)

Star Temporal Classification [16] was proposed as a generalization for the previous work when the transcription is only partial, and between any pair of labels, an arbitrary number of words can be missing. The work uses the "*" star token to represent zero or more text tokens and considers alignments between the encoded signal features and a target, where the "*" token is inserted between all labels and also appended to the start and the end of the utterance. Thus, the loss allows the network to output any sequence of labels corresponding to the presented corrupted ground truth text after removing some of the words (the ASR system should emit all the words from the target but can insert other words between them). This approach significantly increases the number of possible alignments and does not allow the training of the neural network directly. The problem is solved by introducing a penalty for the "*" token λ, with exponential decay during training, starting from the high values and decreasing once the network converges. The penalty hyperparameter is also adjusted based on the number of missing words in transcriptions. The authors demonstrate the approach using synthetic data derived from LibriSpeech [2], using partially masked transcripts with different probabilities (up to 70%). The approach is practical even when using greedy decoding, but the authors also show that decoding with an n-gram language model (LM) and rescoring hypotheses with Transformer [11] based LM also improves the system's quality. For the implementation, unlike previous work, the GTN [21] framework for differentiable WFSTs is used, which simplifies the development of the new loss.

This paper is crucial for our work since it allows us to solve the problem with deletions in the transcribed texts for CTC. The critical part is that the direct solution (considering all possible alignments in the loss) leads to failure, and using the adjustable penalty is crucial to solve the issue. Also, the construction of LibriSpeech-based data is an important part. We will consider an analogous approach for investigating similar cases for RNN-T.

2.2.4 Bypass Temporal Classification (BTC)

Bypass Temporal Classification [17] solves part of a weakly-supervised setup with unreliable transcripts: substitutions and insertions. Similar to the previous work, the authors propose CTC target graph modification to allow the alignments when some text units from the target are changed (substitutions) or not used (insertions in the text, thus allowing deletion from the alignment). Remarkably, the ground truth is represented as a linear WFST, where the forward arcs contain ground truth labels, and each arc also presents a parallel "bypass" arc with a penalty, which allows to skip the token by emitting zero or more tokens. The authors also use the penalty with an exponential decay to make the network learn using the increased number of possible alignments. The authors evaluate their solution on both TIMIT [19] and LibriSpeech [2] datasets, constructing the training data with substitutions and insertions separately and combining them. Interestingly, it is shown that for the CTC criterion, the impact of insertions is larger than for substitutions, and with 50% of insertions, it is impossible to train a system with the pure CTC criterion.

This paper solves the remaining piece of the puzzle of training a CTC-based system with partially incorrect transcripts. We will also investigate similar approaches for RNN-T. Moreover, this paper encouraged us to start the exploration of the problem by comparing the impact of different errors on the training criterion behavior and starting our solution from the most disruptive problem.

2.2.5 Omni-temporal Classification (OTC)

The Omni-temporal Classification [18] finally combines the approaches to construct a universal loss to handle all the possible cases of errors. The authors utilize the "bypass" arcs to skip frames, along with self-loops that represent substitutions and insertions, and two separate penalties for them following the previous work. The authors used synthetic data generated from the train-clean-100 subset of LibriSpeech [2] and also trained a system on the original subsets from LibriVox [22] that were originally used to construct the LibriSpeech data, using segmentation and filtering with pretrained models, as described in [2]. The combination of the losses leads to significant improvements for all types of errors and allows training of the ASR system even on the original unsegmented data with errors without a cleaning pipeline.

This work contains a detailed exploration of the impacts of different penalty values on the quality of the network, which can provide some insights for working with similar alignment-based modification approaches for RNN-T. The code is published, and we can try to reproduce the setup if necessary. Moreover, we can try this criterion as a part of the potential solution with hybrid CTC-RNN-T models for a "CTC head," which we will discuss further. Moreover, we found it curious that despite the beneficial impact of the loss when training ASR system with unfiltered raw data from LibriVox, the gap still exists for the carefully filtered LibriSpeech data (e.g., authors report [18] 12.5% vs 8.2% WER on test-other for these setups), which can indicate that the problem is still not fully solved even for CTC criterion, and multi-stage pipeline is still essential to obtain the best performance.

2.3 RNN-Transducer and Modifications

2.3.1 RNN-T loss

Recurrent Neural Network Transducer (RNN-Transducer, RNN-T) [3] was developed as an improvement of the CTC [5] systems to overcome problems related to conditional inde-

pendent assumption. The system contains three components: (1) transcription network, which is usually referred to as an "Encoder"(e.g., [23]), that operates on the features derived from audio and is similar to the one used in CTC system; (2) prediction network, that outputs the predictions based on the previously decoded symbol in inference time, and utilizes a ground truth text in training time; and (3) a combination of the outputs of the previous two networks that makes final predictions, which is usually referred as a Joint network in the literature [23]. The prediction network is autoregressive and provides a crucial improvement for the ASR system. The system also uses a $\langle blank \rangle$ symbol, but the meaning differs from CTC: this symbol means the system should end decoding the current frame and transition to the next frame from the encoder. Unlike CTC, repeated symbols are not allowed, but the advantage is that the system can predict more than one text unit for each frame. The RNN-T loss function takes into account all possible monotonic alignments between encoder and prediction network outputs, which makes the system similar to CTC with implicit alignment learning.

The crucial difference for the RNN-T system is that the prediction network is autoregressive (usually uses LSTM as a backbone), making it significantly more challenging to deal with imperfect transcripts. Since ground truth text is used during training (so-called "teacher forcing" algorithm), corrupted tokens can prevent the correct alignment learning. The autoregressive nature also prevents applying CTC-based approaches directly to RNN-T and may require modifications of the network itself along with the training criterion.

2.3.2 Graph-based RNN-Transducer framework

The recent work about RNN-Transducer modifications [1][1] brings the connection between WFSTs and Transducer architecture, allowing representing the RNN-T loss computation with the graph, where the arc weights are taken from the Joint network output, and thus the direct application of graph algorithms, including full-sum alignment learning, are possible. The work presents two approaches for representing original and modified RNN-T graphs, either as a direct grid ("Grid-Transducer") or as a composition of acoustic and textual schemas ("Compose-Transducer"), which after the composition and connect operations exactly matches the Grid-Transducer representation ("connect" operation removes the states that do not belong to any path from the start and the end state; it is optional for loss computation since such states do not affect the alignment probabilities since not belonging to any alignment). The composition is slower to compute but allows for the development of losses using graphs that can be easily debugged visually. The authors build the framework on top of k2 [14] library for differentiable WFSTs and show that the loss computation can be as efficient as the optimized CUDA-based code. Moreover, the authors present a W-Transducer, which can solve the problem of deletions at the start and the end of the utterance, similar to the W-CTC discussed above. The graph for the loss uses two groups of skip-connections from the start of the time grid to all other time steps and from each time step in the grid to the last, which allows to use the alignments where some frames at the start and the end are skipped, but the network should emit the full training text. The effectiveness of this approach was demonstrated on LibriSpeech data by randomly removing 20% and 50% of the labels from text from both sides of each utterance.

In our work, we will use the proposed WFST framework for training RNN-T on noisy targets to simplify the development of the losses. W-Transducer solves the easiest case when the autoregressive prediction network does not use a corrupted input and thus

---

[1]Disclaimer: the author of the Final Project participated in the development of this approach and is a co-author of the paper.

can remain unmodified. We are planning to work with more complex cases, and such modifications can be required along with the loss customization. We will discuss this approach in more detail in Section 3.6.

2.3.3 Stateless RNN-T

The work [8] proposes "RNN-T with StateLess Prediction Network (RNNT-SLP)," revising the need for recurrent networks for the prediction network part of the RNN-T system. The authors questioned the popular opinion that the prediction network behaves similarly to the language model in traditional ASR systems. They tried pretraining the recurrent network on text-only data to predict the next symbol and initializing the prediction network with pretrained weights and found that such pretraining does not lead to any improvement. Moreover, replacing the prediction network with a simple "stateless" module, in which prediction is based only on the last symbol (which is similar to 2-gram LM), leads to a comparable overall system performance with the RNN-based system.

Despite there are other works that show that the prediction network can behave like a language model, especially in a factorized architecture [24, 25], we are interested in this work showing the possibility of using simpler networks for this part of the model. Since a stateless prediction network can be significantly more robust for corrupted targets, in which prediction is dependent only on a small context, we consider this work an important option for changing the prediction network architecture.

2.3.4 Hybrid CTC-Transducer models

As a last item of our review, we will discuss the hybrid architecture, which uses both RNN-T and CTC losses, proposed in [26]. The work proposes training the neural transducer with an auxiliary CTC loss on top of the encoder (as a separate "head"), combining the systems for further accelerating the decoding: if the CTC head predicts the blank label, such frame can be skipped in decoding, resulting in a significant inference acceleration with tiny quality degradation. The work [27] proposes the use of such hybrid systems to accelerate not only inference but also the training speed, using more lightweight CTC head prediction to compute RNN-T loss with the smaller number of possible alignments (pruning RNN-T loss lattice).

This work can be relevant for our research due to the discussed above OTC criterion, which is robust to noisy targets: in a hybrid training setup, we can use the prediction of the additional head, trained with the OTC loss, to identify corrupted tokens and thus modify the input for prediction network or the loss based on this information.

3 Project Design

In our project, we plan to investigate the behavior of the RNN-Transducer architecture in a weakly supervised setup when part of the data is corrupted and modify the RNN-T loss to improve the system's quality. We will consider loss modification techniques that do not require any change in the overall neural architecture and decoding algorithms to make them compatible with current production systems.

3.1 Objectives

We define the following objectives for our work:

- (1) Investigate how using partially incorrect transcripts impacts the quality of the RNN-Transducer system, separating deletions, substitutions, and insertion cases

- (2) Investigate techniques that allow to improve the performance of the system in the conditions described above

- (3) Combine the solutions provided in (2) to solve the general case of training the ASR Transducer-based system with unreliable transcript and evaluate the final solution when it is unknown what type of errors the data contains.

3.2 Metrics

The key metric for assessing the quality of an ASR system is word error rate (WER) [28]. The additional metric commonly used to assess the speed of the ASR system is a real-time factor (RTF), which indicates how much audio the system can process given the fixed time, and usually, the tradeoff between speed and quality is important when considering different models. In our project, we focus only on modifications that have a minor impact on the inference speed after the model is trained; thus, we report only WER in our experiment.

The word error rate is a specification of a Levenshtein distance [28, 29], defined for two sequences of words (hypothesis and ground truth), and is calculated as a number of substitutions (SUB), insertions (INS) and deletions (DEL) divided by the number of ground truth (correct) words.

$$WER = \frac{SUB + INS + DEL}{CORRECT}$$

Lower WER means that the system makes more accurate predictions. We will also use the term "accuracy," usually defined as $accuracy = 1 - WER$ (higher accuracy value means better prediction).

Since we are working with the conditions that show the degradation of the standard ASR training pipeline, we introduce two additional metrics. WER difference (WERD) indicates the system degradation and is a difference between the WER that the system achieves in a particular setup and the WER of the baseline on the non-modified (original) data.

$$WERD = WER_{modified\_data} - WER_{original\_data}$$

We are interested in minimizing the degradation of the system, thus defining the relative improvement of the system as WERDR.

$$WERDR = \frac{WERD_{RNN-T} - WERD_{Proposed}}{WERD_{RNN-T}}$$

3.3 Data

The primary data for the project is a LibriSpeech [2] corpus, which consists of 3 subsets for training data (960 hours total), two development sets (*dev-clean* and *dev-other*, 5.4 and 5.3 hours respectively), and two test sets (*test-clean* and *test-other*, 5.4 and 5.1 hours). We will use the full training part to generate artificial training data by corrupting the texts

with artificial deletions, substitutions, and insertions. We will use the *dev-other* set for validation during training and choosing the optimal checkpoints, and we will finally assess our models on the *test-other*. As it is common in the ASR research, we will report WER for all development and test sets.

3.4 Backbone Encoder Models

Currently, the dominating architecture [7] for the encoder is the Conformer [9] model. The original Conformer-encoder processes the input sequence of vectors and produces the representation 4 times smaller by the time dimension (4x subsampling). For our initial experiments, we will use Fast Conformer [12] (114M parameters), which subsamples the input by the factor of 8 without accuracy degradation by using depth-wise separable convolutions [30], which allows faster training and inference.

3.5 Tools and Frameworks

We are planning to use NeMo [31] framework for experiments, which is based on PyTorch-Lightning [32] for easiness of extending the models, and train them on clusters. NeMo provides stubs for ASR models and capabilities to use WandB [33] platform for experiment management. When required, we will use pure PyTorch [34] and k2 [14] library to modify losses and prediction network, also using a WFST framework for RNN-Transducer [1] implemented in NeMo.

3.6 RNN-Transducer Details and Graph-based representation

RNN-Trasducer [3] model schema is shown in Fig. 1. The model consists of three parts. **Encoder** neural network transforms a sequence of input feature vectors derived from an audio into a latent representation - a sequence of vectors $\langle e_0, e_1, ..., e_{t-1} \rangle$ with the length of T. The number of output vectors is usually smaller than the input due to applying strided convolutions or pooling layers along the time axis, which reduces the output size by a fixed factor (subsampling) and makes the model more computationally efficient. The **Prediction network** (usually a recurrent network) in training time takes a ground truth sequence of text units as an input padded with a special start-of-sequence symbol $\langle SOS \rangle$ (in practice usually $\langle blank \rangle$ symbol is reused for this purpose) and produces a latent representation – sequence of vectors $\langle p_0, p_1, ..., p_u \rangle$ with the length of $U + 1$ (U is the number of text units in the text representation). The **Joint network** combines all combinations of e_i with p_j vectors and produces a 3-dimensional tensor for the size $Tx(U+1)xV$, where V is a vocabulary size augmented with the $\langle b \rangle$ ($\langle blank \rangle$) symbol (in practice, the 4-dimensional tensor is used due to additional batch dimension). Thus, $j_{i,j}$ represents the vector produced from a combination of p_i and p_j. In practice, the Joint network is relatively simple and computes the sum of vectors, non-linearity (e.g., ReLU), and a projection into the space of size V, (e.g., $j_{i,j} = Project(ReLU(e_i + p_i))$). If the vectors e_i end p_j are of different sizes, they are firstly projected to the space with the same dimension. The Softmax activation produces a distribution over the vocabulary units, and the dynamic programming algorithm is used to compute the probability over all possible paths in the graph (in the code, all computations are produced in log scale for numerical stability purposes). Each path represents a possible alignment, where blank symbols can be inserted between labels (e.g., for the Fig.1 1 "C $\langle b \rangle$ A T $\langle b \rangle$ $\langle b \rangle$ $\langle b \rangle$" is the possible path for the target

text "CAT" represented as "C," "A," and "T" units). The number of $\langle b \rangle$ labels is the number of frames the Encoder produces. The minus log probability of all possible alignments forms an RNN-T loss value.

For **greedy decoding**, since the ground truth text is unknown, a nested loop is used: for each encoder vector, the algorithm sequentially obtains the next prediction with the maximum probability, starting from $\langle SOS \rangle$ symbol, and computes the Prediction network output for the new decoded symbol, combining with the current encoder vector and computing Joint network output. Once the $\langle b \rangle$ symbol is found, this symbol is not fed into the Prediction network, but the inner loop stops, and the decoding process starts for the next encoder vector. More complex decoding algorithms exist, but greedy decoding is widely used in practice due to the best speed and near-optimal quality in most scenarios [7], so we are focusing on this approach.

The computation of the RNN-T loss can be represented with **WFSTs**, as shown in [1]. The original work presents the approach more theoretically, and here we will discuss the practical implementation, which is a basis for our work. A weighted finite state transducer is a graph with states and arcs, whose arcs represent transitions from input to output labels with a corresponding weight. In k2 [14] library, it is allowed to store an arbitrary number of labels (not only an input/output label for each arc). Thus, we use a tuple of labels as input labels for some graphs in our representation 2. We can construct a computational graph using 3 labels for each transition: output (ground truth) label, unit index (in a sequence of ground truth units), and encoder vector index, see 2c. Given these labels by using "index select" [2] operation, we can populate a lattice with the weights from the Joint network output, and apply a differentiable computation of the full-sum loss[3]. This is a "Grid-Transducer," described in [1]. To simplify the development, we can construct two schemas ("Compose-Transducer"). Unit schema 2a represents text units, and each transition has a tuple of input symbols $(Unit : Unit\_Index)$ and an output symbol $Unit$. $\langle b \rangle$ units represent self-loops, and other units represent transitions over the ground truth text. The time schema 2b is a simple linear graph representing transition over time and self-loops for each state with all the vocabulary symbols as inputs and time index as output. It is much easier to visually debug the representation with the separated schemas, and their composition matches the final lattice 2c. We are planning to experiment with the separated schemas to make the RNN-T loss more robust to corrupted targets, but for the final stage, we are planning to provide an efficient code for lattice construction.

3.7 Plan

3.7.1 Initial Project Plan

Our work initial work plan published in the "Draft Report" (Midterm) is shown in Fig. 3.

(a) Minimal RNN-T Implementation. After initial preparation, which was already done before finishing this report (initial exploration, project proposal, literature review), we will implement a minimal codebase to experiment with RNN-T. During our exploration, we found that RNN-T models in NeMo provide a lot of functionality but are not easy to extend. So, we will implement the lightweight pipeline with the following capabilities:

- Lightweight Joint and Prediction networks

- Greedy Decoding

---

[2] https://pytorch.org/docs/stable/generated/torch.index_select.html

[3] https://k2-fsa.github.io/k2/python_api/api.html?highlight=get_tot_scores#k2.Fsa.get_tot_scores

Figure 1: RNN-Transducer Schema

- Full model training pipeline

- Compatibility with NeMo encoders (including Conformer-based encoders as described above), since we are not planning to customize the encoder part.

Then, we plan to train the baseline on the original LibriSpeech data with a Fast Conformer encoder to check that our model can achieve comparable quality with the native implementation of RNN-T in NeMo.

(b) Impact of imperfect transcripts. We will generate the data derived from LibriSpeech with 20% and 50% of deletions, substitutions, and insertions separately (6 sets in total) and train the models to study the model's behavior in such conditions.

(c) "Deletions" case. Our preliminary studies showed that the deletions in the transcripts are the hardest case for RNN-T. We will try to solve this case by modifying the loss function. At this stage, we focus on improving the performance and not trying to find the perfect hyperparameters and/or provide the fastest implementation for the loss function.

(d) "Insertions" and "substitutions." We will explore different approaches discussed in the literature review to improve the system's performance in such conditions.

(e) Combined Solution. Combining the solutions (c) and (d) will allow for solving the universal problem of partially correct transcripts. We will generate additional training data for cases with all types of errors.

(a) RNN-Transducer Unit Schema. Labels:
$(text\_unit, unit\_position) : text\_unit$

(b) RNN-Transducer Time Schema. Labels:
$text\_unit : frame\_number$

(c) RNN-Transducer Lattice. Labels: $(text\_unit, unit\_position) : frame\_number$

Figure 2: WFSTs for RNN-Transducer, following [1]

(f) Conformer Medium (4x subsampling). At this stage, we will apply our solution to the Conformer Medium model to study its behavior when the encoder reduces the input 4 times.

(g) Fast loss implementation. This task focuses on the quality and speed of our code. We will provide a clean and fast "ready to use" solution for our loss and prediction network modifications.

(h) Study hyperparameters in detail. This task will allow us to get insights about the hyperparameters of the systems proposed in (c)-(e).

3.7.2 Project Plan Reflection

The sections (a)-(b) were crucial for our project but imposed minimal risks due to well-explored existing solutions. The most critical risks came from (c) and (d) cases since, according to our knowledge, no solutions existed for such tasks. To mitigate these risks, we also considered using a hybrid CTC-Transducer architecture with OTC loss, which can solve the problem, at least for the CTC head, and we can use its predictions to train the Transducer part instead of corrupted ground truth labels. Combined solution (e) was a key part of finalizing our project. We considered parts (f)-(h) as a fair improvement but not an essential contribution to our work and planned to focus on them only after finishing the main part.

In our final stage, we elaborated a successful solution for all discussed cases, provid-

Figure 3: Project Plan (Gantt Chart)

ing a "drop-in" replacement for the RNN-Transducer loss for the cases without changing the model architecture at all. We also provided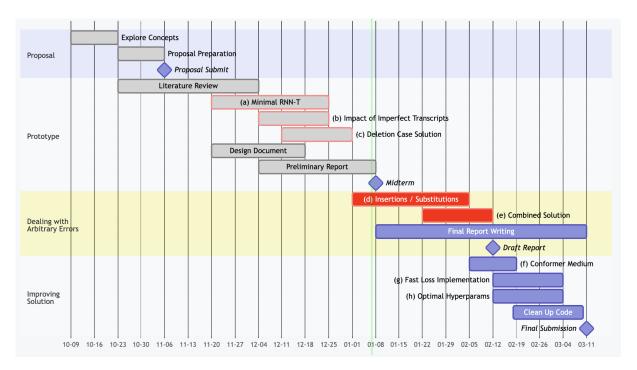 a fast implementation for the losses. Due to a lack of computational resources and difficulties training the system within the novel setup, we focused more on exploring parameters for Bypass Transducer loss, as discussed in Section 5.4. We omitted experiments with Conformer Medium (f) and left them for future work.

4 Implementation

The implementation is published in the **GitHub repository**: `https://github.com/artbataev/uol_final`. In this section, we describe the implementation with the links to the original files. Additional visualization of the produced lattices for all proposed losses can be found in the Jupyter Notebook `https://github.com/artbataev/uol_final/blob/main/notebooks/Loss_Demo.ipynb`.

4.1 Project Implementation Overview

The main goal of the project is to provide a solution to deal with different types of errors in the RNN-Transducer framework. We want to make our models comparable and compatible with the publicly available state-of-the-art models and want further to propose solutions to the NeMo [31] framework. So, we reuse components from NeMo, focusing on customization and modifications of the necessary parts.

Firstly, we make a minimal necessary code of RNN-T for our experiments, providing implementation containing the RNN-T model and customizable Joint and Prediction networks. We reuse Conformer [9] blocks from NeMo [31] for the Encoder network.

The Prediction network is a 1-layer LSTM with 640 hidden units, implemented in `MinPredictionNetwork` class. The Joint network, as discussed in Section 3.6, applies

two projections of the output of Encoder and Prediction networks (2 linear layers) to the dimension of 640, sums the vectors, applies ReLU non-linearity, and projects the output (one more linear layer) into 1025-dimensional space (1024 BPE units and $\langle b \rangle$). It is implemented in `MinJoint` class.

We also implement a greedy decoding algorithm for evaluation in `min_rnnt/decoding.py`.

In our experimental setup, we are following the Fast Conformer [12] training pipeline[4]. The encoder has 108.7M parameters, prediction network 3.9M, and Joint 1.4M (totally 114M).

4.2 Data Preprocessing

We preprocess LibriSpeech [2] data and apply speed perturbation with rates 0.9 and 1.1 (3x audio data), using the published preprocessing script[5] to make our pipeline comparable with published models.

We use log-mel filterbanks extracted from audio every 10ms with the window 25ms and apply SpecAugment [35] in training. We use the vocabulary of 1024 BPE [36] tokens extracted using SentencePiece [37] library for text units.

4.3 Model and Training Pipeline

We set up training of our model for 200 epochs using AdamW [38] optimizer with Cosine annealing [39] learning rate schedule with a linear warmup for 40 epochs and the maximum learning rate of $5e-3$. For experiments except for the baseline, we stopped training the model after 60 epochs since we are interested in the relative difference in model quality, and achieving the best possible accuracy is not our priority at this stage. We are reporting the results for the best checkpoint chosen on the *dev-other* validation set. For all experiments, we maintain a global batch size of 2048. We are training models on clusters using NVIDIA A100 (mixed-precision with bfloat16) and V100 GPUs (float32 full-precision), and depending on the availability of the resources varying local batch size from 8 to 32 to fit into memory and adjusting gradient accumulation to make the global batch size constant. We did not observe any difference in quality for a fixed global batch size when using an arbitrary number of nodes, varying local batch size, and using mixed or full precision. So, we are not reporting these details for each experiment.

4.4 Proposed Losses

In our work, we propose three modifications of the RNN-T loss:

- Star-Transducer to dial with arbitrary deletions

- Bypass-Transducer to solve the case of insertions

- Target-Robust-Transducer, which is the combination of the previous modifications, allows to mitigate the problems of substitutions in target texts and also can be used as a universal loss when the type of errors is unknown.

---

[4]github.com/NVIDIA/NeMo/blob/v1.21.0/examples/asr/conf/fastconformer/fast-conformer_transducer_bpe.yaml

[5]https://github.com/NVIDIA/NeMo/blob/v1.21.0/scripts/dataset_processing/get_librispeech_data.py

The modifications are implemented in `minrnnt/losses` subpackage. All the classes follow Graph-RNNT [1] framework and inherit `GraphRnntLoss` class from the NeMo [31] framework and reuse its methods. The implementation uses k2 [14] library. As described in Section 3.6, the computational lattice can be constructed as a composition of temporal and unit schemas, implemented in `get_temporal_schema` and `get_unit_schema` respectively. The faster implementation constructs the lattice directly in `get_grid` method. In the initial development, we used the composition and then made the `get_grid` implementation as a faster option. We also customize the `forward` method to assign appropriate scores to the arcs corresponding to special tokens, as described below.

For all losses, we add unit tests (in `tests` directory) to make the sanity check for the following:

- Graphs produced by composition of the temporal and unit schemas are equivalent to the graph produced by `get_grid` method.

- When the weight of special arcs is $-\infty$, this is the equivalent of removing such arcs from a computational graph ($e^{-\infty} = 0$, such a transition does not contribute to loss computation); and the loss should be equivalent to original RNN-T loss. This is tested by comparing the loss value and gradient based on random input for the proposed loss and etalon RNN-T implementation.

The graph construction is debugged visually in the Jupyter Notebook [40] using automatic visualization from the k2 [14] library with GraphViz package [41].

4.4.1 Star-Transducer (Star-T)

We propose a simple but effective modification of the RNN-T loss computational graph to solve the problem of deletions. Star Transducer takes into account, along with the alignments with blank labels, the sequences when the blank label is substituted with a special "skip frame" $\langle sf \rangle$ symbol, which can be viewed as an allowance to skip frames produced by the encoder in training time. This approach is similar to the "*" token used in Star Temporal Classification [16] loss for CTC. For such frames, the transcription is missing in the ground truth, and the core idea was to allow skipping such frames when considering all possible alignments for loss computation. We add parallel arcs to those with $\langle b \rangle$ label to achieve this, as shown in 4c. Unlike other arcs, the weight for this arc is a hyperparameter and assigned directly after populating the lattice with other weights.

The Star-Transducer loss is implemented in the `GraphStarTransducerLoss` class.

4.4.2 Bypass-Transducer (Bypass-T)

For dealing with insertions, we propose a modification of the RNN-T computational graph, adding arcs with a special "skip token" $\langle st \rangle$ symbol, inspired by Bypass Temporal Classification [17] approach. These arcs are parallel to the arcs with tokens. This means that the loss can consider alignments where some tokens are skipped. Fig. 5 shows the temporal and unit schemas and the full constructed lattice.

The Bypass-Transducer loss is implemented in the `GraphBypassTransducerLoss` class.

In our experiments, we found that assigning constant weight similar to the Star-Transducer approach does not work. With small absolute values (e.g, 0 or -3) the model is prone to produce deletions, and the system behaves worse than the original RNN-T. With high

(a) Star-Transducer Unit Schema. Labels:
$(text\_unit, unit\_position) : text\_unit$

(b) Star-Transducer Time Schema. Labels:
$text\_unit : frame\_number$

(c) Star-Transducer Lattice. Labels: $(text\_unit, unit\_position) : frame\_number$

Figure 4: WFSTs for Star-Transducer. $\langle sf \rangle$ is a special symbol indicating skipping the frame.

absolute weight values (e.g., -20), such transitions do not contribute to the loss computation: e^{-20} is close to zero, and the loss is close to the original RNN-T. Similar to the approaches applied in the paper about BTC [17], we apply a schedule to the penalty weight ($skip\_token\_penalty$), combined with the probability derived from the output of the Joint network. For the probability we considered different options ($skip\_token\_mode$ parameter in the implementation):

- "constant": only penalty constant, similar to one used in the Star-Transducer loss.

- "mean": mean probability for all labels (in log scale) excluding blank, similar to BTC [17].

- "max": maximum log-probability for all labels excluding blank.

- "maxexcl": maximum of the log probabilities of all labels excluding blank and ground truth labels.

- "sumexcl": logarithm of the sum of the probabilities (in log scale) of all labels excluding blank and ground truth labels.

We found that the "mean" and "max" options were not better than the original RNN-T loss. The "maxexcl" option was the first working solution used in the Preliminary Report. The intuition behind the "sumexcl" option is to assign the "unused" probability of outputs.

18

(a) Bypass-Transducer Unit Schema. Labels:
$(\mathit{text\_unit}, \mathit{unit\_position}) : \mathit{text\_unit}$

(b) Bypass-Transducer Time Schema. Labels:
$\mathit{text\_unit} : \mathit{frame\_number}$

(c) Bypass-Transducer Lattice. Labels: $(\mathit{text\_unit}, \mathit{unit\_position}) : \mathit{frame\_number}$

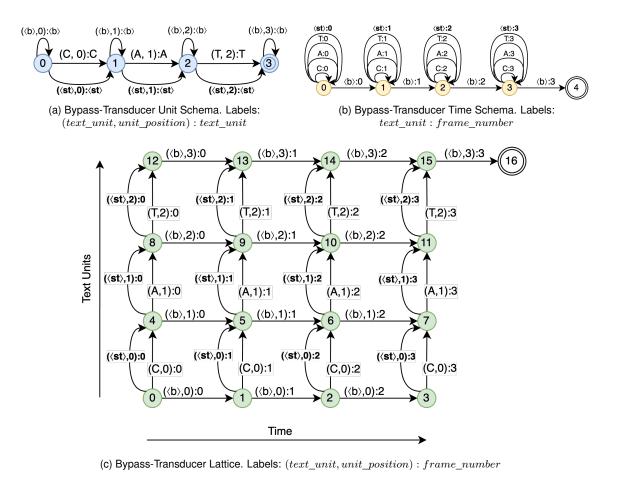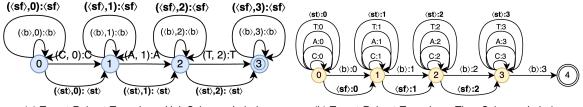Figure 5: WFSTs for Bypass-Transducer. $\langle st \rangle$ is a special symbol indicating skipping the token.

The "sumexcl" option allows for the alignments to be considered when the network outputs a high probability for any token other than the target as "appropriate." We found that the "sumexcl" option outperforms other cases, as discussed further in Section 5.4.

4.4.3 Target-Robust-Transducer (TRT)

Target-Robust-Transducer loss is a combination of Star-Transducer and Bypass-Transducer. We add both types of arcs that allow skipping frames and tokens, as shown in Fig. 6. It is worth mentioning that assigning $-\infty$ weight for "skip frame" arcs makes the loss identical to Bypass-Transducer (skipping frames is not allowed in this case), and $-\infty$ weight for "skip token" arcs makes it similar to Star-Transducer (skipping tokens is not allowed). This makes this loss a universal replacement for the previous two modifications (but the system makes more computations since the arcs are still present, even with $-\infty$ weight). We also test this behavior in unit tests.

The Target-Robust-Transducer loss is implemented in the class `GraphTargetRobustTransducerLoss`. The implementation combines hyperparameters and code for `GraphStarTransducerLoss` and `GraphBypassTransducerLoss`.

(a) Target-Robust-Transducer Unit Schema. Labels: $(text\_unit, unit\_position) : text\_unit$

(b) Target-Robust-Transducer Time Schema. Labels: $text\_unit : frame\_number$

(c) Target-Robust-Transducer Lattice. Labels: $(text\_unit, unit\_position) : frame\_number$

Figure 6: WFSTs for Target-Robust-Transducer. $\langle sf \rangle$ is a special symbol indicating skipping the frame. $\langle st \rangle$ is a special symbol indicating skipping the token.

Table 1: Baseline on LibriSpeech, WER [%].

Source	dev		test	
	clean	other	clean	other
NeMo	2.0	5.0	2.2	5.0
Ours (200 epochs)	2.1	4.9	2.2	5.1
Ours (60 epochs)	2.6	6.8	2.8	6.8
Ours (100 epochs)	2.4	5.9	2.5	6.0

5 Evaluation

5.1 Baseline

The results for our implementation are shown in Table 1. For comparison, we use a publicly available Fast Conformer checkpoint[6] trained on LibriSpeech data for 200 epochs. Our implementation provides results comparable to those of the state-of-the-art pipeline.

Table 2: Training RNN-T on data with errors, Fast Conformer, 60 epochs, WER [%].

Type	Corrupt %	dev		test		WERD↓
		clean	other	clean	other	
	–	2.6	6.8	2.8	6.8	
DEL	20%	4.3	9.9	4.7	10.3	3.5
DEL	50%	79.2	81.7	80.3	81.4	74.6
SUB	20%	4.0	9.4	3.9	9.7	2.9
SUB	50%	11.5	23.2	11.2	23.8	17.0
INS	20%	4.0	10.3	4.2	10.2	3.4
INS	50%	5.1	12.7	5.3	13.5	6.7

So we can proceed further and investigate the system behavior of corrupted targets. Additionally, we show the results for 60 and 100 epochs (also using the best checkpoint selected on `dev-other` for these epochs): to save computational resources, we evaluate different cases, training the models for 60 epochs, and for the final case with arbitrary errors we train the system for 100 epochs.

5.2   Error Impact Exploration

To explore the training pipeline on partially incorrect transcripts, we generate additional training data sets by mutating the original training texts with the mutation probability p_m of 20% and 50%. We are randomly removing words for the "deletions" case. We use randomly selected words from the training vocabulary for substitutions and insertions, substituting/inserting words with the probability p_m.

Table 2 shows the training results on corrupted transcripts. With a small amount of corruption, all cases lead to system degradation, but the difference between cases is tiny (from 2.9% to 3.5% absolute WER degradation on test-other). We found that the deletions are most disruptive for the high corruption rate of 50%, and the ASR system can not achieve a reasonable quality (81.4% WER on test-other compared to 6.8% on original data). Thus, we prioritized the work with this part of the problem. Substitutions are the next hard case for RNN-T, which is the opposite of observations for the behavior of CTC systems in [18].

5.3   Dealing with Deletions: Star Transducer

For the setup when the ground truth transcripts contain deletions, we apply Star-Transducer loss as a drop-in replacement for the RNN-T loss. The results of training the model are shown in Table 3. In both scenarios, we can close the gap between the baseline for more than 70%: 77.1% WERDR for 20% deletions and 94.4% for 50%. We found that the training is stable even without penalty, but applying the small constant penalty for the "skip frame" transition (-0.5) improves the quality when the number of deletions is low. We were surprised that such a simple solution works and that modifying the autoregressive prediction network is unnecessary. This can mean that the encoder and joint are more sensitive to incorrect transcripts than the prediction network.

Table 3: Star-Transducer (Star-T) Loss for deletions, Fast Conformer, 60 epochs, WER [%].

Loss	Skip Weight	DEL %	dev		test		WERD↓	WERDR↑
			clean	other	clean	other		
RNN-T	-	–	2.6	6.8	2.8	6.8		
RNN-T	-	20%	4.3	9.9	4.7	10.3	3.5	
Star-T	0	20%	3.9	8.2	4.3	8.5	1.7	51.4%
Star-T	-0.5	20%	3.1	7.5	3.4	**7.6**	0.8	**77.1%**
RNN-T	-	50%	79.2	81.7	80.3	81.4	74.6	
Star-T	0	50%	5.1	10.6	5.2	**11.0**	4.2	**94.4%**
Star-T	-0.5	50%	5.4	12.4	5.9	12.5	5.7	92.4%

Table 4: Bypass-Transducer (Bypass-T) Loss for insertions, Fast Conformer, 60 epochs, WER [%].

Loss	Skip Weight	INS	dev		test		WERD↓	WERDR↑
			clean	other	clean	other		
RNN-T	-	–	2.6	6.8	2.8	6.8		
RNN-T	–	20%	4.0	10.3	4.2	10.2	3.4	
Bypass-T	-6	20%	3.0	7.5	3.3	7.9	1.1	**67.6%**
RNN-T	–	50%	5.1	12.7	5.3	13.5	6.7	
Bypass-T	-6	50%	3.9	10.3	4.3	10.5	3.7	44.8%
Bypass-T	-5	50%	3.6	9.2	4.0	9.4	2.6	**61.2%**

5.4   Dealing with Insertions: Bypass Transducer

For insertions case, we apply the Bypass-Transducer loss described in Section 4.4.2. The results are shown in the Table 4. The transition weight for the "skip token" arcs is a sum of the constant weight and the total log-probability of all outputs excluding blank and target ("sumexcl" option), as discussed in the Section 4.4.2. The training starts with a constant weight of -20.0 and is adjusted with the decay after each epoch (starting 3rd epoch): $weight_{next} = min(max\_weight, weight * decay)$. We use $decay = 0.9$ for all experiments. Table 4 also reports the maximum constant penalty applied in training. The proposed loss can restore more than 60% of the system quality for the texts with insertions. Further evaluation of the "sumexcl" and "maxexcl" options for assigning the weight can be found in Appendix B.

5.5   Dealing with Substitutions: Target-Robust-Transducer

We apply Target-Robust-Transducer for the case with substitutions since any "substitution" can be viewed as a combination of "deletion" and "insertion." The results are shown in Table 5. In the preliminary experiments, we found that assigning low absolute values for weights (0 for skip frame as in Star-Transducer, and -5 or -6 for skip token as in

Table 5: Target Robust Transducer (TRT) Loss for substitutions, Fast Conformer, 60 epochs, WER [%].

Loss	Skip token,frame	SUB	dev clean	dev other	test clean	test other	WERD↓	WERDR↑
RNN-T	-	–	2.6	6.8	2.8	6.8		
RNN-T	-	20%	4.0	9.4	3.9	9.7	2.9	
TRT	-8,-0.5	20%	3.4	8.2	3.8	8.5	1.7	**41.4%**
RNN-T	-	50%	11.5	23.2	11.2	23.8	17.0	
TRT	-8,-0.5	50%	8.2	16.3	8.5	17.0	10.2	45.3%
TRT	-8,-1	50%	6.9	16.0	7.1	15.8	9.0	**47.1%**

Table 6: Target Robust Transducer (TRT) Loss for arbitrary errors, Fast Conformer, 60 and 100 epochs, WER [%]. 50% of data is corrupted, using 15% for each class of errors.

Loss	Epochs	ERR	dev clean	dev other	test clean	test other	WERD↓	WERDR↑
RNN-T	60	-	2.6	6.8	2.8	6.8		
RNN-T	60	50%	4.2	10.2	4.3	10.1	3.3	
TRT	60	50%	3.3	8.0	3.6	8.4	1.6	**51.5%**
RNN-T	100	–	2.4	5.9	2.5	6.0		
RNN-T	100	50%	3.5	9.4	3.8	9.5	3.5	
TRT	100	50%	2.9	7.0	3.2	7.0	1.0	**71.4%**

Bypass-Transducer) results in fast model overfitting, but when the penalty is more significant, the training is stable. Since the loss can skip both frames and tokens, applying a more significant penalty is reasonable. We use -8 for skip frame penalty and the "sumexcl" option for assigning the weight, which we found the best when experimenting with Bypass-Transducer, along with the penalty schedule, as discussed above. With the proposed loss, we can restore more than 40% of the system degradation on `test-other`: we achieve WERDR of 41.4% for 20% substitutions and 47.1% for 50%.

5.6 Arbitrary Errors

For evaluating the system trained with Target-Robust-Transducer loss, we construct the extra training data by corrupting only 50% of all utterances. For each corrupted utterance, we apply random substitutions, insertions, and deletions with probability for each type of 15%. We consider such case closer to the actual conditions used for production systems when the well-curated datasets are mixed with unreliable data from different sources. Also, we train the system for longer (100 epochs). As shown in Table 6, we are able to restore more than 51% system quality when the system is trained for 60 epochs (compared to RNN-T baseline also trained for 60 epochs). When trained for an extra 40 epochs, the system can restore more than 71% quality (WERDR 71.4%). In the Appendix A, we

also publish the learning curves for the system demonstrating its effectiveness in reducing substitutions and deletions on the `dev-clean` data.

6 Conclusion

In our project, we trained speech recognition neural systems on the LibriSpeech [2] dataset. We explored the system's robustness to errors in target texts by artificially corrupting the ground truth target texts from the dataset. We also explored different RNN-T loss modifications to solve the problem of quality degradation in the discussed scenarios and proposed three losses:

- **Star-Transducer**, which mitigates the effect of missing words in transcripts and is able to restore more than 90% of the system quality in such case

- **Bypass-Transducer**, which allows insertions (extra words) in the transcripts and allows the restoration of more than 60% of the quality in such cases compared to "clean" transcripts

- **Target-Robust-Transducer**, which combines the approaches applied in the previous two losses. This loss can deal with arbitrary types of errors. It improves the system's quality when some words of transcripts are incorrect (substitutions), mitigating more than 40% of the quality loss for this case. For arbitrary types of errors, we also show that it can restore more than 70% of the quality compared to the baseline with the well-transcribed data.

Our work is based on the previous solutions for CTC loss [16, 17, 18] and Graph-RNN-T framework [1], and proposes a novel valuable solution for RNN-Transducer-based ASR systems. We demonstrated the effectiveness of the losses using the Fast Conformer [12] model.

The proposed Target-Robust-Transducer system can be applied in real-world scenarios when training models on a large amount of data from unreliable sources that usually contain transcription errors.

We also see direct applications for Star-Transducer beyond the discussed case with missing words in transcripts. Modern ASR systems are trained not only to provide transcription (in words) but also to provide punctuation, e.g., Whisper [42]. Since many curated ASR corpora do not contain punctuation (e.g., LibriSpeech [2] which we use in our work), such missing punctuation can be viewed as "deletions" in the transcripts, and with Star-Transducer loss the model can be trained directly on a mixture of datasets with and without punctuation.

In further work, we plan to investigate other models (e.g., Conformer [9] with 4x subsampling) and datasets. We also plan to apply the losses to train ASR systems on a large scale for production usage.

The losses are planned to be proposed to the open-source NeMo [31] framework.

7 Report Parameters and Additional Notes

The implementation is published in the GitHub repository: `https://github.com/artbataev/uol_final`.

The report contains 6 tables and 6 figures. The appendix contains an additional 1 table and 1 figure. We comply with word limits for each section.

8 Acknowledgments

References

[1] A. Laptev, V. Bataev, I. Gitman, and B. Ginsburg, "Powerful and extensible wfst framework for rnn-transducer losses," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[2] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "LibriSpeech: an ASR corpus based on public domain audio books," in *ICASSP*, 2015.

[3] A. Graves, "Sequence transduction with Recurrent Neural Networks," in *ICML: workshop on representation learning*, 2012.

[4] J. Li, "Recent advances in end-to-end automatic speech recognition," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

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

[6] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2016, pp. 4960–4964.

[7] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schluter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 325–351, 2023.

[8] M. Ghodsi, X. Liu, J. Apfel, R. Cabrera, and E. Weinstein, "Rnn-transducer with stateless prediction network," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7049–7053.

[9] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented Transformer for speech recognition," in *Interspeech*, 2020.

[10] Huggingface: Open ASR leaderboard. [Online]. Available: https://huggingface.co/spaces/hf-audio/open_asr_leaderboard

[11] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Neural Information Processing Systems*, 2017.

[12] D. Rekesh, N. R. Koluguri, S. Kriman, S. Majumdar, V. Noroozi, H. Huang, O. Hrinchuk, K. Puvvada, A. Kumar, J. Balam *et al.*, "Fast conformer with linearly scalable attention for efficient speech recognition," in *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2023, pp. 1–8.

[13] A. Laptev, S. Majumdar, and B. Ginsburg, "Ctc variations through new wfst topologies," in *Interspeech*, 2021.

[14] D. Povey, P. Żelasko, and S. Khudanpur, "Speech recognition with next-generation Kaldi (k2, Lhotse, Icefall)," *Interspeech: tutorials*, 2021.

[15] X. Cai, J. Yuan, Y. Bian, G. Xun, J. Huang, and K. Church, "W-CTC: a connectionist temporal classification loss with wild cards," in *ICLR*, 2022.

[16] V. Pratap, A. Hannun, G. Synnaeve, and R. Collobert, "Star Temporal Classification: Sequence classification with partially labeled data," in *NeurIPS*, 2022.

[17] D. Gao, M. Wiesner, H. Xu, L. P. Garcia, D. Povey, and S. Khudanpur, "Bypass Temporal Classification: Weakly Supervised Automatic Speech Recognition with Imperfect Transcripts," in *Proc. INTERSPEECH 2023*, 2023, pp. 924–928.

[18] D. Gao, H. Xu, D. Raj, L. P. G. Perera, D. Povey, and S. Khudanpur, "Learning from flawed data: Weakly supervised automatic speech recognition," *arXiv preprint arXiv:2309.15796*, 2023.

[19] J. S. Garofolo, "Timit acoustic phonetic continuous speech corpus," *Linguistic Data Consortium, 1993*, 1993.

[20] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[21] A. Y. Hannun, V. Pratap, J. Kahn, and W.-N. Hsu, "Differentiable weighted finite-state transducers," *ArXiv*, vol. abs/2010.01003, 2020.

[22] Librivox: Free public domain audiobooks. [Online]. Available: https://librivox.org

[23] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Álvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S. yiin Chang, K. Rao, and A. Gruenstein, "Streaming end-to-end speech recognition for mobile devices," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6381–6385, 2018.

[24] E. Variani, D. Rybach, C. Allauzen, and M. Riley, "Hybrid autoregressive transducer (hat)," *ICASSP*, 2020.

[25] Z. Meng, T. Chen, R. Prabhavalkar, Y. Zhang, G. Wang, K. Audhkhasi, J. Emond, T. Strohman, B. Ramabhadran, W. R. Huang, E. Variani, Y. Huang, and P. J. Moreno, "Modular hybrid autoregressive transducer," *SLT*, 2022.

[26] Z. Tian, J. Yi, Y. Bai, J. Tao, S. Zhang, and Z. Wen, "Fsr: Accelerating the inference process of transducer-based models by applying fast-skip regularization," *arXiv preprint arXiv:2104.02882*, 2021.

[27] Y. Wang, Z. Chen, C. yong Zheng, Y. Zhang, W. Han, and P. Haghani, "Accelerating rnn-t training and inference using ctc guidance," *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2022.

[28] Word Error Rate, "Word error rate — Wikipedia, the free encyclopedia," 2023, [Online; accessed 2024-01-05]. [Online]. Available: https://en.wikipedia.org/wiki/Word_error_rate

[29] Levenshtein Distance, "Levenshtein distance — Wikipedia, the free encyclopedia," 2023, [Online; accessed 2024-01-05]. [Online]. Available: https://en.wikipedia.org/wiki/Levenshtein_distance

[30] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2016.

[31] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, P. Castonguay, M. Popova, J. Huang, and J. Cohen, "NeMo: a toolkit for building AI applications using neural modules," *arXiv:1909.09577*, 2019.

[32] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," Mar. 2019. [Online]. Available: https://github.com/Lightning-AI/lightning

[33] Weights & biases: The developer-first mlops platform. [Online]. Available: https://wandb.ai/

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[35] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," *Interspeech*, 2019.

[36] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[37] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. [Online]. Available: https://aclanthology.org/D18-2012

[38] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019.

[39] ——, "SGDR: Stochastic gradient descent with warm restarts," in *ICLR*, 2017.

[40] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[41] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz— open source graph drawing tools," in *Graph Drawing*, P. Mutzel, M. Jünger, and S. Leipert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 483–484.

[42] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning.* PMLR, 2023, pp. 28 492–28 518.

Appendices

A Arbitrary Errors - Learning Curve

We provide an additional plot with the learning curve, demonstrating WER and its components for RNN-T and Target Robust Transducer training with arbitrary errors for 100 epochs, as discussed in Section 5.6. We can see in Figure 7 that the number of insertions produced in all cases is similar, but the number of deletions and substitutions produced by the system trained on corrupted data with the TRT loss is significantly lower than for RNN-T and is close to the number of errors produced by the RNN-T on the original non-corrupted data. The screenshot is produced by WandB [33].
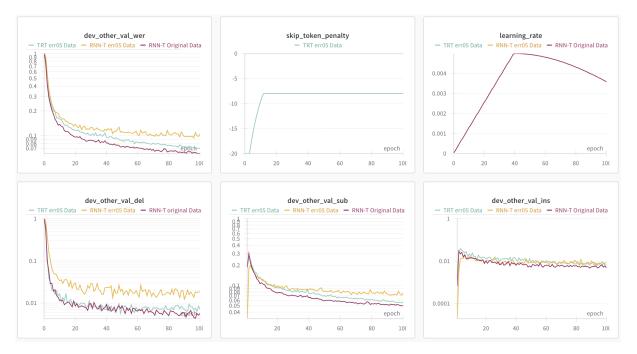

Figure 7: Arbitrary errors: learning curves for RNN-T (original and corrupted data) and Target Robust Transducer (corrupted data).

B Bypass-Transducer: Extended evaluation of hyperparameters

In this appendix section, we show the extended hyperparameter evaluation of the options for Bypass-Transducer loss regarding assigning weights for skip token transitions. In initial experiments, as discussed in Section 4.4.2, we tried different options and found that the system is trainable only with "maxexcl" and "sumexcl" options. In the system exploration process, we found the "sumexcl" option, which considers total "unassigned" log-probability (log-probability for all outputs excluding blank and target labels) to provide the best value. We provide an extended version of the Table 4. The results in 7 show that the "sumexcl" option outperforms the "maxexcl" by a significant margin.

Table 7: Bypass-Transducer Loss for insertions, Fast Conformer, 60 epochs, WER [%]. Extended evaluation.

Loss	Skip Weight,Mode	INS	dev clean	dev other	test clean	test other	WERD↓	WERDR↑
RNN-T	-	–	2.6	6.8	2.8	6.8		
RNN-T	–	20%	4.0	10.3	4.2	10.2	3.4	
Bypass-T	-6,maxexcl	20%	3.2	7.9	3.3	8.0	1.2	65.7%
Bypass-T	-6,sumexcl	20%	3.0	7.5	3.3	7.9	1.1	**67.6%**
RNN-T	–	50%	5.1	12.7	5.3	13.5	6.7	
Bypass-T	-6,maxexcl	50%	4.4	10.3	4.1	10.7	3.9	41.8%
Bypass-T	-6,sumexcl	50%	3.9	10.3	4.3	10.5	3.7	44.8%
Bypass-T	-5,maxexcl	50%	4.1	10.2	4.5	10.4	3.6	46.3%
Bypass-T	-5,sumexcl	50%	3.6	9.2	4.0	9.4	2.6	**61.2%**

C Losses Visualization

We additionally publish the Jupyter Notebook, which visualizes the lattices of the proposed losses. The notebook can be found in the repository
`https://github.com/artbataev/uol_final/blob/main/notebooks/Loss_Demo.ipynb`.