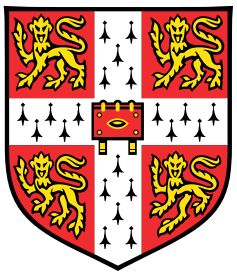
Knowledge Distillation for End-to-End Automatic Speech Recognition



Xiaoyu Yang

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Sidney Sussex College August 2021

I would like to dedicate this thesis to my loving parents.

Declaration

I, Xiaoyu Yang of Sidney Sussex College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

All experiments in this project are carried out using the PyTorch-based speech processing toolkit ESPnet (Li et al., 2021b), which provides basic building blocks for E2E ASR models in chapter 5. The original software package is modified to include functionalities such as knowledge distillation and streaming ASR models for the purpose of this project. Fairseq (Ott et al., 2019), a Pytorch-based sequence modeling toolkit is also used for Wav2vec 2.0 pre-trained models and related scripts in chapter 5.

Word count: 13769

Xiaoyu Yang
August 2021

Acknowledgements

I would like to acknowledge Prof. Phil Woodland and Qiujia Li for their interesting project proposal, which allows me to have an in-depth investigation of end-to-end automatic speech recognition and knowledge distillation. I am beyond grateful for their patient guidance and constant support throughout the project. Our weekly meeting was always helpful and enjoyable to me. I would also like to thank Guangzhi Sun for HPC usage tips, which saved me a lot of painful debugging time.

I would also like to express my greatest gratitude to my parents, who made my studies in Cambridge possible. I also owe sincere thanks to Wenora Liu, who accompanied me through countless working nights and cheered me up when I encountered obstacles.

Abstract

With the development of deep learning, larger and deeper neural networks for automatic speech recognition (ASR) which yield excellent results can be trained more efficiently. Although increased model size leads to larger performance improvement, it also poses more challenges for real-life applications due to memory and latency requirements. This project proposes to use knowledge distillation (KD) to utilise Wav2vec 2.0, a large self-supervised model for various speech downstream tasks, to train smaller models, which are suitable for real-life applications and shows very good performance at the same time.

As initial validation of the effectiveness of distilling knowledge from self-supervised ASR models, several distillation strategies for connectionist temporal classification (CTC) model are designed. Experiments on Librispeech clean-100h subset confirm a relative word-error-rate (WER) reduction of 17.7% and 16.4% on Librispeech clean and other test sets respectively compared to the model trained without KD. As a novel contribution, a distillation loss for transducer-based speech recognition models which is both memory efficient and effective is proposed. This method preserves full distribution and focuses on one-best alignment information, unlike existing methods that used full-lattice collapsed distribution. A relative WER reduction of 12.6% and 9.2% on Librispeech test-clean and test-other sets is observed in experiments with Librispeech clean-100h subset when compared to transducer models without KD, which is comparable against existing methods. Further experiments with labels of different qualities demonstrate the robustness of the proposed method. Experiments are also carried out on streaming transducer models. By introducing a time shift operation, preliminary KD results confirm a relative WER improvement of 7.4% and 6.8% on clean and other test sets respectively.

Experiments are further extended to a semi-supervised learning scenario. By incorporating clean-360h and other-360h as extra untranscribed speech for KD, even larger WER reduction is achieved on test-clean and test-other sets, suggesting a promising approach of combining knowledge distillation with semi-supervised learning to improve ASR models.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

X (or other bold capital letters) Matrix or sequence of vectors

x (or other bold letters) Vector or sequence of scalars

Pr Probability

x (or other letters) Scalar

Greek Symbols

κ Temperature

λ Weight of distillation loss

σ Sigmoid function

Acronyms / Abbreviations

ASR Automatic Speech Recognition

BPE Byte-Pair-Encoding

CTC Connectionist Temporal Classification

E2E ASR End-to-End Automatic Speech Recognition

GLU Gated Linear Unit

GRU Gated Recurrent Unit

HMM Hidden Markov Model

KD Knowledge Distillation

KL-divergence Kullback-Leibler Divergence

LSTM Long Short-Term Memory

MFCC Mel Frequency Cepstral Coefficients

MHSA Multi-Head Self-Attention

RNN-T Recurrent Neural Network Transducer

RNN Recurrent Neural Network

W2v2 Wav2vec 2.0

WER Word-Error-Rate

Chapter 1

Introduction

1.1 Introduction

In human society, speaking is one of the most informative and effective ways of communicating. Thus, it has always been of great interest to develop AI systems that can transform the incoming speech audio to a sequence of text, so that the communication between human and human or human and machine can be more natural and convenient. From an interpersonal perspective, a great example would be speech to text applications for people with hearing disability, allowing them to understand other people more easily. From a human-machine interaction perspective, most of the AI assistants such as Siri, Google assistant and Alexa, have an automatic speech recognition (ASR) system as a cornerstone to transform user instructions and quires to word sequences, so that they can be then processed by natural language understanding units.

With the great success of BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020), self-supervised learning has emerged as a paradigm to learn general data representations from unlabelled data. With only limited amount of labelled data for fine-tuning, the self-supervised pre-trained models are able to yield great results on various downstream tasks. This is especially meaningful for certain areas with limited labelled data available. This idea was recently adopted in speech recognition and the wav2vec 2.0 approach in (Baevski et al., 2020) has shown promising results in ASR. Although fine-tuning the pre-trained wav2vec 2.0 model can yield state-of-the-art performance in ASR tasks, the model itself is generally enormous with hundreds of millions of parameters, preventing its direct application in real-life scenarios such as on-device speech recognition due to both memory and latency requirements. Therefore, it is of great interest to somehow utilise the information learned by the enormous self-supervised ASR model, so that a much smaller model which is ideal for

real-life ASR applications can benefit from it.

Knowledge distillation is one common approach for model compression. It extracts the knowledge/information learned by a large teacher model with better performance, which will be then used to supervise a small student model with relative inferior performance, so that the student model can imitate the behaviour of the teacher model. With the knowledge provided from the teacher, knowledge distillation helps student model achieve better performance compared to being trained without external guidance. However, the performance of knowledge distillation depends greatly on the selection of knowledge and the way in which the student model is trained. A less informative knowledge or an improper way of supervising the student model's training might limit the potential performance gain and even deteriorate the training. In ASR, various approaches have been proposed to perform knowledge distillation under different ASR frameworks (Panchapagesan et al., 2021; Takashima et al., 2018). Each method has its own advantages and drawbacks, suggesting the difficulty of performing knowledge distillation in ASR tasks.

In this thesis, we propose to apply knowledge distillation to exploit large self-supervised models so that small models with relatively better performance can be obtained with only limited amount of labelled data. We also select representative knowledge extracted from the teacher model after analysing its output behaviour and design novel distillation methods for the training of student models.

1.2 Contribution

- An adaptation of the original wav2vec 2.0 model (Baevski et al., 2020) which prepares it for knowledge distillation and also improves its recognition accuracy.
- A distillation strategy for connectionist temporal classification (CTC) models (Graves et al., 2006) that further enhances the knowledge distillation performance.
- A novel distillation loss for transducer-based models(Graves, 2012) which significantly improves the student model's performance while being memory efficient. The modified version of this distillation method is then adopted in streaming transducer-based models and successfully improves the student model's performance without increasing latency.
- An investigation of extending knowledge distillation to semi-supervised learning. This approach leads to even greater performance boost when extra unlabelled data is used.

1.3 Thesis Outline

The structure of this thesis is as follows:

- Chapter 2 introduces background information in ASR. Common frameworks in End-to-End ASR are explained and compared.
- Chapter 3 reviews existing methods of knowledge distillation in end-to-end ASR. A novel distillation loss for transducer-based model is proposed.
- Chapter 4 presents the teacher model and student model architectures.
- Chapter 5 presents the knowledge distillation results in CTC models and transducer models. An investigation of our novel distillation method is carried out on both streaming and non-streaming transducer models. Experimental results with semi-supervised learning is also reported.
- Chapter 6 concludes the thesis and discusses possible future work.

Chapter 2

End-to-End Automatic Speech Recognition

Automatic speech recognition (ASR), or speech to text, is the process of mapping a sequence of audio features to a sequence of text. Traditionally, Hidden Markov Model (HMM) based models (Rabiner, 1989) are the mainstream for speech recognition. An ASR model normally consists of three parts: an acoustic model, a pronunciation model and a language model. These three parts are usually treated independently and optimised separately, making the training process complex. With the development of deep learning techniques, end-to-end automatic speech recognition (E2E ASR) has achieved comparable performance against HMM-based systems. By fusing multiple modules as a whole, E2E ASR enables the joint optimisation of multiple modules, which greatly simplifies the training process compared to HMM-based models. In this section, key components in E2E ASR models are first briefly introduced. Two common E2E ASR architectures, Connectionist Temporal Classification (CTC) (Graves et al., 2006) and Recurrent Neural Network Transducer (RNN-T) (Graves, 2012), will be explained. In the end, a brief introduction to self-supervised E2E ASR framework will be given.

2.1 Key Components in E2E ASR Model

2.1.1 Recurrent Neural Network

Speech recognition is a sequence modelling problem, where a sequence of text is generated given a series of input audio features. Conventional HMM-based models can be applied to model speech sequences, however, their performance are limited by the conditional independency assumption in HMM. On the other hand, recurrent layers (Elman, 1990;

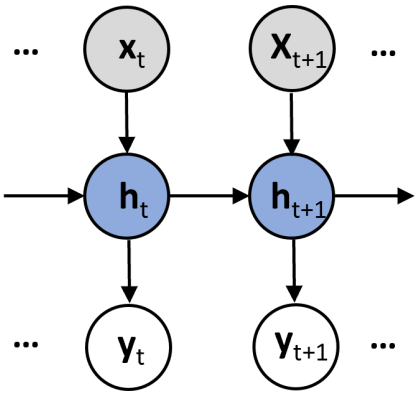


Fig. 2.1 A vanilla recurrent layer. $\mathbf{x}_t$, $\mathbf{y}_t$ and h_t are input, output and hidden state at time t respectively.

Jordan, 1997) are capable of modelling sequences without such conditional independency assumption. In (Elman, 1990), the recurrent layer not only produces an output vector, but also generates a hidden state that will be passed to the next timestamp when receiving an input vector (see Figure 2.1). The hidden state h_t incorporates the history information up to time t, which is the key to sequence modelling capability of a recurrent layer. To distinguish between other more advanced recurrent layers, the recurrent layer in Figure 2.1 will be referred to as vanilla RNN in later sections. To improve the performance of a vanilla RNN, a bi-directional architecture can be employed (Schuster and Paliwal, 1997). Another vanilla RNN layer that processes the input sequence in a reversed way is stacked on top of the first RNN layer. Despite of the performance gain obtained with the backward information, the bidirectional RNN requires to first digest the whole sequence before generating output, which can be a drawback in scenarios where real-time processing is needed.

To further improve the sequence modelling capability of recurrent layers, Long Short-Term Memory (LSTM) layers were proposed (Hochreiter and Schmidhuber, 1997). The key difference between a LSTM layer and a vanilla RNN layer is the usage of multiple gating functions. Forget gate ($\mathbf{i}_f$), input gate ($\mathbf{i}_i$) and output gate ($\mathbf{i}_o$) (see Figure 2.2) allow a LSTM cell to control the amount of information flow of hidden vector, input vector and output vector. Such an information pre-processing process leads to a performance gain in LSTM

layer. The operations performed in a LSTM cell are shown below

$$\mathbf{i}_f = \sigma(\mathbf{W}_f^f \mathbf{x}_t + \mathbf{W}_f^r \mathbf{h}_{t-1} + \mathbf{W}_f^m \mathbf{c}_{t-1} + \mathbf{b}_f) \tag{2.1}$$

$$\mathbf{i}_i = \sigma(\mathbf{W}_i^f \mathbf{x}_t + \mathbf{W}_i^r \mathbf{h}_{t-1} + \mathbf{W}_i^n \mathbf{c}_{t-1} + \mathbf{b}_i) \tag{2.2}$$

$$\mathbf{i}_o = \sigma(\mathbf{W}_o^f \mathbf{x}_t + \mathbf{W}_o^r \mathbf{h}_{t-1} + \mathbf{W}_o^m \mathbf{c}_t + \mathbf{b}_o) \tag{2.3}$$

$$\mathbf{c}_t = \mathbf{i}_f \odot \mathbf{c}_{t-1} + \mathbf{i}_f \odot f^m(\mathbf{W}_c^f \mathbf{x}_t + \mathbf{W}_c^r \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{2.4}$$

$$\mathbf{h}_t = \mathbf{i}_o \odot f^h(\mathbf{c}_t) \tag{2.5}$$

where all $\mathbf{W}$ are weight matrices that can be learned, $\mathbf{c}_t$, $\mathbf{h}_t$ and $\mathbf{h}_t$ are the memory cell and hidden state respectively. $\odot$ is the operation of elementwise multiplication and σ is the sigmoid activation function.

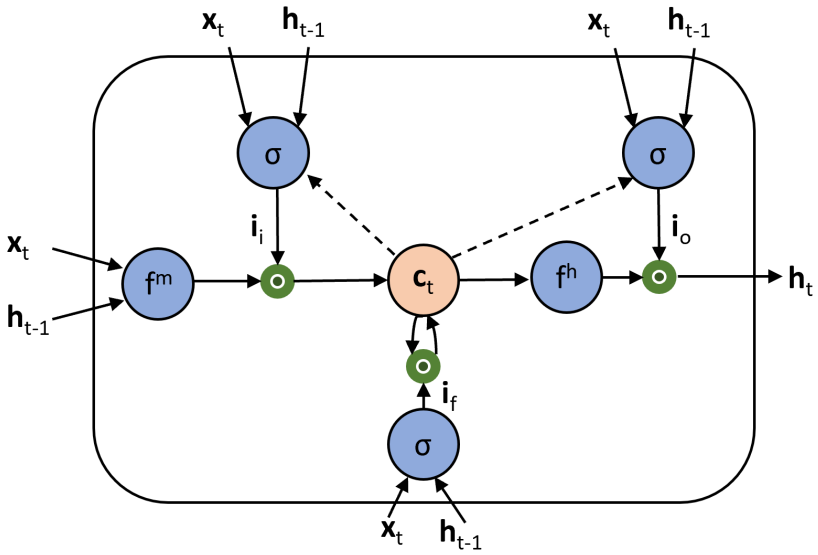


Fig. 2.2 An illustration of a LSTM cell. f^m and f^h are neural network functions.

Similarly, a bidirectional LSTM layer (Graves et al., 2005) can be also applied to incorporate backward information in sequence modelling. Beside LSTM layer, many recurrent layers such as Gated Recurrent Unit (GRU) (Chung et al., 2014) with gating functions are also introduced and show superior performance over vanilla RNN.

2.1.2 Transformer

Recurrent layers dominate E2E ASR for a long period of time (Chan et al., 2016; Huang et al., 2017). However, recurrent layers are inefficient because the processing in temporal order is unfriendly to parallel computation. To cope with this, a new architecture called transformer (Vaswani et al., 2017) was introduced and outperformed LSTM-based networks significantly in machine translation. In E2E ASR, transformer has also been adopted and yielded better performance compared to pure-RNN based models (Dong et al., 2018; Yeh et al., 2019). The key building block in a transformer model is transformer block, which is shown in Figure 2.3.

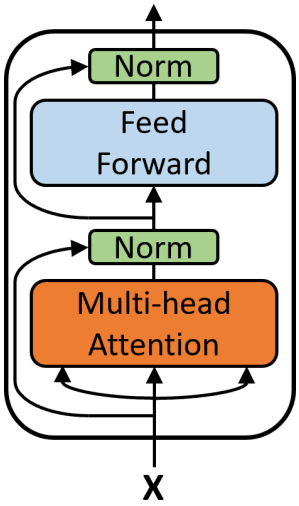


Fig. 2.3 An illustration of a transformer block.

As can be seen, a transformer block consists of a multi-head attention module and a feed-forward module. Both modules are combined with a residual connection (He et al., 2016) to allow training of deeper networks. Although two types of attention mechanisms are adopted in (Vaswani et al., 2017), only the self-attention will be introduced here as it is frequently used in E2E ASR. A multi-head self-attention (MHSA) module consists of multiple parallel attention layers, whose outputs are concatenated in the end. An illustration of MHSA's working mechanism is shown in Figure 2.4.

A dot-product self-attention layer is parameterised by three matrices $\mathbf{W}^Q, \mathbf{W}^K$ and $\mathbf{W}^V$. For an input sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)^T$, the weight matrices transform $\mathbf{X}$ to $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q \tag{2.6}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^K \tag{2.7}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^V \tag{2.8}$$

$\mathbf{Q}$ and $\mathbf{K}$ will be responsible for the generation of attention score

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T \tag{2.9}$$

where $\mathbf{S}$ is the attention score matrix of size $T \times T$. The attention matrix goes through a softmax activation that operates on each row of $\mathbf{S}$ so that each row vector $\mathbf{S}_i$ sums up to one. Each entry $\mathbf{S}_{ij}$ in the final attention matrix can be interpreted as how important $\mathbf{x}_j$ is to the output at time i. As can be seen, the attention mechanism allows modelling global dependencies in a sequence without regard to the distance in the input sequences: each input position may contribute to the output at all time. Note that an extra attention mask can be introduced to remove correlations within the sequence (see section 4.2 for more details).

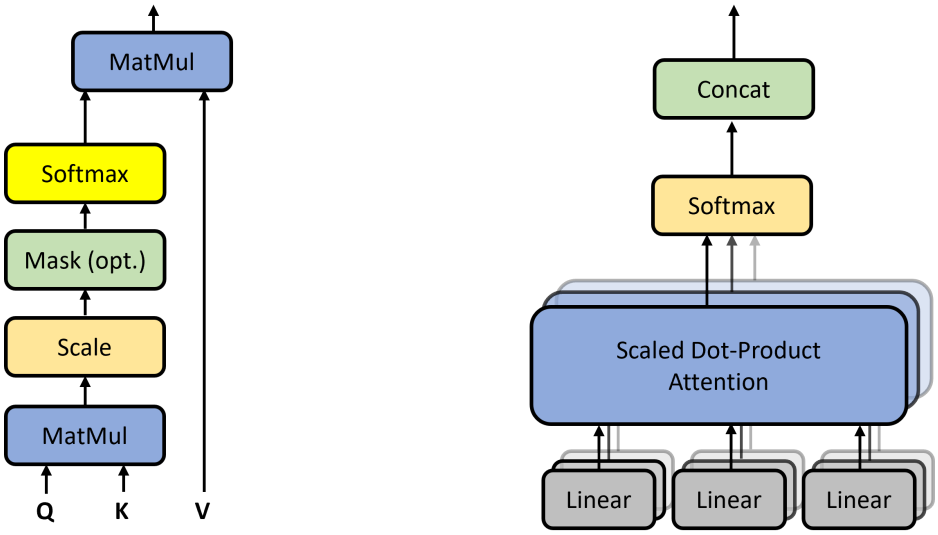


Fig. 2.4 An illustration of the Multi-head self-attention module. The left figure describes the dot-product attention mechanism adopted in the MHSA module. The right figure is a complete MHSA module. Each head is an independent dot-product attention layer as depicted on the left.

Another component that is crucial to the sequence modelling capability of transformer is the positional embedding. The module adopted in (Vaswani et al., 2017) adds absolute positional embedding to the input vector so that the positional information is included, otherwise the order of the sequence will be irrelevant to the attention score calculation. The form of the positional embedding vector is

$$PE(t,i) = \begin{cases} \sin(t/10000^{2i/d_{model}}) \\ \cos(t/10000^{2i/d_{model}}) \end{cases} \tag{2.10}$$

where t is the position in the sequence $\mathbf{x}$ and i is the dimension in the positional encoding. This means each dimension of the positional encoding is a sinusoid function. Except the absolute positional embedding in (Vaswani et al., 2017), other positional embeddings methods using relative positional information were also proposed and showed better results when dealing with longer sequences (Dai et al., 2019).

In transformer-based E2E ASR, multiple transformer blocks with self-attention are usually stacked to form an acoustic model. This acoustic model will be incorporated in E2E ASR frameworks such as CTC and RNN-T that will be introduced later in this chapter.

2.2 Connectionist Temporal Classification (CTC)

When solving speech recognition as a classification task where each input audio frame is classified as a word or sub-word unit, segmented speech labels with one-to-one mappings are required. Acquiring a large amount of such labels is almost impractical. Besides, due to the nature of speech, such labels are not unique as there are usually multiple sensible alignments between input audio features and output units. To address this issue, Graves et al. (2006) proposed Connectionist Temporal Classification (CTC) to focus on sequence labelling instead of individually labelled speech segments.

A CTC model comprises two parts: an encoder F and a softmax output layer F_o. The encoder is equivalent to an acoustic mode that encodes the input audio features to a feature space. Previously, the encoder is usually a RNN-based network to utilise context information while processing the input utterance. Recently, transformer (Dong et al., 2018; Synnaeve et al., 2020; Vaswani et al., 2017) and its adaptations (Gulati et al., 2020) have shown superior performance over traditional RNN-based models. The encoder output will be then fed to the softmax output layer to generate a categorical distribution. Note that the soft-

max output layer has one more unit than the output vocabulary size, which is the "blank" symbol, denoted as $\varnothing$. An activation of $\varnothing$ means that no unit in the output vocabulary will be generated. By introducing the "blank" symbol, a CTC model is able to define the total probability of any transcribed utterance by summing over all alignments associated with it.

Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ be an input sequence of length T, where each $\mathbf{x}_t$ would be a vector of Mel Frequency Cepstral Coefficients (MFCCs) or filter bank features. Let $\mathbf{y} = (y_1, y_2, ..., y_U)$ be an output sequence of length U, where each y_u represents an output unit in the output vocabulary $\mathscr{Y}$. Two typical output vocabularies would be graphemes or word pieces. The extended output vocabulary with "blank" symbol incorporated $\bar{\mathscr{Y}}$ is then defined as $\mathscr{Y} \cup \varnothing$. A blank symbol $\varnothing$ in an extended output sequence means output nothing. For example, the following sequences drawn from $\bar{\mathscr{Y}}^*$ of length 5

$$(c, \varnothing, a, \varnothing, t)$$

$$(c, \varnothing, \varnothing, a, t)$$

$$(\varnothing, c, \varnothing, a, t)$$

$$(c, a, t, \varnothing, \varnothing)$$

are all equivalent to the sequence $(c, a, t) \in \mathscr{Y}^*$ since they share the same sequence after removing all $\varnothing$. Therefore, the sequences $\mathbf{a} \in \bar{\mathscr{Y}}^*$ can be treated as alignments as the location of $\varnothing$ determines how input and output sequence are mapped with each other. The conditional probability of an alignment $\mathbf{a}$ is then defined as

$$Pr(\mathbf{a} \in \bar{\mathscr{Y}}^* | \mathbf{X}) = \prod_{t=1}^{T} Pr(a_t | \mathbf{x}_t) \tag{2.11}$$

where $Pr(a_t|\mathbf{x}_t)$ is a categorical distribution generated by the encoder. Then, the conditional probability of a labelled sequence $\mathbf{y} \in \bar{\mathscr{Y}}^*$ can be calculated by summing over all its possible alignments

$$Pr(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{a} \in \mathscr{B}^{-1}(\mathbf{y})} Pr(\mathbf{a}|\mathbf{x}) \tag{2.12}$$

where $\mathscr{B}$ is a mapping from $\bar{\mathscr{Y}}^*$ to $\mathscr{Y}^*$ that removes all $\varnothing$ and consecutive duplicated symbols.

2.2.1 Training of CTC

Before diving to the training process of a CTC network, it is important to understand how the output of a CTC model are transformed to an actual sequence. Basically, this process consists of the following two steps:

1. When duplicate units appear in a series, the series is collapsed to only one unit. For example, the alignment "c,c,a,a,a,t" will be decoded as "c,a,t" after removing the consecutive duplicated units. To output consecutive duplicated units after decoding, the network needs to output a blank symbol in between. For example, the alignment "f,o,o,∅,o,o,d" will be decoded as "f,o,o,d".

2. After the first step, all ∅ is removed. For example, the sequence "f,∅,o,o,∅,o,o,∅,d,∅" is first reduced to "f,∅,o,∅,o,∅,d,∅" after the step 1). Then, the sequence is further reduced to "f,o,o,d" after removing all blanks.

Training a CTC model is performed by maximising the probability of the target label sequence $\mathbf{y}$ defined in Equation 2.12. To calculate the conditional probability $Pr(\mathbf{a}|\mathbf{X})$ efficiently, a modified version of forward-backward algorithm can be applied. The idea is to decompose the conditional probability of generating a partial sequence into a weighted sum of the probabilities of generating its prefixes in a dynamic programming manner.

First, we define an extended label sequence $\mathbf{y}'$ by adding ∅ at both ends and between each pair of labels. Therefore, $\mathbf{y}'$ has a length of $2|\mathbf{y}|+1$. This helps to demonstrate all paths (with blanks) in the CTC output graph (see Figure 2.5). For a ground truth label sequence $\mathbf{y}$, we define the probability of generating $\mathbf{y}_{1:s}$ at timestamp t after decoding as $\alpha_t(s)$. We now show how $\alpha_t(s)$ can be calculated from $\alpha_{t-1}(s)$, $\alpha_t(s-1)$ and $\alpha_t(s-2)$.

As can be seen from Figure 2.5, when the output unit y'_s at time t is ∅, it could come from the previous timestamp $t-1$ by either emitting ∅ or the previous unit y'_{s-2} again. When the output unit l'_s at time t is a non-blank unit, an extra path is possible apart from the two mentioned before, which is from the previous timestamp $t-1$ by emitting this unit without duplication. Thus, the recursive relationship of the forward variable $\alpha_t(s)$ is

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1))Pr(y'_s|\mathbf{x}_t) & y'_s = \varnothing \quad or \quad y'_s = y'_{s-2} \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2))Pr(y'_s|\mathbf{x}_t) & else \end{cases} \tag{2.13}$$

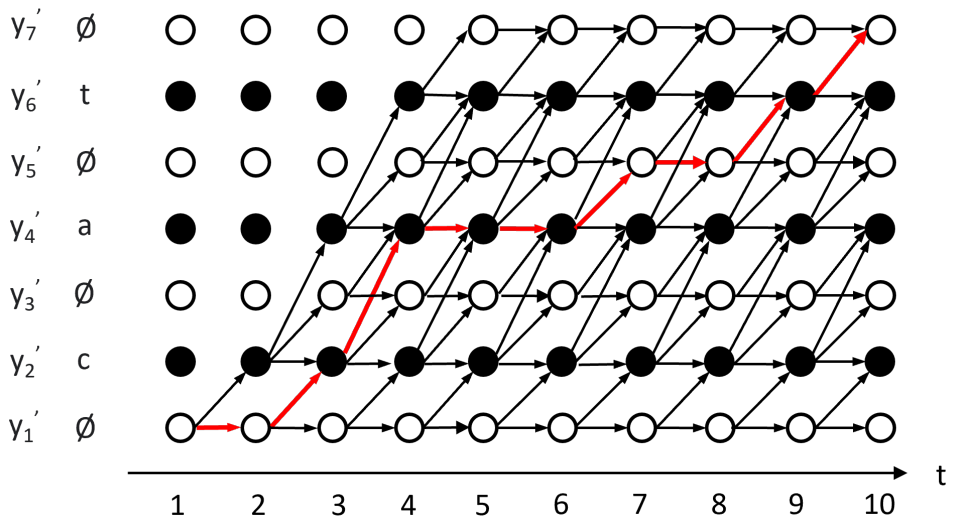


Fig. 2.5 The output graph of a CTC model. The horizontal axis is time axis. To distinguish between non-blank symbol and blank symbol, each non-blank symbol is denoted as a solid circle. Each possible transition is denoted as an arrow. An alignment "∅,c,a,a,a,∅,∅,t,∅" for the target label "cat" is highlighted in red.

where $Pr(y'_s|\mathbf{x}_t)$ is the probability of outputting y'_s at time t.Similarly, the backward variable $\beta_t(s)$ as the probability of generating $\mathbf{y}_{s:|\mathbf{y}|}$ at time t is defined as

$$\beta_t(s) = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1))Pr(y'_s|\mathbf{x}_t) & y'_s = \varnothing \quad or \quad y'_s = y'_{s+2} \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2))Pr(y'_s|\mathbf{x}_t) & else \end{cases} \tag{2.14}$$

With the forward and backward variables, the conditional probability of all alignments going through symbol s at time t can be calculated:

$$\alpha_t(s)\beta_t(s) = \sum_{\mathbf{a}\in\mathscr{B}^{-1}(\mathbf{y}),a_t=l_s} Pr(y'_s|\mathbf{x}_t)Pr(\mathbf{a}|\mathbf{x}) \tag{2.15}$$

Thus, the probability of all alignments $Pr(\mathbf{y}|\mathbf{x})$ given the input sequence $\mathbf{x}$ can be calculated by summing Equation 2.15 over all symbols in $\mathbf{y}$ at all time

$$Pr(\mathbf{y}|\mathbf{x}) = \sum_{t=1}^{T}\sum_{s=1}^{|\mathbf{y}|} \frac{\alpha_t(s)\beta_t(s)}{Pr(y'_s|\mathbf{x}_t)} \tag{2.16}$$

During training, the value in Equation 2.16 is maximised with optimisers such as SGD or Adam(Kingma and Ba, 2015).

2.2.2 Inference with CTC

Various search algorithms can be applied when evaluating a CTC model on test data. Greedy search simply chooses the output unit with the highest probability at each timestamp, leading to the alignment with the highest probability. Greedy search is fast and easy to implement, but its accuracy tends to be low as the sequence decoded from the most probable alignment is not necessarily the most probable one. As an extension of greedy search, beam search is frequently used in CTC inference, which improves the recognition accuracy while keeping the search time manageable.

2.2.3 Problems with CTC

Although being simple in network architecture, CTC-based models suffer from the following limitations:

1. As the network is only able to emit one symbol (either an actual output unit or a blank symbol) at each timestamp, it is incapable of generating output sequences that are longer than the input sequence. Fortunately, this limitation will have little impact in ASR as the output text sequence tend to be much shorter than the input audio sequence.

2. As can be seen from Equation 2.11, the calculation of the probability of an alignment is based on the assumption that each element in the alignment is independent: the output unit from other timestamps do not affect the categorical distribution at other timestamps. Although the encoder (or acoustic model) in CTC considers context information, the softmax layer (or language model) does not. This output characteristic limits CTC model's performance greatly. Therefore, a CTC model is usually used jointly with an external language model to improve recognition accuracy. However, the usage of an external language model violates the nature of an E2E ASR model.

2.3 Recurrent Neural Network Transducer (RNN-T)

To cope with the two deficiencies in CTC models, RNN-T models was proposed. An RNN-T model consists of three modules: A prediction network $\mathscr{G}$, a transcription network $\mathscr{F}$ and a joint network $\mathscr{J}$ (see Figure 2.6). Similar to CTC models, a blank symbol is also introduced to allow representations of alignments. However, the generation of output sequences in RNN-T is much more complicated due to the introduction of new modules and will be discussed below.

2.3.1 RNN-T Architecture

The transcription network $\mathscr{F}$ shares the same function as the encoder in CTC models. Given an input sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ of length T, the encoder transforms it to $\mathbf{F} = (\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_T) = \mathscr{F}(\mathbf{X})$. Note that the length of $\mathbf{F}$ is assumed to be the same as $\mathbf{X}$ here for simplicity. However, an arbitrary subsampling factor can be used. Like in CTC models, popular choices of the encoder include recurrent layers and transformer-based blocks. The name "RNN-T" originated from the initial proposal in (Graves, 2012), where the transcription network (or encoder) is based on bi-LSTM layers. This would cause confusion as most of the current RNN-T networks are not RNN-based any more. Therefore, we would refer it as neural transducer or transducer to avoid confusion in this project when necessary.

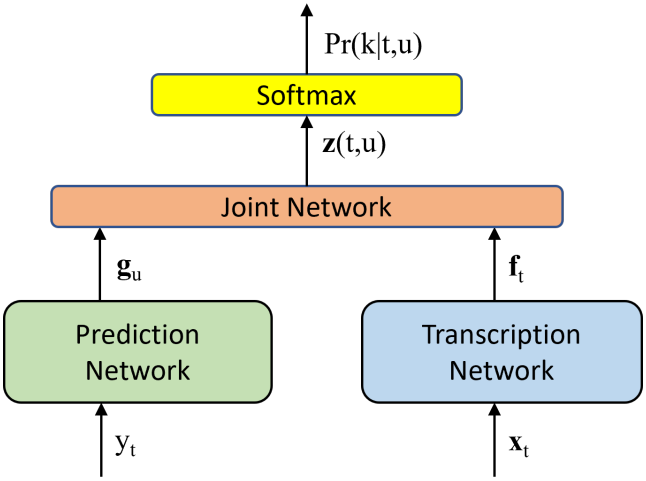


Fig. 2.6 An illustration of an RNN-T model.

The key difference between an RNN-T model and a CTC model lies in the introduction of the prediction network $\mathscr{G}$ in RNN-T. Let $\mathscr{Y}$ be the output vocabulary, $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_U)$ with $\mathbf{y}_u \in \mathscr{Y}$ be the target label sequence of length U and $\hat{\mathbf{y}} = (\varnothing, \mathbf{y}_1, ..., \mathbf{y}_U)$ the extended sequence with $\varnothing$ prepended. The prediction network $\mathscr{G}$ consumes the label token in $\hat{\mathbf{y}}$ in temporal order and output $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_U) = \mathscr{G}(\hat{\mathbf{y}})$ as the prediction sequence. A typical construction of the prediction network consists of an embedding layer and an output module. The embedding layer receives the one-hot encoded vectors generated from $\hat{\mathbf{y}} = (\varnothing, y_1, ..., y_U)$. Note that if a token is $\varnothing$, its embedding will be a zero vector of length $|\mathscr{Y}|$. Therefore, the input size of the embedding layer is $|\mathscr{Y}|$. This embedding will then be fed to the output module that is able to model the next token given all previous tokens. Recurrent layers such as LSTM(Hochreiter and Schmidhuber, 1997) layers and GRU(Chung et al., 2014) layers have shown great success in temporal prediction and are therefore popular choices for the

prediction network. A prediction network plays the role of modelling each token in **y** given all the previous tokens, which is similar to a language model.

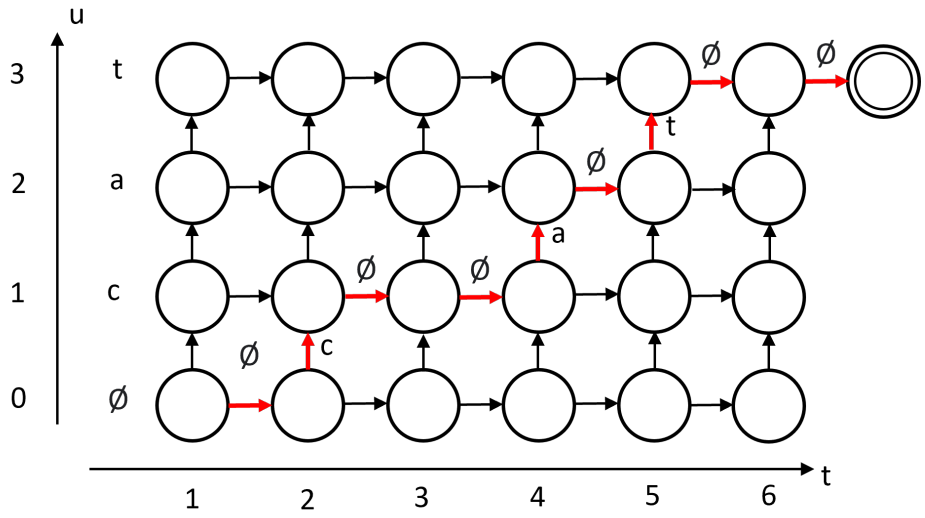


Fig. 2.7 An illustration of an RNN-T output lattice **Z** for the target label sequence "cat". Each node is denoted as a circle. One possible path is highlighted using red arrows. As can be seen, only horizontal and vertical transitions are allowed in an RNN-T lattice.

Given the output $\mathbf{F} = (\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_T)$ of the transcription network and the prediction vector $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_U)$, the joint network $\mathscr{J}$ produces a lattice as shown in Figure 2.7 . Each node $\mathbf{z}(t,u)$ in the lattice **Z** is a vector of length $|\bar{\mathscr{Y}}|$ and is calculated by

$$z(k,t,u) = exp(l_f(\mathbf{f}_t)^k + l_g(\mathbf{g}_u)^k) \tag{2.17}$$

where k denotes the k-th entry of the vector. l_f and l_g are two linear layers that transform $\mathbf{f}_t$ and $\mathbf{g}_u$ to the same embedding space. Each vector in h will be normalised to yield a valid transition probability in the lattice

$$Pr(k \in \bar{\mathscr{Y}}|t,u) = \frac{z(k,t,u)}{\sum_{j \in \bar{\mathscr{Y}}} z(j,t,u)} \tag{2.18}$$

2.3.2 Training of RNN-T

As can be seen from Figure 2.7, the probability defined in Equation 2.18 corresponds to the transition probability (including blank) in the lattice: an emission of a non-blank symbol indicates a vertical transition whereas a blank symbol indicates a horizontal transition. This

means that multiple units can be generated at one single timestamp, allowing the output sequence (after decoding) to be longer than the input sequence. A path from the bottom left corner to top right corner is a valid alignment between $\mathbf{X}$ and $\mathbf{y}$ and its probability can be easily calculated as the product of all emitted symbols' probabilities. The training of RNN-T model is performed by maximising the probability of the ground truth label $\mathbf{y}$ given input sequence $\mathbf{X}$. Obviously, $Pr(\mathbf{y}|\mathbf{X})$ can be written as the sum of probabilities of all alignments associated with $\mathbf{y}$. Similar to CTC models, a modified version of forward-backward algorithm can be applied to calculate the $Pr(\mathbf{y}|\mathbf{X})$ efficiently.

Let $\alpha_t(u)$ be the forward variable, which is the probability of emitting the sequence $\mathbf{y}_{1:u}$ at the time stamp t after consuming $\mathbf{F}_{1:t}$. We show that $\alpha(t,u)$ can be recursively expressed with $\alpha(t-1,u)$ and $\alpha(t,u-1)$. As can be seen from Figure 2.7, the network must first emits a blank symbol before consuming the next audio feature $\mathbf{f}_t$. Thus, the node (t,u) can only be reached from two nodes: from $(t-1,u)$ by emitting $\varnothing$ or from $(t,u-1)$ by emitting y_u. We have then

$$\alpha(t,u) = \alpha(t-1,u)Pr(\varnothing|t-1,u) + \alpha(t,u-1)Pr(y_u|t,u-1) \tag{2.19}$$

for all $1 \leq t \leq T$ and $0 \leq u \leq U$ and an initialisation of $\alpha(1,0) = 1$. Similarly, the backward variable $\beta(t,u)$ is defined as the probability of emitting $\mathbf{y}_{u+1:U}$ after consuming $\mathbf{F}_{t:T}$. Like the forward variable, $\beta(t,u)$ can also be expressed recursively with

$$\beta(t,u) = \beta(t+1,u)Pr(\varnothing|t,u) + \beta(t,u+1)Pr(y_{u+1}|t,u) \tag{2.20}$$

for all $1 \leq t \leq T$ and $0 \leq u \leq U$ and an initialisation of $\beta(T,U) = Pr(\varnothing|t,u)$. The product of a pair of forward and backward variable $\alpha(t,u)$ and $\beta(t,u)$ is the probabilities of all alignments that pass through the node (t,u) in the lattice. To obtain $Pr(\mathbf{y}|\mathbf{X})$, we can find a set of nodes $\{t_i,u_i\}$, so that all alignments between $\mathbf{X}$ and $\mathbf{y}$ must go through exactly one node in the set. With such a set of nodes $\mathscr{S}$, we now derive the total probability of the target sequence $\mathbf{y}$ given $\mathbf{X}$

$$Pr(\mathbf{y}|\mathbf{X}) = \sum_{(t,u)\in\mathscr{S}} \alpha(t,u)\beta(t,u) \tag{2.21}$$

There are multiple $\mathscr{S}$ fulfilling the requirements. A common choice would be the set consisting of nodes lying on the top-left to bottom-right diagonal. The network is then trained with gradient descent methods by minimising the negative log-likelihood $-\ln Pr(\mathbf{y}|\mathbf{X})$.

2.3.3 Inference with RNN-T

Evaluating an RNN-T model involves a more complicated process than training. Due to the introduction of the prediction network, the output distribution for the current unit depends on all previous emitted units during decoding. The previously generated units will be fed back to the prediction network, with which new predictions are made. Finding the most probable output sequence is computationally intractable as the search space expands drastically. Therefore, Graves (2012) proposed to use a beam search algorithm to find a set of hypotheses with high scores. Beam search is not guaranteed to find the most probable sequence, however, it allows a trade-off between computational cost and search accuracy. The RNN-T beam search algorithm consumes each input audio feature $\mathbf{f}_t$ at each step and extends the current hypotheses alongside the u-axis before the next audio feature $\mathbf{f}_{t+1}$ is consumed. Although the generation of $\mathbf{f}_t$ from 1 to T is not necessarily sequential (depending on the architecture of the encoder), the decoding process is sequential regarding to the encoder output.

2.3.4 Streaming RNN-T

Since an RNN-T model emits output units in a sequential manner during decoding, it becomes a popular choice for streaming E2E ASR model due to its high-accuracy. A streaming ASR model attempts to start recognition without waiting for the sentence to end, which is the so-called real-time speech recognition. Ideally, the model should predict the output units causally: no future context is needed when making the current prediction. However, this will significantly deteriorate the recognition accuracy as less contextual information is available. Therefore, a certain amount of lookahead into the future is usually allowed to compensate for recognition accuracy. A larger lookahead will improve the recognition accuracy, but also causes higher latency. To build streaming transducer models, Zhang et al. (2020) limited the past and future self-attention context instead of attending to the whole sequence in the transformer encoder. Chen et al. (2021) proposed to divide the input sequence into fixed-sized chunks. By doing so, the history context grows as the network goes deeper whereas the maximum lookahead remains the same.

A lot of research has been done in improving the performance of an RNN-T streaming ASR model mainly from two perspectives. 1) Encouraging the model to emit output units more quickly. As a streaming RNN-T has limited future context information, the network inclines to emit symbols (non-blank) later to wait until future frames are processed, leading to a large emission delay. Therefore, extra regularisations are made to discourage this behaviour

(Sainath et al., 2020; Yu et al., 2021) and reduce latency. 2) Improving the recognition accuracy without increasing the latency (Doutre et al., 2021; Kurata and Saon, 2020).

Although the term "streaming" does not have a unified definition in ASR, a proper streaming model should have as few lookahead frames as possible. For example, a lookahead of 8 frames, which corresponds to $8 \times 40ms = 320ms$ delay of encoder time at 25Hz input frame-rate, is quite noticeable as it is the average duration of a spoken English word. In this project, the transducer models which only has a limited amount of lookahead (or no lookahead) will be referred to as streaming models. If no limitation is posed on lookahead, the model will be called off-line transducer or non-streaing transducer.

2.4 Self-supervised Learning for E2E ASR

Self-supervised learning from unlabelled data has recently emerged as a paradigm to learn general data representations. In the field of natural language processing (Brown et al., 2020; Devlin et al., 2019) and in computer vision (Bachman et al., 2019; Henaff, 2020), the effectiveness of self-supervised learning has been proven. Self-supervised learning is attractive since the training process requires only unlabelled data. The model is then able to achieve great performance after fine-tuning with little labelled data. Building a good ASR model usually requires large amount of labelled data, which is challenging to collect. Therefore, a lot of research has been done to apply self-supervised learning to learn general speech representations. In (Jiang et al., 2019), self-supervised training is performed by reconstructing the filter bank input features. In (Baevski et al., 2019), a fixed quantised representation is first learned and masked training with a self-attention model is performed afterwards. Recently, Baevski et al. (2020) proposed a self-supervised training process which performs the learning of quantisation and masked training simultaneously. After extensive pre-training on untranscribed speech data, the model is able to achieve state-of-the-art recognition accuracy after fine-tuning with only limited amount of transcribed speech.

Pre-training in wav2vec 2.0 model (W2v2) shares a similar idea as in BERT (Devlin et al., 2019): predicting a proportion of the masked representations. Unlike most ASR models, a W2v2 model receives raw audio waveform as input instead of filter bank features or MFCCs. A convolutional feature encoder receives the raw waveform $\mathbf{x}$ and converts it to the latent speech representations $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_T)$ for T time-steps. This module can be treated as a frontend that subsamples the waveform from 16kHz to around 50Hz. $\mathbf{Z}$ is then fed to a transformer to build contextualised representation $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_T)$. The network also has

a quantisation module that receives $\mathbf{z}_t$ and generates a quantised vector $\mathbf{q}_t$. Quantisation is done in a fully differentiable way by using product quantisation (Jegou et al., 2010) and gumbel softmax (Jang et al., 2017) so that the quantisation module can be optimised in a gradient descent manner jointly with the rest of the network.

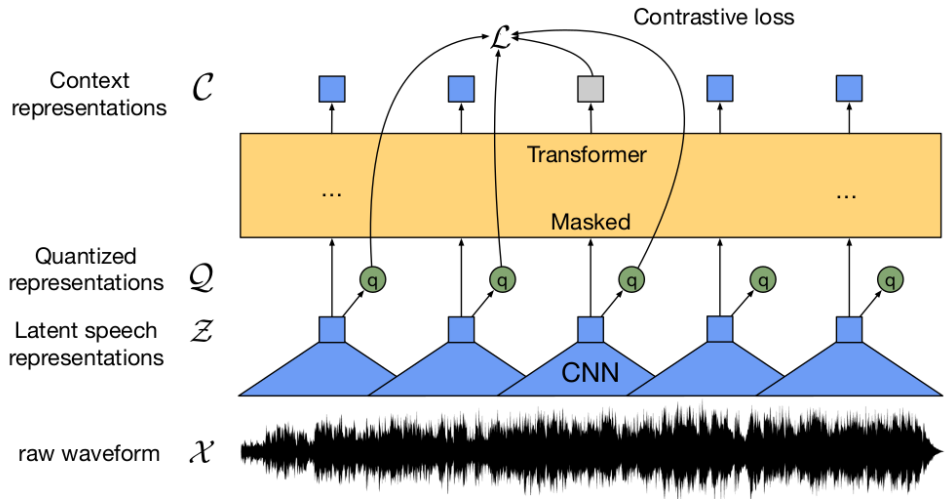


Fig. 2.8 Architecture of a W2v2 model [1].

Inspired by BERT(Devlin et al., 2019), multiple segments of the latent representation sequence $\mathbf{Z}$ are masked during the pre-training process. The starting positions of the segments are randomly selected while the length of each segment is fixed. All masked positions share the same feature vector after masking. The objective is to identify the the correct quantised latent audio feature for each masked timestamp. For a masked latent representation $\mathbf{z}_t$ at time t, its quantised vector $\mathbf{q}_t$ is first obtained using the quantisation module. Then, the partially masked sequence $\mathbf{z}$ is fed as input to generate a contextualised representation $\mathbf{c}$. The transformer is trained to identify $\mathbf{q}_t$ among the set Q_t of $K+1$ quantised candidate representations containing both distractors and $\mathbf{q}_t$ for all t that are masked. Such kind of a loss is called a contrastive loss and is expressed as

$$L = -\log \frac{exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\mathbf{q} \in Q_t} exp(sim(\mathbf{c}_t, \mathbf{q})/\kappa)} \tag{2.22}$$

where κ is a temperature parameter that can be tuned and *sim* is the cosine similarity measure. Apart from the contrastive loss, a diversity loss is also minimised at the same time to encourage the quantisation module to generate more diverse quantised vectors. The final loss

[1] This figure is taken from (Baevski et al., 2020).

is a weighted sum of both losses.

Strictly speaking, a W2v2 model is only a pre-trained acoustic model that generates good audio representations: it still needs to be incorporated in an ASR architecture to perform ASR tasks. After the model is pre-trained on unlabelled data, Baevski et al. (2020) integrated the W2v2 model as encoder in a CTC framework and fine-tuned the network using labelled data. The W2v2+CTC model showed state-of-the-art performance with only limited amount of fine-tuning data and the word-error-rate (WER) further decreased when more labelled data was used for fine-tuning. However, it can be integrated in ASR architectures other than CTC, which is shown in section 4.1.

Chapter 3

Knowledge Distillation in ASR

With the advances in hardware (GPUs and TPUs) and neural network training techniques, training larger and deeper neural network has become possible. For example GPT-3 (Brown et al., 2020), one of the largest models, has around 175 billion parameters. Although a larger size of a model pushes the performance boundary further, it also poses more challenges for deployment in real-life scenario such as on-device usage due to limited storage and computational resources. This restriction also applies to ASR models. Various methods such as low-rank factorisationYu et al. (2017), parameter pruning and sharing (Wang et al., 2018) and knowledge distillation (Hinton et al., 2015) have been proposed to compress a large model while attempting to affect the model's performance as little as possible. Here, we will mainly focus on applying knowledge distillation on ASR models.

3.1 Knowledge Distillation (KD)

Knowledge distillation (KD) is the process of transferring knowledge from a teacher model to a student model, so that the student model's performance can be improved. The teacher model could be cumbersome ensemble or a single very deep neural network with a large number of parameters while the student model is usually much smaller in size. When training the small model, instead of optimising its performance solely on the training data, we train it to generalise in the same way as the large model do. By mimicking the output behaviour of the large model, the student model might achieve even better performance on test data compared to the normal way of training.

To ensure the small model to imitate teacher model's generalisation's behaviour successfully, it is crucial to select the correct supervision signal. However, the definition of

"knowledge" varies from area to area. Hinton et al. (2015) proposed to use the conditional output distribution generated from the teacher model as the training label for the student model in image classification. The training objective is the Kullback-leibler divergence (KL-divergence) between teacher and student output distribution. This distribution is usually called "soft-labels" as they reveal the underlying relationships across different classes compared to the one-hot encoded hard-labels. This idea is then extended to more complicated multi-class classification problem in object detection Chen et al. (2017). Knowledge distillation is also applicable in the area of speech recognition. An obvious KD solution is to treat speech recognition as a classification problem, for example in a CTC model, where each frame can be assigned with a label. However, such a perspective assumes independency between all the frames, which does not fully agree with the nature of speech as a sequence. To cope with this, various methods have been proposed to utilise sequence-level information extracted from the teacher model (Takashima et al., 2018,?). To apply knowledge distillation in more complex frameworks such as RNN-T, Listen-Attend-Spell(Chan et al., 2016) and their streaming versions, other approaches are investigated (Kurata and Saon, 2020; Panchapagesan et al., 2021). In the rest of this chapter, some knowledge distillation methods on CTC and RNN-T ASR frameworks are investigated. A novel distillation loss on RNN-T framework designed by us is also proposed and its usage in both streaming and non-streaming scenario is discussed.

3.2 Knowledge Distillation in CTC

3.2.1 Frame-level KD in CTC

As introduced in section 2.2, a CTC model produces a sequence of conditional probability distribution of length T. This is similar to a classification problem, where each frame x_t of the input sequence $\mathbf{X}$ is classified. To apply knowledge distillation on a CTC model, a straightforward method is to enforce the student model to imitate teacher model's output distribution given the same input $\mathbf{X}$. In a classification knowledge distillation scenario (Hinton et al., 2015), the KL-divergence between teacher and student distribution is minimised:

$$L_{kd} = \sum_{k=1}^{K} l_k^t \log \frac{l_k^t}{l_k^s} \tag{3.1}$$

$$= -\sum_{k=1}^{K} l_k^t \log l_k^s + \sum_{k=1}^{K} l_k^t \log l_k^t \tag{3.2}$$

where l_k^t and l_k^s is the probability of class k predicted by the teach and student model respectively. As can be seen, the minimisation of KL-divergence is equivalent to minimisation of the cross-entropy as the second term in the second equation is a constant value (entropy of the teacher distribution) if the teacher model is fixed. Therefore, only the cross-entropy term will be used in the implementation.

When applied on a CTC model, the distillation loss in Equation 3.2 is extended with one more summation over the length of the input sequence. For the input sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$, the distillation loss between the teacher model and the student model is for input sequence $\mathbf{X}$

$$L_{kd} = -\sum_{t=1}^{T}\sum_{k=1}^{|\bar{\mathscr{Y}}|} Pr_t(y_t = k|\mathbf{x}_t)\log Pr_s(y_t = k|\mathbf{x}_t) \tag{3.3}$$

This is usually referred to as frame-level distillation loss, as the loss function is obtained by summing over all the frames. This distillation loss can be further extended by introducing a hyper-parameter κ named temperature. The teacher logits before the softmax operation are scaled by the temperature

$$Pr_t(y_t = k|\mathbf{x}_t) = \frac{exp(h_k^t/\kappa)}{\sum_{j=1}^{K} exp(h_j^t/\kappa)} \tag{3.4}$$

where h_k^t is the k-th entry of the teacher logit vector. The modified teacher distribution will then be used in Equation 3.3, where the student distribution is equally scaled. As can be seen from Equation 3.4, a larger temperature smooths out the distribution, allowing probabilities from all classes to contribute to L_{kd}. A smaller temperature, however, cuts out the classes with relatively lower probabilities, leaving out only the peaks in the distribution.

Although the soft-label contains all output information from the teacher model in CTC, it is still dangerous to rely on the teacher soft-labels entirely as they contain error and might guide the student model to an unwanted local minimum. Therefore, a weighted sum of the original CTC loss and distillation loss is adopted to form the final loss function

$$L = (1-\lambda)L_{ctc} + \lambda L_{kd} \tag{3.5}$$

where λ is between 0 and 1. Obviously, $\lambda = 0$ means training without using soft-labels and $\lambda = 1$ means training purely with soft-labels.

3.2.2 Sequence-level KD in CTC

Although the output distribution of each frame contains all information that a teacher CTC model has learnt, the way frame-level CTC KD (Equation 3.3) makes use of all the distributions assumes independency between all timestamps, which does not conform to the nature of speech as a sequence.

To deal with this, Huang et al. (2018) proposed to use the posterior $p(y_t = k|\mathbf{X}, t)$, the probability of the k-th output unit at frame t generated by the teacher model as distillation label. This probability can be obtained using the same forward-backward algorithm used for CTC training (see subsection 2.2.1). Takashima et al. (2018) proposed to conduct sequence-level KD in CTC by generating a set of hypotheses candidates using the teacher model. The top-50 hypotheses found using beam search, for example, can be used to perform multi-target training. A CTC loss can be calculated for each of the hypotheses and the final loss is a sum of all the individual loss weighted by each hypothesis's score.

Despite of reported improvement against frame-level KD, sequence-level KD in CTC is still inherently contradictory. A CTC model assumes independency between individual frames, indicating that the frame-level distribution should carry all information from the teacher model. The methods in (Huang et al., 2018; Takashima et al., 2018) are therefore only post-processing of the frame-level distillation labels. Thus, the inherent incompetence of modelling frame inter-dependency in CTC models limits the potential performance gain brought by sequence-level KD.

3.3 KD in RNN-T

Compared to KD in CTC models, KD in RNN-T models is more challenging due to the more complex output format in RNN-T. Given a training input sequence $\mathbf{X}$ of length T and a target label sequence $\mathbf{y}$ of length U, an RNN-T model generates a lattice of size $T \times U \times K$, where K is the size of the extended output vocabulary (including $\varnothing$). A naive implementation of KD would involve the calculation of KL-divergence over the entire lattice. Although theoretical possible, this method requires huge amount of GPU memory during training. Assume a training pair $(\mathbf{X}, \mathbf{y})$ with $T = 500$, $U = 150$ and an output vocabulary of size 1000, the distillation label for 4 such training pair will take up more than 1GB GPU memory. Such an implementation not only prevents training in large batch-size but also requires significantly longer computation time, which is undesirable. Therefore,a more compact form of distillation

label should be used in RNN-T KD.

3.3.1 Full-lattice KD with Collapsed Distribution

To reduce memory cost and computation effort, Panchapagesan et al. (2021) proposed to use a collapsed distribution on the full lattice as distillation label. Let $\mathbf{z}$ be the 3-dimensional lattice generated by the teacher model given input sequence $\mathbf{x}$ and target label sequence $\mathbf{y}$, each node $\mathbf{z}(t,u)$ will be reduced from a K-dimensional vector to a 3-dimensional vector, where K is the size of the output vocabulary. Only the probability of emitting $\varnothing$ (blank) and the input unit y_u to prediction network are kept. The remaining probabilities will be merged together as one single value, which is called "rest". By doing so, the third dimension of the output lattice will be 3 regardless of the size of the output vocabulary, reducing the space complexity to $\mathcal{O}(TU)$. The distillation loss is then calculated as

$$L_{kd} = -\sum_{t=1}^{T}\sum_{u=1}^{U}\sum_{k=1}^{3} z'_t(k,t,u)\log z'_s(k,t,u) \tag{3.6}$$

where z'_t and z'_s are the teacher and student lattice with collapsed distribution respectively. This loss can be easily extended to include a temperature hyper-parameter. Finally, the total loss L for one input utterance is a weighted sum

$$L = L_{trans} + \lambda L_{kd} \tag{3.7}$$

where λ is a tunable hyperparameter controlling the contribution of L_{kd}.

Since a blank indicates a transition from current timestamp to next timestamp, this distillation loss focuses on transition and emission of the correct symbol. However, the collapsed distribution contains much less information compared to the original full distribution. Based on the way in which the collapsed distribution is calculated, no information on other symbols are provided except for the blank symbol and symbol to be emitted, which might affect the training of the student model. Also, adopting the full lattice as distillation label might not be fully necessary. Jain et al. (2019) pointed out that a large proportion of the lattice do not contribute to the decoding results and the performance degradation after pruning those regions is negligible because paths going through those regions have very low probabilities.

3.3.2 One-best Path KD in RNN-T

To tackle the problems with the aforementioned full-lattice collapsed distribution KD, we propose a distillation method that preserves the full distribution while still being memory efficient. The idea is simple and easy to implement.

To preserve the full distribution, only partial nodes in the output lattice $\mathbf{Z}$ can be stored due to the memory limitation. Ideally, we would like to store the nodes that are crucial to the decoding results, so that the most amount of information in the lattice is kept. Such nodes can be found by analysing the lattice, where a set of paths with high probabilities can be found. We propose to consider nodes that are alongside the best paths as crucial nodes, as they contribute more to the decoding results. Formally, for an input sequence $\mathbf{X}$, the target sequence $\mathbf{y}$ and lattice $\mathbf{Z}$, we find the one-best path as $\mathbf{h}$ using greedy search in output lattice $\mathbf{Z}$. $\mathbf{h}$ is a sequence of nodes $z(t,u)$, which is also one possible alignment between $\mathbf{X}$ and $\mathbf{y}$. The token output distribution at each node in $\mathbf{h}$ will be stored, which will be used as soft-labels for KD. The pseudo code for one-best path generation is shown below.

Algorithm 1 Generation of one-best path from a complete lattice $\mathbf{Z}$

Inputs: $\mathbf{Z}$, T, U;
Initialize: $t \leftarrow 0$, $u \leftarrow 0$, $list_t \leftarrow [\,]$, $list_u \leftarrow [\,]$, $\mathbf{h} \leftarrow [\,]$;
while True **do**
  $o = \text{argmax}_k\, Z(t,u,k)$;
  append $Z(t,u)$ to $\mathbf{h}$;
  append t to $list_t$;
  append u to $list_u$;
  if $o == 0$ **then**
    $t \leftarrow t+1$;
  else
    $u \leftarrow u+1$;
  end if
  if $t \geq T_{max}$ **then**
    break;
  end if
end while
Return h;

The one-best path distillation loss is then defined as

$$L_{kd} = -\sum_{(t,u)\in\mathbf{h}} \sum_{k=1}^{K} z_t(k,t,u) \log z_s(k,t,u) \tag{3.8}$$

As before, a temperature parameter κ can be applied to scale the distillation loss. The distillation loss is then combined with the original transducer loss in the same way as Equation 3.7. As can be seen from the lattice illustration in subsection 2.3.2, an alignment in RNN-T model has a length of $T+U$, indicating that the space complexity of our distillation method is $\mathcal{O}((T+U)K)$, where K is the size of the output vocabulary. As long as the output vocabulary size is not enormous, the memory consumption would be manageable.

Our method can be also naturally applied on unlabelled data to perform semi-supervised learning. Semi-supervised learning utilises a mixture of both labelled data and unlabelled data to train the model. Utilisation of pseudo labels from unlabelled speech data decoded using a teacher ASR model showed effectiveness in improving student model's performance when a large amount of unlabelled data is used (Weninger et al., 2020; Xiao et al., 2021). In our case, given an input sequence $\mathbf{X}$, a set of sequence hypotheses (with blank) $\mathscr{H}$ with the highest score can be found using fixed-size beam search subsection 2.3.3. The hypotheses $\mathbf{h}$ with the highest score will be responsible for the generation of pseudo label $\mathbf{y}_p$, which is obtained simply by removing the blank symbol $\varnothing$. The distribution alongside $\mathbf{h}$ will be stored as teacher soft-label and the loss calculation can be carried out using Equation 3.8 by combining with the pseudo transcription $\mathbf{y}_p$.

On the contrary, one should be more careful when applying the full-lattice reduced distribution KD method (Panchapagesan et al., 2021) in semi-supervised learning. As the target label sequence is unknown, no complete lattice is properly defined during decoding: beam search decoding only explores partial regions from a set of possible lattices. However, one single full lattice is required in for the distillation method in (Panchapagesan et al., 2021). One possible solution would be using the pseudo transcription obtained with beam search decoding for the full lattice generation. However, the pseudo transcription itself is already inaccurate, it could be possible that the full lattice generated based on it would in low quality and misleads the student model.

3.3.3 KD for Streaming RNN-T

As discussed in subsection 2.3.4, RNN-T is a popular choice for streaming ASR application due to its sequential decoding mechanism and high recognition quality. However, the recognition accuracy still degrades significantly compared to their non-streaming counterparts. To deal with the performance reduction, Kurata and Saon (2020) proposed to use knowledge distillation to improve streaming RNN-T models. A direct distillation from a non-streaming teacher model to a student streaming model tends to fail due to the completely different

position of the posterior peak in the teacher and student lattices. Therefore, a teacher non-streaming ASR model was trained with extra regularisations from a streaming model to delay the posterior peak in the lattice, so that the teacher lattice and student lattice could match (Kurata and Saon, 2020). Then, the KL-divergence over the whole lattice was used as an auxiliary loss. However, this method involves a complicated training process and is computationally inefficient. Our one-best KD method, despite of being computationally inexpensive, enforces the student model to learn an fixed alignment extracted from the non-streaming teacher model, is also not directly applicable when the student model is a streaming model.

To apply our one-best distillation loss in streaming RNN-T models, we propose to shift the teacher one-best path $\mathbf{h}$ alongside the time-axis by an offset ΔT. The shifted one-best path will then be aligned with the student lattice, which leads to a modified version of Equation 3.8

$$L_{kd} = -\sum_{(t,u)\in\mathbf{h}}\sum_{k=1}^{K} z_t(k,t,u)\log z_s(k,t+\Delta T,u) \tag{3.9}$$

Note that after the shift operation, some nodes in $\mathbf{h}$ will be out of the lattice. Such nodes are excluded from the loss computation in Equation 3.9. Obviously, the distillation performance relies heavily on the choice of ΔT. To have a good estimate of ΔT, the posterior peak in the output lattice of both teacher and student model can be inspected. Intuitively, a larger ΔT imposes a softer regularisation on the student model to emit symbol as early as the teacher model, which could lead to a better distillation result but larger latency, whereas a smaller ΔT enforces the student model to emit earlier but also increase the difficulty for the learning of student model. Thus, ΔT can be treated as a factor controlling the trade-off between recognition accuracy and latency.

Chapter 4

Model Selection

As discussed in chapter 3, knowledge distillation in ASR involves two models: a teacher model and a student model. The teacher model tends to be a larger model with better performance, whereas the student model tends to be a smaller model with inferior performance. The goal is then to extract the information learned by the large teacher model, so that it could be utilised to improve the performance of the small student model. Obviously, the performance upper bound would be the teacher model as the student model is imitating the teacher model's behaviours. The lower bound would be the small model trained without knowledge distillation. In this chapter, the architectures of teacher model and student model are introduced.

4.1 Teacher model

In Self-supervised Learning for E2E ASR, the self-supervised ASR framework W2v2 is introduced. In (Baevski et al., 2020), two sizes of W2v2 model is introduced. A comparison between the two models is shown below:

	Base	Large
Num parameters	95M	317M

Table 4.1 Number of parameters in two versions of W2v2 model.

Due to limited computational resources, the small model pre-trained on 960h Librispeech (Panayotov et al., 2015) data is used. According to (Baevski et al., 2020), the convolutional feature extractor reduces the frame rate from 16kHz to 50Hz, i.e 20ms per frame. However, common E2E ASR models have a frame rate of 25Hz, i.e 40ms per frame. This frequency

discrepancy prevents knowledge distillation to be directly applied as the outputs of teacher and student model are in different shape. To cope with this, we design an extra subsampling module to ensure the teacher model having the same frame rate as the student model. Here, various designs of subsampling module are experimented.

Pooling is commonly used to reduce feature dimension. To reduce the frame rate from 50Hz to 25Hz, a subsampling factor of 2 is required. Therefore, an average pooling layer with window size 2 and stride 2 can be applied. This method does not introduce new trainable parameters. However, a simple average operation could result in the loss of information. Therefore, other subsampling modules are also investigated. By first concatenating every two adjacent output frames, a subsampling factor of 2 is achieved. Then, the concatenated feature vector goes through a linear layer appended with an non-linear activation to retain the same feature dimension before concatenation. An illustration of such a subsampling module is shown in Figure 4.1. ReLU and tanh are experimented activation function are experimented. A comparison between all three subsampling modules can be found in section 5.3.

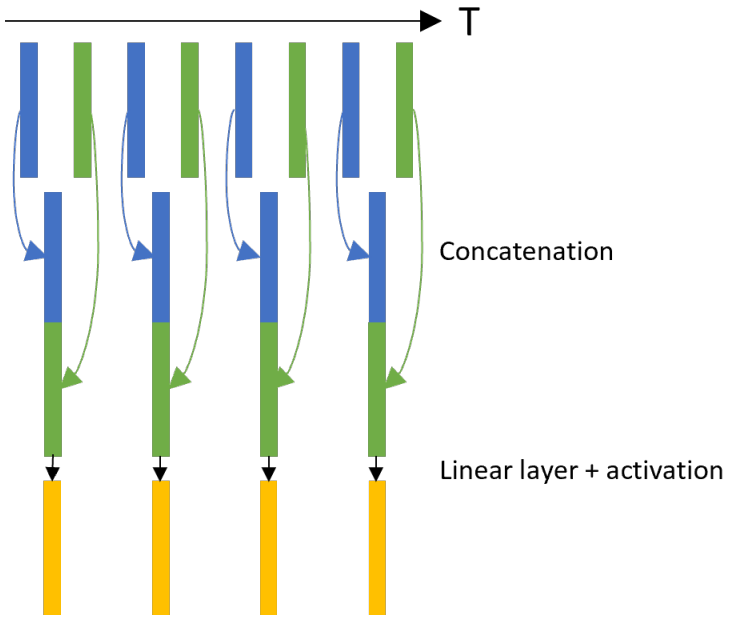


Fig. 4.1 The subsampling module with concatenation operation appended after the W2v2 model.

To investigate the KD capability of W2v2 pre-trained model, we would like to carry out distillation under different ASR frameworks. Although Baevski et al. (2020) only integrated

W2v2 in a CTC architecture, it can be also applied in transducer architecture. This can be easily achieved by adding an extra prediction network to generate an output lattice.

4.2 Student model

Due to the superior sequence modeling capability and good parallelisation as mentioned in subsection 2.1.2, transformer-based models quickly took over RNN-based models and became the basis of today's E2E ASR model (Dong et al., 2018; Synnaeve et al., 2020). Besides the ability to model sequences, a good ASR model should also be able to capture local information which is somehow lacking in the original transformer block in (Vaswani et al., 2017). On the other hand, convolutional layers are efficient in computation and also great at focusing on local information by the usage of convolutional filters, making it an essential component in several ASR models (Han et al., 2020; Li et al., 2019). However, their sequence modelling capability is not as good as transformer models

4.2.1 Conformer

As mentioned before, transformer-based and convolution-based models both have their own strength and weakness. Thus, various methods of combining both architectures have been proposed so that the models will be able to attend to both sequence and local relationships. Wu* et al. (2020) proposed to add an extra head parallel to the original self-attention head, resulting in a multi-branch architecture. The extra branch containing convolutional layers was able to capture local dependencies, showing significant performance gain on several machine translation tasks on mobile devices. Inspired by this, Gulati et al. (2020) proposed an architecture named *conformer* that organically incorporates convolution in the transformer block without adding new branches. Due to its superior performance in ASR tasks, conformer-based ASR models are adopted for student model training in this project.

In E2E ASR, a conventional transformer block (Vaswani et al., 2017) consists of a multi-head self-attention (MHSA) module and a feed-forward module, both with residual connection and normalisation layer. A conformer block extends the original transformer block by introducing a convolution module. Unlike (Wu* et al., 2020), the convolutional module in (Gulati et al., 2020) is integrated in the transformer block. An illustration of the conformer block is shown in Figure 4.2.

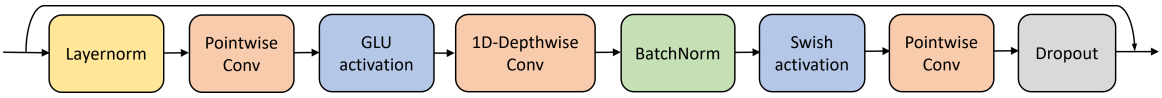


Fig. 4.2 An illustration of the convolution module in a conformer block.

As can be seen, the main components of a convolution module include a point-wise convolution and a depth-wise convolution with different activation functions. After a layer normalisation layer at the beginning of the convolution module, a point-wise convolution, i.e 1x1 convolution, operates on each point over the whole input. Due to the small filter size, it can be used to increase or reduce feature dimension while being computational efficient. The output of the point-wise convolution goes through a gated linear unit (GLU) (Dauphin et al., 2017) activation. Intuitively, a GLU activation control the amount of information that will be passed through like a gating function. A depth-wise convolution (Howard et al., 2017) that separates different input channels follows the GLU activation. The normalised output goes through swish activation and another point-wise convolution layer to retain the same channel dimension at the beginning of the convolution module. A dropout layer is appended at the end of the convolution module to increase the generalisation ability. This convolution module is then organically incorporated in a transformer block, which forms the conformer block as shown in Figure 4.3.

Apart from the usage of the convolution module, Gulati et al. (2020) also modified the position of the feed-forward module and MHSA module in the original transformer block, which further improves the model performance. Inspired by (Lu* et al., 2020), two half-step feed forward modules were adopted in the conformer block, sandwiching the MHSA module and convolution module (see Figure 4.3). The MHSA module was combined with a relative positional embedding to promote better generalisation in the MHSA as the relative length is considered when calculating the attention matrix (Dai et al., 2019).

4.2.2 Streaming Conformer

To investigate the distillation performance in a streaming RNN-T model, the conformer block should be adapted accordingly to limit the usage of future context. Unlike recurrent layers, a conformer block does not support streaming initially due to the MHSA module and the convolution module. However, certain modifications can be made to enable streaming scenario. Inspired by (Li et al., 2021a), we add an attention mask to mask out attention scores from certain future frames when calculating the output of the current frame. An illustration

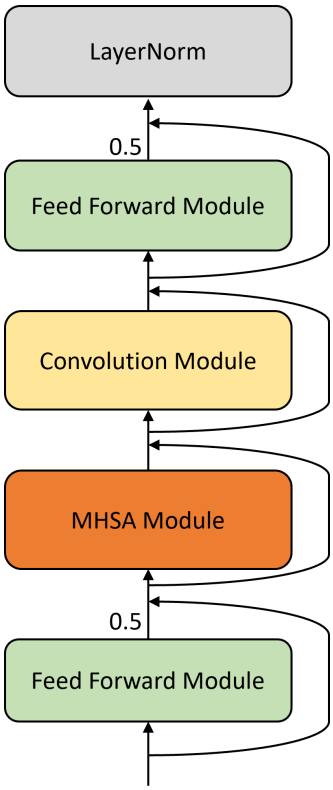


Fig. 4.3 An illustration of the conformer block.

of such mask is shown in Figure 4.4.

The convolution operation in a conformer block also results in lookahead as the filters cover both previous frames and future frames. To deal with this, causal convolution (van den Oord et al., 2016) can be applied so that the filter will only takes history frames into consideration (see).

Depending on the number of encoder layer and dimension of feed-forward modules, three sizes of conformer models were proposed in (Gulati et al., 2020). In this project, the smallest conformer with approximately 10M parameters is adopted. In a knowledge distillation scenario, the student model is usually much smaller than the teacher model, which allows for larger potential performance improvement and higher model compression rate. The conformer-based encoder is then integrated in CTC and RNN-T architecture to form an E2E ASR system.

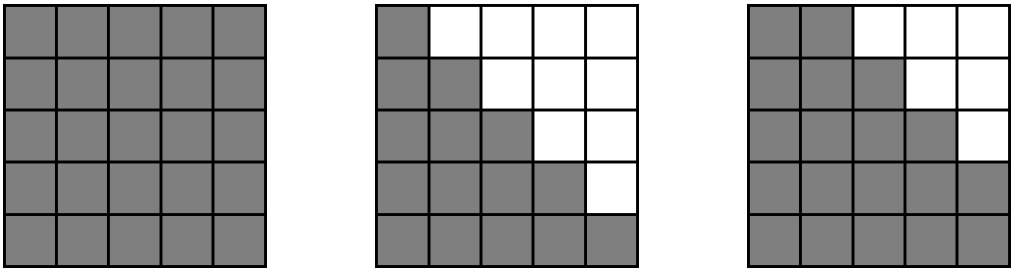


Fig. 4.4 Three examples of attention mask in streaming conformer. The first mask attend to the whole sequence. The second mask masked out the attention score from all future frames, which is equivalent to no lookahead. The last mask allows a limited future context with maximum 1 lookahead.

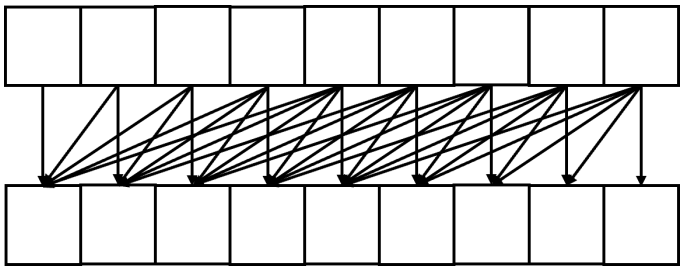


Fig. 4.5 An illustration of 1D causal convolution. The filter is not centred and only incorporates previous frames for output calculation.

Chapter 5

Results

In this chapter, multiple experiments with W2v2 adaptation and knowledge distillation are carried out. In section 5.1 and section 5.2, details about the dataset, evaluation metrices and model hyperparamters are given. Integration of W2v2 model in CTC and RNN-T framework are investigated in section 5.3. Frame-level distillation in CTC models are carried out in section 5.4. In section 5.5, we demonstrate the effectiveness of our one-best distillation loss in transducer models and compare our method against existing methods. In the end, preliminary KD results on streaming transducer models are also presented.

5.1 Dataset

In this project, Librispeech (Panayotov et al., 2015) dataset is used for training and evaluation. As a reproduction and extension of the results in (Baevski et al., 2020), this dataset is used for fine-tuning a pre-trained W2v2 model. Librispeech corpus contains approximately 1000 hours of 16kHz read English speech from audio books. The training set of Librispeech is paritioned into 3 subsets: train-clean-100h, train-clean-360h and train-other-500h. The WERs are first calculated on each speaker in Librispeech corpus using an external acoustic model trained on another dataset and the speakers are ranked based on their WERs. The "clean" subsets are collected from speakers with lower WER, whereas the "other" subset is collected from speakers with higher WER, therefore more challenging. Development sets and test sets are also splitted in "clean" and "other" for performance evaluation. More detailed information on Librispeech dataset is shown in Table 5.1.

Word-error-rate (WER) on test-clean and test-other is used as the quantitative performance measurement of all models. For all experiments below, the model with the lowest WER on

Subset	Total length	Num utterances	Avg utterance duration
dev-clean	5.4h	2703	7.2s
test-clean	5.4h	2620	7.4s
dev-other	5.3h	2864	6.7s
test-other	5.1h	2939	6.3s
train-clean-100	100.6h	28539	12.7s
train-clean-360	363.6h	104014	12.6s
train-other-500	496.7h	148688	12s

Table 5.1 A summary on all subsets in Librispeech dataset.

dev-other subset is chosen as the best model, which will then be evaluated on test-clean and test-other subset.

5.2 Model Details

For teacher model, the base W2v2 model (W2v2(B)) pre-trained on 960h Librispeech data is selected. This model has 12 transformer blocks, each with 8 attention heads and an inner dimension of 3072 in the feed forward module. As mentioned in section 4.1, a subsampling module is appended after the transformer to further reduce the output frame-rate from 50 Hz to 25Hz. Then, the base W2v2 model and the subsampling module together will be integrated in CTC and RNN-T framework as an encoder. For output vocabulary, Byte-Pair-Encoding (BPE) (Sennrich et al., 2016) of size 256 is adopted instead of grapheme in (Baevski et al., 2020) as we expect BPE to yield better results. A performance comparison between BPE and grapheme is also made in the following section. For W2v2 transducer models, a single LSTM layer with 320 hidden units is used as prediction network. The model sizes of all W2v2 models are shown in Table 5.2. Unlike most ASR models that uses MFCC or filter bank features, W2v2 model uses raw waveform as input. Therefore, each input is in the shape of $(T, 1)$, where T is the length of the utterance in number of frames. A 10-second utterance at 16kHz, for example, has a T of 160,000.

For the student model, the smallest conformer in (Gulati et al., 2020) is adopted to build encoder in CTC and transducer framework. The same size of the output vocabulary is used as the teacher W2v2 models. Like the teacher model, the student transducer model also has a single LSTM layer with 320 hidden units as prediction network. The size of the student models is listed in Table 5.3. The input to the student model is the commonly used 83-dimensional fil-

Model Name	Num Params
W2v2(B) + avgpooling subsample + CTC	96.4M
W2v2(B) + avgpooling subsample + transducer	98M
W2v2(B) + concat subsample + CTC	97.6M
W2v2(B) + concat subsample + transducer	99.2M

Table 5.2 Number of parameters in different teacher W2v2-based models. The number listed here are taken from models using word piece as target. W2v2(B) stands for the base model in (Baevski et al., 2020).

ter bank features without pitch at a frame-rate of 100Hz. For a 10-second utterance at 16kHz, its corresponding input sequence is in the shape of $(100, 83)$. SpecAugment (Park et al., 2019) is used with mask parameter (F=27) and 10 masks with a maximum time-mask ratio of 0.05 to increase the model's generalisation ability. The input frame-rate will then be subsampled by a convolutional module from 100Hz to 25Hz (40ms per frame), which is the same as the teacher model. The subsampled input feature will then be fed to the conformer network.

Model	Conformer(S) + CTC	Conformer(S) + RNN-T
Num Params	9.7M	8.75M
Num Encoder Block	16	16
Encoder dim	144	144
Num Head	4	4
Conv Kernel Size	31	31
Decoder layer	1	-
Decoder Dim	320	-

Table 5.3 Number of parameters in different student conformer-based models. No decoder is needed in a CTC model.

As can be seen, our knowledge distillation setup results in a compression rate of 11.2 and 10.2 for CTC and transducer framework respectively. With such a large compression rate, an obvious performance degradation should be expected from the student models compared to the teacher model. However, this also leaves out more space for performance improvement after KD.

5.3 W2v2 Teacher Model

First, the impact of different subsampling modules designed in section 4.1 is investigated. The subsampling modules are integrated in the W2v2 model and trained using the clean-100h subset with BPE256 as output vocabulary. The WERs of W2v2 CTC and W2v2 transducer are shown in Table 5.4 and Table 5.5 respectively. For comparison, models without subsampling module are also trained. All results are obtained using beam search decoding with a beam size of 8. An insertion penalty of 0.25 is used as this gives the lowest WER on dev-other set. The modified spec-augment proposed in (Baevski et al., 2020) is adopted to improve the generalisation ability. The hyperparameters used for fine-tuning the teacher W2v2v model is shown in section A.1.

CTC models	dev		test	
	clean	other	clean	other
Grapheme				
Original W2v2+CTC, no subsample (Baevski et al., 2020)	6.1	13.5	6.1	13.3
Ours, W2v2 + no subsample	6.1	13.8	6.1	13.3
BPE256				
W2v2 + avgpooling	6.2	14.0	6.2	13.3
W2v2 + concat_relu	6.1	13.8	6.1	13.3
W2v2 + concat_tanh	6.1	13.9	6.1	13.2
W2v2 + no subsample	6.3	14.0	6.6	13.5

Table 5.4 WERs of W2v2 CTC models with different subsampling modules. The models are fine-tuned on clean-100h subset. The first row is the official model provided by (Baevski et al., 2020).

As can be seen, the concat_tanh subsampling module achieves the lowest WER among all three subsampling modules. Surprisingly, the new subsampling module reduces the WER in both CTC and transducer models, suggesting that the subsampling modules are well designed. Although the usage of BPE256 as output vocabulary does not improve teacher model's performance in CTC models (see Table 5.4), a slight WER reduction can be observed when BPE256 is adopted in transducer-based models (see Table 5.5) compared to the grapheme output in the official implementation (Baevski et al., 2020). This is also as expected as transducer-based models are able to model output dependency because of the prediction network, which is more crucial for byte-pair-encoding than for grapheme. For consistency consideration, BPE256 will be used as the output vocabulary in all the following experiments.

Transducer models	dev		test	
	clean	other	clean	other
Grapheme				
W2v2 + avgpooling	5.5	12.8	5.5	12.5
W2v2 + concat_relu	5.2	12.7	5.3	12.3
W2v2 + concat_tanh	5.2	12.7	5.3	12.1
W2v2 + no subsample	5.4	13.0	5.3	12.5
BPE256				
W2v2 + avgpooling	5.0	12.4	5.3	12.0
W2v2 + concat_relu	5.2	12.3	5.2	11.9
W2v2 + concat_tanh	5.1	12.2	5.2	11.8
W2v2 + no subsample	5.0	12.4	5.3	12.2

Table 5.5 WERs of W2v2 transducer models with different subsampling modules. The models are fine-tuned on clean-100h subset.

5.4 CTC KD Results

Here, experiments on CTC frame-level distillation are performed and the results are investigated. The W2v2 CTC model with a concat_tanh subsampling module (see the second last row in Table 5.4) is selected as the teacher model. The student model is a conformer CTC model (see Table 5.3) described in section 5.2.

Experiments are first carried out on clean-100h subset of Librispeech. The logits output of the all utterances in clean-100h generated by the teacher model are stored for distillation label preparation. We choose to store the logits in instead of the softmax output since we would also like to investigate the effect of different temperature κ in frame-level CTC KD. As an indicator of the soft-labels' quality, the WER of the teacher model on clean-100h is shown in Table 5.7. To demonstrate the effectiveness of KD, a baseline conformer CTC model is trained using hard-label of clean-100h without KD. Three different training strategies for CTC frame-level KD are explored and will be referred to as ST1, ST2 and ST3 respectively:

1. First, train the student model from a random initialisation only with soft-label. Then fine-tune the well-trained model with frame-level KD using soft and hard-label at the same time. During fine-tuning, a much smaller learning rate is picked to prevent the model from diverging and catastrophic forgetting.

2. Train the student model from a random initialisation with frame-level KD using soft and hard-label at the same time. This strategy does not require fine-tuning.

3. Initialise the student model from a well-trained conformer CTC model. Then fine-tune this model with both soft and hard-label. As in ST1, a much smaller fine-tuning learning rate is used.

CTC model	dev		test	
	clean	other	clean	other
(a) Conformer, no KD (baseline)	10.7	28.1	11.3	28.7
W2v2 (upper bound)	6.1	13.9	6.1	13.2
ST1				
Only soft-label, κ=1.0	9.8	26.0	10.2	26.3
Only soft-label, κ=2.5	10.2	25.9	10.5	26.6
(b) Only soft-label, κ=4.0	9.8	25.0	10.1	25.5
from (b), $\lambda = 0$, κ=4.0	9.0	24.0	9.3	24.0
from (b), $\lambda = 0.01$, κ=4.0	9.3	24.4	9.6	24.5
from (b), $\lambda = 0.05$, κ=4.0	9.0	24.0	9.3	24.0
from (b), $\lambda = 0.1$, κ=4.0	9.0	24.0	9.3	24.1
ST2				
λ=0.1, κ=1.0	9.3	24.9	9.4	25.2
λ=0.1, κ=2.5	9.8	25.9	9.8	26.3
λ=0.1, κ=4.0	9.3	25.1	9.6	25.5
ST3				
λ=0.1, κ=1.0	9.8	26.1	10.1	26.3
λ=0.1, κ=2.5	10.1	26.3	10.1	26.3
λ=0.1, κ=4.0	9.8	26.3	9.5	25.5

Table 5.6 CTC distillation results. λ is the weight factor of L_{kd} in L introduced in Equation 3.5. In ST3, the model are initialised from the baseline model (a) on the first row of the table. All models are trained on clean-100h subset.

The KD results of three strategies are shown in Table 5.6. Hyperparameters used during training and fine-tuning can be found in section A.2. For each strategy, a few combinations of λ and κ are experimented. For ST1, three temperature κ from $\{1.0, 2.5, 4.0\}$ are explored in the first step of the training that purely relies on unlabelled data. The best performing model is then fine-tuned with different λ using the same temperature κ on which it was pre-trained in the first step. For ST2 and ST3, the weight λ of the distillation loss is fixed to 0.1 and three κ from $\{1.0, 2.5, 4.0\}$ are experimented in ST2 and ST3. The same decoding settings are used as for the teacher W2v2 CTC models. As can be seen, all three strategies of frame-level KD significantly improves the student model's performance. Among all strategies, ST1 yields the lowest WER. For ST1, we also listed the WERs intermediate model

which is trained only with soft-labels without ground truth hard label. Interestingly, with only soft-labels as supervision signal, the model trained with $\kappa = 4.0$ (fifth row in Table 5.6) already outperforms the baseline conformer model (first row in Table 5.6) with a relative WER reduction of 10.6% and 11.1% on test-clean and test-other respectively. This could be due to the reason that the soft-labels have very high accuracy and contain much richer information compared to the one-hot encoded ground truth hard-labels. After fine-tuning with a combination of hard and soft-labels, a larger relative WER reduction of 17.7% and 16.4% on both test sets is achieved because of the regularisation brought by the hard-labels (eighth row in Table 5.6). For ST1, the variation of λ does not affect the final WER obviously. Among all three strategies, fine-tuning a well-trained conformer CTC student (ST3) yields the worst performance.

Note that when applying Equation 3.3 in our distillation setup, a slight length discrepancy between the teacher soft-label and student output can exist. As the W2v2 teacher model receives raw waveform at 16kHz as input, a stack of 7 convolutional layers reduces the input frequency to 25Hz. The padding operation in the convolutional layers will cause unavoidable length mismatching between teacher label and student output. After careful inspection, it is found that the length difference is never larger than 1. Therefore, the extra frame in the longer sequence will be discarded and not contribute to the distillation loss defined in Equation 3.3. Since the last frame is a $\varnothing$ under most circumstances, discarding it will only have minor effects on KD training.

We then extend our experiments to a semi-supervised learning scenario and investigate how KD would perform when extra unlabelled data is utilised. Besides the clean-100h data, on which the teacher model is trained, the clean-360h subset is also incorporated in the training set. However, we treat it as an unlabelled dataset and only generates the soft-labels from it without utilising the ground truth transcription. Note that by doing so, the total amount of labelled speech data used during training is still 100h. The teacher model's WER on clean-360h is listed in Table 5.7. The final training set is then increased to a mixture of 460h speech data, of which only 100h has corresponding ground truth transcription. According to the results in Table 5.6, we used $\kappa = 4.0$ as this yields the best results in ST1. Experiments with purely soft-label training is shown below in Table 5.8. As can be seen, a relative WER improvement of 25.7% and 29.6% is observed on test-clean and test-other respectively when extra soft-labels from clean-360h are provided. An even larger WER reduction should be expected if the model is further fine-tuned with a combination of soft and hard-labels like in

ST1.

Model	clean-100h	clean-360h
W2v2+concat_tanh +CTC	2.4	5.8

Table 5.7 WER of the teacher CTC model on clean-100h and clean-360h. The model is obtained by fine-tuning on clean-100h.

Model name	dev		test	
	clean	other	clean	other
clean-100h, Conformer, no KD (baseline)	10.7	28.1	11.3	28.7
460h soft-label, $\lambda = 0.0$, κ=4.0	8.2	20.3	8.4	20.2

Table 5.8 CTC KD results using extra unlabelled data. Results are obtained using the soft-labels of 460h mixture set.

5.5 Transducer KD results

In this section, the KD results on non-streaming transducer models are reported. Preliminary results on streaming transducer KD are also mentioned.

5.5.1 Non-streaming Transducer

Here, KD experiments on non-streaming transducer models are carried out. The W2v2 transducer model with a concat_tanh subsampling module (see Table 5.5) is chosen as the teacher model. The student model is a conformer transducer model described in Table 5.3. Experiments are first carried out using clean-100h data as labelled dataset. For comparison, we also implement the full-lattice collapsed distribution distillation loss in (Panchapagesan et al., 2021) and evaluate its performance on clean-100h subset. The teacher model's WER on clean-100h subset is shown in Table 5.9. Unlike CTC distillation, ST1 (see section 5.4) cannot be adopted in transducer KD: the information carried in the one-best path or the full-lattice collapsed distribution is very limited and is only suitable to be used as an auxiliary loss to regularise training in transducer KD. All experiments share the same temperature κ=1.0. For full-lattice collapsed distribution, we use λ=0.001 as suggested in (Panchapagesan et al.,

2021). We set $\lambda = 0.1$ and $\kappa = 1.0$ for our one-best distillation without extra tuning. A conformer transducer model is also trained without KD as a baseline model. The same decoding settings are adopted as before. The results are shown below in Table 5.10.

Model	clean-100h	clean-360h	other-360h
W2v2 + concat_tanh + transducer	1.3	4.9	6.4

Table 5.9 WER of the teacher W2v2 transducer model on clean-100h, clean-360h and other-360h. Results are decoded with beam size 8 and insertion penalty 0.25.

As can be seen, our one-best distillation loss significantly improves the student model's performance by a relative WER reduction of 12.6% and 9.2% on test-clean and test-other respectively compared to the baseline student model trained without KD. Compared to the method in (Panchapagesan et al., 2021), our method achieves comparable performance with a slightly higher WER on clean set but a lower WER on other set, suggesting that our one-best KD is beneficial to the student model training.

Transducer model	dev		test	
	clean	other	clean	other
clean-100h hard-label, conformer (baseline)	8.3	22.5	8.7	22.9
clean-100h hard-label, W2v2 (upper bound, teacher)	5.1	12.2	5.2	11.8
Full lattice reduced distribution KD(Panchapagesan et al., 2021)				
clean-100h, λ=0.001, κ=1.0	7.1	20.7	7.5	21.0
One-best KD (ours)				
clean-100h, λ=0.1, κ=1.0	7.2	20.0	7.6	20.8
Semi-supervised + one-best KD				
clean-100h + clean-360h (unlabelled), conformer no KD	5.7	15.3	5.8	15.1
clean-100h + clean-360h (unlabelled), λ=0.1, κ=1.0	5.4	14.9	5.7	14.6
clean-100h + other-360h (unlabelled), λ=0.1, κ=1.0	6.0	12.5	5.9	12.8

Table 5.10 Transducer KD results. Experiments are carried out on clean-100h data and two extra unlabelled subsets: clean-360h and other-360h. Results are decoded with beam size 8 and insertion penalty 0.25. All student models are conformer transducer model.

Experiments with semi-supervised learning are also carried out for our transducer one-best KD. Similar to CTC experiments, the clean-360h subset is used as unlabelled dataset for soft-label generation. To further demonstrate the robustness of our one-best KD loss, a 360h subset from other-500h subset is randomly selected to represent distillation labels

with slightly lower qualities, as the other-500h subset is more challenging. These two extra unlabelled dataset simulate the scenario where the unlabelled data shows different difficulties and distillation labels are in different qualities. The WER of our teacher model on these three subsets are shown in Table 5.9, which can be seen as an indicator of the quality of distillation labels. As mentioned earlier, information in the one-best path is not rich enough to train student model. Therefore, pseudo transcriptions are first obtained by beam search decoding with beam size 8 for unlabelled data. They are then used with the one-best distillation labels described in subsection 3.3.2 to define the total loss in Equation 3.7. As a comparison, a baseline model is trained solely using clean-100h ground truth and clean-360h pseudo transcriptions without KD. The results are shown in Table 5.9.

As can be seen, using a mixture of ground truth and pseudo transcriptions alone without KD results in a significant WER reduction compared to the baseline model in Table 5.10. This is also expected as the pseudo transcriptions have high accuracy (see Table 5.9). When clean-360 is used as extra unlabelled data, the WER is further improved, achieving a relative reduction of 34.5% and 36.2% on test-other and test-clean respectively. When compared with the teacher model, we see that after performing semi-supervised learning, the WER is very close to the upper limit (especially for clean set). When other-360h set is used, the WER is still significantly improved compared to the baseline conformer model. Since the other-360h set contains more challenging utterances than clean-360h and shares more similarities with the test-other evaluation set, using it as extra unlabelled data leads to a larger WER reduction on test-other set, which also significantly closes the gap between teacher and student model. We can thus infer that the model's performance is highly dependent on the unlabelled dataset being used.

5.5.2 Streaming Transducer

In this section, one-best KD on streaming transducer models are explored. As discussed in subsection 3.3.3, the misalignment of posterior peak between teacher and student lattice poses challenges on the direct application of our one-best distillation loss. To address this issue, the time shift ΔT is introduced (see subsection 3.3.3).During experiments, ΔT is carefully selected after analysing the output lattice of teacher and student model. Two distillation strategies are experimented:

1. Initialise from a well-trained streaming conformer. Then fine-tune it using one-best KD with both hard and soft-labels at a certain time shift ΔT.

2. Initialise randomly and train the streaming conformer student model with one-best KD using hard and soft-label from the beginning.

We set λ=0.1 and κ=1.0 for all experiments involving one-best KD. Three different ΔT from $\{0, 6, 7\}$ are experimented. The last two values are inferred by observing the lattice output of teacher and student models. For comparison, we also trained student models with 0 and limited number of lookahead. As an indicator of the model's streaming capability, we calculate the emission delay of each model. Since the non-streaming model tends to emit symbol earlier in the lattice, we treat the posterior peak in the non-streaming conformer baseline model (second row in Table 5.11, denoted as Cfms-B) as the earliest possible emission time. The emission delay for streaming models is then defined as the offset of posterior peak in the lattice compared to Cfms-B (see first row in Table 5.11) plus the number of lookahead required in that model. For example, if a model has m encoder lookahead and emits the symbols n frames later in average than Cfms-B, the total emission delay is then $m+n$.

The WERs of different streaming conformers and their emission delay are shown in Table 5.11. The posterior peak in streaming and non-streaming model's output lattice is also illustrated in Figure 5.1. Note that all results here are decoded with beam size 20, which is different to the previous decoding strategy.

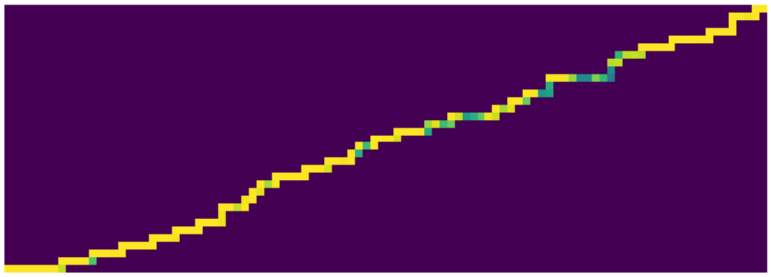
Transducer model	dev		test		Emission delay
	clean	other	clean	other	
W2v2 transducer	5.1	12.2	5.2	11.8	-
Conformer-transducer (Cfms-B)	8.3	22.5	8.7	22.9	-
No lookahead					
(a) streaming conformer (baseline)	11.2	28.4	12.2	29.6	7
fine-tuned from (a), KD with $\Delta T = 6$	10.7	26.7	11.3	28.0	6-7
fine-tuned from (a), KD with $\Delta T = 7$	10.6	26.6	11.3	27.6	7
$\Delta T = 0$, λ=0.1, κ=1.0	20.2	39.7	20.5	41.4	0-1
$\Delta T = 6$, λ=0.1, κ=1.0	11.3	28.5	11.9	29.3	6-7
$\Delta T = 7$, λ=0.1, κ=1.0	11.2	27.3	11.9	28.7	7
Limited lookahead					
3 lookahead	11.0	28.1	11.3	28.8	7
4 lookahead	10.8	27.9	11.3	28.8	7
8 lookahead	10.5	27.0	10.7	28.1	7

Table 5.11 Streaming transducer KD results. All experiments are carried out using clean-100h subset. The first two rows are offline transducer models for performance comparison.

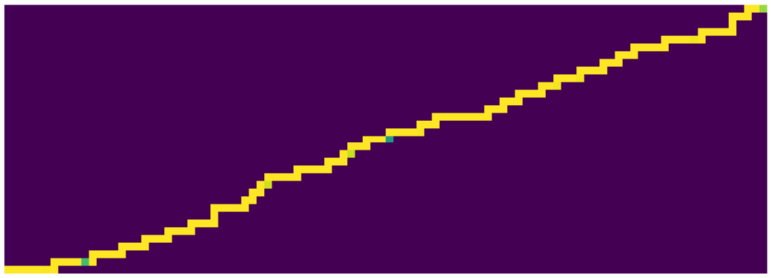
As can be seen, both distillation strategies successfully reduce the WER on test sets, showing the effectiveness of the shifted one-best KD loss. Fine-tuning a well-trained streaming model using one-best KD yields better results, achieving a relative WER reduction of 7.4% and 6.8% under the KD setting with $\Delta T = 7$. As expected, the WER improvement depends on the value of ΔT, which allows a trade-off between emission delay reduction and WER reduction. When hard alignment $\Delta T = 0$ is used, the student model learns terribly. As the streaming transducer tends emit symbols later, this results is also as expected because the contextual information provided for a streaming conformer with no lookahead is not enough to align with the teacher one-best path. Although its output posterior peak's position is roughly aligned with the teacher model after the distillation, the overall recognition accuracy is very poor. This further supports our decision of introducing time shift for streaming transducer KD.

The bottom three rows in Table 5.11 show that allowing extra lookahead into the future also improves the WER. However,the posterior peaks of them (see Figure 5.1) are not brought earlier after allowing extra lookahead compared to the previous models with no lookahead. This results in an even larger emission delay as more future contexts are required during decoding. Compared the streaming conformer with 4 lookahead (second last row in Table 5.11) with the student models trained one-best KD, we see that the one-best distillation not only yield a smaller WER, but also produces a smaller emission delay. Therefore, our shifted one-best KD loss could be a potential solution for improving streaming transducer's performance.

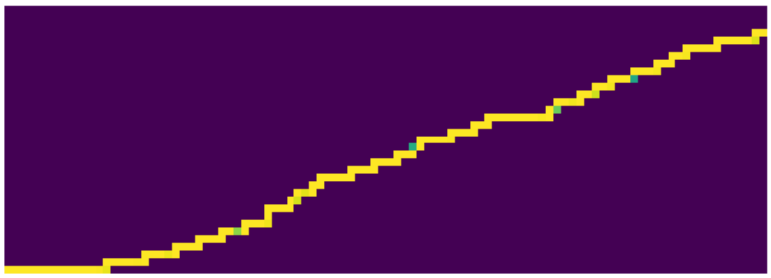
In Figure 5.1, the partial lattice generated by different models using the same training example is shown. As expected, the non-streaming conformer models tend to emit output symbols more quickly (see Figure 5.1a and Figure 5.1b). The streaming student model tends to delay the emission to wait for more contextual information (see Figure 5.1c and Figure 5.1d). After allowing extra lookahead, the position of the posterior peak shows no obvious change (see Figure 5.1e).



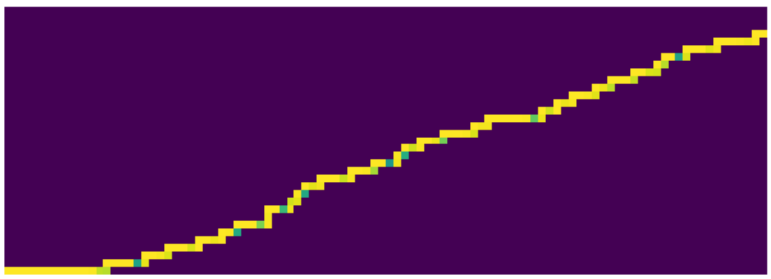
(a) Teacher W2v2 transducer model.



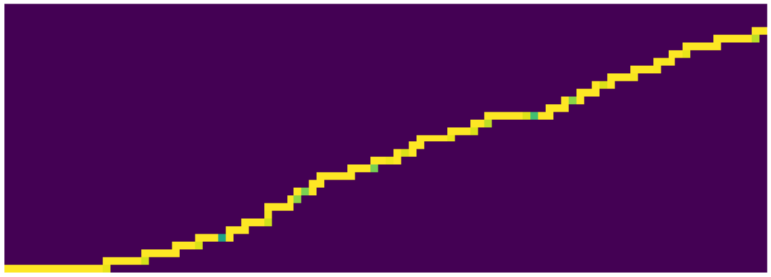
(b) Non-streaming conformer model (Cfms-B).



(c) Streaming conformer transducer with no lookahead trained without KD.



(d) Streaming conformer transducer fine-tuned using one-best KD with $\Delta T = 6$.



(e) Streaming conformer transducer with 4 lookahead without KD.

Fig. 5.1 Posterior peak in streaming and non-streaming transducer models. Horizontal axis is t-axis and vertical axis is u-axis. All figures are drawn using the same utterance selected from clean-100h data. The yellow nodes represent the one-best path in the lattice. We limit the t and u for better visualisation. A brighter colour indicates a higher probability.

Chapter 6

Conclusion and Future Work

6.1 Summary

In this thesis, the usage of self-supervised ASR models for KD is explored. The original self-supversied W2v2 model in (Baevski et al., 2020) is adapted for the need of KD, leading to a slight performance gain compared to the original architecture. This adapted W2v2 model is then used as teacher model in the context of KD after integration in two ASR frameworks: CTC and RNN-T.

Experiments are carried out using Librispeech dataset and evaluated on clean and other test set in Librispeech. In CTC framework, the proposed KD strategy under clean-100h training set is experimented and compared with a baseline CTC model trained without KD. With a significant WER reduction, the effectiveness of using W2v2 model as teacher model in KD is confirmed. A novel memory-efficient distillation method for transducer-based ASR models that exploits the one-best alignment information in the teacher transducer's output lattice is proposed and compared with the existing method (Panchapagesan et al., 2021). Both methods are implemented and experiments with clean-100h training set are carried out. The results of the proposed distillation method show a relative WER reduction of 12.6% and 9.2% on two evaluation sets compared to a student model trained without KD, which is comparbale to the existing method (Panchapagesan et al., 2021). The proposed method is then applied to streaming transducer models. By introducing a time shift operation while performing KD, the proposed method successfully improves the WER of the baseline streaming transducer student model with no lookahead by 7.4% and 6.8% on two evaluation sets. Without requiring any lookahead, this streaming transducer model fine-tuned with our proposed KD loss yields even lower WERs than a streaming transducer model with 4

lookahead without KD.

The KD experiments are also extended to semi-supervised learning scenario by utilising extra unlabelled data. The proposed CTC distillation strategy yields a larger relative WER improvement of 25.7% and 29.6% on two evaluation sets with clean-360h subset as extra unlabelled data, further closing the gap between student and teacher models. Semi-supervised learning experiments were also carried out using the transducer KD proposal on non-streaming transducer models with two extra unlabelled datasets: clean-360h and other-360h. Experiments with both unlabelled datasets produce a significant larger WER reduction on both evaluation sets compared to the student models distilled with only labelled clean-100h subset. This also validates the robustness of the proposed distillation method as unlabelled dataset of different qualities are used.

In summary, the feasibility of using self-supervised ASR models as teacher model in knowledge distillation for training a good small student model is demonstrated. As a self-supervised model only requires a small amount of fine-tuning data to achieve high recognition accuracy, this thesis shows that a good student model can be trained with proper KD strategy when supervised by self-supervised ASR models with only limited amount of labelled speech data. This would be particular meaningful for scenarios where only limited amount of labelled speech data is available and a small model with good performance is required.

6.2 Future Work

Multiple directions of future work are possible. From the perspective of semi-supervised learning, further experiments with unlabelled data could be carried out. Currently, only the one-best KD method has been applied to semi-supervised transducer models training. To obtain a more complete comparison, it would be helpful to also carry out semi-supervised learning experiments on the full-lattice collapsed distribution method(Panchapagesan et al., 2021). This would provide more insights into the advantages and drawbacks of both methods. To push the student model's performance boundary. experiments with the whole 960h Librispeech dataset, or even data from other datasets can be carried out. Such experiments would provide more information about the robustness of our KD method. For streaming conformer, the preliminary results are only obtained using labelled data. It would be interesting to observe how much benefits semi-supervised learning could bring to streaming transducer models.

In multiple experiments, the temperature parameter κ is not tuned. We noticed that the choice of κ significantly affects the magnitude of the distillation loss L_{kd}. Thus, the KD loss weight λ should also be changed accordingly to keep balance between two distillation loss and original loss. Better distillation results could be obtained if this had been done.

Currently, the proposed one-best transducer KD method only selects the one-best path for distillation label generation. It would be very useful to investigate the posterior probability of the one-best decoded label sequence compared to other hypotheses, so that the relative importance of the one-best sequence can be evaluated. This method can be further extended to include multiple paths, which results in a weighted sum of the loss associated with each path.

For streaming transducer models, a shifted version of one-best path is adopted to cope with the mismatching of posterior peaks in teacher and student models. However, multiple ΔT would be sensible and training all of them makes the KD process troublesome. Also, the value of ΔT may also depend on many other factors such as training set and model architectures, which further complicates the training process. A shifted window may be applied to address the aforementioned issue by allowing each node in the one-best path to shift with ΔT chosen from a window. This would hopefully improve the flexibility of the proposed distillation method.

References

Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. *Proc.NIPS*.

Baevski, A., Schneider, S., and Auli, M. (2019). vq-wav2vec: Self-supervised learning of discrete speech representations. In *Proc. ICLR*, New Orleans.

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., and Dhariwal (2020). Language models are few-shot learners. In *Proc.NIPS*.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proc.ICASSP*, Shanghai.

Chen, G., Choi, W., Yu, X., Han, T., and Chandraker, M. (2017). Learning efficient object detection models with knowledge distillation. *Proc.NIPS*.

Chen, X., Wu, Y., Wang, Z., Liu, S., and Li, J. (2021). Developing real-time streaming transformer transducer for speech recognition on large-scale dataset. In *Proc.ICASSP*, Toronto.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proc. ACL*, Florence.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proc.ICML*, Toulon.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *Proc.ICASSP*, Alberta. IEEE.

Doutre, T., Han, W., Chiu, C.-C., Pang, R., Siohan, O., and Cao, L. (2021). Bridging the gap between streaming and non-streaming asr systems bydistilling ensembles of ctc and rnn-t models. *arXiv preprint arXiv:2104.14346*.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*.

Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

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

Graves, A., Fernández, S., and Schmidhuber, J. (2005). Bidirectional lstm networks for improved phoneme classification and recognition. In *International conference on artificial neural networks*.

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

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proc.CVPR*, Las Vegas.

Henaff, O. (2020). Data-efficient image recognition with contrastive predictive coding. In *Proc.ICML*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, L., Xu, J., Sun, J., and Yang, Y. (2017). An improved residual lstm architecture for acoustic modeling. In *2017 2nd International Conference on Computer and Communication Systems (ICCCS)*, Nanjing.

Huang, M., You, Y., Chen, Z., Qian, Y., and Yu, K. (2018). Knowledge distillation for sequence model. In *Proc. Interspeech*, Graz.

Jain, M., Schubert, K., Mahadeokar, J., Yeh, C.-F., Kalgaonkar, K., Sriram, A., Fuegen, C., and Seltzer, M. L. (2019). Rnn-t for latency controlled asr with improved beam search. *CoRR*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. *Proc.ICLR*.

Jegou, H., Douze, M., and Schmid, C. (2010). Product quantization for nearest neighbor search. *Proc.TPAMI*.

Jiang, D., Lei, X., Li, W., Luo, N., Hu, Y., Zou, W., and Li, X. (2019). Improving transformer-based speech recognition using unsupervised pre-training. *Proc. Interspeech*.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In *Advances in psychology*. Elsevier.

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

Kurata, G. and Saon, G. (2020). Knowledge distillation from offline to streaming rnn transducer for end-to-end speech recognition. In *Proc. Interspeech*, Shanghai.

Li, B., Gulati, A., Yu, J., Sainath, T. N., Chiu, C.-C., Narayanan, A., Chang, S.-Y., Pang, R., He, Y., Qin, J., et al. (2021a). A better and faster end-to-end model for streaming asr. In *Proc.ICASSP*, Toronto.

Li, C., Shi, J., Zhang, W., Subramanian, A. S., Chang, X., Kamo, N., Hira, M., Hayashi, T., Boeddeker, C., Chen, Z., and Watanabe, S. (2021b). ESPnet-SE: End-to-end speech enhancement and separation toolkit designed for ASR integration. In *Proceedings of IEEE Spoken Language Technology Workshop (SLT)*, pages 785–792. IEEE.

Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., Nguyen, H., and Gadde, R. T. (2019). Jasper: An end-to-end convolutional neural acoustic model. *Proc. Interspeech*.

Lu*, Y., Li*, Z., He, D., Sun, Z., Dong, B., Qin, T., Wang, L., and yan Liu, T. (2020). Understanding and improving transformer from a multi-particle dynamic system point of view. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. (2019). fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. In *Proc. ICASSP*, South Brisbane.

Panchapagesan, S., Park, D. S., Chiu, C.-C., Shangguan, Y., Liang, Q., and Gruenstein, A. (2021). Efficient knowledge distillation for rnn-transducer models. In *Proc.ICASSP*, Toronto.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., and Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *Proc. Interspeech*.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*.

Sainath, T. N., Pang, R., Rybach, D., García, B., and Strohman, T. (2020). Emitting word timings with end-to-end models. In *Proc.Interspeech*, Shanghai.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proc.ACL*, Berlin.

Synnaeve, G., Xu, Q., Kahn, J., Likhomanenko, T., Grave, E., Pratap, V., Sriram, A., Liptchinsky, V., and Collobert, R. (2020). End-to-end asr: from supervised to semi-supervised learning with modern architectures. *ICML Workshop SAS*.

Takashima, R., Li, S., and Kawai, H. (2018). An investigation of a knowledge distillation method for ctc acoustic models. In *Proc.ICASSP*, Alberta.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Proc.NIPS*, Long Beach.

Wang, Y., Xu, C., Xu, C., and Tao, D. (2018). Packing convolutional neural networks in the frequency domain. *Proc.TPAMI*.

Weninger, F., Mana, F., Gemello, R., Andrés-Ferrer, J., and Zhan, P. (2020). Semi-supervised learning with data augmentation for end-to-end asr. *Proc. Interspeech*.

Wu*, Z., Liu*, Z., Lin, J., Lin, Y., and Han, S. (2020). Lite transformer with long-short range attention. In *Proc.ICLR*.

Xiao, A., Fuegen, C., and Mohamed, A. (2021). Contrastive semi-supervised learning for asr. In *Proc.ICASSP*, Toronto.

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

Yu, J., Chiu, C.-C., Li, B., Chang, S.-y., Sainath, T. N., He, Y., Narayanan, A., Han, W., Gulati, A., Wu, Y., and Pang, R. (2021). Fastemit: Low-latency streaming asr with sequence-level emission regularization. In *Proc. ICASSP*, Shanghai.

Yu, X., Liu, T., Wang, X., and Tao, D. (2017). On compressing deep models by low rank and sparse decomposition. In *Proc.CVPR*, Honolulu.

Zhang, Q., Lu, H., Sak, H., Tripathi, A., McDermott, E., Koo, S., and Kumar, S. (2020). Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss. In *Proc.ICASSP*, Barcelona.

Appendix A

Appendix

A.1 Hyperparameters for Teacher W2v2 Model Training

For fine-tuning with clean-100h Librispeech data, we used Adam optimizer with a tri-state learning rate scheduler and a total number of 80000 updates with a batch size of 16. The initial learning rate is 5×10^{-7}, it is then linearly increased to 5×10^{-5} with a 8000 warm-up updates. The learning rate then remains constant for 32000 steps and starts to decrease linearly until it reaches 5×10^{-7} at 80000 steps. During training, we froze the convolutional subsampling module as suggested in Baevski et al. (2020). A channel mask probability of 0.5 was adopted. We did not freeze the transformer at the beginning as this gives us the best results. In the end, the model with the lowest WER on dev-other subset is chosen.

A.2 Hyperparameters for Student Conformer Training

In student conformer training experiments, we adopted noam optimizer from Vaswani et al. (2017). For non-finetuning experiments, we set the base learning rate to 2.5 and used a total of 25000 warm-up steps. For fine-tuning experiments, the base learning rate was set to 0.1 and the warm-up step is set to 10000 to avoid model divergence. All experiments share the same batch size, which is set to 72. For 100h data experiments, we terminated training after 220k updates. For 460h semi-supervised experiments, training was terminated after 350k updates.