Effective Attention-Based Sequence-to-Sequence Modelling for Automatic Speech Recognition

Shucong Zhang

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2021

Abstract

With sufficient training data, attentional encoder-decoder models have given outstanding ASR results. In such models, the encoder encodes the input sequence into a sequence of hidden representations. The attention mechanism generates a soft alignment between the encoder hidden states and the decoder hidden states. The decoder produces the current output by considering the alignment and the previous outputs.

However, attentional encoder-decoder models are originally designed for machine translation tasks, where the input and output sequences are relatively short and the alignments between them are flexible. For ASR tasks, the input sequences are notably long. Further, acoustic frames (or their hidden representations) typically can be aligned with output units in a left-to-right order, and compared to the length of the entire utterance, the duration of each output unit is usually small. Conventional encoder-decoder models have difficulties in modelling long sequences, and the attention mechanism does not guarantee the monotonic left-to-right alignments.

In this thesis, we study attention-based sequence-to-sequence ASR models and address the aforementioned issues. We investigate recurrent neural network (RNN) encoder-decoder models and self-attention encoder-decoder models. For RNN encoder-decoder models, we develop a dynamic subsampling RNN (dsRNN) encoder to shorten the lengths of the input sequences. The dsRNN learns to skip redundant frames. Furthermore, the skip ratio may vary at different stages of training, thus allowing the encoder to learn the most relevant information for each epoch. Thus, the dsRNN alleviates the difficulties of encoding long sequences. We also propose a fully trainable windowed attention mechanism, in which both the window shift and window length are learned by the model. Our windowed method forces the attention mechanism to attend inputs within small sliding windows in a strict left-to-right order. The proposed dsRNN and windowed attention give significant performance gains over traditional encoder-decoder ASR models.

We next study self-attention encoder-decoder models. For RNN encoder-decoder models, we have shown that restricting the attention within small windows is beneficial. However, self-attention encodes input sequences by comparing each element of the sequence with all other elements of the sequence. Therefore, we investigate if the global view of self-attention is necessary for ASR. We note that the range of the learned context increases from the lower to the upper self-attention layers, and suggest that the upper encoder layers may have seen sufficient contextual information without the need for self-attention. This would imply that the upper self-attention layers can be replaced with feed-forward layers (we can view the feed-forward layers as strict local left-to-right self-attention). In practice, we observe replacing upper encoder self-attention layers with feed forward layers does not impact the performance. We also observe that there are individual attention heads that only attend local information, and thus the self-attention mechanism is redundant for these attention heads. Based on these observations, we propose randomly removing attention heads during training but keep all heads at testing. The proposed method achieves state-of-the-art ASR results on benchmark datasets of different ASR scenarios.

Finally, we investigate top-down level-wise training of sequence-to-sequence ASR models. We find that when training sequence-to-sequence ASR models on noisy data, the use of upper layers trained on clean data forces the lower layers to learn noise-invariant features, since the features which fit the clean-trained upper layers are more general. We further show that within the same dataset, conventional joint training makes the upper layers quickly overfit. Therefore, we propose to freeze the upper layers and retrain the lower layers. The proposed method is a general training strategy; we use it not only to train ASR models but also to train other neural networks in other domains. The proposed training method yields consistent performance gains across different tasks (e.g., language modelling, image classification).

In summary, we propose methods which enable attention-based sequence-to-sequence ASR systems to better model sequential data, and demonstrate the benefits of training neural networks in a top-down cascade manner.

Lay Summary

Speech recognition refer to the task of transcribe audio to text. It is relatively easy for humans to distinguish relevant acoustic information for speech recognition from non-useful information. For example, background noise (if the amplitude is not large) and pauses between words usually do not affect humans' speech recognition performance. Furthermore, people typically process speech following the time order in real time. People usually do not depend on what is to be said to recognise what has been said.

However, sequence-to-sequence automatic speech recognition (ASR) systems (systems which transcribe audio to text directly) are generally vulnerable to noise or other redundant input information. Also, offline sequence-to-sequence ASR systems typically begin to generate outputs after they have processed the entire acoustic frame input sequence which is corresponding to an utterance. Therefore, it is worth exploring if using the future frames to generate the current output (e.g., a character or a word) is beneficial. In this thesis, we develop methods which enable the sequence-to-sequence ASR models to more effectively capture useful information of the inputs and to better use the contextual information of the inputs to generate the outputs.

In this thesis, we also explore the training of neural networks. Neural networks have achieved impressive performance in many tasks (e.g., speech recognition, image classification and machine translation). People usually have intuitions on if objects are similar to each other. For example, people generally think apples and oranges are similar in some degree, while these two kinds of fruit are not similar to vehicles. In this thesis, we propose methods to make neural networks firstly have such "intuitions" (in simple terms, a sense of relationships between objects). We then use such intuitions to further train the neural networks. Our proposed training methods have obtained consistent performance gains on a variety of tasks.

Acknowledgements

I would like to give great thanks to my supervisors, Professor Steve Renals, Dr Peter Bell, and Dr Erfan Loweimi. Steve is very kind and patient. I still remember during my first year, I struggled with problems that may seem to be relatively straightforward for a senior researcher. Steve gave me sufficient time to address these problems, which enabled me to be more sophisticated in doing research and have good outcomes in my later years of study. I enjoyed the weekly meetings and all the discussions with my supervisors, where I learned a lot in both the aspect of practical work and the aspect of novel ideas. Finally, my supervisors did not only support me in my research but also helped me a lot when I had difficulties in my personal life. I would never achieve what I have achieved without their guidance and support.

Next, I would like to thank my committee members Dr Shinji Watanabe and Dr Hao Tang. Their detailed questions enabled me to understand my work more profoundly. They also inspired me to think about my research in a broader context. I also would like to thank my committee members for all the encouragement they have given me.

I would like to thank Toshiba Cambridge Research Lab for the financial support. I would like to thank Dr Rama Doddipatla, Dr Cong-Thanh Do, and Dr Catalin Zorila for their guidance and support during the time when I did the internship at Toshiba.

I would like to thank my MSc advisor Dr Shay Cohen, who has continued to give me great advice in my personal life during my PhD study.

I am grateful to the people in the Centre for Speech Technology Research (CSTR). I especially would like to thank Dr Ondrej Klejch, Dr Joachim Fainberg, and Dr Joanna Rownicka for their help in my early study.

I would like to thank my friends in the UK for their supports. I especially would like to thank Chunchuan Lyu, Hao Zheng, Jisi Zhang, Yanzhi Chen, Yumo Xu, and Zhunxuan Wang for their help and supports.

Finally, I would like to thank my parents, my wife and all my family members. Without their supports, I cannot be who I am now.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shucong Zhang*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Statistical sequence-to-sequence models directly map an input sequence to an output sequence. In this thesis, we use the term "sequence-to-sequence" to denote fully differentiable models. That is, we say a model is "sequence-to-sequence" if every component of it can be jointly optimised with gradient descent rather than be trained through a pipeline.

Automatic speech recognition (ASR) refers to the task of translating a sequence of acoustic frames into a sequence of words, and thus sequence-to-sequence models are suitable for solving these tasks. Given a sequence of acoustic frames $\mathbf{X}$, sequence-to-sequence models estimate the probability $P(\mathbf{Y}|\mathbf{X})$, where $\mathbf{Y}$ is a sequence of words or sub-word units.

For ASR tasks, the length of the input sequence and the output sequence generally differ from each other significantly. For an utterance, the input sequence can have hundreds to thousands of frames, while the output sequence usually contains a few words (or the corresponding sub-word units). Also, the alignment between the input sequence and the output sequence is unknown. For sequence-to-sequence models, there are two common methods to address these issues. In the first method, the model assumes monotonic alignments and has an output distribution for each input frame. During training, for each valid alignment between the inputs and the outputs, each input frame is assigned an output label. Since each output unit usually spans for a few frames and the input sequence is longer than the output sequence, consecutive input frames are allowed to be assigned a same output label. Since the alignment between the input frames and output units is unknown, the model considers all valid alignments.

Connectionist Temporal Classification (CTC) (Graves et al., 2006), RNN-Transducers (Graves, 2012) and End-to-End Lattice-free Maximum Mutual Information (Hadian et al., 2018) employ the aforementioned method. HMM based acoustic models in hybrid ASR systems are also sequence-to-sequence models. However, hybrid ASR systems, which contain an acoustic model, a lexicon and a language model, are not fully differentiable. Therefore, hybrid ASR systems are beyond the scope of this thesis.

The second approach does not assume monotonic alignments and does not assign an output for each input. Rather, by using an encoder-decoder strategy, the outputs can be generated at a different rate. The conventional building blocks of encoder-decoder models are recurrent neural networks (RNNs). Sutskever et al. (2014) proposed to use an RNN encoder to map the input sequence into a sequence of hidden states, and then use an RNN decoder to decode the outputs from the last hidden state of the encoder hidden state sequence. Therefore, unlike the first approach, the encoder-decoder approach does not estimate an output probability for each input frame. It worth noting that conventionally "sequence-to-sequence" models refer to the model proposed by Sutskever et al. (2014) and its variants. However, in the scope of this thesis, we use this term based our own definition.

For the model proposed by Sutskever et al. (2014), the last encoder hidden state is a fixed dimension vector, and thus it may not have the representation power to include all relevant information of the input sequence, especially when the input sequence is long. In addition, the RNN encoder tends to "forget" the earlier frames when it processes the the later frames. Attention mechanisms (Bahdanau et al., 2015; Luong et al., 2015) alleviate these issues. For attention-based encoder-decoder models, at each decoding time step, the attention mechanism uses the current decoder hidden state and the encoder hidden states to produce weights. Then, a context vector is generated as the weighted sum of encoder hidden states. Both the context vector and the decoder hidden state are used to generate the current output distribution. Since the attention mechanism looks all the encoder hidden states, the aforementioned issues are alleviated. In addition, the attention weights can be viewed as a soft alignment between the input and output units. Therefore, the attention mechanism also learns the alignment between the inputs and the outputs. Also, with the attention mechanism, the decoder does not need to be initialised with the last encoder state.

Recently, a self-attention based encoder-decoder model (i.e., Transformer) is proposed (Vaswani et al., 2017). The self-attention encodes sequences by comparing each element of the sequence with all the elements of the sequence, and thus it has a global view of the sequence and can capture temporal relationships within any range.

Although attentional encoder-decoder models have given impressive results, they have characteristics which may not be suitable for ASR tasks. In this thesis, we aim to make attentional encoder-decoder models more suitable for ASR tasks. We first study the attentional encoder-decoder models thoroughly and pinpoint their weaknesses for ASR task. We then present our methods which make attentional encoder-decoder models more suitable for ASR tasks.

We particularly focus on improving the encoder, the attention mechanism and the training of the attentional encoder-decoder models. For the RNN encoder, it is challenging to encode speech sequences which are usually noticeably long (Lu et al., 2015; Chan et al., 2016). The attention mechanism in RNN encoder-decoder models is too flexible to model the underlying alignments in ASR tasks. For ASR, each output unit typically only relates to a small window of consecutive input frames, and the alignments between the output and the input usually follow a monotonic left-to-right order. The conventional attention mechanism (Bahdanau et al., 2015; Luong et al., 2015) does not guarantee such alignments. For self-attention based ASR models, since acoustic events which relate to output units usually happen within relatively small time spans, the global view of the sequence may not be useful.

To make attentional encoder-decoder models more suitable for ASR tasks, we present methods which make the RNN encoder better able to model long input sequences, and methods which enable the attention mechanism in RNN encoder-decoder model more natural alignments. For Transformer-based ASR models, we investigate the usefulness of the self-attention mechanism, and aim to improve self-attention based models based on our investigation.

During our investigation of the training of sequence-to-sequence models, we generalise our study to the training of deep neural networks (DNNs) in general. For DNNs, the lower layers can be viewed as a feature extractor, which learns to extract useful pattern from the data. With more training data, the feature extractor becomes better able to extract useful patterns. Based on this, previous works have built DNNs in a bottom-up manner through transfer learning or self-supervised pre-training (Bengio, 2012; Swietojanski et al., 2012; Ghoshal et al., 2013; Devlin et al., 2018; Schneider et al., 2019). In transfer learning, a base model is firstly trained on well-resource data; then, the trained feature extractor is used to extract features on low-resource data. In self-supervised pre-training, large amount of unlabeled training data is used to make the feature extractor learn underlying patterns.

However, to the best of our knowledge, there are few works on building DNNs in a top-down manner. While the lower layers of DNNs can be viewed as a feature extractor, the upper layers of DNNs can be viewed as a classifier. Since there are general feature extractors which are able to extract relevant features among different datasets, we investigate if there are “general classifiers” which give suitable classification boundaries across different tasks. We also aim to develop methods of constructing general classifiers, along with algorithms which utilise general classifiers to train DNNs.

1.2 Contributions

In this thesis, we make attentional encoder-decoder models more effectively model the input sequences and the alignments between the outputs and inputs. For RNN encoder-decoder models, our contributions are the following:

- **A dynamical subsampling RNN**. The proposed dynamical subsampling RNN (dsRNN) learns to skip frames which are not useful for the training. By dropping redundant frames, dsRNN shortens the length of the input sequences, and dsRNN encoder can more effectively model the input sequence.
- **A fully-trainable windowed attention mechanism**. The proposed windowed attention mechanism restricts the attention mechanism to only attend frames in a sliding window following a left-to-right order. All the window parameters (i.e., window shift and window length) are trainable. The proposed method helps the attention mechanism to generate more natural alignments between the input and the output. Also, since the learned window length is small, at each decoding time step, the time complexity of attention mechanism is reduced from $O(n)$ to $O(1)$, where n denotes the length of the input sequence.

For self-attention based encoder-decoders (Transformer models), we present the following contributions:

- **Experiments which show that self-attention is not useful for upper encoder layers**. To investigate if self-attention is useful, we propose to replace the self-attention layers in Transformer encoders with feed-forward layers. When the upper self-attention layers are replaced with feed-forward layers, we observe no performance drop, but in fact minor accuracy gains. We also develop a novel metric of the diagonality of attention matrices, and find the averaged diagonality of each encoder layer increases from the lower layers to the upper layers, indicating the self-attention mechanism is essential for the lower layers but redundant for the upper layers.

- **Experiments which show there are redundant individual attention heads**. We further analyse if the global view of the sequence is necessary for each individual attention head. We find in trained Transformers, there are individual attention heads which almost always generate high diagonality attention matrices, which indicates the self-attention mechanism in these attention heads only focus on local information.
- **A training method which randomly drops attention heads during training**. Since the search for the redundant attention heads is time-consuming, we propose to randomly drop attention heads during training and keep all the attention heads at testing. As a result, the trained model is an ensemble of structures with reduced numbers of attention heads. However, at testing, the model has the full representation power of all the attention heads.

For the training of DNNs, our contributions is as follows:

- **Demonstration of the existence of general classifiers**. We view the lower layers of DNNs as a feature extractor and the upper layers of DNNs as a classifier. We show that in analogy to the concept of the general feature extractor, there are also general classifiers. We conceptually show that general classifiers should properly reflect the relationships between classes. We also experimentally show that the classifiers become more general if we increase the amount of training data.
- **A top-down transfer learning algorithm**. We develop a top-down transfer learning algorithm, which trains the classifier on clean dataset, and then trains the feature extractor on the noisy dataset with the trained clean classifier. We find the clean classifier helps the feature extractor to learn noise-invariant features.
- **A top-down layer-wise training algorithm**. We present a training algorithm which trains DNNs from upper layers to lower layers in a cascade. We observe that in conventional training where all the layers are trained jointly, the upper layers tend to overfit. We propose to freeze the upper layers to prevent this behaviour, and retrain the lower layers to make the lower layers sufficiently trained.

1.3 Thesis Outline

The structure of the remainder of the thesis are the followings:

- In Chapter 2, we describe RNNs and self-attention networks, which are the basic building blocks of sequence-to-sequence models. We then present the architectures of attentional encoder-decoder models. We also introduce the ASR datasets on which we conduct experiments.
- In Chapter 3, we present the dynamical subsampling RNN. We show the experimental results on the TIMIT corpus. We also analyse the skip ratios of dsRNN encoders, and compare dsRNN with static subsampling RNN as well as random subsampling RNN.
- In Chapter 4, we present our fully-trainable windowed attention mechanism. We show experimental results on TIMIT and WSJ datasets. On TIMIT, we also give a detailed analysis of our method. In terms of accuracy, we compare our windowed attention mechanism with conventional attention mechanisms and CTC-attention.
- In Chapter 5, we investigate the usefulness of self-attention for Transformer based ASR models. We show empirical evidence that for upper encoder layers, the self-attention mechanism is redundant. On WSJ and Switchboard dataset, we present experiments of replacing upper encoder self-attention layers with feed-forward layers. We also develop a metric of the diagonality of attention matrices. Based on this metric, we analyse the averaged diagonality of each self-attention layers.
- In Chapter 6, we present a training algorithm which randomly removes attention heads at training. We firstly analyse the usefulness of individual attention heads. Next, we present the proposed training algorithm. We test the proposed method on WSJ, Switchboard, AISHELL and AMI datasets. We also gave significance tests upon the improvements, and analyse the behaviour of attention heads that are trained through the proposed method.
- In Chapter 7, we describe our top-down level-wise training methods. We firstly introduce the concept of general classifier. Next, we show top-down training methods across different datasets (i.e., transfer learning) and within the same

dataset (i.e., layer-wise training). We demonstrate the effectiveness of the top-down transfer learning method by experiments of transferring classifiers from WSJ to CHiME-4, and also the benefits of top-down layer-wise training through ASR experiments on WSJ, CHiME-4, Switchboard, image classification experiments on CIFAR-10 and language modelling experiments on WikiText-2.

- In Chapter 8, we conclude the thesis by summarising our work and discussing potential future work.

1.4 List of Publications

The thesis is based on the following published works:

1. Shucong Zhang, Erfan Loweimi, Peter Bell, and Steve Renals. "Stochastic Attention Head Removal: A Simple and Effective Method for Improving Automatic Speech Recognition with Transformers." INTERSPEECH 2021.

2. Shucong Zhang, Cong-Thanh Do, Rama Doddipatla, Erfan Loweimi, Peter Bell, and Steve Renals. "Train Your Classifier First: Cascade Neural Networks Training from Upper Layers to Lower Layers." ICASSP 2021.

3. Shucong Zhang, Erfan Loweimi, Peter Bell, and Steve Renals. "On the Usefulness of Self-Attention for Automatic Speech Recognition with Transformers" SLT 2021.

4. Shucong Zhang, Cong-Thanh Do, Rama Doddipatla, and Steve Renals. "Learning Noise Invariant Features Through Transfer Learning for Robust End-to-End Speech Recognition." ICASSP 2020.

5. Shucong Zhang, Erfan Loweimi, Yumo Xu, Peter Bell, and Steve Renals. "Trainable Dynamic Subsampling for End-to-End Speech Recognition." INTERSPEECH 2019.

6. Shucong Zhang, Erfan Loweimi, Peter Bell, and Steve Renals. "Windowed attention mechanisms for speech recognition." ICASSP 2019.

Chapter 2

Background

In this chapter, we introduce CTC and attention-based encoder-decoder ASR models, which are the models we will study thoroughly in this these.

We firstly introduce recurrent neural networks (Elman, 1990) and self-attention networks (Vaswani et al., 2017), which can be used as the basic building blocks of sequence-to-sequence ASR models. We then discuss CTC. Finally, we present attention-based sequence-to-sequence ASR models, which include RNN encoder-decoder (Chan et al., 2016), CTC-attention (Kim et al., 2017), Transformer (Vaswani et al., 2017) and Conformer (Gulati et al., 2020).

2.1 Recurrent Neural Network

We firstly discuss the building blocks of sequence-to-sequence models. Recurrent neural networks (RNNs) are powerful in modelling sequential data with variable length. An RNN maps a sequence of input vectors $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$ into a sequence of hidden states, $(\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T)$. The RNN processes the input data in a sequential order. For each time step t, it takes the current element $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ as its inputs. Thus, an RNN can be described as:

$$\mathbf{h}_t = f(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{2.1}$$

where $\mathbf{x}_t \in \mathbb{R}^d, \mathbf{h}_{t-1} \in \mathbb{R}^n, \mathbf{V} \in \mathbb{R}^{n\times d}, \mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$. $\mathbf{V}$, $\mathbf{U}$ and $\mathbf{b}$ are trainable. Sigmoid or tanh are usually used as the activation function f. For the first time step, the initial hidden state $\mathbf{h}_0$ is often set as a zero vector.

Although RNNs are designed to model sequential data with variable lengths, due to the vanishing gradient problem (Hochreiter, 1991) it is hard for RNNs to learn long-range dependencies (Bengio et al., 1994). We describe the vanishing gradient problem here. RNNs are usually trained through gradient descent:

$$\theta = \theta - \eta \nabla \theta \tag{2.2}$$

where θ denotes the trainable parameters of the RNN and η is the learning rate.

Consider a time step t. Let $\mathbf{u} = (\mathbf{U}_{11}, \mathbf{U}_{12}, \cdots, \mathbf{U}_{1n}, \mathbf{U}_{21}, \mathbf{U}_{22}, \cdots, \mathbf{U}_{2n}, \cdots, \mathbf{U}_{nn})$ and $\mathbf{a}_t = \mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}$. Then

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{u}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{a}_t}\frac{\partial \mathbf{a}_t}{\partial \mathbf{u}} \tag{2.3}$$

$$\frac{\partial \mathbf{a}_t}{\partial \mathbf{u}} = \mathbf{h}_{t-1} + \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{u}} \tag{2.4}$$

Thus, $\frac{\partial \mathbf{h}_t}{\partial \mathbf{u}}$ can be expressed as:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{u}} = \sum_{i=1}^{t-1} \mathbf{h}_i \prod_{j=i+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{a}_j} \tag{2.5}$$

where $\mathbf{h}_0$ is set as a zero vector. For a time step k, it can be shown that with sigmoid or tanh activation functions whose derivative is bounded, the contribution of $\mathbf{h}_k \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{a}_j}$ to $\frac{\partial \mathbf{h}_t}{\partial \mathbf{u}}$ goes to 0 exponentially fast with $t-k$ (Pascanu et al., 2013), and this phenomenon is referred as vanishing gradient. Due to the vanishing gradient problem, when the distance between two events are significant large, it could be challenging for RNNs to learn long-range dependencies.

To alleviate the vanishing gradient problem, Long Short-Term Memory (LSTM) is proposed (Hochreiter and Schmidhuber, 1997). LSTM also processes the input sequence $(\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_T)$ in a sequential order. However, at each time step, LSTM uses three trainable gates to control the information flow. The three gates contain an input gate, a forget gate, and an output gate. For each time step t, a non-linear function (usually sigmoid function) takes the current frame $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ to generate a candidate vector $\tilde{\mathbf{c}}_t$ which may be used to update the memory of LSTM. The input pair $(\mathbf{x}_t, \mathbf{h}_{t-1})$ is also fed to all the gates. The input gate decides how much information from the candidate vector should be used to update the memory. The forget gate controls what information should be retained in the memory. Finally, the output gate manages how much information from the memory should be used to produce the

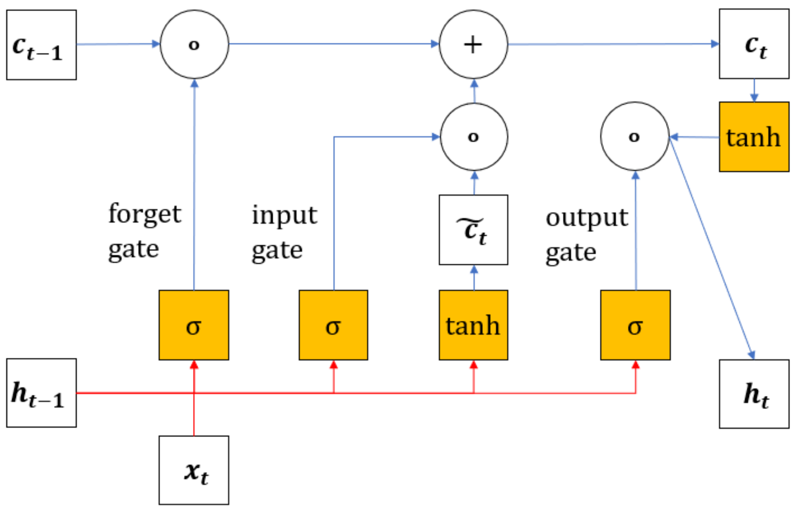


Figure 2.1: An illustration of an LSTM unit. Each gate controls how much information should be passed. The white boxes denote vectors. The yellow boxes are element-wise functions. The red lines indicate affine transformation while the blue lines mean copy of the value.

output vector. Thus, at time step t, an LSTM unit can be described as:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{2.6}$$

$$\mathbf{i}_t = \text{sigmoid}(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{2.7}$$

$$\mathbf{f}_t = \text{sigmoid}(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2.8}$$

$$\mathbf{c}_t = \mathbf{i}_t \circ \tilde{\mathbf{c}}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1} \tag{2.9}$$

$$\mathbf{o}_t = \text{sigmoid}(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{2.10}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \tag{2.11}$$

where $\mathbf{i}_t$ is the output vector of the input gate, $\mathbf{f}_t$ denotes the output vector of the forget gate, $\mathbf{o}_t$ denotes the output vector of the output gate. $\mathbf{W}_c, \mathbf{U}_c, \mathbf{b}_c, \mathbf{W}_i, \mathbf{U}_i, \mathbf{b}_i, \mathbf{W}_f, \mathbf{U}_f, \mathbf{b}_f, \mathbf{W}_o$, $\mathbf{U}_o$ and $\mathbf{b}_o$ are trainable. $\circ$ denotes the element-wise product (Hadamard product). $\mathbf{h}_0$ and $\mathbf{c}_0$ are often initialised as zero vectors. The architecture of an LSTM unit is illustrated in Figure 2.1.

The gates control what information should be kept and what information should be dropped. Thus, they may help the model to learn long-range dependencies. Below is an intuitive explanation. For example, start from time step k and until time step $t-1$, if the input gate always generates zero vectors and the forget gate only gives vectors of ones, then the information in the memory cell from time step k will be preserved to time step t. When executing back-propagation at time step t, the error signal passes

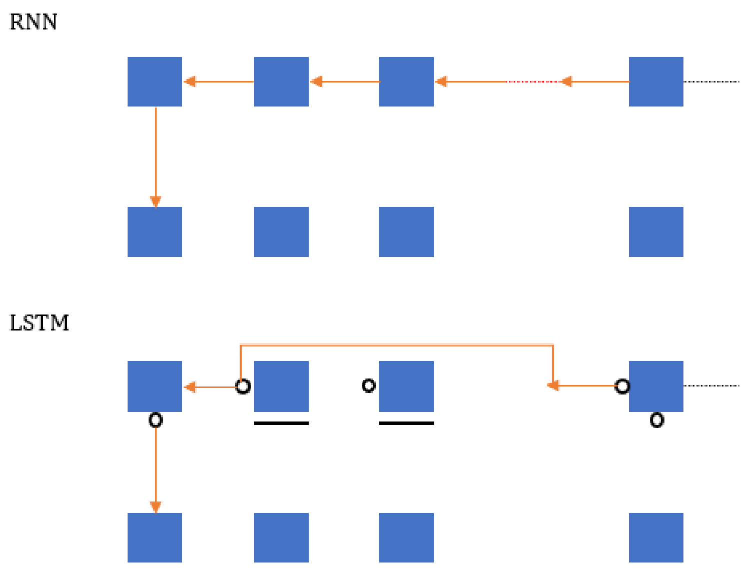
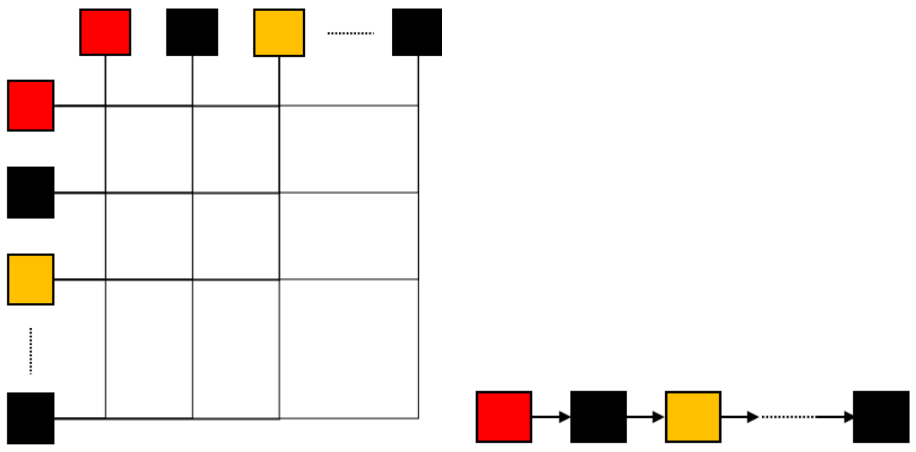


Figure 2.2: A comparison between the gradient flow of the RNN and LSTM. The blue boxes in the lower part of each sub-figure denote the inputs and the blue boxes in the upper part of each sub-figure denote RNN or LSTM units. The red lines denote the gradient flow. In the RNN, the gradient flow passes through each time step. Thus, the vanishing gradient problem may cause the RNN fails to learn the long-range dependency. For LSTM, if all the input gates are closed (denoted by the horizontal bars in the figure) and all the forget gates are open (denoted by the circles in the figure), then the error signal passes to the first time step as if it is directly passed from the last time step to the first time step. Note this is an illustrative comparison. In practice the gates are never fully open or closed.

to time step k as if it is directly passed from the time step t without any intermediate time steps. Thus, if the distance between t and k is large and the inputs at these two time steps has dependency, LSTM may learn this long-range dependency. It worth noting that this example is illustrative, since the gates are never fully open or closed. Figure 2.2 shows an comparison between RNN and LSTM.

2.2 Self-attention Network

RNNs have been widely used for processing sequential data. However, as discussed in the previous section, due to the gradient vanishing problem, the main weakness of RNN is that it is ineffective in capturing long-range dependencies (even with gated structures). Thus, in this section, we introduce self-attention networks (SANs) (Vaswani



(a) Self-attention (b) RNN

Figure 2.3: A comparison between self-attention (a) and RNN (b). Each rectangle denotes an element in the input sequence. For the sub-figure (a) of self-attention, the lines indicate the inputs (after linear transformations) will be compared. For the sub-figure (b) of RNN, the arrow means the hidden representation of the previous input will be used to generated the hidden state of the the current input. The self-attention compares the elements with each other directly to capture the relationships between them. In this figure, the relationship between the second element and the last element can be captured regardless of the distance between them. In contrast, the RNN processes the input elements in sequential order. Therefore, in this example, if the sequence is too long, due to the vanishing gradient problem, the RNN cannot capture the relationship between the second and the last element.

et al., 2017) which can greatly alleviate this problem. SANs have also been widely used to build sequence-to-sequence models.

SANs can model relationships between events without being limited by their distances. Instead of processing inputs in a sequential way, self-attention encodes sequences through the attention mechanism (Bahdanau et al., 2015). For each pair of elements $\mathbf{x}_t$ and $\mathbf{x}_r$ in the input sequence $\mathbf{X}$, self-attention compares $\mathbf{x}_t$ and $\mathbf{x}_r$ (or their hidden representations) directly to learn the relationship between them. Therefore, self-attention has a global view of the input sequence and thus can capture relationships within any range. Figure 2.3 shows a comparison between the self-attention and the RNN.

Formally, the self-attention mechanism takes a 3-tuple of sequences $(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}})$ as its input. When using self-attention to encode a sequence $\mathbf{X}$, we have $\mathbf{X}^{\mathrm{Q}} = \mathbf{X}^{\mathrm{K}} = \mathbf{X}^{\mathrm{V}} = \mathbf{X}$. We discuss the situation where the sequences in the 3-tuple are not equal to each other in Section 2.3.4. The input sequences $(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}})$ are mapped to a Query/Key/Value triplet $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ through linear transformations. After that, a dot-product is applied to compute the similarity scores between each element $\mathbf{q}_t$ in the Query sequence and each element $\mathbf{k}_t$ in the Key sequence. Then, the similarity scores are normalised to generate attention scores. The t^{th} element of the output sequence of the self-attention is a weighted sum of all the elements in the Value sequence $\mathbf{V}$, where the weights are the attention scores between $\mathbf{q}_t$ and all the elements in $\mathbf{K}$. Thus, the self-attention mechanism can be described as:

$$\mathrm{A}(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}) = \mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d^{\mathrm{K}}}})\mathbf{V} \tag{2.12}$$

$$(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = (\mathbf{X}^{\mathrm{Q}}\mathbf{W}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}\mathbf{W}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}\mathbf{W}^{\mathrm{V}}), \tag{2.13}$$

where $\mathbf{X}^{\mathrm{Q}} \in \mathbb{R}^{T \times d^{\mathrm{M}}}$, $\mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}} \in \mathbb{R}^{S \times d^{\mathrm{M}}}$ are inputs and T, S denote the lengths of the input sequences; $\mathbf{W}^{\mathrm{Q}}, \mathbf{W}^{\mathrm{K}} \in \mathbb{R}^{d^{\mathrm{M}} \times d^{\mathrm{K}}}$ and $\mathbf{W}^{\mathrm{V}} \in \mathbb{R}^{d^{\mathrm{M}} \times d^{\mathrm{V}}}$ are trainable matrices.

For the 3-tuple input sequences $(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}})$, multiple independent self-attention mechanisms are usually used to capture different representations of the sequences. The multiple independent self-attention mechanisms are referred to as multi-head attention (MHA). In an MHA that has h heads, each individual attention head A_i can be described as:

$$\mathrm{A}_i(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}) = \mathrm{softmax}(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{d^{\mathrm{K}}}})\mathbf{V}_i \tag{2.14}$$

$$(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = (\mathbf{X}^{\mathrm{Q}}\mathbf{W}_i^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}\mathbf{W}_i^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}\mathbf{W}_i^{\mathrm{V}}) \tag{2.15}$$

where $\mathbf{X}^{\mathrm{Q}} \in \mathbb{R}^{T \times d^{\mathrm{M}}}$, $\mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}} \in \mathbb{R}^{S \times d^{\mathrm{M}}}$ are inputs and T, S denote the lengths of the input sequences; $\mathbf{W}_i^{\mathrm{Q}}, \mathbf{W}_i^{\mathrm{K}} \in \mathbb{R}^{d^{\mathrm{M}} \times d^{\mathrm{K}}}$ and $\mathbf{W}_i^{\mathrm{V}} \in \mathbb{R}^{d^{\mathrm{M}} \times d^{\mathrm{V}}}$ are trainable matrices. Then, the outputs of the h attention heads $(\mathrm{A}_1, \mathrm{A}_2, \cdots, \mathrm{A}_h)$ are combined linearly to generate the output of MHA:

$$\mathrm{MHA}(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}) = (\mathrm{A}_1, \mathrm{A}_2, \cdots, \mathrm{A}_h)\, \mathbf{U}^{\mathrm{H}}, \tag{2.16}$$

where $\mathbf{U}^{\mathrm{H}}$ is a trainable matrix and $\mathbf{U}^{\mathrm{H}} \in \mathbb{R}^{d^{\mathrm{H}} \times d^{\mathrm{M}}}, d^{\mathrm{H}} = h \times d^{\mathrm{V}}$.

The self-attention mechanism only has linear transformations. To introduce non-linearity, the output of the MHA component is fed to a point-wise feed-forward component. To alleviate the gradient vanishing problem and build deep networks, residual connections are used. The feed-forward component can be described as:

$$\mathbf{X}' = \mathbf{X}^{\mathrm{Q}} + \mathrm{MHA}(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}) \tag{2.17}$$

$$\mathbf{X}'' = \mathbf{X}' + \mathrm{ReLU}(\mathbf{X}'\mathbf{W}^{\mathrm{FF}_1} + \mathbf{b}^{\mathrm{FF}_1})\mathbf{W}^{\mathrm{FF}_2} + \mathbf{b}^{\mathrm{FF}_2} \tag{2.18}$$

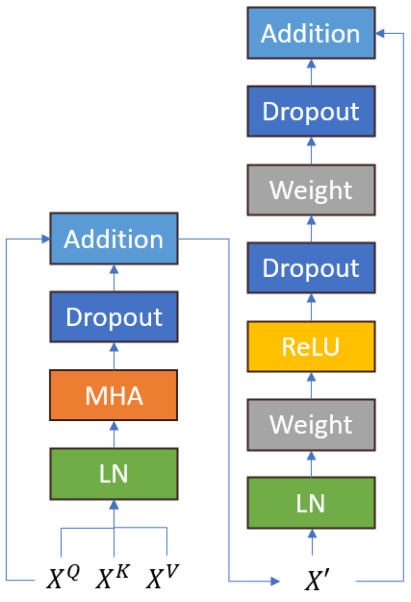


Figure 2.4: Architectures of a self-attention layer. The boxes show the components of a self-attention layer. The arrows indicate the flow of the forward pass. “LN” denotes layer normalisation. “Weight” denotes linear transformation. The position of “LN” follows Wang et al. (2020) and Karita et al. (2019).

where $\mathbf{W}^{\mathrm{FF}_1} \in \mathbb{R}^{d^{\mathrm{M}} \times d^{\mathrm{FF}}}$, $\mathbf{W}^{\mathrm{FF}_2} \in \mathbb{R}^{d^{\mathrm{FF}} \times d^{\mathrm{M}}}$, $\mathbf{b}^{\mathrm{FF}_1} \in \mathbb{R}^{d^{\mathrm{FF}}}$ and $\mathbf{b}^{\mathrm{FF}_2} \in \mathbb{R}^{d^{\mathrm{M}}}$ are trainable matrices and vectors. Dropout (Srivastava et al., 2014) and layer normalisation (Ba et al., 2016) are also used in the self-attention layer. Figure 2.4 shows the architecture of a self-attention layer.

The self-attention mechanism does not consider positional information since each element in the Query sequence is directly compared with each element in the Key sequence. For a given Query sequence, any permutation of the Key sequence (as well as the Value sequence) will lead to the same output sequence of MHA. To address this, positional embedding is used (Gehring et al., 2017; Vaswani et al., 2017). For each input sequence, the positional embedding for the element at position i can be described as:

$$\mathrm{PE}_{(i,2m)} = sin(i/10000^{2m/d^{\mathrm{M}}}) \tag{2.19}$$

$$\mathrm{PE}_{(i,2m)} = cos(i/10000^{2m/d^{\mathrm{M}}}) \tag{2.20}$$

where m denotes the dimension of the elements in the input sequences. The positional embedding is either added (Vaswani et al., 2017) or concatenated (Sperber et al., 2018) to the input sequences $(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}})$. As a result, each element in the input sequences contains the positional information.

2.3 Sequence-to-Sequence ASR Models

We have discussed RNNs and self-attention networks, which can be used as the basic building blocks for sequence-to-sequence ASR models. In this section, we introduce the sequence-to-sequence ASR models which we study in this thesis (CTC and attention-based models). It is worth noting that for the term “sequence-tosequence” we follow our definition in Chapter 1 rather than following the conventional definition as in Sutskever et al. (2014). Other sequence-to-sequence ASR models, such as RNN-Transducers (Graves, 2012) and End-to-End Lattice-free MMI (Hadian et al., 2018), are beyond the scope of this thesis.

In general, sequence-to-sequence models directly map the input sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$ to the output sequence $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M)$ through a single neural network (although for ASR having an additional language model component is usually helpful). The training objective is to maximise the conditional probability $P(\mathbf{Y}|\mathbf{X})$. For sequence-to-sequence ASR, the input sequences consist of acoustic frames and the output sequences consist of characters or other sub-word units. Thus, the input sequences are usually noticeably longer than the output sequences. Further, the alignments between the inputs and the outputs are unknown. Therefore, modelling the alignments between variable length input and output sequences is an essential task for sequence-to-sequence ASR models.

2.3.1 CTC

CTC uses RNNs or SANs to map the input sequence into a sequence of hidden states. Then, each hidden state $\mathbf{h}_t$ is fed into the softmax function to generate an output probability. However, input sequences made of acoustic frames are longer than the output sequences containing sub-word units. To address this, CTC allows repeated output units and blank symbols ($-$). The repeated characters between blank symbols are merged to generate the final output sequence $\mathbf{Y} = (\mathbf{y}_1, \cdots, \mathbf{y}_M)$. For example, for the input sequence $\mathbf{X} = (\mathbf{x}_1, \cdots \mathbf{x}_5)$ and the output sequence $\mathbf{Y} = (c, a, t)$, $\mathbf{A} = (c, c, -, a, t)$ and $\mathbf{A} = (c, a, a, a, t)$ are valid CTC outputs while $\mathbf{A} = (c, -, c, a, t)$ is not a valid CTC output for the word “cat”.

Since the alignment between the inputs and the outputs is unknown, CTC considers all the valid alignments between the inputs and the outputs. During training, for each utterance, the model maximises the probability of the ground truth label sequence, which is the sum of the probabilities of all valid alignments:

$$P(\mathbf{Y}|\mathbf{X}) = \sum_{\mathbf{A} \in S} P(\mathbf{A}|\mathbf{X}), \tag{2.21}$$

where S represents the set of all valid alignments. CTC models assume the outputs are conditionally independent. With this assumption, the probability of each valid sequence is given by:

$$P(\mathbf{A}|\mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{o}_i|\mathbf{X}) \tag{2.22}$$

where $\mathbf{o}_i$ denotes the output at each time step. The sum of probabilities over all the valid alignments can be computed efficiently through a forward-backward algorithm. The decoding can be performed in a greedy way or using beam search. The main weakness of CTC is the outputs are conditionally independent.

2.3.2 RNN Attentional Encoder-Decoder

In this section, we introduce encoder-decoder model, which do not have the conditionally independent problem. The encoder-decoder model can be built through RNNs or SANs. Although the overall structures of RNN based and SAN based encoder-decoder models are similar, they differ in several aspects. We discuss the RNN based attentional encoder-decoder in this section and present SAN based encoder-decoder (i.e.Transformer Models) in Section 2.3.4 and Section 2.3.5.

The RNN attentional encoder-decoder uses an RNN encoder to map the inputs $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ into a sequence of hidden states $\mathbf{H}$. For ASR tasks, the input sequences usually contain hundreds or thousands of frames. Due to the vanishing gradient problem which we have discussed in Section 2.1, it is difficult for RNNs to encode such long sequences. Even with gated structure such as LSTM, vanilla RNN encoders (including LSTM encoders) often give inferior ASR results (Lu et al., 2015; Chan et al., 2016). To alleviate this, subsampling of the input sequence is usually used. One commonly used method is to read every second element in the input sequence. Thus, the hidden state sequence $\mathbf{H} = (\mathbf{x}_1, \cdots, \mathbf{x}_S)$ is usually shorter than the input sequence $\mathbf{X}$. In the original encoder-decoder sequence-to-sequence model Sutskever et al. (2014), since the last hidden state $\mathbf{h}_S$ of $\mathbf{H}$ summaries the information of the input sequence, it is used to initialise the RNN decoder which generates the output sequence. However, a single vector $\mathbf{h}_S$ usually does not have the capacity to include all the essential information of such long sequences. To address this, attention mechanisms are proposed

(Bahdanau et al., 2015; Luong et al., 2015). Through the attention mechanism, at each decoding time step, the model is allowed to revisit all the encoder hidden states and learns soft alignments.

At each decoding time step m, the decoder RNN uses the previous output $\mathbf{y}_{m-1}$, the previous decoder RNN hidden state $\mathbf{q}_{m-1}$ and the previous context vector $\mathbf{c}_{m-1}$ to generate the current decoder hidden state $\mathbf{q}_m$. The context vector summarises the alignment information and it is generated through the attention mechanism. At decoding time m, the attention mechanism uses the current decoder hidden state $\mathbf{q}_m$ and the sequence of encoder hidden states $\mathbf{H}$ to compute an alignment vector α_m. The alignment vector represents a soft alignment between the current output and the inputs. Using the elements in the alignment vector α_m as weights, the context vector $\mathbf{c}_m$ is computed as a weighted sum of the encoder hidden states. Finally, the decoder uses the current context vector $\mathbf{c}_m$ and the current decoder hidden state $\mathbf{q}_m$ to estimate the current label distribution $P(\mathbf{y}_m|\mathbf{X}, \mathbf{Y}_{1:m-1})$. Following the previous work (Chan et al., 2016), the decoder can be described as follows:

$$\mathbf{q}_m = \mathrm{RNN}(\mathbf{q}_{m-1}, [\mathbf{y}_{m-1}, \mathbf{c}_{m-1}]) \tag{2.23}$$

$$\alpha_{ms} = \mathrm{Attention}(\mathbf{q}_m, \mathbf{h}_s) \tag{2.24}$$

$$\mathbf{c}_m = \sum_s \alpha_{ms}\mathbf{h}_s \tag{2.25}$$

$$\mathbf{y}_m \sim \mathrm{LabelDistribution}(\mathbf{c}_m, \mathbf{q}_m) \tag{2.26}$$

There are several methods to compute the attention vector. The following equations describe commonly used approaches:

$$e_{ms} = \psi(\mathbf{q}_m)^T\phi(\mathbf{h}_s) \qquad (\mathrm{dot-product}) \tag{2.27}$$

$$e_{ms} = \mathbf{v}^T \tanh(\mathbf{W}\mathbf{q}_m + \mathbf{V}\mathbf{h}_s + \mathbf{b}) \qquad (\mathrm{Content}) \tag{2.28}$$

$$e_{ms} = \mathbf{v}^T \tanh(\mathbf{W}\mathbf{q}_m + \mathbf{V}\mathbf{h}_s + \mathbf{U}\mathbf{f}_{ms} + \mathbf{b}) \qquad (\mathrm{Location}) \tag{2.29}$$

$$\mathbf{f}_m = F * \alpha_{m-1} \tag{2.30}$$

where ψ and ϕ denotes MLP networks, $\mathbf{v}, \mathbf{W}, \mathbf{V}, \mathbf{U}, F, \mathbf{b}$ are trainable parameters, $*$ denotes convolution. The attention vector is:

$$\alpha_{ms} = \frac{\exp(e_{ms})}{\sum_{k=1}^{S}\exp(e_{mk})} \tag{2.31}$$

We can also use a temperature parameter τ to control how the attention weights are distributed. That is, Equation 2.31 becomes to:

$$\alpha_{ms} = \frac{\exp(\tau^{-1}e_{ms})}{\sum_{k=1}^{S}\exp(\tau^{-1}e_{mk})} \tag{2.32}$$

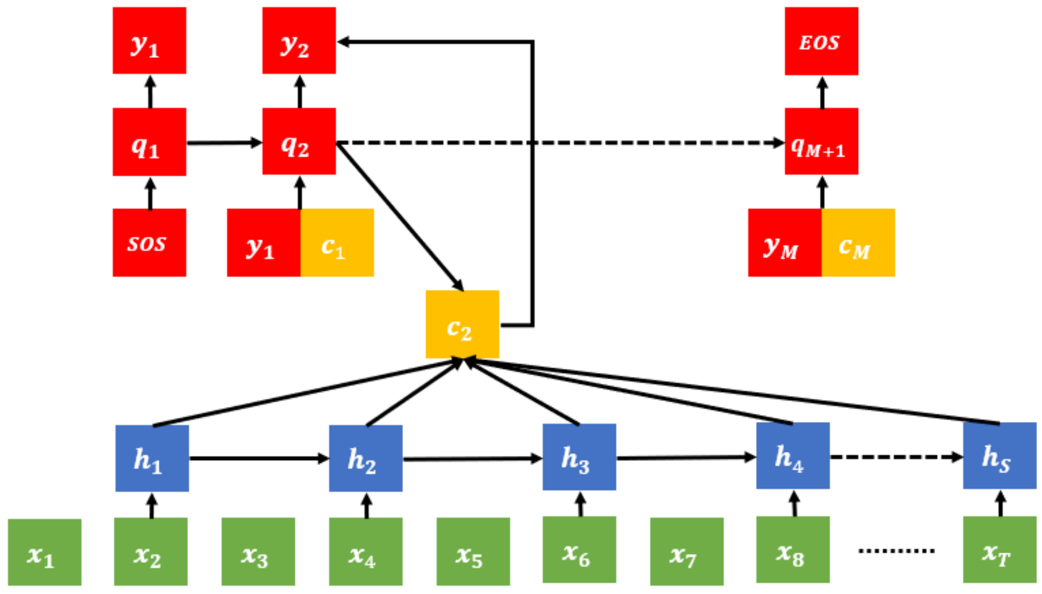


Figure 2.5: A demonstration of the RNN based attentional encoder-decoder. SOS denotes "start of sentence" and EOS denotes "end of sentence". The decoder stops generating outputs after it produced the EOS symbol. Subsampling is used so the encoder only reads the every second input frames.

where τ can be set as a hyper-parameter or learned by the model. When $\tau > 1$, it makes the attention weights distribute more uniformly; when $\tau < 1$, it makes the peak of the attention weight distribution sharper. In this thesis, we do not use the τ parameter (or equivalently, we set it to 1). Figure 2.5 shows an illustration of the RNN based attentional encoder-decoder.

Because of the RNN decoder, RNN encoder-decoder does not have the problem of conditional independent outputs. However, the attention mechanism is too flexible to guarantee the left-to-right alignments between output units and input frames. The conventional attention mechanism is allowed to generate alignments of any kind .

2.3.3 Regularise Attentional Models with CTC

CTC only allows strict left-to-right alignments between the inputs and the outputs. However, the outputs of CTC are conditionally independent. For attentional encoder-decoder models, the current output depends on the previous outputs. Nevertheless, the left-to-right alignments between the inputs and the outputs are not guaranteed.

CTC-attention is proposed to combine CTC with attentional encoder-decoder (Kim et al., 2017). The neural network architecture of CTC-attention and attentional encoder-decoder is the same. However, CTC outputs are generated from the hidden states of the encoder. The training loss L of the CTC-attention is:

$$\mathcal{L} = (1-\lambda)\mathcal{L}^{\mathrm{D}} + \lambda\mathcal{L}^{\mathrm{CTC}}. \tag{2.33}$$

where $\mathcal{L}^{\mathrm{D}}$ denotes the decoder loss, $\mathcal{L}^{\mathrm{CTC}}$ denotes the CTC loss and λ is set as a hyper-parameter. The CTC loss provides additional alignment information to the encoder, and thus implicitly encourages the attention mechanism to generate monotonic alignments between the inputs and the outputs. Furthermore, the outputs of the decoder part do not suffer the problem of conditional independence.

2.3.4 Transformer

Since it is relatively difficult to for RNN to model long-range dependencies and self-attention can capture global information, SANs are used to build sequence-to-sequence models. The self-attention based encoder-decoder is known as Transformer (Vaswani et al., 2017). The encoder is a stack of self-attention layers. Every encoder self-attention layer reads the output of its previous layer (the lowest layer reads the input sequence of the Transformer). For each encoder self-attention layer, the input sequences in the 3-tuple are the same. That is, for each encoder layer j, $\mathbf{X}_j^{\mathrm{Q}} = \mathbf{X}_j^{\mathrm{K}} = \mathbf{X}_j^{\mathrm{V}} = \mathbf{X}_j$. The structure of encoder self-attention layers is the same as the self-attention layer described in Section 2.2. The positional embedding is either added to/concatenated with the inputs of the first self-attention layer or fed to all the self-attention layers.

The decoder also has a stack of self-attention layers. However, in contrast to the encoder self-attention layer which has one MHA component, each decoder self-attention layer has two MHA components. At decoding time step m, for the lowest decoder layer, the first MHA looks at the outputs of the Transformer up to decoding time step $m-1$. When fed into the first decoder layer, the sequence of the outputs begins with a "start of sentence" label, so the outputs are right-shifted by one position. Thus, the current output only depends on the previous outputs. The second MHA takes the outputs of the first MHA as the Query sequence and the outputs of the topmost encoder layer as the Key/Value sequences. The output of the second MHA is fed into a feed-forward component. The subsequent decoder self-attention layers have the same architecture as the first decoder self-attention layer.

Mathematically, for a decoding time step m, the MHA components in the j^{th} self-

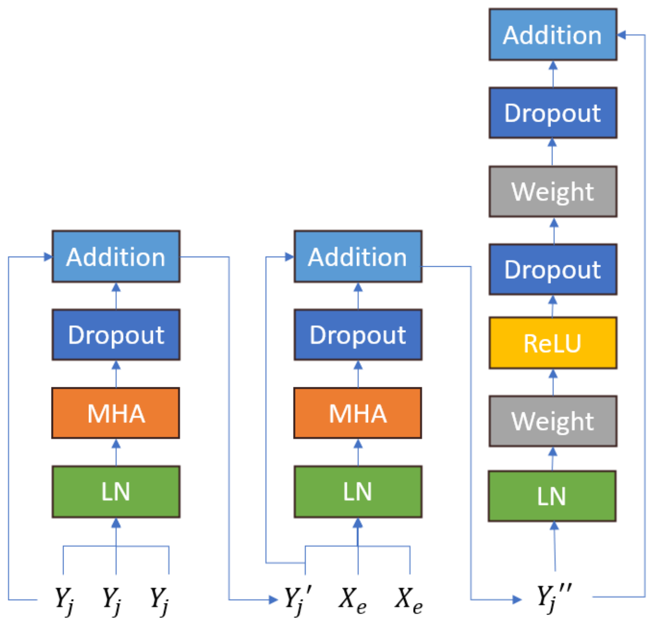


Figure 2.6: Architectures of a self-attention decoder layer. LN denotes layer normalisation. For the second MHA component, $\mathbf{Y}'_j$ is the Query sequence, $\mathbf{X}_e$ is the outputs of the topmost encoder layer and it used as the Key/Value sequence.

attention decoder can be described as:

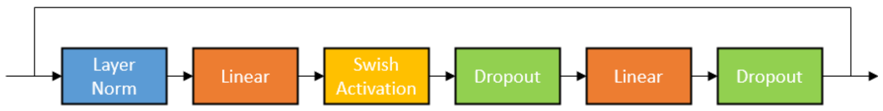
$$\mathbf{Y}'_j = \mathbf{Y}_j + \text{MHA}_j^{\text{dec}}(\mathbf{Y}_j, \mathbf{Y}_j, \mathbf{Y}_j) \tag{2.34}$$

$$\mathbf{Y}''_j = \mathbf{Y}'_j + \text{MHA}_j^{\text{dec-enc}}(\mathbf{Y}'_j, \mathbf{X}_e, \mathbf{X}_e) \tag{2.35}$$

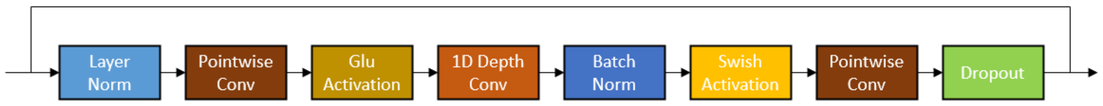
where for the lowest decoder self-attention layer, $\mathbf{Y}_0$ denotes the outputs of the Transformer up to decoding time step $m-1$; for the subsequent decoder self-attention layers, $\mathbf{Y}_j$ denotes the outputs of the previous decoder self-attention layer; and $\mathbf{X}_e$ denotes the outputs (hidden representations) of the topmost encoder layer. The structure of the feed-forward component in the decoder self-attention layer is the same as the structure of the feed-forward component in the encoder self-attention layer (Section 2.2). The positional embedding is either injected into the inputs of the first decoder self-attention layer or fed to all the decoder self-attention layers. Figure 2.6 shows the architecture of a decoder self-attention layer.

2.3.5 Conformer

Since self-attention encodes sequences through attention mechanisms, it can model dependencies within any range. However, previous works have shown that self-attention is less effective in capturing local information (Wu et al., 2020; Zhang et al., 2021).



(a) Feed-forward Component



(b) Convolution Component

Figure 2.7: The feed-forward component (a) and the convolution component (b) of a conformer layer. Glu denotes gated linear units (Dauphin et al., 2017).

As a matter of fact, acoustic events usually happen within relatively short periods and local context information is essential for ASR tasks. Since convolution neural networks (CNNs) are suitable for capturing local context, Gulati et al. (2020) propose Conformer, which augments Transformer encoder self-attention layers with CNNs. A Conformer layer can be described as:

$$\tilde{\mathbf{X}} = \mathbf{X} + \frac{1}{2}\mathrm{FF}(\mathbf{X}) \tag{2.36}$$

$$\mathbf{X}' = \tilde{\mathbf{X}} + \mathrm{MHA}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}, \tilde{\mathbf{X}}) \tag{2.37}$$

$$\mathbf{X}'' = \mathbf{X}' + \mathrm{Conv}(\mathbf{X}') \tag{2.38}$$

$$\mathbf{X}''' = \mathrm{LayerNorm}(\mathbf{X}'' + \frac{1}{2}\mathrm{FF}(\mathbf{X}'')) \tag{2.39}$$

where FF denotes the feed-forward component, Conv denotes the convolution component and LayerNorm denotes layer normalisation. The architecture of the feed-forward component is shown in Figure 2.7 (a) and the architecture of the convolution component is presented in Figure 2.7 (b). The half-step feed-forward networks ($\frac{1}{2}$FF in Equation 2.36 and Equation 2.39) have resulted in better accuracies compared to the full-step feed-forward network (Gulati et al., 2020).

Conformer also uses a different positional embedding method compared to Transformer. Here, we use $\mathbf{X}[i]$ to denote the $i^t h$ element in the sequence $\mathbf{X}$. For an attention head, consider the element $\mathbf{X}^{\mathrm{Q}}[i]$ in $\mathbf{X}^{\mathrm{Q}}$ and the element $\mathbf{X}^{\mathrm{K}}[j]$ in $\mathbf{X}^{\mathrm{K}}$. Suppose the positional embedding is added to the input sequences. Then, the attention score A_{ij}^{abs} between $\mathbf{X}^{\mathrm{Q}}[i]$ and $\mathbf{X}^{\mathrm{K}}[j]$ can be expressed as:

$$\begin{aligned} A_{ij}^{abs} &= (\mathbf{X}^{\mathrm{Q}}[i] + \mathbf{P}_i)\mathbf{W}^{\mathrm{Q}} \cdot (\mathbf{X}^{\mathrm{K}}[j] + \mathbf{P}_j)\mathbf{W}^{\mathrm{K}} \\ &= \mathbf{X}^{\mathrm{Q}}[i]\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{X}^{\mathrm{K}}[j])^T + \mathbf{X}^{\mathrm{Q}}[i]\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{P}_j)^T \\ &+ \mathbf{P}_i\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{X}^{\mathrm{K}}[j])^T + \mathbf{P}_i\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{P}_j)^T \end{aligned} \tag{2.40}$$

where $\mathbf{P}_i$ and $\mathbf{P}_j$ denote the positional embedding. Conformer uses relative positional embedding proposed by Dai et al. (2019). With the relative positional embedding, the attention score A_{ij}^{rel} is:

$$\begin{aligned} A_{ij}^{rel} = &\mathbf{X}^{\mathrm{Q}}[i]\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{X}^{\mathrm{K}}[j])^T + \mathbf{X}^{\mathrm{Q}}[i]\mathbf{W}^{\mathrm{Q}}(\mathbf{W}^{\mathrm{P}})^T(\mathbf{P}_{i-j})^T \\ &+ \mathbf{u}(\mathbf{W}^{\mathrm{K}})^T(\mathbf{X}^{\mathrm{K}}[j])^T + \mathbf{v}(\mathbf{W}^{\mathrm{P}})^T(\mathbf{P}_{i-j})^T \end{aligned} \tag{2.41}$$

where $\mathbf{W}^{\mathrm{P}}$ is a trainable matrix, $\mathbf{u}$ and $\mathbf{v}$ are trainable vectors. There are several differences between Equation 2.40 and Equation 2.41. Firstly, the relative positional embedding only considers the relative positions between $\mathbf{X}^{\mathrm{Q}}[i]$ and $\mathbf{X}^{\mathrm{K}}[j]$. The motivation is when the attention mechanism considers where to attend, only the relative distance between positions matters. Thus, the absolute positional embeddings $\mathbf{P}_i$ and $\mathbf{P}_j$ are replaced by the relative positional embedding $\mathbf{P}_{i-j}$. Secondly, in Equation 2.40, the positional embedding $\mathbf{P}_j$ is multiplied by the matrix $\mathbf{W}^{\mathrm{K}}$. However, $\mathbf{W}^{\mathrm{K}}$ is to transformer the input sequence $\mathbf{X}^{\mathrm{K}}$ to the key sequence $\mathbf{K}$ (Equation 2.13). Therefore, to make the content vector $\mathbf{X}^{\mathrm{K}}$ and the position vector $\mathbf{P}_{i-j}$ have different weight matrices, Equation 2.41 uses a distinct trainable matrix $\mathbf{W}^{\mathrm{P}}$ to multiply with the relative positional embedding $\mathbf{P}_{i-j}$. Lastly, in Equation 2.40, the vector $\mathbf{P}_i\mathbf{W}^{\mathrm{Q}}$ is invariant to the content of the input sequences. Therefore, Equation 2.41 replaces this vector with trainable vectors $\mathbf{u}$ and $\mathbf{v}$.

In the original Conformer paper (Gulati et al., 2020), LSTM was used as the decoder. We use the term "Conformer" to denote the **Conformer Encoder–Transformer Decoder** structure.

2.4 Datasets

In this section, we describe the speech recognition datasets on which we conduct our experiments. Table 2.1 summaries these datasets. Except AISHELL (Bu et al., 2017) which is a Chinese corpus, all other datasets are English datasets.

TIMIT (Garofolo et al., 1993). TIMIT is a read speech dataset. It contains 6300 utterances read by 630 speakers of eight major dialects of American English. Each speaker reads 10 sentences. Phone level transcription is provided. The phones in the transcription are from a 61-phone set. For the evaluation of ASR models, the 61-phone set is usually reduced to a 39-phone set.

Table 2.1: Datasets Description

datasets	language	hours	speech style
TIMIT	EN	5	read (clean)
WSJ	EN	81	read (clean)
AISHELL	ZH	170	read (clean)
CHiME-4	EN	18	read (multi-channel noisy)
SWBD	EN	260	telephone conversation
AMI	EN	100	meeting

Wall Street Journal (WSJ) (Paul and Baker, 1992). WSJ is a corpus which contains read sentences from Wall Street Journal news. For the experiments conducted on WSJ, we use WSJ si284 as the training set. WSJ si284 contains 81 hours of read speech data from 284 speakers.

AISHELL (Bu et al., 2017). AISHELL is a read speech Chinese dataset. It contains 170 hours of Chinese mandarin speech data from 400 speakers of 4 major Chinese accent. The dataset covers topics from 11 different domains.

CHiME-4 (Vincent et al., 2017; Barker et al., 2017). CHiME-4 contains read speech recorded by a tablet that has a 6-channel microphone array. The speech is recorded in 4 noisy environments: on a bus, cafe, pedestrian area, and street junction. There are 4 speakers in total. The data can be divided into two categories: real and simulation. The “real data” contains the utterances which are read in the real noisy environment. The “simulation data” is generated by adding the noise from the environment to the clean read data.

Switchboard (SWBD) (Godfrey et al., 1992). SWBD is a telephone conversation dataset. It contains about 2400 two-side telephone conversations. There are 543 speakers from all areas of the US. In the dataset, no two speakers would have conversations more than once. The dataset covers 70 topics. No one speaker would talk about a topic more than once.

AMI (Renals et al., 2007). AMI consists of meeting recordings. Different recording instruments are used, including close-talk and far-field microphones. The meetings were in three rooms with different acoustic properties. Most speakers are non-native speakers.

Chapter 3

Trainable Dynamic Subsampling Encoder for RNN Attentional Models

3.1 Introduction

The RNN encoder, which learns the hidden representations of the input sequences, is an essential component in attentional encoder-decoder models. However, as discussed in Chapter 2, RNNs are less effective in modelling long sequences. For ASR tasks where the input sequences are often notably long, encoder-decoder models with vanilla RNN/LSTM encoders often yield poor results (Lu et al., 2015; Chan et al., 2016). Chan et al. (2016) demonstrated that with a vanilla multilayer LSTM encoder, the model gives poor ASR results even with an extremely long training time.

Subsampling has been proposed to train neural network acoustic models (Vanhoucke et al., 2013; Miao et al., 2016). To alleviate the training problem of encoder-decoder ASR models, subsampling is also used. Chan et al. (2016) use encoder layers which read the concatenation of every two hidden states of its previous layer, while Lu et al. (2015) and Bahdanau et al. (2016) employ encoder layers which read one of every two hidden states of its previous layer. Thus, in these methods, each encoder layer reduces the the sequence length by a factor of 2. However, these methods subsample statically, and does not consider different acoustic situations. For example, for silence or for a section in which the acoustic features are similar, subsampling rates should be high. When the acoustic features change rapidly, the amount of subsampling should be lower. A fixed, static subsampling method fails to consider these situations, resulting in the retention of redundant information and the loss of relevant information.

To address this, we propose a novel dynamic subsampling RNN (dsRNN) archi-

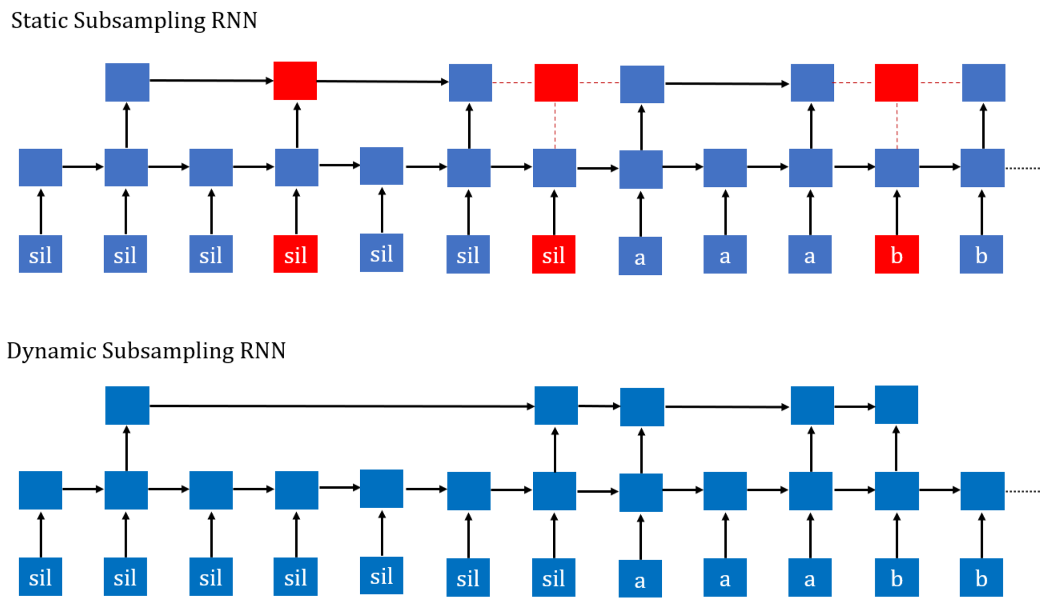

Static Subsampling RNN
sil sil sil sil sil sil sil a a a b b
Dynamic Subsampling RNN
sil sil sil sil sil sil sil a a a b b

Figure 3.1: Subsampling: static (above) vs. dynamic (below). In this example, the static subsampling keeps the unnecessary silence frame. However, it drops the frames at the transitions of acoustic events. In contrast, the dsRNN drops the redundant information while keeping the important frames.

tecture. We use the dsRNN to build the encoder for the encoder-decoder models. This learned subsampling strategy is helpful in two aspects:

- First the network learns *what to skip* – it learns to skip redundant frames but keeps the important frames. Thus, the skip policy forces the RNN encoder only to consider the most relevant inputs during training.

- Second, it learns *how frequently to skip input frames* – in practice this may be adaptively modified during training. At the start of training, we observe the skip rates to be high, enabling the encoder to learn the hidden representations for the most essential frames at the early training stage. As training progresses, the skip rate decays, allowing the encoder to retain more input information.

Because of these two abilities, the dsRNN encoder learns a better representation of the input sequence and obtains a significant error rate reduction compared to the statically subsampled RNN encoder. Figure 3.1 demonstrate a comparison between the static subsampling and the dynamic subsampling. Although our dsRNN encoder is unidirectional, we observe increased accuracy compared to a bidirectional statically subsampled RNN encoder.

3.2 Related Work

For encoder-decoder models for speech recognition, subsampling between stacked encoder RNN layers has been proposed to reduce the length of the input sequence (Lu et al., 2015; Chan et al., 2016). An encoder layer reads either one of every two hidden states of the previous layer (or their concatenation), reducing the input length by a factor of 2. Subsampling is often performed between two or three layers, which leads to a reduction rate of 4 or 8. High reduction rates (e.g. 32) are possible, but it requires careful pre-training (Zeyer et al., 2018).

Although these length reduction methods yield improved results, the subsampling is fixed. As discussed early, it does not consider different acoustic events: when the input acoustic features change rapidly, the skip rate (amount of subsampling) should be low while when the input acoustic features are similar, the skip rate should be high. However, static subsampling fails to recognise these situations. This strategy may retain redundant information, while dropping essential frames.

The skip RNN was proposed to allow an RNN to skip inputs dynamically (Campos et al., 2018). The Skip RNN demonstrated promising results in several sequential modelling tasks. However, the Skip RNN uses only the previous RNN hidden state in order to predict if the current input should be skipped or not. This approach is risky, since relevant inputs may be overlooked while unnecessary inputs may be kept. For speech recognition, a similar Skip RNN model has been proposed, which also skips without considering the current input (Song et al., 2018). Compared with regular RNNs it has no gain in accuracy; neither has it been applied to sequence-to-sequence models.

The Skip RNN and our proposed dsRNN are related to Highway Networks (Srivastava et al., 2015). Highway Networks are feed-forward networks that use a transform gate and a carry gate to decide how much information from the current layer should be copied to the next layer. Similarly, Skip RNN and dsRNN determine if the hidden state of the current time step should be copied to the next time step. However, there are two main differences between Highway Networks and the dsRNN. First, Highway Networks are feed-forward networks which may skip some intermediate layers. The proposed dsRNN is a recurrent network. Although from the viewpoint of unfolding computational graphs, it also omits some intermediate layers, the main characteristic of the dsRNN is the dropping of inputs. Second, Highway Networks use gates to drop intermediate layers in a soft way, while the dsRNN skips inputs in a hard fashion – the inputs are either retained or discarded.

Chung et al. (2017) propose a hierarchical multi-scale RNN which uses a hard gating mechanism to enable the model to learn hierarchical boundaries – the upper layers could learn the event boundaries of the lower layers. In contrast, although the proposed dsRNN could also detect boundaries, the main objective is to drop unessential inputs. It does not learn the hierarchical boundaries through layers.

In the perspective of a single time step, our method has some similarity to dropout (Srivastava et al., 2014). However, dropout randomly drops some weights of the network while our model learns to skip the input. Thus, dropout puts a random mask on the network weights while our model puts a learned mask on the input.

3.3 Dynamic Subsampling RNN

A dsRNN maps a sequence of input frames $(\mathbf{x}_1, \cdots, \mathbf{x}_T)$ to a sequence of hidden representations $(\mathbf{h}_1, \cdots, \mathbf{h}_S)$. At each time step t, for the vanilla RNN, the hidden state $\mathbf{h}_t$ is computed recurrently by the RNN:

$$\mathbf{h}_t = \mathrm{RNN}(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{3.1}$$

Compared to RNN, our dsRNN has an additional update gate. The RNN component of dsRNN can be any variant of RNN, such as LSTM. For simplicity, we use RNN to denote the RNN component of dsRNN, which may be any variant of RNN models. The update gate takes the previous hidden state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$ to output an indicator $u_t \in \{0, 1\}$, where 0 stands for skip $\mathbf{x}_t$ and 1 indicates using $\mathbf{x}_t$ to update the hidden state. The process is described below.

First, the RNN component computes a temporary current hidden state $\tilde{\mathbf{h}}_t$:

$$\tilde{\mathbf{h}}_t = \mathrm{RNN}(\mathbf{x}_t, \mathbf{h}_{t-1}). \tag{3.2}$$

The update gate takes the concatenation of the temporary current RNN hidden state and the previous RNN hidden state as its input and uses a multilayer perceptron (MLP) to compute the increment of the skip probability $\Delta\mathbf{p}_t$:

$$\Delta\mathbf{p}_t = \mathrm{sigmoid}(\mathrm{MLP}([\mathbf{h}_{t-1}, \tilde{\mathbf{h}}_t])) \tag{3.3}$$

When the RNN is composed of multiple layers, $\mathbf{h}_{t-1}$ and $\tilde{\mathbf{h}}_t$ can be the hidden states of one specific layer or the concatenation of the hidden states of all the layers. The MLP can be viewed as measuring the similarity between the current temporary hidden state and the previous hidden state. The advantage of computing an increment skip probability over estimating a skip probability directly is it distributes the probability over a time span, thus the network depends more on the context to make skips. It also prevents the network from making too few skips.

The current skip probability p_t is the sum of the previous accumulated skip probability c_{t-1} and the increment of the skip probability Δp_t:

$$p_t = c_{t-1} + \min(\Delta p_t, 1 - c_{t-1}) \,. \tag{3.4}$$

The min operator is to ensure a valid probability.

The network can choose to skip the current input according to the skip probability p_t. However, it may lead to unstable behaviors of the network. Thus, the probability is fed to a binariser to produce a binary value $u_t \in \{0, 1\}$:

$$u_t = \text{Binariser}(p_t). \tag{3.5}$$

If p_t is above a threshold, then u_t is set to 1. Otherwise, u_t is set to 0. The threshold for the binarization function is set to 0.5 for the original Skip RNN (Campos et al., 2018). In this work, considering the duration of different acoustic events varies, we use an MLP which takes the previous RNN hidden state $\mathbf{h}_{t-1}$ as the input to adjust the threshold. The binarization function is not differentiable. We set the binarization function to behave as a linear function with slope 1 during backpropagation as in Bengio et al. (2013). Thus, during backpropagation,

$$\frac{\partial \,\text{Binariser}(p_t)}{\partial p_t} = 1 \tag{3.6}$$

If u_t is 1, the current input is used to update the RNN hidden state and the accumulation skip probability is reset to 0. Otherwise, the current input is skipped. The current RNN hidden state $\mathbf{h}_i$ and the accumulation skip probability c_t are:

$$\mathbf{h}_t = u_t \tilde{\mathbf{h}}_t + (1 - u_t)\mathbf{h}_{t-1} \tag{3.7}$$

$$c_t = 0 + (1 - u_t)p_t \,. \tag{3.8}$$

Therefore, the proposed network learns to skip input frames dynamically by examining both history and current information. Figure 3.2 shows the architecture of the dsRNN.

The LSTM is similar to the proposed dsRNN, since it also uses gated structures to control the input information. However, in our experiments, we found LSTM and dsRNN behave differently. Furthermore, in our experiments, we use LSTM as the

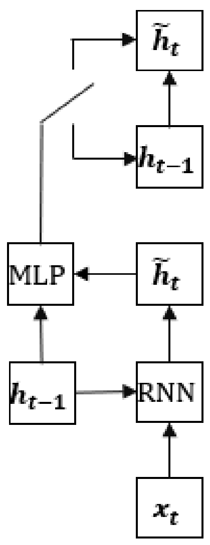


Figure 3.2: An illustration of the dsRNN. It uses both the current temporary hidden state and the previous hidden state to decide if the current hidden state should be a copy of the previous hidden state or it should be updated to the temporary hidden state.

RNN component in our dsRNN. We observed the proposed skip method is compatible with LSTM.

We employ unidirectional dsRNN encoder. If we make dsRNN bidirectional, the forward direction and the backward direction may not always skip the same frame. We have tested concatenating a forward direction dsRNNs with a backward direction dsRNN regardless whether they agree to skip the same frames. However, we observe the bidirectional dsRNN quickly becomes unidirectional during training – in very early epochs the dsRNN in one direction soon learns to skip almost everything. We argue this behavior could be due to that the disagreement of two directions may cause serious mismatches of information in the hidden states of the two directions, and concatenating such hidden states of both directions may harm the performance of the model.

Although our dsRNN is unidirectional, it will be compared with both unidirectional and bidirectional RNNs in our experiments.

3.4 Experimental Setup

We test our model on the TIMIT dataset. The dataset is divided into training, validation and test sets following the Kaldi (Povey et al., 2011) s5 recipe. The input features are 80-dimensional filter-bank features with energy extracted by the Kaldi toolkit. The input features were rescaled by subtracting the mean and divided by the standard deviation from the training set. The outputs are 39 phone with start/end sentence (*sos*/*eos*) and the space token, which make 42 labels in total.

For the encoder-decoder architecture, we use three layers of dynamic subsampling unidirectional LSTM (dsULSTM) as the encoder. The baseline models use three layers of unidirectional LSTM (ULSTM) or bidirectional LSTM (BLSTM) as the encoder. For the baseline encoders, the top two layers read every second hidden state from their previous layers, resulting in the input sequence length being reduced by a factor 4. For all models the decoder is a one layer ULSTM and content-based attention is used (Bahdanau et al., 2015; Luong et al., 2015). We test two settings of the number of hidden units. The first setting is to optimise the dsULSTM encoder. The number of hidden units is 300 for ULSTMs and 150 in each direction for BLSTM. The second setting is to optimise the baseline. The number of hidden units is 512 for ULSTMs and 256 in each direction for BLSTM. The MLP for estimating the increment of the skip probability uses Leaky ReLU (Xu et al., 2015) as its activation function. The number of hidden units of the MLP is 150 for the dsULSTM encoder with 300 hidden units and 100 for the dsULSTM encoder with 512 hidden units respectively.

The Adam algorithm (Kingma and Ba, 2015) is used as the optimisation method. All the models are trained for 25 epochs. Beam search with a beam size 20 is used for decoding. All the models are implemented through the ESPnet toolkit (Watanabe et al., 2018) and PyTorch (Paszke et al., 2019).

3.5 Results and Discussion

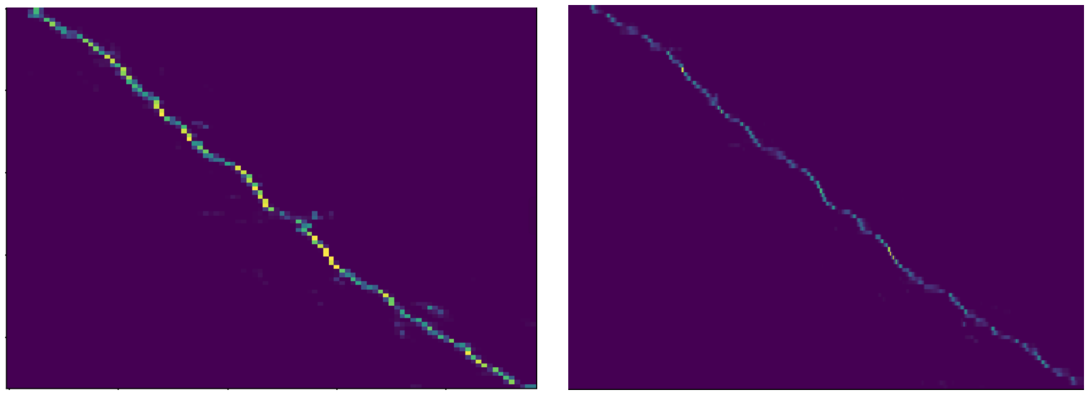
3.5.1 dsULSTM vs. baseline models

First, we compare the dsULSTM with the Skip LSTM (i.e., use the previous hidden state to predict if current input should be skipped), as well as the baseline ULSTM models. Since the LSTM encoders have three layers, we test using the hidden states in the bottom layer, in the middle layer, in the top layer, and the concatenation of the hidden states of all the three layers to make the skip decisions respectively.

The results in Table 3.1 show that subsampling is essential for good accuracy. Our dsULSTM encoder is notably better than LSTM encoder with static subsampling and it indicates that the static subsampling method omits some relevant information. The Skip LSTM does not yield a competitive PER, which implies that for speech recognition, skipping input frames blindly is harmful.

Table 3.1: The phone error rates (PER) on TIMIT test set of using hidden states in different layers to make the skip decisions.

Layer used for skip decision	PER
Bottom	23.8%
Middle	20.5%
Top	**19.4%**
Concatenation of all layers	22.5%
Skip LSTM (use the top layer)	26.4%
ULSTM	29.4%
ULSTM + static subsampling	24.7%



(a) static subsampling (b) dynamic subsampling

Figure 3.3: Attention vectors of the model with the static subsampling ULSTM encoder and the dsULSTM encoder.

Since our proposed dsRNN compares the temporary hidden states of the input frames, it has some similarities with the attention mechanism. We also sampled one utterances from the TIMIT dev set and plotted the attention vectors generated by the dsRNN with the top layer and the ULSTM with static subsampling baseline. Figure 3.3 shows the attention vector. With the dsRNN, the attention mechanism is able to generate a left-to-right alignment, which implicates the dsRNN helps the model to encode most relevant information.

To better understand the dsULSTM encoder, we make a statistic of the average ratio between the number of skipped frames and the number of all frames for each sequence on the TIMIT development set for each training epoch. The average skip ratios are shown in Figure 3.4. The skip ratios go to zero after the first few epochs,

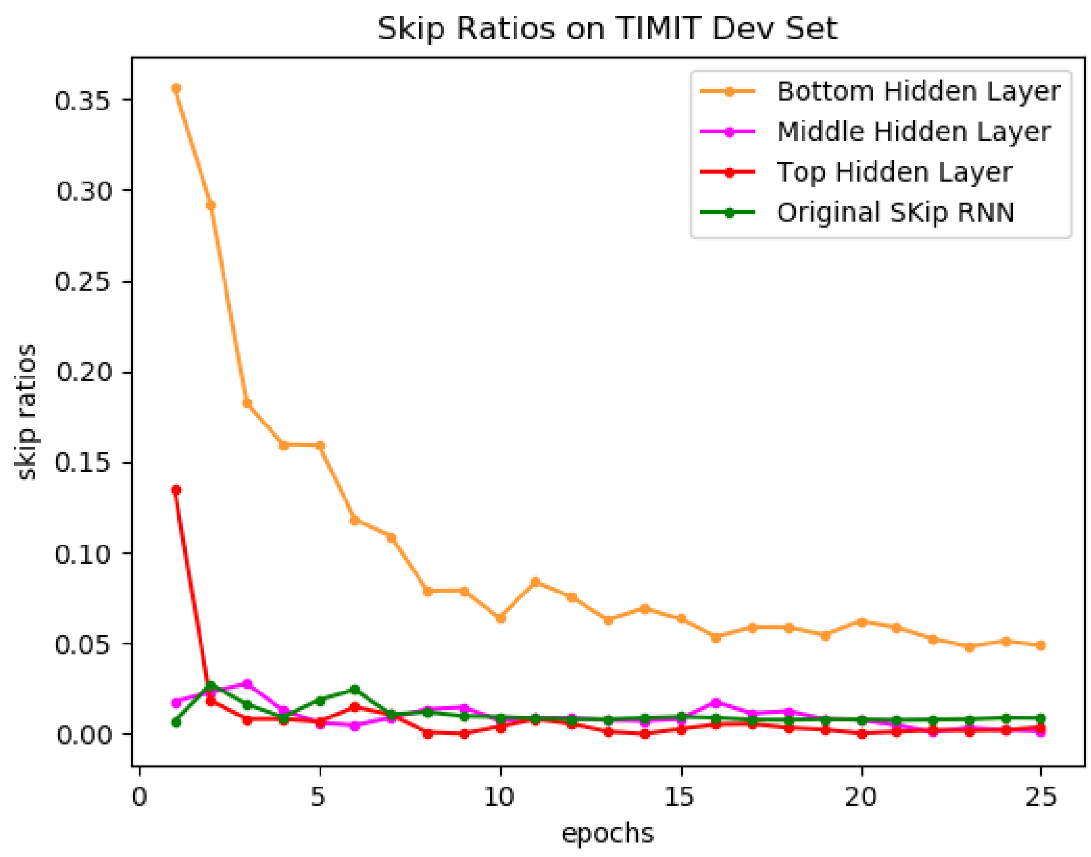


Figure 3.4: The average skip ratios on TIMIT development set for each epoch.

except when using the bottom layer to decide the skips. This is unsurprising. We argue that when using the bottom layer, the hidden states are similar to each other when the input frames are similar, and leads to high skip rates. When using the top layer, the hidden states are more abstract and well separated and resulting in low skip rates. We find that high skip rates in the first epochs is important for low PER – it forces the encoder to learn the abstract information of the most relevant frames at the beginning of the training. Then, the low skip rates in the remaining epochs retain more input information. Both Skip LSTM and dsULSTM using middle layer to decide skips have overall low skip ratios but Skip LSTM has a much higher PER. Furthermore, Skip LSTM also has a higher PER than the static subsampling method. Thus, we conclude Skip LSTM often makes unsuitable skips.

3.5.2 dsULSTM vs. random subsampling

We observe that when using the top layer or the bottom layer to decide the skips, even in the first epoch, the network already learns to make meaningful skips (e.g. it always skips the silence part). To further show our model makes meaningful skips rather than skipping randomly, we compare dsULSTM with random skips. In our best model (using the top layer), the skip rates after the first epoch and the second epoch are 0.14 and 0.02. Thus, we test randomly skipping inputs during training with a fixed probability 0.14 and with a fixed probability 0.02, respectively.

To better mimic the behaviour of dsULSTM, we test randomly skipping input with skip probability decay. Since the weights of dsULSTM are initialised around 0, the initial skip rate is close to 0.5 because of the sigmoid function. Thus, the initial skip probability is set to 0.5 for skip probability decay with random skips. The skip probability is 0.14 for epoch 2 and 0.02 for epoch 3. After the third epoch, the skip probability is 0.004, which is the average skip rates of dsULSTM for the remaining epochs. We also compare the learned skip probability decay policy with two other skip probability decay strategies. The first method is decaying exponentially: the ratio between the current skip probability and the previous skip probability is $(\frac{1}{2})^n$, where n is the number of epochs. The skip probability stops decaying after the third epoch, since the probability is already close to 0. The second approach is to lower the initial skip probabilities.

Surprisingly, Table 3.2 indicates that a random skip probability 0.14 has a better PER compared with ULSTM with static subsampling. We also notice that the PER

Table 3.2: PER on TIMIT test set of randomly skipping inputs during training.

Fixed random skip probability	PER
0.2	26.7%
0.14	21.6%
0.1	22.7%
0.02	29.0%
Skip probability decay	
$0.5 \rightarrow 0.14 \rightarrow 0.02 \rightarrow 0.004$	21.1%
$0.5 \rightarrow 0.25 \rightarrow 0.0625 \rightarrow 0.0078$	23.1%
$0.35 \rightarrow 0.1 \rightarrow 0.02 \rightarrow 0.004$	23.7%
dsULSTM	**19.4%**
ULSTM + static subsampling	24.7%

increases when the random skip probability diverges from 0.14. Thus, we conclude that for this architecture, the average redundant frame rate is around 0.14. A random skip probability of 0.02 does not filter enough irrelevant frames nor sufficiently reduce the length of the input sequence and leads to a poor result.

Although a fixed skip probability of 0.14 is beneficial and a fixed skip probability of 0.02 is harmful, when combined the PER is lower than using the skip probability 0.14 alone. Moreover, all the tested skip probability decay policies yield better results than ULSTM with static subsampling. This further supports the hypothesis that high skip rates at the beginning of training are helpful, even when the skips are randomly made. In the early epochs, due to random skips, the inputs are shortened – it is easier for the encoder to learn the hidden representation of shortened sequences. Then, in the remaining epochs, low skip probabilities ensure the encoder taking all inputs. In contrast, static subsampling misses some input information.

Finally, we argue that our model learns the best dynamic skip strategy and makes meaningful skips. Firstly, both the best random skip rate (0.14) and the best skip rate decay method are learned by our model. Second, with a similar skip rate for each epoch during training, our dsULSTM has a lower PER than the best skip rate decay policy with random skips, which indicates that the dsLSTM makes meaningful skips rather than skipping randomly.

All the experiments above for random skipping and dsULSTM use architectures which are optimised for the dsULSTM (3 hidden layers with 300 hidden states for

Table 3.3: PERs of different encoder models on TIMIT test set.

Encoder: 3 layers with 300 hidden units	PER
ULSTM	29.4%
ULSTM + static subsampling	24.7%
BLSTM + static subsampling	21.1%
dsULSTM	**19.4%**
Encoder: 3 layers with 512 hidden units	
ULSTM	28.3%
ULSTM + static subsampling	23.5%
BLSTM + static subsampling	20.4%
dsULSTM	**19.9%**

ULSTM encoders and 3 hidden layers with 150 hidden states in each direction for the BLSTM encoder). We also test our model on an architecture optimised for the baseline (3 hidden layers with 512 hidden states for ULSTM encoders and 3 hidden layers with 256 hidden state in each direction for the BLSTM encoder). Results are in Table 3.3. The dsULSTM encoders result in lower PER than the BLSTM encoders, indicating that the context information gained from BLSTM is inferior to the information loss from the static subsampling.

3.5.3 dsULSTM on ULSTM

Static subsampling does not allow LSTM encoders to drop inputs directly – the bottom layer always reads the entire input sequence and subsampling operations are between stacked LSTM layers. Thus, we test to stack two dsULSTM layers on one ULSTM layer. Table 3.4 shows for the second architecture of the encoders, two dsULSTM layers stacked on one ULSTM layer yields significant better results than all other models. This may be due to the fact that the bottom ULSTM layer prevents the model from losing any input information, since the bottom ULSTM does not drop any input frame. However, in the first architecture, though better than static subsampling, stacked dsULSTM on ULSTM is no better than the pure dsULSTM encoder. We notice that the performance of the ULSTM as well as the ULSTM with static subsampling of the first architecture (300 hidden units) is also worse than the second architecture (512 hidden units). Therefore, we argue that in this task, the ULSTM with 300 hidden units may have more difficulties in modelling the sequences. In the model where dsULSTM is

Table 3.4: PERs of different models on TIMIT test set. The bidirectional GRU encoder-decoder is better than our BLSTM models. However, it requires complicated training and decoding procedures (Chorowski et al., 2015).

Encoder: 3 layers with 300 hidden units	PER
ULSTM + static subsampling	24.7%
BLSTM + static subsampling	21.1%
dsULSTM	**19.4%**
dsULSTM on ULSTM	23.3%
Encoder: 3 layers with 512 hidden units	
ULSTM + static subsampling	23.5%
BLSTM + static subsampling	20.4%
dsULSTM	19.9%
dsULSTM on ULSTM	**16.8%**
Results of previous end-to-end models	
CNN CTC (Zhang et al., 2017)	18.2%
BLSTM CTC (Graves et al., 2013)	18.4%
Bidirectional GRU encoder-decoder (Chorowski et al., 2015)	18.7%
Segmental RNNs (Lu et al., 2016)	20.5%

stacked on ULSTM, the dsULSTM relies on the ULSTM to access the input information. If the ULSTM has difficulty in encoding the input sequence, dsULSTM may not be able to be trained well. Thus, the inferior ULSTM architecture (300 hidden units) leads to inferior results. A comprehensive investigation is left as a future work.

3.6 Summary

We proposed a novel dynamic subsampling RNN and used it as the encoder in encoder-decoder models. Compared with statically subsampled RNN encoders, our dsRNN encoder has given significant performance gains. The dsRNN encoder learns a good dynamic subsampling strategy – the skip rate decay. Also, it learns to make meaningful skips rather than skipping randomly. Thereby, at the beginning of training, the skip rates are high and the skips are valid, which makes the encoder learn the hidden representations of the most relevant input frames. As training progresses, the low skip rates retain input information.

Although arguably, the input gate and the forget gate in LSTM should have the ability to drop inputs and copy hidden states dynamically, the dsLSTM has resulted in lower PERs than LSTM for speech recognition experiments on TIMIT. Thus, encouraging RNNs to drop inputs and copy hidden states in a hard fashion could be beneficial for speech recognition tasks.

Although the time complexity of our proposed dsRNN should be similar to the static subsampling dsRNN, we find in practice the training time of the proposed dsRNN is significantly longer. The reason is during training, it is not suitable to assume all the sequences inside the same batch will make skips at the same time steps. To address this, we iterate each sequence in the batch to ensure the model makes suitable skips at proper time steps for every input sequence. However, the iteration inside each batch leads to slow computation. Therefore, although we have obtained positive results on TIMIT, due to computation limitations, we did not test our proposed on other larger datasets such as WSJ.

In conclusion, compared to the RNN/LSTM, our dsRNN enables the encoder to better model the input sequences for ASR tasks.

The proposed method is also suitable for speech translation, since for this task the input sequence is also usually noticeably long. However, This extension is beyond the scope of this thesis.

our proposed dsRNN can also be applied to self-attention. Before we transfer the input sequence into a Key/Query/Value tuple, we can use dsRNN to reduce the length of the input sequence. Encoding the sequence with reduced length may require less computation time. However, adding a dsRNN component to the self-attention harms the parallelisation. This extension is also beyond the scope of this thesis.

Chapter 4

Windowed Attention Mechanisms for RNN Attentional Models

4.1 Introduction

In the previous chapter, we have presented dsRNN, which enables the encoder to better model the long speech input sequences. In this chapter, we study windowing methods which force the attention mechanism to focus on local context information and generate left-to-right alignments.

As discussed in Chapter 2, the attention mechanism models the alignment between the inputs and the outputs. For ASR tasks, the link between output units and acoustic events is usually left-to-right. However, the traditional attention mechanisms are too flexible to guarantee monotonic left-to-right alignments between the input and output sequences. Since the attention mechanism relies heavily on the content of the hidden representations of the input units, a left-to-right alignment is easily corrupted by similar input speech fragments or noise (Chan et al., 2016; Kim et al., 2017). Furthermore, the attention mechanism considers all input frames when calculating the alignment vector. However, since one output unit usually only corresponds to a small input time span, it is arguably unnecessary to consider the entire input sequence when estimating the alignment.

We propose a novel fully-trainable windowed attention for speech recognition to generate left-to-right attention. At each decoding time step, instead of considering the entire input sequence, the attention mechanism only considers the inputs within a window. Within the window, a Gaussian function or a concatenation of two sigmoid functions are used to predict a location score for the alignment. The location score alleviates the problem of extensively using the content of hidden states to generate the alignments. We employ a model to predict the step (shift) size and window size, consequently making these two window parameters trainable.

We also use content-based attention to generate content scores, since the location information is processed by the windowing mechanism. In the proposed approach the final attention score is the product of the location score and the content score.

We test the proposed method on TIMIT and WSJ. We observe the learned window size is notably small. Thus, in term of speed, the time complexity of the proposed windowed attention is $\mathbf{O}(1)$ at each decoding time step. In contrast, at each decoding time step, the time complexity of conventional attention mechanisms is $\mathbf{O}(n)$, where n denotes the length of the input sequence. In terms of accuracy, the proposed model outperforms content-based attention on TIMIT and WSJ. It also has comparable results to CTC-attention on WSJ. However, the proposed model is a pure encoder-decoder model and it does not have the additional CTC loss. When combined with CTC-attention, the proposed method leads to faster decoding without harming the performance.

4.2 Related Work

We have presented content-based attention mechanism and location based attention mechanism in Section 2.3.2. Content-based attention does not consider the location information or the alignments of the previous time steps. Several mechanisms have been proposed for *location-aware* or *coverage-aware* attention. At a decoding time step, the attention vector at the previous time step or the alignments of all previous output units are exploited to generate the current attention vector. Thereby, the attention mechanism has an awareness of the location or the coverage (Chorowski et al., 2015; Chorowski and Jaitly, 2017; Zeyer et al., 2018). However, there is no restriction on where the attention mechanism could attend, and these approaches does not guarantee the best performance (Prabhavalkar et al., 2017).

CTC-attention implicitly encourages monotonic attention by adding a CTC loss to the encoder of encoder-decoder models (Kim et al., 2017; Hori et al., 2017). In this case, CTC provides a soft left-to-right constraint. Windowing methods put a hard location constraint on the attention mechanism. Previous works have restricted attention to inputs within a fixed length window. Bahdanau et al. (2016) set the window centre to the position of the median of the attention weights and used a window size of 8 seconds. Chiu and Raffel (2018) proposed to restrict the attention into moving chunks.

However, it uses both the encoder hidden states and the decoder hidden state to decide the movement of the chunk. Also, the chunk size is fixed. Prabhavalkar et al. (2017) proposed to move the fixed length window if an "end-of-window" symbol is generated, and they found the model's performance improves as the window size increases. When the step size is learned, a short fixed-size window can yield good results (Tjandra et al., 2017). However, there is no analysis on why with a trained step size small window can suffice. and the window size is still not trainable. Nguyen et al. (2020) proposed to use differentiable windows to restrict the attention in self-attention based models. However, rather than generating left-to-right attention, the goal of this previous work is to encourage the attention to focus more on specific areas on the sequence.

4.3 Windowed Attention Mechanism

As mentioned above, windowed attention mechanisms force the attention mechanism only to attend inputs within a shift window. If the window size is small, the windowing methods force the attention mechanism only consider the local context and generate monotonic left-to-right alignments. We firstly introduce the rule-based windowed attention, then present our proposed fully-trainable windowed attention.

4.3.1 Rule-based windowed attention

In rule-based windowed attention methods (Jaitly et al., 2016; Bahdanau et al., 2016; Prabhavalkar et al., 2017; Sainath et al., 2018), at each decoding time step m, the attention vector α_m can be described as:

$$\alpha_{ms} = \begin{cases} \frac{\exp(e_{ms})}{\sum_{k=o_m-D_l}^{o_m+D_r} \exp(e_{mk})}, & s \in [o_m - D_l, o_m + D_r] \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where s is the index of the encoder hidden states, o_m is the centre of the window, and D_l and D_r are the size of the left half and right half of the window respectively. The method of moving the window and the window length are pre-defined. The window shift and window length are not trainable.

Such rule-based window methods have been found to require a large window size – often close to the length of the entire utterance – in order to get good performance (Bahdanau et al., 2016; Prabhavalkar et al., 2017). However, if the window size is

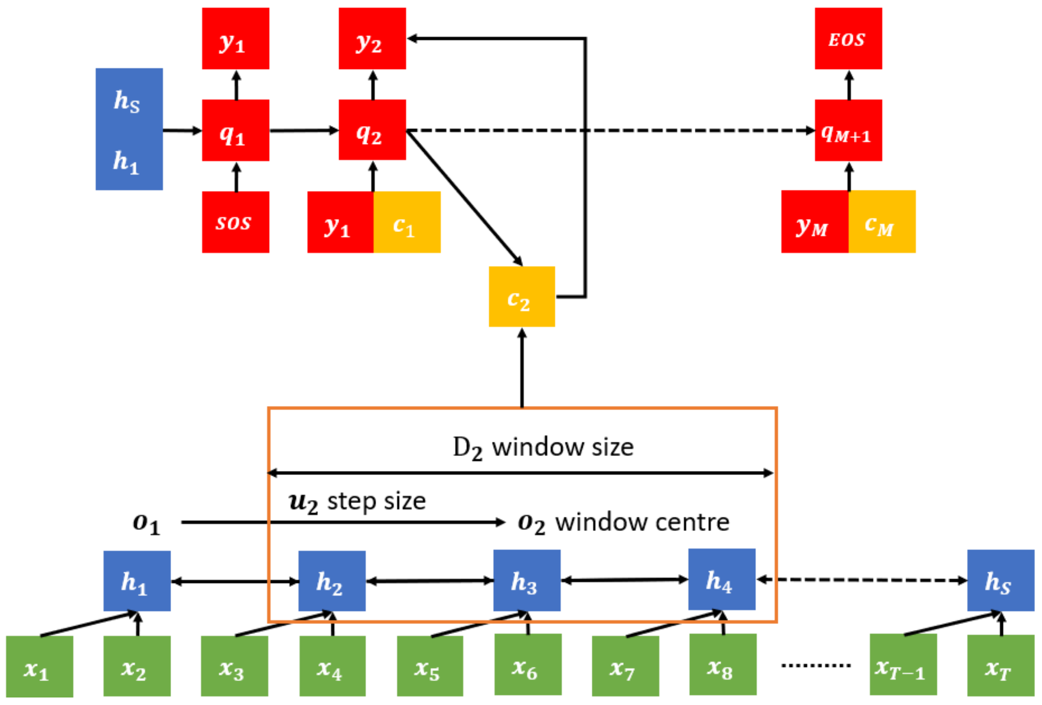


Figure 4.1: The windowed attention mechanism. At decoding time step m, the system uses an MLP to estimate a step size u_m. The centre of the window is moved from o_{m-1} to o_m. Only the encoder hidden states which are within the window will be exploited to calculate the context vector. The window length can either be set as a hyper-parameter or learned.

comparable to the utterance length, then it does not particularly help to generate a monotonic alignment nor to reduce the computational effort.

4.3.2 Fully-trainable windowed attention

We propose a fully-trainable windowed attention (Figure 4.1). In this model, all the window parameters (window shift and window size) are trainable and the learned window size is small. The step size u_m is predicted by an MLP,

$$u_m = U \,\mathrm{sigmoid}(\mathrm{MLP}(\mathbf{q}_m))\,, \tag{4.2}$$

where U is the maximum allowed step size, set as a hyper-parameter. The model records the window centre at the previous decoding time step.

Within the window, we use content-based attention to compute the content score e_{ms}. We use a differentiable function to calculate the location score l_{ms}. The differentiable function also makes the step size and window size trainable. We choose two differentiable functions, the Gaussian function,

$$l_{ms} = \begin{cases} \exp(-\frac{(s-o_m)^2}{2(D_{ml}/2)^2}), & s \in [o_m - D_{ml}, o_m] \\ \exp(-\frac{(s-o_m)^2}{2(D_{mr}/2)^2}), & s \in (o_m, o_m + D_{mr}] \end{cases} \tag{4.3}$$

and the concatenation of two sigmoid functions,

$$l_{ms} = \begin{cases} \sigma(k(s - o_m) + b), & s \in [o_m - D_{ml}, o_m] \\ \sigma(k(o_m - s) + b), & s \in (o_m, o_m + D_{mr}] \end{cases} \tag{4.4}$$

where D_{ml} and D_{mr} denote the length of the left and the right half windows respectively while k and b are hyper-parameters. The final weight is defined as:

$$\alpha_{ms} = \frac{\exp(e_{ms}) l_{ms}}{\sum_{k=o_m-D_{ml}}^{o_m+D_{mr}} \exp(e_{mk}) l_{mk}}. \tag{4.5}$$

and the attention weights α_{ms} still sum to one.

The Gaussian function makes the final weight tend to be large near the centre of the window. Thus, it provides a strong location constraint. The sigmoid functions, when the hyper-parameter b is set large, give location scores that are almost uniformly distributed within the window. Thus, they nearly only provide the indices of the most related encoder hidden states; within the window, the content-based attention almost solely determines the final weight.

When the half window sizes D_{ml} and D_{mr} are equal, the location scores are symmetrically distributed within the window. When they are different, the location score makes the model tend to further rely on the past or the future context. The half window sizes can either be set as hyper-parameters or learned by one MLP so the window is symmetric or learned by two MLPs so the window may be asymmetric. The window size MLP is defined as:

$$D_m = D\,\text{sigmoid}(\text{MLP}(\mathbf{q}_m)) \tag{4.6}$$

where D is the maximum allowed half window size. To prevent the window size from shrinking to zero, we also compute it using:

$$D_m = d + D\,\text{sigmoid}(\text{MLP}(\mathbf{q}_m)) \tag{4.7}$$

where d is set as a hyper-parameter and it controls the minimal length of the window.

We only use decoder hidden state to decide the window parameters since first, our goal is to compute the location score without using the content of the encoder hidden states; second, the decoder hidden state contains the relevant information about the current output unit, and thus we can model the duration of the current output unit through the decoder hidden state.

4.4 Experimental Setups

We performed two sets of experiments: phone recognition on TIMIT and character recognition on WSJ and subsets of WSJ. The TIMIT dataset is split into training, development, and test sets following the Kaldi s5 recipe. We use 80 mel-scale filter-bank features with energy as input features. The outputs are 39 phones with start/end sentence (*sos*/*eos*) and space tokens.

For the WSJ dataset, we use *train_si284* as the training set, *dev93* as the development set and *eval92* as the test set. The input features are 40 mel-scale filter-bank features with delta and delta delta. There are 33 output labels: 26 letters, and the apostrophe, period, dash, space, noise and *sos*/*eos* tokens.

All models are implemented using ESPnet and PyTorch. For the experiments on TIMIT, the encoder is a BLSTM layer on top of two pyramid BLSTM (Chan et al., 2016) layers with 256 hidden units in each direction. The decoder is a one layer LSTM with 512 hidden units. For the experiments on WSJ, the encoder is two BLSTM layers on top of two pyramid BLSTM layers with 320 hidden units in each direction. The decoder is a one layer LSTM with 320 hidden units. We use the AdaDelta (Zeiler, 2012) learning algorithm with gradient clipping (Pascanu et al., 2013). All weights are initialised uniformly within the range $[-0.1, 0.1]$. For decoding, we use a beam search with beam size 20.

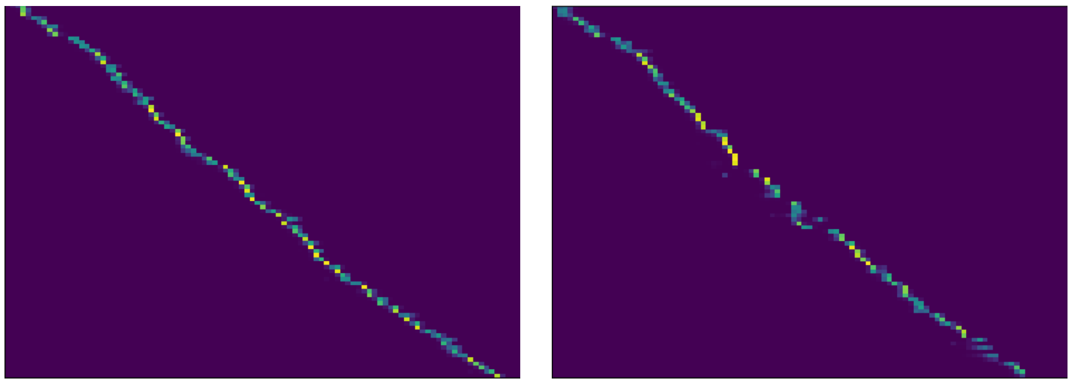
4.5 Results and Discussion

On the TIMIT dataset, the results in Table 4.1 show our fully-trainable model outperforms the content-based attention baseline significantly, reducing the phone error rate (PER) by about 17% relative. Our windowed attention mechanism yields the best result when the window size is trained by two separate MLPs.

When the window size is learned, we set a minimum window size which covers 5 encoder hidden states (0.2s; $d = 2$ in Equation 4.7 for the left/right window) – this is key to successfully learning the step size. Without using a minimum window size, the window sizes for some output units shrink to zero quickly. We also used other methods to alleviate this problem, such as pre-training the model with a fixed size window, making the window size MLP share one bottom layer with the step size MLP, or setting both the maximum step size and window size to a large value. However, although these methods are able to give valid window sizes, they do not yield the best

Table 4.1: PER on TIMIT test set. U, D_l, D_r denote the maximum allowed step size, left window size and right window size. "2 window MLPs" denotes using two separate MLPs to compute the left/right window size. One unit of the step/window size is 0.04s. If using fMLLR features and changing the activation function of the step size MLP and the window size MLP from tanh to LeakyRELU, the network achieves 14.9% PER on the test set.

Function Type	U, D_l, D_r	PER (Test)
Baseline: content-based attention		20.1%
Gaussian-Fixed window	5, 3, 3	17.0%
Gaussian-Fixed window	5, 4, 2	17.3%
Gaussian-Fixed window	5, 2, 4	17.3%
Gaussian-1 window MLP	5, 12, 12	16.8%
Gaussian-2 window MLPs	4, 6, 6	**16.7%**
sigmoid$(\pm 1.5x + 3)$	5, 4, 4	17.8%
sigmoid$(\pm 1.5x + 7)$	5, 4, 4	23.4%
sigmoid$(\pm 1.5x + 7)$	5, 7, 7	19.1%
Hierarchical maxout CNN (Tóth, 2015)		16.5%
Wavenet (Oord et al., 2016)		18.8%



(a) Gaussian (b) Sigmoid

Figure 4.2: Attention vectors generated by different constraint functions. The utterance is fgjd0_sx279 in the TIMIT development set (*/sil ae l s ih z ah sil b ih l ah sil t iy sil t ah w er sil k w ih th aw sil s uw sil p er v ih sh ih n ih z n ow sil w er dh iy sil/*). The horizontal axis denotes the encoder states and the vertical axis denotes the decoder states.

results. Furthermore, we found that if a fixed size window length of 0.2s is used, with the same maximum allowed step size, the model fails completely and is unable to learn a meaningful alignment vector, resulting in a very high PER.

As shown in Figure 4.2, the concatenation of two sigmoid functions can predict the suitable position of the window but the alignments are not proper at many positions. Moreover, when the sigmoid functions are flat within the window, this approach gives worse PER compared to the baseline. When the sigmoid functions are flat, the location score is almost uniformly distributed within the window. Thus, if only provided with the small window, and without a strong location constraint, content-based attention alone cannot generate the best alignment.

We summarise the average step size and its standard deviation of all the 39 phones using all the utterances in TIMIT development set and test set (Figure 4.3). Based on the statistics and our experiments using sigmoid window functions, we hypothesise two reasons for why a large window size is required when using a rule-based step size (Bahdanau et al., 2016; Prabhavalkar et al., 2017). First, the standard deviation is large. Thus, if the step size is fixed or rule-based, it will be often that the step size is smaller or bigger than the best step size. If the window size is small, then the most related encoder hidden states may not be included in the window. Second, even given the proper step size, our experiments indicate that without a strong location constraint within the window the attention mechanism is unable to generate the best alignment when the window size is small.

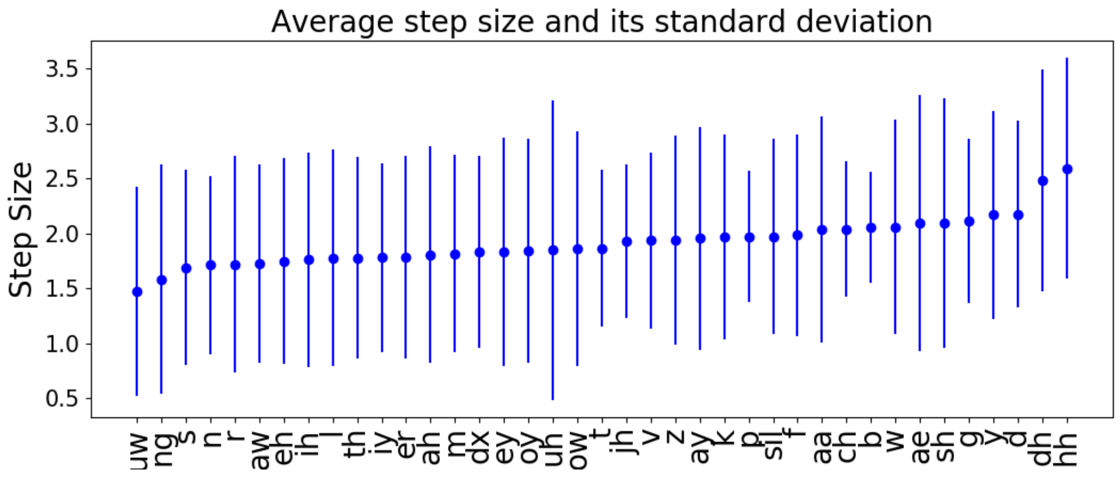


Figure 4.3: The average learned step size for each phone and its standard deviation. The data is collected on TIMIT development set and test set.

On the WSJ dataset our model again outperforms the baseline but does not surpass the CTC-attention model (Table 4.2). We believe a major reason for this is that long pauses often occur in WSJ utterances. Thus, the maximum allowed step size and window size have to be relatively large so the model can jump over the silence part. However, large maximum window and step sizes are not optimal for non-silence parts.

For WSJ, we set both the maximum window size and step size to 1.32s (33 encoder hidden states inside the window), which is small compared to the window size (8.0s) in the rule-based method with a fixed step size (Bahdanau et al., 2016). Furthermore, except for the silence part, the learned step size and window size are much smaller than the maximum size. When combined with CTC, our model yields almost identical results to CTC-attention, but the computational requirements of the CTC-Gaussian attention approach are noticeably lower.

Since the utterances in WSJ begin with silence of varying lengths, and the silence part makes our model difficult to train, we pad the beginning of the text for each utterance with a padding word which is made of eight padding characters. This approach improves the performance of the content-based attention significantly. Thus, we argue using padding to handle the silence can also improve the performance of the baseline system. The lower part of Table 4.2 gives results for reduced subsets of the *train_si284* training data, using 15k and 30k utterances.

Table 4.2: Character error rate (CER) on WSJ. Settings of the encoder-decoder and the optimiser are similar to Kim et al. (2017). After removing the padding characters in the decoding text generated by the baseline location-based attention encoder-decoder model, the CER on eval92 is 7.5%, which is still significantly better the previous work without padding.

Model(train)	CER(dev)	CER(eval)
train_si284	dev93	eval92
Baseline: content-based attention	11.1%	8.9%
location-based attention	9.6%	6.9%
Gaussian-2 window MLPs	9.0%	6.5%
CTC-attention	**7.7%**	5.9%
CTC-Gaussian	7.8%	**5.8%**
Previous results (Kim et al., 2017) (no padding):		
content-based attention	13.7%	11.1%
location-based attention	12.0%	8.2%
CTC-attention	11.3%	7.4%
train_si284 subset (30K)	dev93	eval92
Baseline: content-based attention	9.9%	7.9%
Gaussian-2 window MLPs	9.5%	7.2%
CTC-attention	**9.1%**	**6.9%**
train_si284 subset (15K)	dev93	eval92
Baseline: content-based attention	15.7%	13.7%
Gaussian-2 window MLPs	13.2%	9.6%
CTC-attention	**10.8%**	**8.3%**

4.6 Summary

In this chapter, we made the attention mechanism more effectively model the alignment between the input sequence and the output sequence. Since the link between acoustic events and output units is usually left-to-right, and each output unit is usually corresponding to a relatively small time span, we proposed a fully-trainable windowed attention mechanism to force the attention mechanism only consider local information and generate monotonic left-to-right attention. Compared to previous windowed attention approaches which use a rule-based step size, both the window shift and window length are learned. With small learned window size, our method has given competitive results. The proposed model outperformed the content-based attention and location-based attention on TIMIT and WSJ. It also offered comparable ASR results to CTC-attention on WSJ while having lower computation costs.

Since our proposed method predicts the window size and the step size, it could be applied to streaming ASR. When applying our method to streaming ASR, the encoder does not need to process the entire sequence before the decoder generating outputs. Instead, at each decoding time step, the encoder only needs to process the inputs within the predicted window. Applying our proposed method to streaming ASR is left as a future work.

The proposed method could also be applied to Transformer based models. It may not be suitable to apply our proposed method to all the self-attention layers, since it will eliminate the global information which could be learned by the standard self-attention architecture. For the cross-attention between the Transformer encoder and the Transformer decoder, the proposed method could still be beneficial. However, the goal of this chapter is to study and improve the attention mechanism between the RNN encoder and the RNN decoder. Thus, detailed experiments of applying the proposed method to Transformer models are beyond the scope of this thesis.

Chapter 5

Usefulness of Self-attention Layers for Transformer Models

5.1 Introduction

In Chapter 3 and Chapter 4, we have studied RNN based encoder-decoder models. Based on the characteristics of ASR tasks, we propose methods which enable RNN encoder-decoder models to more effectively model the input sequences and the alignments between the outputs and inputs.

In this chapter, we investigate self-attention networks (SANs). SANs have recently been widely employed for NLP and ASR tasks. For example, they have been used for machine translation (Vaswani et al., 2017), language model pre-training (Devlin et al., 2018; Brown et al., 2020), acoustic modeling (Povey et al., 2018; Wang et al., 2020) and end-to-end ASR (Karita et al., 2019; Gulati et al., 2020). SAN base models can yield superior results compared to RNN based models. When learning the hidden representation for each time step of a sequence, the self-attention mechanism has a global view of the entire sequence and thus can capture temporal relationships without the limitation of range. This is believed to be a key factor for the success of SANs (Vaswani et al., 2017).

However, the reason for the success of SANs leads to a seeming contradiction: if the global view provided by the attention module of self-attention layers is beneficial, why then does forcing the attention mechanism to focus on local information in a left-to-right order result in performance gains for RNN encoder-decoder models?

To investigate this, we study self-attention based encoder-decoder models (i.e., Transformers). Since the self-attention decoder attends both the acoustic information and the language information, we focus on the self-attention encoder which solely processes acoustic frames. We notice that the range of encoded context stacks up from the lower encoder self-attention layers to the upper encoder self-attention layers. Thus, we argue that if the lower layers have learned sufficient contextual information, it is not necessary for the upper layers to have the global view. To verify this, we train Transformers with upper (further from the input) feed-forward layers and lower self-attention layers in the encoder. The feed-forward layers can be viewed as monotonic left-to-right diagonal attention. We perform a series of experiments on the WSJ read speech corpus and the SWBD conversational telephone speech corpus, finding that the upper feed-forward layers do not lead to higher error rates – they even give minor improved accuracies.

To further analyse each self-attention layer, we develop a novel metric of the diagonality of attention matrices. Based on this metric, we find that the overall trend of the average diagonality of each layer indeed increases from the lower layers to the upper layers. The high diagonality of the upper layers implies although given a global view of inputs, the upper layers learn to only attend local information. In contrast, the low diagonality of the lower layers indicates long-range contextual contextual information is captured by the lower layers.

These observations resolve the apparent contradiction between the previous studies on forcing the attention mechanism in RNN encoder-decoder models to be monotonic and the observed success of SAN-based models. For RNN attentional models, the attention mechanism interacts with both the decoder and the encoder. The attention mechanism attends high-level hidden representations of the inputs. Since an output unit (e.g. a character) is often related to a short time span of acoustic features, the attention layer should attend to a small window of the encoded inputs in a left-to-right order. In contrast, self-attention encoders learn the hidden representations of the input sequences. The global view of the input sequences enables the lower layers to encode context information well. Our observations show that when the lower layers capture sufficient contextual information, the self-attention mechanism is not useful for the upper layers.

5.2 Related Work

Press et al. (2019) viewed the multi-head attention component and the feed-forward component in the self-attention layer as individual layers, and they randomly arranged the order of multi-head attention layers and feed-forward layers. They observed the structures that have lower multi-head attention layers and upper feed-forward layers outperform other structures. In this work, we do not decompose the self-attention layers. We demonstrate the upper self-attention layers behave like feed-forward layers. The observations of the previous work and our experiments are consistent.

Previous works have investigated restricting each self-attention layer to attend a small window of context and observed a decrease in accuracy (Wang et al., 2020; Lu et al., 2020). In this work we observed that the lower self-attention layers tend to learn a larger window of context compared to the upper layers. Thus assigning a uniform window length to each layer may not be optimal. We investigate replacing the upper self-attention layers with feed-forward layers. The upper feed-forward lower self-attention layers encoders can be viewed as imposing a window of length one for the upper layers, without restricting the window length for the lower layers.

When the upper self-attention layers are replaced with feed-forward layers, the architecture of the encoder is similar to the CLDNN (Convolutional, Long Short-Term Memory Deep Neural Network) (Sainath et al., 2015). The CLDNN uses an LSTM to model the sequential information and a deep neural network (DNN) to learn further abstract representation for each time step. Stacking a DNN on an LSTM results in a notable error rate reduction compared to pure LSTM models. In this work, we observe that the self-attention layers gradually behave like feed-forward layers. However, we find stacking more feed-forward layers on self-attention layers does not result in further performance gains.

5.3 Feed-Forward Transformer Encoder Layers

In Transformer encoder, since each self-attention layer learns contextual information from its lower layer, the span of the learned context increases from the lower layers to the upper layers. Since acoustic events often happen within small time spans in a left-to-right order, we argue that if the inputs to the upper layer have encoded a sufficient large span of context, then it is unnecessary for the upper layers to learn further temporal relationships. In this situation, the self-attention module which extracts the contextual information could be redundant. However, if modelling contextual information is unnecessary, the self-attention mechanism could only focus on a narrow range of inputs to model local information. That is, we argue that when the lower layers have encoded sufficient context and when the upper layers only need to model local information, for each attention head in each upper layer:

$$\text{softmax}(\frac{\mathbf{QK}^T}{\sqrt{d^{\mathrm{K}}}}) \approx \mathbf{I} \tag{5.1}$$

$$\text{softmax}(\frac{\mathbf{QK}^T}{\sqrt{d^{\mathrm{K}}}})\mathbf{V} \approx \mathbf{V} \tag{5.2}$$

where **I** denotes the identity matrix. Thus, the multi-head attention is close to a linear transformation and the self-attention mechanism is not useful. We therefore argue that in this situation, replacing self-attention layers with feed-forward layers will not lead to a drop in accuracy. Thus, for each encoder self-attention layer:

$$\mathbf{X}'_j = \mathbf{X}_j + \text{MHA}(\mathbf{X}_j, \mathbf{X}_j, \mathbf{X}_j) \tag{5.3}$$

$$\mathbf{X}''_j = \mathbf{X}'_j + \text{ReLU}(\mathbf{X}'_j\mathbf{W}^{\text{FF}_1} + \mathbf{b}^{\text{FF}_1})\mathbf{W}^{\text{FF}_2} + \mathbf{b}^{\text{FF}_2} \tag{5.4}$$

where $\mathbf{X}_j = \mathbf{X}''_{j-1}$ and thus denotes the output sequence of the previous layer, MHA denotes the multi-head attention component, we propose to remove the attention heads so the self-attention layer becomes a feed-forward layer:

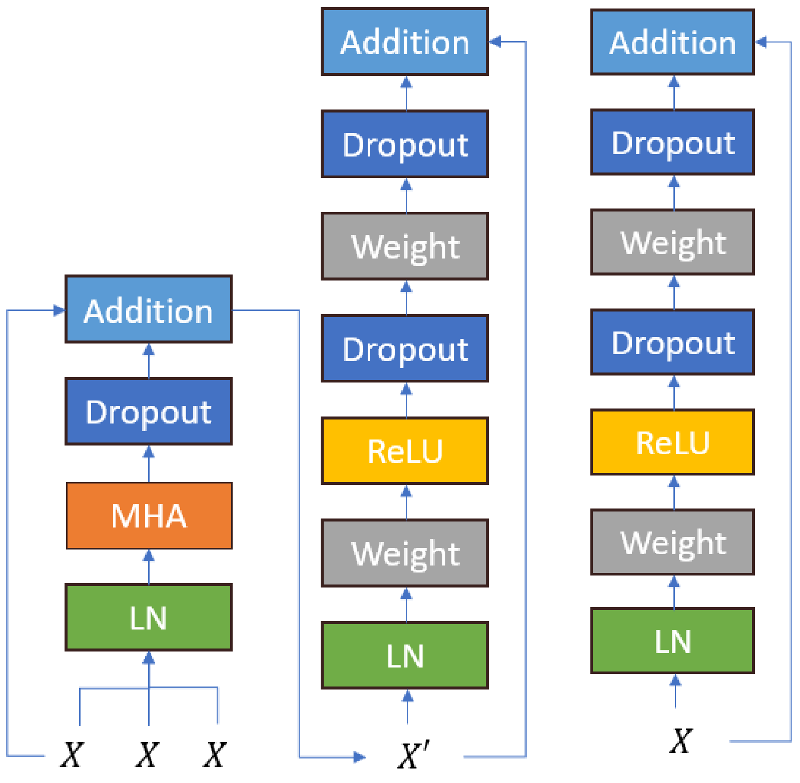
$$\mathbf{X}''_j = \mathbf{X}_j + \text{ReLU}(\mathbf{X}_j\mathbf{W}^{\text{FF}_1} + \mathbf{b}^{\text{FF}_1})\ \mathbf{W}^{\text{FF}_2} + \mathbf{b}^{\text{FF}_2} \tag{5.5}$$

Figure 5.1 demonstrates the architecture of a self-attention layer and a feed-forward layer.

5.4 Metric of Diagonality of Attention Matrices

To further analyse each attention layer, we propose a novel metric of the diagonality of attention matrices, since high diagonality indicates the attention mechanism only focus on local information in a left-to-right order. The j^{th} element in the i^{th} row of the attention matrix is the attention weight between the i^{th} element and the j^{th} element of the input sequence of the self-attention layer. The attention weights sum to 1 in each row and the attention vector can be viewed as a probability distribution over each row. In the i^{th} row, if all the probability mass is allocated to the i^{th} element then it indicates that for this row, all the attention weight is on the diagonal of the attention matrix. When the probability mass is assigned to be as far as possible from the i^{th} element in the i^{th} row for all rows, then the attention matrix has the lowest diagonality. Based on this, we first define the *centrality* C_i of row i:

$$C_i = 1 - \frac{\sum_{j=1}^{n} a_{ij}\,|i-j|}{\text{Max}(|i-1|, |i-2|, \cdots, |i-n|)} \tag{5.6}$$



(a) SA (b) FF

Figure 5.1: Architectures of a self-attention (SA) encoder layer with multi-head attention (MHA) and a feed-forward (FF) encoder layer. LN is layer normalisation (Ba et al., 2016). We omit LN and dropout (Srivastava et al., 2014) in the equations of encoder layers (Equation 5.3 and Equation5.4) but they are applied in the experiments.

where j denotes the index of each column, n denotes the length of the input sequence, a_{ij} denotes the attention weight between the i^{th} element and the j^{th} element of the input sequence, and $|i-j|$ is the distance between the i^{th} element and the j^{th} element of the input sequence. Based on this definition, consider the first row of a 5×5 attention matrix. For such a matrix, $(1,0,0,0,0)$ will have centrality 1, $(0,0,0,0,1)$ will have centrality 0, and $(0.2,0.2,0.2,0.2,0.2)$ will have centrality 0.5. We define the *diagonality* D of an attention matrix as the average over the centrality of all its rows:

$$D = \frac{\sum_{i=1}^{n} C_i}{n}. \tag{5.7}$$

5.5 Experimental Setups

We experiment on WSJ. We use WSJ si284 as the training set. The test sets are WSJ dev93 and eval92. We use Kaldi for data preparation and feature extraction – 83-dim log-mel filterbank frames with pitch (Ghahremani et al., 2014). The output units for the WSJ experiments are 26 characters, and the apostrophe, period, dash, space, noise and *sos* / *eos* tokens.

We employ 12-layer encoders, since a 12-layer architecture is consistent with previous works and has been widely used for Transformer models (Dong et al., 2018; Karita et al., 2019; Michel et al., 2019; Nakatani, 2019; Lu et al., 2020). We also test 6-layer encoders. For both encoder architectures, the Transformer decoder has 6 layers. In each model, below the Transformer's encoder there are two convolutional layers with 256 channels, with a stride of 2 and a kernel size of 3, which map the dimension of the input sequence to $d^{\mathrm{M}} = 256$. The multi-head attention components of the self-attention layers have 4 attention heads. For each attention head, the dimension of the input sequences is $d^{\mathrm{M}} = 256$. The dimension of the Value, Key, Query sequences for each head is $d^{\mathrm{V}} = d^{\mathrm{K}} = d^{\mathrm{Q}} = 64$. And thus the output of the multi-head attention components of each self-attention layer is $d^{\mathrm{M}} = 256$. For the feed-forward module of the self-attention layers, the hidden dimension d^{FF} is 2048. Input sequences to the encoder and the decoder are concatenated with sinusoidal positional encoding (Vaswani et al., 2017). Models are implemented using ESPnet and PyTorch.

We follow the training configuration of previous works (Dong et al., 2018; Karita et al., 2019). Adam is used as the optimiser. The number of warm up steps is 25000. Gradient clip with a factor 3 is used. The batch size is 32. Label smoothing with smoothing weight 0.1 is used. Dropout rate 0.1 is used when dropout is applied. We train the model for 100 epochs and the averaged parameters of the last 10 epochs are used as the parameters of the final model Dong et al. (2018). Besides the loss from the Transformer's decoder L^{D}, a connectionist temporal classification (CTC) loss L^{CTC} is also applied to the Transformer encoder. Following the previous work (Karita et al., 2019), the final loss L for the model is:

$$L = (1-\lambda)L^{\mathrm{D}} + \lambda L^{\mathrm{CTC}} \tag{5.8}$$

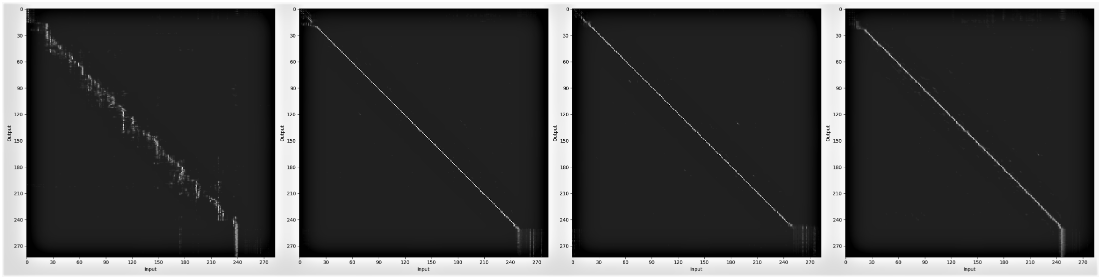
where $\lambda = 0.3$ for WSJ and $\lambda = 0.2$ for SWBD.

5.6 Results and Discussion

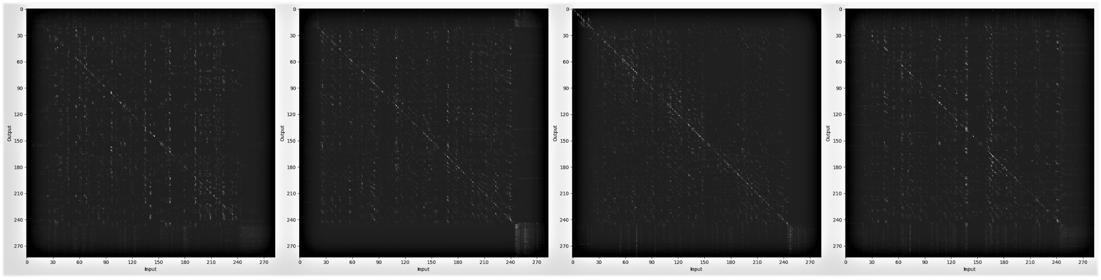
We first train a baseline model with a 12-layer self-attention encoder on WSJ. We then use the baseline model to decode WSJ eval92 and compute the attention matrices of a randomly sampled utterance from eval92. Figure 5.2 shows the plots of the attention matrices for each attention head of the lowest layer, a middle layer and the highest layer. The lowest layer attends to a wider range of context. The middle layers put more attention weight on the diagonal, and the middle two heads of the topmost layer have close to pure diagonal attention matrices. This implies that even given a global view of inputs during training, the topmost layer learned to only focus on local information and the context information is captured by the lower layers.

5.6.1 Replacing Self-attention Layers with Feed-forward Layers

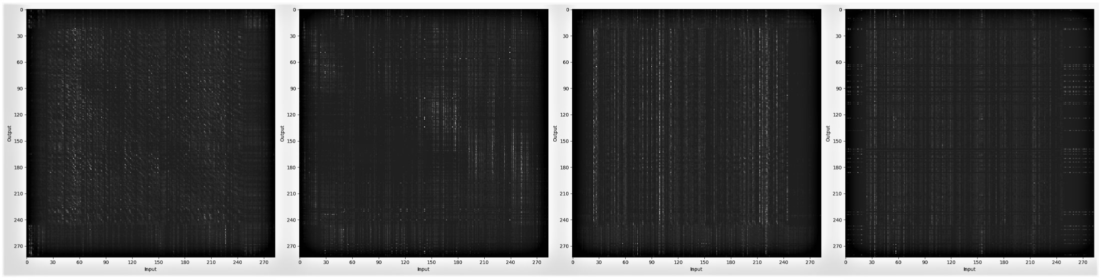
After training the baseline, we train models whose encoders are built by different numbers of self-attention layers and feed-forward layers. For the encoder of these models, there are 12 layers in total and the lower layers are self-attention layers while the upper layers are feed-forward layers. We start from an encoder with 6 self-attention layers and 6 feed-forward layers. Keeping the total number of layers the same, we then increase the number of self-attention layers and decrease the number of feed-forward layers. The purpose of this set of experiments is not to obtain significant improvement gains. Instead, our purpose is to test the importance of the self-attention mechanism for each layer. Compared to the baseline, if there is no drop in performance, then this is an indication of the redundancy of the self-attention. Table 5.1 shows that as the number of self-attention layers increases, the character error rate (CER) decreases, which implies learning further contextual information is beneficial.



(a) Attention matrices of each attention head of encoder self-attention layer 12



(b) Attention matrices of each attention head of encoder self-attention layer 5



(c) Attention matrices of each attention head of encoder self-attention layer 1

Figure 5.2: A sample of attention matrices of encoder self-attention layers generated by the baseline Transformer with a 12-layer encoder. The sampled utterance is form WSJ eval92. While the lowest layer (layer1, near input) attends a wide range of context, the middle layer focus more on the local information and the topmost layer assigns nearly all the attention weight to the diagonal.

However, when the number of self-attention layers increases to 10, with 2 upper feed-forward layers, the encoder gives almost identical results compared to the 12-layer self-attention baseline, although the 10-layer self-attention encoder has notably higher CERs. Furthermore, although the 11-layer self-attention encoder gives worse results compared to the 12-layer baseline, the encoder which has 11 self-attention layers and one upper feed-forward layer yields the best results. Increasing the number of self-attention layers to 12 and decreasing the number of feed-forward layers to 0 is not helpful. This set of experiments shows it is crucial for the layers below the 10^{th} layer to encode temporal relationships. Upon the 10^{th} layer the global view of the sequence is not useful, indicating that the contextual information is well captured by the lower layers.

We further tested if stacking more feed-forward layers to make deeper encoders is beneficial. As shown in Table 5.1, this does not give performance gains. We also investigated modifications to the architecture of the stacked feed-forward layers, such as removing residual connections or using an identity mapping (He et al., 2016). These modifications did not result in a CER reduction compared to the 11-layer self-attention 1-layer feed-forward encoder.

We also tested the 6-layer encoder architecture and the results are also shown in Table 5.1. The baseline model has 6 self-attention layers as its encoder. Then we replace the top one, two and three layers with feed-forward layers respectively. We observe that replacing the topmost layer of the 6-layer self-attention encoder does not lead to reductions in accuracy but to minor improvements, which is consistent with the experimental results for the 12-layer encoder.

It is interesting to see that the performance of the model with 6 self-attention layers and 6 feed-forward layers and the model with only 6 self-attention layers are almost the same. Along with the experiments which stacking feed-forward layers on 11 self-attention layers or 12 self-layers, these experiments show simply stacking feed-forward layers on self-attention layers may not lead to performance gains. However, it is not contradict with the argument when self-attention is not useful, it can be replaced with feed-forward layers. We also observe the upper self-attention layers indeed behave like feed-forward layers, see Section 5.6.2 for a detailed discussion.

We further test replacing upper self-attention layers on the larger and more challenging SWBD corpus. The results are shown in Table 5.2. The encoder with 10 self-attention layers is less accurate than the encoders with 11 and 12 self-attention layers. Also, the 12-layer self-attention encoder has higher word error rates (WERs)

Table 5.1: CERs on WSJ for the Transformer models with different encoders. The evaluation sets are WSJ eval92 and dev93. SA denotes self-attention layer and FF denotes feed-forward layer.

Number of Layers			CER/%	
Total	SA	FF	eval92	dev93
12	12	0	3.5	4.6
12	11	1	**3.4**	**4.5**
12	10	2	3.6	4.6
12	9	3	3.8	4.8
12	8	4	3.9	4.9
12	7	5	4.0	5.1
12	6	6	4.2	5.3
11	11	0	3.6	4.7
10	10	0	4.0	5.2
13	12	1	3.6	4.7
13	11	2	3.6	4.6
14	11	3	3.7	4.6
6	6	0	4.2	5.4
6	5	1	4.2	**5.3**
6	4	2	**4.1**	5.6
6	3	3	4.4	5.9

Table 5.2: WERs of the experiments on SWBD for the Transformer models with different encoders. The evaluation sets are eval 2000 SWBD/callhome. SA denotes self-attention layer and FF denotes feed-forward layer.

Number of Layers			WER/%	
Total	SA	FF	SWBD	Callhome
12	12	0	9.0	18.1
12	11	1	9.0	17.8
12	10	2	**8.9**	**17.6**
12	9	3	9.5	18.5
11	11	0	9.0	17.7
10	10	0	9.2	18.4
Transformer(Karita et al., 2019)			9.0	18.1
Transformer(Pham et al., 2019)			10.4	18.6
Transformer(Zeyer et al., 2019)			10.6	22.3

than the 11-layer encoder. However, the encoder with 10 self-attention layers and 2 feed-forward layers, which has 12 layers in total, gives the lowest WERs. The 9 self-attention layers + 3 feed-forward layers encoder yields higher WERs. Thus, the layers below the 10^{th} layer are crucial in learning contextual information. Upon the 10^{th} self-attention layer feed-forward layers are sufficient in learning further abstract representations.

5.6.2 Diagonality of Self-attention Layers

To further evaluate the usefulness of self-attention for each layer, we compute the average diagonality of each attention head for every layer of the baseline 12-layer encoder model on WSJ eval92, and the average diagonality over all attention heads of each layer. As shown in Figure 5.3, the overall trend of the average diagonality indeed increases from the lower layers to the upper layers. In the experiments on replacing self-attention layers, models with more than 2 feed-forward layers and fewer than 10 self-attention layers yield higher error rates (Table 5.1). Figure 5.3 shows that the average diagonality from the 9^{th} layer to the 10^{th} layer is relatively low, compared to the topmost two layers. These consistent observations indicate contextual information is necessary for the 9^{th} layer and the 10^{th} layer and thus the self-attention mechanism is essential for these two layers. For the topmost two layers, even with the self-attention mechanism, the diagonality is close to 1, which shows that they focus on local information. This is also consistent with the finding in Table 5.1 that replacing these self-attention layers with feed-forward layers leads to no increase in error rate. We also computed the average diagonality of each layer of the baseline 6-layer Transformer encoder. Figure 5.4 shows the trend of the average diagonality increases from the lower layers to the upper layers.

To investigate how the diagonality changes after the upper layers are replaced by feed-forward layers, we compute the average diagonality of each layer of the models with one and two feed-forward layers in the encoder, where the performance does not drop. Figure 5.5 and Figure 5.6 show the overall trend of the average diagonality still increases from the lower layers to the upper layers.

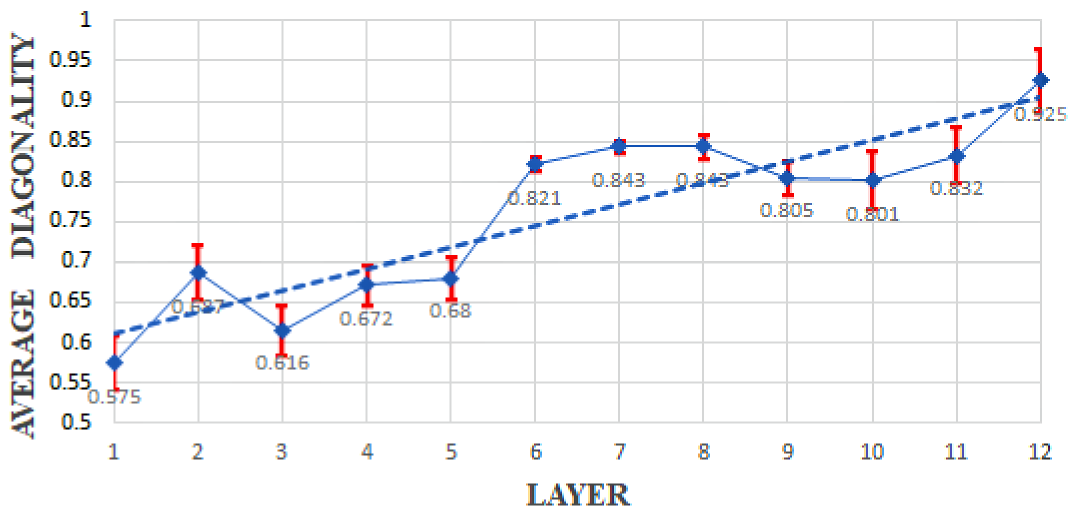


Figure 5.3: The averaged diagonality with ± standard deviation of each self-attention layer of the 12-layer encoder baseline. Layer 12 is the topmost layer. The dash line is the trend line.

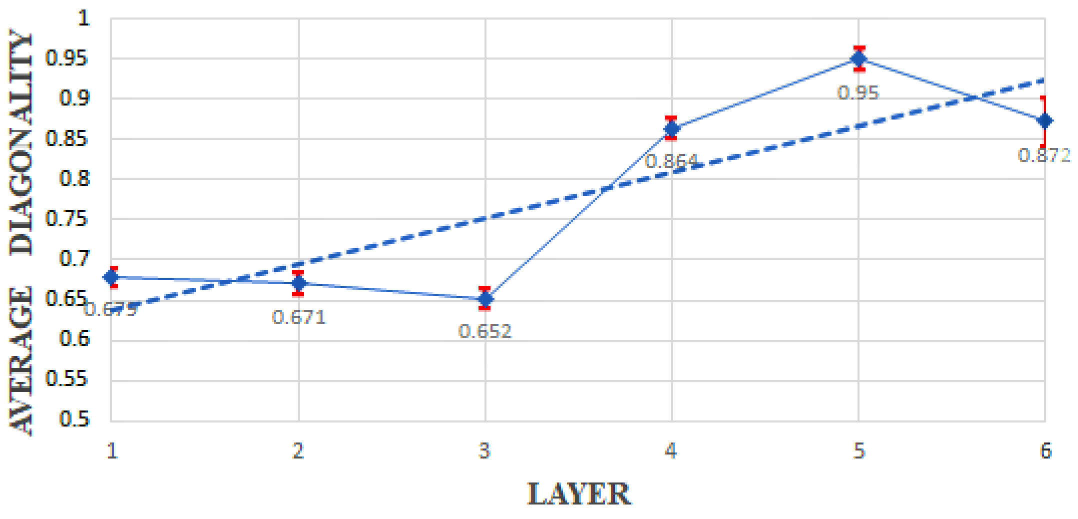


Figure 5.4: The averaged diagonality with ± standard deviation of each self-attention layer of the 6-layer encoder baseline. Layer 6 is the topmost layer. The dash line is the trend line.

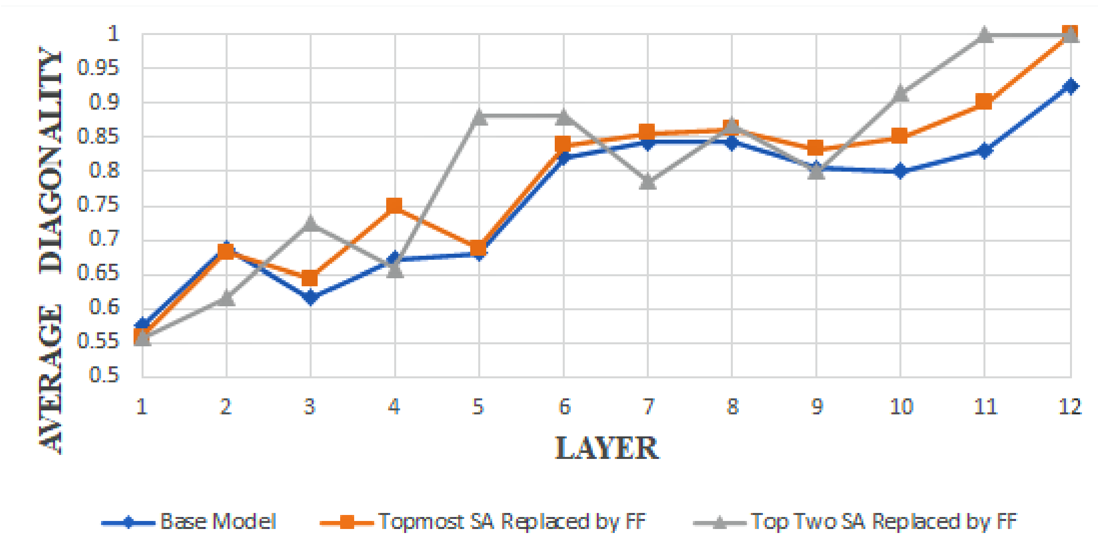


Figure 5.5: The averaged diagonality of each layer of the encoders. Layer 12 is the topmost layer. Feed-forward layers have diagonality 1.

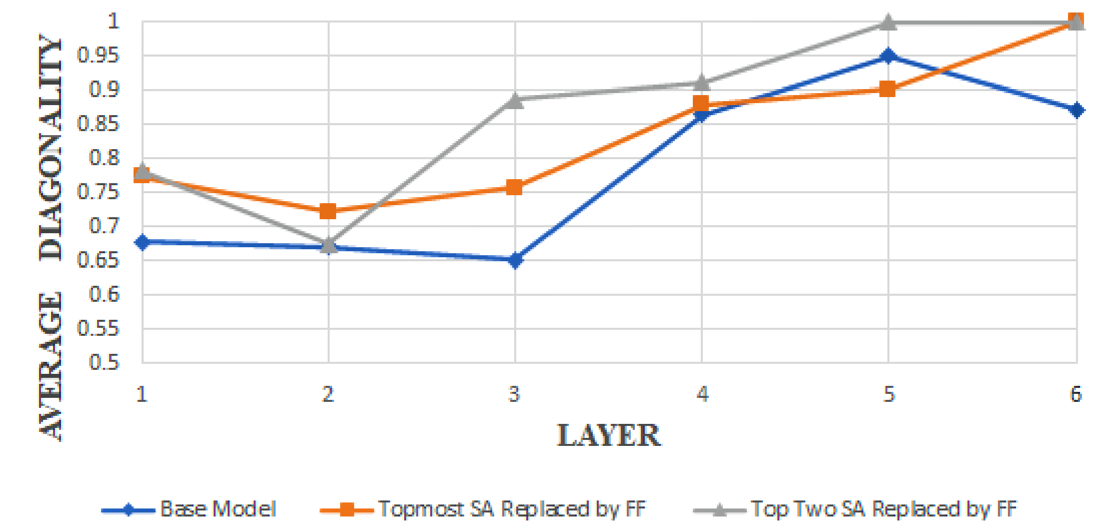


Figure 5.6: The averaged diagonality of each layer of the encoders. Layer 6 is the topmost layer. Feed-forward layers have diagonality 1.

5.7 Summary

In this chapter, based on the observation that restricting the attention mechanism to use only local context to generate left-to-right alignments is beneficial for the RNN encoder-decoder, we investigated if the global view of the self-attention is necessary for ASR tasks. Our experiments on WSJ and SWBD showed that for the encoder in Transformer models, replacing the upper self-attention layers with feed-forward layers does not increase the error rates. We also developed a novel metric of the diagonality of the attention matrix, finding the overall diagonality indeed increases from the lower layers to the upper layers. These observations indicated that the lower layers are able to capture sufficient context information, and in this situation the global view and the self-attention mechanism are not necessary for the upper layers.

Chapter 6

Stochastic Attention Head Removal for Transformer Models

6.1 Introduction

In the previous chapter (Chapter 5), we investigated the usefulness of the self-attention encoder layers, finding that the upper encoder layers can be replaced with feed-forward layers. In this chapter, instead of considering self-attention layers, we study individual attention heads. We analyse the diagonality of individual attention heads. We find that in trained Transformer ASR models, the attention matrices of some attention heads are close to the identity matrices, indicating that the attention mechanism is just an identity mapping. Thus, the input sequence (after a linear transformation) can be directly used as the output of the corresponding attention head.

This observation leads to a question: if there are redundant attention heads in trained Transformer models, then when we train a model from scratch, can we remove some attention heads without harming the model performance? In our experiments, we find some architectures with reduced numbers of attention heads offer better accuracies. However, searching for the best structure is time consuming. To address this, we propose to randomly remove attention heads during training, and keep all the attention heads at the test time. Through this method, the trained model is an ensemble of different architectures.

Our method also enables the networks to better utilise the representation power of the multi-head attention architecture. The redundancy of attention heads in the baselines indicates some heads learn the most crucial patterns while other heads merely extract redundant features which can be disregarded (this is also observed by Voita et al. (2019)). In contrast, our method forces all attention heads to learn crucial features – since heads are randomly removed during training, the model cannot only rely on some specific heads for extracting key features. Thus, the proposed method forces each head to learn useful information and enables the model to better use the representation power of the multi-head attention structure.

We employ the proposed method to train Transformer and Conformer ASR models. Our method offers consistent accuracy improvements on datasets of different ASR scenarios, including WSJ, AISHELL, SWBD and AMI. On SWBD and AMI, to the best of our knowledge, we have achieved state-of-the-art (SOTA) performance for sequence-to-sequence Transformer based models.

6.2 Related Work

At first sight, the proposed method appears to be similar to dropout (Srivastava et al., 2014); however, the objectives of these two methods are different. Dropout randomly drops units in neural networks to prevent the co-adaptation between these units. In this work, we randomly remove entire attention heads, motivated by the observation that some architectures with reduced numbers of attention heads can lead to better performance, but avoiding the need for a time prohibitive search for the best architecture. Instead, we effectively train an ensemble of different structures. Furthermore, we observe our method forces the attention heads which are redundant in the baseline systems to learn useful patterns. We also have employed dropout in our experiments and found it is complementary to our method.

Previous works have proposed to use a development set (Michel et al., 2019) or a trainable hard gate head (Voita et al., 2019) to drop attention heads. These methods remove attention heads in a static manner – once an attention head is removed during training, it will never be present in the structure. Also, little performance gain is obtained by these methods. Our proposed method removes attention heads randomly and consistently yields better accuracies.

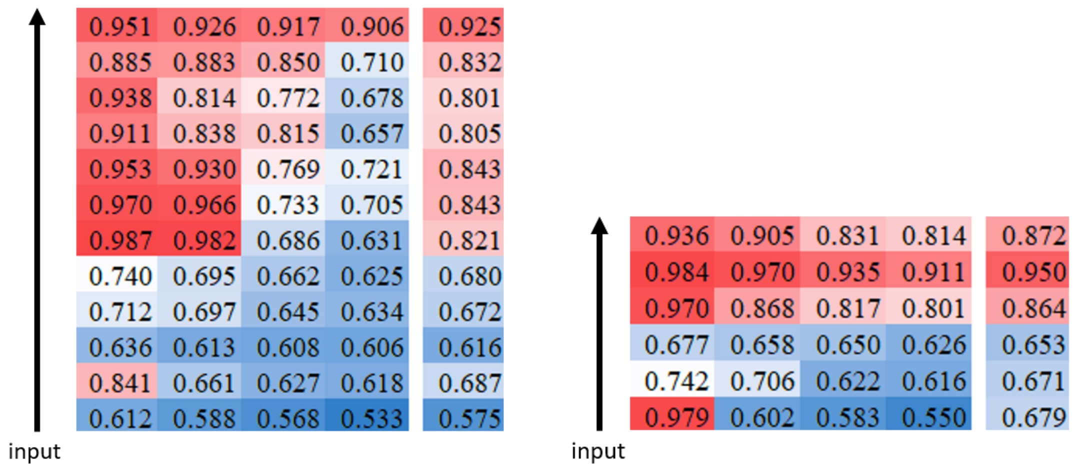
To build very deep networks, previous works have proposed to drop self-attention layers randomly or following a schedule (Pham et al., 2019; Fan et al., 2019). During training, the network becomes shallower dynamically and thus the gradient vanishing problem is alleviated. At testing, all the layers are kept and thus the final trained model is still a deep neural network. Rather than dropping self-attention layers, in this work we remove attention heads randomly to train models with different number of attention heads. Our method is compatible with these previous works. The combination of our method and the previous work will results in shallower networks with different number of attention heads during training.

Peng et al. (2020) proposed to train subsets that contain $h-1$ elements of the h attention heads. During testing, the output is a linear combination of the outputs of these subsets. In our work, the attention heads are randomly removed. Therefore, the number of trainable attention heads changes dynamically.

6.3 Experimental Setups

We experiment on datasets of a variety of scenarios, including WSJ (reading), AISHELL (reading Chinese), SWBD (telephone conversation) and AMI (meeting). We use Kaldi for data preparation and feature extraction – 83-dim log-mel filterbank frames with pitch (Ghahremani et al., 2014). SpecAugument (Park et al., 2019) is used on all datasets but WSJ. On all datasets, following the previous works (Dong et al., 2018; Karita et al., 2019; Lu et al., 2020), we employ the **12layer Encoder-6layer Decoder** architecture. The multi-head attention components of the self-attention layers have 4 attention heads and $d^{\mathrm{V}} = d^{\mathrm{K}} = 64$, $d^{\mathrm{M}} = 256$. For the feed-forward module of the self-attention layers, $d^{\mathrm{FF}} = 2048$. Below the encoder of the Transformer, there are two convolutional layers with 256 channels, with a stride of 2 and a kernel size of 3, which map the dimension of the input sequence to d^{M}. For Conformer models, the structure of the self-attention component is the same as the Transformer. The kernel size of the convolution component is either 31 for SWBD and 15 for other datasets. Input sequences to the encoder and the decoder are concatenated with sinusoidal positional encoding (Vaswani et al., 2017; Dai et al., 2019). All models are implemented using ESPnet and PyTorch.

The training schedule follows Dong et al. (2018). Dropout rate 0.1 is used. The training lasts 100 epochs for WSJ and 50 epochs for other datasets. The averaged parameters of the last 10 epochs are used as the parameters of the final model (Dong et al., 2018). Adam is used as the optimiser. The output units for WSJ/AISHELL/AMI are characters and the output tokens for SWBD experiments are tokenised at word-level using Byte Pair Encoding (BPE) (Sennrich et al., 2016). The batch size is 32. Label smoothing with smoothing weight 0.1 is used. Besides the loss from the Transformer's decoder L^{D}, a CTC loss L^{CTC} is also applied to the Transformer/Conformer encoder.



0.951	0.926	0.917	0.906	0.925
0.885	0.883	0.850	0.710	0.832
0.938	0.814	0.772	0.678	0.801
0.911	0.838	0.815	0.657	0.805
0.953	0.930	0.769	0.721	0.843
0.970	0.966	0.733	0.705	0.843
0.987	0.982	0.686	0.631	0.821
0.740	0.695	0.662	0.625	0.680
0.712	0.697	0.645	0.634	0.672
0.636	0.613	0.608	0.606	0.616
0.841	0.661	0.627	0.618	0.687
0.612	0.588	0.568	0.533	0.575

(a) 12-layer Encoder

0.936	0.905	0.831	0.814	0.872
0.984	0.970	0.935	0.911	0.950
0.970	0.868	0.817	0.801	0.864
0.677	0.658	0.650	0.626	0.653
0.742	0.706	0.622	0.616	0.671
0.979	0.602	0.583	0.550	0.679

(b) 6-layer Encoder

Figure 6.1: The heat map of the averaged diagonality of each attention head in each layer. The topmost row denotes the topmost layer. The 5^{th} column shows the average diagonality over all heads of each layer. The red color denotes high diagonality and the blue color indicates low diagonality.

Following Karita et al. (2019), the final loss L for the model is:

$$L = (1-\lambda)L^{\mathrm{D}} + \lambda L^{\mathrm{CTC}}. \tag{6.1}$$

where $\lambda = 0.2$ for SWBD and $\lambda = 0.3$ for other datasets.

6.4 Diagonality of Individual Attention Heads

We study the diagonality of individual attention heads in this section. We train two baseline Transformers on WSJ. For the first baseline, the encoder has 12 layers. For the second baseline, the encoder has 6 layers. Then, for all utterances of WSJ eval 92, we compute the average diagonality of each individual attention head in each encoder layer of these two baselines.

Figure 6.1 shows the heat map of the mean diagonality of each attention head. Table 6.1 and Table 6.2 show the averaged diagonality with one standard deviation of each individual attention heads. We observe that the standard deviation is small, which indicates whether an attention head focuses on local or global information varies marginally over different utterances. The attention heads which have large diagonality almost always only attend local information, indicating the redundancy of the self-attention mechanism for these heads. We also notice that for some layers, the diagonality of individual heads differ significantly from each other – although the averaged diagonality of the entire layer is high, the diagonality of some individual heads could be low. In Chapter 5, replacing the entire self-attention layer with feed-forward layer does not consider the possibility that the diagonality of individual attention heads may vary within the same layer. Thus, based on these observations, we propose to remove the individual attention heads which have high diagonality and retrain the model from scratch.

Table 6.1: The mean diagonality $\pm$ standard deviation for all attention heads of each layer for the 12-layer encoder. The topmost row denotes the topmost layer.

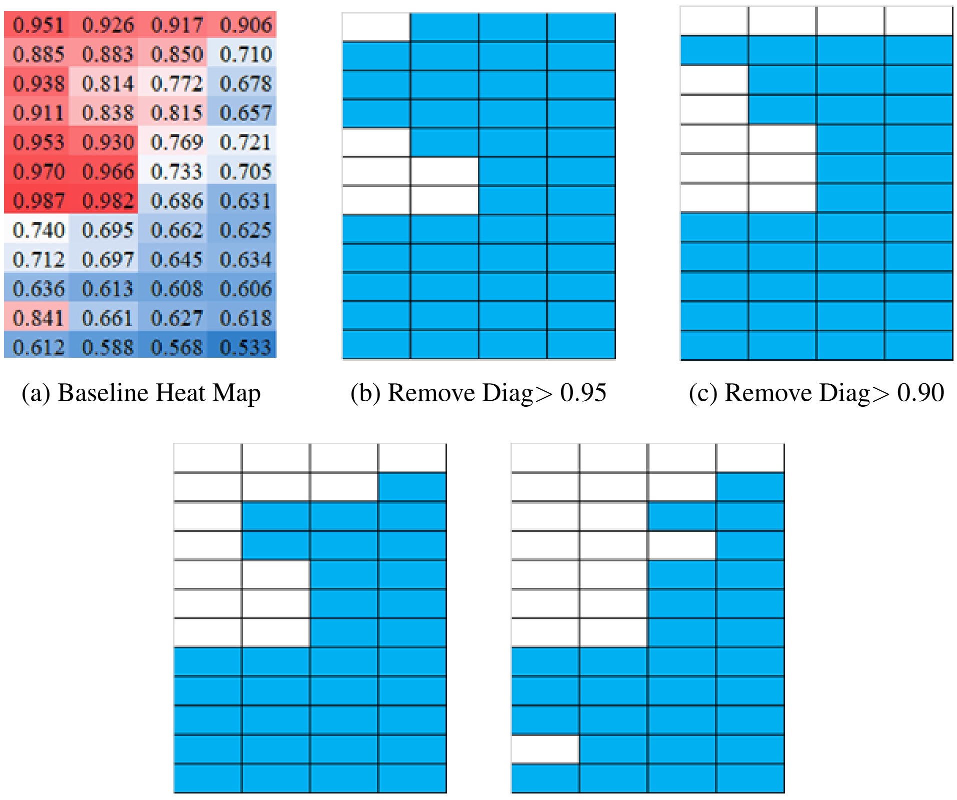
(near output)	head 1	head 2	head 3	head 4
Layer 12	.906 $\pm$ .041	.951$\pm$ .028	.917 $\pm$ .044	.926 $\pm$ .026
Layer 11	.850 $\pm$ .037	.885$\pm$ .032	.710 $\pm$ .043	.883 $\pm$ .043
Layer 10	.938 $\pm$ .028	.814$\pm$ .060	.772 $\pm$ .041	.678 $\pm$ .055
Layer 9	.815 $\pm$ .038	.838$\pm$ .034	.657 $\pm$ .035	.911 $\pm$ .036
Layer 8	.930 $\pm$ .025	.769$\pm$ .027	.721 $\pm$ .030	.953 $\pm$ .021
Layer 7	.966 $\pm$ .022	.970$\pm$ .016	.705 $\pm$ .027	.733 $\pm$ .026
Layer 6	.686 $\pm$ .026	.631$\pm$ .027	.982 $\pm$ .009	.987 $\pm$ .009
Layer 5	.662 $\pm$ .031	.695$\pm$ .031	.740 $\pm$ .021	.625 $\pm$ .033
Layer 4	.697 $\pm$ .030	.645$\pm$ .024	.634 $\pm$ .032	.712 $\pm$ .027
Layer 3	.608 $\pm$ .029	.613$\pm$ .032	.636 $\pm$ .035	.606 $\pm$ .032
Layer 2	.627 $\pm$ .033	.661$\pm$ .056	.618 $\pm$ .035	.841 $\pm$ .031
Layer 1	.533 $\pm$ .038	.568$\pm$ .034	.612 $\pm$ .035	.588 $\pm$ .040

Table 6.2: The mean diagonality $\pm$ standard deviation for all attention heads of each layer of the 6-layer encoder. The topmost row denotes the topmost layer.

(near output)	head 1	head 2	head 3	head 4
Layer 6	.905 $\pm$ .028	.814$\pm$ .023	.831 $\pm$ .049	.936 $\pm$ .025
Layer 5	.984 $\pm$ .006	.911$\pm$ .023	.935 $\pm$ .020	.970 $\pm$ .011
Layer 4	.817 $\pm$ .028	.970$\pm$ .009	.801 $\pm$ .022	.868 $\pm$ .023
Layer 3	.658 $\pm$ .021	.650$\pm$ .015	.677 $\pm$ .018	.625 $\pm$ .022
Layer 2	.622 $\pm$ .023	.616$\pm$ .014	.706 $\pm$ .019	.742 $\pm$ .004
Layer 1	.550 $\pm$ .044	.979$\pm$ .005	.583 $\pm$ .013	.602 $\pm$ .017

6.5 Static Attention Head Removal

We firstly test removing attention heads in a static way. That is, we remove the attention heads that have diagonality above a threshold in the trained model. We then reinitialise and and retrain the model. Figure 6.2 shows the diagonality heatmap of the baseline model and the structures with the corresponding head removal strategy.



0.951	0.926	0.917	0.906
0.885	0.883	0.850	0.710
0.938	0.814	0.772	0.678
0.911	0.838	0.815	0.657
0.953	0.930	0.769	0.721
0.970	0.966	0.733	0.705
0.987	0.982	0.686	0.631
0.740	0.695	0.662	0.625
0.712	0.697	0.645	0.634
0.636	0.613	0.608	0.606
0.841	0.661	0.627	0.618
0.612	0.588	0.568	0.533

(a) Baseline Heat Map (b) Remove Diag> 0.95 (c) Remove Diag> 0.90

(d) Remove Diag> 0.85 (e) Remove Diag> 0.80

Figure 6.2: The tested encoder architectures. We remove the attention heads based on their diagonality in the baseline model. The baseline has 12 self-attention layers. Each non-white cell represents an attention head. Each white cell denotes there is no attention head. When all the attention heads of a layer are removed (four white cells), the feed-forward component of that layer is preserved. (a) heatmap of the averaged diagonality of each attention head. (b-e) structures where some attention heads are removed.

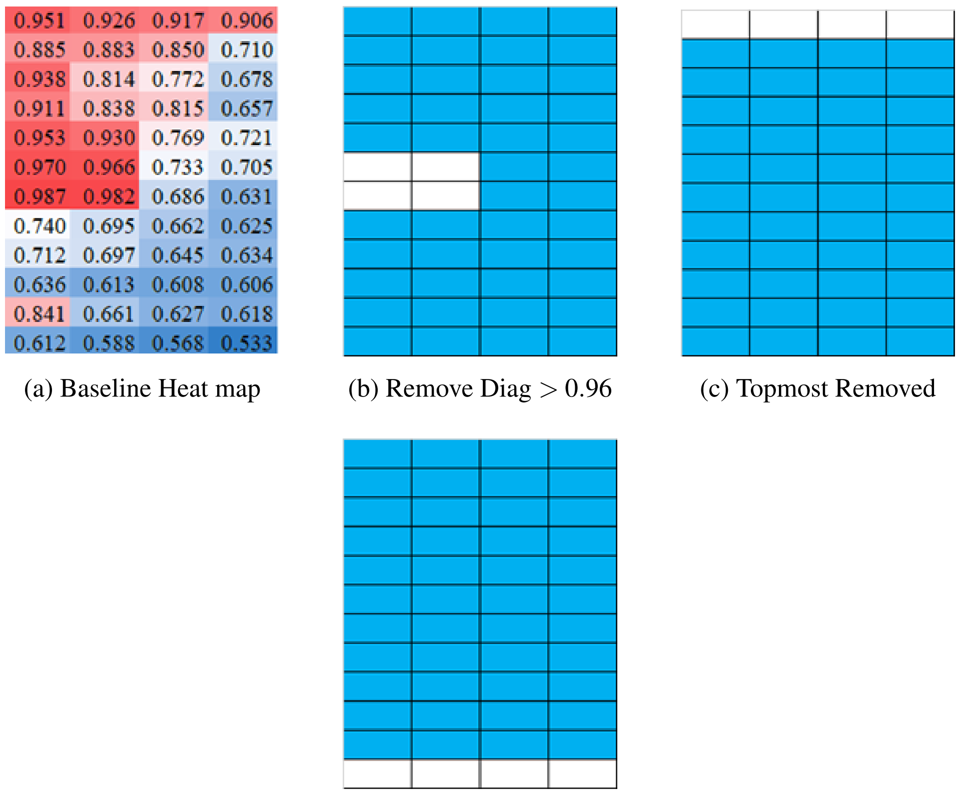
Table 6.3 shows the CERs of each model. We observed that if the attention heads with diagonality larger than 0.95 are removed and the remaining architecture is trained from scratch (re-initialising the weights), there will be no performance drop. There are minor performance drops if we remove the attention heads whose diagonality is

Table 6.3: CERs of trained Transformer models with different encoder architectures.

Removal Strategy	# Encoder Heads	eval92 (CER)	dev 93 (CER)
Baseline	48	3.5	4.6
Diagonality>0.95	42	3.4	4.6
Diagonality>0.90	36	3.5	4.7
Diagonality>0.85	33	3.6	4.7
Diagonality>0.80	29	3.7	4.7

larger than 0.90. However, in this situation, the total number of attention heads is only 3/4 compared to the total number of attention heads of the baseline. As we drop more attention heads, the performance of the models becomes slightly worse. This is expected, since small diagonality indicates the global view and the self-attention mechanism are still useful.

We observe that although the topmost layer of the encoder has the largest averaged diagonality, the individual attention heads which have the largest averaged diagonality are not necessarily in the upper most layer. Thus, we further investigate removing individual heads that have the top-four largest diagonality. Figure 6.3 shows the corresponding architectures and Table 6.4 shows the CERs of each removal strategy. Surprisingly, removing the attention heads with the top-four diagonality leads to inferior performance. However, replacing the topmost self-attention layer with a feed-forward layer, which is equivalent to removing all the attention heads in the topmost layer, gives better performance compared to the baseline. We also test dropping all the four attention heads in the lowest layer and it results in higher CERs. Thus, the position of the attention heads may influence on whether the heads can be removed (with minor effect on performance).



0.951	0.926	0.917	0.906
0.885	0.883	0.850	0.710
0.938	0.814	0.772	0.678
0.911	0.838	0.815	0.657
0.953	0.930	0.769	0.721
0.970	0.966	0.733	0.705
0.987	0.982	0.686	0.631
0.740	0.695	0.662	0.625
0.712	0.697	0.645	0.634
0.636	0.613	0.608	0.606
0.841	0.661	0.627	0.618
0.612	0.588	0.568	0.533

(a) Baseline Heat map

(b) Remove Diag > 0.96

(c) Topmost Removed

(d) Lowest Removed

Figure 6.3: The tested encoder architectures. The baseline has 12 self-attention layers. Each non-white cell represents an attention head. Each white cell denotes there is no attention head. When all the attention heads of a layer are removed (four white cells), the feed-forward component of that layer is preserved. (a) heatmap of the averaged diagonality of each attention head. (b-d) structures where some attention heads are removed.

6.6 Stochastic Attention Head Removal

In the previous section, we observed that training some architectures with reduced number of attention heads from scratch could lead to better accuracies. However, the search for the best architecture is time prohibitive . To address this, we propose to randomly remove attention heads during training and keep all heads at the test time. The proposed method can be viewed as training different architectures and using the ensemble of these trained architectures as the final model at the test time. When processing a training example, each attention head has a probability q of being removed, where q is set as a hyper-parameter. If a head is kept, then the output of that head is scaled by $\frac{1}{1-q}$:

$$\mathrm{A}_i(\mathbf{X}^{\mathrm{Q}}, \mathbf{X}^{\mathrm{K}}, \mathbf{X}^{\mathrm{V}}) = \frac{1}{1-q} \times \mathrm{softmax}(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d^{\mathrm{K}}}}) \mathbf{V}_i \tag{6.2}$$

At the test time, the scale factor $\frac{1}{1-q}$ is removed and all the attention heads are always present. Thus, the expected output of the attention head is the same during training and testing. During training, if all the attention heads of a layer are removed for an utterance, then that layer becomes a feed-forward layer. The stochastic attention head removal strategy is applied for both the encoder and the decoder. We apply the proposed method in training both the Transformer models and the Conformer models. Considering both the encoder and the decoder has 48 attention heads and in the experiments of static attention head removal some encoder architectures with $48-4=44$ heads have yielded improved accuracies, we test removal probability values starting from $\frac{4}{48}$.

Table 6.4: CERs of trained Transformer models with different encoder architectures.

Removal Strategy	# Encoder Heads	eval92 (CER)	dev 93 (CER)
Baseline	48	3.5	4.6
Diagonality>0.96	44	3.7	4.6
Topmost Heads Removed	44	3.4	4.5
Lowest Heads Removed	44	3.7	4.6

6.6.1 Results and Discussion

Table 6.5 summarises the experimental results on WSJ. The attention head removal probability $\frac{8}{48}$ consistently gives the best results on eval92, and it yields almost identical CERs on dev93 compared to the baseline. The removal probability $\frac{10}{48}$ has CERs

Table 6.5: CERs on WSJ for stochastic head removal.

	eval92/dev93 (CER%)		
removal probability	seed1	seed 2	seed 3
0/48 (baseline)	3.5/4.6	3.6/**4.5**	3.5/4.6
4/48	3.6/**4.5**	3.5/4.6	3.5/**4.4**
6/48	**3.4/4.5**	3.5/4.6	**3.4**/4.7
8/48	**3.4**/4.6	**3.4**/4.6	**3.4**/4.6
10/48	3.5/4.6	3.5/**4.5**	**3.4**/4.5

Table 6.6: WERs on SWBD for Transformer models.

	SWBD/Callhome (WER%)		
Removal Probability	seed 1	seed 2	seed 3
Transformer Models			
0/48 (baseline)	9.0/18.1	9.1/18.2	9.1/17.9
4/48	8.9/17.5	8.7/17.6	-
6/48	**8.6/17.2**	**8.7/16.9**	**8.7/16.8**
8/48	8.8/17.6	8.9/17.2	-
Transformer (Karita et al., 2019)			9.0/18.1
Very deep self-attention (Pham et al., 2019)			10.4/18.6
Multi-stride self-attention (Han et al., 2019)			9.1/-

which are never worse than the baseline. Thus, we conclude the proposed stochastic attention head removal strategy consistently improves the model's performance.

We further test the proposed training method on larger and more challenging datasets. Table 6.6 shows the results of applying the proposed method to train Transformers on SWBD. We run the experiments with three different random seeds, and have obtained consistently reduced word error rates (WERs) with all the seeds. To further show our method has statistically significant gains, we apply Matched Pairs Sentence-Segment Word Error significance tests (mapsswe) (Gillick and Cox, 1989) for the best Transformer models. On both SWBD/Callhome test sets, $p < 0.001$.

We also use our method to train Conformer models on SWBD. Noticing that in the Transformer experiments the probability values which lead to the best models are within the range of 0.1 to 0.2, we test using 0.1 or 0.2 as the removal probability. That is, we test setting the removal probability without considering the total number of at-

Table 6.7: WERs on SWBD for Conformer models. * indicates the number is the average of 5 runs. In these five runs, the WERs on SWBD and Callhome ranges from 6.7 to 6.8 and from 13.5 to 13.9, respectively.

Removal Probability	SWBD/Callhome (WER%)
Conformer Models	
0.0 (baseline)	8.4/17.1
0.1	8.2/**16.4**
0.2	**8.1/16.4**
+Speed Perturbation + RNNLM	
0.0 (baseline)	6.9/14.0
0.1	**6.4** /14.0
0.2	6.7*/**13.7***
Conformer (Guo et al., 2020)	7.1/15.0

tention heads. In addition, we augment SWBD by speed perturbation (Ko et al., 2015) with ratio 0.9 and 1.1 and decode with a recurrent neural network language model (RNNLM) (Hori et al., 2017). Table 6.7 shows that our method is effective in this set of experiments. We train Conformer models using a fixed random seed. The only random factor is how the attention heads are removed in each training update and the performance of the baseline is constant. We apply a two-tailed t-test to examine if the mean WERs of the five runs of the conformer models trained with removal probability 0.2 are significantly different from the baseline Conformer. On SWBD $p < 0.003$ and on Callhome $p < 0.02$, which shows the gains are statistically significant. To the best of our knowledge, we have achieved the SOTA for end-to-end Transformer based models on SWBD.

Table 6.8 shows the results on AISHELL. We have applied mapsswe. On the Dev set, $p < 0.001$ for both the Transformer and Conformer models. On the test set, $p = 0.030$ for the Transformer and $p = 0.023$ for the Conformer. We notice there is a performance gap between our Conformer models and the previous work (Guo et al., 2020). This is because of different training configurations. We did not further tune our Conformer models on AISHELL due to the large number of experiments and we have indeed obtained statistically significant gains. Thus, we conclude on all the datasets, our method has offered statistically significant performance gains.

We use AMI to test our method on a more difficult ASR task. We use the individual

Table 6.8: CERs on AISHELL of stochastic head removal.

Removal Probability	Dev/Test (CER%)
Transformer Models	
0.0 + speed perturbation + RNNLM	6.0/6.5
0.1 + speed perturbation + RNNLM	**5.8/6.3**
0.2 + speed perturbation + RNNLM	**5.8/6.3**
Transformer (Karita et al., 2019)	6.0/6.7
+ 5,000 hours pretrain (Jiang et al., 2020)	-/6.26
Memory equipped self-attention (Gao et al., 2020)	5.74/6.46
Conformer Models	
0.0 + speed perturbation + RNNLM	5.3/6.0
0.1 + speed perturbation + RNNLM	**5.2/5.8**
Conformer (Guo et al., 2020)	4.4/4.7

headset microphone (IHM) setup. Table 6.9 shows our method has outperformed the baselines. Our method has achieved the SOTA for Transformer based models on AMI. We also applied mapsswe and $p < 0.001$ for both Transformer and Conformer models on both dev and test sets.

We notice that the performance of our method varies among different removal probabilities. However, the probabilities between 0.1 and 0.2 have given the best accuracies. Also, the improvements are upon strong or even SOTA baselines with data augmentation and RNN-based LM.

We further analyse the effect of the proposed method. We have plotted the training loss of the baseline Conformer and the Conformer trained with removal probability 0.1 on SWBD. Figure 6.4 (a) shows the train loss curves. The proposed method has higher train loss in each epoch. This is as excepted since in each training update only a fraction of the network is trained. We also compute the averaged similarity of the attention matrices of each attention head pair in each layer. We use the averaged cosine similarity of rows of the same indices as the similarity of attention matrices since each row vector shows how the attention weights are distributed to encode the input at the row index. Figure 6.4 (b) show that with the random removal, the similarities between attention heads increase. Thus, it indicates that the proposed method forces each attention head to extract the most essential patterns while allowing different heads to encode individual additional information. The baseline model has a smaller similarity

Table 6.9: WERs on AMI for stochastic head removal.

Removal Probability	dev/test (WER %)
Transformer Models	
0.0 (baseline)	24.9/25.8
0.1	**24.2/24.6**
0.2	24.5/24.9
Conformer Models	
0.0 (baseline)	24.9/25.8
0.1	**24.1/24.2**
2D Conv (Oglic et al., 2020)	24.9/24.6
TDNN (Oglic et al., 2020)	25.3/26.0

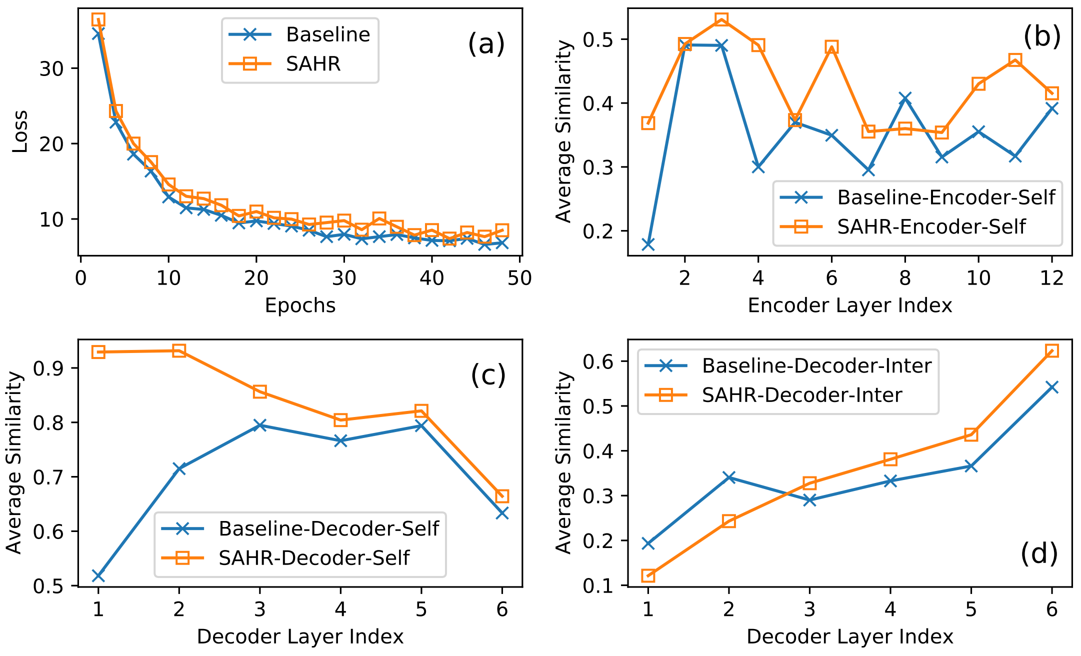


Figure 6.4: The train loss and the averaged similarity between attention heads of the baseline Conformer model and the Conformer model trained with stochastic attention head removal (SAHR) probability 0.1. ”decoder-inter” denotes the MHA component that interacts between the encoder and the decoder.

between attention heads, further indicating each layer relies on some attention heads to learn the most useful features while other heads encode unessential information. Therefore, the proposed method more effectively utilises the representation powers of the multi-head attention and gives improved accuracies.

6.7 Summary

In this chapter, we studied individual attention heads in Transformer based ASR models. We observed some attention heads are redundant – they only focus on local information. Also, removing some attention heads and retraining the model from scratch could lead to performance gains. Instead of searching for the best structure with reduced number of attention heads (which is time prohibitive), we proposed to randomly remove attention heads during training and keeping all the attention heads at testing. The proposed method also encourages each attention head to learn the key patterns which share some similarities across all the heads in the same layer. The stochastic attention head removal method has offered statistically significant improvements for Transformer and Conformer models on datasets of different ASR scenarios.

Since our proposed SAHR can be viewed as a method in training self-attention based models in general, this method can also be applied to other related tasks, such as machine translation or speech translation. However, these further applications are beyond the scope of this thesis.

Chapter 7

Top-down Level-wise Neural Networks Training

7.1 Introduction

In the previous chapters (Chapter 3-6), we have studied attentional encoder-decoder models. In this chapter, we present novel top-down level-wise neural network training methods. That is, we propose to train neural networks from the upper layers (near the output) to the lower layers (close to the input) in a cascaded way.

The lower layers of a deep neural network (DNN) can be interpreted as a feature extractor while the upper layers can be viewed as a classifier. The feature extractor learns useful features from the data (Krizhevsky et al., 2012; Bengio, 2012; Yosinski et al., 2014). Transfer learning exploits this property – if the underlying features of two datasets are similar, then the feature extractor which is trained on the base dataset can be transferred to the target dataset, with the classifier trained on the target dataset. For speech recognition, transfer learning has been widely employed in training ASR models for low-resource languages – Swietojanski et al. (2012) applied unsupervised restricted Boltzmann machine (RBM) (Smolensky, 1986) pre-training using different languages; Thomas et al. (2012), Ghoshal et al. (2013) and Huang et al. (2013) employed shared feature extractors across different languages; Tong et al. (2017) and Cho et al. (2018) explored building sequence-to-sequence ASR models for low-resource language through transfer learning. The motivation of these previous works is that the low-level features of different languages are similar while the output units are distinct.

Although the feature extractor can be transferred across different datasets, surprisingly, it is not transferable within the same dataset – Yosinski et al. (2014) found that retraining the classifier based on a frozen trained feature extractor using the same dataset usually results in a performance drop.

In this chapter, in contrast to training neural networks in a bottom-up way, we investigate training neural networks from the upper layers to the lower layers in a **top-down** manner. Since we train the network from the upper layers to the lower layers rather than fixing one part and train another part in iteration, the proposed method cannot be categorised as a coordinate descent or an alternating minimization algorithm. The upper layers and the lower layers can be trained using different datasets or the same dataset. Figure 7.1 shows the process of the proposed top-down training. We demonstrate that analogously to general feature, there are also general classifiers. That is, 1) while general feature extractors extract relevant features, general classifiers define suitable classification boundaries which better reflect the relationships between different classes; 2) when transferred to unseen data, general feature extractors/classifiers should lead to models with good generalisation. We show that general classifiers can be transferred across **different datasets** experimentally. In addition, we empirically show unlike feature extractors, classifiers can be transferable **within the same dataset** – retraining the feature extractor based on the frozen general classifier on the same dataset can lead to performance gains.

The top-down training across different datasets is suitable for training noise-invariant feature extractor. If a model is trained with a large amount of clean data, then it is relatively easy for the feature extractor to learn extracting general features, and thus the features which fit the classifier are close to the underlying patterns. Then, on the noisy dataset, if we train the feature extractor with respect to the classifier which is trained on large amount of clean data, we force the feature extractor to provide features which fit the trained clean classifier. As a result, the feature extractor is constrained to provide noise-invariant features. Experimentally, for CTC models, we train the feature extractor on CHiME-4 based on the classifier trained on WSJ. We use Aurora-4 (Pearce, 2002) test sets to test the model's performance on unseen noisy conditions. We observe the model trained with the proposed method has significantly lower CERs on all 14 Aurora-4 test sets, indicating the feature extractor is indeed noise-invariant.

The top-down training within the same dataset prevents the classifier from (further) overfitting. Within the same dataset, for the conventional training where all the layers are trained jointly, we empirically find that the classifier becomes general then overfits as the training continues. Thus, freezing the classifier before it begins to overfit, and only retraining the feature extractor is beneficial – it prevents training the feature ex-

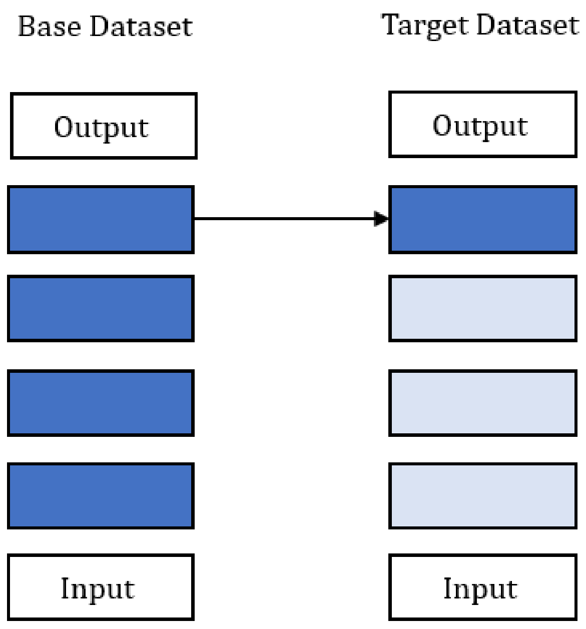


Figure 7.1: An illustration of the top-down training. A base model is trained on the base dataset. Then, on the target dataset, the feature extractor is trained with the trained base classifier (we can group arbitrary many upper layers as the classifier; however, the feature extractor should at least have one layer). The deep blue square indicates the weights of the layer in the target model are from the weights of the layer in the base model. The shallow blue squares indicate the weights of these layers are optionally reinitialised. The target dataset and the base dataset can be either the same or different. When the target dataset and the base dataset are different, we either freeze the classifier or fine-tune it with a small learning rate. If the target dataset and the base dataset are the same, we always freeze the classifier.

tractor with respect to an overfitting classifier. We experimentally show the proposed method is effective in training CTC, CTC-attention and Transformer models for ASR; VGG (Simonyan and Zisserman, 2015) and ResNet (He et al., 2016) models for image classification; AWD-LSTM (Merity et al., 2018) for language modelling.

In summary, this chapter presents three novel contributions:

Demonstration of the existence of general classifiers;

A Top-down training algorithm across different datasets;

A Top-down training algorithm within the same dataset.

7.2 Related Work

The proposed top-down training method can be viewed from different aspects. Thus, it is related to previous works of different fields. We review these previous works of different areas in this section.

7.2.1 Transfer Learning

Conventional transfer learning transfers the feature extractor across different datasets. For ASR, this learning paradigm is widely used in training models for low-resource languages (Thomas et al., 2012; Swietojanski et al., 2012; Ghoshal et al., 2013; Huang et al., 2013; Tong et al., 2017; Cho et al., 2018), since the low-level features of different languages are generally similar. However, this method is not suitable in training a noise-invariant feature extractor. Although the underlying patterns should be invariant to the noise conditions, it is not appropriate to transfer the feature extractor from the clean dataset to the noisy dataset owing to the mismatch of acoustic conditions. In contrast, our proposed method is suitable for training noise-invariant features. Also, transferring the feature extractor within the same dataset (i.e., retraining the classifier with the trained feature extractor) usually leads to performance drops (Yosinski et al., 2014), while our top-down training method also leads to performance gains when applied within the same dataset.

7.2.2 Bottom-up Layer-wise Training

Top-down training within the same dataset trains neural networks from the upper layers to the lower layers in a cascade manner, and thus can be viewed as a layer-wise training method. Layer-wise training without a joint fine-tuning stage has been investigated by a number of researchers (Hettinger et al., 2017; Zhao et al., 2018; Mostafa et al., 2018; Malach and Shalev-Shwartz, 2018; Marquez et al., 2018; Belilovsky et al., 2019; Nøkland and Eidnes, 2019). However, these training methods build DNNs in a bottom-up manner. In general, the reported experimental results do not indicate that these training methods surpass conventional joint training.

Belilovsky et al. (2019) introduced auxiliary networks trained using layer-wise training, demonstrating that it can generate models which outperform Alexnet (Krizhevsky et al., 2012) and VGG on ImageNet (Deng et al., 2009). However, when using networks with the same auxiliary architecture, conventional joint training outperforms layer-wise training.

Nøkland and Eidnes (2019) presented a layer-wise training approach which interpolates a similarity loss with the cross-entropy loss. They found that in this case layer-wise training surpasses joint training when using VGG-like models on CIFAR-10. However, this training method is not useful for training networks with residual connections. In contrast, our top-down training methods presented here surpasses joint training for both VGG and ResNet on CIFAR-10. Our training method also leads to better accuracies of Transformer based ASR models on SWBD, which further shows it is effective for training networks with residual connections.

7.2.3 Dynamically Changed Learning Rate

In our top-down training method, we either fine-tune the trained classifier or freeze it (in this case the learning rate can be viewed as 0). In gradient based optimization, the learning rate can either be dynamically tuned by the user or be tuned by optimization algorithms (Qian, 1999; Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015; Dozat, 2016). Our experiments show that the top-down procedure is compatible with popular methods of changing the learning rate dynamically.

7.2.4 Reduction of Trainable Network Size

During training, both the proposed top-down methods and dropout (Srivastava et al., 2014) reduce the size of the trainable part of neural network. Dropout can be viewed as reducing the effective width of the network in a random manner, while our method reduces the effective depth of the network by following a fixed schedule. When these two methods are combined, during training narrower and shallower sub-networks are trained, while during testing, the model has the representation power of a wide and deep network. In our experiments, the combination of these two methods yields significant performance gains.

Hoffer et al. (2018) found training the lower layers with respect to a frozen softmax layer with a random orthogonal weight matrix does not give big accuracy drops on image classification, but yields inferior results on language modelling – the orthogonal matrix does not capture correlations between classes. Furthermore, when the numbers of output labels are not equal to the numbers of the hidden units, it is impossible to construct an orthogonal weight matrix. In contrast, our method reduces the perplexity of the trained language model and has no constraint on the shape of the weight matrices.

7.3 General Classifiers

In this section, we firstly consider general classifiers from a conceptual viewpoint. Then, to experimentally show if a classifier is more general, we make an analogy to the concept of general feature extractor: if the classifier is more general, then training the feature extractor with respect to it on unseen data should lead to better generalisation compared to a classifier which is less general. Also, analogously to general feature extractor, we hypothesise that with more training data, the trained classifier becomes more general and equivalently has higher quality. We conduct a series of CTC experiments to verify our hypothesis.

7.3.1 A Conceptual Illustration

Although we can group arbitrary many layers as the classifier, for simplicity, we firstly view the softmax layer, which is usually the uppermost layer of neural networks, as the classifier. We consider the case where the weights of the softmax layer are frozen.

Suppose there are k output classes $(c_1, c_2, \cdots, c_k)$ for the network. If a training sample belongs to class c_i, then the training label for the neural network is a one-hot vector $\mathbf{y}$ where the i^{th} element y_i is 1 and all other elements are zeros. For the neural network, suppose there are n hidden units $\mathbf{h} = (h_1, h_2, \cdots, h_n)$ below the softmax layer. Then, the weight matrix of the softmax layer is a $k \times n$ matrix. For each class i, there is a corresponding weight vector $\mathbf{w}_i$ in the weight matrix. The output of the softmax layer is $(o_1, o_2, \cdots, o_k)$ and for each o_i:

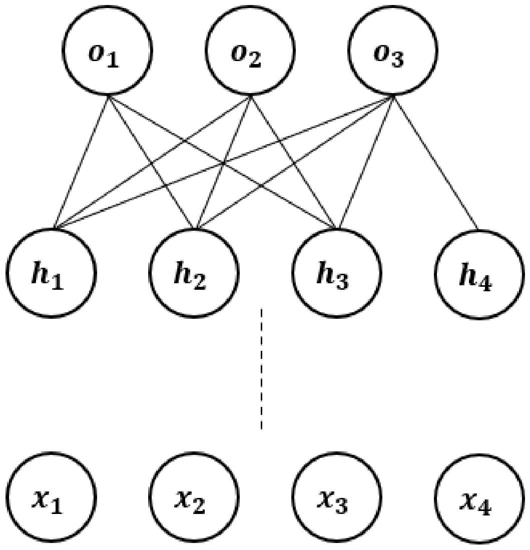


Figure 7.2: An illustration of a neural network. The final layer is a softmax layer and $\mathbf{o} = (o_1, o_2, o_3)$ is the output of the neural network. $\mathbf{h} = (h_1, h_2, h_3, h_4)$ denotes the final hidden representation before the softmax layer. $\mathbf{x} = (x_1, x_2, x_3, x_4)$ is the input.

$$o_i = \frac{\exp(s_i)}{\sum_{j=1}^{k} \exp(s_j)} \tag{7.1}$$

$$s_i = \mathbf{w}_i \cdot \mathbf{h} \tag{7.2}$$

The dot product $s_i = \mathbf{w}_i \cdot \mathbf{h}$ decides the 'score' (logit) of class i. Since all the weights of the softmax layer is frozen, each $\mathbf{w}_i$ can be viewed as the centre or the proxy of each class. In this case, the dot product computes the similarities between the vector of hidden units $\mathbf{h}$ and the centre of each class. During testing, if $\mathbf{h}$ is closest to $\mathbf{w}_i$, then o_i will be the maximum among $(o_1, o_2, \cdots, o_k)$ and the output label of the neural network will be c_i. Figure 7.2 shows an illustration of such a neural network.

It is easy to show that some weight matrices of the softmax layer will lead to bad performance (e.g., for the weight matrix, $\mathbf{w}_1 = \mathbf{w}_2 = \cdots = \mathbf{w}_k$). We argue that a general classifier should capture the underlying relationships between the classes well and thus training with respect to it will lead to a feature extractor which learns the general underlying patterns. Considering training with the frozen softmax layer. Suppose a training example belongs to classes i and the loss function L is cross-entropy. Then, during backpropagation:

$$\frac{\partial L}{\partial \mathbf{h}} = \sum_{j=1}^{k} (o_j - y_j)\mathbf{w}_j \tag{7.3}$$

Thus, the final hidden vector **h**, or the final hidden representation of class i, will be pushed towards $\mathbf{w}_i$ but away from all other $\mathbf{w}_j$. Below is a toy example where the weight vectors of the classifier do not capture the relationship well, and training with respect to such a classifier will lead to poor generalisation. Suppose we have three classes for the neural network to classify – apple, orange and car. Suppose the norms of $\mathbf{w}_{apple}$, $\mathbf{w}_{orange}$ and $\mathbf{w}_{car}$ are the same and $\mathbf{w}_{apple} \cdot \mathbf{w}_{car} > \mathbf{w}_{apple} \cdot \mathbf{w}_{orange}$. During training, the final hidden representation of each class will be pushed to the corresponding weight vector of each class. Thus, in this example, the final hidden representations of apples will be more similar (closer) to cars rather than oranges, which indicates the learned features are not general. Therefore, the neural network will have poor generalisation.

Arguably, it is possible that before the final layer, the general patterns which reflect the relationship between different classes are already learned at some lower layer. Then, the layers above that layer transfer the underlying patterns to features which fit the softmax layer. However, the general features must be learned at some layer of the network. We can always group the layers from where the general features are learned to the uppermost layer as the classifier, and demonstrate a general classifier is necessary for good generalisation in a similar way as the case where we view the softmax layer as the classifier.

It may seem beneficial to use a softmax layer with a frozen orthogonal weight matrix to force the network learn well-separated features. However, previous works (Hoffer et al., 2018) have shown this approach usually leads to worse performance.

Also, it is tempting to construct handcrafted classifiers which may reflect people's prior knowledge about the classes. However, when the relationships between classes are not straight-forward and when the network's structure is complicated (we can group arbitrary many layers as the classifier as long as there is one layer left acts as the feature extractor), it is not feasible to build such a classifier by hand.

7.3.2 Experimental Justifications

We have discussed general classifiers conceptually. However, it is difficult, if not impossible, to find the weight vectors which define the relationships between classes in the most rigorous way, and using these vectors to construct the weight matrix for the ideal general classifier. Therefore, to justify our claim that there are classifiers that are more general than other classifiers, we make an analogy to general feature extractors: if a classifier is more general, then training feature extractors with respect to it on unseen data should lead to better generalisation compared to a classifier which is less general. We hypothesise that with more training data, the trained classifier becomes more general and has higher quality. To verify this, we conduct a series of end-to-end speech recognition experiments.

Experimental setup. We use Kaldi to extract 40-dimension mel-scale filterbank features with three pitch features, and we use the toolkit ESPnet to implement BLSTM CTC models. All CTC models have the same architecture, similar to that used by Kim et al. (2017). The network has 4 BLSTM layers. On top of each BLSTM layer is a projection layer, which is an affine layer with tanh activation functions. The top-most layer is a softmax layer. We refer to this model architecture as **(BLSTM-projection)**$\times$**4–softmax**. Except for the softmax layer, all BLSTM and projection layers has 320 units. There are 50 output labels: 26 characters, apostrophe, period, dash, space, noise, sos/eos tokens and some other special tokens. We used the Adadelta optimization algorithm and gradient clipping. The training stops if after 5 epochs there is no improvement upon the best validation loss. We divide the WSJ training set into 5%, 10%, 20%, 40% and 80% size subsets, leaving 20% unseen training data. We train CTC models on these subsets following the same experimental setup. We group the softmax layer and the topmost BLSTM layer, i.e. **(BLSTM-projection)–softmax**, as the classifier.

As shown in Table 7.1, with more training data, the model achieves better ASR performance. To test if the classifiers from each trained model become more general as the size of the training subsets increases, on the unseen 20% training set, we run experiments of training the feature extractor with these frozen trained classifiers (the feature extractor is always reinitialised). Figure 7.3 shows on the unseen training data, training with respect to the frozen classifier trained on larger subsets consistently yields lower character error rates (CERs). Thus, we conclude that the classifiers trained on larger subsets are indeed more general and have higher quality, and training the feature extractor with them leads to better generalisation.

We next show that the classifier trained with more clean data is also more general than the classifier trained with a smaller amount of noisy data. We train a model using the full WSJ si284 dataset, and refer the classifier of that model as the “clean classifier”. We then freeze the clean classifier and retrain the feature extractor on CHiME-4. We

Table 7.1: CERs of models trained on si284 subsets.

Training set	dev93/eval92 (CER)
5% train si284	34.4/32.9
10% train si284	28.3/26.1
20% train si284	22.5/19.9
40% train si284	17.5/14.7
80% train si284	13.2/10.6

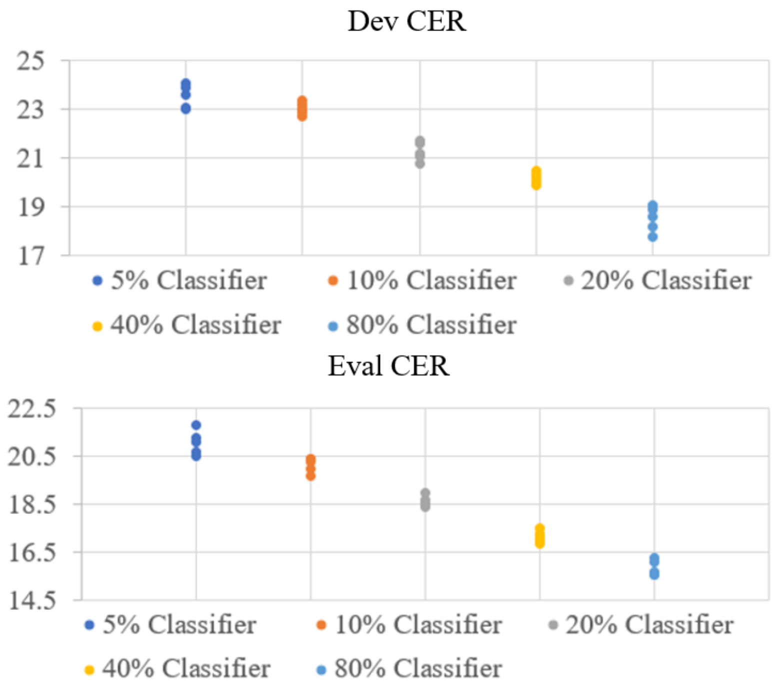
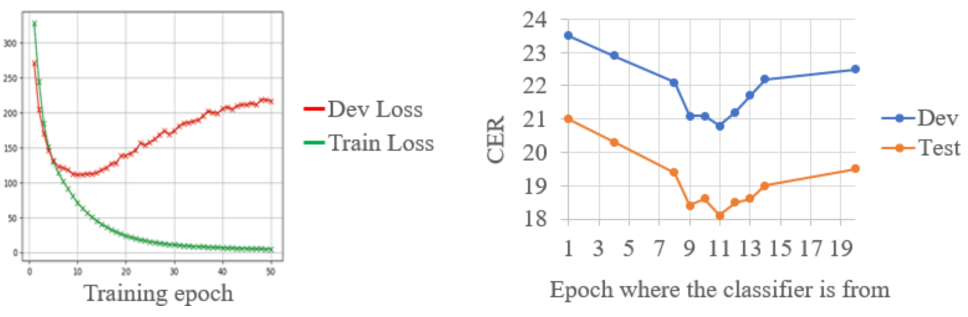


Figure 7.3: The CERs of models trained on the 20% unseen data with frozen classifiers from trained models on different si284 subsets. The experiments are repeated 5 times with different random seeds.

Table 7.2: CERs of the model where all the layers are jointly trained on CHiME-4 and the model whose classifier is trained on WSJ.

Training method	dt05_multi/et05_real(CER)
Joint Train	29.0/38.7
Frozen Clean Classifier	25.7/36.0



(a) Loss on 80% joint training (b) CERs on 20% unseen

Figure 7.4: Train/dev loss for the joint training on the 80% subset and CERs of the models trained on the 20% unseen data with frozen classifiers taken from the epochs of the joint training.

compare the model trained with the clean classifier with a baseline CTC model where all the layers are jointly trained on CHiME-4. Table 7.2 shows that compared to the jointly trained baseline model, training with the frozen clean classifier leads to lower CERs.

We have experimentally shown that we can obtain classifiers which are more general by using more training data or increasing the quality of the training data. We further explore in the joint training within the same dataset, whether the classifier becomes more general as the training proceeds. We train the CTC model on the 80% subset for a sufficiently large number of epochs. We then take the classifiers from the (jointly) learned models at different epochs and retrain the feature extractor (from scratch) on the 20% unseen data using these frozen classifiers. Figure 7.4 shows that on the 20% unseen data, the model trained with the frozen classifier which is from epoch 11 of the joint training on the 80% subset gives the lowest CERs. In the joint training on the 80% subset, the model around epoch 11 also gives the lowest development loss. On the 20% unseen data, the models trained with frozen classifiers which are from later epochs of the joint training on the 80% subset have increasing CERs. Also, in the joint training on the 80% subset, the development loss increases after epoch 11. Thus, the classifier firstly becomes general (low CERs when transferred to the unseen data), and then becomes overfitting, leading to worse generalisation (high CERs when transferred to the unseen data).

In summary, in this section, we have empirically shown that the classifier becomes more general as the amount of training data increases; the classifier becomes more general as the quality of training data increases and within the same dataset, the classifier becomes more general then becomes overfitting as the joint training proceeds.

7.4 Top-down Training Algorithms

Based on the observations in the previous section, in this section, we develop top-down training algorithms across different datasets and within the same dataset.

7.4.1 Top-down Training across Different Datasets

When applying top-down training across different datasets, it is suitable to use clean data as the base dataset and the noisy data as the target dataset. Thus, this method can be viewed as a transfer learning from the clean data to the noisy data. Specifically, we propose to firstly train the model on the clean dataset to obtain the clean general classifier. We then train the model on the noisy dataset. While training on the noisy data, the clean classifier is either frozen or tuned with a small learning rate. The feature extractor is either reinitialised or initialised with the weights of the clean feature extractor, as if the training on the clean data is considered as a pre-training stage. The feature extractor is trained with the normal learning rate without any learning rate rescaling. In the proposed method, the feature extractor is constrained to learn features that match the clean classifier. Since the features fit the clean classifier reflect more about the underlying patterns, the clean classifier helps the feature extractor to learn the noise invariant patterns from the noisy data. Algorithm 1 shows the the training procedure.

In Algorithm 1, we do not only consider the output softmax layer as the classifier. We also view the grouping of the softmax layer and several other upper layers as the classifier. With more layers, the classifier is more powerful and may ease the burden of learning features from noisy data for the feature extractor. However, more layers for the classifier also means fewer layers for the feature extractor. It also makes the

Algorithm 1 Top-down Transfer Learning From Clean Data to Noisy Data

```
Input: An $n$ layer network $M$. A clean training set. A noisy training set. A noisy validation set.
Train $M$ on the clean training set.
$e \leftarrow$ validation error of $M$ on the noisy validation set
$N \leftarrow M$
for $i = 1$ to $n - 1$ do
    $M' \leftarrow M$
    Optionally reinitialise bottom $n - i$ layers (near input) of $M'$
    Train $M'$ using the noisy dataset; the top $i$ layers are either frozen or fine-tuned with a reduced learning rate.
    $e' \leftarrow$ validation error of $M'$ on the noisy validation set
    if $e' > e$ then
        BREAK
    end if
    $N \leftarrow M'$
    $e \leftarrow e'$
end for
Output: A trained n-layer network $N$.
```

feature extractor less powerful and less flexible. Thus, with too few layers, the feature extractor may not have the capacity to fit the classifier. Therefore, we use a validation set to decide how many layers should we group as the clean classifier.

7.4.2 Top-down Training within the Same Dataset

Since in the joint training within the same dataset, the classifier becomes more general before overfits, we develop a top-down training algorithm within the same dataset, which can be viewed as a process of searching for high-quality classifiers. Since the training is within the same dataset, the proposed method can be viewed as a top-down, layer-wise training algorithm. The proposed algorithm searches among models trained by conventional joint training along the epoch dimension and the layer dimension (layer index). Algorithm 2 shows the top-down layer-wise training algorithm.

Algorithm 2 Greedy Layer-wise Top-down Training

```
Input: A trained $n$ layer network $M$
$e \leftarrow$ validation error of $M$
for $i = 1$ to $n - 1$ do
    $M' \leftarrow M$
    Freeze top $i$ layers (near output) of $M'$
    Reinitialise bottom $n - i$ layers of $M'$
    Retrain bottom $n - i$ layers of $M'$
    $e' \leftarrow$ validation error of $M'$
    if $e' > e$ then
        BREAK
    end if
    $M \leftarrow M'$
    $e \leftarrow e'$
end for
Output: A retrained n-layer network $M$
```

Searching the epoch dimension is useful since at the beginning of the training the classifier is under-trained and at the end of the training the classifier tends to be overfitting. We denote the model at epoch p as M^p. We search for the best classifier through all M^p.

Searching the layer dimension is essential since arbitrarily many upper layers can be grouped and viewed as the classifier (the feature extractor should at least have one layer). For a n-layer neural network M which is trained by joint training, freezing the top i layers of it then reinitialising and retraining the bottom $n-i$ layers generates a new model $M_{i,n-i}$. For the model $M_{i,n-i}$, we can add newly retrained lower layers to the classifier and retrain the remaining lower layers. $M_{i,j,n-i-j}$ denotes a model trained by freezing the top $i+j$ layers of model $M_{i,n-i}$ followed by retraining the lower $n-i-j$ layers. This corresponds to a search of ordered sequences of natural numbers $(i,j,k,\cdots,r)$ such that $i+j+k+\cdots+r=n$; for example, for a network M with three layers, the search would be across $M_{1,1,1}$, $M_{1,2}$ and $M_{2,1}$ (e.g., $M_{1,1,1}$ indicates training all the layers jointly;then freezing the topmost layer and retraining the bottom two layers; then freezing the top two layers and retraining the bottom most layer. $M_{2,1}$ denotes training all layers jointly; then freezing the topmost two layers and retraining the bottom most layer).

As the complete search along these two dimensions is expensive, we use a greedy search algorithm for layer-wise training (Algorithm 2), which uses a converged model instead of doing a complete search along the epoch dimension. For the layer dimension, we freeze layers from top to bottom in a layer-by-layer manner (i.e. we only consider $M_{1,n-1}, M_{1,1,n-2}, \cdots M_{1,1,\cdots,1}$). We also use the validation set to halt the search. The complexity of this algorithm is $O(n)$, where n denotes the number of layers of the network. We omit the complexity of training the network since it is independent of the top-down layer-wise training algorithm.

Unlike top-down transfer learning, in top-down layer-wise training, the trained classifier is always frozen – further fine-tuning is not allowed. In the top-down transfer learning, fine-tuning the classifier enables it to adjust the mismatched acoustic environment. However, the purpose of top-down layer-wise training is to prevent the classifier from overfitting. Fine-tuning the classifier has the risk of making it overfit.

7.5 Experiments of Top-down Transfer Learning

We apply the proposed method to train CTC models. We use WSJ si284 as the clean training data and CHiME-4 as the noisy training data. In our experiments, for CHiME-4, the training data is the single channel simulated noisy data and the real noisy data from all channels. We use dt05_multi_isolated_1ch_track as the development set and et05_real_isolated_1ch_track as the evaluation set. The noise conditions in CHiME-4 training sets and CHiME-4 test sets match. To test the performance of the models in

Table 7.3: The CNN architecture for the CNN-BLSTM model.

	in_channel	out_channel	kernel	stride
conv	1	64	3×3	1
conv	64	64	3×3	1
maxpool			2×2	2
conv	64	128	3×3	1
conv	128	128	3×3	1
maxpool			2×2	2

unseen noise conditions, we also use all 14 test sets in the Aurora-4 corpus as additional test sets.

We use Kaldi to extract 40-dimension mel-scale filterbank features with three pitch features, and the ESPnet toolkit to build convolutional neutral networks (CNNs) – bidirectional long short-term memory (BLSTM) models. The architecture of the CNNs is in Table 7.3. There are four BLSTM layers on top of the CNNs. Each BLSTM layer is followed by a linear layer with tanh activation. All the BLSTM layers and linear layers have 320 hidden units. There are 50 output labels in total, corresponding to 26 characters, apostrophe, period, dash, space, noise, sos/eos tokens, and some other special tokens. Adadelta is used as the optimiser. The training stops after 5 epochs if there is no reduction upon the lowest validation loss.

In the experiments, firstly a CTC model is trained using the clean WSJ si284 training set. This model is referred as the "clean model" and used as the base model for the transfer learning. The top layers of this clean CTC model are viewed as a clean classifiers. While performing TL(transfer learning) in CHiME-4, we either freeze or tune the clean classifier with a small learning rate (the learning rate given by Adadelta is reduced by a factor). The bottom layers are optionally reinitialised. We compare the performance of the proposed method against models that trained only using the CHiME-4 dataset and also models trained using conventional TL from WSJ to CHiME-4, where all the layers are tuned without learning rate rescaling.

7.5.1 Random Reinitialisation of the Feature Extractor

We execute the top-down transfer learning algorithm (Algorithm 1). Table 7.4 shows the performance of the proposed TL approach to freeze the clean classifier trained in WSJ, reinitialise randomly and retrain the feature extractor using CHiME-4 data. After

Table 7.4: CERs of different models. No transfer learning means the models are trained only using CHiME-4.

Model/CER	dt05_multi	et05_real
Freeze one layer	28.5	38.9
Freeze two layers	**25.7**	**36.0**
Freeze three layers	25.7	37.1
No transfer learning		
CNN-BLSTM CTC	29.0	38.7
BLSTM CTC (Kim et al., 2017)	/	48.8

freezing the top three layers, we halt the search among the layers of the clean model, since the performance on the validation set (dt05_multi) stops improving. When the top two layers (the softmax layer and the topmost BLSTM layer with its following projection layer) are frozen, the model gives significantly smaller CER compared to the CNN-BLSTM model trained only using CHiME-4 data. We also notice that if only the softmax layer is frozen, the model does not outperform the baseline, which implies the frozen softmax does not have the capacity to force the feature extractor to learn better features. On the other hand, although freezing three layers surpasses the baseline, it gives inferior results compared to freezing two layers, which indicates that although the three-layered classifier has more capacity, the shallower feature extractor is not powerful or flexible enough to well fit the classifier. The capacity of the classifier and the feature extractor are well balanced when the top two layers are frozen.

7.5.2 Pre-training of the Feature Extractor

Here we consider the case of pre-training the feature extractor using WSJ and further training using CHiME-4. That is, the feature extractor is not randomly reinitialised. Instead, it is initialised by the weights of the clean CTC model. When training on CHiME-4, the classifier is either frozen, or the learning rate is reduced by a small factor. There is no learning rate re-scaling for the feature extractor. Due to the pre-training, in the transfer learning, all the layers have seen more data compared to the model which is trained by only using the noisy data. Thus, we compare the proposed transfer learning method with the conventional transfer learning method. That is, for all layers, there is no learning rate re-scaling. Table 7.5 summarises the results.

The CERs in Table 7.5 are lower than the CERs in Table 7.4, indicating that pre-

Table 7.5: CERs for using the clean CTC model to initialise the training on CHiME-4. The classifier is made of the topmost layer(s). During the training on CHiME-4, the classifier is either frozen or the learning rate (LR) is scaled by a small factor. No transfer learning means the model is trained only using CHiME-4.

Model	dt05_Multi	et05_real
Classifier: One layer		
Frozen	22.0	33.8
LR scaled by 0.1	22.2	33.5
LR scaled by 0.5	22.6	34.2
Classifier: two layers		
Frozen	22.5	33.8
LR scaled by 0.1	22.0	33.3
LR scaled by 0.5 (Model A)	**21.9**	**32.9**
Classifier: three layers		
Frozen	24.6	36.6
LR scaled by 0.1	22.7	34.5
LR scaled by 0.5	21.9	33.4
No LR re-scale (Model B)	21.9	33.3
LR scaled by 0.5 for all layers (Model C)	22.1	33.4
No transfer learning	29.0	38.7

training on WSJ gives a good initialisation for all the layers. Again, the capacity of the classifier and the feature extractor are well balanced when the top two layers are grouped as the classifier. The best CER is achieved by tuning the two-layered classifier using a scaled learning rate (0.5). To ascertain if the performance gains of the best model (Model A) is due to the smaller learning rate, we train a model by scaling the learning rate for all layers by 0.5 (Model C), which gives inferior results. Thus, these experiments imply that the classifier needs to be slightly tuned (compared to the feature extractor) for the noisy data to get the best performance. We also test a two-stage fine tune, i.e., first freeze the softmax layer then unfreeze it and do a fine-tuning. However, it does not give better results compared to the best model.

To show our proposed method forces the models to learn noise invariant features, we test the performance of Model A and Model B (Model B is trained by the conventional transfer learning, in where all layers are trained jointly with no learning rate re-scale) in unseen noise conditions by decoding them on all the 14 test sets of Aurora-4 (Pearce, 2002) dataset. Aurora-4 has 14 test sets. The 14 test sets contain two clean sets which were recorded by a primary closed microphone and a distant secondary microphone. These two sets were corrupted by six different additive noises to create the other 12 test sets, making 14 test sets in total. The additive noise includes noise from street traffic, train stations, cars, babble, restaurants and airports.

Table 7.6 shows that Model A surpasses Model B for all 14 test sets. Compared to Model B, Model A has 11.3% relatively lower CER on average. These results show that Model A is more capable of extracting domain invariant features. In the CHiME-4 dataset, the training set contains the noise conditions of the test set, and the conventional transfer learning method makes Model B biased towards these observed noise conditions. Thus, Model B has a good performance in the test set of CHiME-4 but inferior results in the test sets of Aurora-4. Our transfer learning method only slightly tuned the clean classifier and forces the feature extractor to extract the noisy invariant underlying patterns. Thus, Model A has the best performance in both seen and unseen noisy conditions.

Table 7.6: CERs on all 14 test sets of Aurora-4. Both Model A and Model B are initialised using the clean CTC model trained on WSJ. For Model A, the learning rate for the top two layers (the softmax layer and the topmost BLSTM layer with its following linear layer) are scaled by a factor of 0.5. For Model B, there is no learning rate re-scale. wv1: utterances recorded by head-mounted close-talking microphone, wv2: utterances recorded by desktop (far-field) microphone of a set of 18 microphones.

Test Set	Model A	Model B
airport wv1	**11.3**	12.2
babble wv1	**12.0**	13.1
car wv1	**9.8**	11.2
clean wv1	**8.1**	9.7
restaurant wv1	**13.1**	14.1
street wv1	**12.7**	14.0
train wv1	**13.6**	14.6
Average (wv1)	**11.5**	12.7
airport wv2	**26.6**	30.5
babble wv2	**27.3**	30.8
car wv2	**24.1**	26.9
clean wv2	**22.1**	25.2
restaurant wv2	**27.4**	31.5
street wv2	**27.9**	31.2
train wv2	**28.2**	32.5
Average (wv2)	**26.2**	29.8

Table 7.7: Tasks and models used for top-down layer-wise training.

Task	Dataset	Model
Speech Recognition (Clean)	WSJ	CTC CTC-Attention
Speech Recognition (Noisy)	CHiME-4	CTC
Speech Recognition (Telephone Conversation)	SWBD	Transformer
Image Classification	CIFAR-10	VGG-13 ResNet-101
Language Modeling	WikiText-2	AWD-LSTM

7.6 Experiments of Top-down Layer-wise Training

We apply the top-down layer-wise training algorithm to train a range of neural networks on speech, image and text domains. The models and datasets used are summarised in Table 7.7. For the CTC experiments on WSJ, we apply the greedy top-down layer-wise training algorithm as in Algorithm 2. For the other experiments, we either perform part of the greedy search or part of the complete search.

7.6.1 CTC Models on Clean Speech

For CTC models trained on clean speech we used the experiment setup described in Section 7.3, with the full WSJ-si284 training set. We group each BLSTM layer and the following projection layer as one layer and we view the model as a 5 layer network, i.e., **(BLSTM-projection)$\times$4–softmax**. We used three different random seeds to build three baselines, and executed greedy top-down layer-wise training (Algorithm 2) for each baseline. In each round of retraining, we used a different random seed. For Baseline 1, we also tested freezing the lowest BLSTM-projection layer, followed by reinitializing and retraining the remaining upper layers.

Top-down layer-wise training significantly reduces the CER for each of the baseline models (Table 7.8). For Baseline 1 and Baseline 2, the layer-wise training stops at the lowest layer. For Baseline 3 (where the proposed method brings the least improvement), the training procedure stops at the layer above the lowest layer (i.e. the final model is $M_{1,1,1,2}$). An additional experiment on Baseline 1 indicates that if we

Table 7.8: CERs for CTC experiments on WSJ.

Model(si284)	dev93	eval92
Baseline 1	12.4	9.7
+ greedy top-down layer-wise training	**10.8**	**8.2**
+ freeze the bottom most layer	13.1	10.5
Baseline 2	12.6	10.4
+ greedy top-down layer-wise training	**10.6**	**8.5**
Baseline 3	12.6	10.2
+ greedy top-down layer-wise training	**11.5**	**8.9**
Dropout 0.2	11.0	8.7
Dropout 0.5	9.6	7.6
Dropout 0.7	11.1	8.9
Dropout 0.5	9.6	7.6
+ greedy top-down layer-wise training	**8.2**	**6.3**
Previous work		
CTC DNN (Hannun et al., 2014)	-	10.0
BLSTM (Graves and Jaitly, 2014)	-	9.2
BLSTM (Kim et al., 2017)	11.5	9.0
Attention (Kim et al., 2017)	12.0	8.2
CTC-Attention (Kim et al., 2017)	**11.3**	**7.4**

freeze the lowest layer and retrain the top layers, the accuracy of the model drops, which is consistent with our hypothesis – within the same dataset, freezing the lower layers and retraining the upper layers is not helpful, since the lower layers are in general under-trained . This observation is also consistent with the findings of (Yosinski et al., 2014).

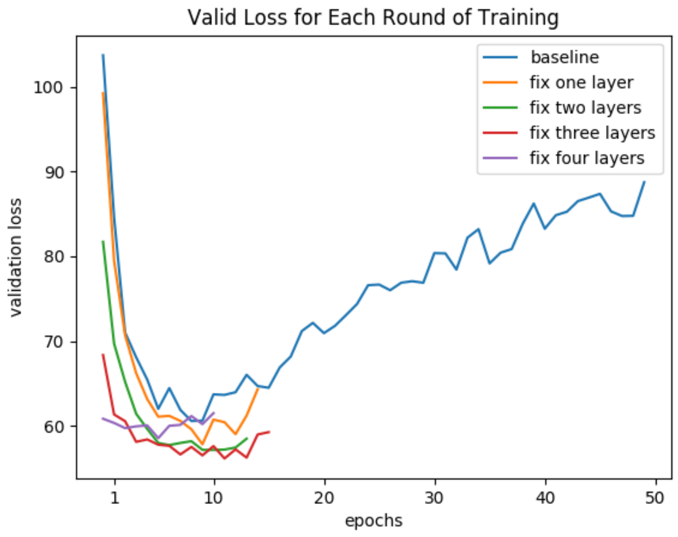
Using the same initialisation as Baseline 3, we trained three models with different dropout probabilities, with dropout probability 0.5 yielding the best model, followed by top-down layer-wise training (still using dropout probability 0.5). The results are shown in Table 7.8. When top-down layer-wise training is combined with dropout, we observe large gains in test accuracy over both the baseline and the model trained with dropout (38% and 17% relative gain, respectively). The combination of these two training methods performs impressively, with the CTC model significantly outperforming the joint CTC-attention model (15% relative gain), which is more flexible and has more capacity.

In Section 7.3, we observe that if we train the model on the 80% subset of WSJ si284 for a sufficiently long time, the classifier will become overfitting. Using the full WSJ si284, we also train Baseline 1 for a large number of epochs. Figure 7.5 shows that further training does not bring any improvement. Rather, the validation loss increases rapidly during the extended training. In contrast, as shown in Figure 7.5, in each iteration of the top-down layer-wise training, the develop error drops. Thus, although in the joint training and the top-down layer-wise training, the number of the total training epochs is large, the long joint training only makes the classifier overfit, while in each iteration of the top-down layer-wise training, the proposed method offers better models.

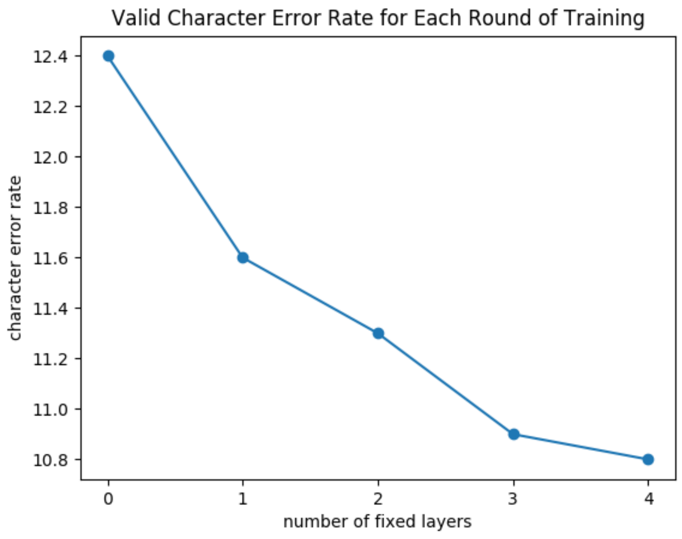
7.6.2 CTC Models on Noisy Speech

We use top-down layer-wise training to train CTC models on CHiME-4. The experimental setup is described in Section 7.3, however, in these experiments we augmented the noisy training dataset with WSJ si284. We view WSJ si284 and CHiME-4 as an entire dataset. Section 7.5 demonstrates the experimental results of transferring the classifier from WSJ to CHiME-4. The architecture of the network is **(BLSTM-projection)$\times$4–softmax** and we view each BLSTM layer with its projection layer as one layer. We perform part of the complete search in these experiments.

Top-down layer-wise training method gives a notable CER reduction over the base-



(a) Validation loss of the joint training and each iteration of the top-down layer-wise training



(b) Validation CERs of each iteration of the top-down layer-wise training

Figure 7.5: The validation loss/CERs of the joint training and each iteration of the top-down layer-wise training algorithm.

Table 7.9: CERs of CTC models on CHiME-4.

tr05+si284	dt05	et05
Baseline 4	25.2	36.0
+ top-down layer-wise training	**23.4**	**34.2**
dropout 0.5	19.9	30.8
+ top-down layer-wise training	**18.5**	**28.7**
+ freeze the bottom most layer	20.6	31.4
Previous work (Kim et al., 2017)		
CTC	37.6	48.8
Attention	35.0	47.6
CTC-Attention	**32.1**	**45.0**

line models (Table 7.9). The model trained with dropout has the same initial weights as Baseline 4. In the top-down retraining of Baseline 4, we notice that in the first round of retraining, it is essential to group the top three layers (BLSTM-projection $\times$ 2 - softmax) as the classifier. Retraining with a shallow classifier does not bring a performance gain. The final model after top-down layer-wise training for Baseline 4 is $M_{3,1,1}$. For the model trained with dropout, grouping the top two layers as the classifier in the retraining brings a CER reduction. The final model is $M_{2,1,1,1}$. Freezing the bottom most layer and retraining the top layers yields a poorer performance. The previous work only uses single channel noisy data and does not have data augmentation, thus it has inferior results compared to the baseline model.

7.6.3 CTC-Attention Models on Clean Speech

We apply the proposed training method to train hybrid CTC-attention models on WSJ. The feature extraction process and the labels are described in Section 7.3. We test two architecture settings, which are similar to the settings used by Delcroix et al. (2018) and Zhang et al. (2019). In the first architecture, the encoder is a 4 layer BLSTM, with each BLSTM is followed by a projection layer, i.e., **(BLSTM-projection)**$\times$**4**. For the second architecture, the lowest layers of the encoder form a VGG-like convolutional network, comprising 2×2 max-pooling layers with stride 2 and 3×3 convolution layers with stride 1 and padding 1. Above the VGG-like network are 6 BLSTM-projection layers. The full architecture of the encoder is **conv64–conv64–pool–conv64–conv128–pool–(BLSTM-projection)**$\times$**6**. For both settings, the decoder

Table 7.10: CERs of CTC-Attention models on WSJ.

Model(si284)	dev93	eval92
A: BLSTM CTC-Attention	8.0	6.2
+ freeze softmax layers from A	**7.3**	**5.4**
+ freeze softmax layers from B	7.5	5.7
B: VGG-BLSTM CTC-Attention	7.2	5.4
+ freeze softmax layers from B	7.0	**5.2**
+ freeze softmax layers from A	**6.7**	**5.2**
Previous works		
Attention (Bahdanau et al., 2016)	9.0	6.8
+TwinNet (Serdyuk et al., 2018)	8.4	6.2
CTC-Attention (Kim et al., 2017)	11.3	7.4
+ pad silence (Zhang et al., 2019)	7.8	5.8
+ deep encoder (Delcroix et al., 2018)	**7.4**	**5.5**

is a 1-layer BLSTM; all the BLSTM and projection layers have 320 hidden units. Location-based attention (Chorowski et al., 2015) with 10 centered convolution filters of width 100 is used for both architectures. Adadelta with epsilon decay of 0.01 and gradient clipping are used for optimization. The training stops when the best validation loss does not drop after 5 epochs.

Top-down layer-wise training, joint training, and retraining use the same initialisation weights. We freeze only the softmax layer of the CTC part and the softmax layer of the attention based encoder-decoder part (the final model is $M_{1,n-1}$) due to the large number of experiments; and performance gains are already observed on $M_{1,n-1}$.

Table 7.10 demonstrates that, as far as we know, top-down layer-wise training results in state-of-the art character error rates for LSTM-based end-to-end models on WSJ, without language model fusion or adaptations.

The architectures of the decoder/attention parts in Model A and Model B are the same. However, Model B has a deeper encoder. Table 7.10 also indicates that by executing a better training algorithm, a shallow model (Model A) achieves similar performance to a deep model (Model B).Besides retraining each model with its own trained frozen softmax layers, we also retrained Model A based on the frozen softmax layers of Model B and vice versa. As shown in Table 7.10, retraining according to the softmax layers of Model B leads to worse results than retraining based on the

Table 7.11: WERs for experiments on SWBD

Model	SWBD/Callhm (WER)
Baseline	9.0/18.1
+ freeze softmax layers	**8.6/17.2**
Previous works	
Transformer (Karita et al., 2019)	9.0/18.1
Very deep self-attention (Pham et al., 2019)	10.4/18.6
Multi-stride self-attention (Han et al., 2019)	9.1/_

softmax layers of Model A. This indicates that, compared to the shallow baseline, the better performance of the deep baseline (CNN-BLSTM) comes from the deep feature extractor (encoder), rather than the classifier.

7.6.4 Transformer Models on Telephone Conversations

We test the proposed training method on SWBD. We employ the Transformer based CTC-attention model and the experimental setup follows Karita et al. (2019). We freeze only the softmax layer of the CTC part and the softmax layer of the Transformer due to large number of experiments and performance gains already observed, although executing the full top-down layer-wise training algorithm may give further improvements. Table 7.11 shows the proposed method significantly reduces the word error rates over the baseline, and gives noticeably better results compared to the previous works.

To the best of our knowledge, the work of Nøkland and Eidnes (2019) is the only time that layer-wise training has outperformed joint training . However, that approach was not effective for models with residual connections. In contrast, the proposed method is able to improve the performance of Transformers, which has residual connection within and across each self-attention layer. Section 7.6.5 further shows the proposed top-down layer-wise training method is effective in training ResNet.

7.6.5 CNN Models for Image Classification

We apply top-down layer-wise training for training VGG-13 and ResNet-101 models on CIFAR-10. For VGG-13, the architecture of the convolution layers was the same as that used by Simonyan and Zisserman (2015). After the convolution layers, there

Table 7.12: Test error in percent of VGG13 models on CIFAR-10.

Model	Seed 1	Seed 2	Seed 3
VGG13	5.83	–	–
top-down layer-wise training $M_{1,12}$	**5.49**	5.73	5.75
top-down layer-wise training $M_{2,11}$	5.62	5.55	5.67
top-down layer-wise training $M_{3,10}$	5.58	5.67	5.69
Previous works			
VGG (Belilovsky et al., 2019)			7.5
VGG11B (Nøkland and Eidnes, 2019)			5.56

is an average pooling layer with output size 7×7. For ResNet-101, the architecture is the same as that used by He et al. (2016). Batch normalisation (Ioffe and Szegedy, 2015) was used for both models. Stochastic gradient descent with momentum 0.9 and weight decay 0.0005 is used as the optimization method. The training lasts 350 epochs. The initial training rate is 0.1 and it is decayed by 0.1 after every 100 epochs. Random cropping and horizontal flipping are used for data augmentation. All models were built using PyTorch.

For the VGG experiments, we grouped the topmost layer, the top two layers and all the top three layers (the entire fully-connected sub-network) as the frozen classifier, respectively. Retraining the lower layers results in final models $M_{1,12}$, $M_{2,11}$ and $M_{3,10}$. Each retraining is performed three times, one time with the same initial weights as the baseline and twice initializing the weights with different random seeds. Table 7.12 shows that top-down layer-wise training gives a lower error rate, compared to the baseline. Due to high number of experiments, we do not further execute the top-down layer-wise training (e.g. freeze more layers from $M_{1,12}$ then retrain), although achieving further performance gain is plausible.

For the ResNet experiments, we explored using the fully connected layer and the fully connected layer with the topmost convolution layer as the frozen classifier, respectively. We did not freeze any batch normalisation layer. In each round of retraining, we used the same initial weights as the base model. In this set of experiments, we executed the search along both the epoch dimension and the layer dimension. The classifier from an arbitrarily picked early epoch (epoch 209) is also used for the top-down layer-wise training. Table 7.13 summarises the results. The early epoch shows a higher error rate for the joint training but lower error rates for the retraining, indicating that,

Table 7.13: Test error in percent of ResNet 101 models on CIFAR-10.

Model	Fully Trained	Epoch 209
ResNet 101	4.41	4.89
+ freeze fully connected	4.34	**4.18**
+ freeze topmost CN	4.53	4.29

Table 7.14: Perplexity of the AWD-LSTM language models on WikiText-2.

Model	Dev	Test
Baseline seed 1	68.7	65.6
+ freeze topmost layer seed 1	68.2	65.3
+ freeze topmost layer seed 2	68.2	**65.2**
+ freeze topmost layer seed 3	**68.1**	**65.2**
Previous work (Merity et al., 2018)	68.6	65.8

for the baseline model, the classifier is over-trained at the end of training and searching across the time dimension is beneficial. We did not perform a further search due to the large number of experiments, but the current results already show that complete search is beneficial.

7.6.6 AWD-LSTM Language Model

We apply the proposed method in training the AWD-LSTM language model (Merity et al., 2018) on WikiText-2 dataset. The architecture of the networks and the training procedure are same as in the previous work (Merity et al., 2018). All models are built through PyTorch.

We freeze the topmost layer and retrain the bottom layers. The input/output embedding are tied in both of the joint training and the retraining (so the input embedding is also frozen during the retraining). This does not contradict to the top-down approach since although tied with the input embedding, the output embedding is still the closest to the output and in general the gains of the tied embedding is from the output embedding (Press and Wolf, 2017). The top-down method can be viewed as making the LSTM sufficiently trained by retraining the LSTM based on the trained and frozen word embedding.

We run the retraining three times – with the same/different initial weights as the baseline. As shown in Table 7.14, although the performance gains from the top-down layer-wise training are not as significant as the gains in previous experiments, the reduction of the perplexity is consistent. The proposed training method may implicitly overlap with some regularization techniques used in the training of AWD-LSTM. However, these regularization methods do not nullify the advantages of the top-down layer-wise training.

7.7 Summary

In this chapter, we have shown the existence of general classifiers. Based one this, we proposed top-down training methods which transfer general classifiers across different datasets and within the same dataset.

We demonstrated that classifiers trained on a large amount of clean data are more general than classifiers trained with a small amount of noisy data. Based on this, we presented a novel top-down transfer learning method that transfers classifiers from clean data to noisy data for speech recognition. Our experimental results showed the proposed transfer learning method forces the feature extractor to learn noise invariant features and leads to significant performance gains.

We showed that in the conventional training where all the layers are jointly trained, the classifier firstly becomes general, and then tends to overfit. To address this, we developed a novel layer-wise top-down training method, which freezes the trained classifier to prevent it from overfitting. We demonstrated the proposed top-down layer-wise training consistently outperforms conventional joint training across speech recognition, image classification, and language modelling tasks.

Chapter 8

Conclusion and Future Work

8.1 Work Done

In this thesis, we proposed methods which enable attentional sequence-to-sequence ASR models to more effectively model sequential data.

For RNN based encoder-decoder ASR models, we proposed a dynamical subsampling RNN which enables the encoder to skip unessential frames during training. Our proposed dynamical subsampling RNN encoder has outperformed baseline static subsampling RNN encoders by a large margin (up to 23.5% relative PER reduction). We also developed a fully trainable windowed attention mechanism which guarantees monotonic left-to-right alignments between inputs and outputs. Compared to the conventional attention mechanisms, our trainable windowed attention mechanism achieved 17% relative CER reduction.

We further investigated the usefulness of self-attention for ASR models. We observed that the self-attention mechanism is not essential for upper layers of Transformer encoders, and there are also redundant individual attention heads across all the layers of Transformer models. We therefore proposed randomly removing attention heads during training. Our proposed training method has improved very strong baseline models and has achieved SOTA results on datasets with different ASR scenarios.

We next demonstrated that it is feasible to train neural networks in a top-down manner. We showed that there are classifiers (upper layers of DNNs) which are more general than other classifiers. We developed algorithms for searching for general classifiers, and showed that training DNNs with general classifiers enable the models to have better generalisation on ASR, NLP and image classification tasks.

8.2 Future Work

In this section, we discuss potential directions in which this thesis could be extended. The areas for future work include online ASR and the pre-training of neural networks.

8.2.1 Online ASR Systems

To achieve low latency, offline ASR systems typically start to emit outputs before seeing the entire input sequence. For the same reason, when generating outputs, these models usually only have access to a chunk of inputs.

For online attentional encoder-decoder models, Jaitly et al. (2016) and Sainath et al. (2018) proposed to process inputs chunk by chunk; when processing the current chunk, if an "end-of-chunk" symbol is generated, the model will discard the current chunk and begin to process the next chunk. Chiu and Raffel (2018) and Tsunoo et al. (2019) compared the encoder hidden states with the decoder hidden states to decide the start point of the current chunk. However, the chunk size is fixed and it is set as a hyper-parameter.

RNN-Transducers, which consist of an audio encoder, a prediction network and a joint network, are suitable for online ASR. Since self-attention based offline models can outperform RNN based offline models, previous works have explored building online Transformer-Transducer models with a self-attention audio encoder (Yeh et al., 2019; Chen et al., 2021) or with a self-attention audio encoder and a self-attention prediction network (Yeh et al., 2019; Zhang et al., 2020). In these models, the self-attention audio encoder is restricted to only process inputs inside fixed length chunks. The chunk size for each self-attention layer, number of frames for looking back and looking ahead, and the step size of moving the chunk are hyper-parameters.

In this thesis, we have proposed methods which make sequence-to-sequence ASR models better utilise contextual information. Although the models which we have studied are offline models, we can develop online models based on our findings. For example, rather than setting chunk parameters as hyper-parameters, we can further explore making these parameters trainable, so they could be adaptively changed during the test time.

8.2.2 Self-supervised Pre-training

Previous works have studied self-supervised pre-training for natural language processing tasks and speech processing tasks. For NLP, Devlin et al. (2018) and Brown et al. (2020) explored the pre-training of deep self-attention models. For speech processing, Schneider et al. (2019) and Kawakami et al. (2020) used contrastive loss to push the hidden representations of temporally distant acoustic frames away from each other, while Liu et al. (2020) and Chi et al. (2021) used reconstruction loss to predict the masked frame through the hidden representations of the surrounding frames of the masked frame. In these self-supervised pre-training methods, the learned hidden representations are not task-specific. The DNNs which are obtained through these methods can be used as feature extractors for downstream tasks.

In this thesis, we demonstrated that DNNs can be constructed based on trained classifiers (upper layers) and the training with general classifiers leads to better generalisation. While the self-supervised pre-training methods yield feature extractors which are not task-specific, our proposed top-down training method gives classifiers which are task-specific. Therefore, it is worth exploring combining self-supervised pre-training methods with our top-down training method, so that the model could have both a strong feature extractor and a general classifier for the specific task.

Bibliography

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. In *ICASSP*, pages 4945–4949.

Barker, J., Marxer, R., Vincent, E., and Watanabe, S. (2017). The third 'chime' speech separation and recognition challenge: Analysis and outcomes. *Computer Speech and Language*, 46:605–626.

Belilovsky, E., Eickenberg, M., and Oyallon, E. (2019). Greedy layerwise learning can scale to ImageNet. In *ICML*, pages 583–593.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *ICML workshop*.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *NeurIPS*.

Bu, H., Du, J., Na, X., Wu, B., and Zheng, H. (2017). Aishell-1: An open-source mandarin speech corpus and a speech recognition baseline. In *O-COCOSDA*.

Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., and Chang, S.-F. (2018). Skip rnn: Learning to skip state updates in recurrent neural networks. In *ICLR*.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *ICASSP*, pages 4960–4964.

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 *ICASSP*, pages 5904–5908.

Chi, P.-H., Chung, P.-H., Wu, T.-H., Hsieh, C.-C., Chen, Y.-H., Li, S.-W., and Lee, H.-y. (2021). Audio albert: A lite bert for self-supervised learning of audio representation. In *IEEE SLT*, pages 344–350.

Chiu, C.-C. and Raffel, C. (2018). Monotonic chunkwise attention. In *ICLR*.

Cho, J., Baskar, M. K., Li, R., Wiesner, M., Mallidi, S. H., Yalta, N., Karafiat, M., Watanabe, S., and Hori, T. (2018). Multilingual sequence-to-sequence speech recognition: architecture, transfer learning, and language modeling. In *IEEE SLT*, pages 521–527.

Chorowski, J. and Jaitly, N. (2017). Towards better decoding and language model integration in sequence to sequence models. In *INTERSPEECH*, pages 523–527.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *NeurIPS*, pages 577–585.

Chung, J., Ahn, S., and Bengio, Y. (2017). Hierarchical multiscale recurrent neural networks. In *ICLR*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*, pages 2978–2988.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *ICML*, pages 933–941.

Delcroix, M., Watanabe, S., Ogawa, A., Karita, S., and Nakatani, T. (2018). Auxiliary feature based adaptation of end-to-end ASR systems. In *INTERSPEECH*, pages 2444–2448.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *IEEE CVPR*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *ICASSP*, pages 5884–5888.

Dozat, T. (2016). Incorporating Nesterov momentum into Adam. *ICLR 2016 workshop*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

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

Fan, A., Grave, E., and Joulin, A. (2019). Reducing transformer depth on demand with structured dropout. In *ICLR*.

Gao, Z., Zhang, S., Lei, M., and McLoughlin, I. (2020). San-m: Memory equipped self-attention for end-to-end speech recognition. In *INTERSPEECH*.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett, D. S. (1993). Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *ICML*, pages 1243–1252.

Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *ICASSP*, pages 2494–2498.

Ghoshal, A., Swietojanski, P., and Renals, S. (2013). Multilingual training of deep neural networks. In *ICASSP*, pages 7319–7323.

Gillick, L. and Cox, S. (1989). Some statistical issues in the comparison of speech recognition algorithms. In *ICASSP*.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). Switchboard: telephone speech corpus for research and development. In *ICASSP*.

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 *ICML*, pages 369–376.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, pages 1764–1772.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649.

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. In *INTERSPEECH*.

Guo, P., Boyer, F., Chang, X., Hayashi, T., Higuchi, Y., Inaguma, H., Kamo, N., Li, C., Garcia-Romero, D., Shi, J., Shi, J., Watanabe, S., Wei, K., Zhang, W., and Zhang, Y. (2020). Recent developments on espnet toolkit boosted by conformer. *arXiv preprint arXiv:2010.13956*.

Hadian, H., Sameti, H., Povey, D., and Khudanpur, S. (2018). End-to-end speech recognition using lattice-free mmi. In *INTERSPEECH*, pages 12–16.

Han, K. J., Huang, J., Tang, Y., He, X., and Zhou, B. (2019). Multi-stride self-attention for speech recognition. In *INTERSPEECH*, pages 2788–2792.

Hannun, A. Y., Maas, A. L., Jurafsky, D., and Ng, A. Y. (2014). First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs. *arXiv:1408.2873*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. In *ECCV*, pages 630–645.

Hettinger, C., Christensen, T., Ehlert, B., Humpherys, J., Jarvis, T., and Wade, S. (2017). Forward thinking: Building and training neural networks one layer at a time. *arXiv:1706.02480*.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoffer, E., Hubara, I., and Soudry, D. (2018). Fix your classifier: the marginal value of training the last weight layer. In *ICLR*.

Hori, T., Watanabe, S., Zhang, Y., and Chan, W. (2017). Advances in joint ctc-attention based end-to-end speech recognition with a deep cnn encoder and rnn-lm. In *INTERSPEECH*.

Huang, J.-T., Li, J., Yu, D., Deng, L., and Gong, Y. (2013). Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *ICASSP*, pages 7304–7308.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456.

Jaitly, N., Le, Q. V., Vinyals, O., Sutskever, I., Sussillo, D., and Bengio, S. (2016). An online sequence-to-sequence model using partial conditioning. In *NeurIPS*, pages 5067–5075.

Jiang, D., Li, W., Zhang, R., Cao, M., Luo, N., Han, Y., Zou, W., and Li, X. (2020). A further study of unsupervised pre-training for transformer based speech recognition. *arXiv preprint arXiv:2005.09862*.

Karita, S., Chen, N., Hayashi, T., Hori, T., Inaguma, H., Jiang, Z., Someki, M., Soplin, N. E. Y., Yamamoto, R., Wang, X., et al. (2019). A comparative study on transformer vs rnn in speech applications. In *ASRU*, pages 449–456.

Kawakami, K., Wang, L., Dyer, C., Blunsom, P., and van den Oord, A. (2020). Learning robust and multilingual speech representations. In *EMNLP*, pages 1182–1192.

Kim, S., Hori, T., and Watanabe, S. (2017). Joint CTC-attention based end-to-end speech recognition using multi-task learning. In *ICASSP*, pages 4835–4839.

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

Ko, T., Peddinti, V., Povey, D., and Khudanpur, S. (2015). Audio augmentation for speech recognition. In *INTERSPEECH*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105.

Liu, A. T., Yang, S.-w., Chi, P.-H., Hsu, P.-c., and Lee, H.-y. (2020). Mockingjay: Unsupervised speech representation learning with deep bidirectional transformer encoders. In *ICASSP*, pages 6419–6423.

Lu, L., Kong, L., Dyer, C., Smith, N. A., and Renals, S. (2016). Segmental recurrent neural networks for end-to-end speech recognition. *arXiv preprint arXiv:1603.00223*.

Lu, L., Liu, C., Li, J., and Gong, Y. (2020). Exploring transformers for large-scale speech recognition. *INTERSPEECH*, pages 5041–5045.

Lu, L., Zhang, X., Cho, K., and Renals, S. (2015). A study of the recurrent neural network encoder-decoder for large vocabulary speech recognition. In *INTERSPEECH*, pages 3249–3253.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421.

Malach, E. and Shalev-Shwartz, S. (2018). A provably correct algorithm for deep learning that actually works. *arXiv:1803.09522*.

Marquez, E. S., Hare, J. S., and Niranjan, M. (2018). Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5475–5485.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and optimizing LSTM language models. In *ICLR*.

Miao, Y., Li, J., Wang, Y., Zhang, S.-X., and Gong, Y. (2016). Simplifying long short-term memory acoustic models for fast training and decoding. In *ICASSP*, pages 2284–2288.

Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one. In *NeurIPS 2019 : Thirty-third Conference on Neural Information Processing Systems*, pages 14014–14024.

Mostafa, H., Ramesh, V., and Cauwenberghs, G. (2018). Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608.

Nakatani, T. (2019). Improving transformer-based end-to-end speech recognition with connectionist temporal classification and language model integration. *INTERSPEECH*.

Nguyen, T.-T., Nguyen, X.-P., Joty, S., and Li, X. (2020). Differentiable window for dynamic local attention. In *ACL*, pages 6589–6599.

Nøkland, A. and Eidnes, L. H. (2019). Training neural networks with local error signals. In *ICML*, pages 4839–4850.

Oglic, D., Cvetkovic, Z., Bell, P., and Renals, S. (2020). A deep 2d convolutional network for waveform-based speech recognition. In *INTERSPEECH*.

Oord, A. v. d., 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. *arXiv preprint arXiv:1609.03499*.

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. *INTERSPEECH*, pages 2613–2617.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Paul, D. B. and Baker, J. M. (1992). The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. ACL.

Pearce, D. (2002). Aurora working group: Dsr front end lvcsr evaluation au/384/02.

Peng, H., Schwartz, R., Li, D., and Smith, N. A. (2020). A mixture of $h-1$ heads is better than h heads. *ACL*, pages 6566–6577.

Pham, N.-Q., Nguyen, T.-S., Niehues, J., Müller, M., and Waibel, A. (2019). Very deep self-attention networks for end-to-end speech recognition. In *INTERSPEECH*, pages 66–70.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The Kaldi speech recognition toolkit. In *IEEE ASRU*.

Povey, D., Hadian, H., Ghahremani, P., Li, K., and Khudanpur, S. (2018). A time-restricted self-attention layer for asr. In *ICASSP*, pages 5874–5878.

Prabhavalkar, R., Sainath, T. N., Li, B., Rao, K., and Jaitly, N. (2017). An analysis of attention in sequence-to-sequence models. In *INTERSPEECH*, pages 3702–3706.

Press, O., Smith, N. A., and Levy, O. (2019). Improving transformer models by re-ordering their sublayers. *arXiv preprint arXiv:1911.03864*.

Press, O. and Wolf, L. (2017). Using the output embedding to improve language models. In *EACL*, pages 157–163.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Renals, S., Hain, T., and Bourlard, H. (2007). Recognition and understanding of meetings the ami and amida projects. In *ASRU*.

Sainath, T. N., Chiu, C.-C., Prabhavalkar, R., Kannan, A., Wu, Y., Nguyen, P., and Chen, Z. (2018). Improving the performance of online neural transducer models. In *ICASSP*, pages 5864–5868.

Sainath, T. N., Vinyals, O., Senior, A., and Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *ICASSP*, pages 4580–4584.

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. In *INTERSPEECH*, pages 3465–3469.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725.

Serdyuk, D., Ke, N. R., Sordoni, A., Trischler, A., Pal, C., and Bengio, Y. (2018). Twin networks: Matching the future for sequence generation. In *ICLR*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science.

Song, I., Chung, J., Kim, T., and Bengio, Y. (2018). Dynamic frame skipping for fast speech recognition in recurrent neural network based acoustic models. In *ICASSP*, pages 4984–4988.

Sperber, M., Niehues, J., Neubig, G., Stüker, S., and Waibel, A. (2018). Self-attentional acoustic models. In *INTERSPEECH*, pages 3723–3727.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NeurIPS*, pages 3104–3112.

Swietojanski, P., Ghoshal, A., and Renals, S. (2012). Unsupervised cross-lingual knowledge transfer in dnn-based lvcsr. In *IEEE SLT*, pages 246–251.

Thomas, S., Ganapathy, S., and Hermansky, H. (2012). Multilingual mlp features for low-resource lvcsr systems. In *ICASSP*, pages 4269–4272.

Tjandra, A., Sakti, S., and Nakamura, S. (2017). Local monotonic attention mechanism for end-to-end speech and language processing. In *IJCNLP*, pages 431–440.

Tong, S., Garner, P. N., and Bourlard, H. (2017). Multilingual training and cross-lingual adaptation on ctc-based acoustic model. *arXiv preprint arXiv:1711.10025*.

Tóth, L. (2015). Phone recognition with hierarchical convolutional deep maxout networks. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015(1):25.

Tsunoo, E., Kashiwagi, Y., Kumakura, T., and Watanabe, S. (2019). Towards online end-to-end transformer automatic speech recognition. *arXiv preprint arXiv:1910.11871*.

Vanhoucke, V., Devin, M., and Heigold, G. (2013). Multiframe deep neural networks for acoustic modeling. In *ICASSP*, pages 7582–7585.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*, pages 5998–6008.

Vincent, E., Watanabe, S., Nugraha, A. A., Barker, J., and Marxer, R. (2017). An analysis of environment, microphone and data simulation mismatches in robust speech recognition. *Computer Speech and Language*, 46:535–557.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*.

Wang, Y., Mohamed, A., Le, D., Liu, C., Xiao, A., Mahadeokar, J., Huang, H., Tjandra, A., Zhang, X., Zhang, F., et al. (2020). Transformer-based acoustic modeling for hybrid speech recognition. In *ICASSP*, pages 6874–6878.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Soplin, N. E. Y., Heymann, J., Wiesner, M., Chen, N., et al. (2018). ESPnet: End-to-end speech processing toolkit. *arXiv preprint arXiv:1804.00015*.

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

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

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*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *NeurIPS*, pages 3320–3328.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zeyer, A., Bahar, P., Irie, K., Schlüter, R., and Ney, H. (2019). A comparison of Transformer and LSTM encoder decoder models for ASR. In *IEEE ASRU*, pages 8–15.

Zeyer, A., Irie, K., Schlüter, R., and Ney, H. (2018). Improved training of end-to-end attention models for speech recognition. *INTERSPEECH*, pages 7–11.

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 *ICASSP*, pages 7829–7833.

Zhang, S., Loweimi, E., Bell, P., and Renals, S. (2019). Windowed attention mechanisms for speech recognition. In *ICASSP*, pages 7100–7104.

Zhang, S., Loweimi, E., Bell, P., and Renals, S. (2021). On the usefulness of self-attention for automatic speech recognition with transformers. In *IEEE SLT*, pages 89–96.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Bengio, C. L. Y., and Courville, A. (2017). Towards end-to-end speech recognition with deep convolutional neural networks. *arXiv preprint arXiv:1701.02720*.

Zhao, K., Matsukawa, T., and Suzuki, E. (2018). Retraining: a simple way to improve the ensemble accuracy of deep neural networks for image classification. In *IEEE ICPR*, pages 860–867.