浙江大学

本 科 生 毕 业 论 文

文献综述和开题报告

学生姓名	王佳明
学生学号	3190105268
指导教师	张东祥
年级与专业	计算机科学与技术
所在学院	计算机科学与技术学院

一、题目：浙江方言的数据集构建与识别技术研究

二、指导教师对文献综述和开题报告的具体要求：

严格遵守学术道德规范。引用他人的观点、参考资料、及文字，均应加以注释和说明；严禁抄袭 他人研究成果和伪造数据等行为。

包括：

1）研究背景、研究意义与相关技术现状分析等。

2）研究内容定义清楚，可在规定时间内完成。

3）提出解决任务的大致技术思路和研究方法。

4）4 月份需要完成原型设计，5 月份完成论文初稿。

外文翻译：翻译一篇内容与毕业设计课题相关的英文文献，要求行文流利，部分难以翻译的语句可采用意译，译文字数要求 3000 字以上。

文献综述：围绕所研究问题，查阅相关文献 10 篇以上（外文不少于 5 篇），所选文献 应主要选自学术期刊或学术会议的文章，其次是专著和教材。文献综述要条理清楚，内容要切题，包括或部分包括国内外现状、研究方向、进展情况、存在问题、参考依据， 文献综述字数要求 3000 字以上。

指导教师（签名）

年　　月　　日

目　　录

一、文献综述

1. 背景介绍

在当今社会，和随身智能设备的语音交互已经随处可见，并且极大方便了人们的日常生活。在这背后，语音识别技术发挥了不可替代的作用。语音识别，又称为自动语音识别（Automatic Speech Recognition， ASR)，通过处理语音信号并将之转换为文本实现了高效便捷的人机交互方式，已经成为人们生活中不可或缺的一环。而方言识别又是语音识别领域中的热点问题。一方面，方言以其独特的文学性和在地区治理方面的作用彰显了自身的价值，另一方面，方言又因其低资源的特点使得相关研究长期停滞不前。深度学习的兴起，则为方言识别领域注入了新的活力，通过神经网络，模型对方言的识别准确率可以得到巨大提升。本文将以浙江方言为例，对方言语音识别进行综述。

1.1 浙江方言体系

浙江方言的分布情况较为复杂，境内存在吴语、闽语、徽语、赣语等不同汉语方言[1]。其中吴语是浙江的主体方言，使用人口占浙江总人口的98%以上。其具有如下特点[2]：

1. 保留古汉语全浊声母。中古汉语字分全清（不送气）、次清（送气）、全浊、次浊四类。古代《切韵》《韵镜》提到清浊概念。声带振动的音为浊音，世界各国均有大量浊音。全浊为浊阻碍音（塞音、塞擦音和擦音），次浊为响音（鼻音、边通音和通音）。除吴语外的所有汉语方言都无浊辅音（仅湘、闽南有少许），吴语有全部浊音。
2. 具有古汉语整齐八声调。阴平/阴上/阴去/阴入/阳平/阳上/阳去/阳入。狭义江南的江南话沿袭传统八声调，江南各城基本都是八调俱全，如常熟、宜兴、绍兴等。四声因清浊而各分阴阳。
3. 保留古汉语平仄音韵。保留全部入声。吴语有深厚的汉语资历，读诗朗朗上口，平仄合韵。普通话丢入声等多方面原因导致不分平仄。舒促音为汉语语音的一种划分，入声为促音，短而刚劲有力，是最基本的仄音之一，是汉语的重要组成部分。江南汉语集“强迫性的连续变调”“舒促音”于一体，共同构成汉语“平仄”系统。入声有韵尾，如很多吴越人把“合”读成“合hep”，但考虑到方便性，江南人一般忽略韵尾。

4. 保留古汉语中强制性的规则连续变调。吴语一句话或一个短语有时只有第一个字是保持其原本的声调，后面的字根据第一个字的声调以及说话者想要表达的意思会改变声调的高低和走向，称作广式连续变调。汉语的连续变调原则使得句子能够连成一个整体，显得不突兀不生硬。
5. 保留古汉语尖团分化音。如：箭zian—剑、清cing—轻、小siao—晓、西—希、息—吸，每对读音都不同，前者为尖音、后者为团音。“西—希、息—吸”四字为四个不同音，分别为“si:—xi、sih—xih”。
6. 保留鼻音、边音声母有紧喉和带浊流的区别。紧喉音譬如：我/鹅/饿/牙/碍等（“我”字长安音为ngè，吴语读ngǒu/ngo，不可读如普wo/淮wu）
7. 元音、辅音较多。元音另有如英音标的ae、e。元音分长短，短元音即上述第3点的入声。辅音是全国最多的，因其有大量浊辅音。

目前，领先的语音识别公司的相关产品，如科大讯飞的讯飞听见，百度的百度语音识别等，虽然在普通话上表现优异，但是都尚未支持浙江方言。因此如何识别浙江方言就是一个尚待解决的问题。

1.2 语音识别

语音识别（Speech Recognition）是一种将人类口头语言转换为计算机可以理解的指令进行输出人类能够理解的文本的计算机技术。语音信号在时域方面的描述能力较差，而其频率成分则相对固定，特别是不同元音之间的频谱会呈现出明显的差别，因而在实际应用中常常使用频域特征构建声学模型。为了处理这些离散的数字信号，研究者设计了不同的特征提取器，并采用滑动窗口技术对语音信号进行分帧处理，进而计算每一帧语音的特征。目前常用的声学特征有梅尔频率倒谱系数特征（MFCC）[3]、感知线性预测特征（PLP）[4]和滤波器组特征[5]等。

语音识别技术按照模型框架可以分成两大类，分别是传统语音识别框架和端到端语音识别框架，接下来将分别介绍两种语音识别框架。

1.2.1 传统语音识别模型

传统语音识别模型通常由预处理、特征提取、声学模型、语音模型、发音词典以及解码器六个部分组成。其中声学模型、语言模型、发音词典和解码器最为关键。声学模型是

一个概率模型，用于将语音信号的特征向量序列映射为文本。典型的例子是高斯混合模型-隐马尔可夫模型（Gaussian Mixture Model-Hidden Markov Model， GMM-HMM）[6]和深度神经网络-隐马尔可夫模型（DNN-HMM）[7]。语言模型主要通过语言学分析来解决语音识别中的歧义问题，从而进一步提高语音识别的准确性。目前主流的语言模型有基于多元文法（N-gram)和循环神经网络语言模型(Recurrent Neural Network Language Model，RNNLM）两种类型。其中基于RNN的模型能够更好利用上下文信息，对长距离序列进行建模。发音词典用于描述不同音素和词汇之间的映射关系。解码器根据声学模型、语言模型和发音词典构建的解码网络，通过搜索算法寻找最优解作为语音识别的结果。

1.2.2 端到端语音识别模型

与传统语音识别模型相比，在端到端语音识别方法中，可以使用标注文本序列中的最小合法单元作为建模单元。采用自动对齐的方法，避免模型与对齐标签之间的循环依赖问题。常用的端到端语音识别模型有连接时序分类模型和基于注意力机制的模型。端到端语音识别模型通过使用单个网络架构降低了构建语音识别系统的难度，省去了传统方法中需要构建发音词典等繁琐步骤。对于方言而言这些优势更加明显。与传统模型不同，端到端的方法关注输出序列和输入序列是否相同，而非预测序列和输入序列在某个时间点上是否对齐[8]。传统模型通过隐马尔可夫模型来约束输出和输入的对齐关系，而连接时序分类（Connectionist Temporal Classification， CTC）[9]则用穷举的方法枚举所有合法输入输出组合。在解码阶段，CTC通过对目标序列和原始输入序列的分布进行建模，消除单个声学特征建模单元到文本序列建模单元的手动对齐操作，从而实现了端到端的声学模型训练。由于语音输出标签的长度常常小于输入语音帧的长度，所以CTC通过允许输出重复的输出标签，以及在重复标签中插入blank的方式，构建与输入语音帧长度相同的CTC路径。而自监督学习技术[10]的出现进一步促进了CTC的发展。

Transformer[11]是由Google提出的一种基于编码器-解码器架构的模型，由一个接受输入序列并将其映射成矢量表示的编码器和一个接受矢量表示并输出序列的解码器组成。其中编码器和解码器由多层自我注意和前馈神经网络组成。开始时Transformer被应用在机器翻译领域，随后被引入到语音识别中来[12]。通过注意力机制，模型可以专注于输入序列的特定部分，从而使得模型能够捕捉全局信息，形成上下文的长期依赖关系。尽管Transformer模型在建模上下文方面表现优异，但是其提取细粒度的局部特征的能力比较差。与此相对，CNN在学习局部信息方面表现更佳，可以捕获边缘和形状等特征。因此，为

了实现更好的性能，谷歌提出了Conformer[13],通过结合CNN和Transformer，达到了当时最先进的识别准确率。具体来说，Conformer在解码器结构上类似于Transformer，但是编码器部分充分利用卷积操作来提取局部特征，同时使用注意力机制捕获全局信息。

1.2.3 语音识别的评价指标

对于语音识别而言，不同的建模单元将采用不同的模型评价指标。这里将介绍一些常用的评价标准。

句错误率（Sentence Error Rate，SER）描述了在语音识别任务中，语音识别系统所识别出的语句和真实语句之间的误差。其计算方式是计算错误语句占总体的比例。SER越低，表明该系统的识别准确率越高。

词错误率（Word Error Rate，WER）是最常用的评价指标之一，计算方式是插入、删除和替换操作的总次数除以参考文本中的总词数。WER越低，表示语音识别系统的性能越好。

实时因子（Real-Time Factor，RTF）衡量了处理语音所需的时间与实际语音持续时间的比值。RTF小于1表示系统能够实时处理语音，而大于1表示系统处理存在延迟。

2. 国内外研究现状

2.1 研究方向及进展

在训练数据充足的条件下，端到端的语音识别模型更具优势。百度推出的DeepSpeech[14,15]模型在超过一万小时的语音数据上训练了一个深层CTC模型，获得了优异表现。谷歌也使用了超过一万小时的语音数据训练了LAS[16,17]模型以应用于自家语音搜索任务中，并成为了当时的SOTA。此外，考虑到端到端模型构建简单，现已成为学术界和工业界关注的重点方向。传统的端到端的语音识别模型基于RNN或其变体长短时记忆网络（Long Short Term Memory Network，LSTM）构建，而RNN等存在无法并行输入等问题。后续谷歌公司提出的Transformer模型超越了RNN的局限性，通过多头注意力机制直接实现序列到序列的建模。Speech-Transformer模型则是Transformer在语音识别领域的体现，后续为了加强对局部特征的建模而加入了CNN，两者相结合构成了Conformer模型，它综合了Transformer和CNN的优势，能够对音频序列的全局依赖性和局部特征进行建模，在LibriSpeech数据集上取得了当时最先进的结果。最近，许多优秀的端到端语音识别系统如

Espnet[18]、Wenet[19]和SpeechBrain[20]被提出，极大推动了语音识别的发展。同时，对Conformer架构的改进也还在继续，诸如Squeezeformer[21]、Branchformer[22]，E-Branchformer[23]等改进架构都对Conformer进行提升，进一步提高了语音识别的准确率。

2.2 存在问题

现在主流的端到端语音识别模型更适合应用于数据资源充足的语音识别任务中，其原因在于端到端的语音识别模型中的参数的更新需要梯度下降算法进行，这一过程是由数据驱动的，而且这一过程容易受到模型训练机制和数据包含信息的影响，一般认为这一性质可以通过大量的训练数据进行弥补，但是在方言识别领域，方言的样本量往往不够充足，不足以支撑大数据量的训练。如何处理现有方言中的口音以及噪声也是一个问题。此外，方言作为一门语言并不是一成不变的，它们会随着时间和空间发生变化[24]，这也给方言的识别带来了巨大的困难。同时方言具有极强的地域性，以浙江方言为例，在浙江之外的地方便很少能够接触，这也导致此前关于浙江方言的研究寥寥无几，这也增加了进行相关研究的难度。

此外，有部分研究者尝试使用迁移学习方法处理方言识别问题。但是迁移学习方法也存在一定缺陷：迁移学习的成功很大程度上取决于原始模型与目标任务之间的相似性，但是不同语言之间可能存在显著的语音差异，包括语音、语调、发音方式等，如果原始模型未能捕捉到这些差异，迁移后的模型可能无法有效识别目标方言。

3. 研究展望

我们尝试通过构建浙江方言数据集的方式来解决目前的方言资源缺少的问题。具体而言，我们将采用 OCR 技术处理大量方言相关视频，并通过对齐文本和音频获得大量方言数据。此外，我们将根据浙江方言的特点，精心挑选适合浙江方言的语音识别模型。通过将两种方法相结合，来尝试解决浙江方言的语音识别问题。

4. 参考文献

[1] 邢福义; 汪国胜 (编). 现代汉语（第 2 版）

[2] 游汝杰. 吴语方言学. 上海：上海教育出版社.

[3] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua

Bengio. Attention-based models for speech recognition.Advances in neural information processing systems, 28, 2015.

[4] Davis S B . Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences[J]. IEEE Trans. Acoust. Speech Signal Process. 1980, 28(4):65-74.

[5]Hermansky, Hynek. Perceptual linear predictive (PLP) analysis of speech[J]. The Journal of the Acoustical Society of America, 1990, 87(4):1738.

[6] Bourlard H, Morgan N. Continuous speech recognition by connectionist statistical methods[J]. IEEE Trans Neural Netw, 1993, 4(6): 893-909.

[7] Yao K, Dong Y, Seide F, et al. Adaptation of context-dependent deep neural networks for automatic speech recognition[C]. 2012 IEEE Spoken Language Technology Workshop (SLT). IEEE, 2012: 366-369.

[8]蒋竺芳.端到端自动语音识别技术研究[D]. 北京: 北京邮电大学, 2019.

[9]Graves A, Fernández S, Gomez F, et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks[C]//Proceedings of the 23rd international conference on Machine learning. 2006: 369-376.

[10] Schneider S, Baevski A, Collobert R, et al. wav2vec: Unsupervised pre-training for speech recognition[J]. arXiv preprint arXiv:1904.05862, 2019.

[11]Waswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//NIPS. 2017.

[12]Dong L, Xu S, Xu B. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition[C]//2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2018: 5884-5888.

[13]Gulati A, Qin J, Chiu C C, et al. Conformer: Convolution-augmented transformer for speech recognition[J]. arXiv preprint arXiv:2005.08100, 2020.

[14]Hannun A, Case C, Casper J, et al. Deep speech: Scaling up end-to-end speech recognition[J]. arXiv preprint arXiv:1412.5567, 2014.

[15]Amodei D, Ananthanarayanan S, Anubhai R, et al. Deep speech 2: End-to-end speech recognition in english and mandarin[C]//International conference on machine learning. PMLR, 2016: 173-182.

[16]Chan W, Jaitly N, Le Q, et al. Listen, attend and spell: A neural network for large vocabulary

conversational speech recognition[C]//2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2016: 4960-4964.

[17]Chiu C C, Sainath T N, Wu Y, et al. State-of-the-art speech recognition with sequence-to-sequence models[C]//2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2018: 4774-4778.

[18]Watanabe S, Hori T, Karita S, et al. Espnet: End-to-end speech processing toolkit[J]. arXiv preprint arXiv:1804.00015, 2018.

[19]Zhang B, Wu D, Yang C, et al. Wenet: Production first and production ready end-to-end speech recognition toolkit[J]. arXiv e-prints, 2021: arXiv: 2102.01547.

[20]Ravanelli M, Parcollet T, Plantinga P, et al. SpeechBrain: A general-purpose speech toolkit[J]. arXiv preprint arXiv:2106.04624, 2021.

[21]Sehoon Kim, Amir Gholami, Albert Shaw, Nicholas Lee, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, and Kurt Keutzer. Squeezeformer: An efficient transformer for automatic speech recognition. Advances in Neural Information Processing Systems, 35:9361–9373, 2022.

[22]Peng Y, Dalmia S, Lane I, et al. Branchformer: Parallel mlp-attention architectures to capture local and global context for speech recognition and understanding[C]//International Conference on Machine Learning. PMLR, 2022: 17627-17643.

[23] Kwangyoun Kim, Felix Wu, Yifan Peng, Jing Pan, Prashant Sridhar, Kyu J Han, and Shinji Watanabe. E-branchformer: Branchformer with enhanced merging for speech recognition. In 2022 IEEE Spoken Language Technology Workshop (SLT), pp. 84–91. IEEE, 2023

[24]Etman A, Beex A A L. Language and dialect identification: A survey[C]//2015 SAI intelligent systems conference (IntelliSys). IEEE, 2015: 220-231.

二、开题报告

1. 问题提出的背景

1.1 背景介绍

我国地大物博，人口众多，素有“十里不同天，百里不同俗”之说。与此相对应，我国方言文化也极其繁荣。以浙江地区为例，就有吴语、闽语、徽语、畲话等数种不同方言。同时我国历史悠久，在漫长的文明历程中，方言不仅是人与人之间交流的重要工具，也承载了中华民族宝贵的文化积淀。随着普通话的逐渐推广，方言的使用范围逐渐萎缩，越来越少的人能够听懂方言，更不用说可以讲述方言了。因此，在新的时代背景下，如何做好方言文化保护，收集现有文化资源，让传承千年的方言文化在当下焕发生机，历久弥新就成了时代的重大命题。

语音识别技术可以将语音识别成人类能够理解的文本，这一特性在方言保护以及传承上可以发挥巨大的作用。语音识别技术发展至今，从最初的基于高斯混合模型-隐马尔可夫模型[1]，到深度学习兴起时的深度神经网络-隐马尔科夫模型[2]，再到如今各种基于Conformer的端到端模型[3,4]，可谓是百花齐放。在众多的模型中，端到端模型由于其结构简单，效果卓越而受到了学术界和工业界的追捧。在本文中，我们将从收集方言语料出发，利用OCR相关技术构建方言数据集，并使用先进的端到端模型对进行训练推理，助力方言在这个时代的保护和传播。

1.2 本研究的意义和目的

方言文化在许多方面都有不可忽视的价值。在文化方面，传承千年的方言文化孕育了不可胜数的经典，在文化传承上，也有接续文脉，开拓创新的功用；在经济方面，体现地方特色，推动当地发展，拓而远之，有利于不同地区加入地域大融合，促进共同繁荣；在个人权益方面，可以减少方言使用者的日常生活障碍，维护社会稳定。而本研究旨在立足浙江方言文化，通过收集语料，构建浙江方言数据集，发展创新相关深度学习语音识别模型，来提升方言识别的准确率，乃至进一步进行语音生成相关工作，发扬浙江方言，保护地方文化，推动方言识别发展，并将相关模型方法进一步推广，运用于更多方言之上，为这一领域做出贡献。

2. 论文的主要内容和技术路线

2.1 主要研究内容

本研究的主要研究内容是基于浙江方言的语音识别。研究主要分成两个部分，首先是方言数据集的制作，其次是语音识别模型的选择。这两个部分分别存在如下的挑战，在方言数据集方面，需要处理数据量少的问题。端到端的模型往往是数据驱动的，需要较大量的数据才能获得比较好的识别准确率。但是方言往往局限于某一个特定地理范畴，数据量常常达不到要求，所以如何处理数据量不足的问题，适应模型训练的需要，是本研究面临的第一个问题。

在模型方面，如何选择一个适合浙江方言的语音识别模型？我们挑选了近期被提出的，优秀的语音识别，并将基于我们构建的数据集进行比较，最终得出一个合适的语音识别模型。我们的挑选的语音识别模型如下：

Model	Code	Conference
Zipformer	https://github.com/zipformer	ICLR2024
Fast Conformer	https://github.com/FastConformer	ASRU2023
E-Branchformer	https://github.com/e-branchformer	SLT2022
Squeezeformer	https://github.com/squeezeformer	Interspeech2022
Branchformer	https://github.com/branchformer	ICML2022
Paraformer	https://github.com/paraformer	Insterspeech2022
Blockformer	https://github.com/blockformer	ICASSP2022
3M	https://github.com/3m-asr	ISCSLP2022
MFCCA	https://github.com/mfcca	SLT2022
ImportantAug	https://github.com/importantAug	ICASSP2022
Wav2vec 2.0	https://github.com/wav2vec2.0	NIPS2020
Conformer	https://github.com/conformer	Interspeech2020

2.2 技术路线

在方言数据集制作方面，本研究采用光学字符识别（Optical Character Recognition, OCR）技术处理海量方言视频文件，通过字幕识别以及音频分割获取大量方言数据。同时进行方言文本和音频文件的对齐工作。对于视频中的额外信息（如滚动插入的广告等），设计相应过滤模块进行无关信息过滤。通过这种方式，我们可以自动生成方言数据，从而弥补了方言研究面对的低资源问题。

在选择模型方面，我们将基于上文描述的表格进行模型的挑选，并使用我们构建的数据集作为衡量的标准。通过选择适合浙江方言的语音识别模型，来提高浙江方言的识别准确率。

2.3 可行性分析

以往的方言语音识别研究往往囿于数据量不足，导致最终模型的识别准确率偏低。但是当今互联网时代，网上音视频资源丰富，而且在数据驱动下端到端的语音识别模型能够发挥出极佳的效果，这就为我们的方言识别研究铺平了道路。同时，OCR 技术已经较为成熟，这就为大量获取相关数据资源提供了可能性。

当前主流语音识别模型都基于 Conformer 架构进行设计，Conformer 综合了 Transformer 和 CNN 的优点，能够兼顾局部和全局的依赖性。基于这样的特性，就为方言识别提供了理论上的可行性。

3. 研究计划进度安排及预期目标

3.1 进度安排

在三月份进行背景资料收集和相关文献阅读，设计实验方案。四月份进行实验数据集构建，并设计相应语音识别模型，进行代码编写，完成实验并对比实验效果，五月份进行毕业论文撰写以及毕业论文答辩。

3.2 预期目标

本研究预计构建一个浙江方言数据集，并基于该数据集挑选最适合浙江方言的语音识别模型。通过在多种不同语音识别模型间进行对比，进行原因分析，总结高识别率的特点，为浙江方言相关研究作出贡献。

4. 参考文献

[1] Bourlard H, Morgan N. Continuous speech recognition by connectionist statistical methods[J]. IEEE Trans Neural Netw, 1993, 4(6): 893-909.

[2] Yao K, Dong Y, Seide F, et al. Adaptation of context-dependent deep neural networks for automatic speech recognition[C]. 2012 IEEE Spoken Language Technology Workshop (SLT). IEEE, 2012: 366-369.

[3]Gulati A, Qin J, Chiu C C, et al. Conformer: Convolution-augmented transformer for speech recognition[J]. arXiv preprint arXiv:2005.08100, 2020.

[4]Sehoon Kim, Amir Gholami, Albert Shaw, Nicholas Lee, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, and Kurt Keutzer. Squeezeformer: An efficient transformer for automatic speech recognition. Advances in Neural Information Processing Systems, 35:9361–9373, 2022.

[5]Graves A, Fernández S, Gomez F, et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks[C]//Proceedings of the 23rd international conference on Machine learning. 2006: 369-376.

[6] Kwangyoun Kim, Felix Wu, Yifan Peng, Jing Pan, Prashant Sridhar, Kyu J Han, and Shinji Watanabe. E-branchformer: Branchformer with enhanced merging for speech recognition. In 2022 IEEE Spoken Language Technology Workshop (SLT), pp. 84–91. IEEE, 2023

[7]Waswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//NIPS. 2017.

三、外文翻译

ZIPFORMER：一种更快更好的自动语音识别器

Zengwei Yao，Liyong Guo，Xiaoyu Yang，Wei Kang，Fangjun Kuang，
Yifan Yang，Zengrui Jin，Long Lin，Daniel Povey
Xiaomi Corp.，Beijing，China
ICLR2024

0.摘要

Conformer 已经成为了最受欢迎的自动语音识别编码模型，它向 Transformer 中添加了卷积模块来学习局部和全局性的依赖。在这篇文章中，我们描述了 Zipformer，它是一个更快，内存效率更高同时性能更好的 Transformer。模型的变化包括：1）一个类 U-Net 的编码器架构，其中中间栈以一个较低的帧率运行；2）用更多的模块重组块结构，并在其中重用注意力权重以提高效率；3）BiasNorm 作为一种修改过的 LayerNorm 允许我们保持一些长度信息；4）新的激活函数 Swooshr 和 SwooshL，它们的表现优于 Swish。我们还提出了一种新的优化器，叫做 ScaleAdam。它通过每个张量的当前比例来缩放更新以保持相对变化的大致相同，同时它显示的学习参数比例。相比 Adam，ScaleAdam 实现了更快的收敛和更好的性能。基于 LibriSpeech，Aishell-1 和 WenetSpeech 等数据集的大量实验证明了我们提出的 Zipformer 相比其他先进自动语音识别模型的有效性。

1.简介

端到端模型已经在语音自动识别领域取得了巨大的成功。在端到端的自动语音识别中，一个有效的编码器架构在语音序列的时间建模中发挥了重要的作用。一个突出的例子是 Conformer（Gulati et al.,2020），它结合了卷积神经网络（CNN）模型（Zhang et al.,2017;Li et al.,2019;Kriman et al.,2020）和 Transformer 模型（Dong et al.,2018;Karita et al.,2019;Zhang et al.,2020b）。通过将 CNN 集成到 Transformer（Vaswani et al.,2017）中，Conformer 能够提取局部和全局的语音序列的依赖关系，并且实现自动语音识别的最先进的性能。

在这项工作中，我们提出了一种更快，更节省内存并且性能更好的 Transformer 作为自动语音识别的编码器，其名为 Zipformer。首先，不同于在恒定帧率上操作语音序列的 Conformer，Zipformer 采用了类 U-Net 架构（Ronneberger et al.,2015），它包含多个堆叠，将语音序列下采样到不同的更低的帧率。其次，我们重新设计了模块结构，配备了更

多的模块，比如我们配备了两个 Conformer 模块。同时重复使用注意力权重来提高效率。我们提出了 BiasNorm 作为 LayerNorm 的一个更简单的替代，这允许我们在规范化中保留长度信息。我们还使用心得激活函数 SwooshR 和 SwooshL 替代了 Swish 来取得更好的效果。此外，我们设计了 ScaledAdam 作为 Adam 的参数尺度不变的版本，它通过当前参数尺度来缩放更新，还明确地学习参数尺度。与 Adam 相比，ScaledAdam 能够更快的收敛并且拥有更好的性能。

在 LibriSpeech，Aishell 和 WenetSpeech 等数据集上进行的大量实验证明了我们所提出的模型和相关创新优化的有效性。Zipformer 在所有三个数据集上均取得了最先进的结果。值得一提的是，Zipformer 是第一个在 LibriSpeech 数据集上取得与 Conformer 论文报告结果相媲美的结果的模型（这些结果对于其他人而言是难以复现的）。在效率方面，Zipformer 相较之前的研究在训练过程中收敛速度更快，推断速度则提高了 50%以上，同时在 GPU 内存上的需求更小。我们进行了详细的消融研究，以调查各个组件的贡献。

2. 相关工作

模型架构。深度卷积架构已经被应用到端到端的自动语音识别中(Zhang et al.，2017; Li et al.，2019)。后续研究通过使用深度可分离卷积（Howard et al.,2017）来提高效率（Kriman et al.,2020），结合了压缩激发模块（Hu et al.,2018）来捕获更长的上下文（Han et al.,2020）。受到 Transformer（Vaswani et al.，2017）在自然语言处理领域的成功的启发，一些研究将 Transformer 应用到语音应用中（Dong et al.，2018; Karita et al.，2019; Zhang et al.，2020b; Wang et al.，2020; Zhang et al.，2020a）。与 CNN 相比，Transformer 的优势在于它可以基于自注意力学习全局的依赖关系，而这对语音处理至关重要。通过将卷积整合到 Transformer 中，Conformer 获得了建模局部和全局上下文信息的强大能力，并超越了之间所有的自动语音识别模型。

近期的研究探索了 Conformer 架构上的变化，以进一步降低计算成本并提高识别性能。Squeezeformer（Kim et al.，2022）采用了时间 U-Net 架构，其中的中间模块一半的帧率运行，同时重新设计了块结构，使之类似于标准 Transformer（Vaswani et al.，2017）块。Branchformer（Peng et al.，2022）结合了并行分支来建模不同范围的上下文，其中一个分支利用卷积门控多层感知器（MLP）捕获本地上下文，而另一个分支利用自注意力学习长距离的依赖关系。E-Branchformer（Kim et al.，2023）通过基于卷积的模块进一步改进了 Branchformer，增强了分支合并机制。

Zipformer 与之前的工作 Squeezeformer 在时间下采样方面采用了类似的思路。然而，与 Squeezeformer 中的固定下采样比率相比，Zipformer 在不同的编码器堆栈中使用不同的下采样比率，并在中间编码器堆栈中采用了更为激进的下采样比率。除了建模上的差异，我们的工作还专注于包括新的优化器 ScaledAdam 在内的与优化相关的改变，实验证明这些改变有助于改善收敛性。

端到端框架。连接时序分类（CTC）（Graves et al.，2006）是最早的端到端的自动语音识别框架之一。但是其性能会受到帧独立的假设的限制。为此，提出了在 CTC 中集成基于注意力的编码器-解码器（AED）（Chan et al.，2015）的混合架构（CTC/AED），以改善性能（Watanabe et al.，2017）。神经传导器（Graves，2012），通常称为 RNN-T，通过使用标签解码器和联合网络来解决帧独立的假设，并因其优越的性能而成为一种流行的框架。最近，例如修剪（Kuang et al.，2022；Wang et al.，2023；Mahadeokar et al.，2021）或批次拆分（Kuchaiev et al.，2019）等方法被提出以加速神经传导器的训练速度并减少内存使用。

3. 方法

3.1 下采样编码架构

图 1 展示了我们所提出的 Zipformer 模型的总体架构。与以固定帧率 25Hz 处理序列的 Conformer（Gulati et al.,2020）不同，Zipformer 以更有效的方式使用类似 U-Net 的结构在不同分辨率上学习时间表示。具体而言，给定帧率为 100Hz 的声学特征，名为 Conv-Embed 的基于卷积的模块首先以 2 的倍数减少长度，得到 50Hz 的嵌入序列。然后，获得的序列被送入 6 个级联堆栈，分别以 50Hz、25Hz、12.5Hz、6.25Hz、12.5Hz 和 25Hz 的帧率学习时间表示。除了第一个堆栈外，其他堆栈都采用了下采样结构，以更低的帧率处理序列。堆栈之间的帧率始终为 50Hz。不同的堆栈具有不同的嵌入维度，而中间堆栈具有较大的维度。每个堆栈的输出被截断或填充为零，以匹配下一个堆栈的维度。最终编码器输出维度被设置为所有堆栈维度的最大值。具体而言，如果最后一个堆栈的输出具有最大的维度，则将其作为编码器的输出；否则，它将从不同堆栈输出的各个部分连接而成，每个维度都从最近的具有该维度的输出中提取。最后，一个下采样模块将序列转换为 25Hz，得到编码器的输出。

在 Conv-Embed 模块中，我们使用了三个二维卷积层，其时间 $\times$ 频率步长分别为 1 $\times$ 2、2 $\times$ 2 和 1 $\times$ 2，输出通道分别为 8、32 和 128。随后，我们利用了一个类似

于 Nextformer（Liu et al.，2022）中的 ConvNeXt 层（Jiang et al.，2022），它由一个尺寸为 7 × 7 的深度可分离卷积、一个输出通道数为 384 的逐点卷积、一个 SwooshL 激活函数(在第 3.4 节中描述)、以及一个输出通道数为 128 的逐点卷积组成。在 ConvNeXt 模块上应用了残差连接。最后，我们使用了一个线性层，后跟 BiasNorm（在第 3.3 节中描述）来调整特征维度以匹配第一个堆栈。

下采样堆栈。在下采样堆栈中，成对的 Downsample 和 Upsample 模块通过一种几乎是最简单的方法执行对称的序列长度缩放和放大操作。例如，对于因子为 2，Downsample 模块对每 2 个帧进行平均，使用 2 个可学习的标量权重（经过 softmax 标准化），而 Upsample 模块只是简单地将每个帧重复两次。在下采样后，它使用堆叠的 Zipformer 块（在第 3.2 节中描述）对更低帧率进行时间建模。最后，它利用 Bypass 模块（在第 3.2 节中描述）以一种可学习的方式组合堆栈输入和堆栈输出。

3.2 Zipformer 块

Conformer 块由四个模块组成：前馈（feed-forward)、多头自注意力（MHSA)、卷积以及前馈（feed-forward)。多头自注意力通过两个步骤学习全局上下文：使用点积操作计算注意力权重，并利用这些注意力权重聚合不同帧的信息。然而，多头自注意力通常会带来较大的计算成本，因为上述两个步骤对于序列长度都需要二次复杂度。因此，我们按照上述两个步骤将多头自注意力分解为两个单独的模块：多头注意力权重（MHAW）和自注意力（SA)。这种改变允许在每个块中通过使用一个 MHAW 模块和两个 SA 模块两倍更高效地执行注意力计算。另外，我们提出了一个称为非线性注意力（NLA）的新模块，以充分利用计算得到的注意力权重来捕获全局信息。

正如图 2（左侧）所示，Zipformer 块的深度大约是 Conformer 块的两倍（Gulati et al.，2020)。主要目的是为了允许重复利用注意力权重以节省时间和内存。具体来说，块输入首先传入一个 MHAW 模块，该模块计算注意力权重并与一个 NLA 模块和两个 SA 模块共享这些权重。同时，块输入还经过一个前馈模块，然后是 NLA 模块。

然后，它应用两个由 SA、卷积和前馈组成的模块组。最后，使用 BiasNorm（在第 3.3 节中描述）来对块输出进行归一化。除了使用加法操作进行常规残差连接外，每个块还利用两个 Bypass 模块来结合块输入和模块输出，分别放置在块的中间和末尾。与常规的 Transformer 模型(Vaswani et al.，2017)不同，我们不会为每个模块使用诸如 LayerNorm（Ba et al.，2016）之类的归一化层来周期性地防止激活变得过大或过小，因为我们提出的 ScaledAdam 优化器能够学习参数的缩放（在第 3.5 节中描述)。

非线性注意力。图 2（右）展示了 NLA 的结构。它还利用了从 MHAW 预先计算的注意力权重来沿着时间轴聚合嵌入向量，这类似于 SA。它首先使用 3 个线性层将输入投影到 A、B 和 C，每个层的维度为输入维度的 3/4。模块的输出为 linear(A ⊙ attention(tanh(B) ⊙ C))，其中 ⊙ 表示逐元素乘法，attention 表示在时间轴上通过先前计算的单个注意力权重进行矩阵乘法，线性层将维度恢复为与输入相同。

旁路。旁路模块学习通道方向的标量权重 c，用于组合模块输入 x 和模块输出 y：(1 - c) ⊙ x + c ⊙ y。在训练中，我们最初将 c 的值限制在[0.9, 1.0]的范围内，然后在进行了 20000 步之后将最小值更改为 0.2。我们发现，在开始时使模块“直通”（即几乎不进行旁路）有助于模型的收敛。

3.3 BiasNorm

Conformer（Gulati 等，2020）利用 LayerNorm（Ba 等，2016）来归一化模块的激活。对于具有 D 个通道的 x，LayerNorm 的公式如下：

$$\mathrm{LayerNorm(x)} = \frac{\mathrm{x} - \mathrm{E[x]}}{\sqrt{(\mathrm{Var}[x] + \epsilon)}} \odot \gamma + \beta$$

具体地，它首先计算均值 E[x]和标准差$\sqrt{\mathrm{Var}[\mathrm{x}]}$以进行归一化，将向量长度调整为$\sqrt{\mathrm{D}}$。然后它使用可学习的通道方式尺度$\gamma$和偏置$\beta$进行转换，这有助于调整激活的大小并平衡特定模块的相对贡献。然而，我们观察到，使用 LayerNorm 训练的 Conformer 存在两个问题：1）有时会将一个通道设置为大的常数值，例如 50。我们认为它旨在“击败”LayerNorm，因为 LayerNorm 会完全消除向量长度，所以它的作用就像是一个非常大的值，以便在归一化后保留长度信息。2）一些模块（通常是前馈或卷积）“失效”，因为它们的输出值极小，例如10^{-6}。我们认为在训练初期，未经训练的模块无效，因此它们被 LayerNorm 的尺度γ接近零“关闭”。如果尺度γ围绕零振荡，不一致的符号会不断改变反向传播到模块的梯度方向。由于梯度符号的不一致性，这些模块永远无法学到任何有用的东西，因为这是一个很难逃离的坏的局部最优解，这是由于类似随机梯度下降的动态更新的动态性导致的。

为了解决上述问题，我们提出了 BiasNorm，它旨在成为 LayerNorm 的更简单的替代方案。BiasNorm 的公式如下：

$$\mathrm{BiasNorm}(x) = \frac{x}{\mathrm{RMS}[x - b]} \cdot \exp(\gamma)$$

其中，b 是可学习的按通道计算的偏置，RMS[x - b]是计算通道上的均方根值，而 γ 是一个标量。我们首先去除均值减法的操作，因为除非其后跟着非线性操作，否则这样做是浪

费时间。偏置 b 充当大常数值，允许在归一化后保留向量长度信息。由于尺度 exp(γ)始终为正值，它避免了梯度振荡问题。

3.4 SwooshR 和 SwooshL 激活函数

Conformer（Gulati et al.，2020）采用了 Swish（Ramachandran 等，2017）激活函数，其公式如下：

$$\mathrm{Swish}(x) = x \cdot (1 + \exp(-x))^{-1}$$

在这项工作中，我们提出了两个新的激活函数，分别称为 SwooshR 和 SwooshL，作为 Swish 的替代品：

$$\mathrm{SwooshR}(x) = \log(1 + \exp(x - 1)) - 0.08x - 0.313261687$$

$$\mathrm{SwooshL}(x) = \log(1 + \exp(x - 4)) - 0.08x - 0.035$$

在 SwooshR 中，偏移量 0.313261687 是为了使其通过原点；在 SwooshL 中，调整了偏移量 0.035，它略微优于让曲线通过原点的确切值。我们在附录 A.2 节中给出了 Swish、SwooshR 和 SwooshL 的曲线。SwooshL 大致是 SwooshR 的右移版本。需要注意的是，后缀“L”或“R”表示左零点或右零点在或接近 x = 0。与 Swish 类似，SwooshR 和 SwooshL 有下界，并且都是非单调的。与 Swish 相比，最显著的区别在于 SwooshR 和 SwooshL 对负输入具有非零斜率,这有助于避免输入始终为负并防止 Adam 类型更新的分母变得过小。当用 SwooshR 替换 Swish 时，我们观察到具有旁路连接的模块，例如前馈和 ConvNeXt，倾向于在先前的线性层中学习大的负偏置以学习“正常关闭”行为。因此，我们对于这些“正常关闭”的模块使用 SwooshL，对卷积模块和其余的 Conv-Embed 使用 SwooshR。

3.5 ScaledAdam 优化器

我们提出了 Adam（Kingma & Ba, 2014）的参数尺度不变版本，称为 ScaledAdam。ScaledAdam 的收敛速度更快，性能更好。ScaledAdam 将每个参数的更新与该参数的尺度成比例地调整，并明确学习参数的尺度。附录部分 A.1.1 中的算法 1 呈现了 ScaledAdam 的伪代码。

设 f(θ) 为我们的损失函数，我们的目标是最小化它，该函数对可学习参数 θ 可微。在每一步 t，Adam 计算参数梯度 $\mathrm{g_t} = \nabla_\theta \mathrm{f}(\theta_{t-1})$，并更新梯度的一阶矩 $\mathrm{m_t} = \beta_1 \cdot m_t - 1 + (1-\beta_1)\cdot g_t$ 和二阶矩 $\mathrm{v_t} = \beta_2 \cdot v_t - 1 + (1-\beta_2)\cdot g_t^2$，其中 $\beta 1, \beta 2 \in [0,1)$ 是用于计算移动平均的系数。第 t 步的参数更新 Δt 定义为：

$$\Delta t = \alpha_t \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}$$

其中α_t 通常由外部调度指定的学习率，$\frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t}$ 是校正项，$\epsilon = 10^{-8}$。

虽然 Adam 对每个参数的梯度尺度是不变的，我们认为它仍然存在两个限制：1）方程 5 中的更新Δ_t 没有考虑参数尺度（记为r_{t-1}）。考虑相对参数变化Δ_t/r_{t-1}，Adam 可能导致具有大尺度参数的相对学习速度过慢，或具有小尺度参数的相对学习速度过快。2）直接学习参数尺度很困难，因为在高维空间中，增加或减小参数张量的方向是非常特定的。特别是减小参数尤其困难，因为每个梯度步骤 g_t 都会导致参数范数的增长。

为了保持各种尺度参数的相对变化 Δ_t/r_{t-1} 大致相同，我们通过参数尺度 r_{t-1} 对方程中的更新 Δt 进行了缩放：

$$\Delta_t^{\prime} = \alpha_t \cdot r_{t-1} \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \frac{m_t}{\sqrt{v_t}+\epsilon}$$

我们计算参数尺度 r_{t-1} 作为均方根值$RMS[\theta t-1]$。由于 ScaledAdam 的更新比 Adam 更不容易发散，我们使用一个名为 Eden 的学习率调度，它不需要长时间的热身期；我们还使用绝对较大的学习率值，因为参数的RMS值通常远小于 1。

学习参数尺度。为了明确学习参数尺度，我们将其视为待学习的常规参数，就像将每个参数都分解为 $\theta = r \cdot \theta'$，我们对参数尺度r和底层参数θ'进行梯度下降优化。设h为参数尺度 r 的梯度，在第 t 步，我们得到 $h_t = \nabla_r f(\theta_{t-1}) = g_t \cdot \theta'_{t-1}$。由于 Adam 对梯度尺度的变化几乎是不变的，为简单起见，我们将其替换为$h_t = g_t \cdot (r_{t-1} \odot \theta'_{t-1}) = g_t \cdot \theta_{t-1}$。遵循 Adam 算法，我们维护尺度梯度的第一时刻 $n_t = \beta_1 \cdot n_{t-1} + (1-\beta_1) \cdot h_t$ 和第二时刻 $w_t = \beta_2 \cdot w_{t-1} + (1-\beta_2) \cdot h_t^2$。由于更新参数尺度从 r_{t-1} 到r_t 引起的参数 θ 的变化量为 $\Delta'_t, r = (r_t - r_{t-1}) \odot \theta'_{t-1}$。我们还将参数尺度 r_{t-1} 整合到更新Δ'_t, r中：

$$\Delta_{t}^{\prime}, r = \eta \cdot \alpha_t \cdot r_{t-1} \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \sqrt{w_t} + \epsilon \odot \theta'_{t-1} = \eta \cdot \alpha_t \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \sqrt{w_t} + \epsilon \odot \theta_{t-1}$$

其中 η 是学习率 α_t 的缩放因子，我们发现设定 $\eta = 0.1$ 有助于稳定训练。现在更新 Δ'_t 被 $\Delta'_{t,\ r} + \Delta'_t$取代，这相当于在每个参数的变化方向上添加了额外的梯度项。这也可以通过简化网络结构来除去我们 Zipformer Block（第 3.2 节中描述）中的大多数归一化层，因为现在模块可以轻松学习将激活值按合适的范围进行缩放。一种类似的方法称为权重归一化 (Salimans & Kingma, 2016) 将参数范数与其方向解耦以加速收敛。它用两个参数替换每个参数，分别指定方向和大小。然而，ScaledAdam 通过添加额外的更新项 $\Delta'_{t,r}$ 来学习参数尺度，这使得编写建模代码更加简单。

这里提出的 Eden 学习率调度被公式化为：

$$\alpha_t = \alpha_{\text{base}} \cdot \frac{t^{\frac{2}{\alpha_{\text{step}}}}}{\alpha_{\text{step}}^{\frac{2}{\alpha_{\text{step}}}}} \cdot \frac{e^{\frac{2}{\alpha_{\text{epoch}}}}}{\alpha_{\text{epoch}}^{\frac{2}{\alpha_{\text{epoch}}}}} \cdot \text{linear}(\alpha_{\text{start}}, t_{\text{warmup}}, t).$$

其中，t是步骤索引，e是纪元索引，α_{step} 和 α_{epoch} 控制开始显著降低学习率的步数和纪元数，$linear(\alpha_{\text{start}}, t_{\text{warmup}}, t)$ 是一个暖身比例，它在 t_{warmup}步骤内从 α_{start}线性增加到 1，然后保持恒定在 1。 α_{base}是在设置 $\alpha_{\text{start}} = 1, \alpha_{\text{warmup}} = 0$ 时的最大值。将 Eden 学习率调度设计为依赖步骤索引t和纪元索引e的原因是为了在改变批量大小后保持在一定量的训练数据（例如，一个小时）之后的参数变化量大致恒定，因此如果我们改变批处理大小，调度参数就不需要重新调整。Eden 的其他版本用一些合适的数据量测来替换公式中的“纪元”部分。在这项工作中，我们使用了$\alpha_{\text{base}} = 0.045, \alpha_{\text{start}} = 0.5$, 和 $t_{\text{warmup}} = 500$。

高效的实现方法。为了加速在 ScaledAdam 中的优化，我们根据其形状将参数分批，并逐批进行计算。注意这不会影响结果。ScaledAdam 只是需要比 Adam 稍多一点的内存来缓存用于参数尺度的梯度时刻n_t和w_t。

4. 实验

4.1 实验设置

结构变体。我们使用三种模型规模来构建我们的 Zipformer 变种：小型（Zipformer-S），中型（Zipformer-M）和大型（Zipformer-L）。对于 6 个编码器堆栈，注意力头的数量设置为{4, 4, 4, 8, 4, 4}，卷积核大小设置为{31, 31, 15, 15, 15, 31}。在每个注意力头中，查询维度和值维度分别设置为 32 和 12。对于每个 Zipformer 块中的三个前馈模块，第一个和最后一个中的隐藏维度分别为中间隐藏维度的 3/4 和 5/4。我们调整每个堆栈中的中间前馈的层数、嵌入维度和隐藏维度以获得不同的模型规模。

数据集。我们进行实验，将我们的 Zipformer 与其他最先进的模型在三个开源数据集上进行比较：1) LibriSpeech（Panayotov et al., 2015），包含约 1000 小时的英文有声读物；2) Aishell-1（Bu et al., 2017），包含 170 小时的中文语音；3) WenetSpeech（Zhang et al., 2022a），包含 10000 多小时的多领域的中文语音。

实现细节。我们使用 Lhotse（Zelasko et al.，2021）工具包来准备语音数据。模型的输入是在 25ms 帧上提取的 80 维 Mel 滤波器组特征，帧移为 10ms。我们使用速度扰动（Ko et al.，2015），其中扰动因子分别为 0.9、1.0 和 1.1 来增强训练数据。在训练过程中还应用了 SpecAugment（Park et al.，2019）。我们为 Zipformer 模型使用混合精度训练。此外，我们还采用激活约束，包括 Balancer 和 Whitener，以确保训练的一致性和稳定性。Balancer 和 Whitener 的详细信息见附录 A.3。我们使用剪枝的转录器（Kuang et al.，2022），这是转录损失的高效内存版本，用于剪枝具有较小后验概率的路径，作为训练目标。在解码过程中，采用大小为 4 的波束搜索，限制每帧最多发射一个符号（Kang et al.，2023）。我们不使用外部语言模型进行重新评分，因为在这项工作中我们专注于改进编码器模型。我们分别使用单词错误率（WER）和字符错误率（CER）作为英语和中文数据集的评估指标。默认情况下，我们所有的模型都是在 32GB 的 NVIDIA Tesla V100 GPU 上训练的。对于 Librispeech 数据集，Zipformer-M 和 Zipformer-L 在 4 个 GPU 上训练 50 个 epochs，Zipformer-S 在 2 个 GPU 上训练 50 个 epochs。对于 Aishell-1 数据集，我们的模型在 2 个 GPU 上训练 56 个 epochs。对于 WenetSpeech 数据集，我们的模型在 4 个 GPU 上训练 14 个 epochs。

4.2 与 SOTA 模型的比较

在这一部分，我们用 Zipformer 模型与其他最先进的模型进行了比较。

LibriSpeech 数据集。表 2 显示了 Zipformer 和其他最先进模型在 LibriSpeech 测试数据集上的结果。对于 Conformer 模型，我们还列出了我们复现的 WER 以及其他开源框架上的结果。需要注意的是，开源复现的 Conformer 模型与原始 Conformer 模型之间存在性能差距。我们的 Zipformer-S 模型在拥有更少参数和浮点运算(FLOPs)的情况下，实现了比 Squeezeformer 所有变种更低的 WER。我们的 Zipformer-L 模型在节省 50%以上的 FLOPs 的同时，远远超过了 Squeezeformer-L、Branchformer 和我们复现的 Conformer-L 的性能。值得注意的是，当在 8 个 80G 的 NVIDIA Tesla A100 GPU 上进行了 170 个 epochs 的训练后,Zipformer-L 在有足够计算资源的情况下取得了 2.00%/4.38%的 WER，从我们所知，这是第一个接近 Conformer-L 的模型。我们还比较了提出的 Zipfomer 模型与其他最先进模型在速度和内存使用方面的差异。图 3 给出了在 NVIDIA Tesla V100 GPU 上对 30 秒音频批处理的平均推断时间和峰值内存使用的比较结果。批处理大小设置为 30，以确保所有模型在推断过程中不会出现内存不足的问题。总的来说，Zipformer 模型在性能和效率之间取得了更好的平衡。特别是对于大规模任务，

Zipformer-L 需要的计算时间和内存要比其他模型少得多。

Aishell-1 数据集。表 3 显示了在 Aishell-1 数据集上的 CER。与 ESPnet 工具包中实现的 Conformer 模型相比，我们的 Zipformer-S 在参数更少的情况下取得了更好的性能。扩大模型规模可以降低 WER，并且 Zipformer-M/L 的性能超过了所有其他模型。

WenetSpeech 数据集。表 4 呈现了在 WenetSpeech 数据集上的实验结果。同样，我们的 Zipformer-M 和 Zipformer-L 在 Test Net 和 Test Meeting 测试集上的表现均超过了所有其他模型。虽然仅用了三分之一的参数，但是我们的 Zipformer-S 的 WER 比 Conformer 模型更低。

4.3 消融研究

我们在 LibriSpeech 数据集上进行消融实验，以研究每项提出的功能技术的影响。以 Zipformer-M 作为基础模型，我们每次进行一个更改，同时保持其他部分不变。表 5 呈现了实验结果。

编码器结构。我们从 Zipformer 模型中移除了时间下采样结构，并使用了下采样率为 4 的 Conv-Embed，就像 Conformer 模型一样。得到的模型具有 12 个 Zipformer 块，恒定的嵌入维度为 512，并且比基础模型具有更多参数。实验结果表 5 显示，在没有下采样结构的模型中，在测试集上产生了更高的词错误率（WER）。这表明，为了效率而引入的时间下采样结构并没有导致信息丢失，而是在更少的参数的情况下加强了建模能力。

模块结构。由于每个 Zipformer 模块大致具有 Conformer 模块的两倍大，我们用两个叠放在一起的 Conformer 模块替换基础模型中的每个 Zipformer 模块。即使模型更大，测试集中的测试其他数据的绝对 WER 降低了 0.16%，表明了 Zipformer 模块结构的优势。移除 NLA 或 Bypass 中的任一部分会导致性能下降。如果我们在移除 NLA 后进一步移除注意力权重共享机制，模型的参数略多，推理速度较慢，但 WER 没有改善。我们假设 Zipformer 模块内的两个注意力权重是相当一致的，共享它们不会对模型造成伤害。

规范化层。在 Zipformer 中用 LayerNorm 替换 BiasNorm 会导致分别在测试清晰数据和测试其他数据上 WER 降低了 0.08% 和 0.18%，表明了提出的 BiasNorm 允许在规范化中保留一些长度信息的优势。

激活函数。当在 Zipformer 的所有模块中只使用 SwooshR 时，分别在测试清晰数据和测试其他数据上的 WER 分别下降了 0.11% 和 0.42%，验证了特别使用 SwooshL 用于

“通常关闭”模块的有效性。使用 Swish 会导致更多性能下降，这表明了 SwooshR 相对于 Swish 的优势。

优化器。当使用 Adam 训练 Zipformer 时，我们必须对 Zipformer 模块中的每个模块应用 BiasNorm，以避免模型发散，因为 Adam 无法学习调整模块激活的每个参数的比例，就像 ScaledAdam。我们分别尝试了不同的学习率因子（表示为 α_{base}）和 Adam（2.5、5.0、7.5、10.0）。根据（Gulati et al.，2020），Adam 的学习率调度为 $\alpha_t = \alpha_{base} \cdot 512 - 0.5 \cdot min(t^{-0.5}, t \cdot 10000^{-1.5})$。附录 A.1.2 节的图 A.2 显示了在不同的时期以及不同步骤的学习率上的测试清晰数据和测试其他数据的平均 WER。我们在表 5 中展示了使用 $\alpha\_\{base\} = 0.045$ 的 ScaledAdam 和使用 $\alpha_{base} = 7.5$的 Adam 的最佳结果。相比 Adam，ScaledAdam 在测试清晰数据和测试其他数据上的性能分别提高了 0.17% 和 0.72%，这些结果表明 ScaledAdam 能够比 Adam 实现更快的收敛和更好的性能。

5 结论

在这项工作中，我们提出了 Zipformer，它作为一种高效的自动语音识别（ASR）编码器。它具有类似 U-Net 的编码器结构，可以将序列下采样到不同的较低帧率。重新设计的块结构配备更多模块，可以重复使用计算得到的注意力权重以提高效率。它还采用了新的规范化方法 BiasNorm，以及新的激活函数 SwooshR 和 SwooshL。同时，提出的优化器 ScaledAdam 实现了更快的收敛和更好的性能。基于 LibriSpeech、Aishell-1 和 WenetSpeech 数据集的大量实验证明了我们提出的 Zipformer 的有效性。

四、外文原文

ZIPFORMER: A FASTER AND BETTER ENCODER FOR AUTOMATIC SPEECH RECOGNITION

Zengwei Yao, Liyong Guo, Xiaoyu Yang, Wei Kang, Fangjun Kuang,
Yifan Yang, Zengrui Jin, Long Lin, Daniel Povey
Xiaomi Corp., Beijing, China
ICLR2024

0 Abstract

The Conformer has become the most popular encoder model for automatic speech recognition (ASR). It adds convolution modules to a Transformer to learn both local and global dependencies. In this work we describe a faster, more memoryefficient, and better-performing Transformer, called Zipformer. Modeling changes include: 1) a U-Net-like encoder structure where middle stacks operate at lower frame rates; 2) reorganized block structure with more modules, within which we re-use attention weights for efficiency; 3) a modified form of LayerNorm called BiasNorm allows us to retain some length information; 4) new activation functions SwooshR and SwooshL work better than Swish. We also propose a new optimizer, called ScaledAdam, which scales the update by each tensor’s current scale to keep the relative change about the same, and also explictly learns the parameter scale. It achieves faster convergence and better performance than Adam. Extensive experiments on LibriSpeech, Aishell-1, and WenetSpeech datasets demonstrate the effectiveness of our proposed Zipformer over other state-of-the-art ASR models. Our code is publicly available at https://github.com/k2-fsa/icefall.

1 Introduction

End-to-end models have achieved remarkable success in automatic speech recognition (ASR). An effective encoder architecture that performs temporal modeling on the speech sequence plays a vital role in end-to-end ASR models. A most prominent example is Conformer (Gulati et al., 2020), which combines the advantages of the convolutional neural network (CNN) models (Zhang et al., 2017; Li et al., 2019; Kriman et al., 2020) and Transformer models (Dong et al., 2018; Karita et al., 2019; Zhang et al., 2020b). By integrating CNN into Transformer (Vaswani et al., 2017), Conformer is able to extract both local and global dependencies on speech sequences, and achieves state-of-theart performance in ASR.

In this work, we propose a faster, more memory-efficient, and better-performing Transformer as ASR encoder, called Zipformer. First, unlike Conformer that operates on the sequence at a constant frame rate, Zipformer adopts a U-Net-like (Ronneberger et al., 2015) structure, which consists of multiple stacks downsamping the sequence to various lower frame rates. Second, we re-design the block structure, which is equipped with more modules like two Conformer blocks, and reuses the attention weights for efficiency. We propose BiasNorm as a simpler replacement of LayerNorm, which allows for retaining length information in normalization. We also replace Swish with our new activation functions SwooshR and SwooshL to achieve better results. In addition, we devise a parameter-scale-invariant version of Adam, called ScaledAdam, which scales the update by the current parameter scale and also explicitly learns the parameter scale. Compared to Adam, ScaledAdam enables faster convergence and better performance.

Extensive experiments are conducted on LibriSpeech, Aishell-1, and WenetSpeech datasets, and results demonstrate the effectiveness of the proposed modeling and optimization-related innovations. Zipformer achieves state-of-the-art results on all three datasets. It is worth mentioning that Zipformer is the first model ever to achieve results comparable to those reported in the Conformer paper on the LibriSpeech dataset (these results have proved difficult for others to reproduce). In terms of efficiency, Zipformer converges faster during training and speeds up the inference by more than 50% compared to previous studies while requiring less GPU memory. We perform detailed ablation studies to investigate the contribution of individual components.

2 Related Work

Model architecture. Deep convolution architectures have been applied to end-to-end ASR (Zhang et al., 2017; Li et al., 2019). Follow-up works explore improvements by using depthwise separable convolutions (Howard et al., 2017) for efficiency (Kriman et al., 2020), and incorporating squeezeand-excitation module (Hu et al., 2018) to capture longer context (Han et al., 2020). Inspired by the success of Transformer (Vaswani et al., 2017) in natural language processing (NLP) field, some works adapt Transformer to speech applications (Dong et al., 2018; Karita et al., 2019; Zhang et al., 2020b; Wang et al., 2020; Zhang et al., 2020a). Compared to CNN, the remarkable benefit of Transformer is that it can learn global dependencies based on self-attention, which is essential for speech processing task. By integrating convolution into Transformer, Conformer (Gulati et al., 2020) gains powerful capability of modeling both local and global contextual information, and outperforms all previous ASR models.

Recent works explore architecture changes on Conformer to further reduce the computational cost and improve the recognition performance. Squeezeformer (Kim et al., 2022) adopts a temporal U-Net structure in which the middle modules operate at half frame rates, and also redesigns the block structure to make it similar to the standard Transformer block (Vaswani et al., 2017). Branchformer (Peng et al., 2022) incorporates parallel branches to model various ranged context, in which one branch captures the local context with convolutional gating multi-layer perceptron (MLP), while the other branch learns long-range dependencies with self-attention. E-Branchformer (Kim et al., 2023) further improves Branchformer by enhancing the branch merging mechanism by convolutionbased module.

Zipformer shares similar ideas about temporal downsampling as the previous work Squeezeformer. However, compared to the fixed downsampling ratio in Squeezeformer, Zipformer operates at different downsampling ratios at different encoder stacks and uses much more aggressive downsampling ratios in the middle encoder stacks. In addition to the modeling differences, our work also focuses on optimization-related changes including a new optimizer ScaledAdam, which are shown to improve convergence in the experiments.

End-to-end framework. Connectionist temporal classification (CTC) (Graves et al., 2006) is one of the earliest frameworks for end-to-end ASR, but its performance is limited by the frame independent assumption. To this end, a hybrid architecture that integrates attention-based encoder-deocder (AED) (Chan et al., 2015) in CTC (Watanabe et al., 2017) (CTC/AED) is proposed to improve the performance. Neural transducer (Graves, 2012), commonly known as RNN-T, addresses the frame independence assumption using a label decoder and a joint network and becomes a popular framework due to its superior performance. Recently, various approaches such as pruning (Kuang et al., 2022; Wang et al., 2023; Mahadeokar et al., 2021) or batch-splitting (Kuchaiev et al., 2019) are proposed to accelerate the training speed and reduce memory usage of neural transducers.

3 Method

3.1Downsampled Encoder Structure

Figure 1 presents the overall architecture of the proposed Zipformer model. Different from Conformer (Gulati et al., 2020) that processes the sequence at a fixed frame rate of 25Hz, Zipformer uses a U-Net-like structure learning temporal representation at different resolutions in a more efficient way. Specifically, given the acoustic features with frame rate of 100Hz, the convolution-based module called Conv-Embed first reduces the length by a factor of 2, resulting in a 50Hz embedding sequence. The obtained sequence is then fed into 6 cascaded stacks to learn temporal representation at frame rates of 50Hz, 25Hz, 12.5Hz, 6.25Hz, 12.5Hz, and 25Hz, respectively. Except for the first stack, the other stacks all adopt the downsampled structures, processing the sequence at lower frame rates. The frame rate between stacks is consistently 50Hz. Different stacks have different embedding dimensions, and the middle stacks have larger dimensions. The output of each stack is truncated or padded with zeros to match the dimension of the next stack. The final encoder output dimension is set to the maximum of all stacks’ dimensions. Specifically, if the last stack output has the largest dimension, it is taken as the encoder output; otherwise, it is concatenated from different pieces of stack outputs, taking each dimension from the most recent output that has it present. Finally, a Downsample module converts the sequence to 25Hz, resulting in the encoder output.

Conv-Embed. In Conv-Embed we use three 2-D convolutional layers with time × frequency strides of 1 × 2, 2 × 2, and 1 × 2, and output channels of 8, 32, and 128, respectively. Subsequently, we utilize one ConvNeXt layer (Liu et al., 2022) similar to Nextformer (Jiang et al., 2022), which is composed of a depth-wise convolution with kernel size of 7 × 7, a point-wise convolution with 384 output channels, a SwooshL activation function (described in Section 3.4), and a point-wise convolution with 128 output channels. Residual connection is applied on the ConvNeXt module. Finally, a linear layer followed by a BiasNorm (described in Section 3.3) is used to adjust the feature dimension to match the first stack.

Downsampled stacks. In the downsampled stacks, the pairwise Downsample and Upsample modules perform symmetric scaling down and scaling up in sequence length, respectively, using almost the simplest methods. For example, with a factor of 2, the Downsample module averages every 2 frames with 2 learnable scalar weights (after softmax normalization), and the Upsample module just repeats each frame twice. After downsampling, it employs the stacking Zipformer blocks (described in Section 3.2) for temporal modeling at lower frame rates. Finally, it utilizes the Bypass module (described in Section 3.2) to combine the stack input and stack output in a learnable way.

3.2 Zipformer Block

Conformer block consists of four modules: feed-forward, Multi-Head Self-Attention (MHSA), convolution, and feed-forward. MHSA learns global context by two steps: computing attention weights using the dot-product operation and aggregating different frames with these attention weights. However, MHSA typically accounts for a large computational cost, since above two steps both require quadratic complexity with respect to the sequence length. Hence, we decompose MHSA into two individual modules according to above two steps: Multi-Head Attention Weight (MHAW) and SelfAttention (SA). This change allows to perform the attention computation twice more efficiently in each block by using one MHAW module and two SA modules. In addition, we propose a new module called Non-Linear Attention (NLA) to make full use of the computed attention weights to capture the global information.

As illustrated in Figure 2 (Left), Zipformer block is equipped with about twice the depth of the Conformer block (Gulati et al., 2020). The main motivation is to allow the re-use of the attention weights to save time and memory. Specifically, the block input is first fed into an MHAW module, which calculates the attention weights and shares them with an NLA module and two SA modules. Meanwhile, the block input is also fed into a feed-forward module followed by the NLA module.

Then it applies two module groups, each consisting of SA, convolution, and feed-forward. Finally, a BiasNorm (described in Section 3.3) is used to normalize the block output. In addition to the regular residual connections using adding operation, each block utilizes two Bypass modules to combine the block input and the module outputs, placed in the middle and end of the block. Note that different from regular Transformer models (Vaswani et al., 2017), we don’t use normalization layer such as LayerNorm (Ba et al., 2016) for each module to periodically prevent activations from becoming either too large or too small, since our proposed ScaledAdam optimizer is able to learn the parameter scales (described in Section 3.5).

Non-Linear Attention. Figure 2 (Right) presents the NLA structure. It also leverages the precomputed attention weights from MHAW to aggregate the embedding vectors over the time axis, which is similar to SA. Specifically, it first projects the input with 3 linear layers to A, B, and C, each being of 3/4 input dimension. The module output is linear(A ⊙ attention(tanh(B) ⊙ C)), where ⊙ denotes the element-wise multiplication, attention represents matrix-multiplying on the time axis by a single head of previously computed attention weights, and the linear layer recovers the dimension to the same as the input.

Bypass. The Bypass module learns channel-wise scalar weights c to combine the module input x and module output y: $(1 - c) \odot x + c \odot y$. In training, we initially limit the values of c in range of [0.9, 1.0] and then change the minimum to 0.2 after 20000 steps. We found that making modules "straight-through" at the beginning (i.e. allowing very little bypass) helps model convergence.

3.3 BIASNORM

Conformer (Gulati et al., 2020) utilizes LayerNorm (Ba et al., 2016) to normalize the module activations. Given x with D channels, LayerNorm is formulated as:

$$LayerNorm(x) = x - E[x]\, p\, Var[x] + \epsilon \odot \gamma + \beta$$

Specifically, it first computes the mean E[x] and the standard-deviation p Var[x] for normalizing, scaling the vector length to $\sqrt{D}$. Then it uses the learnable channel-wise scale γ and bias β for transformation, which helps to adjust the size of activations and balance the relative contributions of specific modules. However, we observe that the trained Conformer using LayerNorm suffers from two problems: 1) It sometimes sets one channel to a large constant value, e.g. 50. We argue that it aims to "defeat" the LayerNorm which fully removes the vector length, functioning as a very large value so that length information could be retained after normalization. 2) Some modules (typically feed-forward or convolution) are “dead” as they have extremely small output values, e.g., 10^{-6} . We argue that early in training, the un-trained modules are not useful so they are “turned off” by the LayerNorm scale γ approaching zero. If the scale γ oscillates around zero, the inconsistent sign constantly reverses the gradient directions back-propagating to the modules. Because of the inconsistent gradient sign, the modules never learn anything useful, since this is a bad local optimum which is hard to escape because of the dynamics of stochastic gradient descent-like updates.

To address above problems, we propose the BiasNorm which is intended to be a simpler replacement of LayerNorm. Specifically, BiasNorm is formulated as: $\mathrm{BiasNorm}(x) = \frac{x}{\mathrm{RMS}[x-b]} \cdot \exp(\gamma)$ (2) where b is the learnable channel-wise bias, RMS[x - b] is the root-mean-square value taken over channels, and γ is a scalar. We first remove the operation of mean subtraction since it is a waste of time unless it follows a non-linearity. The bias b serves as the large constant value which allows to retain the vector length information after normalization. Since the scale exp(γ) is always positive, it avoids the gradient oscillation problem.

3.4 SwooshR and SwooshL Activation Functions

Conformer (Gulati et al., 2020) adopts Swish (Ramachandran et al., 2017) activation function with the following formula: $Swish(x) = x \cdot (1 + exp(-x)) - 1$.

In this work, we propose two new activation functions respectively called SwooshR and SwooshL as replacements of Swish: $SwooshR(x) = log(1 + exp(x - 1)) - 0.08x - 0.313261687, SwooshL(x) = log(1 + exp(x - 4)) - 0.08x - 0.035$. (4) In SwooshR, the offset 0.313261687 is to make it pass through the origin; in SwooshL, the offset 0.035 was tuned, which slightly outperformed the value exactly making the curve pass through the origin. We present the curves of Swish, SwooshR, and SwooshL in Appendix Section A.2. SwooshL is roughly a right shifted version of SwooshR. Note that the suffix “L” or “R” represents whether the left or right zero-crossing is at or around x = 0. Similar to Swish, SwooshR and SwooshL have lower bounds and are non-monotonic. Compared to Swish, the most striking difference is that SwooshR and SwooshL have non-vanishing slopes for negative inputs, which helps to escape from situations where the input is always negative and prevents the denominator term in Adam-type updates from getting dangerously small. When replacing Swish with SwooshR, we observe that the modules with bypass connections, such as feed-forward and ConvNeXt, tend to learn a large negative bias in the preceding linear layer to learn “normally-off” behavior. Therefore, we use SwooshL for these “normally-off” modules and use SwooshR for convolution modules and the rest of Conv-Embed.

3.5 ScaledAdam Optimizer

We propose a parameter-scale-invariant version of Adam (Kingma & Ba, 2014) called ScaledAdam, which enables faster convergence and better performance. ScaledAdam scales each parameter’s update proportional to the scale of that parameter, and also explicitly learns the parameter scale. Algorithm 1 in Appendix Section A.1.1 presents the pseudo-code of the ScaledAdam. Let f(θ) be the loss function that we aim to minimize, which is differentiable w.r.t. the learnable parameter θ. At each step t, Adam computes the parameter gradient $g_t = \nabla_\theta f(\theta_{t-1})$, and updates the first moment $m_t = \beta_1 \cdot m_t - 1 + (1-\beta_1) \cdot g_t$ and the second moment $v_t = \beta_2 \cdot v_t - 1 + (1-\beta_2) \cdot g_t^2$ of gradients, where β1, β2 ∈ [0, 1) are coefficients used to compute the moving averages. The parameter update Δt at step t is formulated as: $\Delta t = \alpha_t \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \frac{m_t}{\sqrt{v_t}+\epsilon}$, (5) where αt is the learning rate typically specified by an external schedule, $\frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t}$ is the bias-correction term, and $\epsilon = 10^{-8}$. Whilst Adam is invariant to gradient scale of each parameter, we argue that it still suffers from two limitations: 1) The update Δt in Equation 5 does not take into account the parameter scale (denoted as rt−1). Considering the relative parameter change Δt/rt−1, Adam might cause learning in relative terms too slowly for parameters with large scales, or too fast for parameters with small scales. 2) It is difficult to learn the parameter scale directly, as the direction of growing or shrinking the parameter tensor is a very specific direction in a large-dimensional space. It’s particularly difficult to shrink a parameter, since each gradient step gt adds noise which tends to grow the parameter norm.

Scaling update. To keep the relative change Δt/rt−1 over parameters of varying scales about the same, we scale the update Δt in Equation 5 by the parameter scale rt−1: $\Delta'_t = \alpha_t \cdot r_{t-1} \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \frac{m_t}{\sqrt{v_t}+\epsilon}$. (6) We compute the parameter scale rt−1 as the root-mean-square value RMS[θt−1]. Because the ScaledAdam update is less prone to divergence than Adam, we use a learning rate schedule called Eden that does not have a long warm-up period; we also use absolutely larger learning rate values because the parameter RMS value is normally much less than one.

Learning parameter scale. To explicitly learn the parameter scale, we treat it as a regular parameter to be learned, as if we have factored each parameter as θ = r • θ′ , and we are doing gradient descent on the parameter scale r and the underlying parameter θ′ . Let h be the gradient of the parameter scale r, at step t we get $h_t = \nabla_r f(\theta_{t-1}) = g_t \cdot \theta'_{t-1}$. Since Adam is nearly invariant to changes in the gradient scale, for simplicity we replace this with $h_t = g_t \cdot (r_{t-1} \odot \theta'_{t-1}) = g_t \cdot \theta_{t-1}$. Following the Adam algorithm, we maintain the first moment $n_t = \beta_1 \cdot n_{t-1} + (1-\beta_1) \cdot h_t$ and the second moment $w_t = \beta_2 \cdot w_{t-1} + (1-\beta_2) \cdot h_t^2$ of the scale gradients ht. The parameter change on θ caused by updating parameter scale from rt−1 to rt is $\Delta'_t, r = (r_t - r_{t-1}) \odot \theta'_{t-1}$. . Similar to Equation 6, we also integrate the parameter scale rt−1 into the update Δ′ t,r:

$\Delta'_{t}, r = \eta \cdot \alpha_t \cdot r_{t-1} \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \sqrt{w_t} + \epsilon \odot \theta'_{t-1} = \eta \cdot \alpha_t \cdot \frac{\sqrt{1-\beta_t^2}}{1-\beta_1^t} \cdot \sqrt{w_t} + \epsilon \odot \theta_{t-1}$. (7) where η is a scaling factor on learning rate αt, and we found that setting η = 0.1 helps to stabilize the training. Now the update **Δ′** t is replaced with **Δ′** t,r + **Δ′** t , which amounts to adding an extra gradient term in the direction of growing or shrinking each parameter. This also allows to simplify the network structure by removing most of normalization layers in our Zipformer Block (described in Section 3.2), since the modules now can easily learn to scale the activations in a suitable range. One similar method called weight normalization (Salimans & Kingma, 2016) decouples the parameter norm from its direction to speed up the convergence. It replaces each parameter with two parameters, respectively specifying the direction and the magnitude. However, ScaledAdam learns the parameter scales by adding an extra update term **Δ′** t,r, which makes writing the modeling code simpler.

Eden schedule. The proposed Eden learning rate schedule is formulated as:

$$\alpha_t = \alpha_{\text{base}} \cdot \frac{t^{\frac{2}{\alpha_{\text{step}}}}}{\alpha_{\text{step}}^{\frac{2}{\alpha_{\text{step}}}}} \cdot \frac{e^{\frac{2}{\alpha_{\text{epoch}}}}}{\alpha_{\text{epoch}}^{\frac{2}{\alpha_{\text{epoch}}}}} \cdot \text{linear}(\alpha_{\text{start}}, t_{\text{warmup}}, t)$$

(8) Herein, t is the step index, e is the epoch index, αstep and αepoch respectively control the number of steps and number of epochs after which we start significantly decreasing the learning rate, linear(αstart, twarmup, t) is a warmup scale increasing linearly from αstart to 1 over twarmup steps and then staying constant at 1, αbase is the maximum value when setting αstart = 1, αwarmup = 0. The reason for making Eden dependent on both the step index t and the epoch index e is to keep the amount of parameter change after certain amount of training data (e.g., one hour) approximately constant when we change the batch size, so the schedule parameters should not have to be re-tuned if we change the batch size. Other versions of Eden replace the “epoch” parts of the formula with some suitable measure of the amount of data seen. In this work,

we use αbase = 0.045, αstart = 0.5, and twarmup = 500.

Efficient implementation. To speedup the optimization in ScaledAdam, we group the parameters into batches according to their shape and perform the computation batch by batch. Note that this doesn’t affect the outcome. ScaledAdam just requires a little more memory than Adam to cache the gradient moments nt and wt (in Equation 7) for the parameter scales.

4 Experiments

4.1 Experimental Setup

Architecture variants. We build our Zipformer variants with three model scales: small (ZipformerS), medium (Zipformer-M), and large (Zipformer-L). For the 6 encoder stacks, the numbers of attention heads are set to {4,4,4,8,4,4}, the convolution kernel sizes are set to {31,31,15,15,15,31}. In each attention head, the query dimension and value dimension are set to 32 and 12, respectively. For the three feed-forward modules in each Zipformer block, the hidden dimensions in the first one and the last one are 3/4 and 5/4 of that in the middle one. We adjust the layers numbers, the embedding dimensions, and the hidden dimensions of the middle feed-forward in each stack to obtain different model scales.

Datasets. We perform experiments to compare our Zipformer with state-of-the-other models on three open-source datasets: 1) LibriSpeech (Panayotov et al., 2015) which consists of about 1000 hours of English audiobook reading; 2) Aishell-1 (Bu et al., 2017) which contains 170 hours of Mandarin speech; 3) WenetSpeech (Zhang et al., 2022a) which consists of 10000+ hours of multidomain Mandarin speech.

Implementation details. We use Lhotse (Zelasko et al., 2021) toolkit for speech data preparation. · The model inputs are 80-dimension Mel filter-bank features extracted on 25ms frames with frame shift of 10ms. Speed perturbation (Ko et al., 2015) with factors of 0.9, 1.0, and 1.1 is used to augment the training data. SpecAugment (Park et al., 2019) is also applied during training.

We use mixed precision training for our Zipformer models. We also employ the activation constraints including Balancer and Whitener to ensure training consistency and stability. The details of Balancer and Whitener are presented in Appendix Section A.3. Pruned transducer (Kuang et al., 2022), a memoryefficient version of transducer loss that prunes path with minor posterior is used as the training objective. During decoding, beam search of size 4 with the constraint of emitting at most one symbol per frame is employed (Kang et al., 2023). We don’t use external language models for rescoring, since in this work we focus on improving the encoder model. We employ word-error-rate (WER) and character error rate (CER) as evaluation metric for English and Mandarin datasets, respectively. By default, all of our models are trained on 32GB NVIDIA Tesla V100 GPUs. For Librispeech dataset, Zipformer-M and Zipformer-L are trained for 50 epochs on 4 GPUs, and Zipformer-S is trained for 50 epochs on 2 GPUs. For Aishell-1 dataset, our models are trained for 56 epochs on 2 GPUs. For WenetSpeech dataset, our models are trained for 14 epochs on 4 GPUs.

4.2 Comparsion With State-of-the-Art Models

In this section, we compare the proposed Zipformer with other state-of-the-art models.

LibriSpeech dataset. Table 2 shows the results on LibriSpeech test datasets for Zipformer and other state-of-the-art models. For Conformer, we also list the WERs reproduced by us and other open-source frameworks. Note that there is a performance gap between the open-source reproduced Conformer and the original Conformer. Our Zipformer-S model achieves lower WERs than all variants of Squeezeformer while having much fewer parameters and floating point operations (FLOPs). Our Zipformer-L outperforms Squeezeformer-L, Branchformer and our reproduced Conformer-L by a large margin while saving over 50% FLOPs. Noticeably, when trained on 8 80G NVIDIA Tesla A100 GPUs for 170 epochs, Zipformer-L achieves WERs of 2.00%/4.38% with sufficient computing resources (last row), which is the first model to approach Conformer-L to the best of our knowledge.

We also compare the speed and memory usage between the proposed Zipfomer and other state-ofthe-art models. Figure 3 presents the comparison results in terms of averaged inference time and peak memory usage in inference mode for batches of 30-second audios on an NVIDIA Tesla V100 GPU. The batch size is set to 30 to ensure all models do not have out of memory problems during inference. In overall, Zipformer models achieves better trade-off between performance and efficiency than other models. Especially for the large scale, Zipformer-L requires much less computation time and memory than other counterparts.

Aishell-1 dataset. Table 3 shows the CERs on Aishell-1 dataset. Compared to the Conformer model implemented in ESPnet toolkit, our Zipformer-S achieves better performance with fewer parameters. Scaling up the model leads to lower WERs, and Zipformer-M/L outperform all other models.

WenetSpeech. Table 4 presents the experimental results on WenetSpeech dataset. Again, our Zipformer-M and Zipformer-L outperform all other models on Test Net and Test Meeting test sets. With only one third of the parameters, our Zipformer-S yields lower WERs than Conformer models.

4.3 Ablation Studies

We perform ablation experiments on LibriSpeech dataset to investigate the effect of each proposed functional technique. With Zipformer-M as the base model, we make one change each time while keeping the others untouched. Table 5 presents the experimental results.

Encoder structure. We remove the temporal downsampling structure from Zipformer and use Conv-Embed with downsampling rate of 4 like Conformer. The resulting model has 12 Zipformer blocks with a constant embedding dimension of 512 and has more parameters than the base model.

Block structure. As each Zipformer block has roughly twice modules as a Conformer block, we replace each Zipformer block in the base model with two Conformer blocks stacked together. This leads to 0.16% absolute WER reduction on test-other even with a larger model size, suggesting the benefits of Zipformer block structure. Removing either NLA or Bypass leads to performance degradation. If we further remove the attention weights sharing mechanism after removing NLA, the model has slightly more parameters and slower inference speed, but the WERs are not improved. We hypothesize that the two attention weights inside one Zipformer block are quite consistent and sharing them does not harm the model.

Normalization layer. Replacing BiasNorm with LayerNorm in Zipformer leads to WER drops of 0.08% and 0.18% on test-clean and test-other, respectively. It indicates the advantage of the proposed BiasNorm which allows to retain some length information in normalization.

Activation function. When using only SwooshR for all modules in Zipformer, the WER drops by 0.11% and 0.42% on test-clean and test-other, respectively, which validates the effectiveness of particularly using SwooshL for the "normally-off" modules. Employing Swish leads to more performance degradation, which indicates the advantage of SwooshR over Swish.

Optimizer. When using Adam to train Zipformer, we have to apply BiasNorm for each module in Zipformer block to avoid model divergence, since Adam cannot learn the scale of each parameter to adjust the module activations like ScaledAdam. We try different learning rate factors (denoted as αbase) for ScaledAdam (0.025, 0.035, 0.045, 0.055) and Adam (2.5, 5.0, 7.5, 10.0) separately. Following (Gulati et al., 2020), the learning rate schedule for Adam is $\alpha_t = \alpha_{base} \cdot 512 - 0.5 \cdot min(t_{-0.5}, t \cdot 10000_{-1.5})$. Figure A.2 in Appendix Section A.1.2 presents the averaged WERs on test-clean and test-other at different epochs as well as the learning rates at different steps. We show the best results of ScaledAdam with αbase = 0.045 and Adam with $\alpha\_\{base\} = 7.5$ in Table 5. ScaledAdam outperforms Adam by 0.17% and 0.72% on test-clean and test-other, respectively. The results indicate that ScaledAdam enables faster convergence and better performance than Adam.

5 CONCLUSION

In this work, we present the Zipformer, which serves as an efficient ASR encoder. It has an UNet-like encoder structure, which downsamples the sequence to various lower frame rates. The re-designed block structure equipped with more modules reuses the computed attention weights for efficiency. It also employs the new normalization method BiasNorm, as well as the new activation functions SwooshR and SwooshL. Meanwhile, the proposed optimizer ScaledAdam enables faster convergence and better performance. Extensive experiments on LibriSpeech, Aishell-1 and WenetSpeech datasets have demonstrated the effectiveness of the proposed Zipformer.

毕业论文（设计）文献综述和开题报告考核

导师对开题报告、外文翻译和文献综述的评语及成绩评定：

成绩比例	文献综述（10%）	开题报告（15%）	外文翻译（5%）
分 值			

导师签名___________

年　　月　　日

学院盲审专家对开题报告、外文翻译和文献综述的评语及成绩评定：

成绩比例	文献综述（10%）	开题报告（15%）	外文翻译（5%）
分 值			

开题报告审核负责人（签名/签章）___________

年　　月　　日