南京大学

研 究 生 毕 业 论 文

（申请硕士学位）

论 文 题 目　基于神经网络结构的压缩加速研究

作 者 姓 名　赖碧兰

专 业 名 称　计算机科学与技术

研 究 方 向　人工智能与模式识别

指 导 教 师　申富饶

2022 年 5 月 17 日

学　　号：　MG1933032

论文答辩日期：　2022 年 5 月 17 日

指 导 教 师：　  　（签字）

Research on Compression Acceleration Based on Neural Network Structure

by

Lai Bi-Lan

Supervised by

Professor Shen Fu-Rao

A dissertation submitted to

the graduate school of Nanjing University

in partial fulfilment of the requirements for the degree of

MASTER

in

Computer Science and Technology

Department of Computer Science and Technology

Nanjing University

May 17, 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目： 基于神经网络结构的压缩加速研究

计算机科学与技术 专业 2019 级硕士生姓名： 赖碧兰

指导教师（姓名、职称）： 申富饶 教授

随着深度网络的发展，参数量大量增加，模型压缩成为当前深度网络研究的重要领域，而模型剪枝更是模型压缩中的重要部分。本文主要对模型网络的剪枝压缩进行了深入研究，模型剪枝的核心问题是找到一个可靠的重要性评价指标，使得模型能够精准地剪掉不重要的参数，保留重要参数。在之前的研究工作中，有许多工作是通过权重大小、输入数据和重构损失误差等方式来判断权重重要性的，这些指标无法与权重重要性直接关联。本文基于一种稳健统计方法——影响函数进行权重的影响力测量，通过影响力指标进行权重的重要性判断，从而实现剪枝工作。影响函数测量方法在之前的研究中证明其能够对神经网络中的重要性进行计算测量，因此影响力计算权重重要性是具备可解释性的。基于此，本文针对不同的网络结构设计了基于影响函数的压缩算法，并进行了理论证明和实验验证。本文主要内容如下：

1．本文提出了一种针对卷积神经网络的自动剪枝方法。这种方法通过影响函数更加准确地测出权重的影响力，根据影响力进行通道剪枝。深度网络的反向传播是求导过程，与经验影响函数形式类似，因此我们借助反向传播过程进行影响力测算。我们在模型损失函数增加了正则项，用于监督模型的剪枝策略。在实验中，本文证明了依据影响函数进行修剪，剪枝效果将会比现有的最新模型要有效得多。

2．本文利用影响函数测量理论针对 LSTM 记忆单元结构设计了一种权重压缩方法。该方法是一种不依靠硬件支撑的非结构化剪枝算法，其主要依靠经验影响函数测算权重的影响力，通过权重影响力进行非结构化剪枝，从而得到稀疏矩阵，然后将矩阵通过 SVD 分解来实现低秩分解压缩。

3．本文将卷积网络和长短期记忆网络两种结构的压缩算法应用于语音增强系统中，系统中的语音增强模型均涉及了卷积网络结构和长短期记忆网络结构，将本文提出的算法应用于该系统中进行测试，验证了基于影响函数的压缩算法在实际应用中的有效性。

关键词：神经网络压缩；影响函数；模型剪枝；低秩分解

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Research on Compression Acceleration Based on Neural Network Structure

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: Lai Bi-Lan

MENTOR: Professor Shen Fu-Rao

With the development of deep networks, the amount of parameters has increased significantly, and model compression has become an important area of current deep network research, and model pruning is an important part of model compression. In this paper, the core problem of model pruning is to find a reliable importance evaluation index, so that the model can accurately cut off unimportant parameters and retain important parameters. In previous research work, there was a lot of work to determine the importance of weights by means of weight size, input data, and reconstruction loss errors, etc., and these indicators could not be directly related to weight importance. This paper is based on a robust statistical method, the influence function to measure the influence of the weights, and the weights are judged by the influence indicators, so as to achieve pruning work. The influence function measurement method has proved in previous studies that it can make computational measurements of the importance in neural networks, so the weight importance of influence calculation is interpretable. Based on this, this paper designs a compression algorithm based on the influence function for different network structures, and conducts theoretical proof and experimental verification. The main contents of this article are as follows:

1. This paper proposes an automatic pruning method for convolutional neural networks. This method measures the influence of the weights more accurately through the influence function, and performs channel pruning according to the influence. Backpropagation of deep network is a derivation process, which is similar to the form of empirical influence function, so we use the backpropagation process to measure influence. We add a regular term to the model loss function to supervise the model's pruning strategy. In experiments,

this paper proves that pruning according to the influence function, the pruning effect will be much more effective than the existing state-of-the-art models.

2. In this paper, a weight compression method is designed for the LSTM memory cell structure using the influence function measurement theory. This method is an unstructured pruning algorithm that does not rely on hardware support. It mainly relies on the empirical influence function to measure the influence of the weight, and performs unstructured pruning through the weight influence to obtain a sparse matrix, and then the matrix is passed through SVD. decomposition to achieve low-rank decomposition compression.

3. In this paper, the compression algorithms of the convolutional network and the long short-term memory network are applied to the speech enhancement system. The speech enhancement models in the system all involve the convolutional network structure and the long and short-term memory network structure. The algorithm proposed in this paper is applied to the speech enhancement system. The system is tested to verify the effectiveness of the compression algorithm based on the influence function in practical applications.

Keywords: Neural Network Compression; Influence Function; Model Pruning; Low-rank Decomposition

目　　录

插图目录

表格目录

第一章　绪论

1.1　研究背景与意义

人工神经网络是现如今最热门的人工智能研究方向，人工神经网络具有独特的网络结构，且信息处理方法特殊，使得它被广泛应用于自动控制领域、模式识别、组合优化、自然语言处理等多个领域。人工神经网络的兴起是在 2006 年，Hinton 等人[1]利用限制玻尔兹曼机对神经网络的连接层进行建模，采用逐层训练的方法抽取模型数据中的高维特征，后来又提出了深度信念网络。随着硬件计算能力和算法的提升，网络隐层不断增加，最终 Hinton[2-4]为代表的研究人员重新将人工神经网络定义为深度学习，并进行推广。深度学习是基于神经网络发展起来的技术，已成为机器学习中最主流的分支之一。它的广泛应用不计其数。自 2011 年起，神经网络在语音识别和图像处理基准测试中获得了压倒性的优势，这迎来了神经网络发展的第三次崛起。神经网络由此其衍生了一系列深度学习开源框架，为如今深度学习在无人驾驶、机器翻译、金融风控等诸多领域的广泛应用打下坚实的基础。同时，硬件随着人工神经网络的发展，其承载的算力和计算速度大幅度提升，大量训练数据的收集也更加容易。研究人员开始通过 GPU 并行运算，只需要数天时间即可完成深层网络的训练。此外，随着互联网的普及，当前技术已经能够获得大量的训练数据，从而抑制过拟合。这些硬件上的辅助和可观的数据量，为当前深度网络的技术进步提供了有力支撑。

近几年，深度神经网络受到了很多关注。它被应用于不同的领域，而且在许多任务中实现了显着的精度提升。这些工作依赖于具有数百万甚至数十亿个参数的深度网络，而具有非常高计算能力的 GPU 的可用性在其成功中起着关键作用。例如，Krizhevsky 等人[3]的工作在 2012 年 ImageNet[5]挑战赛中取得了突破性成果。该项工作使用了包含 6000 万个参数的网络，其中包含五个卷积层和三个全连接层。通常，使用 NVIDIA K40 机器在 ImagetNet 数据集上训练整个模型

需要两到三天。另一个例子，Wild[6]数据集中的人脸通过包含数亿个参数的网络获得了标签上的最佳人脸验证效果。该模型使用卷积、局部连接和全连接层的混合体进行训练。这样的模型获得最优性能，但也非常耗时。在仅依赖全连接层的架构中，参数数量可能会增长到数十亿。

随着具有更多层和节点的更大神经网络的出现，降低其存储和计算成本变得至关重要，特别是对于一些实时应用，如在线学习和增量学习。近年来，虚拟现实技术、增强现实技术和智能可穿戴设备取得了重大进展，为研究人员创造了前所未有的机会，以应对将深度学习系统部署到资源有限（如内存、CPU、能源、带宽）的便携式设备中的巨大挑战。对于只有几兆字节资源的手机和 FPGA 等设备，很大程度阻碍了基于深度学习方法的产品化，尤其是在一些边缘设备上。因为边缘设备大多不是为计算密集任务设计的，如果简单部署上去则功耗、时延等都会成为问题。即使是在服务端，更多的计算也会直接导致成本的增加。因此需要考虑模型的轻量化。而神经网络压缩是其中至关重要的一种方式。当前的高效深度学习方法对分布式系统、嵌入式设备和面向人工智能的 FPGA 产生了重大影响。例如，具有 50 个卷积层的 ResNet-50[7]在处理图像时需要超过 95MB 的内存进行存储，并需要超过 38 亿次浮点数乘法。但是，在丢弃一些冗余权重后，网络仍可照常工作，可节省 75% 以上的参数和 50% 的计算时间。这证明了，网络确实存在参数压缩的空间，具备压缩的可行性。

在以往的机器学习中，通常需要大量的人工干预。这些人工干预表现在各个方面，从特征提取、模型选择，到参数调节等各个步骤都需要运用人工经验。自动化机器学习（AutoML）不仅像 Michell[8]的书中描述的那样仍然是一个学术梦想，而且还吸引了从业者的更多关注。如果我们能够将人类从这些机器学习应用程序中解放出来，我们就可以跨组织更快地部署机器学习解决方案，有效地验证和基准测试已部署解决方案的性能，并使专家更加关注具有更多应用程序和业务价值的问题。这将使机器学习更容易被现实场景使用，从而达到新的能力和制造水平，是拥有巨大影响力的工作。在上述学术梦想和实践需求的推动下，近年来，AutoML 已成为机器学习的一个新子领域。研究人员开始关注自动化的机器学习，以此来降低人工成本和人工经验误差带来的实验干扰。不仅如此，自动化学习在多个方面受到广泛关注，除了机器学习之外，在计算机视觉、数据挖掘和自然语言处理等方面也获得了更多的关注。到目前为止，AutoML 已

经成功应用在许多重要任务上。而自动化机器学习在网络压缩中的应用也十分广泛，由于人工压缩[9-14]需要调参，而调参工作需要经验积累，这对调参人员的技术要求更高，且调参过程消耗的资源比自动压缩所消耗的资源更多的多。随着近年来深度网络的发展，人们对网络运算速度要求变高，为了更便捷高效的获取一个好的压缩网络，使得自动化网络压缩成为了一种势在必行的趋势。

上述分析表明，对深度神经网络的自动化加速具有较大的必要性，同时也具有理论可行性。实现设计研究一种能够实现端到端的自适应剪枝方法能够有效降低网络参数量及运算数量，从而降低存储空间的占用率及运算过程中的资源消耗，并且能够有效应用在现实场景下。这对人工智能的长足发展具有深刻意义。

1.2　研究现状与难点

随着深度网络的运用越来越广泛，模型的深度也越来越深，模型参数量指数级增长，因此深度模型的压缩需求愈加迫切。而什么是好的模型压缩方法？一个好的模型压缩方法是既能保持模型性能最优，又能保持模型参数最为精简。对于一个好的深度模型，其性能必定是尽可能的达到最优水平，而轻量模型则要求模型参数最大程度的精简。但是，通常性能最优的模型泛化性强，这也往往导致了模型参数较多，对于极度轻量化的模型，由于过度精简也会导致模型性能损失。在目前的终端设备中，不同种类的硬件对神经网络压缩加速工作的要求也各不相同，要想实现更精细化、更具有针对性的压缩工作，则具备更大的实现难度。目前模型的压缩加速方法分为四类[15]：参数修剪和共享、低秩分解、紧凑卷积滤波器和知识蒸馏。基于参数修剪和共享的方法主要是探索模型参数中的冗余，并尝试删除冗余和非关键参数。基于低秩分解的技术使用矩阵分解来估计深度网络的信息参数，这种方法使用简单，易于与其他方法结合，但是由于矩阵分解的计算量较大，对矩阵大小的要求增加。基于转移/紧凑卷积滤波器的方法设计了特殊的结构卷积滤波器，以减少参数空间并节省存储和计算。但这种方法的一个问题是，粗暴直接的结构紧凑会损害性能，因为结构紧凑可能会给模型带来偏差；另一方面，如何找到合适的结构矩阵是困难的，没有理论上的方法可以推导出来。知识蒸馏[16]方法是将待压缩模型作为教师模型（Teacher

Model），选择一个较小的模型（Student Model）学习教师模型的预测结果，最终获得更紧凑的神经网络来重现更大网络的输出。但这种方法对小模型的结构敏感，而模型结构的选择需要凭借人工经验。

参数剪枝方法起源于 Oracle pruning[17]的方法，主要工作是选择对网络性能影响力较小的参数部分进行剪枝，使得最终网络的性能尽可能的保持原来的水平。模型剪枝工作能够降低深度模型的计算量，使得计算过程加速。但是剪枝方法使用不当或者剪枝过度，则可能导致模型性能大幅度退化，甚至模型无法实际应用。在模型剪枝的工作中，最核心的问题的就是如何进行冗余参数的搜索。正确的搜索指标能够获得准确的模型参数，从而最大程度保留模型的性能。目前主流的方法层出不穷，花样百出，最终都能够获得相对较好的压缩加速效果。在参数剪枝中，又具体细分为结构化剪枝和非结构化剪枝，划分依据的是剪枝的细粒度不同。结构化剪枝主要是实现网络层级间、神经元间的剪枝，由于细粒大较大，因此剪枝压缩效果较为明显，但是，大细粒度剪枝是较为直接粗暴的删减网络层、神经元，所以可能会对网络性能造成一定程度的损失，而性能损失程度也许剪枝细粒度大小呈正相关，细粒度越大，损失的性能也越多。非结构化剪枝则主要实现卷积核级别的小细粒度精细化剪枝，这种工作只会删减卷积核内部的一些不重要参数，因此这种方法通常能够最大程度保证网络精度的水平不下降，但是这需要特定的硬件支撑，才能够完成模型的压缩加速工作，这使得非结构化剪枝对部署的平台具有更高的要求。综上所述，最终不同场景下的剪枝方法选择，要依据实际应用的场景进行选择，以求得模型在保证最优精度的同时，实现模型最大化的压缩加速。

紧凑卷积核的设计对矩阵设计有更高的要求，这需要更多的专家经验，即要求对神经网络具备深刻的理解能力，同时又需要对神经网络现有的问题具备敏锐的观察力。研究者需要具备以上能力才能实现对模型的针对性设计，最大程度简化神经网络的结构，达到降低网络计算量的目的。近几年的相关工作[7,18-20]主要集中于利用分组卷积的方法，实现降低模型参数量与计算复杂度的目的。因此要求能够保障网络性能、设计具有实用意义的分组策略，是紧凑卷积核设计的一个核心问题。

低秩分解主要是利用数学方法将卷积参数矩阵进行分解简化，神经网络中典型的卷积核就是一个四维矩阵，通常矩阵会存在较大的冗余，可以使用 SVD

分解[21]、CP 分解[22]等方法将矩阵分解，得到维数更低的卷积核，实现降低网络参数计算量的目的。这种方法使得模型性能下降程度小，模型压缩效果好。但由于分解操作的计算复杂度高，导致其计算成本高，且训练时间长，计算收敛慢，因此在实际应用中较为小众。

神经网络知识蒸馏使用的是“教师-学生”的学习范式来对网络进行优化学习，这种方法需要一个预训练的大网络模型和一个模型结构较为相似的小模型，通常大网络模型更加复杂，模型深度比小模型更深。通过大网络的 softmax 输出来获得精准的类别分布输出，提供给小网络训练学习。通常这种方法不会对小型网络的结构进行改变，且小网络结构在训练前后其计算复杂度也并未变化。由于小型网络学习了含有更多信息的类分布输出数据，使得小网络具备更强的学习目标任务特征信息的能力，达到有效提升网络性能的目的。当前网络有着比其结构更复杂的网络的学习泛化能力，也是另一种维度的压缩。基于知识蒸馏的方法，对降低网络参数和加快计算速度卓有成效，但这种学习范式假设相对严格，压缩效果有时与其他方法相比还会略逊一筹。

当前，深度网络的自动剪枝压缩大多受自动机器学习算法的启发。使压缩算法具备自动搜索相应的压缩方法的能力，从而达到算法精度与计算效率的最佳平衡点。AutoML 的最终目的是实现模型自动地学习恰当的模型参数和配置，使其不需要人工干预，实现端到端的自动学习，使其降低设计和部署成本。AutoML 在多个方面亟待探索[23]，例如特征工程、超参优化、模型架构搜索等。目前研究者针对这些问题提出的解决办法多种多样：进化算法[24]、强化学习[25]、深度强化学习[26]、遗传算法[27]等等。深度网络自动压缩基于 AutoML 算法，能够实现模型精度和计算速度的平衡，为目标深度网络模型自动搜索合适的压缩方法。例如，AMC[28]实用深度确定性策略梯度，经每层的参数作为状态参考值，逐层自动计算出合适的剪枝比，在同等压缩率情况下，AMC 算法比人工调参具备更好的压缩效果。HAQ[29]通过硬件的计算资源作为状态参考值，利用强化学习机制自动推理出每层的量化位宽度，寻找到最匹配的量化模型。

在上述网络压缩方法中，本文主要聚焦于神经网络自动剪枝部分，针对卷积神经网络、全连接层和长短期记忆网络等不同结构进行端到端的神经网络剪枝压缩，通过根据模型自身情况自动确定剪枝策略，最终达到较好的压缩效果。

1.3 研究内容

本文在研究网络剪枝压缩的基础上，结合了神经网络的可解释性工作[30]，通过一种数学上的稳健统计[31]（Robust Statistics）方法计算模型权重的重要性，即影响函数计算法。通过影响函数计算权重的重要程度，依据重要程度进行排序剪枝，同时结合了 AutoML 的方法。这种剪枝算法具备了自动调节的能力，利用损失函数中附加了一个正则鼓励项，鼓励模型剪枝策略向其目标策略靠拢。这促使模型同时考虑准确性和模型剪枝策略，实现剪枝策略的自动调节。基于这种自动剪枝算法分别设计了针对卷积神经网络（Convolutional Neural Networks, CNN）和长短期记忆网络（Long Short-Term Memory, LSTM）两种模型结构的不同剪枝方法，并将算法成功应用在了实际系统中，证明了两种算法具备有效性。其中，本文的研究内容如下：

- 本文提出了一种针对卷积神经网络的自动剪枝方法。这种方法通过影响函数（一种来自稳健统计的经典技术）更加准确的测出权重的影响力，根据影响力进行通道剪枝。深度网络的反向传播是一阶导过程，与经验影响函数形式类似，借助反向传播进行影响力测算是可行的。在实验中，本文证明了依据影响函数进行修剪，剪枝效果将会比现有的最新模型要有效的多。我们在模型损失函数增加了正则项，用于监督模型的剪枝策略。本文提出的方法应用在 ResNet-56 和 VGG-16 上，在 CIFAR-10 和 CIFAR-100 的数据集中进行的实验验证，其中 VGG-16 在 CIFAR-16 上可以提升 0.28% 的精度，但实现了对模型 87.69% 的参数压缩效果，且降低了 57% 的 FLOPs 计算量。
- 本文利用影响函数测量理论针对 LSTM 记忆单元结构设计了一种权重压缩方法。主要依靠经验影响函数测算权重的影响力，通过权重影响力进行非结构化剪枝，从而得到稀疏矩阵，然后将矩阵通过 SVD 分解来实现低秩分解压缩。实验主要在语言模型中进行测试，主要应用了 Penn Tree-Bank[32]和 Enwik8 两个语料库，同时我们还进行了消融实验。根据实验结果可知，LSTM 模型在 Penn TreeBank 语料库的测试中，当影响力剪枝率为 80% 时，其 SVD 分解之后的参数量能达到 65.20% 的压缩率，且测试损失与原模型相比不变，测试的 PPL（perplexity）值降低了 0.26，测试的 bpt

（bits-per-character）值降低了 0.005，这证明了模型不仅能够实现参数压缩任务，而且使效果得到了小幅度提升。

- 本文将卷积网络和长短期记忆网络两种结构的压缩算法应用于语音增强系统中，验证了基于影响函数的压缩算法在实际应用中的有效性。该语音系统主要对语音文件进行降噪处理，由于其需要部署于移动手机端平台，硬件条件有限，因此需要模型参数量降低，同时保证模型的效果不下降。系统中的语音增强模型均涉及了卷积网络结构和长短期记忆网络结构，因此可以使用本文提出的算法进行测试。结果证明，语音增强系统通过本文算法压缩之后，模型参数量显著下降，而模型精度基本不变，甚至略有提高。

1.4　论文纲要

本文主要基于影响函数的神经网络自动剪枝压缩课题进行研究，并从卷积网络、长短记忆网络两个不同的网络结构提出了模型优化加速的算法。它们基于神经网络的可解释性角度提出利用数学统计方法测算权重的重要性，推理出冗余权重，实现自动剪枝工作，并且已经成功在显示环境当中应用。全文一共分为以下六章：第一章是绪论，主要介绍了神经网络压缩加速、自动机器学习和神经网络可解释性的研究背景及其意义，同时还详细介绍了模型压缩加速的现状及难点；第二章则介绍目前针对神经网络加速的相关研究工作；第三章主要介绍基于影响函数的卷积网络自动剪枝方法；第四章介绍基于长短期记忆网络结构的影响力剪枝和 SVD 分解压缩方法；第五章主要介绍本文提出的模型优化方法在语音增强系统中的应用；第六章总结全文，同时对未来工作进行了展望。

第二章　相关工作

1890 年，心理学家 William James[33]在他的出版专著《心理学原理》中详细描述了神经细胞的传播过程，他认为神经细胞的激活是细胞所有输入叠加的结果。他的这个猜想后来获得证实，当前人工神经网络也是基于此理论所设计的。1943 年，McCulloch 和 Pitts 在发表文章中将神经元以数学形式进行描述，并描绘其数学结构。同时，证明足够简单的神经元互相连接、同步运行，可以模拟任何计算函数[34]。因此他们提出的神经元数学形式的表述被认为是人工神经网络（ANN）的起点。1949 年，Hebb[35]在他出版的《行为组织学》中描述了神经元权重的调整规划方法，开创性地提出神经网络中的信息是存储在连接的权重当中。20 世纪 60 年代感知机[36]诞生，这种感知机是由三层网络层组成的神经网络结构，是世界上第一个人工神经网络。感知机的诞生，让神经网络研究工作迎来第一次热潮。人们认为只要有足够的神经元就能解决一切问题。研究的热潮仅持续了 10 年，由于学术界大多对人工神经网络持悲观看法，这波热潮也逐渐退去。直到 1982 年，Hopfield 网络模型理论[37]的提出重新激发了人工神经网络（Artificial Neural Network，ANN ）的研究热情。

1982 年，美国科学院发表了著名的 Hopfield 网络模型的理论，不仅对 ANN 信息存储和提取功能进行了非线性数学概括，提出了动力方程和学习方程，使得 ANN 的构造与学习有了理论指导。这一研究激发了 ANN 的研究热情。随后 BP 算法的提出，解决了长期以来 ANN 计算过程中的权值自动调整的难题。从此，ANN 逐渐蓬勃发展起来，循环神经网络（Recurrent Neural Network, RNN）和深度学习 (DL, Deep Learning) 也逐渐被研究发现。现代的深度网络主要起源于 2012 年 Krizhevsky 等人提出的 AlexNet 网络[3]。此后陆续提出多种新型的网络结构，例如 ShuffleNet[13]、VGG[19]、MobileNet[38]等等。

现代深度网络的发展，使得深度学习模型存在较大的冗余。参数量的增加，提高了模型的复杂程度，这与要解决的实际问题通常不相匹配，甚至导致模型

过拟合，任务泛化性能差。同时，随着移动端和嵌入式设备的多样化部署，深度学习的应用承受了更大的挑战，这要求模型计算资源消耗更小、计算更快。所以，如果能有效的找到冗余参数，运用合理的方式将它们删减，可以极大的缩减网络的复杂程度，甚至可以缓解模型过拟合问题，提升模型实际应用效果。综上所述，神经网络压缩是深度模型能够在实际应用的重要手段。目前，很多研究提出了有效的神经网络压缩算法。不同压缩方法应对不同的场景。需要根据具体研究环境所决定。本章首先简单介绍卷积神经网络、长短记忆网络的组成部分，然后介绍相关神经网络压缩加速的概念和当下主流方向上的经典压缩方法，并分析和总结不同方向的方法所具备的优势。最后描述了影响函数的数学概念以及其思想在神经网络中的应用。

2.1 卷积神经网络

2.1.1 神经元模型

1904 年，神经元的组织结构被生物学家通过观察和实验所描绘出来[39]。生物体内的神经元拥有多个用作接收输入信息的树突组织；轴突是树突上用来与其他的神经元树突连接的部分，轴突的末端有数个轴突末梢。20 实际 50 年代，受生物神经元结构的启发，研究提出了与生物神经元原理相近的 MP 模型。神经元的数学模型包含了输入、计算功能和输出三个部分。如图2-1为神经元的数学模型示意图：图中模型有 N 个输入，加和计算功能区、激活计算功能区和一个输出，箭头表示连接，每个连接都有相应的权重。

图中 $x_i(1 \leq i \leq N)$ 表示神经元输入，w_i 表示连接权值。x_i 的每个连接的箭头都表示初始输入信号值为 x_i。信号连接传递之后，通过 w_i 加权，信号转变为 $x_i \times w_i$。每个加权过后的信号传递进入计算功能区进行求和步骤，见公式2-1。

$$y = \sum_{i=1}^{N} x_i w_i + b \tag{2-1}$$

其中，b 是偏置值，经过求和计算之后，进行激活函数激活神经元，激活函数

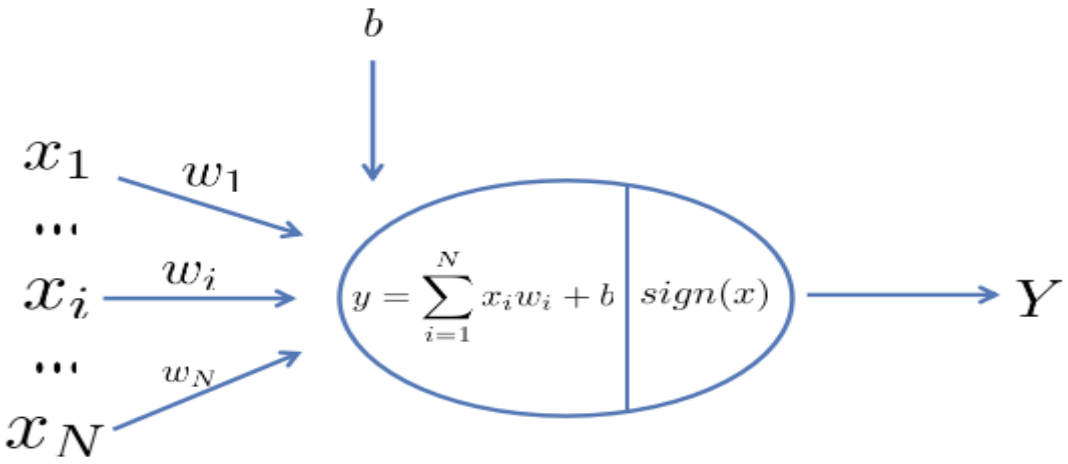


图 2-1　神经元的数学模型示意图

sign 公式如2-3:

$$f(x) = \text{sign}(x) = \begin{cases} -1, x \leq 0 \\ 1, x > 0 \end{cases} \tag{2-2}$$

因此，神经元的数学计算形式可表示为公式2-3：

$$Y = f(y) = f\left(\sum_{i=1}^{N} x_i w_i + b\right) \tag{2-3}$$

通过输入的特征信息，神经元经过两个计算功能区的计算，就能获得新的目标输出结果。但是，起初提出的 MP 模型其连接权重的值是预先设定好的，在计算中这种权重是无法改变的。直到 1949 年 Hebbina 提出的 Hebb 学习规则[35]解决了感知机权重无法自动调节的难题，同时 Hebbina 也证明了 A 神经元到 B 神经元的连接权重与 B 神经元到 A 神经元的连接权重是相同的，这一发现，为卷积神经网络的发展提供了理论基础。但由于当时硬件设备发展有限，使得模型的算力受限，感知机在十年后才诞生，卷积网络才逐渐发展起来。

2.1.2　卷积网络的组成结构

从 20 世纪 60 年代感知机诞生以来，陆续出现了许多基于卷积网络的新研究，如 20 世纪 60 年代 Hubel 和 Wiesel 提出了 Receptive fields 的概念[40]；1962 年，他们又发表了关于猫大脑的视觉系统研究[41]。过了 30 年，日本科学家福

岛邦彦设计了包含卷积层和池化层的神经网络结构[42]。1998 年，Lecun 提出了 LeNet-5 网络结构[43]，模型借用 BP 算法的思想，将其运用到神经网络的训练当中，初步形成了现代卷积神经网络的结构。在 2012 年的 Imagenet 图像识别大赛中，Hinton 组使用 AlexNet 引入全新的网络结构和创新的 dropout 方法[2]，将错误率从 25% 降低到了 15%，超越了图像识别领域的其他模型。以 AlexNet 模型结构为基础，Lecun 等人在 2013 年提出了 Dropconnect[44]概念，把错误率降低到了 11%。至此之后，CNN 结构的发展越来越复杂，结构组成也越来越多元化。由于 CNN 的发展，也带来了其他领域的更多变革。CNN 的应用也更加广泛，例如图像分割、目标检测、图像分类等多种任务。卷积神经网络的发展也进入了一个新的时期。图2-2包含了卷积神经网络的各个组成结构，包含了卷积层 (Convolution)、池化层 (Pooling)、激活函数 (Activation)、全连接层 (Fully-connected layer) 等多个部分。

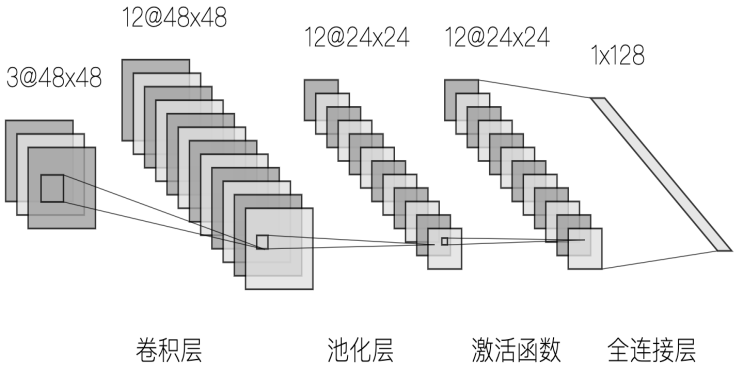


图 2-2 卷积神经网络的组成结构示意图

2.1.2.1 卷积

卷积层是卷积网络中最基础、最重要的部分。卷积在数学定义上，是一种数学运算，记 $(f * g)(n)$ 为 f，g 的卷积，其连续定义为公式2-4，离散定义为公式2-5。

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n-\tau)d\tau \tag{2-4}$$

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau) \tag{2-5}$$

根据公式2-4和公式2-5定义，表示存在两个函数 $f(x)$ 和 $g(x)$ 的卷积运算是

指对 $f(x)\cdot g(y)$ 的乘积求和（连续则求积分，离散则求和），其中 x 与 y 点只能在二维直角坐标系 $y=n-x$ 的直线上。在卷积神经网络中的卷积运算是在两个矩阵之间进行的，如图2-3所示[1]。其中，卷积操作计算过程如公式2-6所示。

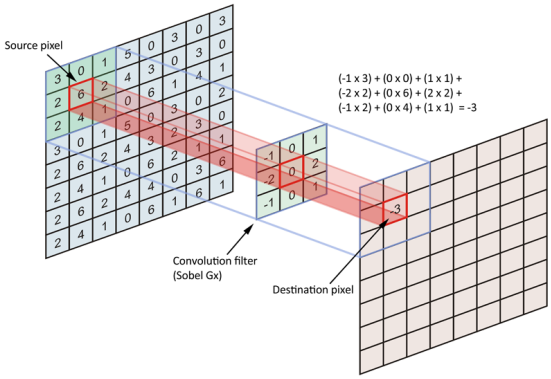


图 2-3　卷积神经网络的卷积层计算示意图

$$\begin{aligned}\mathrm{conv}_{x,y} &= (-1\times 3)+(0\times 0)+(1\times 1)+(-2\times 2)+(0\times 6)+(2\times 2)+\\ &(-1\times 2)+(0\times 4)+(1\times 1)=-3\end{aligned} \tag{2-6}$$

图 2-3 是二维卷积核和输入的二维特征图的卷积计算过程。设定输入特征为 $X^{h\times w\times N}$，其中 w 表示输入数据的宽度，h 表示输入特征图的高度。在实际的运用中，不同模型的输入数据量也不一定相同，数量由实际运算环境所决定，输入样本数设为 N。在神经网络的卷积运算中，卷积核通常是一个大小为 $W\times H$ 的矩阵，W 表示卷积核的宽度，H 表示卷积核的高度，一般情况下 $W=H$，所以卷积核通常是方阵，又被称为过滤器。这个矩阵的大小叫做感受野，过滤器的通道数 C 通常和输入的通道数 C 保持一致。假设输入 N 个样本数，那么一个卷积核是参数量为 $N\times W\times H\times C$ 的四维向量，假设记为 $K^{N\times H\times W\times C}$。通过卷积计算，卷积层通过计算最终会生成一个输出结果矩阵。上述过程如公式2-7所示：

$$f=\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{d=1}^{N}K_{i,j,d}\times X_{W+i,H+j,d}+b \tag{2-7}$$

通常一个卷积层中存在数个卷积核，具体数量由实际运算环境所决定。卷积核内的卷积通过平面堆叠组成一个大的四维卷积核向量。卷积核的通道数通

[1]图　片　来　源：https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size

常等于输入数据的通道数，而且，当前层的卷积核数量又决定了下一层网络输入的通道数。因此，删减卷积核的数量，在某种程度上可以降低卷积网络的模型参数量。卷积层的输出矩阵大小需要考虑当前卷积层中零的填充（Padding）数量，以及步长（Stride）大小，将填充数和步长设为 p、s，那么输出矩阵的大小如公式2-8和2-9所示：

$$W_{\text{out}} = \frac{W_{\text{in}} + 2 * p - W_{\text{kernel}}}{s} + 1 \tag{2-8}$$

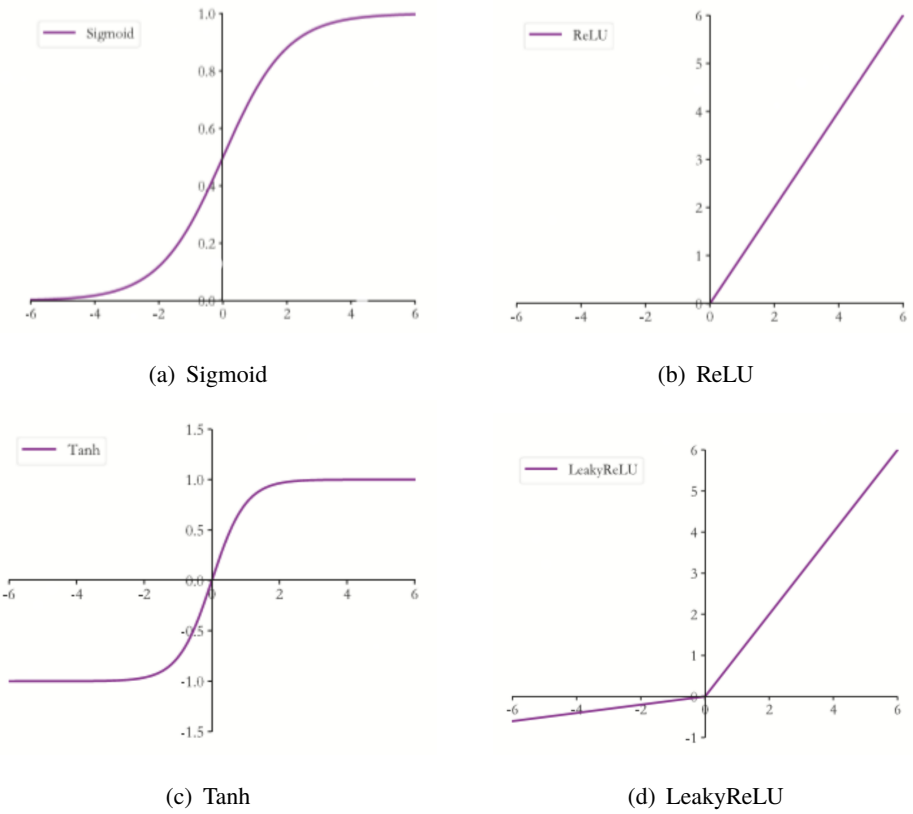
$$H_{out} = \frac{H_{in} + 2 * p - H_{kernel}}{s} + 1 \tag{2-9}$$

2.1.2.2 池化

池化又称作下采样，通常的操作是将特征图的长宽两个维度通过一个 2×2 的方形滑框，以一定的步长在图像上移动，使用设定好的运算方式对当前滑框所在位置的数据进行计算，得到的计算结果即为池化结果。经典的池化计算方法有：最大值池化（Max pooling）、平均值池化（Average pooling）、可训练池化以及高斯池化等。最大池化通常是选择滑块选中区域等最大值作为池化的结果；平均池化一般是选择滑块选中的特征图区域的数据均值作为池化的计算结果；可训练池化的方法是直接训练一个计算函数作为池化函数，通过函数对特征图选中的区域进行池化运算作为池化结果，这种做法因为较为复杂，不常被使用；而高斯池化操作则类似图像操作中的高斯模糊方式来处理特征图，这种方法也不常使用。

2.1.2.3 激活函数

激活函数的主要作用是加强对卷积层的输出结果的非线性映射，提升网络的拟合、表达能力。它在神经网络任务中的作用是至关重要的。如果神经网络中缺少激活函数，那么叠加再多的网络层数，本质上也仅仅是原始的感知机。所以，当前卷积网络结构的设计都将卷积层和激活函数结合起来。为此激活函数设计也随着卷积网络的发展，逐渐种类多样起来。经典的激活函数有：sigmid 函数、tanh 函数、ReLU 函数以及 LeakyReLU[45]函数等，它们的函数直角坐标图像如图2-4所示。



(a) Sigmoid　(b) ReLU

(c) Tanh　(d) LeakyReLU

图 2-4　几种常见的激活函数

1. 在非线性激活函数中，Sigmoid 是最常用到的函数，其数学形式如公式2-10所示。

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2-10}$$

Sigmoid 函数主要的作用是把输入的连续数值转换为 0 到 1 之间的数据，当输入数据为越小的负数，那么输出数值则越接近 0；当输入数据是越大的正数，那么输出的数据则越接近 1。但由于 Sigmoid 存在一些缺陷，因此近年来使用 Sigmoid 的工作逐渐变少。由于深度学习中梯度的反向传播行为使其计算遵循连乘法则，而 Sigmoid 导数较小，连乘的结果易于导致发生梯度消失的情况。另外 Sigmoid 函数的输出是非零均值（zero-centered）的，这将造成后一层网络的输入数据分布总体发生偏移，导致训练收敛速度变缓。即使是进行批训练模型，也仅是缓解这一问题，并不能完全避免。而 Sigmoid 计算还涉及了幂运算，较大程度增加了模型的计算消耗。因此 Sigmoid 的应用越来越少。

2. Tanh 函数全称为 Hyperbolic Tangent，其数学表示如公式2-11。Tanh 函数的出现解决了 Sigmoid 函数一直以来难以解决的输出非零均值的问题，但是

因为 Tanh 函数的几何特性和数学特性，它仍然无法解决 Sigmoid 存在的梯度消失和幂运算两大问题。

$$\mathrm{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2-11}$$

3. 由于 Tanh 函数并不能解决 Sigmoid 种种问题，研究提出了 ReLU 函数[46]，它基本解决了 Sigmoid 函数存在的问题。ReLU 的数学表示见公式2-12。在 ReLU 函数中，当输入数据是正数时，ReLU 的计算变成取最大值函数；当输入数据是负数时，则输出为 0。但 ReLU 在部分点上不可导，在不可导点可以取下梯度（sub-gradient）作为导数值。ReLU 的问世，直接解决了激活函数带来的梯度消失和梯度爆炸问题，同时由于没有幂运算的阻碍，计算的速度极快，只需要判断输入的数据是否大于 0，而且，收敛的速度比 Sigmoid 和 Tanh 函数都要更加迅速。但是 ReLU 由于其数学特性，输出数据并不是零均值数据。不仅如此，当输入数据小于 0 时，则输出为 0，这会导致 Dead ReLU 问题，即神经网络中的部分神经元很可能从始至终都不会激活，但是其参数无法更新。尽管存在这些缺陷，目前为止，ReLU 依然常被用于不同研究工作中，是设计搭建网络当中的热点尝试对象。

$$\mathrm{ReLU} = \max(0, x) \tag{2-12}$$

4. 由于 ReLU 在模型训练中容易出现 Dead ReLU 的问题，因此研究[45]提出了 LeakyReLU 激活函数。与 ReLU 的区别在于，当输入为负值时，对负值赋予了一个非零的斜率。它首次提出时，是被运用于声学模型中，该函数的表示形式如公式2-13所示。

$$y_i = \begin{cases} x_i & x_i \geq 0 \\ \frac{x_i}{a_i} & x_i < 0, a_i \in (1, +\infty) \end{cases} \tag{2-13}$$

2.1.2.4 全连接层

全连接层是卷积神经网络中的重要组成部分，通常都将其设计在卷积神经网络最后部分。全连接层主要负责担任 “分类器” 的工作。具体来说就是将前面

任务学习的“分布式特征表示”作为输入数据，通过全连接层以一定的权重映射到样本标记空间当中。实际操作中，全连接层通过矩阵乘积操作，将输入的“分布式特征表示”展成一维向量，实现映射分类的任务。但是全连接层参数十分冗余，全连接层的参数量甚至能达到全网络参数量的 80% 左右，同时全连接层在训练过程中消耗了大量的计算资源。因此全连接层的参数删减也是模型压缩的重要部分。

2.2　长短期记忆神经网络

长短期记忆网络（LSTM，Long Short-Term Memory）是 1997 年 Hochreiter 和 Schmidhuber 提出的人工神经网络[47]。它是 RNN 的一种特殊类型，其具备学习长期以来信息的能力。LSTM 与一般 RNN 的区别在于其结构中增加了门单元设计，让其具备很强的记忆力。LSTM 的设计主要是为了解决长期依赖性的问题，缓解长时间的信息遗忘，使其在不需要额外操作下，能够记住很久之前的信息。但一个 LSTM 单元有 8 个较大的权重矩阵，因此存在较多的冗余，随着循环神经网络的发展壮大，循环神经网络的参数删减工作也是势在必行的。

2.2.1　LSTM 的结构与概念

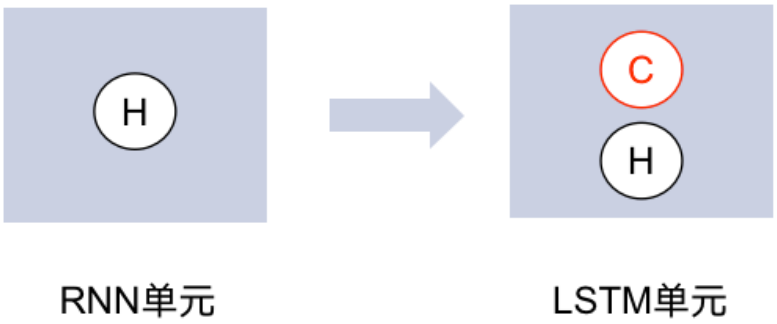


图 2-5　RNN 单元与 LSTM 单元隐含层的区别

在 RNN 的结构中，其隐含层的单元中只存在一个状态 H，因此只具有短期记忆的功能，为了让 RNN 能够长时间记住所学的信息，研究者们创造了许多 RNN 的变体。LSTM 就是其中应用最广泛的一种变体。LSTM 的具体工作主要是在隐含层的状态中间增加了一个单元状态 C，如图2-5所示，状态数增加，比

传统 RNN 更适合用于建模长距离的上下文信息，同时也缓解了训练中因为连乘问题导致的梯度消失和梯度爆炸问题。

单元间的连接如图2-6所示，LSTM 的输入值为三个：x_t、h_{t-1} 和 C_{t-1}，输出值为：h_t 和 C_t。x 表示数据输入，h 表示隐含层的输出状态值，c 表示单元的状态指，下标 t 表示时刻。

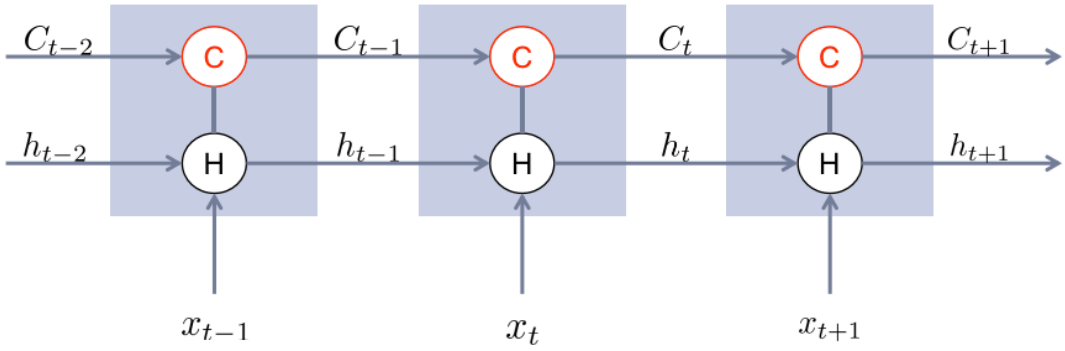


图 2-6　LSTM 的状态转移过程

LSTM 单元中单元状态 C 在长期记忆功能中起到至关重要的作用，如图2-7所示，LSTM 有三个开关控制单元的长期记忆学习状态，其中，左边开关控制的是前一个状态信息的保留情况，最下的开关用来控制当前短时状态的保留情况，右边开关用来控制当前状态计算信息的输出。

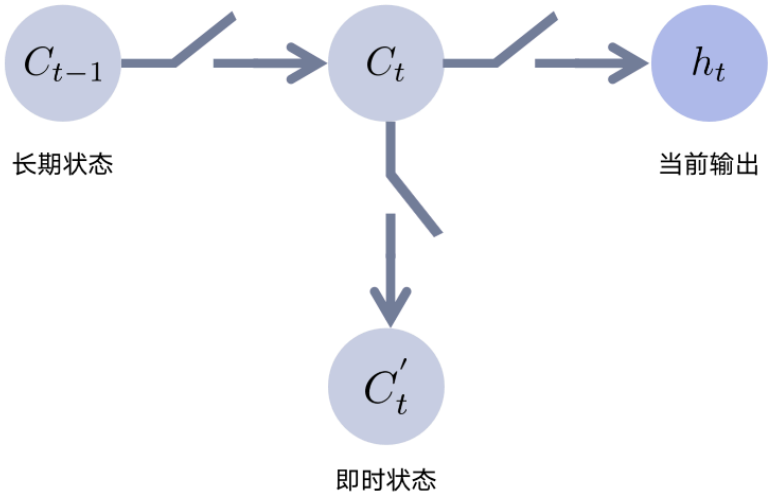


图 2-7　LSTM 单元状态控制

在 LSTM 模型单元中，控制开关具体以门（gate）形式存在于单元结构中，即 LSTM 相比于 RNN 多了输入门、输出门和遗忘门单元结构，利用这些门结构

来保证实现长期记忆的功能。如图2-8所示，输入的数据信息在通过门结构之后，会输出生成一个在 0 到 1 值域的数值，将这个数值于其他状态信息相乘可以控制状态信息的保留情况。比如希望保留全部信息时，门单元可以通过计算输出为 1 的值即可，因为 1 与任何信息相乘都等于原信息；反之，则令门单元输出 0 即可。

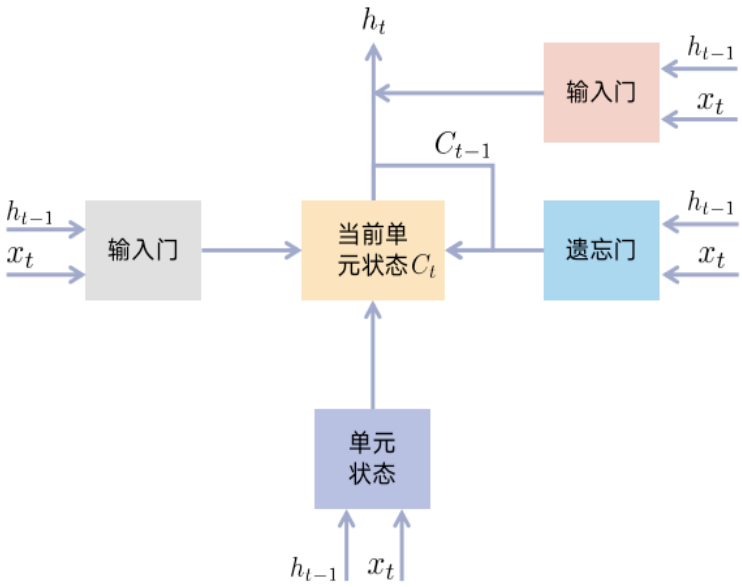


图 2-8　LSTM 内部单元门结构

在三个门中，遗忘门主要是用来控制前一个时刻输入的状态信息的保留情况，即上文所说的是控制前一个状态信息保留情况的开关。遗忘门的输入是当前时刻的输入 x_t 和上一个时刻状态输出 h_{t-1}，它们将共同控制上一个计算周期中的单元状态 C_{t-1} 的遗忘程度。输入门的主要任务是控制当前时刻的输入数据保留情况，即为上文所说的控制当前时刻短时状态的保留开关，其输入也与遗忘门相同，都是当前时刻的输入信息 x_t 和上一个时刻的状态输出 h_{t-1}，它们共同决定即时信息的保留情况，决定哪些信息值得被单元学习记忆。输出门的任务则是控制当前单元的输出状态，决定当前单元输出状态信息的保留度，即上文所述的控制当前状态计算信息输出的开关。其输入与遗忘门、输入门相同，就不再重复。输出门主要用来决定当前时刻的单元状态信息 C_t 的保留输出程度。LSTM 常用的激活函数为 Sigmoid 函数和 Tanh 函数，在卷积神经网络部分介绍过，在此就不过多赘述。

2.2.2 LSTM 的前向传播及其参数量计算

LSTM 的训练任务一般是由 GPU 搭载完成的，在一定的模型框架之下，设计建立 LSTM 网络的模型，将待训练数据分批次，逐批送入 LSTM 计算，通过运算获得预测值，计算预测值与已知的数据信息标签的差值，再将误差值通过反向传播算法，逐步更新 LSTM 单元的权重结构，直到模型的损失函数收敛或者达到了指定的迭代总次数，则训练结束。实际应用之时，模型权重已经训练结束，参数已经确定好，因此不再需要反向传播步骤，仅需完成 LSTM 的前向传播任务，由此可知，LSTM 的前向传播计算与 LSTM 的反向传播相比更加重要，它在 LSTM 的加速过程中占有重要地位。

由上文可知，LSTM 当前时刻的输出信息是作为下一时刻的输入数据，且结构中加入了三个门单元设置，这让模型结构比普通 RNN 来说较为复杂得多。LSTM 单元的详细内部构造如图2-9，其中，σ 指代 Sigmoid 激活函数，x_t 表示当前的输入信息，h_{t-1} 指代上一时刻的输出信号，C_{t-1} 指代上一时刻单元中单元状态数据，C_t 则指代目前时刻的单元状态信号，h_t 表示目前时刻的输出信号。

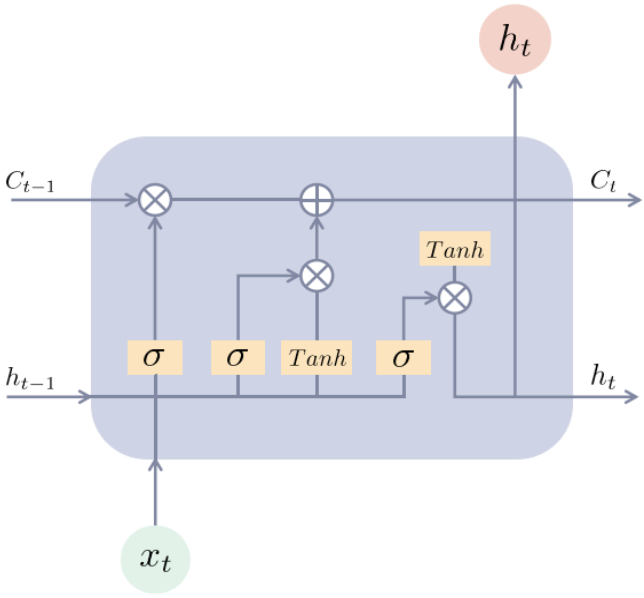


图 2-9 LSTM 内部单元结构图

在 LSTM 的前向传播过程中，需要先确定之前的前一个时刻的单元状态信号有哪些信息需要保留，仅保留重要的信息，丢弃不重要的数据，这部分工作通过遗忘门完成。具体来说，主要是使用 h_{t-1} 和 x_t 作为输入与遗忘门的参数矩阵进行相乘计算，然后利用 Sigmoid 函数激活，得到遗忘门的输出结果，数学表示

如公式2-14所示。遗忘门的输出和前一时刻的状态信号输出 C_{t-1} 相乘，其中遗忘门的输出值大小在 0~1 之间，这样可以控制单元状态信号的保留度，1 表示完全保留上一时刻的状态信号，而 0 则表示完全遗忘之前的状态。

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right) \tag{2-14}$$

通过遗忘门计算之后，再确认当前输入的即时信号需要存储保留哪些信息。这个工作主要由两个部分组成，第一，将 h_{t-1} 和 x_t 作为输入门的输入，与输入门的参数矩阵求积，然后利用 Sigmoid 函数激活得到输入门单元的输出信息，来确定目前时刻的输入信号将保留哪些单元状态信息。这一过程的数学表示如公式2-15所示。第二，将 h_{t-1} 和 x_t 作为输入，与单元状态信息的参数矩阵求积，利用 Tanh 激活函数激活之后获得目前时刻下的单元状态 C_t。这一过程的数学表示如公式2-16所示。

$$\mathrm{i}_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) \tag{2-15}$$

$$\tilde{c}_t = \tanh\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) \tag{2-16}$$

确认当前的即时信号的保留度之后，还需要更新目前时刻的单元状态信号 C_{t-1}，将其转换为 C_t，具体来说，将上个时刻的单元状态值 C_{t-1} 和遗忘门的输出信号 f_t 计算向量内积，其计算结果表示需要遗忘的信息；将当前即时信号的保留度 C_t 与输出信号 i_t 计算向量的内积，其计算结果表示需要补充到记忆中的新知识；将遗忘之后保留的信息和新补充的知识信息求和即可得到当前时刻的单元状态信息。整个过程如公式2-17所示。

$$c_t = f_t \odot c_{t-1} + \mathrm{i}_t \odot \tilde{c}_t \tag{2-17}$$

在更新网当前单元状态的信号数据之后，最后一步就是确定当前时刻的最终输出信息，这个输出信息是在状态单元值 C_t 的基础上决定的。具体来说，将 h_{t-1} 和 x_t 分别与输出门的参数矩阵求积，在利用 Sigmoid 激活函数来激活获得输出信号。输出值在 0 到 1 之间。这一过程的数学形式如公式2-18所示。然后，将上一步计算更新得到的当前时刻单元状态信息 C_t 通过 Tanh 激活函数激活，将结果映射到-1 到 1 之内，同时与输出门的输出信息求积，其计算结果即为最终

当前时刻单元的输出信号。这一过程的数学形式如公式2-19所示。

$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right) \tag{2-18}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{2-19}$$

LSTM 是 RNN 一种特殊变种，因此也具有 RNN 的特点，即每个时刻都有一个结构相同的长短期记忆单元，且每一时刻单元内的权重矩阵都是相同的，如图2-10所示，图中是以时间序列为例子将单元逐个展开。

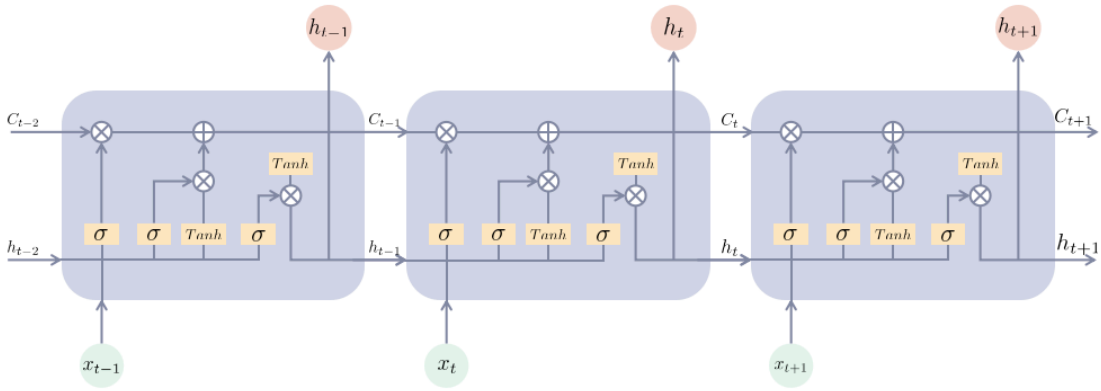


图 2-10　以时间序列展开的 LSTM 结构

经过推理过程中的了解发现，可知 LSTM 单元中共有 8 个权重矩阵，其中 4 个权重矩阵与输入信息 x_t 有关联，分别是 W_{ix}、W_{fx}、W_{cx} 和 W_{ox}，因此每个矩阵的行数与 x_t 中的元素数量相等，列数则是隐含单元的大小；还有 4 个权重矩阵与上一时刻隐含层的输出信息有关系，它们分别为 W_{ih}、W_{fh}、W_{ch} 和 W_{oh}，这四个矩阵的行列数相等，都是隐含单元的数量，整个推理过程如图2-11所示。

设定输入信息 X 的维度是 N，隐含层单元数为 M，则当前时刻的状态输入 $[h_{t-1}, X_t]$ 的向量长度为 $N+M$，因此输入门的输出状态信息向量长度为 M，即 $[h_{t-1}, X_t]$ 的向量维度为 $(1, M+N)$，与参数矩阵 $[W_{fx}, W_{fh}]$ 相乘可得到大小为 $(1, M)$ 的输出向量，由矩阵的乘积规律可以反推出 $[W_{fx}, W_{fh}]$ 的矩阵维度大小为 $[M+N, M]$。将得到的 $(1, M)$ 的输出向量与输入门的偏置向量相加，修正输出结果，因此可知偏置的维数也与输出向量相同，即 $(1, M)$，所以整个输入门的参数总量为 $((M+N)\times M + M)$。同样的方法推理遗忘门、和输出门，可知其参数总量与输入门相同。综上所述，单个 LSTM 单元的参数总量为 $4\times((M+N)\times M + M)$。

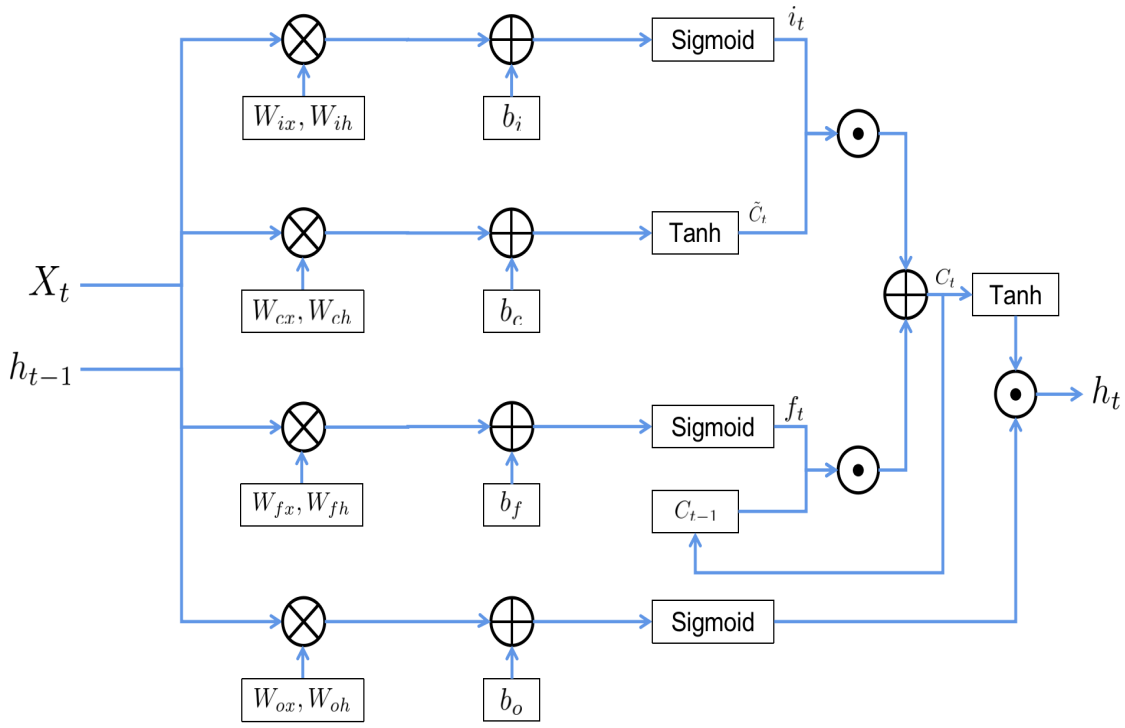


图 2-11　LSTM 的前向计算过程

2.3　模型压缩算法

本节主要介绍卷积神经网络常见的压缩方法，分为：紧凑型的模型结构设计、参数共享、低秩分解、知识蒸馏以及参数剪枝等主要的几种模型压缩加速方法。本文与参数剪枝工作关联较大，因此本节介绍的重点是基于模型参数量剪枝的加速方法。

2.3.1　紧凑型的轻量化结构

为了能够有效降低模型的参数量冗余程度，可以设计紧凑型神经网络结构来代替原本的网络模型结构，最终达到网络轻量化的目的。基于这种轻量化思想，近年来，关于紧凑型网络结构的设计的研究层出不穷。ResNet 结构的设计就是一个典型例子，它是何凯明组提出的一种轻量化结构，利用残差块结构降低整个网络的复杂度。随着网络层数的增加，使得神经网络模型的性能得以大幅度提升，但这种性能提升带来了模型参数量和计算时间大的幅度增加。但是 ResNet 网络结构针对这类问题设计了 shortcut 模型，结构优化更加简单易操作，所以它不仅不会因为网络深度的增加而提升模型的复杂度，相反甚至还会提升

整个网络的性能。

SqueezeNet 网络也是基于模型结构的紧凑型改造设计的，它原网络中的卷积块结构替换成了 Fire Module，而 Fire Module 的结构由两个层组成，分别是 Squeeze 层和 Expand 层，图2-12展示了整个 Fire Module 的结构。由图中可知，Squeeze 层只有 1×1 的过滤器，与原始模型中 3×3 的过滤器相比，大大降低了模型的复杂度；而 Expand 层中有 1×1 和 3×3 两种尺寸的过滤器，跟原始模型结构中只有 3×3 的过滤器比较，既降低了网络本身的计算复杂程度，又提升了模型的表征能力。总而言之，SqueezeNet 的思想就是利用 Fire Module 中的 Squeeze 层和 Expand 层结构在保证模型的表征能力的同时，较大程度降低模型的计算复杂度，实现网络的有效压缩加速。

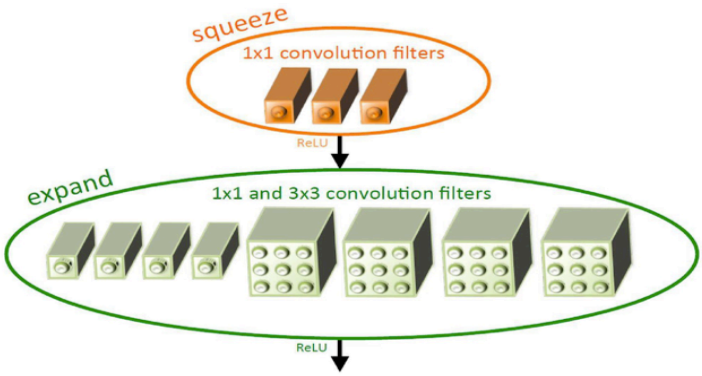
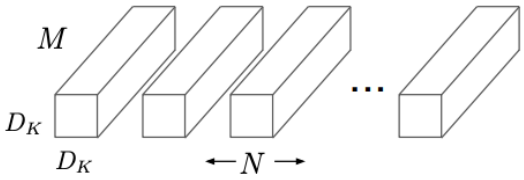
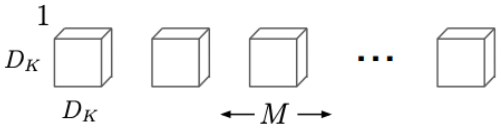


图 2-12 SqueezeNet 的 Fire Module 示意图[20]

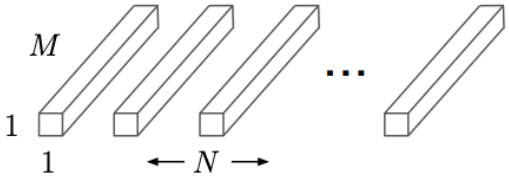
随着终端设备越来越轻量化，模型的设计逐渐倾向于更轻量、更快速的模型。MobileNets 是 Howard 等人[38]提出的一种能够运行在移动终端设备的网络模型。其结构原理是把传统的卷积结构分为深度卷积和逐点卷积。深度卷积主要用于学习提取空间特征；逐点卷积则主要负责融合通道之间的信息，结构如图2-13参数极大程度的减少了冗余参数，降低了模型的浮点数计算量。相似的，旷视科技提出的 ShuffleNet 原理[13]与 MobileNets 是类似的，即把传统卷积模型结构进行拆解，得到分组卷积和通道重排卷积两个结构，分组卷积主要用来提取组内的空间特征数据信息，通道重排卷积则是为了缓解分组卷积导致的模型性能下降所提出的解决方式，用来加强组间的信息互通，从而保证模型的性能。



(a) 标准卷积核



(b) 深度卷积



(c) 逐点卷积

图 2-13　MobileNet 中深度卷积与逐点卷积计算示意图[38]

2.3.2　参数共享

参数共享是神经网络压缩的重要手段，其主要思想是建立一种映射，让多个参数能够共用一个数据表征。其中，参数量化是最为常见一种参数共享的表现形式。参数量化的核心思想是将原始全精度的参数使用更低位数的表示形式，从而节省物理空间存储。例如 Gupta 组[48]提出的随机约束（stochastic rounding）就是使用 16*bits* 形式作为数据表征用来存储模型参数，实现模型的压缩。由于模型中，卷积层和全连接层的结构并不完全相同，因此也有工作将它们分开进行量化处理。例如吴佳祥等人[49]先将全连接层固定，将卷积层进行量化处理，在训练完成输出的数据误差最小化，再微调整个模型，最大程度恢复网络性能，最后进行全连接层的量化工作。也有研究工作为了追求极致的压缩，而去选择设计模型参数二值化的量化[50]表示，这将导致模型权重的大幅度精度损失，因此通常还需要额外的设置。

2.3.3 低秩分解

低秩分解的主要思想是将矩阵或者张量分解开，用分解后的矩阵和张量来近似原网络的表征能力。整个模型训练中，卷积层的运算是一个复杂度最高的计算过程。因此分解成近似的卷积核，可以大幅度降低网络中的参数冗余，分解过程如图2-14所示。

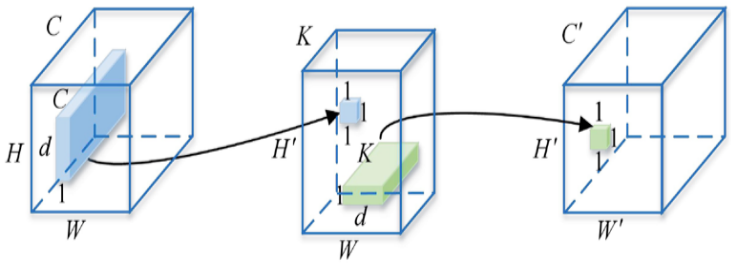


图 2-14 有两个低秩因子的卷积计算

2013 年 Denil[51]提出了一种低秩分解技术，该技术能够仅依靠原模型的极小部分参数（5% 的参数量）就能够重构原始网络。这个工作证明了低秩分解技术的有效性，以及网络模型的参数拥有极大的冗余。基于这项技术的有效性，随后涌现了越来越多的关于低秩分解的研究。例如，CP 分解技术[22]，其本质上是 SVD 分解的变种，也是将卷积核转化为低秩的近似矩阵。主要做法是利用 3 个秩是 1 的矩阵代替原始的卷积核矩阵，然后依靠微调技术，将网络性能恢复到原本的状态。也有研究利用新的权重初始化方法结合卷积核矩阵进行低秩分解，来达到模型轻量化的目的。还有其他主流的压缩方法与低秩分解相结合的工作，就不一一赘述。低秩分解技术虽然能够在保证最小精度损失的情况下，较大程度的删减模型的参数冗余，但由于其分解过程对计算资源消耗较大，因此较少广泛应用。

2.3.4 知识蒸馏

知识蒸馏，又被称为知识迁移学习，指的是原有的已学习的知识对另一个新学习造成的影响。知识迁移方法主要是把大型网络（也称为教师模型，Teacher Model）中的细粒度知识迁移到小型网络（也称为学生模型，Student Model）中，利用对小型网络进行的训练，使其能够近似拥有大模型完成小粒度任务的任务效果。知识迁移的过程如图2-15所示，大模型指的是复杂模型，小模型指的是简

单模型；hard target 表示的是输入信息所附带的对应标签，soft target 表示的是输入信息在输入大模型后，模型的输出结果结合超参 λ 处理之后，得到的软性标签值；超参 λ 表示的是温度系数，它能够对大模型的输出进行软性（soft）调节，λ 值越小，输出值之间的差值越大，soft 程度越小。

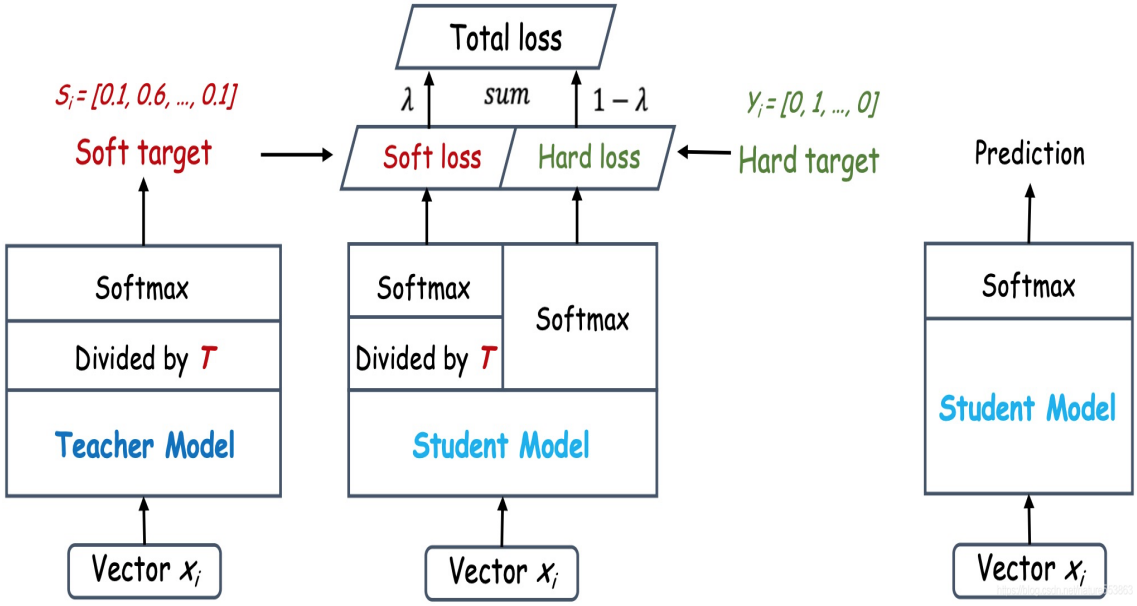


图 2-15　知识蒸馏的训练过程

知识迁移过程如下：第一，先训练大模型（教师模型），通过 hard target 预训练；第二，先设定一个较大的 λ，通过 λ 和 softmax 结合对训好的教师模型进行 soft 处理，获得 soft target 结果；第三，对小模型（学生模型）进行初步简单的训练，先计算小模型的输出信息和 hard target 的交叉熵，然后计算教师模型 soft target 和学生模型的 soft target 的交叉熵，最后将两个交叉熵结果通过加权求和，作为知识迁移的训练损失。

2.3.5　参数剪枝

模型的参数剪枝的核心问题是如何选择一个合适的指标来正确判断网络中的冗余参数，使其在保证网络预测性能的同时能够尽可能的删减冗余的参数。剪枝方法的分类方式有很多，本章中则是按照剪枝的细粒度划分种类，可以划分为结构化剪枝和非结构化剪枝两种。

2.3.5.1 结构化剪枝

结构化剪枝继续按照细粒度划分又可以分为：网络层剪枝、卷积核剪枝、通道剪枝、卷积核的形状剪枝。卷积核剪枝指的是删减整个卷积核参数，即删减输出通道的数量。通道剪枝则是删减卷积核内的输入通道数量，卷积核形状的剪枝工作主要指的是对当前层中所有卷积核的相同位置进行删减。分类如图2-16所示。

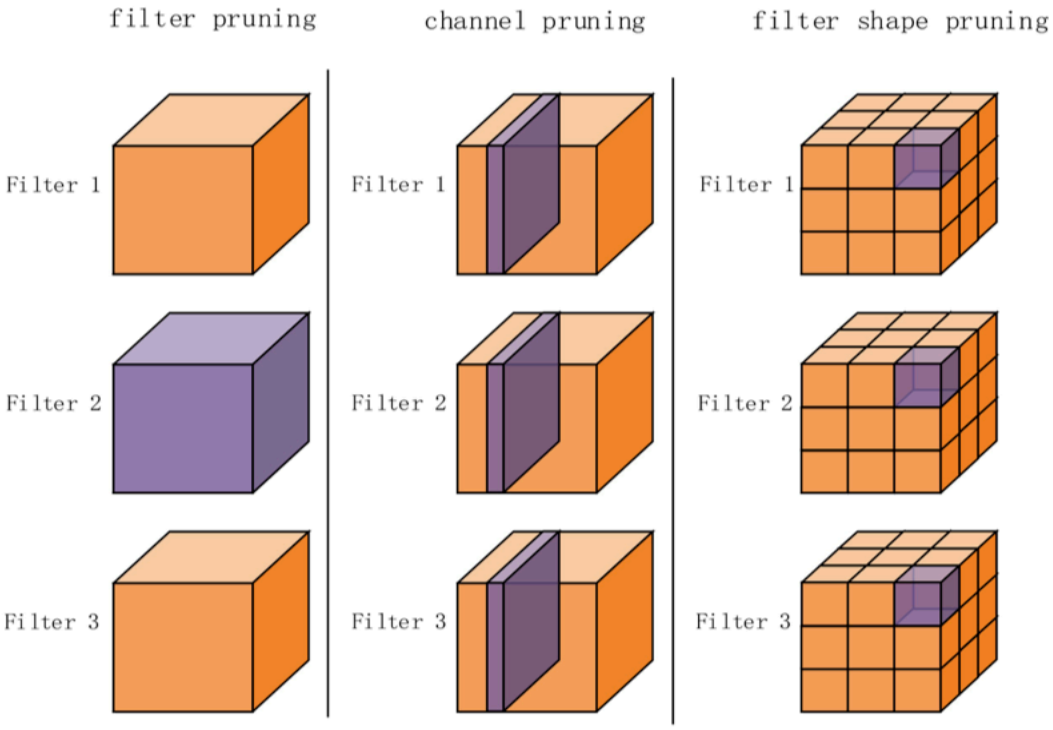


图 2-16 结构化剪枝示意图，由左至右依次为卷积核剪枝、通道剪枝、卷积核形状剪枝

图2-16中的三维张量表示当前卷积层的权重矩阵，其中一个三维张量表示一个 filter 的参数矩阵，一个层的 filter 数量表示的是输入的通道数，我们以紫色部分表示已剪枝参数；filter pruning 则表示以输入的通道为单位进行修剪，那么剪枝之后的效果如图中左图所示；channel 表示的是当前层的输出通道，因此 channel pruning 表示的是以输出通道为单位进行剪枝，那么剪枝之后的效果如图中的中图所示；filter shape pruning 表示的是在每个 filter 的三维张量中，以固定的张量块大小为基本单位删除 filter 中的权重张量，那么剪枝之后的效果如图中右图所示，其中紫色的小立方体表示的是删除的三维权重张量。

结构化剪枝通常是删减整个过滤器或者是部分过滤器，同时也会删除其对应连接。特征图经过修剪，之后每层对应的卷积核都会进行相应的移除。而且

剪枝之后的模型不会出现稀疏结构，微调之后，模型的性能也会重新变回原来的状态。结构化剪枝能够删减较多的冗余参数，使得模型的参数量大幅度降低，但性能损失较小，因此是一类较为理想的剪枝方法。

2.3.5.2　非结构化剪枝

在非结构剪枝中，剪枝细粒度为单个权值剪枝。假设当前层的输入通道为3，输出通道也为3，如图2-17所示，那么就有3个Filter权重张量，因此参数张量大小为3×3×3×3，假设我们进行非结构化剪枝，用紫色表示删除该权重，那么图中则为修剪之后的结果。这种方法通过一个指标和一个确定的阈值，在预训练模型中筛选出冗余的参数，对这些冗余的单个参数进行一一删除。然后重新训练被剪枝的模型，最终得到一个稀疏的卷积核，但是这种剪枝方法如果没有底层硬件和计算库的支持，则剪枝的效果较差，因此这类剪枝方法较少被运用。

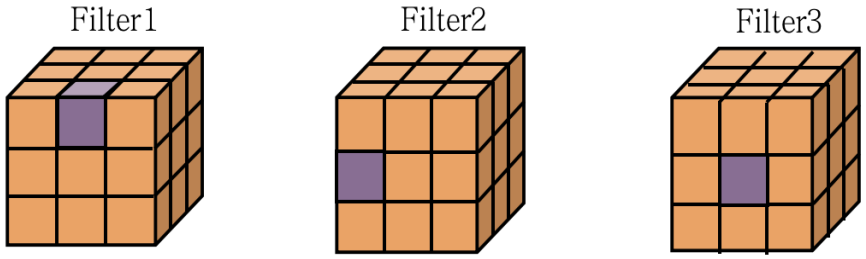


图 2-17　非结构化剪枝示意图，剪掉 Filter 中的单个参数

2.4　影响函数

2.4.1　稳健统计方法

稳健统计[31]是针对从各种概率分布中提取的数据（尤其是对于非正态分布），是具有良好性能的统计数据。针对许多常见问题（例如估计位置、尺度和回归参数）已经开发了稳健的统计方法。一个目的是希望生成不会受到异常值过度影响的统计方法。另一个目的是旨在与参数分布有小偏差时提供具有良好性能的方法。例如，鲁棒方法适用于具有不同标准差的两个正态分布的混合数据；

在这个模型下，像 t 检验这样的非鲁棒方法效果很差。稳健统计旨在提供模拟流行统计方法的方法，但这些方法不会受到异常值或其他与模型假设的微小偏离的不当影响。在统计学中，经典的估计方法严重依赖于在实践中通常无法满足的假设。特别是，通常假设数据误差是正态分布的，至少是近似的，或者可以依靠中心极限定理来产生正态分布的估计值。不幸的是，当数据中存在异常值时，经典估计器在使用细分点和影响函数（如下所述）进行判断时，通常具有非常差的性能。通过在混合模型下检查所提出估计器的抽样分布，可以从经验上研究影响函数中看到的问题的实际效果，其中混合少量（1-5% 通常足够）的噪声数据。例如，可以使用 95% 的正态分布和 5% 的正态分布的正态分布的混合数据，其均值相同，但标准差明显更高（表示异常值）。在影响函数中看到的问题的实际影响可以通过在混合模型下检查提议的估计量的抽样分布来凭经验研究，其中一个混合了少量（1-5% 通常就足够了）噪声数据。例如，可以使用 95% 的正态分布和 5% 的正态分布的混合，均值相同但标准差显着更高（代表异常值）。

2.4.2 经验影响函数的定义

对于数据的稳健性测量，其基本工具包括击穿点、影响函数和灵敏度曲线。而本文工作主要使用了影响函数来测量模型参数权重的鲁棒性。经验影响函数是评估器对样本中任何一个点的值的依赖性的度量。它是一种无模型的度量，因为它只是依赖于使用不同的样本再次计算估计器。如图2-18是 Tukey 的 biweight 函数[52]，它正好描述了一个” 好” 的经验影响函数应该是什么样的例子。在数学

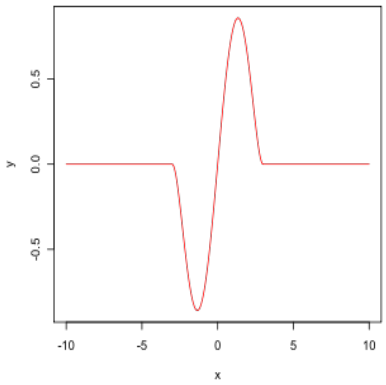


图 2-18 一个” 好” 的经验影响函数示例

术语中，影响函数被定义为估计器空间中的向量，而估计器又被定义为作为总体子集的样本：$(\Omega, \mathcal{A}, P)$ 是一个概率空间；$(\mathcal{X}, \Sigma)$ 是一个可测空间（状态空间）；

θ 是维度的参数空间；$p \in \mathbb{N}^*$;(Γ, S) 是一个可测量空间。那么经验影响函数内容如定义1所示。

定义 1 设 $n \in \mathbb{N}^*$ 和 $X_1, \ldots, X_n : (\Omega, \mathcal{A}) \to (\mathcal{X}, \Sigma)$ 是独立同分布的，$\left(x_1, \ldots, x_n\right)$ 是这些变量的样本，$T_n : \left(\mathcal{X}^n, \Sigma^n\right) \to (\Gamma, S)$ 是一个估计器，让 $i \in \{1, \ldots, n\}$，观察 i 时的经验影响函数定义为公式*2-20*:

$$EIF_i : x \in \mathcal{X} \mapsto n \cdot \left(T_n\left(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n\right) - T_n\left(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n\right)\right) \tag{2-20}$$

上述定义表示只要将样本中的第 i 个值替换为任意值，可以观察估计器的输出情况。换句话说 EIF 被定义为将点 x 添加到样本的估计器上按 $n+1$ 倍而不是 n 缩放的效应。

2.4.3　影响函数和灵敏度曲线

影响函数可以不通过仅依赖数据的方法测量，而是仅使用随机变量的分布进行计算。这种办法与之前所见的做法大不相同。假设现在尝试稍微改变数据的分布时，估计器会发生以下情况：它假设一个分布，并测量对这个分布变化的敏感性。这种做法与经验影响的工作相同，经验影响假设一个样本集，并测量样本中变化的敏感性。

具体来说，设 A 是 Σ 上所有有限有符号度量的集合的凸子集。想要估计 A 中分布 F 的参数 $\theta \in \Theta$.。设函数式 $T : A \to \Gamma$ 是某个估计器 $\left(T_n\right)_{n \in \mathbb{N}}$ 序列的渐近值。假设这个函数是 Fisher 一致的[2]，即 $\forall \theta \in \Theta, T\left(F_\theta\right) = \theta$。这意味着在模型 F 上，估计器序列渐近测量正确的量。让 G 表示为 A 中的某个分布。假设当数据不完全遵循模型 F，而是另一个略有不同的“走向” G 时，会发生什么？根据影响函数的定义分析，可以得到公式2-21。

$$dT_{G-F}(F) = \lim_{t \to 0^+} \frac{T(tG + (1-t)F) - T(F)}{t} \tag{2-21}$$

[2]在统计学中，以 Ronald Fisher 命名的 Fisher 一致性[53]是估计器的理想属性

这是在 $G-F$ 的方向上 F 处 T 的单侧 Gateaux 导数[54]。让 $x \in \mathcal{X}$，Δx 表示将质量 1 变为 $\{x\}$ 的概率度量，令 $G=\Delta_x$，然后，影响函数公式2-22定义：

$$IF(x;T;F) := \lim_{t\to 0^+} \frac{T\left(t\Delta_x + (1-t)F\right) - T(F)}{t} \tag{2-22}$$

公式2-22描述了 x 点的无穷小噪声数据对当前测量的估计值的影响，由噪声数据的质量 t（观测中噪声数据引起的渐近偏差）标准化。其中可以知道，对于一个鲁棒估计器，需要一个有界影响函数，即当 x 变得任意大时，它不会变为无穷大。

影响函数测量公式可以引用到对模型权重的重要性评估。例如，可以在模型某个权重中加入小部分扰动，计算模型的损失变化情况。然后通过影响函数公式2-22的定义，求导得到这个权重参数对于这个模型的影响力。然后通过计算权重影响力，确定模型的参数删减策略。

2.5 本章小结

本章详细的介绍了卷积神经网络、LSTM 网络和影响函数的相关知识，首先通过介绍卷积神经网络的结构和 LSTM 的模型结构，证明了模型压缩加速的必要性和实施的可行性。然后介绍了当前针对神经网络压缩的几种主要的方法分类，依次是紧凑型轻量化网络设计、参数共享、知识蒸馏、参数剪枝等四类。最后介绍了一种稳健的统计方法——影响函数，其定义说明了影响函数的测量法可以通过影响函数定义测得权重在模型中的影响力，且能够通过影响力判断参数权重的冗余程度。

第三章　基于影响函数的卷积网络剪枝压缩方法

3.1　CNN 剪枝压缩的意义

在卷积神经网络中，随着网络深度增加，模型更加复杂，其参数的冗余程度加大。模型剪枝方法是最为广泛的模型压缩方法，这类方法能够是模型在原有模型的基础上获取最为精简的结构，且易于与其他方法结合（例如强化学习、元学习和网络搜索等），而通道修剪最有效的方法是根据每个神经元的重要程度进行修剪。

通道剪枝的核心问题在于剪枝指标的选择。在以前的工作中，通过考虑单个层或多个连续层的统计信息来修剪神经元（例如修剪当前层以最小化下一层的重建误差、当前层的权重的绝对值大小和输入特征值大小等）。但是，这些工作无法消除重建误差过程中不同批次数据对模型的影响，而且参数和特征的数值信息无法直接与权重重要性相关联。更合理的办法是消除不同批数据测量的差异，准确地测出权重的影响力。具体来说，我们通过影响函数（一种来自稳健统计的经典技术）来测量模型权重的重要性，对权重加入极小的扰动，对扰动求导，并返回其导数，从而确定每个参数的负责程度；根据负责程度的大小排序，剪掉负责度较小的权重，使得模型在预测中性能受影响最小；而模型的反向传播过程类似于影响函数的求导工作，因此可以借助模型的反向传播实现影响力的计算，最终以影响力为指标完成模型的剪枝。

3.2 基于影响力的通道剪枝

本章算法以影响函数能够测量数据影响力的理论为基础，提出了一种基于影响函数测量权重影响力的方法。本文将数据集分为多批，对于每批数据测量当前层的每个参数影响力，得到影响力矩阵之后通过卷积层降噪处理，以消除不同数据造成的对测量结果对影响。另外，为了更快的确定模型每层的通道剪枝策略、降低模型微调时间，本章算法通过对损失函数增加正则项来监督模型训练，实现训练和微调相结合的端到端自动剪枝工作。吴建鑫[55]证明了训练和微调处理步骤可以相互促进。两个步骤的结合使用将他们的优势充分发挥，而且这可以大大节省训练时间。

3.2.1 定义

首先，定义本章中运用到的符号参数。假设 CNN 网络有 N 层网络层，设定 F_l 表示第 l 层的 filter 数量，C_l 表示第 l 层的输出通道数，w 表示当前层的卷积核宽度，h 表示当前层的卷积核高度。设定 CNN 模型第 l 层的权重集合 W_l 表示如公式3-1所示；本章算法需要用到扰动矩阵，设定 M_l 则表示第 l 层的扰动矩阵，形式如公式3-2所示，M_l 矩阵中的值都在 0 和 1 的取值范围内，也被用来表示第 l 层的连接状态矩阵，因为扰动矩阵对权重矩阵起到掩码作用；而 $W_l^{'}$ 表示 W_l 经过 M_l 的 Hadamard[1]计算之后的权重矩阵，如公式3-3所示，其中，$\odot$ 表示 Hadamard 积，而 W_l、M_l 和 $W_l^{'}$ 这三个矩阵的大小均为 $F_l \times C_l \times w \times h$。

设定 $w_{l,(f,k,i,j)}$ 表示第 l 层中第 f 个过滤器，第 k 个输出通道中第 i 行第 j 列的连接权重；设定 $m_{l,(f,k,i,j)}$ 表示第 l 层中第 f 个过滤器，第 k 个输出通道中第 i 行第 j 列的扰动值；因此 $w^{'}_{l,(f,k,i,j)}$ 表示 W_l 中的第 l 层中第 f 个过滤器，第 k 个输出通道中第 i 行第 j 列的权重进行 $w_{l,(f,k,i,j)}$ 和 $m_{l,(f,k,i,j)}$ 乘法计算。$W_{l,(f,k,i,j)}$ 表示要对 W_l 中的连接权重 $w_{l,(f,k,i,j)}$ 进行处理；$M_{l,(f,k,i,j)}$ 表示将 M_l 矩阵第 l 层中第 f 个过滤器，第 k 个输出通道中第 i 行第 j 列的扰动值改为 $m_{l,(f,k,i,j)}$，其余值为 1；对矩阵 W_l 中的单个权重 $w_{l,(f,k,i,j)}$ 进行扰动值计算转化为 $w^{'}_{l,(f,k,i,j)}$，用

[1]Hadamard 乘积是矩阵的一类运算，若 $A = (a_{ij})$ 和 $B = (b_{ij})$ 是两个同阶矩阵，若 $c_{ij} = a_{ij} \times b_{ij}$, 则称矩阵 $C = (c_{ij})$ 为 A 和 B 的 Hadamard 积，或称基本积。

$w'_{l,(f,k,i,j)}$ 替换了权重 $w_{l,(f,k,i,j)}$ 的值，转化为 $W'_{l,(f,k,i,j)}$。

$$W_l = \left\{ w_{l,(f,k,i,j)} : 0 \le f \le F_l, 0 \le k \le C_l, 0 \le i \le w, 0 \le j \le h \right\} \tag{3-1}$$

$$M_l = \left\{ m_{l,(f,k,i,j)} : 0 \le f \le F_l, 0 \le k \le C_l, 0 \le i \le w, 0 \le j \le h \right\} \tag{3-2}$$

$$\begin{aligned} w'_{l,(f,k,i,j)} &= w_{l,(f,k,i,j)} \times m_{l,(f,k,i,j)}, \\ W_l^{'} &= W_l \odot M_l \\ W_l^{'} &= \{ w'_{l,(f,k,i,j)} : 0 \le f \le F_l, 0 \le k \le C_l, 0 \le i \le w, 0 \le j \le h \} \end{aligned} \tag{3-3}$$

3.2.2　权重影响力

由于本章的目标是修剪网络通道，因此应从通道中测量卷积层的参数影响力，根据影响力大小进行过滤器的剪枝。显然，关键是放弃不重要的参数并保留重要的参数，而直接找到不重要的过滤器是一个 NP 难的问题，但本文的工作可以找到近似的方法。

影响函数定义及相关研究[30]证明了在黑盒模型中某一数据 $X \in \mathcal{D}$ 中加入一个极小倍的扰动，通过经验风险变化对扰动值求导，当扰动趋向于 0 时，这一求导过程即为测量数据 X 的影响力。对于第 l 层的权重，加入与权重相同尺寸的倍数矩阵，作为扰动矩阵；扰动矩阵与权重矩阵在前向传播中进行元素乘，记录损失变化；保留和剪枝该权重所导致的损失差，通过反向传播的方式，对于每个权重的扰动值进行求导，对应的梯度结果可以作为权重影响函数。权重扰动的对应梯度值可以直接表示权重 W 的影响力值，称为权重 W 对于模型的“权重影响力”。以第 l 层为例，假设模型需要解决一个优化问题，则 CNN 模型的损失函数如公式3-4所示。

$$\begin{aligned} J(W_l; \mathcal{D}) &= L\left(M_l \odot W_l; \mathcal{D}\right), \\ M_l &= \mathbf{1}^{F_l \times C_l \times w \times h}, 0 \le l \le N \end{aligned} \tag{3-4}$$

其中 $L(\cdot)$ 是损失函数，我们将 M_l 初始化为全 1 矩阵，矩阵大小尺寸为 $F_l \times C_l \times w \times h$，这表示没有权重被修剪。那么，在删除第 l 层的某个权重 $w_{l,(f,k,i,j)}$

后，CNN 模型的损失函数如公式3-5所示。

$$J(W^{'}_{l,(f,k,i,j)};\mathcal{D}) = L\left(M_{l,(f,k,i,j)} \odot W_l;\mathcal{D}\right),$$
$$M_{l,(f,k,i,j)} = \begin{cases} 0, & p=f,q=k,r=i,s=j, \\ 1, & \text{otherwise.} \end{cases} \tag{3-5}$$

其中，对 $W_{l,(f,k,i,j)}$ 中的单个权重 $w_{l,(f,k,i,j)}$ 进行处理，让 $M_{l,(f,k,i,j)}$ 中的扰动值 $m_{l,(f,k,i,j)}$ 改为 0，其余值为 1；将 $M_{l,(f,k,i,j)}$ 和 W_l 进行 Hadamard 乘得到权重矩阵 $W^{'}_{l,(f,k,i,j)}$，这意味着权重 W_l 的第 f 个 filter、第 k 个通道中第 i 行、第 j 列的权重被裁剪。根据经验影响函数定义，对于修剪的权重引起的损失差异可以直接用作权重的影响函数的评估函数，结合公式3-4和公式3-5，可得损失差如公式3-6所示。

$$\Delta J_l(\mathcal{D}) = J(W_l;\mathcal{D}) - J(W^{'}_{l,(f,k,i,j)};\mathcal{D}) = L\left(M_l \odot W_l;\mathcal{D}\right) - L\left(M_{l,(f,k,i,j)} \odot W_l;\mathcal{D}\right) \tag{3-6}$$

公式3-6中的 ΔJ_l 表示删减权重 $w_{l,(f,k,i,j)}$ 造成的损失差；我们将公式3-6作为评估函数，进行权重 $w_{l,(f,k,i,j)}$ 的影响力计算，其过程如公式3-7所示。由影响函数定义可知，单个权重的影响测量与其他权重的影响测量没有直接关系，因此通道中单个权重的损失差可以直接加和到公式3-7中作为整个通道的影响力。然而，公式3-6中的 $M_{l,(f,k,i,j)}$ 是一个二进制矩阵，二进制矩阵的存在使得公式3-6无法直接求导。因此，对公式3-6进行近似转换，转换结果如公式3-7所示。

$$\frac{\partial \Delta J_l(\mathcal{D})}{\partial M_{l,(f,k,i,j)}} \approx \lim_{\alpha\to 0} \frac{\partial \Delta J_l(\mathcal{D})}{\partial \tilde{M}_{l,(f,k,i,j)}} = \lim_{\alpha\to 0} \frac{\partial\left(L\left(M_l \odot W_l;\mathcal{D}\right) - L\left(\tilde{M}_{l,(f,k,i,j)} \odot W_l;\mathcal{D}\right)\right)}{\partial \tilde{M}_{l,(f,k,i,j)}},$$
$$\tilde{M}_{l,(f,k,i,j)} = \begin{cases} 1-\alpha, & p=f,q=k,r=i,s=j,0<\alpha<1, \\ 1, & \text{otherwise.} \end{cases} \tag{3-7}$$

$$w^{'}_{l,(f,k,i,j)} = \alpha \cdot w_{l,(f,k,i,j)} \tag{3-8}$$

其中，将 $\tilde{M}_{l,(f,k,i,j)}$ 的矩阵 $M_{l,(f,k,i,j)}$ 中的值 $m_{l,(f,k,i,j)}$ 改为 α，其余值不变；根据影响函数测量定义，将扰动值 α 和单个权重 $w_{l,(f,k,i,j)}$ 进行计算得到新的权重 $w^{'}_{l,(f,k,i,j)}$，过程如公式3-8所示。我们对权重处理之后所造成的损失差求导，来

测量权重 $w_{l,(f,k,i,j)}$ 的影响力。我们通过累积同一通道中每个权重的影响力，可以获得当前通道的权重影响力，然后使用测量通道的权重影响作为修剪策略的决策指标。整个计算过程如公式3-9和公式3-10所示。

$$\mathbf{Inf}_{l,(f,k,i,j)} \approx \lim_{\alpha \to 0} \frac{\partial \Delta J_l(\mathcal{D})}{\partial \tilde{M}_{l,(f,k,i,j)}} = \lim_{\alpha \to 0} \frac{\partial \left(L\left(M_l \odot W_l; \mathcal{D}\right) - L\left(\tilde{M}_{l,(f,k,i,j)} \odot W_l; \mathcal{D}\right)\right)}{\partial \alpha} \tag{3-9}$$

$$\mathbf{Inf}_{l,k} \approx \lim_{\alpha \to 0} \sum_{f}^{F_l} \sum_{i}^{w} \sum_{j}^{h} \frac{\partial \Delta J_l(\mathcal{D})}{\partial \tilde{M}_{l,(f,k,i,j)}} \tag{3-10}$$

如公式3-10中所示，我们通过对通道中的单个权重影响力 $\mathbf{Inf}_{l,(f,k,i,j)}$ 加和来获得通道权重的影响力 $\mathbf{Inf}_{l,k}$。公式3-9可通过反向传播来实现求导计算单个权重 $w_{l,(f,k,i,j)}$ 的影响力，这将在3.3.1中详细描述。

3.3 模型训练

在模型训练部分，本章将介绍该算法训练流程，并且对反向传播过程、二值化处理过程进行详细介绍。根据影响函数测量定义，该算法把公式3-9运用于反向传播当中，通过加入 Mask 卷积层来测量权重的重要性。

3.3.1 前向传播与反向传播

由于影响力可以通过公式3-9确定。公式3-9的工作可以使用反向传播来获得 α 的导数结果。为了使影响力可以进一步用作修剪策略，该算法制定了影响力的计算方案，如图3-1和图3-2所示。

在通道中添加一个与原滤波器大小尺寸相同的辅助滤波器，辅助滤波器初始化为常量 1，我们称为 Mask 滤波器，它与通道中原滤波器通过 Hadamard 乘积结合，组成新的滤波器。图3-2详细描述了输入 X 在带 Mask 滤波器的卷积层中的前向传播过程。当输入数据 X 的维度是 $1 \times f \times W \times H$，而权重矩阵 W 和 Mask 矩阵的维度为 $f \times k \times w \times h$，在每个 Filter 的前向计算中，Mask 矩阵都和 Filter 权重矩阵进行元素乘计算，然后与输入进行矩阵相乘，每个 Filter 得到一个输出记为 $O_i, 0 \le i \le f$，其维度为 $1 \times k \times W \times H$，每个 Filter 的计算结果求和得到当前层的最终结果 O，其维度为 $1 \times k \times W \times H$。Mask 卷积核矩阵数值

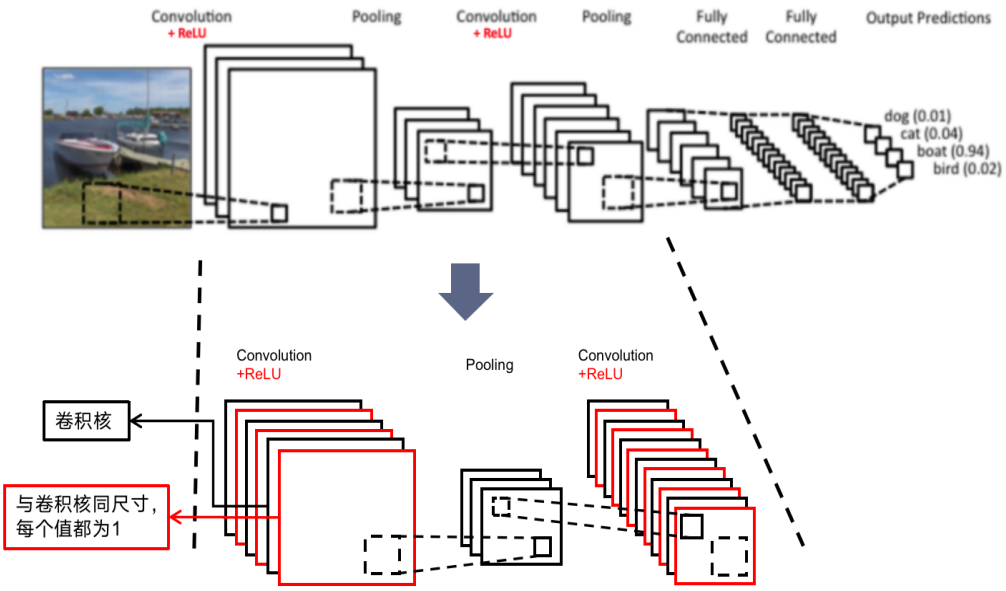


图 3-1 加入与原始卷积核同尺寸 Mask 卷积核

初始化为 1 的原因在于：在加入 Mask 卷积核之后，需要保证模型进行前向传播时，对整个模型的前向输出效果没有影响。对于 Mask 卷积核矩阵与权重矩阵进行 Hadamard 乘积计算，相当于将原始滤波器中的每个权重都乘上一个 0 到 1 的倍数。由于反向传播过程中，我们可以得到 Mask 矩阵的求导结果，这个求导过程正好实现了公式3-9的计算。因此，我们只要获取 Mask 滤波器的反向传播梯度，就能够获得当前层的每个权重对整个模型的影响力。我们可以控制原始过滤器不保存梯度计算，仅在传播过程中保存 Msak 滤波器的梯度。

将 Mask 卷积核矩阵计算出的梯度被作为影响力，为了不同输入数据测算出的影响力带来的偶然性误差影响剪枝策略的决策，我们将得到的梯度矩阵通过卷积层的学习进行降噪，并将得到的结果进行二值化处理，最终得到对应的剪枝策略，二值化的处理过程在3.3.4中详细介绍。

3.3.2 训练流程

在训练阶段，本章的算法为逐层的端到端剪枝。通过选择删除通道影响力小于阈值的通道，生成一个通道目标修剪策略向量，应用于待剪枝模型中。以模型实现分类任务为例，假设当前待剪枝的模型层为 l，那么通道的目标剪枝策略向量和实际剪枝策略向量的维数是 $[1, C_l]$，其值以 0 或 1 表示，向量内的一个值

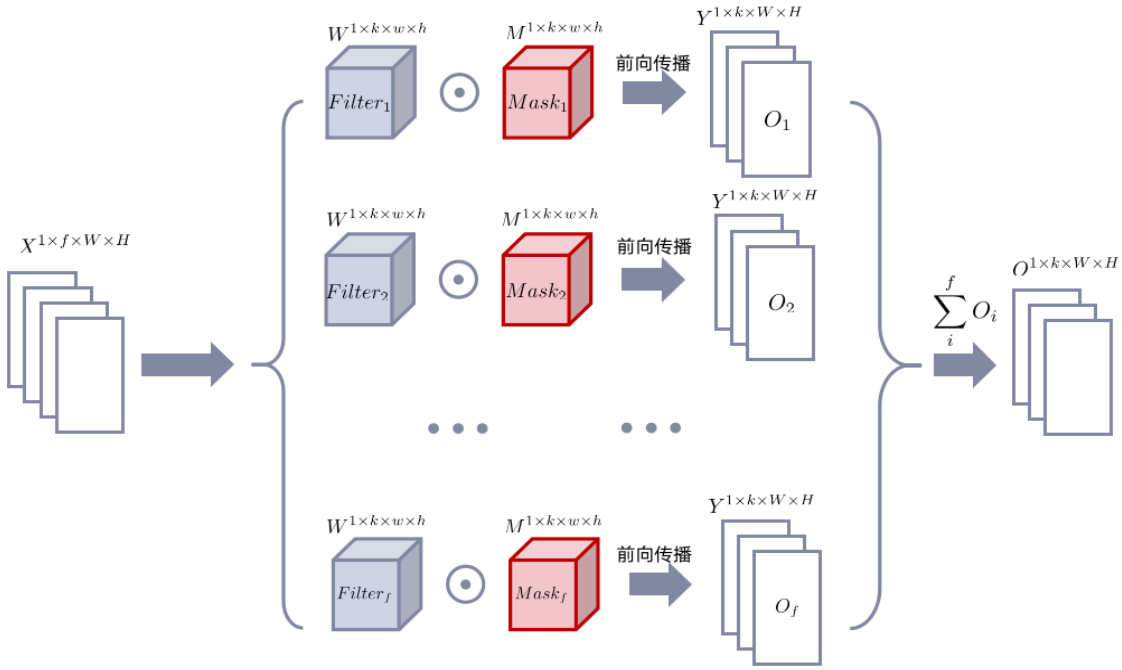


图 3-2　Mask 卷积核与原始卷积核的前向传播细节过程

对应一个通道的剪枝状态，如果是 1，则保留对应通道，如果是 0，则表示删减对应通道。如图3-3所示，当前待剪枝层为 l，设定 T_l 表示通道目标剪枝策略，是剪枝策略学习的目标剪枝策略，用二进制向量表示，即向量的值只为 0 和 1；设 e_l 为通过卷积层 **K** 学习之后，维数是 $[1, C_l]$ 的影响力处理的中间向量；设 E_l 为 Sigmoid 函数处理之后的类通道策略向量，E_l 向量维数为 $[1, C_l]$，其值介于 0-1 之间；设定 B_l 为 E_l 向量经过确定阈值强制转换之后，得到的二值化向量，即值都为 0 或者 1，作为实际的剪枝策略向量。

如图3-3所示，取待剪枝模型结构，通过正态分布进行参数初始化。首先为卷积核配置同尺寸且参数值初始化为 1 的 Mask 卷积核；在模型 l 层中，通过当前批数据前向传播之后，根据公式3-9的原理，进行反向传播计算 Mask 卷积核对应的梯度值，这个梯度值矩阵即为对应的原卷积核参数的影响力；提取 Msak 卷积核梯度矩阵，作为影响力矩阵 $\mathbf{Inf}_l$；将 Mask 卷积核梯度以通道为单位累加求和，通过阈值强制转换，得到目标剪枝策略向量 T_l；然后将影响力矩阵 $\mathbf{Inf}_l$ 转换为概率矩阵 $\mathbf{Inf}_{l,rate}$；再将概率矩阵 $\mathbf{Inf}_{l,rate}$ 送入一层卷积层（设为 **K**）进行学习，输出维度为 $[1, C_l]$ 的向量 e_l；将获取的 e_l 经过 Sigmoid 函数处理，实现 0 到 1 之间的映射，得到初步的通道剪枝策略向量 E_l，获得 E_l 之后，通过设定阈值，强制将 E_l 转换为二进制值（只有 0 和 1 两种数值）的向量 B_l，则 B_l 就是

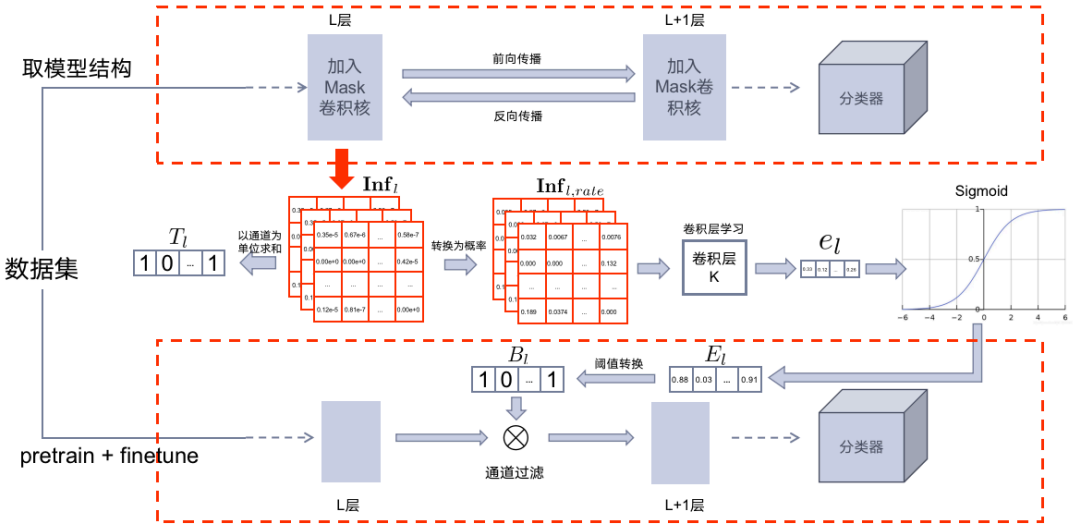


图 3-3 基于影响力的卷积网络剪枝的训练过程

待剪枝模型实际训练时应用的通道剪枝策略。在预训练模型学习剪枝策略的过程中，第 l 层参数根据向量 $\boldsymbol{B}_l$，来确定通道的保留情况，运用到模型的通道连接中，然后判断模型损失函数的变化情况，再作出相应的策略调整。

需要注意的是，为了确定目标修剪策略向量 $\boldsymbol{T}_l$ 的阈值设置，首先对整个模型的影响力以通道为单位求和，将求和的数值进行排序，保留前 n 个通道数，如公式3-11所示。然后保留的通道中影响力最小的数作为阈值。

$$n = \sum_{l} \boldsymbol{C}_l \times r \tag{3-11}$$

其中 $\boldsymbol{C}_l$ 表示第 l 层的通道数，r 表示整个模型的压缩率。$\boldsymbol{T}_l$ 的阈值设置方式解决了不同层之间通道影响力差异过大的问题。此外，这种阈值确定的修剪策略可能导致层中的所有通道都被修剪。当这种情况发生时，选择仅对当前层的通道影响力进行排序，通过排序减去影响较小的通道，并保留全局压缩率 r 的 0.2 倍的通道数。

最后，在实际训练中为了减少修剪方案的偶然性，在模型中应用并不直接将不直接使用向量 $\boldsymbol{B}_l$，而是使用实际剪枝策略 $\boldsymbol{E}_l$ 向量作为通道掩码的形式，并利用从输入数据计算的每个通道的特征和 $\boldsymbol{E}_l$ 以通道掩码的形式进入 Hadamard 乘法计算，就是所谓的” 假剪枝”。通过” 假剪枝” 进行修剪和重新连接，不断更新 $\boldsymbol{E}_l$ 中的连接状态值，直到损失函数收敛，当 $\boldsymbol{E}_l$ 代码向量稳定时（通常数值会

向 0 和 1 两端靠近），将 $\boldsymbol{E}_l$ 通过阈值强制转换为二进制的实际剪枝策略向量 $\boldsymbol{B}_l$，然后应用到模型当中。基于网络当前层的修剪方案，稀疏通道模型具有很好的精度。

3.3.3 损失函数

考虑到修剪模型训练过程中的损失函数比标准的交叉熵损失函数更复杂，我们将采取更多步骤来控制其收敛过程。在训练过程中，以分类模型为例，我们逐层进行修剪。算法设计新的损失函数，它能够使模型不仅关注分类损失，而且关注模型每一层的修剪策略。计算目标剪枝策略向量 $\boldsymbol{T}_l$ 和实际剪枝策略向量 $\boldsymbol{E}_l$ 之间的欧氏距离，并将获得的值用作分类模型的正则项，损失函数如公式3-12所示。

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \lambda \left\| \boldsymbol{E}_l - \boldsymbol{T}_l \right\|_2^2 \tag{3-12}$$

在损失函数公式3-12中，以目标剪枝策略向量 $\boldsymbol{T}_l$ 为教师，指导卷积层 $\mathbf{K}$ 的学习，使卷积层在提取通道影响力信息特征时能更接近目标 $\boldsymbol{T}_l$，同时又具有泛化性。公式中的正则系类似于 L1 和 L2 正则化，它引导了 $\boldsymbol{E}_l$ 的变化趋势。显然，为了保证待剪枝模型的分类任务损失和修剪策略的损失比例能够平衡，损失函数需要一个超参 λ，因此对 λ 设计了动态变化规则如公式3-13所示。

$$\lambda = \begin{cases} 5.0 \times \left| \frac{T_l}{|C_l|} + \frac{B_l}{|C_l|} - 1 \right|, & 1 - \frac{B_l}{|C_l|} \geq \frac{T_l}{C_l} \\ 0, & \text{otherwise.} \end{cases} \tag{3-13}$$

其中 $\boldsymbol{C}_l$ 表示当前层 l 的原始通道数；设定 $\boldsymbol{B}_l$ 为 $\boldsymbol{E}_l$ 向量经过确定阈值强制转换之后，得到的二值化向量，即值都为 0 或者 1，作为实际的剪枝策略向量。在公式3-13中，$\frac{B_l}{|C_l|}$ 表示通道实际保留率，$\frac{T_l}{|C_l|}$ 表示当前层的通道目标保留率。如果实际通道保留率超过目标保留率，那么我们应该降低模型修剪策略的关注度，而更关注模型的分类（或回归）任务准确性。如果实际通道保留率低于通道目标保留率，则应适当加大 λ 的值，以使模型更加注重剪枝策略的调整。具体来说，在实际通道保留率超过目标压缩率时，我们将 λ 设置为 0，否则将 λ 的值增加一定的倍数。

3.3.4 二值化

由于参数重要性的测量会影响网络连接的状态，因此将参数影响力用作修剪的决策指标，需要通过处理，将其映射到 0 或 1 的二进制代码向量，这对于保证修剪质量至关重要。训练中，通常是与小型批处理池操作相结合，二值化可确保小批处理中的所有示例最终都可以转换为相同的唯一索引代码。因此，这里需要使用倍数缩放的 Sigmoid 映射函数来生成近似的二进制代码，如公式3-14所示。

$$E_l = \text{Sigmoid}\left(\beta e_l\right) \tag{3-14}$$

其中，e_l 为通过卷积层 $\mathbf{K}$ 学习之后的向量，E_l 为 Sigmoid 函数处理之后的类通道策略向量。根据公式3-14，以 e_l 作为输入，E_l 作为映射函数的输出。为了提高方法的鲁棒性，引入了超参数 β 以倍数方式增强或削弱输入数据。β 值用作超参数来控制输入值的离散程度。通过逐渐增加 β 值，Sigmoid 函数可以将 e_l 向量逐渐映射到接近 0 和 1 两端的近似二进制值。

当在模型的每次迭代中获得实际二进制通道剪枝策略向量 E_l 时，将暂时修剪影响较小的通道，同时保留其他影响较大的通道。显然，Sigmoid 函数对最终的压缩策略具有重要影响。当 β 很小时，该模型允许卷积层 $\mathbf{K}$ 充分学习影响力分布。卷积层在经过完全的学习之后，将 β 逐渐增大，从而使 e_l 的离散程度逐渐增加。当 β 足够大时，近似二进制值最终将变为 0 或 1。也就是说，修剪是在微调过程中完成的，应该修剪哪个过滤器完全由网络本身决定。这种微调二值化过程有助于获得更准确的模型。当某些通道的影响力变小（$0.5 \rightarrow 0$）时，相应的过滤器将逐渐停止更新。与此同时，其他通道的的影响力也会越来越大（$0.5 \rightarrow 1$），这将迫使网络更加关注留存下来的过滤器。

由于 E_l 在训练过程中会受到输入张量、权重值和 β 的影响，如果收敛速度较慢，则可能的原因是 β 太小，E_l 可能难以收敛甚至不可能收敛到二进制值；即使收敛，E_l 也可能有严重的数据偏移，停留在一个小值。例如，所有元素都小于 0.5，但永远不能拉回 1 左右。遗憾的是，由于输入的差异，不同层中的 β 可能会有很大差异。因此，不可能为所有层找到合适的静态 β。基于上述观察结果，提出了一种自适应方案来调整 β 的近似值。这可以通过首先初始化 β 值，然后根据训练情况动态约束输入的离散程度来完成。在初始阶段将 β 的值设置为较小，

这可能会减慢修剪策略的拟合速度。由于 β 过大会加快修剪策略的确定，而卷积核 $\mathbf{K}$ 还未充分学习，很可能导致修剪策略退化为随机选择。预训练后，卷积层 $\mathbf{K}$ 具有提取影响力分布特征的能力，并逐渐增加 β 值以增加数据离散度，使 e_l 的映射逐渐加速并收敛成二进制策略向量 E_l。

二值化过程的另一个主要优点是现在可以实现动态修剪和连接，修剪可以在模型微调期间完成。如果与 E_l 元素对应的通道代码为 0，则我们知道对于任何输入数据，其激活值始终为 0。相反，如果 E_l 与元素对应的通道代码为 1，则激活值不会更改。因此，在策略向量 E_l 收敛到二进制后，删除所有修剪的块和修剪的过滤器不会改变网络的预测效果。

3.4　实验效果与分析

在本节中，我们进行了广泛的实验，以评估本章算法在卷积模型的压缩效果；卷积模型常被运用于图像分类任务中，因此我们通过图像分类模型去测评模型的压缩效果。我们在3.4.2节和3.4.3节中介绍了我们的主要实验结果。最后，在3.4.4节中，我们讨论了超参数在我们的方法中的作用。

3.4.1　实验配置

3.4.1.1　数据集和基模型

在实际实验中，相同的修剪方法对不同数据集的影响可能不同。因此，我们评估了两种数据集中我们方法的效果。在实验中，我们分别使用 CIFAR-10 和 CIFAR-100 两个数据集[56]。此外，我们还尝试了两种广泛使用的深度 CNN 结构：VGG-16 和 ResNet-56。我们回顾了一些经典方法，如 CCP[57]使用通道间依赖性来筛选通道组合，并要求插入其他分类器，又如 NSP[58]使用用极化正则化的方式进行结构化剪枝，这种方法比使用 L1 正则化的剪枝获得更好的结果。

我们的代码实现基于 PyTorch 和 Torchvision 代码库。此外，每个卷积层在 VGG-16 和 ResNet-56 的基础中添加了 BN[59]层。我们在训练过程中动态调整了超参，分别为：损失函数超参 λ、二值化 β 以及压缩率 r，以控制减少的参数量和 FLOPs。对于 ResNet-56，我们只修剪具有残差连接的块。

表 3-1 VGG-16 和 ResNets-56 在 CIFAR-10 上的原始模型参数量、FLOPs 和分类精度

方法	参数量	FLOPs	Top-1 Acc.(%)
VGG-16	14.73M	314.031M	92.11
ResNet-56	23.52M	1.305G	93.89

3.4.1.2 任务指标

对于卷积网络，本章实验以分类任务进行实验验证，因此对于模型性能的指标，我们以分类任务的任务指标作为参考。对于分类任务指标，我们以 Top-1 的精确度进行比较，精确度越高，证明分类性能越强。而对于模型的剪枝性能，我们以模型的参数下降率（Param Drop Rate）和 FLOPs 下降率（FLOPs Drop Rate）两个指标作为衡量标准；参数下降率指的是剪掉的参数量占原参数量的百分比；FLOPs 下降率指的是剪掉的 FLOPs 计算量占原 FLOPs 计算量的百分比；在分类精度不下降的情况下，参数下降率和 FLOPs 下降率越大越好。由于在之前的工作中[58,60-62]，大多记录的是模型的 FLOPs 下降率，因此本章算法主要比较 FLOPs 的下降率，再通过参数下降率作为辅助参考。

3.4.2 CIFAR-10 的实验结果分析

我们在 VGG、ResNet 两个主流神经网络结构中评估了本章的方法，本节主要介绍模型运用于 CIFAR-10 数据集中的效果分析。VGG 一开始是被 ImageNet 这样的大型数据集所设计的模型，在实际实验中，针对 CIFAR-10 这样的数据集，采用的是 VGG 的一个变体。对于 ResNet，使用了 56 层的预激活瓶颈结构（ResNet-56）。两种架构的参数量、FLOPs 和分类的精度如表3-1所示。

表3-2总结了在 VGG-16 上的实验结果。其中，本章算法在 VGG-16 中的参数量降低了 80.18%，FLOPs 降低了 57%。此时经过剪枝处理的模型其精度甚至比原预训练模型精度更高 0.28%；当参数量降低了 87.69% 时，FLOPs 降低了 48%，此时精度也仍然比原模型高 0.07%；以表3-2中的其他工作为例，如 NSP 算法，其 FLOPs 的下降量比经过本算法处理的模型更少，但精度却不如本算法的提升多；而 FPGM 算法[60]中虽然 FLOPs 下降量比本模型更少，但是却带来了较大的精度损失。因此本章所提的算法应用于 VGG-16 的模型中，其 FLOPs 删减量和精度变化都明显优于表3-2中的其他工作。

表 3-3 总结了在 ResNet-56 上的实验结果。其中，当参数量降低了 59.03%

表 3-2　VGG-16 在 CIFAR-10 上的剪枝算法性能比较

方法	FLOPs Drop Rate	Top-1 Acc. ↑(%)	Params Drop Rate
FPGM[60]	34%	-0.04	-
NS[61]	51%	-0.26	-
NSP[58]	54%	0.04	-
Ours($r = 0.3$)	**57%**	**0.28**	87.69%

表 3-3　ResNet-56 在 CIFAR-10 上的剪枝算法性能比较

方法	FLOPs Drop Rate	Top-1 Acc. ↑ (%)	Params Drop Rate
NS[61]	48%	-0.53	-
CP[63]	50%	-1.00	-
CCP[57]	47%	-0.04	-
NSP[58]	47%	0.03	-
Our($r = 0.4$)	**48%**	**0.82**	59.03%
ResRep[62]	53%	0.00	-
Our($r = 0.3$)	**53%**	**0.43**	78.69%

时，模型的 FLOPs 降低了 48%。此时 ResNet-56 在经过剪枝处理的模型其精度甚至比原预训练模型精度更高 0.19%。以表3-3中的其他工作比较，其中 NS 方法的 FLOPs 下降量于本算法相同，但是 NS 带来了相对较大的精度损失，而本章算法的精度反而是有所提升的；在 NSP 算法中，虽然模型的 FLOPs 下降量比本章算法小，但是模型的精度提升也不如本章算法。由此可得，本章所提的算法应用于 ResNet-56 的模型中，其 FLOPs 删减量和精度变化都明显优于表3-3中的其他类似的剪枝工作。

综上所述，基于影响函数的剪枝算法在相同压缩情况下，其精度和加速比都具有明显优势，这种剪枝算法对于普通的卷积网络结构的压缩表现出了明显的优越性。

3.4.3　CIFAR-100 的结果分析

本节主要介绍 VGG-16 和 ResNet-56 在较大的 CIFAR-100 数据集上的实验结果，我们通过实验来验证算法的有效性。首先需要了解基模型的原始参数量、FLOPs 以及精度，其数据如如表3-4所示。由于 CIFAR-100 相比于 CIFAR-10 来说较大，其分类数更多，因此需要要求模型的表征能力更强。这与 CIFAR-10 的实验是不同的，为了考验本章算法在更大模型中的有效性，我们将算法运用于

表 3-4 VGG-16 和 ResNets-56 在 CIFAR-100 上的原始模型参数量、FLOPs 和分类精度

模型	参数量	FLOPs	Top-1 Acc.(%)
VGG-16	34.015M	333.31M	72.44
ResNet-56	23.705M	1.305G	79.41

表 3-5 VGG-16 在 CIFAR-100 上的剪枝算法性能比较

模型	FLOPs Drop Rate	Top-1 Acc.↑ (%)	Params Drop Rate
COP[64]	43%	-0.82	-
NS[61]	38%	0.37	-
NSP[58]	43%	0.42	-
Ours($r = 0.2$)	**45%**	**1.08**	73.74%

CIFAR-100 的任务中进行尝试。

对于 VGG-16 的模型，其实验结果如表3-5所示，当模型的参数量降低到 73.74% 时, FLOPs 也下降了 45%。与 NS 模型[61]相比, 本章算法具有更高 FLOPs 下降率，NS 的 FLOPs 下降率为 38%，而我们的算法为 45%，此时 NS 模型的 Top-1 精度提升了 0.37%，我们算法的 Top-1 精度提升了 1.08%；再如 NSP 模型，其 FLOPs 下降率与本章的算法略低 2%，且 Top-1 精度提升也不如本章的模型高。因此可以验证本算法运用于 VGG-16 模型，在 CIFAR-100 数据集中的运用效果比最新压缩模型的效果更好。

对于 ResNet-56 模型, 其实验结果如表3-6所示, 当模型参数量降低到 47.7%, FLOPs 也下降了 25%。与 NSP 算法相比，NSP 在 FLOPs 下降 25% 时，Top-1 精度下降了 0.06%, 本章算法的模型在 FLOPs 下降 25% 时, 其 Top-1 的精度反而提升了 0.09%；本章的算法再对比 NS 算法可以知道，我们的算法在更高的 FLOPs 下降率的情况下，模型精度下降相较于 NS 更少的多。因此，可以证明本章算法运用于 ResNet-56 模型，在 CIFAR-100 数据集中的运用效果比最近的压缩模型效果更好。

综上所述，本章算法运用于 VGG-16、ResNet-56 这类常见的卷积模型中，在 CIFAR-100 的数据集实验中，其精度压缩率相比于最新成果，其结果仍然具有极大的优势。

表 3-6　ResNet-56 在 CIFAR-100 上的剪枝算法性能比较

模型	FLOPs Drop Rate	Top-1 Acc.↑(%)	Params Drop Rate
NS[61]	24%	-1.09	-
NSP[58]	25%	-0.06	-
Ours($r = 0.6$)	**25%**	**0.09**	47.70%

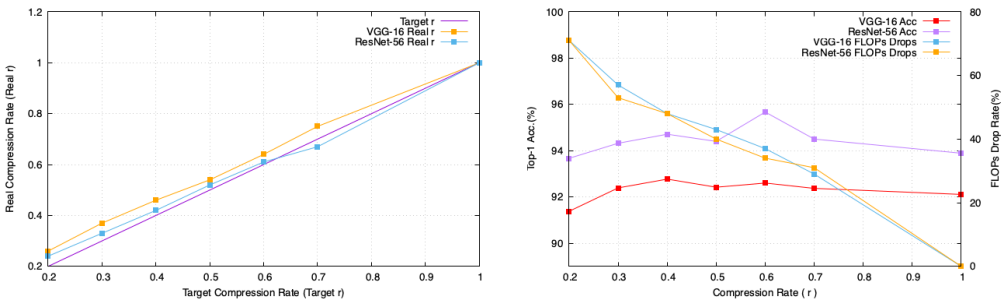
表 3-7　在 CIFAR-10 数据集中不同目标压缩率 r 对 VGG-16 的压缩性能比较

Target Compression Rate	Real Compression Rate	FLOPs	FLOPs Drop Rate	Params	Params Drop Rate	Top-1 Acc.(%)
1.0	-	314.031M	-	14.728M	-	92.11
0.7	0.75	222.696M	29%	8.139M	44.74%	92.37
0.6	0.64	196.987M	37%	5.726M	61.12%	92.60
0.5	0.54	179.204M	43%	4.092M	72.22%	92.42
0.45	0.50	171.454M	45%	3.446M	76.60%	92.29
0.4	0.46	162.574M	48%	2.908M	80.26%	92.77
0.3	0.37	136.123M	57%	1.957M	86.71%	92.39
0.2	0.26	89.581M	71%	1.019M	93.13%	91.38

3.4.4　消融实验

由于3.3.3和3.3.4提到本章算法需要压缩率 r、损失函数的 λ 和二值化的 β 初始化三个超参。在训练中模型对于 λ 是动态调节的，根据任务精度损失和剪枝策略损失共同决定，且对于模型初始化不敏感，因此我们统一将 λ 初始化为 0.1 进行实验，中间 λ 按照公式3-13进行动态调节。因此我们只对另外两个超参进行测试。

3.4.4.1　压缩率



(a) 目标压缩率和实际压缩率变化曲线　　(b) 不同的目标压缩率在 VGG-16 和 ResNet-56 中的性能变化曲线

图 3-4　VGG-16 和 ResNet-56 在 CIFAR-10 中目标压缩率与性能变化曲线

表 3-8 在 CIFAR-10 数据集中不同目标压缩率 r 对 ResNet-56 的压缩性能比较

Target Compression Rate	Real Compression Rate	FLOPs	FLOPs Drop Rate	Params	Params Drop Rate	Top-1 Acc.(%)
1.0	-	1.305G	-	23.521M	-	93.89
0.8	0.74	941.088M	28%	15.404M	34.51%	95.02
0.7	0.67	899.946M	31%	13.593M	42.21%	94.50
0.6	0.61	857.713M	34%	11.978M	49.08%	94.67
0.5	0.52	772.347M	40%	9.635M	59.04%	94.40
0.4	0.42	682.811M	48%	7.754M	67.03%	94.71
0.3	0.33	610.142M	53%	6.255M	73.41%	94.32
0.2	0.24	499.827M	62%	5.013M	78.69%	93.67

表 3-9 VGG-16 在 CIFAR-10 中 β 初始值和 Top-1 测试精度效果比较

β	FLOPs	FLOPs Drop Rate	Params	Params Drop Rate	Top-1 Acc.(%)
0.005	163.822M	47%	2.930M	80.11%	92.00
0.01	163.622M	47%	2.927M	80.13%	92.46
0.05	161.244M	48%	2.883M	80.43%	93.05
0.1	162.574M	48%	2.908M	80.26%	92.77
0.2	163.367M	47%	2.915M	80.21%	91.06

3.4.4.2 二值化调控

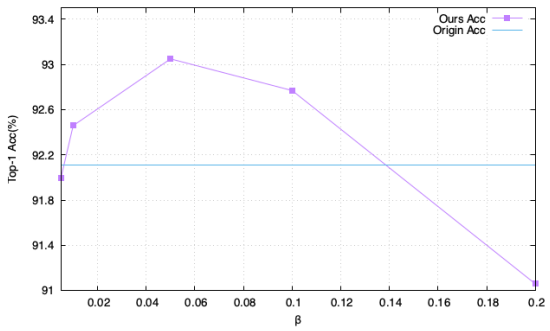


图 3-5 VGG-16 在 CIFAR-10 中 β 初始值和 Top-1 测试精度变化曲线

正如公式3-14所示，β 是一个重要的超参数。我们根据3.3.4分析可知，我们只需要注意 β 的初始化值。研究[55]证明，β 的初始化值与模型结构有关，因此我们针对 VGG-16 的模型结构设计了关于 β 值变化的实验比较，确定最优的 β 初始化值；而对于 ResNet-56 结构，我们参考 AutoPruner[55]中的设置，设置 β 的初始化为 1；对于 VGG-16 和 ResNet-56，我们让 β 随着训练过程逐渐变大，初期开始使数值缓慢增加，训练到一定程度时，令数值的增加速度变快。

在本章算法中，目标压缩率 r 表示通道的目标通道剪枝保留率，即保留下来的通道数占原来通道数的比率。我们将压缩率作为预定义的值，它是过滤器修剪的目标。我们在 CIFAR-10 数据集中针对不同目标压缩率对 VGG-16 和 ResNet-56 模型进行压缩测试。

在实验中，模型的最终压缩率往往可能略高于目标压缩率 r，效果如图3.4(a)所示；其中 VGG-16 模型的实际压缩率都比目标压缩率高，而 ResNet-56 的实际压缩率也基本都大于目标压缩率；原因是我们的修剪方法是自适应的，我们在损失函数中添加了一个正则化器，以鼓励模型注意模型的压缩率，这使得模型需要兼顾精度和通道的压缩率两个方面；在 ResNet-56 中，当 $r = 0.7$ 和 $r = 0.8$ 时，目标压缩率大于实际压缩率，主要原因是在实际剪枝过程中，在较高的目标压缩率下，其精度损失较少，因此损失函数让模型更关注于模型的通道压缩率，从而实际压缩率大于目标的压缩率。

对于模型不同压缩率，我们测试模型的精度和实际压缩率，其结果如表3-7和表3-8所示；而模型的精度变化和实际压缩率的变化趋势如图3.4(b)所示，可以看出不同压缩率下实际压缩精度的变化。表3-10和表3-8中可知，FLOPs Drop Rate 和 Params Drop Rate 随着目标压缩率的减少而增加，主要原因是目标压缩率下降，则模型的通道剪枝量增加，从而需要剪掉更多的参数，这使得模型的 FLOPs 和参数的删减量增加。结合表3-7、表3-8和图3.4(b)可知，VGG-16 和 ResNet-56 通过不同压缩率 r，计算 Top-1 测试精度的变化趋势，可以了解到目标压缩率 r 大约在 0.3~1 之间，其模型精度总体是优于原始模型精度的；这种现象的主要原因在于当进行适当的通道剪枝时，降低了模型的复杂度，使得模型过拟合程度下降，增强了模型的泛化能力，从而让模型测试精度提升；其中在 $r = 0.4$ 时，其 Top-1 的精度是最高的，为 92.77%，说明该模型最适合的目标通道压缩率 r 为 0.4 左右；当 $r = 0.2$ 时，模型的精度是最差的，为 91.38%，其主要原因是模型被过度剪枝，使得相比原模型存在精度损失。而观察图3.4(b)删减 FLOPs 的比例可以发现，其剪掉的比例是和模型的目标通道压缩率 r 大致呈反比，即模型的目标通道压缩率 r 越高，FLOPs 的下降率则越低；从图可以知道在目标通道压缩率 $r < 0.6$ 的时候，FLOPs 下降的趋势相对缓慢一些，主要原因是当模型进行一定程度压缩之后，模型的泛化性能得到了最优，此时如目标通道压缩率 r 持续下降，那么就更需要兼顾剪枝和精度两个方向的平衡，此时为了在模型追求较好

的精度，会自动保留部分根据剪枝规则本应该删除的通道，从而让 FLOPs 下降率会有所减缓。

我们对 VGG-16 进行测试，统一设定模型通道目标压缩率 $r = 0.4$，β 初始化值和模型的精度变化趋势如图3-5和表3-9所示，该组实验是在 CIFAR-10 数据集进行测试所得。从图中可知 β 的初始化值不能太大，如果 β 太大，其测试精度则会较低，如当 β 初始化为 0.2 时，其模型精度低于原始模型精度，Top-1 精度为 91.06%。主要原因是：在 β 初始化值过大的情况下，Sigmoid 映射的值提前完全二值化，这将使学习模型策略的卷积层无法充分了解模型参数影响的特征。在这种情况下，我们的方法将退化为随机通道选择，从而导致了精度降低。图3-5显示，当 β 非常小时，如 $\beta = 0.005$，模型的预测准确性也会下降，Top-1 测试精度为 92.00%。出现这种情况的原因是 β 太小，学习模型影响力分布的卷积层过度学习导致模型略有过拟合，从而导致模型预测准确性降低。实验结果表明，我们的方法需要选择较小的 β，但不能过于小，即在一定范围内，根据测试证明 VGG-16 的最佳 β 初始化大约在 0.05~0.1 之间；由表中可以知道，在 0.05~0.1 之间，模型的 FLOPs Drop Rate 和 Params Drop Rate 最大，这使得我们猜测在相同目标压缩率的情况下，模型的二值化调控值 β 影响了模型的实际精简程度，从而导致模型的性能发生变化。

3.5 本章小结

本章提出了一种端到端通道自动剪枝算法，它基于影响函数测量权重影响力的方式，以影响力大小排序，根据排序进行通道修剪。这种算法能够既兼顾任务精度，又兼顾模型剪枝测量，使模型能够在压缩率和精度直接取一个平衡。具体来说，通过一种稳健统计方法（影响函数）统计模型中单个权重对于模型计算带来的影响力，计算通道影响力，然后以通道为单位进行影响力排序，根据压缩率为参考进行通道修剪。由于通道中的权重是独立的，因此通道的影响力即为通道中所有权重影响力的和。这种方法能够较为近似的判断通道中参数对整个模型的影响程度，能够获取更准确的冗余数据。而且，由于模型的损失函数中增加了一项正则项来监督模型的策略，使得模型具有兼顾精度和压缩率的能力。实验结果表明，本章提出的方法不仅在降低卷积网络的计算消耗方面具备优势，

而且使得模型复杂度降低从而缓解了模型的过拟合情况，导致模型的精度略有提升。更重要的是，本章提出的方法是端到端的自适应方法，且生成的轻量化模型不依靠模型库或者特殊硬件支撑运行，使得这种算法的落地运用更加便捷有效。

第四章　基于影响力剪枝和低秩分解的 LSTM 压缩方法

4.1　LSTM 压缩的重要性

近几年，循环神经网络在文本识别、语音识别、及其翻译等任务上取得了优秀的成果，以 LSTM 为代表的神经网络为这些任务提供了很好的建模工具。但随着模型的层数增加，网络参数量也逐渐增长，网络预测能力也越来越强，算力需求也随之增长。因此通常网络只能部署在非终端平台。终端设备需要通过发送计算请求，云端接受请求之后进行模型计算，将结果发回终端设备，这整个过程造成了较大的时延，这会降低用户体验，无法满足用户的便捷需求。

云计算只能满足对时延要求较低的任务，但要求低时延的任务更需要终端设备能够承载模型的计算。因此嵌入式开发被愈来愈多的研究所关注。终端设备承载模型使得计算仅在本机即可完成，免去了信息发送和接收的时延，大大降低整个计算时间。由于嵌入式设备自身所带的计算力和存储量都较为有限，无法具备云服务器上的计算能力和存储，这就要求终端设备承载的模型只需要较小的算力需求和较少的参数量。例如，假设有两层的 LSTM 模型结构，其隐含层单元数是 200，那么这运用于语言模型中，高达 116 的测试困惑度；如果再把隐藏层为 1500，层数结构依然如旧，在同等的语言模型中，其测试困惑度会再次降低到 78。但是因为隐藏层单元数量增加，导致模型参数量巨大，需要消耗存储资源。例如，假设当隐藏层单元是 200 时，一层的 LSTM 模型有 320K 的参数量，那么两层 LSTM 的参数量即为 640K。当隐层单元增加为 1500 时，一层的 LSTM 模型则由 320K 参数量增加至 18M 参数量，两层则为 36M，这比 200 个隐含层单元的双层网络提升了约 50 倍的参数量。

总的来说，LSTM 网络的参数量下降共有 4 点优势：

1. 在网络传输中，参数量越小，其传输的速度更快，更有利于模型的更新迭代。例如，某品牌手机的某个功能是依靠网络模型搭建实现的，厂商在远程情况下逐渐更新和迭代该模型，提升产品性能，而最新版本的更新布丁包需要用户下载以实现更新，当模型的参数越少、模型越小，那么需要下载的数据量也就越少，让用户的更新更加便捷。
2. 如果模型的参数量下降，则前向传播的速度将会大大加快。模型的前向传播过程中，输入数据需要与大量的参数进行浮点数运算。当模型的参数量减小，其前向计算中的浮点数运算消耗也会相应地降低。另外，如果模型足够小，能够将参数直接放于 RAM 存储器中，则降低模型参数的读取时间，实现更大程度地缩短模型前向传播的时间。
3. 参数量越小越能够降低模型的功耗。当一个模型较大时，它只能搭建在云服务器上进行计算，终端设备无法承载这么大的模型。而当模型压缩到足够小时，更少的运算次数，使得功耗降低，从而能够搭建在小型设备中，使输出结果获取更加便捷
4. 较小的参数量能够大大的节约成本。模型参数减少，使得存储量降低，这免去了更多的存储成本。另外，较小的模型搭建在小型设备上，免去了信息传输的时间成本和硬件成本。

当前的研究表明，大模型如果参数稀疏，那么其性能会比小却密集的模型性能更有优势。换句话说，对大模型参数进行压缩处理比对直接训练一个小模型的预测效果更佳。因此，LSTM 模型的压缩十分必要。

4.2 模型分析

LSTM 模型与传统的卷积模型不同，其结构的基本单位是记忆单元结构，其中参数为 8 个矩阵组成，而卷积网络由网络层结构组成，层内有多个卷积核结构，卷积核为参数矩阵构成，其数量比 LSTM 模型的参数矩阵数量更多的多。且训练模式不同，LSTM 在学习过程中，模型状态输出是同一个记忆单元结构的输入，而卷积网络是前向计算过程，上一层模型的输出作为下一层模型的输入。因此由于模型结构的差异，卷积神经网络的压缩方法无法直接运用于 LSTM 网络中，为此需要先根据 LSTM 结构和参数矩阵的特点再确定具体的模型压缩方案。

4.2.1　LSTM 结构

传统的 LSTM 单元的前向计算如图4-1所示，其中输入计算运用到 8 个参数矩阵，其矩阵大小与状态 h_{t-1} 和输入 x_t 的维度有关，由于输入数据为连续性数据，由第二章可知，假设 x_t 的输入维度为 $[1, M]$，h_{t-1} 的状态信息维度为 $[1, N]$；则一个 LSTM 记忆单元的参数量为 $4\times((M+N)\times M+M)$；实际上，在 LSTM 单元中，参数矩阵大小由 M 和 N 共同决定，这种决定方式，让模型参数很可能存在冗余，因此可以考虑对权重矩阵进行稀疏化处理，再进行降维，从而最大限度减少冗余参数。

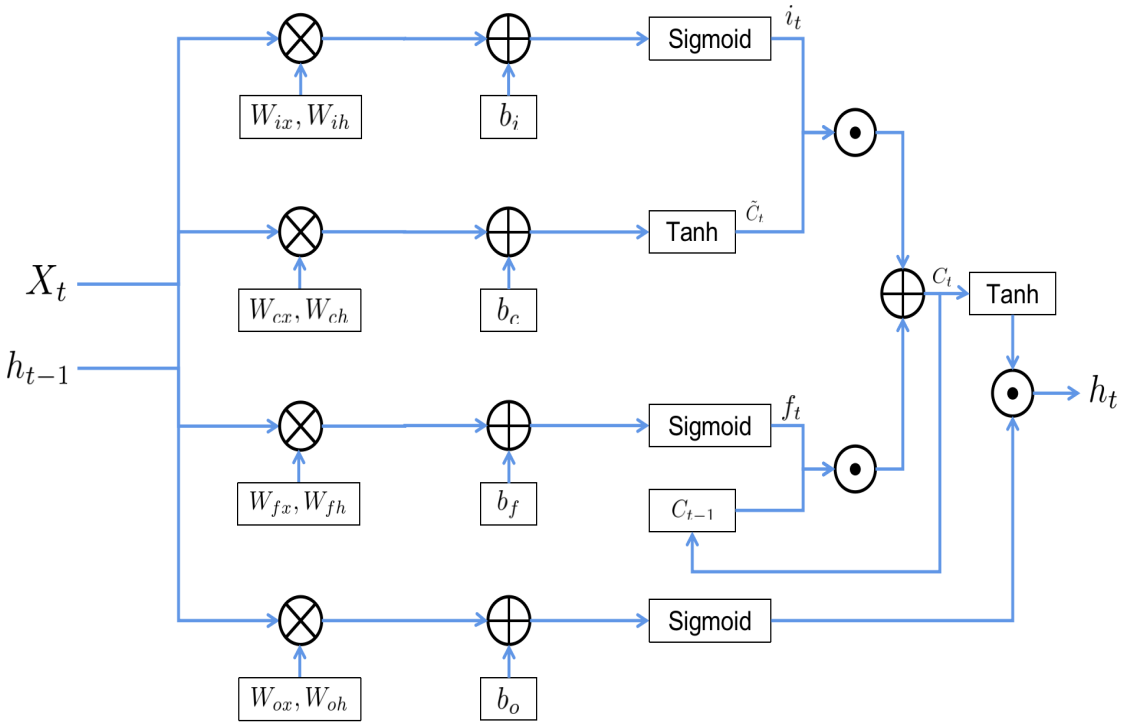


图 4-1　LSTM 记忆单元前向计算过程

首先，根据公式3-10，可以知道模型参数的影响力测量源于影响函数计算所得，具体根据影响函数定义将公式2-22转化为如公式4-1所示，对模型 G 中的权重 W 加入极小的扰动值 t，对 t 进行求导，当 t 接近 0 时，其导数结果即为权重 W 对模型 G 的影响力。

$$\mathrm{IF}(x;G;W) := \lim_{t\to 0^+} \frac{G\left(t\Delta_x + (1-t)W\right) - G(W)}{t} \tag{4-1}$$

因此，根据公式4-1，可以在权重矩阵 W_{ix}、W_{fx}、W_{cx} 和 W_{ox} 中，加入与

对应矩阵同尺寸的极小扰动矩阵，根据影响函数测算影响力，根据影响力排序进行模型的非结构化剪枝，这种剪枝方式使得矩阵转化为稀疏矩阵，如果没有硬件支持，那么很难实现模型的剪枝压缩效果，因此需要结合其他算法进一步处理模型。

4.2.2 低秩分解

由于模型变得稀疏，如果没有特殊的硬件帮助，那么权重的剪枝并不能在普通硬件上实现称为真正意义上的剪枝效果。因此还需要进一步处理。此时，经过影响力排序剪枝之后，模型权重转化为稀疏矩阵，考虑将稀疏矩阵进行低秩分解，以此来去除冗余的同时，解决非结构化剪枝带来的弊端，最大限度保证模型精度不下降。

在稀疏参数矩阵中，其权重向量大部分分布于在一些低秩的子空间中，通常可以用少数几个基向量来重构模型参数矩阵。因此，通过对稀疏矩阵进行矩阵分解，根据 SVD 分解[65]可知，如公式4-2所示，矩阵 X 可以分解为 U、Σ、V 三个矩阵。

$$X = U\Sigma V^T \tag{4-2}$$

其中其中 U 是一个 $m \times m$ 的矩阵，Σ 是一个 $m \times n$ 的对角矩阵，其主对角线上的元素不为 0，其他元素为 0，主对角线上元素为矩阵的奇异值。V 是一个 $n \times n$ 的矩阵，且 U 和 V 都是酉矩阵，满足公式4-3，其计算过程如图4-2所示。

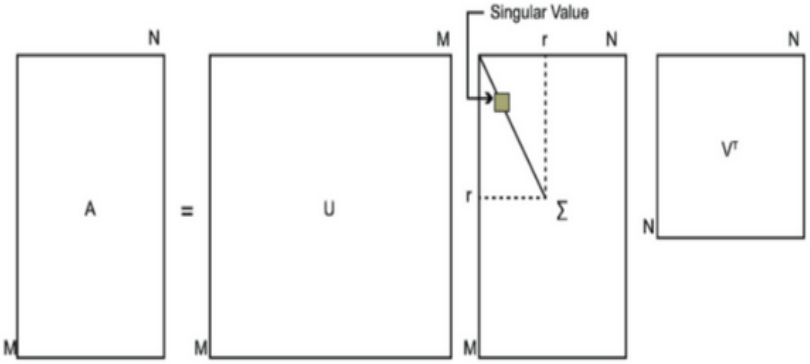


图 4-2 SVD 计算过程

$$U^T U = I, V^T V = I \tag{4-3}$$

如图4-2中 A 矩阵如果由权重矩阵分解得来的，通常不是满秩矩阵，即存在冗余数据，可以将矩阵 A 中非 0 的 r 个奇异值保留，将矩阵转换为 D，那么矩阵可以转化为如公式4-4所示。

$$X_{m\times m} = U_{m\times m}\Sigma_{m\times m}V^{T}_{m\times m} = U_{m\times r}D_{r\times r}V_{r\times m} \tag{4-4}$$

另外，如果存在极小的奇异值，即接近 0 的奇异值可以当作噪声省略，剩下 k 个奇异值。因此矩阵分解进一步转化为如公式4-5所示。将 U 和 D 矩阵合并，进一步转化为如公式4-6所示。

$$X_{m\times m} \approx U_{m\times k}D_{k\times k}V_{k\times m} \tag{4-5}$$

$$X_{m\times m} \approx \left(U_{m\times k}D_{k\times k}\right)V_{k\times m} = A_{m\times k}V_{k\times m} = A_{m\times k}B_{k\times m} \tag{4-6}$$

如图4-3所示，将矩阵 $X_{m\times n}$ 分解为 $A_{m\times r}$ 和 $B_{r\times m}$ 两个矩阵。如公式4-7所示，其

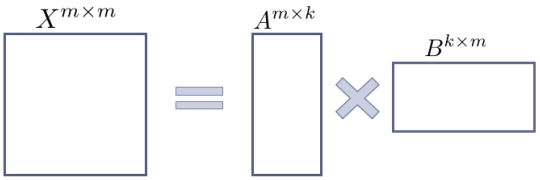


图 4-3　矩阵分解过程

中 $m\times m$ 表示权重矩阵原理的维数，r 表示压缩之后秩的大小，则模型参数量从 $m\times m$ 转变为 $2\times m\times r$ ，当 r 足够小时，模型参数量得到明显的下降，例如，当 $m=10$，$r=2$ 时，分解后参数量压缩原来的 40%。

$$X_{m\times m} \approx A_{m\times k}B_{k\times m} \tag{4-7}$$

这项工作类似于协同过滤，其本质上根据稀疏矩阵已有的信息，来预估矩阵中缺失的信息。这个原理可以运用于非结构化剪枝当中，当剪枝之后，模型权重矩阵稀疏，可以利用协同过滤方法预估模型剪枝掉的参数，这样既降低了模型的参数量，又实现了尽可能保全模型性能的目的。

4.3 基于影响力剪枝和低秩分解的模型压缩

4.3.1 模型原理

在本章算法中，测算模型权重影响力的方式为在记忆单元中，增加模型的 Mask 参数，如图4-4所示，在 W 权重之后加入同尺寸的 Mask 扰动矩阵，Mask 扰动矩阵于模型参数 W 进行元素乘，Mask 权重初始化为 1，这样在前向传播中不会影响模型的计算。

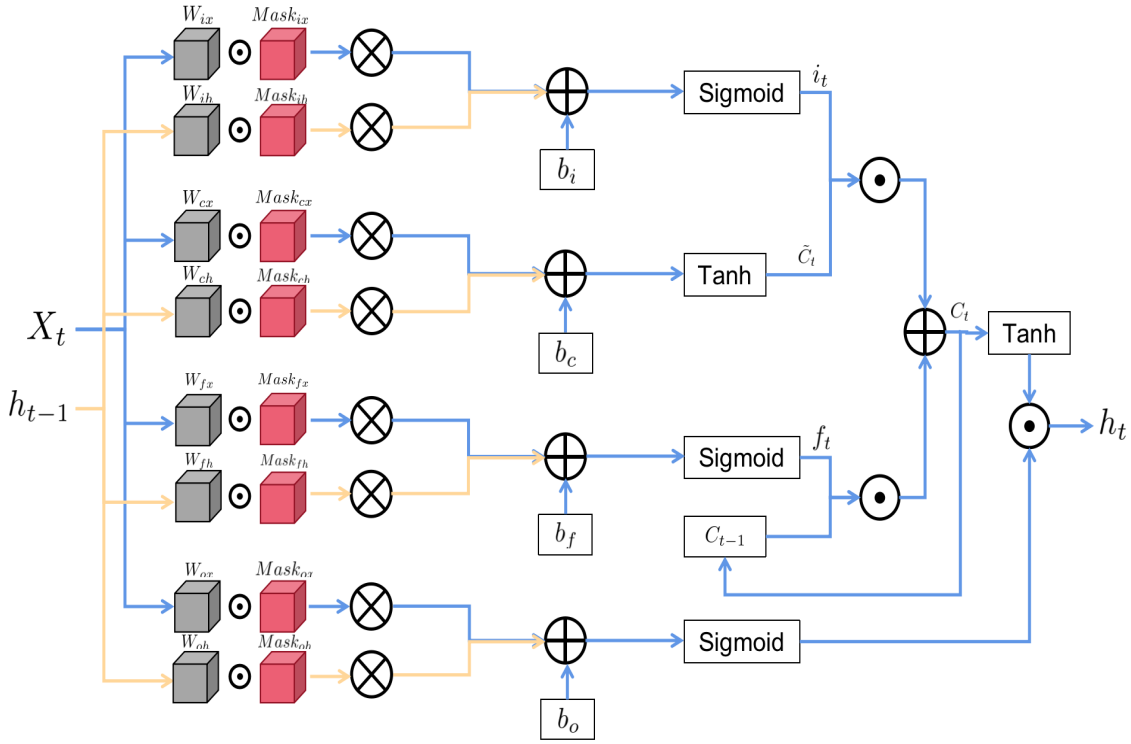


图 4-4 LSTM 记忆单元插入 Mask 权重矩阵的前向传播过程

根据图4-4所示，其中 ⊗ 表示矩阵乘法；⊙ 表示矩阵元素乘；以输入门为例：设 $W_{ix}^{(j,k)}$ 表示 W_{ix} 矩阵中第 j 行第 k 列的单个权值；设 $Mask_{ix}$ 表示输入权重 W_{ix} 增加的对应的同尺寸 Mask 权重矩阵，$Mask_{ix}^{(j,k)}$ 设表示 $Mask_{ix}$ 矩阵中第 j 行第 k 列的单个权值扰动，其中 t 表示当前时刻，X_t 表示当前时刻的输入，h_{t-1} 表示前一时刻的状态输出；在前向传播中，W_{ix} 与 $Mask_{ix}$ 进行元素乘，即对应的第 j 行第 k 列元素进行 $W_{ix}^{(j,k)} \times Mask_{ix}^{(j,k)}$ 计算，同理，W_{ih} 与 $Mask_{ih}$ 也进行元素乘计算，再分别与 X_t 和 h_{t-1} 相乘，将这两个结果与偏差 b_i 加和，通过 Sigmoid 函数激活后即得到输入门的输出 i_t，其他门也同理。在反向传播中，根据公式4-1，对于权重矩阵 W_{ix} 中的单个权重 $W_{ix}^{(j,k)}$，对 $Mask_{ix}$ 以反向求梯度

的方式获取 $Mask_{ix}^{(j,k)}$，即可得到 W_{ix} 中单个权重的的影响力，即为取 $Mask_{ix}^{(j,k)}$ 的导数。通过 $Mask_{ix}$ 的导数作为影响力，通过压缩率确定阈值，利用阈值进行非结构化剪枝，得到稀疏矩阵 $\tilde{W}_{ix}^{(j,k)}$，通过稀疏矩阵进行矩阵分解，分解为矩阵 A_{ix} 和 B_{ix}，整个过程如图4-5所示。

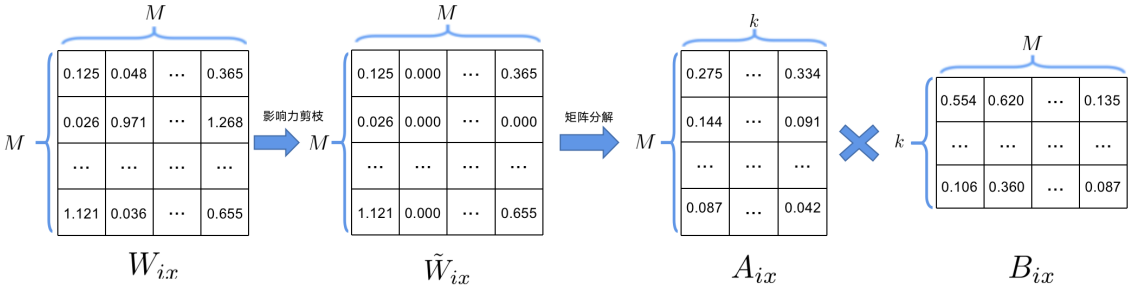


图 4-5　权重矩阵转稀疏的 SVD 分解过程

4.3.2　训练阶段

在训练阶段，由于 LSTM 结构是 RNN 的一种，因此考虑对 LSTM 记忆单元中的权重矩阵进行处理。首先如图4-6所示，对原始 LSTM 记忆单元插入 Mask 权重矩阵，然后对 Mask 权重矩阵和 W 权重矩阵进行模型参数的初始化。从含 Mask 参数矩阵的 LSTM 记忆单元中提取 Mask 梯度矩阵，作为对应权重 W 的影响力矩阵。由于影响力数值可能差距较大，且影响力计算的数值极小，为避免出现梯度消失的问题，将其转化为比例值 $\mathbf{Inf}_{W,r}$，即单个权重对应的影响力占所有影响力总值的百分比。考虑单个数据测算权重 W 的影响力存在差异，造成剪枝策略结果带有偶然性，因此需要 8 卷积核 $\mathbf{K}_i(0 <= i <= 7)$ 分别对权重矩阵的影响力进行降噪处理，来缓解偶然性问题。其中卷积核学习的目标剪枝策略矩阵 $B_{W,S}$ 来源于 $\mathbf{Inf}_W$ 处理得到的。具体来说，将 $\mathbf{Inf}_W$ 进行阈值比较、强制转换得到二进制剪枝策略矩阵；我们通过得到的 $\mathbf{Inf}_W$ 矩阵的值根据大小排序，由压缩率 r 决定保留的权重，选择保留权重对应的最小 $\mathbf{Inf}_W$ 数值作为阈值。本章所提到的二进制剪枝策略矩阵都指的是由 0 和 1 组成的矩阵，该矩阵与记忆单元中的权重尺寸大小相同，其中矩阵内单个值与权重矩阵中的值是一一对应的关系，当矩阵中有值为 1，则表示这个 1 值对应的权重矩阵需要保留，而为 0 时则表示不保留对应的权重。经过卷积核 $\mathbf{K}_i(0 <= i <= 7)$ 处理之后得到中间矩阵，我们通过 Sigmoid 函数进行二值化转化为矩阵，其值为 0-1 之间的数值，另

一方面，从原始 LSTM 记忆单元中，获取预训练模型，得到不含 Mask 矩阵的记忆单元，从记忆单元中提取预训练好的矩阵 W，将矩阵 W 与权重进行元素乘，获得新的权重矩阵，然后将应用到预训练模型中，根据训练损失函数进行反馈更新模型权重。

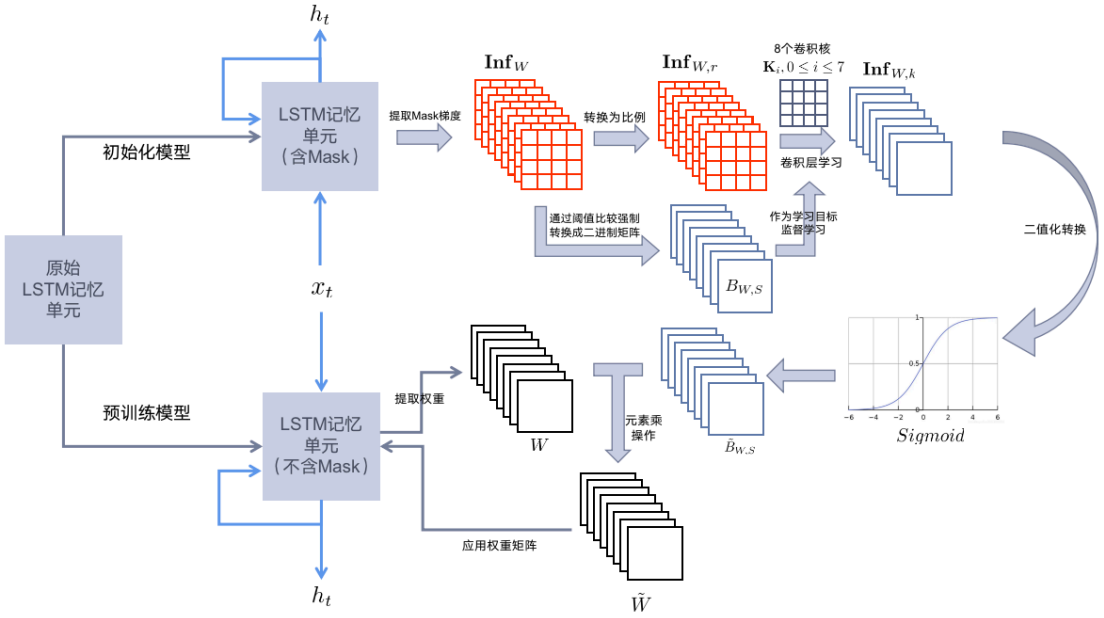


图 4-6　LSTM 压缩的训练过程

经过 Sigmoid 函数映射之后的矩阵 $\tilde{\boldsymbol{B}}_{W,S}$ 相当于对权重 W 添加了 Soft Mask，使得权重变得稀疏，再利用稀疏的权重矩阵 $\tilde{W}$ 对模型进行训练，根据特殊的损失函数，进行参数调整。以分类任务为例，损失函数如公式4-8所示。

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \lambda \left\| \tilde{\boldsymbol{B}}_{W,S} - \boldsymbol{B}_{W,S} \right\|_2^2 \tag{4-8}$$

需要注意的是，在实际训练中得到的中间矩阵 $\mathbf{Inf}_{W,k}$，经过 Sigmoid 转换中，需要通过超参 β 进行放缩。初始化时，将参数 β 设置较小，使得模型能够让 8 个卷积层充分学习影响力分布，学习充分之后，将 β 数值逐渐调大时 Sigmoid 映射结果逐渐趋近于 0 或者 1。训练结束时，最终让 $\tilde{\boldsymbol{B}}_{W,S}$ 矩阵转化为近似仅为 0 和 1 的二进制矩阵。此时 $\tilde{\boldsymbol{B}}_{W,S}$ 内的值无限接近 0 和 1 两极方向，通过四舍五入的方式转换为二进制剪枝策略矩阵，可以得到最终的非结构化剪枝策略 $\tilde{\boldsymbol{B}}_{W,S,end}$，详细细节在本章4.3.3介绍。

通过公式4-9，将 W 和 $\tilde{\boldsymbol{B}}_{W,S_{end}}$ 相乘转化为 $\tilde{W}$ 权重稀疏矩阵，根据4-3的过

程进行低秩分解工作，设置参数 k，使得 $\tilde{W}$ 转化为 A、B 两个矩阵。将拆分矩阵替换原本的矩阵 W 运用于模型当中。如图4-7、图4-8所示。

$$\tilde{W} = W \odot \tilde{B}_{W,S} \tag{4-9}$$

其中，设定参数矩阵 W_{ix}、W_{cx}、W_{fx} 和 W_{ox} 的参数矩阵维度为 $[m,m]$，设定参

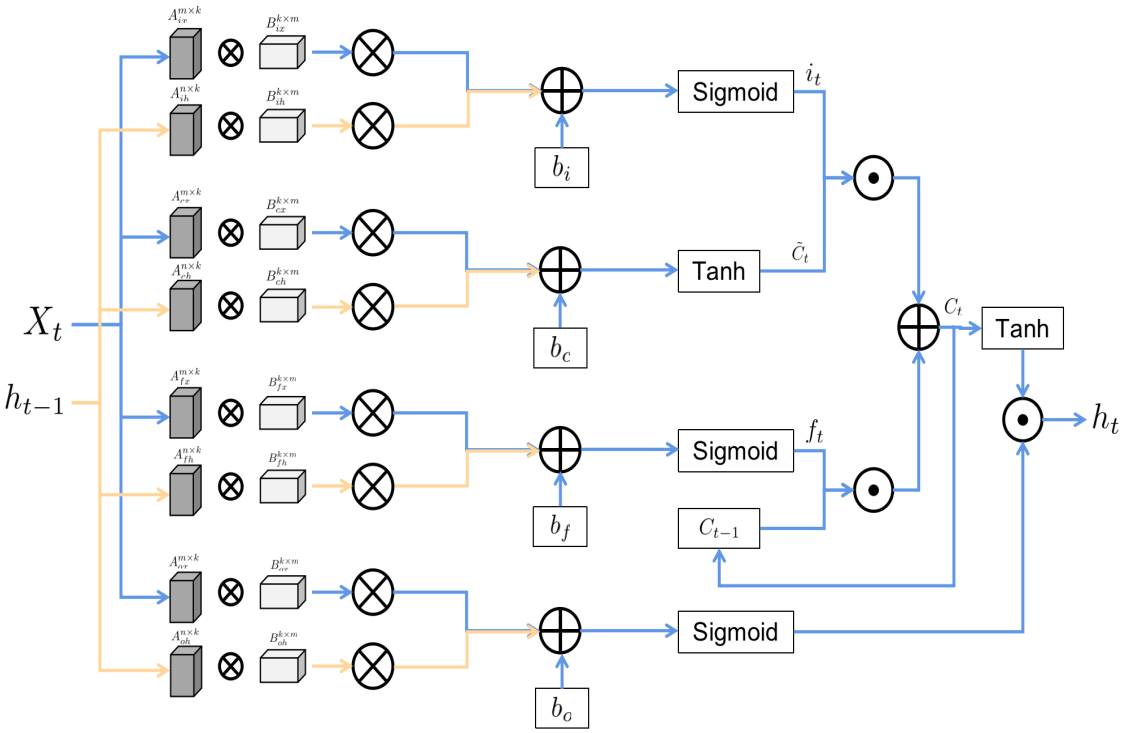


图 4-7　LSTM 记忆单元结构中的权重矩阵分解后的前向传播过程

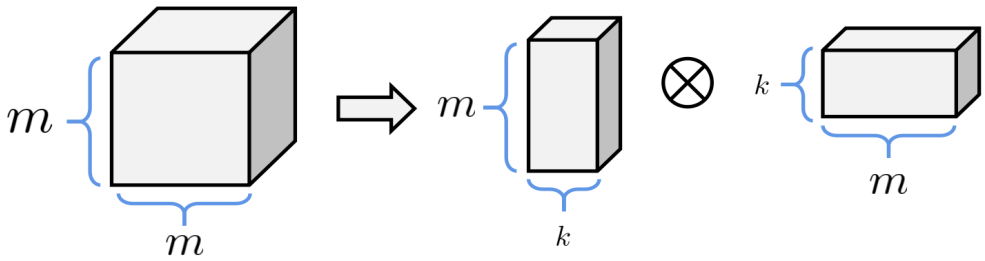


图 4-8　LSTM 记忆单元结构中的权重矩阵分解过程

数矩阵 W_{ix}、W_{ch}、W_{fh} 和 W_{oh} 的参数矩阵维度为 $[n,m]$；假设 SVD 分解矩阵的近似秩为 k，那么以输入门为例，输入门参数矩阵为 W_{ix} 和 W_{ih}，可以将 W_{ix} 矩阵拆分为 A_{ix} 和 B_{ix}，将 W_{ix} 矩阵拆分为 A_{ih} 和 B_{ih}；则 A_{ix} 的维度是 $[m,k]$，A_{ih}

的维度是 $[n,k]$，B_{ix} 和 B_{ih} 的维度是 $[k,m]$；遗忘门和输出门同理进行相同的稀疏矩阵分解过程。

4.3.3 超参设置

在本章算法中，共需要 5 个超参，分别是剪枝压缩率 r、损失函数正则化项权重 λ、二值化调控值（包括输入数据缩放值 β_x 和状态输入缩放值 β_h）以及 SVD 分解的奇异值域值 $thresh$。我们对这些参数分别进行了较为合理的设计。

1. **剪枝压缩率 r：** 本章模型需要通过剪枝和 SVD 分解两个过程，其中剪枝过程为非结构化剪枝。我们需要根据压缩率 r 来确定模型剪枝的参数剪枝率，其中剪枝的基本单位是权重矩阵的列，换句话说，假设权重矩阵列数为 n，那么本章算法保留的参数矩阵列数为 $n \times r$。r 的设定是根据实际应用环境所决定的，在后文中，我们将详细介绍对于压缩率 r 的相关实验。
2. **正则化项权重 λ:** 对于正则化权重 λ 在训练中是动态调节的，且对初始化不敏感，因此我们只需要注重对 λ 的自动调节机制的设计，其设计如公式4-10。其中 E_W 表示当前记忆单元中与权重矩阵 W 同尺寸且初始化值为 1 的矩阵，我们用 $|E_W|$ 表示原始权重矩阵的参数量，即矩阵 E_W 的数值只和；设定 $B_{W,S}$ 为图4-6中的 $\mathbf{Inf}_W$ 经过阈值比较强制转换之后的二进制目标剪枝策略矩阵，而 $\tilde{B}_{W,S}$ 为图4-6中的 $\mathbf{Inf}_{W,k}$ 通过 Sigmoid 降噪之后的剪枝策略矩阵，用来当作实际应用的剪枝策略矩阵；$\frac{\tilde{B}_{W,S}}{|E_W|}$ 表示为参数实际保留率，$\frac{B_{W,S}}{E_W}$ 表示当前层的通道目标参数保留率。如果实际参数保留率超过目标参数保留率，那么我们通过公式4-10将会降低模型修剪策略的关注度，而更关注模型的分类（或回归）任务准确性。如果实际参数保留率低于目标参数保留率，则应适当加大 λ 的值，以使模型更加注重剪枝策略的调整，实际应用中我们将 λ 直接设置为 0。

$$\lambda = \begin{cases} 5.0 \times |\frac{B_{W,S}}{|E_W|} + \frac{\tilde{B}_{W,S}}{|E_W|} - 1|, & 1 - \frac{\tilde{B}_{W,S}}{|E_W|} \geq \frac{B_{W,S}}{E_W} \\ 0, & \text{otherwise.} \end{cases} \tag{4-10}$$

3. **二值化调控值 β_x 和 β_h:** β_x 和 β_h 主要用来对 Sigmoid 函数的输入数据进行离散程度的调控，β_x 和 β_h 分别调控的是单元输入数据和单元输入状态。如

本文3.3.4所说，以较小的值初始化，使二值化之前的卷积核 $\mathbf{K}_i(0 <= i <= 7)$ 能够得到充分的学习，然后再逐渐将 β_x 和 β_h 值增加，让参数进行二值化映射时，能够逐渐往 0 和 1 方向收敛。Sigmoid 函数如公式4-11所示。其中 $\mathbf{Inf}_{W_x,k}$ 和 $\mathbf{Inf}_{W_h,k}$ 分别表示进过卷积核学习之后的中间矩阵，其合并起来就是图4-6的 $\mathbf{Inf}_{W,k}$ 矩阵，$\tilde{B}_{W_x,S}$ 和 $\tilde{B}_{W_h,S}$ 分别表示进行二值化处理之后的近似二值化的矩阵，它们合并起来就是图4-6的 $\tilde{B}_{W,S}$ 矩阵。在 LSTM 压缩中保留二值化过程的主要原因在于，一方面模型可以动态调节卷积核 $\mathbf{K}_i(0 <= i <= 7)$ 的学习情况，使模型的训练能够灵活调控，另一方面模型训练中可以实现动态修剪和连接，正式修剪可以在模型微调期间完成，这能大大缩短微调的时间。

$$\begin{aligned} \tilde{B}_{W_x,S} &= \text{Sigmoid}\left(\beta_x \mathbf{Inf}_{W_x,k}\right), \\ \tilde{B}_{W_h,S} &= \text{Sigmoid}\left(\beta_h \mathbf{Inf}_{W_h,k}\right). \end{aligned} \tag{4-11}$$

4. **低秩分解奇异值阈值 *thresh*：** SVD 分解的近似秩 $rank_i$ 和 $rank_h$ 由 W 权重矩阵分解之后的奇异值大小所决定；$rank_i$ 表示的是 LSTM 记忆单元中与输入数据有关的权重矩阵的近似秩；$rank_h$ 表示的是 LSTM 记忆单元中与输入状态有关的权重矩阵的近似秩其中近似秩。在实际应用中通常考虑保留奇异值较大的数即可，这需要在模型训练的实际情况和本章前面部分的非结构化剪枝工作所决定，本算法的 $rank_i$ 和 $rank_h$ 通过设置阈值门限 *thresh* 选择最合适的近似秩进行地址分解压缩。通常 *thresh* 保持在一个较小的范围，使得模型的精度损失相对较小；$rank_i$ 和 $rank_h$ 的确定可以根据4-12到4-14这几个公式表示。基于低秩分解压缩的目的，我们需要保证参数量不比原先的参数量更多，因此需要设定一个最大的秩，保证参数量在不增加的前提下进行近似的低秩分解压缩。在实际应用中，设定 m 表示 LSTM 记忆单元的数据输入宽度，n 表示 LSTM 记忆单元的状态输入宽度；对于输入的参数矩阵的最大秩 k_i 和当前输入状态的参数矩阵的最大秩 k_h，

我们设定的值如公式4-12所示；

$$\begin{aligned} k_i &= \frac{mn}{m+n} \\ k_h &= \frac{nn}{n+n} = \frac{n}{2} \end{aligned} \tag{4-12}$$

而在公式4-13中，$rank_{xi}$ 表示输入门的数据输入参数权重经过低秩分解之后得到的近似秩，$rank_{hi}$ 表示输入门的状态输入参数权重经过低秩分解之后得到的近似秩；$rank_{xf}$ 表示遗忘门的数据输入参数权重经过低秩分解之后得到的近似秩，$rank_{hf}$ 表示遗忘门的状态输入参数权重经过低秩分解之后得到的近似秩；$rank_{xc}$ 表示当前状态更新过程的数据输入参数权重经过低秩分解之后得到的近似秩，$rank_{hc}$ 表示当前状态更新过程的状态输入参数权重经过低秩分解之后得到的近似秩；$rank_{xo}$ 表示输出门的数据输入参数权重经过低秩分解之后得到的近似秩，$rank_{ho}$ 表示输出门的状态输入参数权重经过低秩分解之后得到的近似秩。D 表示权重矩阵的奇异值向量，而 D_y 表示权重矩阵的奇异值向量的分量；与输入数据 x 有关的参数矩阵的奇异值向量长度为 $\min(m,n)$，与状态输入有关的参数矩阵的奇异值的向量长度为 n。

$$\begin{aligned} rank_{xa} &= \sum_{y}^{\min(m,n)} \text{sign}(D_y \geq thresh), a \in \{i,f,c,o\} \\ rank_{ha} &= \sum_{y}^{m} \text{sign}(D_y \geq thresh), a \in \{x,f,c,o\} \end{aligned} \tag{4-13}$$

$$\begin{aligned} rank_i &= \min(rank_{ix}, rank_{fx}, rank_{cx}, rank_{ox}, k_i) \\ rank_h &= \min(rank_{ih}, rank_{fh}, rank_{ch}, rank_{oh}, k_h) \end{aligned} \tag{4-14}$$

4.4 实验效果与分析

在实际实验中，相同的修剪方法及 RNN 模型结构，对于不同语料库的效果影响可能不同。因此，我们评估了在两种语料库中，针对语言模型测试我们方法的效果。

4.4.1　实验配置

4.4.1.1　数据集

我们分别通过 Penn TreeBank[1]和 Enwik8[2]两个语料库对本章算法进行实验。Penn TreeBank 是自然语言处理（NLP）领域常用的语料库之一，其语料来源是 1989 年的华尔街日报内容，包含了 1M 的词汇量、2499 篇文章，标注的内容包括了词性标注和句法分析标注。本章算法在运用了 Penn TreeBank 的句法分析标注部分，对语料进行语言模型学习。Enwik8 是从 Wikipedia 收集下来的字节级数据库，主要由 Wikipedia XML 转换存储下来的前一亿个字节组合而成。我们通过在 Penn TreeBank 、Enwik8 数据集中进行语言模型的训练与测试，来比较不同模型的压缩性能。

4.4.1.2　模型结构及参数设置

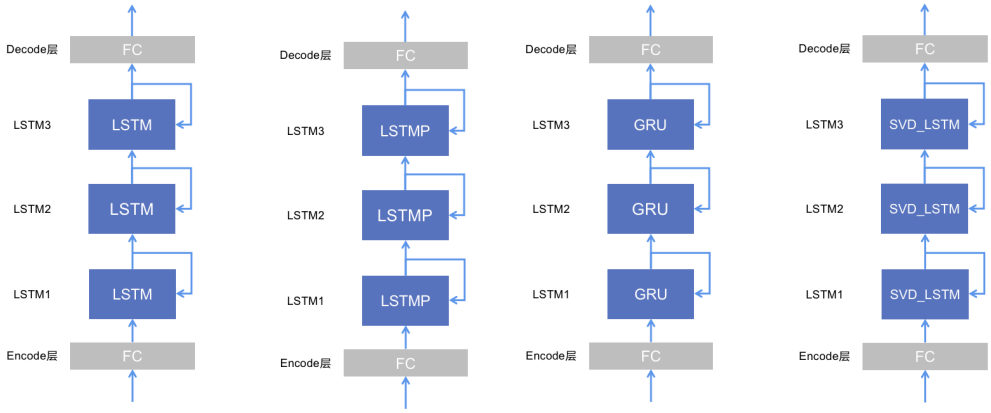
显然，我们很难与非结构化剪枝的工作进行直接比较，因为更多 LSTM 非结构化剪枝压缩工作需要与硬件相结合[66-69]，共同促进模型的加速压缩任务，而本章算法在不需要硬件支持的情况下达到模型压缩。因此，我们选择不依靠硬件来压缩 LSTM 单元的已发表方法进行实验，将其结果和本章压缩算法的实验结果进行比较。我们回顾了一些经典方法，如 LSTMP[70]其原理类似于在输出向量中加一个全连接层映射，使得输出数据压缩降维，从而达到降低 LSTM 记忆单元中相关的权重矩阵数量的目的：又如 GRU[71]将 LSTM 结构进行精简，降低了模型的参数量。由于原文的算法运行环境与本章算法的运行环境和实现的任务都不同，将这些方法与其他方法直接进行比较是不公平的。因此，我们将使用已发表作品的原始方法进行相同环境、相同数据集下的实验计算，并将其结果进行比较。

对语料库进行语言模型学习，我们设置了实验模型为 3 层的循环网络模型。类比的循环网络结构为 LSTMP、GRU 和原始 LSTM 结构。LSTMP 最是基于模型结构化参数压缩方法中运用最广泛的方法之一；GRU 的是 LSTM 结构的一种变体，其参数量更少，结构也更加简单。在实验中的语言模型训练任务中，模型

[1]Penn TreeBank 数据集来源：http://www.fit.vutbr.cz/ imikolov/rnnlm/simple-examples.tgz

[2]Enwik8 数据集来源：http://mattmahoney.net/dc/enwik8.zip

结构如图4-9所示，其中4.9(a)表示的是原始的三层预训练的语言模型，图4.9(b)表示以 LSTMP 单元为基础的三层语言模型，图4.9(c)表示以 GRU 单元为基础的三层语言模型，图4.9(d)表示通过影响力剪枝和低秩分解处理的 LSTM 结构单元组成的三层语言模型。图4-8中除了 LSTM 单元部分的结构不同，其他结构、输入数据和运算环境皆相同。



(a) 原始的三层 LSTM 结构　(b) 三层的 LSTMP 结构　(c) 三层的 GRU 结构　(d) 三层的 SVD_LSTM 结构

图 4-9 实验的模型结构图

我们的代码实现基于 PyTorch 和 Torchvision 代码库。此外，为了公平起见，每个对比模型都是用 torch.jit 来重写每个模型的 LSTM 的记忆单元，保证运算工具和环境的统一。为了提升 3 层 LSTM 模型在数据集中的表现性能，我们在所有的实验模型和对比模型的 Encode 层中都加入了 Dropout 操作。同时，我们在训练过程中设计了多个动态调整的超参，如4.3.3所介绍，我们对这些超参进行了消融实验。

在这些对比模型中，我们均设置每个记忆单元中的隐层状态数量为 1150 个，输入的 Embedding 维度为 400；在实验中，我们对 Encode 层的输出设置了 Dropout 操作，其值为 0.1；对记忆单元中的输入状态权重矩阵设置 Dropout，值为 0.5；对初始学习率 lr 设置为 30；对于 LSTMP 模型结构，设置 LSTMP 的映射层（Projection Layer）为输入的 Embedding 维度的 0.8 倍；对于所有数据集，本章算法设置模型的压缩率为 $r = 0.8$，设置二值化超参为 $\beta_i = 0.1$ 和 $\beta_h = 0.1$。在本章算法的作用下，对原始 LSTM 进行剪枝和 SVD 分解压缩过程，由于在实验中，剪枝过程是非结构化剪枝，在没有硬件辅助的情况下，非结构化剪枝相当于加入一个掩码层对个别参数置 0，实际模型的参数量是不变的，因此不计算原

始 LSTM 模型剪枝之后的 Params 指标和 Params Drop Rate 指标，但我们展示了非结构化剪枝之后的模型任务性能。

4.4.1.3　任务指标

对于语言模型学习任务性能好坏，我们使用 Test Loss、Test Perplexity[72]（PPL）和 Test Bits-per-character[73]（BPC）来衡量。其中 Test Loss 指的是测试集数据的计算损失。PPL 是自然语言处理中衡量语言模型是好是坏的一种指标，主要依据每个词汇来估计判断一句话能够出现的概率，并对这个句子的长度作标准化处理（Normalize），而 BPC 则是计算以一个字符长度为单位的混淆度。在实际应用中，PPL 和 BPC 的值越小越好

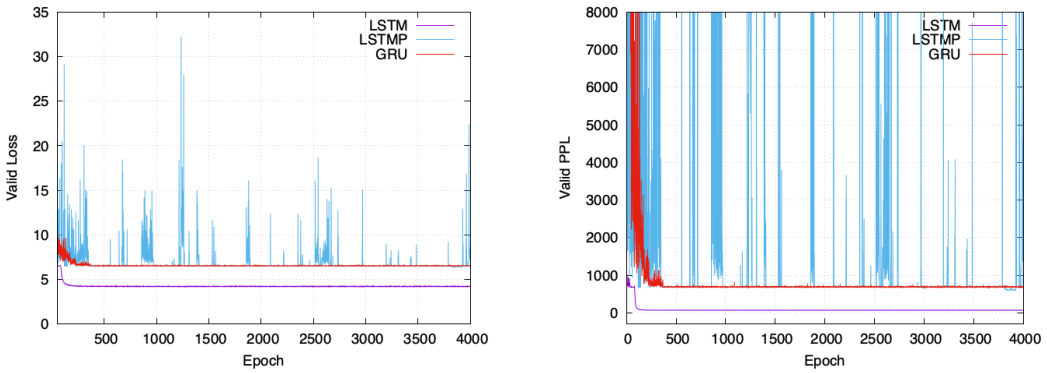
对于模型压缩性能的比较，本章基于参数量（Params）指标进行比较，因此主要关注于不同压缩算法对同一模型带来的参数量的减少量（Params Drop）占原本模型的百分比（Params Drop Rate）。在同样的语言模型学习模型中，分类性能相同或更优时，模型的参数量下降越多，则压缩算法的效果越好。

4.4.2　在 Penn TreeBank 中的实验效果

我们将 LSTM、LSTMP 和 GRU 模型都进行了 4000 个 Epoch 的训练，其训练过程中的验证集损失变化如图4-10所示。由于本章算法需要取预训练好的原始 LSTM 模型再进行数次的 Epoch 训练，为了公平起见，我们应需让对比模型也进行同样多的 Epoch 训练，但是，在实际的 4000 次训练过程中，模型训练已经逐渐稳定，额外的多次训练无法提升模型性能。我们对原始 LSTM、LSTMP 和 GRU 这三个模型进行了训练过程中的性能变化分析。由训练曲线图4-10可知，在 1000 个 Epoch 之后模型验证集 Loss 基本平稳，我们选取验证集 Loss 最小的模型进行比较，同时取验证集 Loss 最小的原始 LSTM 模型进行非结构化剪枝和 SVD 参数压缩。从验证集的 PPL 和 Loss 可以看出 LSTMP 模型在实验中是不稳定的，而在 1000Epoch 之后，GRU 和原始的 LSTM 模型基本处于基本稳定状态。总体来看，如果原始的 LSTM、LSTMP 和 GRU 三个模型继续进行数次的 Epoch 训练，其性能也没有获得更多的提升，因此我们可以将训练好的模型直接与本章算法的压缩模型进行比较。

表 4-1 基于 Penn TreeBank 语料库的模型压缩实验效果

模型	Test Loss	PPL	BPC	Params	Params Drop Rate(%)
Origin LSTM	4.15	63.33	5.985	35.442M	-
LSTMP	6.33	563.44	9.138	32.060M	9.54
GRU	6.47	643.44	9.330	27.576M	22.19
Ours(Pruning)	**4.14**	**62.85**	**5.974**	-	-
Ours(Pruning+SVD)	**4.29**	**73.21**	**6.194**	**21.368M**	**39.71**



(a) 在 Penn TreeBank 进行 4000 个 Epoch 训练的验证集 Loss 变化曲线

(b) 在 Penn TreeBank 进行 4000 个 Epoch 训练的验证集 PPL 变化曲线

图 4-10 模型在 Penn TreeBank 进行 4000 个 Epoch 训练的验证集性能变化曲线

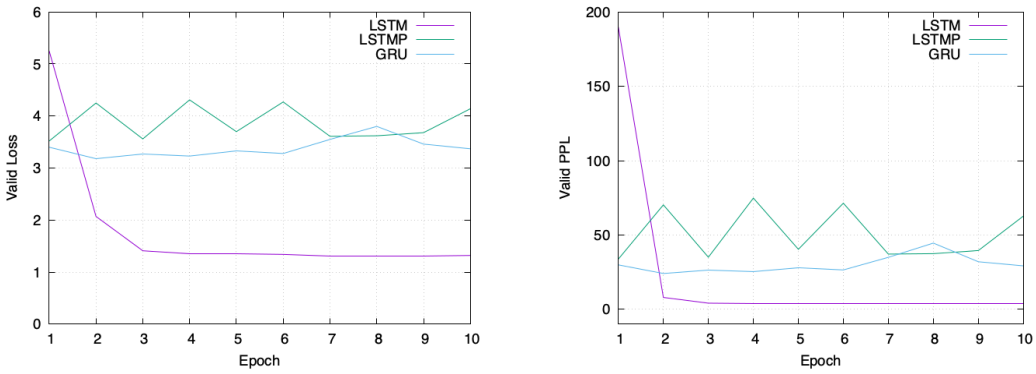
我们通过影响力剪枝和 SVD 低秩分解对 LSTM 网络进行压缩，与 GRU、LSTMP 和原始的 LSTM 进行比较，其相关性能效果比较如表4-1所示。其中在 Penn TreeBank 数据集实验中，虽然模型 LSTMP 和 GRU 模型的参数量分别为 32.060M 和 27.576M，他们的参数量都比原始的 LSTM 模型要小，但是他们的 PPL 和 BPC 值都远大于原始的 LSTM 模型。这说明他们的语言模型性能和原始 LSTM 模型相比，虽然参数量更少，但是他们的任务性能较差；其中 LSTMP 的测试损失为 6.52，PPL 为 678.51，BPC 为 9.406；GRU 的 Test Loss 为 6.52，PPL 为 681.26，BPC 为 9.412。通过本章算法的剪枝之后，模型的测试 Loss 进一步下降至 4.14，而测试 PPL 下降至 62.85，测试 BPC 下降至 5.974，可见 LSTM 模型仍然具有参数压缩的空间，同时证明了影响力剪枝的有效性；由于 SVD 分解是近似的低秩分解过程，存在部分损失，因此模型在剪枝和 SVD 分解之后，其参数量虽然降至 21.36M，但是模型性能有所下降，模型的 Test Loss 提升至 4.29，PPL 增加至 73.21，BPC 提升至 6.194；但是本章的压缩算法相较于 GRU 和 LSTMP 模型，其参数量更少，任务性能的损失也更少，因此总体而言本章算法在 Penn TreeBank 的语言任务中，其压缩效果是具备极大优势的。

表 4-2　基于 Enwik8 数据集的模型压缩实验效果

模型	Test Loss	PPL	BPC	Params	Params Drop Rate(%)
Origin LSTM	1.29	3.64	1.864	31.514M	-
LSTMP	3.55	34.79	5.121	28.132M	10.73
GRU	4.44	84.46	6.400	23.648M	24.96
Ours(Pruning)	**1.31**	**3.70**	**1.888**	-	-
Ours(Pruning+SVD)	1.59	4.88	2.287	**16.487M**	**47.68**

4.4.3　在 Enwik8 中的实验效果

在 Enwik8 数据集的实验中，我们将原始 LSTM、LSTMP 和 GRU 三个结构都进行了 10 个 Epoch 训练。由于本章算法额外进行数次的 Epoch 训练，因此，为了公平起见，我们应需让对比模型也进行同样多的 Epoch 训练，但是，在实际的 10 次训练过程中，模型训练已经逐渐稳定，额外的多次训练无法提升模型性能。我们对原始 LSTM、LSTMP 和 GRU 这三个模型进行了训练过程中的性能变化分析，他们的训练过程中的验证集 Loss 和 PPL 变化如图4-11所示，由训练曲线可知，在 5 个 Epoch 之后，原始 LSTM 的验证集 Loss 基本平稳，但是 LSTMP 和 GRU 结构的压缩在实验中仍然是不稳定的，我们选取验证集中 Loss 最小的模型进行比较，同时对原始 LSTM 的预训练模型进行剪枝和压缩工作。由于三个模型在 10 个 Epoch 中的训练效果判断，我们可以将压缩后的模型直接与上述三个模型进行比较，结果如表4-2所示。



(a) 在 Enwik8 进行 10 个 Epoch 训练的 Valid Loss 变化曲线

(b) 在 Enwik8 进行 10 个 Epoch 训练的 Valid PPL 变化曲线

图 4-11　模型在 Enwik8 进行 10 个 Epoch 验证集性能变化曲线

我们通过影响力剪枝和 SVD 低秩分解对 LSTM 网络进行压缩，与 GRU、LSTMP 和原始的 LSTM 进行比较。其中在 Enwik8 数据集实验中，虽然模型 LSTMP 和 GRU 模型的参数量分别为 28.132M 和 23.648M，他们的参数量都比

原始的 LSTM 模型要小，但是 LSTMP 的 Test Loss 比原始的 LSTM 结构的 Test Loss 要多 2.26，LSTMP 模型的 PPL 和 BPC 分别比原始 LSTM 模型多 30.15 和 3.257；而 GRU 的 Test Loss、PPL 和 BPC 值都略大于 LSTMP 模型，更是远大于原始的 LSTM 模型。这说明 GRU 在 Enwik8 数据集中的语言模型效果相比 LSTMP 模型性能较差，与原始 LSTM 模型比较性能更差；即使 GRU 模型的参数量得到有效减少，但是模型性能损失巨大。原始 LSTM 模型在通过本章算法的剪枝之后，模型的 Test Loss 只是略微升高，其值为 1.31，而 Test PPL 提升至 3.70，测试 BPC 提升至 1.888，可见 LSTM 模型仍然具有少部分参数压缩的空间，在剪枝之后模型性能仅略微的下降，证明了影响力剪枝是非常有效的剪枝方式，在 Params Drop Rate 较大的情况下，保证了精度的损失量在极小的范围；由于 SVD 分解存在部分损失，因此在进行剪枝和 SVD 分解之后，模型参数量虽然降至 16.487M，但是模型性能会有小部分损失，其 Test Loss 提升至 1.59，PPL 增加至 4.88，BPC 提升至 2.287。通过实验表明，压缩之后的模型性能远优于 GRU 和 LSTMP 模型的任务性能，同时本章算法压缩得到的模型参数量更少；因此总体而言本章算法在 Enwik8 数据集的语言模型学习任务中，其压缩效果仍然是具有绝对性优势的。

4.4.4 消融实验

通过4.3.3中介绍可知，本章算法需要应用到多个超参，其中包括剪枝压缩率 r、正则化项权重 λ、二值化调控值（β_x 和 β_h）以及奇异值域值 $thresh$。其中正则化项 λ 对初始化值不敏感，且在计算中依据公式4-10动态调整，不需要对其进行特别调控。因此本节主要对剪枝压缩率 r、二值化调控值（β_x 和 β_h）和奇异值域值 $thresh$ 进行消融实验。

4.4.4.1 压缩率

本章算法需要设定具体的目标剪枝压缩率 r，我们将压缩率与剪枝之后的模型分别在 Enwik8 和 Penn TreeBank 两个数据集中进行测试，实验结果如表4-3和表4-4所示。我们将不同 r 的 Test Loss 结果和对应的 Param Drop Rate 制成图4-12，观察其趋势变化。图4.12(a)和4.12(b)中“Params DR”指的是 Params Drop Rate 指

表 4-3　在 Enwik8 数据集中针对不同压缩率的模型性能比较

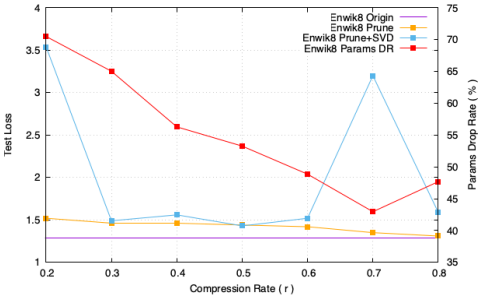
Compression Rate	Prune			Prune+SVD			Params Drop Rate(%)
	Loss	PPL	BPC	Loss	PPL	BPC	
1.0	1.29	3.64	1.864	-	-	-	-
0.8	1.31	3.70	1.888	1.59	4.88	2.287	47.68
0.7	1.35	3.87	1.951	3.20	24.53	4.616	42.96
0.6	1.42	4.12	2.043	1.52	4.55	2.187	48.86
0.5	1.44	4.20	2.071	1.43	4.18	2.063	53.27
0.4	1.46	4.30	2.104	1.56	4.74	2.245	56.29
0.3	1.46	4.30	2.105	1.49	4.44	2.152	65.03
0.2	1.52	4.58	2.194	3.54	34.39	5.104	70.50

表 4-4　在 Penn TreeBank 数据集中针对不同压缩率的模型性能比较

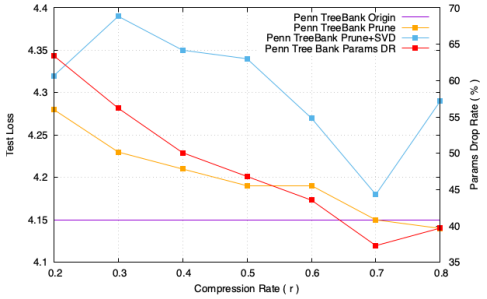
Compression Rate	Prune			Prune+SVD			Params Drop Rate(%)
	Loss	PPL	BPC	Loss	PPL	BPC	
1.0	4.15	63.33	5.985	-	-	-	-
0.8	4.14	62.85	5.974	4.29	73.21	6.194	39.71
0.7	4.15	63.63	5.992	4.18	65.61	6.036	37.30
0.6	4.19	66.35	6.052	4.27	71.73	6.164	43.55
0.5	4.19	66.11	6.047	4.34	77.08	6.268	46.81
0.4	4.21	67.04	6.067	4.35	77.51	6.276	50.07
0.3	4.23	69.00	6.109	4.39	80.27	6.327	56.20
0.2	4.28	71.95	6.169	4.32	74.85	6.226	63.40

标。从两个数据集的实验结果可知，随着压缩率 r 的增加，模型的 Params Drop Rate 逐步下降；当 $r = 0.7$ 时，模型的 Params Drop Rate 达到最低点，当 $r = 0.8$ 时模型的 Params Drop Rate 有所回升，主要原因是在实际剪枝过程中，在较高的目标压缩率下，其精度损失较少，因此损失函数让模型更关注于模型的压缩策略，从而提升了一定程度降低了实际的压缩率，即如图4.12(c)所示；这个现象同样证明了影响力剪枝的主要特点，即动态兼顾模型的人物性能和剪枝性能。

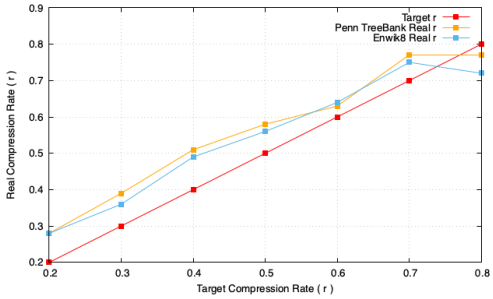
通过两个数据集实验可知，通过影响力剪枝造成的损失只在-0.01~0.15 之间，LSTM 压缩中，大部分的损失来源于 SVD 近似低秩分解的过程；例如在 Enwik8 数据集中，当 $r = 0.7$ 和 $r = 0.2$ 时损失最为明显，当 $r = 0.7$ 时模型剪枝之后的 Loss 为 3.20，而 $r = 0.2$ 时损失为 1.59，较大程度的影响了模型的人物性能；又如在 Penn TreeBank 数据集中，SVD 近似低秩分解造成的损失远超过剪枝压缩带来的损失。由图4-12可知，压缩算法很难兼顾模型性能最优且参数量最少两个优势；因此在应用中，我们需要根据实际背景对模型性能和压缩效果进行权衡。



(a) 在 Enwik8 任务中不同 r 的性能变化曲线



(b) 在 Penn TreeBank 任务中不同 r 的性能变化曲线



(c) 在不同 r 下的实际压缩率变化曲线

图 4-12 不同的 r 下的模型性能变化曲线

4.4.4.2 二值化调控

由于 β_x 和 β_h 初始化值的设置与模型的结构有关[55]，因此对于同一个 LSTM 网络中的二值化调控，我们可以将 β_x 和 β_h 设置为同一个初始化值。我们在 Penn TreeBank 数据集上进行了一组关于二值化调控的实验，设定目标压缩率 $r = 0.5$，结果如图4-13和表4-5所示。

由表4-5可知，当 β_x 和 β_h 初始化在 0~0.1 之间时，模型任务性能相对较差，Test Loss 相对较高；当 β_x 和 β_h 初始化在 0.1~0.2 之间时，模型任务性能相对较好；主要原因是在 β_x 和 β_h 初始化值过小的情况下，Sigmoid 映射的值延迟二值化过程，这将使对影响力矩阵进行降噪处理的卷积层 **K** 过度学习影响力矩阵，这种情况下，我们的剪枝方法将会存在偏差；例如当 β_x 和 β_h 初始化为 0.01 时，剪枝之后的 Test Loss 为 4.20，Test PPL 和 Test BPC 分别为 66.44 和 6.054，这相比于 β_x 和 β_h 初始化为 0.15 时的任务性能相对较差。当 β_x 和 β_h 初始化值过大的情况下，Sigmoid 映射的值提前完全二值化，这将使卷积层 **K** 无法充分对影响力参数进行降噪影响处理，我们的方法将退化为随机通道选择，从而导致了模型的任务性能下降；例如当 β_x 和 β_h 初始化为 0.15 时，Test Loss 为 4.18，Test

表 4-5　不同的 β_x 和 β_h 的初始化值对剪枝模型性能影响效果比较

β_x 和 β_h 的初始化值	Prune		
	Loss	PPL	BPC
0.01	4.20	66.44	6.054
0.02	4.20	66.38	6.053
0.05	4.20	66.40	6.053
0.08	4.20	66.48	6.055
0.10	4.19	66.11	6.047
0.12	4.19	66.13	6.047
0.15	4.18	65.67	6.037
0.20	4.19	65.93	6.043

PPL 和 Test BPC 分别为 65.67 和 6.037，相比于实验中的其他 β_x 和 β_h 初始值，其模型性能是最优的。综上所述，β_x 和 β_h 当初始化值需要在一个相对合适的范围中，从图4-13可知，本章剪枝算法的最优 β_x 和 β_h 初始化值应该在 0.15~0.2 之间。

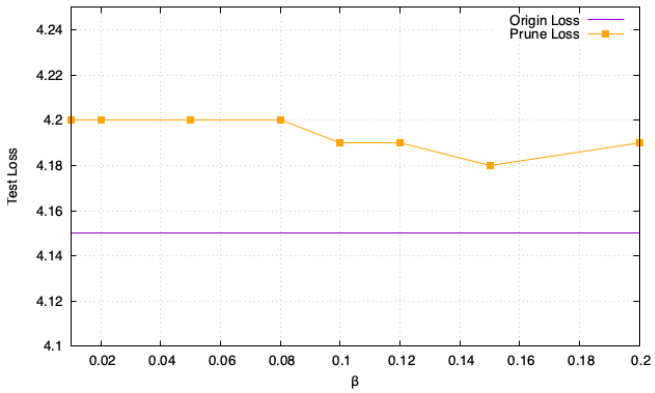
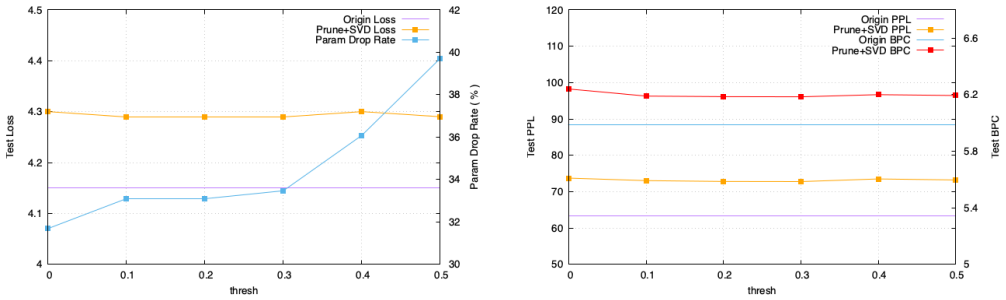


图 4-13　不同的 β_x 和 β_h 对剪枝模型性能影响曲线

4.4.4.3　低秩分解奇异值阈值

在实验中，*thresh* 过大则会将矩阵的秩降为 0，造成分解过度，无法进行正常的模型计算；*thresh* 过小则难以有效降低模型的参数量。本章算法的 *thresh* 需要保持在一个较小的范围之中。通过实验可知，进行低秩分解计算时，LSTM 网络的某个权重矩阵存在的最大奇异值为 0.7，因此我们只需要将 *thresh* 的值设置在 0~0.5 之间。我们在 Penn TreeBank 数据集下进行实验，设定压缩率 $r = 0.8$，设定二值化调控值 β_x 和 β_h 均为 0.1，则由本章算法压缩之后的模型性能变化如图4-14所示。由实验可知，模型性能对 *thresh* 的值敏感度较小，且模型通过 SVD

分解之后性能存在一定损失，这个损失量与 *thresh* 的值的小幅度变化没有直接关联；而模型的参数删减比例大致与 *thresh* 值呈正比关系，*thresh* 值愈小，模型参数删减的量也越小；此外，在实验中观察得知，模型的 Test PPL 值的变化趋势与 Test BPC 变化趋势是基本一致的。



(a) 删减的参数量变化和 Test Loss 变化曲线　　(b) Test PPL 和 BPC 变化曲线

图 4-14 不同的 *thresh* 下的模型性能变化曲线

4.5 本章小结

本章基于影响力测算的方式实现了非结构化剪枝和低秩分解的压缩算法，该算法主要针对 LSTM 模型进行压缩，因此首先对 LSTM 模型的记忆单元结构进行分析，研究其结构特点，根据模型参数矩阵的情况，获取权重矩阵的影响力数据，依据影响力分布进行非结构化剪枝，剪枝之后，权重矩阵变成了稀疏矩阵。再将稀疏矩阵进行矩阵分解，将矩阵分解为两个参数量更小的矩阵，降低模型的参数量，极大缩短了模型的运算时间，降低了模型存储，在语言模型中的效果得到了有效的提升。同时在语音识别的效果也十分显著。

第五章　网络加速在多通道语音增强系统的应用

5.1　语音增强和模型压缩

近年来，基于深度学习的单通道语音增强方法大大提高了语音增强系统的语音质量和清晰度。这些方法通常在监督环境中进行训练，可以分为时域和频域方法。时域方法[74]使用神经网络将嘈杂的语音波形映射到直接清晰的语音波形。频域方法通常使用噪声频谱特征（例如，复谱，幅度谱）作为神经模型的输入。学习目标是清晰语音或特定掩码的频谱特征（例如，理想二进制掩码[75]，理想比率掩码[76]，复杂理想比掩模[77]）。一般来说，由于时域信号的维数高且缺乏明显的几何结构，频域方法仍然主导着绝大多数语音增强方法。

在之前的工作中，有研究提出了一种基于子频段的单通道语音增强方法。与传统的基于全频段的方法不同，该方法以子频段样式执行。子频段模型满足了实时性要求，性能也极具竞争力。但是，由于它无法对全局光谱模式进行建模并利用长距离跨波段依赖关系。另一方面，全频段模型被训练来学习高维输入和输出之间的回归，缺乏专用于子带信息的机制，例如信号平稳性。随后，研究提出了许多全频带和子频段融合模型。全频段模型的输出是子频段模型的输入。通过有效的联合训练，这两种模式共同优化。

基于深度学习模型的语音增强算法对于非平稳噪声有较好的抑制作用，超过了传统的降噪算法。因此，语音增强任务通常采用的是深度网络模型，但模型参数量大，使得终端设备上运行模型较为困难。但是，随着小型边缘设备应用智能化需求增多，多通道语音增强系统往往需要部署在小型设备当中，这就要求语音增强模型必须进行轻量化处理。

5.2 语音增强系统

多通道语音增强需要在会议场景下，对获取的语音数据进行预处理，然后通过模型进行非平稳噪声的降噪和获取清晰的语音波形，从而能够在实际场景中获取纯净的原始语音。因此，多通道语音增强技术主要是对语音进行降噪提取工作。本系统需要使用到语音增强模型，这类模型可以部署到云端服务器中，也可以基于便捷性考虑，部署在终端设备上。但是由于云计算需要远程传输数据，对于语音信号传输来回增加了较多的时延，消耗较多的存储，且信息安全性下降。如果仅部署于手机等移动边缘设备中，那么就可以将手机获取的语音信息直接进行降噪增强处理，这样既降低了时延，节省了存储空间，也保证了数据的安全性。但普通语音部署于小型边缘设备中是一个较大的挑战。因此，本章基于前文中提出的两个加速压缩的算法，从不同的模型结构角度，针对多通道语音增强模型进行压缩加速，较好地完成了模型在小型移动设备上的部署工作。

5.2.1 系统需求

对于在会议场景下运行的语音增强系统，我们将系统的核心需求总结为以下三个方面：

1. 边缘计算能力强：考虑到在会议场景下进行语音增强，需要移动边缘设备部署，所以需要进行模型压缩。通常小米移动设备例如手机都是基于 ADSP 卡等性能相比于 GPU 更弱的硬件。而语音增强模型对于语音处理需要收集大量的语音数据，如果这些语音数据全部传输给云服务器，不仅耗费较多的传输和存储资源，也大大增加了处理时延，无法实现实时性处理。所以在本系统中的模型必须有强大的边缘计算能力。
2. 非平稳噪声的抑制能力强：系统的最终目标是要求能够保证语音数据纯净且不失真。那么保证模型的语音增强性能更加重要，否则使用深度模型代替传统处理方式则失去了本来的意义。在深度模型中，模型对于非平稳噪声应该具有较强的抑制能力，才能保证模型本身的性能。
3. 模型迭代能力高效：在手机 ADSP 卡上部署模型相对云计算部署工作是更加繁琐的。如果每次迭代都需要影响平台，使得平台的改动较多，那必然会极大的降低整个系统的效率，所以需要一个方便搭建部署的稳定系统，

保证长时间内较少的改动，只需要软件上的更新迭代，这样才能保证整个系统软件运行的可持续化。

由上述三点需求可知，如果要求模型的边缘计算能力强，那么复杂的网络结构和较深的网络深度应当不合适作为系统模型，同时为了系统能够保证长期平稳运行，减少模型的迭代，应减少过多的深度模型应用，因此数据针对预处理采用了基于传统的语音增强预处理方法，而传统方法对许多非平稳噪声的抑制能力不如深度学习，也能够通过语音增强模型进行针对性学习训练来弥补。因此整个系统的主要部分就在于语音增强模型，其计算的主要消耗点也来源于语音增强模型。考虑到会议场景模式下，所具有的噪声是具有固定种类的，相应的网络模型就不需过于复杂，考虑模型的部署环境，由此要求模型需要进行轻量化处理。在会议场景下的模型精度对于网络参数的裁剪是不太敏感的，所以在精度和速度的权衡中，系统更应该注重模型的速度提升和参数量的下降，通过加大对模型参数的删减，来换得模型运行的速度提升。

基于系统需求的分析研判，本系统主要重点在于语音增强工作，采用了深度模型来实现对语音中非平稳噪声的处理，但需要在移动设备上便捷使用。对于本系统中的深度模型，其结构都含有 LSTM、CNN 和 FC，我们可以根据不同结构使用我们的压缩方法进行参数量的压缩；具体来说，系统对于 CNN、LSTM 和 FC 结构均采用了基于影响力剪枝的算法，然后对于模型中带有的 LSTM 结构在进行假性剪枝之后，进行 SVD 分解压缩，最终实现性能优越的压缩网络，使得模型的运算速度得到较大提升。

5.2.2　系统架构

本文将整个系统的运行主要划分为三个部分，第一部分主要是信息的的采集获取和预处理，然后是语音增强部分，最后是对语音增强处理后的数据进行信息整合。本系统更加关注第二部分，它是唯一一个与深度网络有关联的部分。系统的运行的流程如图5-1所示。首先开始发出语音，然后进行信号采集工作，当采集的语音信号满 3s 以上，则将采集到的语音数据进行分析缓存，否则丢弃现有的已收集但未缓存的数据，重新进行语音采集工作。对分析缓存的语音数据进行预处理，包括语音预加重处理和信号的分帧加窗处理。其中语音预加重处理

主要目的在于去除语音信息中，口唇辐射的影响，提升语音高频信息的分辨率；而信号的分帧加窗处理来保证语音数据的连续性，依靠帧与帧之间的关联性，让帧之间的过渡更加平滑，因此也需要让分段交叠在一起。再将预处理好的数据送入深度模型中进行进一步的语音增强工作。这里的语音增强模型即采用了通过影响力剪枝算法得到的深度模型。语音增强的基模型使用的是 FullSubNet[78]和 TCN[79]，这些模型同时具有 LSTM 结构和卷积结构，基于不同结构进行剪枝和 SVD 分解压缩，实现模型最大程度的精简。而且，模型剪枝和 SVD 分解工作的结合，实现了模型进一步的加速。在语音数据经过语音增强模型处理之后，得到原始纯净的语音数据，结果数据需要通过相应的指标评价原始语音是否符合标准。如果处理后的数据不符合标准，系统则将语音丢弃，前端界面返回处理失败提示；而通过语音质量测试的数据，会调回前段界面展示相关信息，同时保存已处理的语音结果在指定目录中。

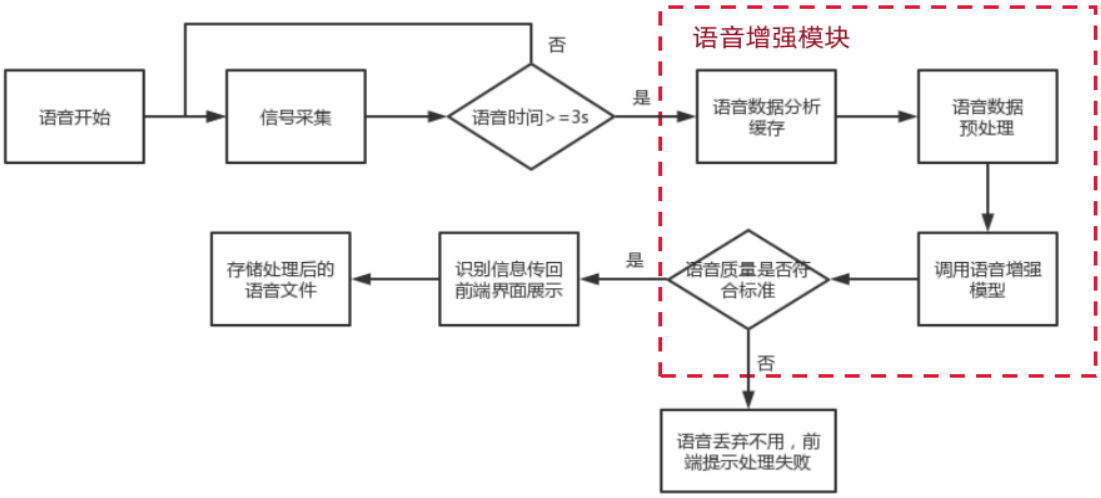
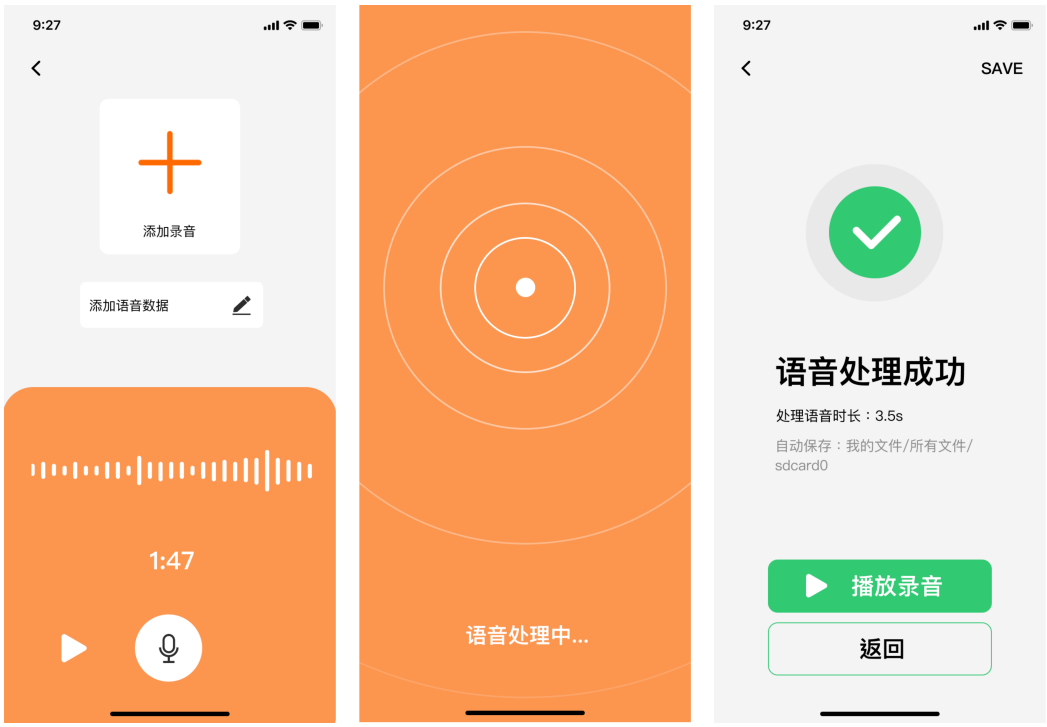


图 5-1 语音增强系统流程图

5.2.3 系统效果

本系统主要注重于语音增强模型的优化加速，同时也为用户提供了移动段的可视化界面，利于语音收集和处理工作更方便的进行。本系统经过多次实际检测，经过优化加速的模型的其时延与原模型时延相比大大缩短时间，满足了语音增强功能的实时性需求，同时保证模型性能较好的达到系统的需求，使得系统具有较高的实际应用价值，极大的提升了移动端进行语音处理的工作效率。

5.2.3.1　前端界面演示



(a) 主界面　　(b) 正在处理语音　　(c) 语音处理成功

图 5-2　语音增强系统界面

本系统在执行语音增强任务时，设置了前端页面。前端页面首页能够和添加选择语音现场录音或者加载录音文件。如图5.2(a)所示，在移动端的开始页面正下方有一个录音按钮，点击录音按钮则开始进行录音；图中正中间有一个添加语音文件的按钮，当点击添加语音时，能够跳转到移动端本地的文件存储库来点击你想要处理的语音文件。系统需要判断语音长度是否符合要求，如果文件的语音或者录音的长度小于 3 秒，则前端给出提示，显示语音过短需要重新输入。如图5.2(b)所示，当语音长度符合要求时，系统将文件传入后台缓存，然后前端界面显示文件正在处理中，此时文件送入移动端的后台进行语音预处理工作，然后调用语音增强模型处理语音文件。如果出现语音处理结果不达标，则将处理后语音文件丢弃，处理失败的信息发回前端界面，界面提示语音处理失败，然后在左上角和页面中间提供返回按钮，返回主页面，方便重新处理语音。如图5.2(c)所示，将处理成功的文件发回前端界面，界面提示处理成功，且显示文件的语音长度，同时，界面设置了播放按钮和返回按钮，此时，界面提示文件自动保存且显示保存的移动端存储地址，播放按钮可以播放处理好的语音用于人工判断是否符合要求，返回按钮主要用于返回主界面继续处理新的语音文

表 5-1 影响力剪枝及 SVD 分解压缩在语音增强系统中的应用效果

方法	SI_SDR	STOI	PESQ	Params	RTF
FullSubNet	20.34	0.9208	2.5201	6.32M	1.02
FullSubNet(影响力剪枝 +SVD 分解)	19.52	0.9156	2.4451	**1.39M**	**0.81**
TCN	20.52	0.9244	2.7129	6.16M	0.99
TCN(影响力剪枝 +SVD 分解)	**20.57**	**0.9267**	**2.7373**	**0.81M**	**0.41**

件。如果无法自动保存，则可以点击页面右上方的“SAVE”按钮。详细内容如图5-2所示。

5.2.3.2 加速效果

本系统使用的是开源数据集,但基于系统的实际需求,需要对数据进行整合,并规范为算法能够直接调用的数据接口。首先将 THCHS-30、Primeword、MAGICDATA、Aishell、Wham_noise、MUSAN_noise(经过数据增强处理)、DNS_noise 等多个开源数据集通过数据整合、切分之后混合成大数据集进行测试；在整合噪声数据的过程中发现 MUSAN_noise 数据集中的噪声数据仅仅只有数千条，比起 Wham_noise 和 DNS_noise 的噪声数目要少得多。而 Wham_noise 主要是环境噪声中的人声噪声，MUSAN_noise 中噪声种类相对会比较丰富，我们想尽可能利用这些信息。所以我们对 MUSAN_noise 做了数据增强。噪声数据集总数据量为：训练集 104541 条，验证集 29868 条，测试集 38990 条。由于数据集为多为单通道数据，而本系统要支持多通道语音增强工作，因此对语音数据进行了数据的动态增强、加噪和加混响处理.

在系统中运用了本文提出的基于不同结构的影响力剪枝和 SVD 分解压缩的加速方法，而系统的语音增强模型因为对其结构有针对性的剪枝，让模型最大程度的精简且性能保持较好的水平。将模型部署在移动端中，模型的前向传播过程时间大大缩短。本文提出的每一个方法经过实际应用的验证，均为模型的运行加速和时延缩短带来了显著效果。我们使用实时率（RTF）来测试时延情况，其中当 RTF 小于 1 时，则可以认为时实时的，RTF 越小，时延越短；其中最佳的 RTF 在 0.2 到 0.3 之间。语音增强模型的测评指标有三个，分别是 SI_SDR[80]、STOI[81]和 PESQ[82]。SI_SDR 表示的是尺度不变信噪比，主要衡量的是降噪算法的信噪比提升；STOI 指的是短时客观可懂度，反映了人类的听觉感知系统对语音可懂度的客观评价，STOI 值介于 0-1 之间，值越大代表语音可懂度越高，越

清晰；PESQ 属于客观评价，和主观分数之间的相关性约为 0.935。其取值在-0.5 到 4.5 的范围内，得分越高表示语音质量越好。系统的语音增强基模型采用的是 FullSubNet 和 TCN 模型结构。在测试过程中，通过我们的压缩加速方法优化实现的模型大大降低了模型的参数量，且均保证了模型的实时性，例如 TCN 模型经过我们的方法进行压缩加速处理，实现实时率从 0.99 降低到 0.41，大幅度降低了时延；而其模型参数量从 6.16M 降低到了 0.81M，使模型降低了存储量，且 SI_SDR、STOI 和 PESQ 都得到了小幅度的提升，其值分别为 20.57、0.9267 和 2.7313。具体结果图表5-1所示。

5.3　本章小结

本章中介绍的重点是基于本文算法的一个系统应用实例——基于影响力剪枝和 SVD 分解压缩的语音增强系统，通过深度模型将实际收集的语音信息进行处理，从而获取纯净的原始语音。基于本文算法的压缩，语音增强模型才能够实现边缘计算，同时保证低时延，实现实时性语音处理，而且能够降低模型存储、计算以及部署的成本。以上都证明了本文算法充分具备有效性，同时证实了模型加速在实际多个领域中都有着很大需求和研究价值。

第六章　总结与展望

随着网络的深度越来越深，参数量越来越大，模型的压缩工作也变得日益重要。本文在调研中发现，在模型剪枝压缩工作中，其剪枝的参考指标选择是至关重要的。常见的研究使用模型参数量大小、重构误差以及数值大小等多种网络层信息，但是目前还未有文献证明这些的剪枝工作具备可解释性。本文的工作以神经网络的可解释性出发，提出了基于影响函数理论进行模型权重的影响力计算，通过影响力进行剪枝。并且将这种影响力测量理论运用于不同的网络结构中进行实现。对于卷积神经网络，本文提出了在卷积层和全连接层中基于影响力的通道剪枝算法。该算法对卷积网络模型进行逐层剪枝，模型自身通过反向传播获取影响力矩阵，通过对影响力矩阵进行降噪、二值化和假性剪枝训练等工作，得到最终的通道剪枝策略；该算法等评价指标是各个权重的影响力，其相对于根据网络参数值大小、输入数据大小等指标更具有可解释性；而且，在卷积网络剪枝过程中，该算法将剪枝和微调过程结合进行，同时不依靠别的其他辅助模型，实现了端到端的自动剪枝工作。我们将基于卷积网络的最新剪枝压缩算法与本文的算法在 CIFAR-10 和 CIFAR-100 的数据集中进行比较，结果显示，在算法比较中同样的待压缩模型和同样 FLOPs 下降比率下，我们的算法进行模型压缩导致的精度损失量要少于其他算法的压缩精度损失，甚至我们的算法能够小幅度提升模型的精度。而对于长短期记忆网络结构，本文提出了在长短期记忆单元中基于影响力的非结构化剪枝策略，为了让非结构化剪枝不需要特定的硬件支持，我们又在剪枝工作之后进行了低秩分解压缩，将剪枝之后的稀疏矩阵转化为维度更小的矩阵，从而实现网络的结构化压缩。该算法在 Penn TreeBank 数据集中的实验可知非结构化剪枝的精度优于原模型。基于长短期记忆网络的剪枝压缩既降低模型参数量、最大程度的保证精度损失在极小范围内，同时又解决了非结构化剪枝需要特定硬件支撑的问题。最后，本文将基于影响力剪枝压缩算法运用于语音增强系统中，该系统涉及了长短期记忆单元结构和

卷积网络结构。通过压缩后的模型能够成功部署于移动端中。系统的测试结果证明了该算法在实际的系统应用中具备有效性。

按照本文现有的研究进展，我们根据各个研究工作分析总结了一下几点可以继续提升改进的方向。

- 改进超参 β 的初始化设计。本文提出的基于影响力剪枝的算法都需要对超参 β 进行初始化，且根据模型结构的不同，β 初始化数值也应设置为不同的值。超参的设置主要是作用在模型剪枝策略的二值化过程中的调控部分。因此我们可以考虑选择更加智能化的二值化过程，使其不需要考虑超参 β 的初始化问题。
- 对更多结构进行扩展。目前本文提出的算法仅在卷积网络和长短期记忆网络中进行实验，但当前神经网络结构十分多样，我们可以在其他常用的网络结构中进一步推广这种剪枝算法，这能让更多的模型结构通过我们的算法在实际应用中更易于被使用，从而推动神经网络剪枝压缩的发展。
- 集成剪枝的工具包。本文的算法是根据不同的结构设计不同的剪枝算法，虽然理论基础是统一的，但是实际剪枝措施各有千秋。我们可以在之后的工作中，将基于影响力的剪枝算法集成一个剪枝的工具包，工具包能够在读取深度网络结构之后进行分析，然后进行端到端的剪枝，最终输出一个剪枝好的网络结构。而如何将针对不同结构的基于影响力的剪枝算法集成一个剪枝工具包，将是我们长久探究的一个课题。

参考文献

[1] HINTON G E, OSINDERO S, TEH Y W. A fast learning algorithm for deep belief nets[J]. Neural computation, 2006, 18(7): 1527-1554.

[2] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The journal of machine learning research, 2014, 15(1): 1929-1958.

[3] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25.

[4] BA J, CARUANA R. Do deep nets really need to be deep?[J]. Advances in neural information processing systems, 2014, 27.

[5] DENG J, DONG W, SOCHER R, et al. Imagenet: A large-scale hierarchical image database[C]//2009 IEEE conference on computer vision and pattern recognition. 2009: 248-255.

[6] HUANG G B, MATTAR M, BERG T, et al. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments[C]//Workshop on faces in'Real-Life'Images: detection, alignment, and recognition. 2008.

[7] HE K, ZHANG X, REN S, et al. Identity mappings in deep residual networks[C]//European conference on computer vision. 2016: 630-645.

[8] MITCHELL T M, et al. Machine learning. 1997[J]. Burr Ridge, IL: McGraw Hill, 1997, 45(37): 870-877.

[9] WEN W, WU C, WANG Y, et al. Learning structured sparsity in deep neural networks[J]. Advances in neural information processing systems, 2016, 29.

[10] RASTEGARI M, ORDONEZ V, REDMON J, et al. Xnor-net: Imagenet classification using binary convolutional neural networks[C]//European conference on computer vision. 2016: 525-542.

[11] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[J]. Advances in neural information processing systems, 2015, 28.

[12] MOLCHANOV D, ASHUKHA A, VETROV D. Variational dropout sparsifies deep neural networks[C]//International Conference on Machine Learning. 2017: 2498-2507.

[13] MA N, ZHANG X, ZHENG H T, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 116-131.

[14] ZHANG X, ZHOU X, LIN M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 6848-6856.

[15] CHENG Y, WANG D, ZHOU P, et al. A survey of model compression and acceleration for deep neural networks[J]. arXiv preprint arXiv:1710.09282, 2017.

[16] ZAGORUYKO S, KOMODAKIS N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer[J]. arXiv preprint arXiv:1612.03928, 2016.

[17] MOLCHANOV P, TYREE S, KARRAS T, et al. Pruning convolutional neural networks for resource efficient inference[J]. arXiv preprint arXiv:1611.06440, 2016.

[18] SRIVASTAVA R K, GREFF K, SCHMIDHUBER J. Highway networks[J]. arXiv preprint arXiv:1505.00387, 2015.

[19] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[20] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. SqueezeNet: AlexNet-level

accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[21] ZHANG X, ZOU J, HE K, et al. Accelerating very deep convolutional networks for classification and detection[J]. IEEE transactions on pattern analysis and machine intelligence, 2015, 38(10): 1943-1955.

[22] LEBEDEV V, GANIN Y, RAKHUBA M, et al. Speeding-up convolutional neural networks using fine-tuned cp-decomposition[J]. arXiv preprint arXiv:1412.6553, 2014.

[23] YAO Q, WANG M, CHEN Y, et al. Taking human out of learning applications: A survey on automated machine learning[J]. arXiv preprint arXiv:1810.13306, 2018.

[24] BACK T. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms[M]. Oxford university press, 1996.

[25] WILLIAMS R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. Machine learning, 1992, 8(3): 229-256.

[26] ZOPH B, LE Q V. Neural architecture search with reinforcement learning[J]. arXiv preprint arXiv:1611.01578, 2016.

[27] SMITH M G, BULL L. Genetic programming with a genetic algorithm for feature construction and selection[J]. Genetic Programming and Evolvable Machines, 2005, 6(3): 265-281.

[28] HE Y, LIN J, LIU Z, et al. Amc: Automl for model compression and acceleration on mobile devices[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 784-800.

[29] WANG K, LIU Z, LIN Y, et al. Haq: Hardware-aware automated quantization [J]. arXiv preprint arXiv:1811.08886, 2018, 3(4).

[30] KOH P W, LIANG P. Understanding black-box predictions via influence functions[C]//International conference on machine learning. 2017: 1885-1894.

[31] HUBER P J. Robust statistics: vol. 523[M]. John Wiley & Sons, 2004.

[32] MARCINKIEWICZ M A. Building a large annotated corpus of English: The Penn Treebank[J]. Using Large Corpora, 1994: 273.

[33] JAMES W, BURKHARDT F, BOWERS F, et al. The principles of psychology: vol. 1[M]. Macmillan London, 1890.

[34] MARSLAND S. Machine learning: an algorithmic perspective[M]. Chapman, 2011.

[35] HEBB D O. The Organization of Behavior: A Psychological Theory[M]. Wiley New York, 1949.

[36] ROSENBLATT F. The perceptron: a probabilistic model for information storage and organization in the brain.[J]. Psychological review, 1958, 65(6): 386.

[37] HOPFIELD J J. Neural networks and physical systems with emergent collective computational abilities[J]. Proceedings of the national academy of sciences, 1982, 79(8): 2554-2558.

[38] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[39] GOLGI C, y CAJAL S R. The Nobel Prize in Physiology or Medicine, 1906[J]. Nobel Lecture, December, 1906, 11.

[40] HUBEL D H, WIESEL T N. Receptive fields of single neurones in the cat's striate cortex[J]. The Journal of physiology, 1959, 148(3): 574.

[41] HUBEL D H, WIESEL T N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex[J]. The Journal of physiology, 1962, 160(1): 106.

[42] FUKUSHIMA K, MIYAKE S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition[G]//Competition and cooperation in neural nets. Springer, 1982: 267-285.

[43] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[44] WAN L, ZEILER M, ZHANG S, et al. Regularization of neural networks using dropconnect[C]//International conference on machine learning. 2013: 1058-1066.

[45] MAAS A L, HANNUN A Y, NG A Y, et al. Rectifier nonlinearities improve neural network acoustic models[C]//Proc. icml: vol. 30: 1. 2013: 3.

[46] HINTON G E. Rectified linear units improve restricted boltzmann machines vinod nair[J]., 2010.

[47] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[48] GUPTA S, AGRAWAL A, GOPALAKRISHNAN K, et al. Deep learning with limited numerical precision[C]//International conference on machine learning. 2015: 1737-1746.

[49] WU J, LENG C, WANG Y, et al. Quantized convolutional neural networks for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 4820-4828.

[50] COURBARIAUX M, HUBARA I, SOUDRY D, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1[J]. arXiv preprint arXiv:1602.02830, 2016.

[51] DENIL M, SHAKIBI B, DINH L, et al. Predicting parameters in deep learning [J]. Advances in neural information processing systems, 2013, 26.

[52] STREET J O, CARROLL R J, RUPPERT D. A note on computing robust regression estimates via iteratively reweighted least squares[J]. The American Statistician, 1988, 42(2): 152-154.

[53] FISHER R A. On the mathematical foundations of theoretical statistics[J]. Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character, 1922, 222(594-604): 309-368.

[54] GATEAUX R. Sur les fonctionnelles continues et les fonctionnelles analytiques [J]. CR Acad. Sci. Paris, 1913, 157(325-327): 65.

[55] LUO J H, WU J. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference[J]. Pattern Recognition, 2020, 107: 107461.

[56] KRIZHEVSKY A, HINTON G, et al. Learning multiple layers of features from tiny images[J]., 2009.

[57] PENG H, WU J, CHEN S, et al. Collaborative channel pruning for deep networks [C]//International Conference on Machine Learning. 2019: 5113-5122.

[58] ZHUANG T, ZHANG Z, HUANG Y, et al. Neuron-level Structured Pruning using Polarization Regularizer[J]. Advances in Neural Information Processing Systems, 2020, 33.

[59] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. 2015: 448-456.

[60] HE Y, LIU P, WANG Z, et al. Filter pruning via geometric median for deep convolutional neural networks acceleration[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 4340-4349.

[61] LIU Z, LI J, SHEN Z, et al. Learning efficient convolutional networks through network slimming[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2736-2744.

[62] DING X, HAO T, TAN J, et al. Resrep: Lossless cnn pruning via decoupling remembering and forgetting[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 4510-4520.

[63] HE Y, ZHANG X, SUN J. Channel pruning for accelerating very deep neural networks[C]//Proceedings of the IEEE international conference on computer vision. 2017: 1389-1397.

[64] WANG W, FU C, GUO J, et al. Cop: Customized deep model compression via regularized correlation-based filter-level pruning[J]. arXiv preprint

arXiv:1906.10337, 2019.

[65] KALMAN D. A singularly valuable decomposition: the SVD of a matrix[J]. The college mathematics journal, 1996, 27(1): 2-23.

[66] HAN S, KANG J, MAO H, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga[C]//Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2017: 75-84.

[67] CAO S, ZHANG C, YAO Z, et al. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity[C]//Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2019: 63-72.

[68] WANG S, LI Z, DING C, et al. C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs[C]//Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2018: 11-20.

[69] ZHANG Y, WANG C, GONG L, et al. Implementation and optimization of the accelerator based on FPGA hardware for LSTM network[C]//2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). 2017: 614-621.

[70] SAK H, SENIOR A, BEAUFAYS F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling[J]. Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, 2014: 338-342.

[71] CHO K, VAN MERRIËNBOER B, GULCEHRE C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[72] BROWN P F, DELLA PIETRA S A, DELLA PIETRA V J, et al. An estimate of an upper bound for the entropy of English[J]. Computational Linguistics, 1992, 18(1): 31-40.

[73] GRAVES A. Generating sequences with recurrent neural networks[J]. arXiv preprint arXiv:1308.0850, 2013.

[74] TAMURA S. An analysis of a noise reduction neural network[C]//International Conference on Acoustics, Speech, and Signal Processing, 1989: 2001-2004.

[75] SIMPSON A J, ROMA G, PLUMBLEY M D. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network[C]// International Conference on Latent Variable Analysis and Signal Separation. 2015: 429-436.

[76] NARAYANAN A, WANG D. Improving robustness of deep neural network acoustic models via speech separation and joint adaptive training[J]. IEEE/ACM transactions on audio, speech, and language processing, 2014, 23(1): 92-101.

[77] WILLIAMSON D S, WANG Y, WANG D. Complex ratio masking for monaural speech separation[J]. IEEE/ACM transactions on audio, speech, and language processing, 2015, 24(3): 483-492.

[78] HAO X, SU X, HORAUD R, et al. FullSubNet: a full-band and sub-band fusion model for real-time single-channel speech enhancement[C]//ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2021: 6633-6637.

[79] LIN J, van WIJNGAARDEN A J D L, WANG K C, et al. Speech enhancement using multi-stage self-attentive temporal convolutional networks[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2021, 29: 3440-3450.

[80] LE ROUX J, WISDOM S, ERDOGAN H, et al. SDR–half-baked or well done? [C]//ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2019: 626-630.

[81] TAAL C H, HENDRIKS R C, HEUSDENS R, et al. A short-time objective intelligibility measure for time-frequency weighted noisy speech[C]//2010 IEEE international conference on acoustics, speech and signal processing. 2010: 4214-4217.

[82] RIX A W, BEERENDS J G, HOLLIER M P, et al. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs[C]//2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221): vol. 2. 2001: 749-752.

致　　谢

时光蹉跎，白驹过隙。三年将至，毕业在即，念及过往，感触良多，心有不舍，思绪万千。故借此文，感怀于心。

一谢恩师申富饶之提携。承蒙恩师不弃，尽心尽力，言传身教，耳提面命，关爱有加。恩师教导，凡事善寻本源，勤学慎思，严谨踏实，淳淳教诲，吾铭记于心，更受益无穷。师之教诲，如沐春风，师之恩情，永世不忘。

二谢师长赵健之关怀。幸得师长关照，不厌其烦，不辞辛苦，常分享写作之得。每遇学生糊涂之作，必细细改之，字斟句酌，精益求精。得师长所授，获益匪浅。

三谢同窗师门之善待。贵得同窗友善，师门和睦，相呴相济，携手并进，同探学海之涯，共悟“研究”之道。尤谢（张）雅楠与严骅，予文章之指点，言辞谦虚，意见恳切。获二人建议，终成此文。

四谢亲朋好友之助力。幸吾双亲开明，重诗书，教礼仪，予财帛之助力，护吾健康成长。又得友人（梁）耀辉陪伴，任抱怨，忍脾气，伴吾艰难之时，解吾人生之惑。受父母亲朋之爱护，终有今日。

细数过往，乃至今日，多得贵人相助，是吾三生有幸。言语粗糙，不求顺达，寥寥数笔，仅抒胸意。愿师长安康，同窗顺遂，父母称心，亲朋如意。

简历与科研成果

基本信息

赖碧兰，女，汉族，1995 年 11 月出生，福建省三明人

教育背景

2019 年 9 月 — 2022 年 6 月	南京大学计算机科学与技术系	硕士
2014 年 9 月 — 2018 年 6 月	华南农业大学数学与信息学院	本科

攻读硕士学位期间发表的学术成果

[1] LAI B, XIANG H, SHEN F. Inf-CP: A Reliable Channel Pruning based on Channel Influence[J]. arXiv preprint arXiv:2112.02521, **2021**.

攻读硕士学位期间的专利成果

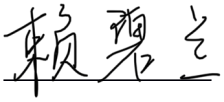
1. 申富饶、**赖碧**兰、安俊逸、赵健. “一种复杂环境下的三维定位追踪方法”(202010927152.1)
2. 申富饶，**赖碧兰**，赵健. “用于室内定位的基于实时轨迹动态进行二维跳点修正方法”(202110047580.X)

攻读硕士学位期间参与的科研课题

1. 国家自然科学基金“基于深度感知增量式联想记忆神经网络的信息融合系统研究”（课题年限 2019.01~2022.12），负责神经网络模型压缩加速相关研究。

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》(以下简称“章程”)，愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名： 

2022 年 5 月 20 日

论文题名	基于神经网络结构的压缩加速研究				
研究生学号	MG1933032	所在院系	计算机科学与技术系	学位年度	2022 年
论文级别	☑学术学位硕士 ☐专业学位硕士 ☐学术学位博士 ☐专业学位博士 (请在方框内画钩)				
作者 Email	laibilan_8968@163.com				
导师姓名	申富饶				

论文涉密情况：

☑ 不保密

☐ 保密，保密期（_____年_____月_____日 至 _____年_____月_____日）

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。