

语音增强初探

The Guidebook of Speech Enhancement

刘文哲

Contents

Part I 基于深度学习的单通道语音增强

1	基于幅度谱的语音增强	9
1.1	基于掩蔽的语音增强算法	12
1.1.1	处理流程	12
1.1.2	用于幅度谱的时频掩蔽	15
1.2	基于谱映射的语音增强算法	17
1.2.1	处理流程	17
1.3	小结	22
2	基于复数谱的语音增强	23
2.1	基于复数谱映射的语音增强	24
2.2	基于笛卡尔坐标系的复值掩蔽方法	27
2.3	基于极坐标系的复值掩蔽方法	31
2.4	基于幅度相位分量估计的语音增强	33
2.5	小结	35
3	混合式和生成式语音增强模型	37
3.1	混合式语音增强模型	37
3.2	生成式语音增强模型	40
3.3	小结	44
4	基于时域的语音增强	45
4.1	基于波形映射的时域语音增强模型	45
4.2	基于变换域掩蔽的时域增强模型	47
5	基于解耦的语音增强	51
5.1	基于评价指标解耦的语音增强算法	51
5.2	基于语音成分解耦的语音增强算法	53
5.2.1	基于幅度相位补偿的语音增强算法	53
5.2.2	基于谱包络-周期性语音成分分解的语音增强方法	58
5.2.3	基于子带分解的语音增强算法	59

5.3	基于语音增强任务理解解耦的语音增强算法	60
5.3.1	基于噪声抑制和语音恢复的语音增强算法	60
5.3.2	基于参数再合成的语音增强方法	63
5.4	小结	64
6	语音增强算法的因果性	65
6.1	语音音量的归一化	66
6.2	因果模块讨论	67
	References	69

Part II 基于深度学习的阵列信号处理

Part I

基于深度学习的单通道语音增强

语音增强算法引入深度学习大抵是一种必然，讨论基于深度学习的语音增强算法我们必须首先把目光放在Prof. Wang的OSU-PNL身上。Prof. Wang 是用于语音分离的计算听觉场景分析(CASA)领域的代表人物，他提出用理想二值掩蔽(IBM)解决语音增强(分离)问题。理想二值掩蔽，其中掩蔽广泛用于图像处理之中，具体到语音增强领域是指对选定的时频区域进行遮挡以控制要处理的区域，二值则是对每个时频点量化的精度非0即1，理想二值掩蔽的含义则是根据纯净语音和噪声之间的能量关系，将语音能量占主导的时频点标记为1，噪声能量占主导的时频点标记为0，由此得到一个滤波器，对带噪特征进行滤波，在语音时频稀疏性假设下留下的即使语音成分。于是，熟悉机器学习的读者应该很容易发现，语音增强问题在IBM的定义下可以被看做一个分类问题，当时OSU-PNL的Y. Wang已经开展了支持向量机(SVM)估计IBM的研究。机器学习中特征提取和目标选择十分重要，作为机器学习模型的SVM当然也不例外。Y. Wang的研究当然也包括评估众多利用听觉特性构造的手工特征对SVM进行语音增强的性能影响，在深度学习大行其道的今天，利用深度学习提取更有效的特征的思维并不难理解，而Y. Wang在2013年也的确选择这样去做。如果你对机器学习有一定了解的话，应该了解机器学习任务可以分为分类任务和回归任务两种，基于回归模型的语音增强算法也于2014年被中科大USTC SPRAT的Y. Xu等人提出。基于回归的语音增强模型直接利用神经网络建立从带噪语音谱特征到纯净语音谱特征的映射，利用网络直接生成增强的对数功率谱。如果说Y. Wang的工作是基于监督学习的CASA算法的延伸的话，Y. Xu的这项工作则更具有基于统计模型的传统语音增强算法的风格。工作中沿用了MMSE-LSA中的对数谱特征以及传统语音增强方法中人耳对相位信息不敏感的假设。然而利用分类和回归去分类这两个极具代表性的工作并不利于对后续方法的研究，因为后续的理想比值掩蔽(IRM)、相位敏感掩蔽(PSM)等训练目标也明显受到了传统语音增强统计模型的影响，这类掩蔽估计由于其定义也不再能被视为分类问题(而同样是回归问题)。这类掩蔽本质上是在设计某种意义上最优的滤波器，毕竟，回顾IBM的介绍“IBM的含义正是根据每个时频点上语音和噪声之间的能量关系，将语音能量占主导的时频点标记为1，噪声能量占主导的时频点标记为0”中“主导”二字充满了不安全感。一个极端的例子是，在极低信噪比情况下IBM会出现非常多的0值导致理想掩蔽的滤波结果的语音质量和可懂度都会十分糟糕。IRM将IBM的硬分类变成了一种软分类，每个时频点的滤波器系数定义为语音能量和语音噪声二者能量之和的比值。熟悉传统语音增强算法的读者不难发现，这个定义和频域维纳滤波算法尽管并不相同但还是十分相像。因此，Prof. Wang在综述中从训练目标角度将上述两大类工作分别定义为基于掩蔽(masking-based)和基于谱映射(mapping-based)的算法用于分类，这种分类方式得到了广泛地接受。

尽管上述内容足以概括基于深度学习的语音增强在早期阶段的历史，然而除了训练目标，模型架构的发展同样值得我们关注。不过在这一段中我们对文献中的训练目标使用掩蔽还是谱特征并不感兴趣，因此，如无必要在介绍中不会涉及映射目标。另一方面，虽然本节介绍的是深度学习语音增强的发展，然而为了保证叙述的连贯性，本段刻意忽略了算法的提出时间，只保证了相关联工作的顺序。由于语音具有时序性，因此时间建模是网络结构设计上不可忽视的问题。前面提及的方法都是利用全连接网络，考虑时序关系的方式也十分朴素——将当前帧前后几帧组合在一起形成上下文窗(context

window)后送入网络。很明显，这种方式并不足以建模长时信息。而循环神经网络(RNN)天然适合时序建模，因此2017年OSU-PNL的J. Chen和USTC-SPRAT的Sun分别开展了基于LSTM的语音增强模型的工作。这类网络和全连接网络唯一结构上的区别在于全连接网络的其中几层(通常是前几层)被替换成LSTM层，但其在增强性能、未见说话人和噪声类型的泛化能力上都明显优于全连接网络。RNNs的提出使得目前基于深度学习的语音增强模型几乎全部放弃了全连接网络。另一个需要提及的内容在于卷积神经网络的应用，熟悉图像分割的读者应该已经感受到了语音增强与图像分割的相似之处，因此卷积神经网络的引入也十分自然。不过在介绍卷积神经网络在语音增强的应用之前，有必要至少提一点语音增强与图像分割的区别，尽管我们可以将语音特征看成一张图片，但是由于语音增强有很多实时应用场景，因此在这张“图片”时间维度的处理需要十分谨慎“实时陷阱”。卷积层由于“局部连接、权重共享”的特点相比全连接层和循环层拥有更少的参数，2015年清华SATLab的L. Hui提出的maxout卷积网络首次在语音增强领域性能超过全连接网络。而后中央研究院Bio-ASP Lab的S. Fu和卡内基梅隆大学的S. R. Park分别将图像领域常用的普通卷积神经网络和卷积编解码器结构成功应用到了语音增强中。由于卷积编解码器结构对于时频点级预测任务的天然适配，基于卷积编解码器的语音增强模型逐渐成为主流。自然地，将卷积层的特征提取能力和循环层的时序建模能力结合的工作应运而生。2018年H. Zhao在微软亚研完成了卷积层特征提取、LSTM时序建模、最后由全连接层进行幅度谱估计的EHNet，同年OSU-PNL的K. Tan将LSTM插入卷积编解码器之间提出一种卷积循环网络(CRN)。除了RNN适合序列建模外，时序卷积网络(TCN)利用一维空洞卷积实现了并行时序建模过程。2018年K. Tan引入门控(gating)TCN取代RNN构造了一种全卷积增强网络。

随着基于深度学习的幅度谱估计算法性能的快速发展，两个很有意思的现象出现了。一是相对简短可以介绍完的，是深度学习反求诸传统，深度学习作为一种参数估计器为传统语音增强中的MMSE、MMSE-LSA和卡尔曼滤波等算法提供依赖先验假设更少且更为准确参数估计。一般将这类深度学习与传统算法结合的方法称为混合式模型，然而混合式模型类工作相比性能提升更重要的贡献在于让基于深度学习的语音增强渐渐褪去纯机器学习或深度学习意味的工作，而逐渐带有语音信号处理的特色。当然，相比于上述这种“主动”结合语音特色，另一种相对“被动”的结合方式产生于另一个有意思的现象——基于深度学习语音增强的性能第一次遇到了瓶颈。当然人们很快意识到了问题——这一性能限制是只进行幅度谱估计带来的，相位恢复似乎对人耳感知是重要的。以至于解决该问题的方案出现得如此之快在今天看来似乎并没有瓶颈的意味在。在相位恢复重要性思潮下，用于同时进行幅度估计和相位恢复的方案大方向上可以分为时域的方案和谱域的方案。不论是哪种方案他们都是为了解决由于相位谱没有清晰的模式结构导致其不易优化的问题。首先介绍时域模型，原因依旧是因为介绍起来可以比较简略。时域模型的兴起主要由于其在语音分离领域大获成功，以一种波形到波形的映射方式提供了一种解决相位谱不易优化问题的方案——放弃建模相位。不过也许是由于噪声模式的复杂性，抑或是房间冲激响应的敏感多变，时域模型在噪声混响场景下的性能和鲁棒性使其在语音增强中并不能成为主流。网络结构上和之前的幅度谱架构基本还是类似的，有卷积编解码器结构也有卷积循环(或TCN)网络，只不过对应的卷积层大多用一维卷积代替了二

维卷积(当然也同样有使用二维卷积作为编解码器的架构)。相比之下，谱域的方案解决相位估计的方式则略显复杂。众所周知，复数可以由幅度相位表示也可以由实部虚部表示。其中实部虚部表示和之前幅度谱模型一样，主要分为基于掩蔽和基于谱映射的方式。2016年OHU-PNL的Williamson提出了一种复数比值掩蔽(cIRM)，分别估计实部和虚部的掩蔽；而2017年S. Fu提出了一种谱映射的CNN，2019年K. Tan将CRN应用到复数谱映射直接利用网络估计增强语音的实部和虚部。2019年首尔国立大学的H. S. Choi提出了一种基于极坐标系的复值掩蔽策略(pcRM)，通过网络估计的cRM得到有界的幅度掩蔽和无界的相位掩蔽。尽管这篇文章成功应用了复值网络并且西工大ASLP的Y. Hu继复值U-Net发展为复值CRN取得了令人瞩目的性能，然而本篇文章对于在不同损失函数情况下的几种典型复值掩蔽的分析可能更加有趣：尽管这类复数目标算法都宣称致力于相位的恢复，然而其带来的性能提升究竟是源于相位的恢复还是噪声成分幅值的抑制(相位可能调整的极少)值得探究，D. Yin等人同样通过实验发现了使用cRM虚部几乎为0的问题。而为了体现相位的修复，基于幅度相位表示的方案得到了考虑，其中包括采用双分支结构同时估计幅度和相位，通过多任务学习的方式达到相位恢复的方法；也有一些算法采用了更具结构性的相位目标与幅度谱同时和先后估计，比如OHU-PNL的Z. Wang提出的群时延(GD)以及西工大CIAIC的N. Zheng提出的瞬时频率倒数(IFD)。

尽管Z. Wang的分析要晚于接下来要介绍的一些工作，然而在此处提及也许是必要。尽管这篇分析的重点在于解释RI+Mag损失函数为何是有效的，但其提供的幅度估计和相位估计的补偿作用无疑为解耦风格工作有效性提供了一种合理解释。所谓幅度和相位估计的补偿作用，是指在迫使网络输出逼近理想复数谱时，幅度谱估计的准确度会大打折扣。而如Griffin-Lim相位重构的很多相位估计算法都在幅度谱估计的基础上进行。一个解决幅度相位估计补偿作用的想法是，将复数谱的优化问题解耦为幅度初估计和包括相位信息在内的“全信息”估计两部分。尽管这仍属于复数谱估计的一类方法，然而这类算法在近年来备受关注且性能优异，因此有必要单独作为成段介绍。2020年，奥尔登堡大学通信声学的N. L. Westhausen先后利用两次信号变换——首先利用STFT幅度谱上进行幅度估计，而后再利用一维卷积生成的可学习的分析/合成基函数直接将语音时域波形映射到高维空间进行语音和噪声成分的分离，利用时域模型的方式完成相位恢复。哈工大SPLab的Z. Du在Mel谱(一种符合人耳听觉特性的幅度谱变换)域进行降噪，而后利用语音合成声码器根据增强的Mel谱合成语音波形从而实现相位信息的引入。中科院声学所IACAS-Lab9的A. Li则在2021年提出一种完全在STFT域上先后完成幅度谱估计和复数谱残差修正的框架，利用复数谱残差修复相位信息。完全在STFT域上处理两部分任务的一个好处是，原本级联的两个子任务出现了并行优化的可能，2022年A. Li和西工大ASLP的Y. Fu分别提出了幅度估计和复数谱修正的两种并行架构。其实，从复数谱的角度考虑，这种解耦是由于幅度相位估计的补偿作用导致幅度谱估计不准确造成的，因此，除了首先约束幅度谱后再根据相位差微调增强结果，也可以先得到复数谱结果后补偿不准确的幅度信息。于是在2022年，北交大IIS的T. Wang则先进行复数谱估计再利用谐波特征进行幅度谱修正的方法。而解耦风格的工作也并非只由从训练目标解耦一种方式。从语音被噪声污染的过程角度，早在2016年USTC-SPRAT的T. Gao就提出以不同信噪比的被污染语音作为目标

将一次降噪过程分解为渐进地、多阶段地、逐信噪比地提升。从语谱高低频特性差异角度，搜狗的J. Li对语谱进行子带分解，多阶段地处理各个子带。而对语音特性进行分析，语音可以分解为包络和周期两部分，2018年，J.-M. Valin首先利用GRU估计Bark域(另一种符合人耳听觉特性的幅度谱变换)的增益因子，而后利用信号处理的方式进行基频滤波抑制谐波间噪声残留，而后在2020年，其又对模型中各部分进行进一步提升，以可观的复杂度达到超过众多复数谱模型的性能。而对该工作神经网络化最杰出的代表当属PAU模式识别实验室的H. Schröter提出的在估计包络增益因子外利用网络估计深度滤波器(deep filtering)系数的框架。除此之外，语音增强任务也可以理解为噪声去除和语音恢复两部分，因为单通道语音增强具有噪声抑制和语音失真之间的折衷问题，彻底去除噪声势必会对语音造成严重损伤。因此，一种先进行激进地噪声抑制而后再恢复在降噪过程中的失真语音的框架首先由TU Braunschweig通信技术所信号与机器学习组的M. Strake于2019年提出，2020年内蒙古大学IMUAI的X. Hao在该思路的基础上引入计算机视觉的概念形象地将该过程命名为“masking and inpainting”即将所有噪声的时频点去除后根据语谱相关性修补被破坏的语音时频点。而传统的谱减法则从带噪语音的合成过程反推，将语音增强看做噪声估计和移除噪声成分的过程。2020年，哥伦比亚大学C2G2的R. Xu受传统谱减法的启发将降噪过程分解为噪声的“masking and inpainting”过程(利用VAD得到静音段而后通过静音段推测出完整的噪声谱)并将噪声信息和带噪语音信息一同作为输入完成降噪。2021年，IACAS-Lab9的W. Liu将基于训练目标解耦的思路与基于带噪语音合成的解耦思路结合首先利用多任务学习幅度域对语音降噪并直接估计噪声成分，而后修正复数谱恢复过度抑制的语音。需要注意的是，尽管本段从解耦角度介绍了以上算法，但是这些算法在提出时受到的启发性工作和动机可能并非如此。这种分类目前也并不作为一种共识，而更多的是一家之言。

需要注意的是，网络架构和学习目标虽然的研究虽然更多然而只是性能提升的一方面，更长时间范围的输入特征以及精心设计的损失函数对模型性能的提升可能更加有益，其中损失函数由于不会在推理过程中引入额外的时延和复杂度而备受关注。一个有力的证据是在ICASSP2021 DNS-Challenge中微软提供的结合一种幂律压缩的复数谱均方误差损失的复数掩蔽网络相比包括解耦风格框架在一众高参数量和计算量方案仍表现出最优的性能。损失函数包括基于距离的损失函数、基于能量比的损失函数和基于感知指标的损失函数等。常见的距离表征最常用的莫过于基于范数的损失函数 L_p 损失，即平均绝对误差(MAE)和均方误差(MSE)，这两类误差都被用于度量时域波形、掩蔽、频谱以及嵌入式特征。时域波形的 L_p 度量形式相对简单，表示为 $\mathcal{L}_{time-L_p} = \|s - \hat{s}\|_p$ ，有时会加入对噪声项 L_p 损失作为正则，以提升语音估计性能，有文章表示，对于时域样本而言MAE相比MSE更关注小信号并具有更强的噪声抑制能力。除了IBM有使用交叉熵损失进行度量外，其余掩蔽大多都会采用 L_p 损失直接计算掩蔽间的误差 $\mathcal{L}_{mask} = \|M - \hat{M}\|_p$ ，J. Valin提出了幂律压缩的掩蔽MSE损失 $\mathcal{L}_{RNNoise} = \|M^c - \hat{M}^c\|_2$ ，利用指数项 c 控制噪声抑制的激进程度。然而，目前普遍认为基于信号近似(SA)技术的损失相比直接度量掩蔽间差异对语音增强的性能更有益： $\mathcal{L}_{SA} = \|S - \hat{M} \cdot Y\|_p$ 。可以看出，此时对掩蔽的范数损失已经和基于频谱的范数损失基本一致，只需将滤波项 $\hat{M} \cdot Y$ 改为网

络映射的谱 \hat{S} : $\mathcal{L}_{spec-L_p} = \|S - \hat{S}\|_p$ 。当然 S 和 \hat{S} 可以是幅度谱或其变换(如对数谱、Mel谱等)，也可以是复数谱。由于幅度谱和复数谱是如此常用，这里我们分别给出其具体形式: $\mathcal{L}_{Mag} = \mathbb{E}[|A - \hat{A}|^p]$ 和 $\mathcal{L}_{com} = \mathbb{E}[|\mathcal{R}(X) - X(\hat{X})|^p + |\mathcal{I}(X) - \mathcal{I}(\hat{X})|^p]$ (其中 A 和 X 分别代表幅度谱和复谱)。 \mathcal{L}_{spec-L_p} 还可能根据如AMR编码等心理声学机制进入加权，以期望其更符合人耳听觉，而复数谱范数损失又常与幅度损失结合以缓解幅度相位估计的补偿作用，该损失被称为RI+Mag损失: $\mathcal{L}_{RI+Mag} = \alpha\mathcal{L}_{Mag} + \beta\mathcal{L}_{com}$ 。在此基础上，与时域波形 L_p 损失结合的 $\mathcal{L}_{time-spec-L_p} = \mathcal{L}_{RI+Mag} + \mathcal{L}_{time-L_p}$ 、基于幂律压缩的RI+Mag损失 $\mathcal{L}_{cprs} = \alpha\mathbb{E}[|A^c - \hat{A}^c|^2] + \beta\mathbb{E}[|\frac{X \cdot A^c}{A} - \frac{\hat{X} \cdot \hat{A}^c}{\hat{A}}|^2]$ 以及多分辨率(假设有 I 种STFT点数对应的 I 个分辨率)的 \mathcal{L}_{cprs} 函数 $\mathcal{L}_{MR-STFT} = \sum_i \mathcal{L}_{cprs}^i, i = 1, \dots, I$ 都成为目前广泛使用的 L_p 损失。另外由于 \mathcal{L}_{cprs} 对噪声的抑制能力较强导致语音失真较多，一种非对称损失有时也被作为正则项 $\mathcal{L}_{asym} = \mathbb{E}[|\max(A^c - \hat{A}^c, 0)|^2]$ 。可以看出，尽管有文献提及由于STFT频点分布更类似拉普拉斯分布因此建议使用 \mathcal{L}_1 范数损失，然而大多数工作更喜欢选择 \mathcal{L}_2 范数损失。 L_p 范数损失最后也见于对嵌入式特征的距离度量，利用wav2vec、PANN等预训练模型生成嵌入式特征之间的距离作为谱距离损失的正则项。此外，一些除了 \mathcal{L}_p 范数作为距离度量，KL散度也有一些工作将其与 \mathcal{L}_p 范数联合度量谱距离: $\mathcal{L}_{KL} = \mathbb{E}[\hat{X} \cdot \log(\frac{\hat{X}}{X})]$ 。基于信号能量比的损失函数主要用于时域波形，常见的有SDR(及其Log形式)和SI-SDR损失，分别定义为 $\mathcal{L}_{SDR} = -10\log(\frac{|s|^2}{|\hat{s}-s|^2})$ 和 $\mathcal{L}_{SI-SDR} = -10\log(\frac{|\xi \cdot \hat{s}|^2}{|s-\xi \cdot \hat{s}|^2})$ ($\xi = \frac{s^T \hat{s}}{s^T s}$)。尽管SI-SDR被广泛应用于语音分离当中，但是由于其过多的语音失真导致最近的工作常将其与 L_p 范数谱距离损失联合使用。上述特征常被诟病并不能反映人耳的听感，因此，一些感知指标，如PESQ、ESTOI和CD，一度被引入作为损失函数训练模型。然而这类损失函数仅能提升被优化指标的性能，并没有带来其余指标的提升。要知道，目前的客观指标与人类的主观听感并不完全一致，单纯提升某些客观指标大多也并未带来主观听感的明显提升。于是近年来，这类感知指标类损失更多是以一种联合形式作为验证集损失被用于选择最优的模型。

尽管生成对抗网络(GAN)常作为一种不同的网络架构，然而这里我们更希望将其作为一种训练手段，即对抗性训练。众所周知，GAN由生成器和鉴别器两部分，在GAN在语音增强的使用中，其生成器可以用前文提到的众多网络代替，而之前提到的各种损失同样可以作为生成器损失函数的其中一项或等价于生成器损失函数。因此，GAN在语音增强中目前表现的作用，更类似于一种类损失函数”用于帮助网络专注于那些原本损失函数难以强调的噪声成分和语音伪影(artifacts)部分。于是，一言以蔽之GAN在语音增强的整体发展：之前的语音增强模型(如LSTM、UNet等)代入生成器，各式GAN的发展技术(如LSGAN、Wasserstein GAN、Relativistic GAN、Conditional GAN和Geometric GAN等)代入鉴别器。如此导致GAN始终并未在语音增强领域至少在实时语音增强领域成为主流技术，这很可能与语音增强任务与图像生成、语音合成等生成类任务的差异过大有关，然而，在无监督或自监督情况下，包括GAN在内的生成式模型(还包括变分自动编码VAE等)也许是值得研究的方法。

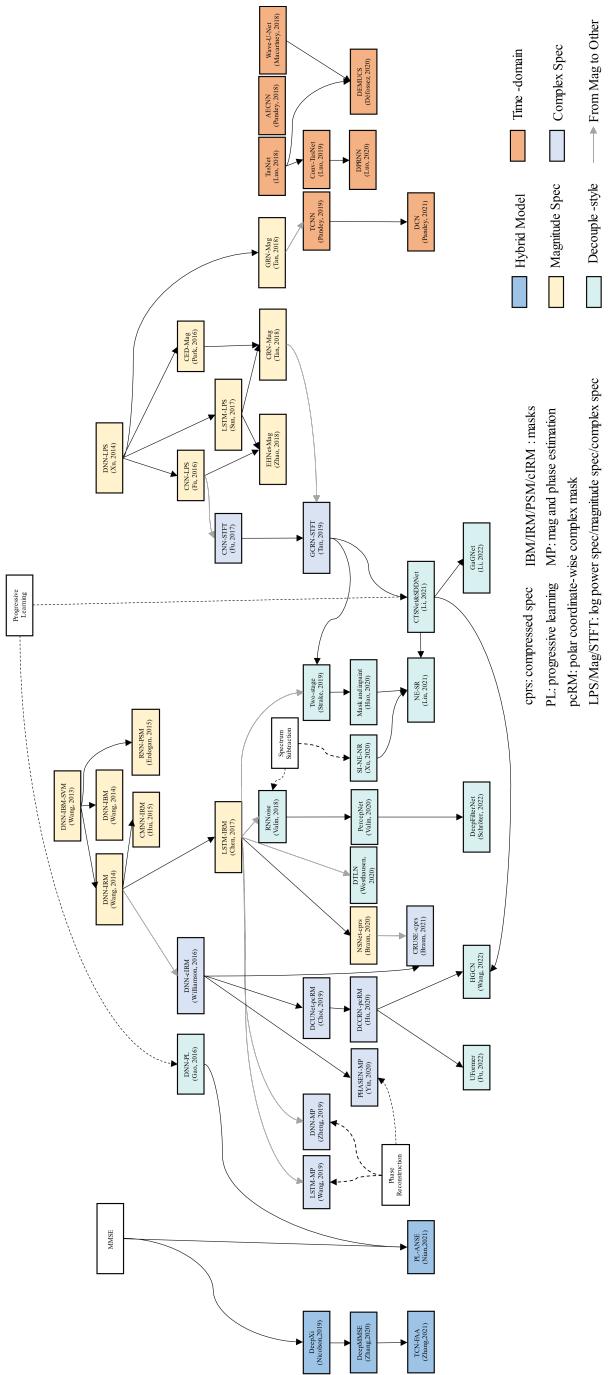


Fig. 0.1 深度学习语音增强算法发展

图0.1梳理了深度学习语音增强发展的部分脉络，然而对于目前处于发展之初的无监督语音增强、超宽带语音增强、多模态语音增强、个性化语音增强等并未涉及，去混响等问题也未讨论。

Chapter 1

基于幅度谱的语音增强

由于人耳对于语音的幅度相比其相位更加敏感，传统的语音增强算法大部分都集中于在幅度谱上进行处理。于是，深度学习最早被引入语音增强时也在幅度谱上最先应用。基于幅度谱的语音增强过程大抵可以按照图1.1所示的流程。混合语音，又常称作带噪语音，通过傅里叶变换基(或基于人耳听觉的滤波器组)将时域信号分解成某种时频表征，而后提取其幅度域特征，如短时傅里叶变换的幅度谱、对数功率谱等。通过纯净语音(有时还需要带噪信号)可以计算得到网络的映射目标(图中的分离目标)，常见的是时频掩蔽或与特征提取对应的谱特征。提取的特征和预测目标组成输入输出对用于对网络模型(图中的分离模型)进行训练，建立从输入特征到输出目标之间的映射。训练好的模型在测试阶段根据混合语音的特征预测出目标的估计值。预测目标(有时还需结合混合信号)通过时频分解的逆变换即可得到增强的信号波形。

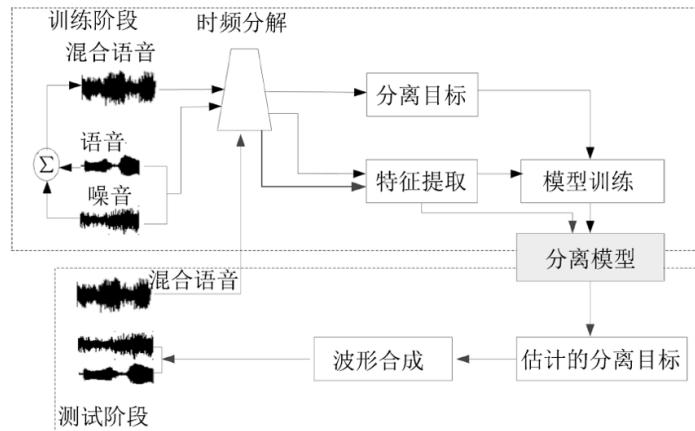


Fig. 1.1 基于幅度谱的语音增强框图[1]

基于幅度谱的语音增强方法通常被分为基于掩蔽(masking-based)和基于谱映射(mapping-based)两大类。基于掩蔽的语音增强方法利用网络预测出一个滤波器(即时频掩蔽，简称掩蔽)对幅度谱进行滤波，而后将滤波后的幅度谱与带噪相位耦合并重构回时域波形；而基于谱映射的语音增强方法利用神经网络直接建立从带噪谱特征到纯净谱特征之间的映射，而后将谱特征与带噪相位耦合得到增强的语音。这两类方法的框图如图1.2所示，可以看出，二者的区别主要在于是否有显式的滤波过程(橙色部分)。

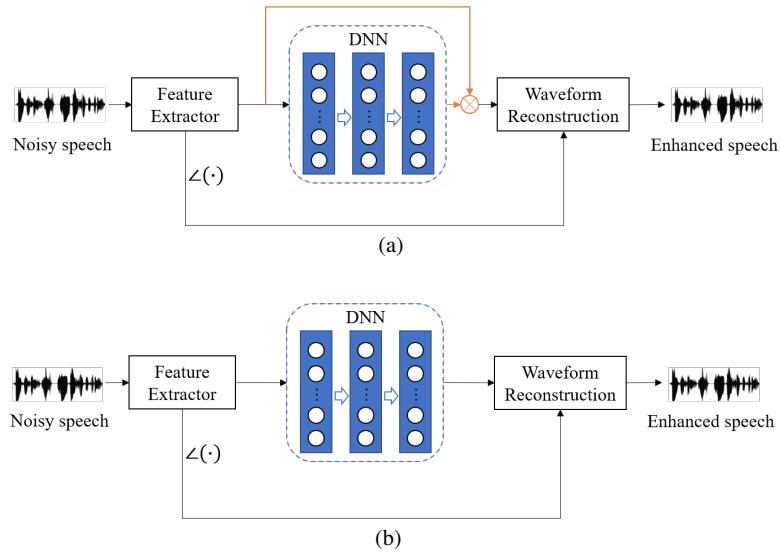


Fig. 1.2 (a) 基于掩蔽的语音增强处理流程, (b) 基于谱映射的语音增强处理过程。

基于掩蔽的方法是估计滤波器并对带噪语谱滤波，与传统的维纳/MMSE等语音增强算法类似，这类算法估计的滤波器的范围大多在 $[0,1]$ ，因此都对幅度谱起抑制作用。这不可避免地在语音能量弱但存在强噪声的区域上形成不连续的黑洞(black holes)，这种黑洞常出现在谐波之间(图1.3(a))、中高频谐波、以及轻音(图1.3(b))，极其影响语音质量和可懂度；而基于谱映射的方法可以理解为“脑补”，算法的作用可以看成生成或者再生，凡是噪声存在的时频点均有可能完全抹去后重生出新的、结构类似的频谱，不过也正是由于谱映射的生成机制，伪影(artifacts)(图1.3(b))、过平滑(over-smoothing)问题和鲁棒性问题在该类方法出现早期常令人诟病。不过随着两类算法后来的发展，上述问题得到缓解。

我们将图1.3(a)或(b)展示的语音的时频关系图称为语谱，语谱中的每个点被称为时频点(time-frequency bin)。

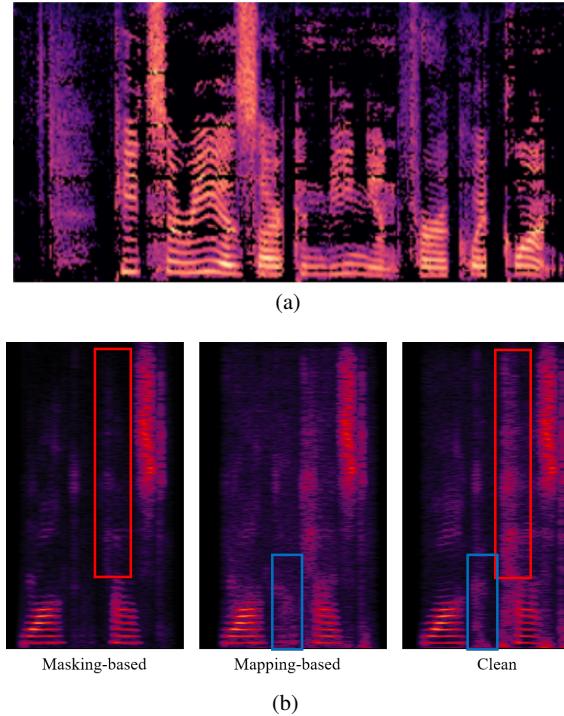


Fig. 1.3 (a) 频谱黑洞[4], (b) 利用基于掩蔽和基于谱映射方法处理的语音, 基于掩蔽的方法的清音被吞掉(见红框), 基于谱映射的方法在无谐波部分“脑补”出谐波(见蓝框)。

需要注意的是, 上述分类方法是以训练目标作为分类依据的。由于语音增强从应用角度可分为面向人耳听感的和面向机器感知的。前者的目的在于提高人耳在通信过程中的语音质量和可懂度, 不舒适的噪声成分的抑制是其处理的重点, 而不易感知的语音失真是能够容忍的; 而后者往往对语音失真更加敏感, 而可以容忍一定的噪声残留。

基于幅度谱的语音增强方法常采用两类特征: 幅度谱及其基于听觉感知的相关变换。前者直接采用短时傅里叶变换的幅度谱或者其对数/幂律压缩形式; 后者则将幅度谱变换到ERB/Bark/Mel/Gammatone等听觉感知域, 有时也会引入线性预测/基音预测/倒谱变换等其他语音相关参数得到一组听觉感知相关系数作为输入特征。随着深度学习模型的快速发展, 神经网络通过部分非线性变换取代了手工特征的提取过程, 使得对后者的研究日趋减少(尽管超宽带/全带语音增强中可能也会出现ERB/Mel等特征的利用)。因此, 我们不再对特征方面展开介绍, 如果读者希望深入了解, 可以参看[2]。

基于幅度谱的语音增强模型于2013年由Wang等人首次提出, 他们将全连接网络应用于语音增强领域, 网络作为子带分类器估计理想二值掩蔽(IBM, 详见1.1)。2014年, Xu等人证明了全连接网络能够通过直接建立带噪语音到纯净语音对数功率谱之间的映射实现语音增强。需要注意的是, 这一时期的工作由于受到早期深度学习发展的限制保留了明显的时代特色, 诸如使用受

限玻尔兹曼机预训练初始化网络权重、对输入特征进行时序平滑滤波、对大量手工提取特征的详细研究、以及一部分应对噪声、信噪比和说话人泛化问题的技术探索等，这些技术或被新兴的深度学习技术取代或被优化。同时，为了让网络感知语谱的时序信息，这一时期的输入特征常将当前帧为中心的前后多帧(context window, 上下文窗)拼接而后送入网络，这极大地增加了输入特征的维度，但不能很好地表征语谱的时序关系。因此，适合时序建模的递归神经网络(RNN)尤其是长短时记忆网络(LSTM)被引入到面向语音识别的语音增强中，Chen等人将LSTM应用在面向听感的语音增强任务中并展示了LSTM对提升未见噪声和说话人泛化能力的作用，64通道的语谱被送入四层LSTM后预测得到理想比值掩蔽(IRM)，改善了原本全连接网络的泛化问题(由于历史惯性，LSTM乃至卷积神经网络CNN在早期也仍有很多工作输入上下文窗作为特征)。在1.1中，基于时频掩蔽的幅度谱语音增强方法将以RNN的另一种形式——门控递归单元(GRU)网络为例进行介绍。此外，同时估计时频掩蔽和谱信息后加权融合的多目标训练(multi-target training)同样是这一时期的常见方法，其期望结合两种训练目标各自的优点或为了处理平稳/瞬态等不同特性的噪声类型。通过图1.3可以发现，幅度谱具有清晰的时频模式，全连接和LSTM都没有很好的建模这种时频相关性。因此，Hui等人引入CNN估计IRM，在此基础上Fu等人利用多任务学习同时估计纯净对数功率谱和SNR从而使网络隐式地关注SNR失配问题，模型由三个卷积层和两层全连接层组成，该工作的一个重要贡献是实验证明了图像领域常用的池化层由于会丢失细节并不利于语音增强任务，目前语音增强中的卷积层已普遍不再级联池化层。一个自然的想法是结合CNN对时频相关性的建模和RNN的时序建模能力，于是卷积递归网络(CRN)被提出。常见的两种组合方式分别是卷积层组成的编码器和递归层组成的时序建模模块级联后再经过全连接层预测时频掩蔽或谱特征(如2018年的EHNet)，和在由卷积层和反卷积层分别作为编码器(encoder)和解码器(decoder)组成的一个对称的卷积编解码器(Convolutional Encoder-Decoder, CED)之间插入递归层组成的时序建模模块的“编码器-时序建模-解码器”架构(如2018年的CRN)。后者对后续工作的影响更大，因此在1.2中选择了一个基于CRN的谱映射语音增强算法进行介绍。除了上述工作之外，应当注意的是这一时期多任务学习的方法已经被引入，在估计时频掩蔽或谱特性的同时，信噪比、基频 f_0 、语音活动检测(Voice Activity Detection, VAD)等与语音增强/分离任务相关的参数也被有选择地同时被估计，以控制模型的泛化性或用于后处理，从而提升增强语音的质量。

1.1 基于掩蔽的语音增强算法

1.1.1 处理流程

为了更好地解释算法处理流程，我们选用NSNet作为基于掩蔽的语音增强算法的例子进行说明。

NSNet的框图如图1.4所示，我们不打算阐述框图的所有部分，因为这里涉及一些后续章节涉及的内容。请将图1.2(a)和图1.4对应起来，图1.4虚

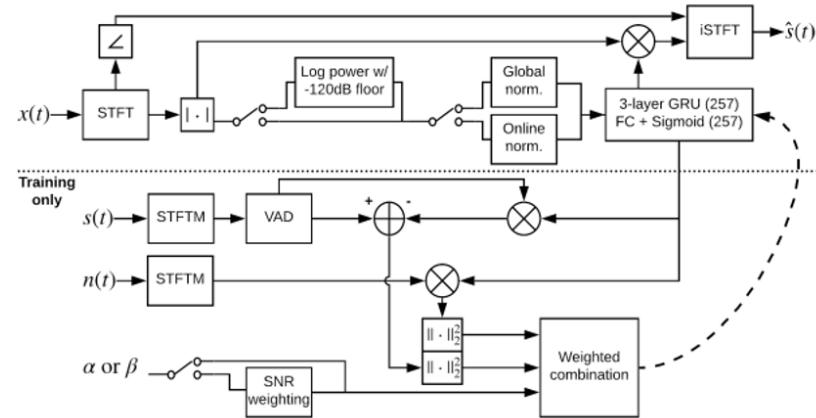


Fig. 1.4 NSNet框图[3]

线以上的部分，从左至右， $x(t)$ 代表图1.2(a)中的带噪语音(noisy speech)，自STFT起至网络("3-layer GRU (257) FC + Sigmoid (257)")输入止的那部分是特征提取器(feature extractor)，注意，图1.2(a)中的特征提取器有两个输出，一个送入网络(DNN)，另一个代表相位分量用于波形重构。请观察图1.4，同样可以发现相位从特征提取器中的STFT模块获得并作为两个输出其中之一传送给iSTFT模块。当然，iSTFT对应的就是图1.2(a)中的波形重构(waveform reconstruction)。iSTFT的另一个输入是被网络输出结果滤波的谱特征。而 $\hat{s}(t)$ 则对应了图1.2(a)中的增强语音(enhanched speech)。至于虚线以下的部分代表的是损失函数的计算过程，与虚线以上的联系是图中唯一的虚线箭头，用于表示通过损失函数计算得到的梯度的反向传播，以更新网络参数。总结一下，基于掩蔽的语音增强方法首先将带噪语音经过特征提取得到幅度特征并解耦得到带噪语音的相位，而后幅度特征经过网络预测得到时频掩蔽，时频掩蔽对幅度谱(或幅度谱相关特征)进行滤波得到增强的幅度谱(或幅度谱相关特征)。而后增强的幅度与带噪语音的相位结合重构得到增强的语音信号。NSNet采用了标准化(normalize，见6)的对数功率谱作为特征，而网络输出的是幅度谱的时频掩蔽。

输入特征不仅可以是对数功率谱，幅度谱、Mel谱和Bark域谱等都可作为输入特征。这些输入特征的选用可能受启发于其他语音相关方向的积累，比如对数谱、Mel谱和Bark域谱分别是传统语音增强MMSE-LSA、语音识别和音频编码领域常用的特征。

```

1 # 特征提取部分 wav_dataset.py lines 51-109
2 x_stft = torch.stft(noisy_waveform.view(-1), ...)
3 x_ps = x_stft.pow(2).sum(-1)
4 x_lps = LogTransform()(x_ps)
5 x_ms = x_ps.sqrt()

```

```

6 ...
7 for frame_counter, frame_feature in enumerate(x_lps):
8     ...
9     norm_feature = (frame_feature - mu) / sigma
10    frames.append(norm_feature)
11    x_lps = torch.stack(frames, dim=0)
12 ...

```

可以看到，NSNet的特征提取过程是将带噪语音经过STFT后，计算对数功率谱 x_lps ，将其进行标准化后作为网络的输入特征；幅度谱 x_ms 则用于与网络估计的时频掩蔽滤波得到增强的幅度谱。

网络部分则很简单，由三层GRU和一层全连接层级联而成，最后的激活函数采用Sigmoid函数以保证网络的输出在[0,1]之间，这是由于IRM的范围即是如此(然而有很多工作表明将激活函数设为无界的ReLU可能带来性能的提升)。输入后的permute()操作是为了使变量维度变为(batch size, num frames, num bins)，使GRU和全连接层对频率维进行操作。

```

1 # 网络部分 nsnet_model.py lines 37-53
2 def __build_model(self):
3     self.gru = nn.GRU(...)
4     self.dense = nn.Linear(...)
5
6     def forward(self, x):
7         x = x.permute(0, 2, 1)
8         x, _ = self.gru(x)
9         x = torch.sigmoid(self.dense(x))
10        x = x.permute(0, 2, 1)
11
12    return x

```

在训练过程中图1.4(也就是虚线下面的部分)，是作者提出的一种损失函数设计方式，同时计算了语音失真和噪声抑制：

$$\mathcal{L} = \alpha \mathbb{E}(\|S - \hat{M} \cdot S\|_2^2) + (1 - \alpha) \mathbb{E}(\|\hat{M} \cdot N\|_2^2), \quad (1.1)$$

语音失真项损失是使用纯净语音幅度谱与掩蔽的哈达玛积作为估计项的，有的工作则采用带噪幅度谱与掩蔽的积作为估计项，称为信号近似(signal approximation, SA)形式，即 $\mathcal{L} = \mathbb{E}(\|S - \hat{M} \cdot X\|_2^2)$ 。式1.1对应的代码为：

```

1 # 损失部分 nsnet_model.py lines 57-71
2 def loss(self, target, prediction):
3     loss = F.mse_loss(prediction, target)
4     return loss
5
6 def training_step(self, batch, batch_idx):
7     ...
8     y_hat = self.forward(x_lps)
9     loss_speech = self.loss(y_ms [...], (y_hat * y_ms)[...])
10    loss_noise = self.loss(torch.zeros_like(y_hat), y_hat *
11                           noise_ms)
12    loss_val = self.alpha * loss_speech + (1 - self.alpha) *
13               loss_noise
14 ...

```

此外，只计算掩蔽之间距离的掩蔽近似(mask approximation, MA)形式同样是基于掩蔽方法的常见损失函数，比如nngev的损失函数就是计算语音(噪声)掩蔽之间的交叉熵损失(阅读nngev源码时请注意论文采用的是非因果网络BLSTM)：

```

1 # nn_models.py lines 17–22
2 def train_and_cv(self, Y, IBM_N, IBM_X, dropout=0.):
3     N_mask_hat, X_mask_hat = self._propagate(Y, dropout)
4     loss_X = binary_cross_entropy(X_mask_hat, IBM_X)
5     loss_N = binary_cross_entropy(N_mask_hat, IBM_N)
6     loss = (loss_X + loss_N) / 2
7     return loss

```

由于采用SA形式的损失函数，因此时频掩蔽并不需要显式定义。而MA作为损失函数则需要显式计算出目标时频掩蔽，掩蔽的相关内容将在下一小节介绍。

接下来关注NSNet项目的推理过程，也就是测试阶段的代码。

```

1 # test_nn.py lines 27–34
2 gain_mask = model(x_lps)
3 y_spectrogram_hat = x_ms * gain_mask
4 y_stft_hat = torch.stack([y_spectrogram_hat * torch.cos(angle(
    x_stft)), y_spectrogram_hat * torch.sin(angle(x_stft))], dim
    =-1)
5
6 y_waveform_hat = istft(y_stft_hat, ...)

```

可以看到，基于掩蔽的语音增强算法在测试阶段模型将提取的带噪对数功率谱特征 x_{lps} 作为输入并预测得到掩蔽 $gain_mask$ ，而掩蔽与带噪幅度谱 x_{ms} 进行哈达玛积得到滤波后的幅度谱 $y_spectrogram_hat$ 。根据 $S_r = |S| \cdot \angle S$ 和 $S_i = |S| \cdot \angle S$ 的幅度相位-实部虚部关系将增强的幅度谱和带噪的相位耦合得到复数谱，并经过iSTFT重构回波形 $y_waveform_hat$ 。

1.1.2 用于幅度谱的时频掩蔽

时频掩蔽无疑是基于掩蔽的语音增强算法的重点研究对象。最早提出的时频掩蔽是理想二值掩蔽(Ideal Binary Mask, IBM)，根据语音时频分布的稀疏性，即语音成分在语谱中只分布于少数时频点，因此每个时频点上的语音和噪声之间的能量差异通常较大，而当某个频率语音能量显著强于噪声能量时，噪声会被掩蔽(mask effect)。于是可以将语音增强问题简单地看做将语音能量占主导的时频点筛选出来的过程。二值掩蔽要做的，就是标记出那些语音占主导的时频点；而网络要做的，就是学习一个分类器，判断每个时频点是语音占主导还是噪声占主导。IBM被定义为：

$$M_{IBM}(t, f) = \begin{cases} 1, & \text{if } SNR(t, f) \geq T \\ 0, & \text{otherwise} \end{cases} \quad (1.2)$$

IBM的代码来自speech-feature-extractor:

```
1 # speech_utils.py lines 114–120
2 noise_spect = stft_extractor(noisy_speech-clean_speech, ...)
3 clean_spect = stft_extractor(clean_speech, ...)
4 ibm = np.where(10.*np.log10(clean_spect/noise_spect)>=local_snr
    , 1., 0.)
```

理想比值掩蔽(Ideal Ratio Mask, IRM)用一个0到1的实值代替了IBM的非0即1，表征了时频点上语音能量和带噪语音能量的关系，这里假设语音和噪声不相关：

$$M_{IRM}(t, f) = \left(\frac{|S(t, f)|^2}{|S(t, f)|^2 + |N(t, f)|^2} \right)^{\beta} \quad (1.3)$$

式中 β 一般取0.5。可以看出，IRM的定义形式上与维纳滤波器完全一致，但维纳滤波器是基于统计的线性滤波器，式中的功率值是求期望的结果。代码同样来自speech-feature-extractor:

```
1 # speech_utils.py lines 122–129
2 noise_spect = stft_extractor(noisy_speech-clean_speech, ...)
3 clean_spect = stft_extractor(clean_speech, ...)
4 mask = np.sqrt(np.square(np.abs(clean_spect)) / (np.square(np.abs
    (noise_spect)) + np.square(np.abs(clean_spect))))
```

此外，还有幅度谱掩蔽(Spectral Magnitude Mask, SMM)。相比IRM，SMM在幅度上而非能量上进行定义，并且也没有进行语音和噪声不相关的假设。这是由于根据IRM的定义，在测试过程中的滤波过程 $\hat{M}_{IRM} \cdot |X| \neq |S|$ 。通过放弃语音和噪声不相关的假设，将SMM定义为 $M_{SMM} = \frac{|S|}{|X|}$ ，其滤波后的结果是幅度谱上的最优滤波器估计，然而，这也导致了SMM的取值范围在 $[0, +\infty)$ 之间。早期为了使训练目标有界，SMM的最大值常被截断为1或2(代码中截断至2)。

```
1 # speech_utils.py line 10 and lines 130–136
2 EPSILON = np.finfo(np.float32).eps
3 ...
4 noisy_spect = stft_extractor(noisy_speech, ...)
5 clean_spect = stft_extractor(clean_speech, ...)
6 mask = np.abs(clean_spect) / (np.abs(noisy_spect) + EPSILON)
7 ...
8 mask = np.where(mask > 2., 2., mask)
```

相位敏感滤波器(Phase-sensitive mask, PSM)则是复数域上的最优的实值滤波器，定义为：

$$M_{PSM} = \frac{|S|}{|X|} \cos(\angle S - \angle X), \quad (1.4)$$

可以看出，PSM的范围是 $(-\infty, +\infty)$ 。和SMM一样，PSM也常被截断以方便训练，常用的截断范围为 $[0, 1]$ 。若采用SA的方式训练，损失函数定义为：

$$\mathcal{L} = \mathbb{E}\{(\hat{M}_{PSM} \cdot |X| - |S| \cos(\angle S - \angle X))^2\}, \quad (1.5)$$

而解码时和其他掩蔽一样直接作用于带噪幅度谱后与带噪相位耦合得到增强的谱。

PSM的代码选自espnet，代码中的 r 是纯净语音的复谱，利用 $\angle S = \frac{S}{|S|}$ 的方式得到相位，并利用三角函数关系式得到相位差的余弦值：

```

1 # espnet2/enh/loss/criterions/tf_domain.py lines 63-73
2 phase_r = r / (abs(r) + EPS)
3 phase_mix = mix_spec / (abs(mix_spec) + EPS)
4 # cos(a - b) = cos(a)*cos(b) + sin(a)*sin(b)
5 cos_theta = phase_r.real * phase_mix.real + phase_r.imag *
   phase_mix.imag
6 mask = (abs(r) / (abs(mix_spec) + EPS)) * cos_theta
7 mask = (
8     mask.clamp(min=0, max=1)
9     ...
10 )

```

1.2 基于谱映射的语音增强算法

1.2.1 处理流程

基于谱映射的语音增强算法的方法更加“硬train一发”，与基于深度学习的图像分割任务类似，这类方法将语谱视为一张图片，网络接收带噪语谱图作为输入，直接预测得到增强的语谱图。本节将以CRN-causal为例展示基于谱映射的语音增强算法的处理过程。

尽管图1.5给出的是SEDNN的框图，但是我们将结合CRN-causal说明谱映射方法的流程。这张流程图看起来就简明得多，和图1.4相反，虚线以上是训练过程而虚线以下是推理过程。训练阶段的带噪语音和纯净语音首先经过特征提取，常见的特征如幅度谱，而后将特征送入DNN。经过损失函数训练后得到优化后的网络参数。测试阶段则将带噪语音 Y^t 经过特征提取得到幅度谱 Y^t （而SEDNN的特征是对数功率谱）和相位谱 $\angle Y^t$ 。幅度谱 Y^t 送入训练好的DNN得到增强的幅度谱 \hat{Y}^t 。最后经过波形重构得到增强的时域波形 \hat{X}^t 。

CRN-causal直接以STFT幅度谱作为输入特征和映射目标。为了方便落地，这里的STFT采用以傅里叶基为卷积核的一维卷积的方式，而后利用 $|X| = \sqrt{\mathcal{R}(X)^2 + \mathcal{I}(X)^2}$ 计算得到幅度谱，具体代码如下：

```

1 # scripts/utils/pipeline_modules.py lines 7-21
2 class NetFeeder(object):
3     def __init__(self, device, win_size=320, hop_size=160):
4         self.eps = torch.finfo(torch.float32).eps
5         self.stft = STFT(win_size, hop_size).to(device)

```

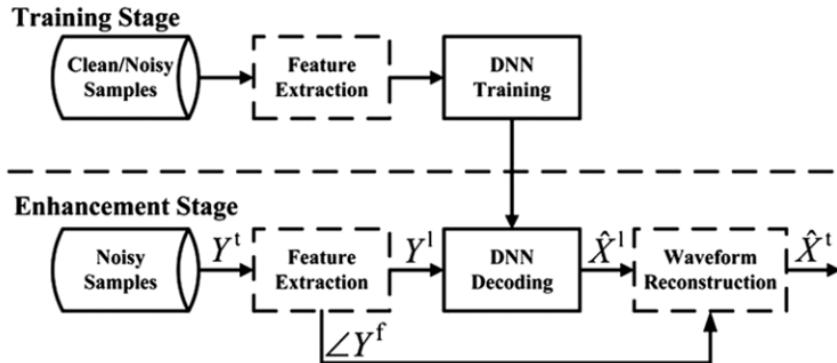


Fig. 1.5 基于谱映射的幅度谱算法框图[5]

```

6     def __call__(self, mix, sph):
7         real_mix, imag_mix = self.stft.stft(mix)
8         mag_mix = torch.sqrt(real_mix**2 + imag_mix**2)
9         feat = mag_mix
10
11
12         real_sph, imag_sph = self.stft.stft(sph)
13         mag_sph = torch.sqrt(real_sph**2 + imag_sph**2)
14         lbl = mag_sph
15         return feat, lbl

```

代码中 $feat$ 和 lbl 分别作为网络的特征和训练目标用于对网络进行训练:

```

1 # scripts/utils/models.py lines 162-174
2 # forward + backward + optimize
3 optimizer.zero_grad()
4 with torch.enable_grad():
5     est = net(feat)
6     loss = criterion(est, lbl, loss_mask, n_frames)
7     loss.backward()
8     if self.clip_norm >= 0.0:
9         clip_grad_norm_(net.parameters(), self.clip_norm)
10    optimizer.step()
11    # calculate loss
12    running_loss = loss.data.item()
13    accu_tr_loss += running_loss * sum(n_frames)
14    accu_n_frames += sum(n_frames)

```

$feat$ 输入网络得到估计的谱 est , 而后计算 est 和 lbl 之间的损失反向传播优化网络参数。

推理过程同样采用NetFeeder类的实例feeder进行特征提取, 将幅度谱特征 $feat$ 送入训练好的网络得到估计的幅度谱 est , 最后经过波形合成过程resynthesizer即可得到增强语音:

```

1 # scripts/utils/models.py lines 337-348
2 feat, lbl = feeder(mix, sph)

```

```

3   with torch.no_grad():
4     ...
5     est = net(feats)
6     ...
7   ...
8   sph_est = resynthesizer(est, mix)

```

波形合成resynthesizer对应的类Resynthesizer按照

$$\begin{aligned}\mathcal{R}(\hat{S}) &= |\hat{S}| \cdot \cos(\angle X), \\ \mathcal{I}(\hat{S}) &= |\hat{S}| \cdot \sin(\angle X)\end{aligned}\quad (1.6)$$

得到增强的复谱，经过iSTFT生成增强的语音波形：

```

1 # scripts/utils/models.py lines 337-348
2 class Resynthesizer(object):
3   def __init__(self, device, win_size=320, hop_size=160):
4     self.stft = STFT(win_size, hop_size).to(device)
5
6   def __call__(self, est, mix):
7     real_mix, imag_mix = self.stft.stft(mix)
8     pha_mix = torch.atan2(imag_mix.data, real_mix.data)
9     real_est = est * torch.cos(pha_mix)
10    imag_est = est * torch.sin(pha_mix)
11    sph_est = self.stft.istft(torch.stack([real_est, imag_est],
12                                         dim=1))
13    sph_est = F.pad(sph_est, [0, mix.shape[1]-sph_est.shape[1]])
14
15   return sph_est

```

最后简单提一下网络结构，网络结构如图1.6所示，结合图1.6和代码可以看出，网络可以分为三部分：编码器、时序建模结构和解码器。编码器由5层二维卷积级联Batch Normalization (BN)和ReLU激活函数组成；时序建模结构是两层LSTM；解码器由5层二维反卷积构成，除了最后一层反卷积层，其余反卷积层同样级联了BN和ReLU激活函数，最后一层反卷积层的激活函数则是Softplus以保证输出结果为非负数从而满足幅度谱非负的要求。幅度谱经过编码器被变换到高维嵌入式(embedding)空间，卷积核用于建模相邻帧和相邻频点之间的相关性。而后通道维和频率特征维被合并到一个维度，经过两层LSTM建模不同帧之间这组特征的时序关系，最后变回与编码器输出相同的张量形状，经过解码器张成与输入幅度谱相同的尺寸，从而得到增强的幅度谱。编解码器之间采用“concatenate形式的skip connection连接”(除此之外还有add形式的skip connection和conv形式的skip connection等)以缓解网络过深可能导致的梯度消失问题。可以看出，该网络架构与图像分割中的编解码器结构极其相似，这是不难理解的，因为以上两个任务都可以看做逐点(时频点/像素点)的滤波问题。然而二者仍有不同之处，相比于图片是一次快拍(借用阵列信号处理中的概念，可以认为是同一时刻)即可全部获得的，语谱图由于其中一个维度是时间帧，需要等待一句话结束后才能得到一张语谱图。在实时应用的推理阶段每次只能获得当前帧(也就是说未来帧的信息是未知

的），因此需要一些技术来保证模型的因果性。这里的代码在编解码器中沿时间帧维度的截断和补零被使用以保证卷积操作的因果性，而我们将在第3章中对因果性进行更详细的讨论。

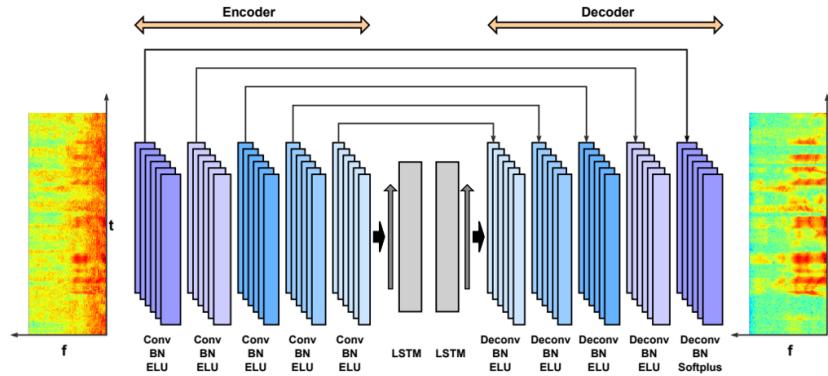


Fig. 1.6 CRN-causal框图[6]

```

1 # scripts/utils/networks.py
2 class Net(nn.Module):
3     def __init__(self):
4         super(Net, self).__init__()
5
6         self.conv1 = nn.Conv2d(in_channels=1, out_channels=16,
7             kernel_size=(2,3), stride=(1,2), padding=(1,0))
8         self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
9             kernel_size=(2,3), stride=(1,2), padding=(1,0))
10        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
11            kernel_size=(2,3), stride=(1,2), padding=(1,0))
12        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128,
13            kernel_size=(2,3), stride=(1,2), padding=(1,0))
14        self.conv5 = nn.Conv2d(in_channels=128, out_channels=256,
15            kernel_size=(2,3), stride=(1,2), padding=(1,0))
16
17        self.lstm = nn.LSTM(256*4, 256*4, 2, batch_first=True)
18
19        self.conv5_t = nn.ConvTranspose2d(in_channels=512,
20            out_channels=128, kernel_size=(2,3), stride=(1,2),
21            padding=(1,0))
22        self.conv4_t = nn.ConvTranspose2d(in_channels=256,
23            out_channels=64, kernel_size=(2,3), stride=(1,2),
24            padding=(1,0))
25        self.conv3_t = nn.ConvTranspose2d(in_channels=128,
26            out_channels=32, kernel_size=(2,3), stride=(1,2),
27            padding=(1,0))
28        self.conv2_t = nn.ConvTranspose2d(in_channels=64,
29            out_channels=16, kernel_size=(2,3), stride=(1,2),
30            padding=(1,0), output_padding=(0,1))

```

```

18     self.conv1_t = nn.ConvTranspose2d(in_channels=32,
19         out_channels=1, kernel_size=(2,3), stride=(1,2),
20         padding=(1,0))
21
22     self.bn1 = nn.BatchNorm2d(16)
23     self.bn2 = nn.BatchNorm2d(32)
24     self.bn3 = nn.BatchNorm2d(64)
25     self.bn4 = nn.BatchNorm2d(128)
26     self.bn5 = nn.BatchNorm2d(256)
27
28     self.bn5_t = nn.BatchNorm2d(128)
29     self.bn4_t = nn.BatchNorm2d(64)
30     self.bn3_t = nn.BatchNorm2d(32)
31     self.bn2_t = nn.BatchNorm2d(16)
32     self.bn1_t = nn.BatchNorm2d(1)
33
34     self.elu = nn.ELU(inplace=True)
35     self.softplus = nn.Softplus()
36
37     def forward(self, x):
38
39         out = x.unsqueeze(dim=1)
40         e1 = self.elu(self.bn1(self.conv1(out)[:, :, :-1, :].
41             contiguous()))
42         e2 = self.elu(self.bn2(self.conv2(e1)[:, :, :-1, :].
43             contiguous()))
44         e3 = self.elu(self.bn3(self.conv3(e2)[:, :, :-1, :].
45             contiguous()))
46         e4 = self.elu(self.bn4(self.conv4(e3)[:, :, :-1, :].
47             contiguous()))
48         e5 = self.elu(self.bn5(self.conv5(e4)[:, :, :-1, :].
49             contiguous()))
50
51         out = e5.contiguous().transpose(1, 2)
52         q1 = out.size(2)
53         q2 = out.size(3)
54         out = out.contiguous().view(out.size(0), out.size(1), -1)
55         out, _ = self.lstm(out)
56         out = out.contiguous().view(out.size(0), out.size(1), q1, q2)
57         out = out.contiguous().transpose(1, 2)
58
59         out = torch.cat([out, e5], dim=1)
60
61         d5 = self.elu(torch.cat([self.bn5_t(F.pad(self.conv5_t(
62             out), [0, 0, 1, 0]).contiguous(), e4], dim=1)))
63         d4 = self.elu(torch.cat([self.bn4_t(F.pad(self.conv4_t(d5
64             ), [0, 0, 1, 0]).contiguous(), e3], dim=1)))
65         d3 = self.elu(torch.cat([self.bn3_t(F.pad(self.conv3_t(d4
66             ), [0, 0, 1, 0]).contiguous(), e2], dim=1)))
67         d2 = self.elu(torch.cat([self.bn2_t(F.pad(self.conv2_t(d3
68             ), [0, 0, 1, 0]).contiguous(), e1], dim=1)))
69         d1 = self.softplus(self.bn1_t(F.pad(self.conv1_t(d2),
70             [0, 0, 1, 0]).contiguous())))

```

```
60     out = torch.squeeze(d1, dim=1)
61
62     return out
```

1.3 小结

本章介绍了基于幅度谱深度学习语音增强技术的两种主流方法：基于掩蔽的语音增强和基于谱映射的语音增强。这两类算法是后续语音增强模型的基础，并将在后续章节中进一步发展。

Chapter 2

基于复数谱的语音增强

随着深度学习的应用和发展，基于幅度谱的语音增强算法相比传统算法取得了显著的性能提升。然而，语谱包含幅度和相位两个分量。这类算法是对幅度进行修正而仍使用带噪信号的相位谱显然不够完美。于是当时语音增强算法的性能瓶颈被理所当然地归咎于这些算法在相位谱上的不作为。相位谱对语音质量恢复的重要性无疑为当时的人们提供了一个绝佳的佐证。因此，一系列意图恢复相位的工作不断被提出。

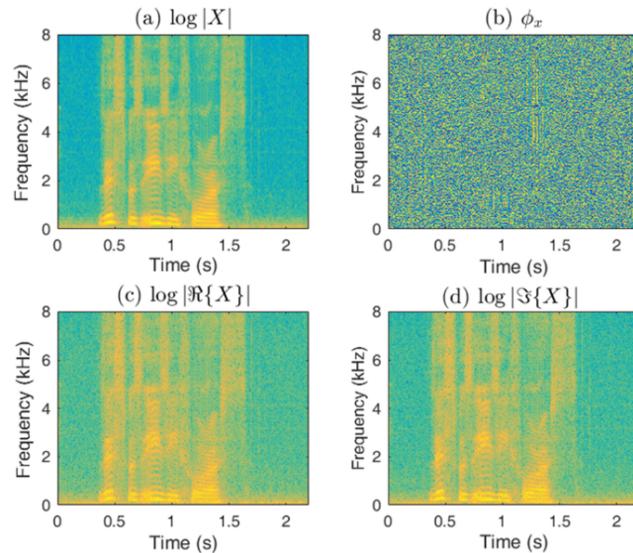


Fig. 2.1 语音复谱图[8]

在谈及基于复数谱语音增强方法的分类之前，有必要对早期语音增强算法局限于幅度谱的原因进行说明。主要原因大致被归结为两点：其一是有证据表明相位信息对人耳的听觉感知不敏感，其二则是相位谱估计不容易。必

须要说明的是，尽管第一点理由经常被相关文献指出，但无论是在传统语音增强的发展中还是在深度学习语音增强的伊始，低信噪比环境下的相位恢复的重要性都始终被提醒。早在[22]中，即在基于幅度谱的深度学习语音增强算法提出时，便强调了第一点只是一种假设，而建立这种假设的苦衷仍然是由于第二点。诚如[2]所言，特征和模型在监督学习中互为补充。强大的模型会放松对特征提取的要求，同样，更具辨别力且结构更加清晰的特征也会使模型学习更加容易。图2.1(b)展示了一段语音的相位谱。可以看出，由于各时频点的相位在时间轴和频率轴上都变化迅速，由于相位缠绕问题，语谱各时频点的相位几乎均匀分布在 $[-\pi, \pi]$ 之间。相位谱并不存在一个清晰的结构模式，这使得模型学习更加困难。而图2.1(c)和(d)展示的复谱的实部和虚部则具有与幅度谱类似的结构模式，使得通过网络估计实部虚部分量从而隐式优化幅度和相位成为可能。

于是，目前针基于复数谱的语音增强方法大致可分为三类：构造值数掩蔽、直接复数谱映射和估计幅度相位分量，其中复值掩蔽按坐标系的区别可分为笛卡尔坐标系下的时频掩蔽和极坐标系下的时频掩蔽。复值掩蔽和复数谱映射通过神经网络预测掩蔽或复谱的实部和虚部。而幅度相位分量估计则是在原有的(基于掩蔽或谱映射的)幅度谱语音增强技术的基础上额外构造模块估计相位谱或与相位相关的群时延、瞬时频率导数等分量。

尽管我们可以大致确认重构相位谱的任务对目前的深度学习技术是影响模型鲁棒性甚至是优化困难的，但将其归因于相位谱缺乏清晰的结构模式似乎只是一种假设解释。另外，需要注意的是试验只表明了相位谱的估计是困难的，但相位谱在一些语音处理领域作为输入特征是成功的。

本章安排如下，在2.1中是基于复数谱映射的语音增强算法的相关描述，基于笛卡尔坐标系下的复值掩蔽和基于极坐标系下的复值掩蔽的介绍分别在2.2和2.3，最后的2.4介绍了幅度相位估计算法的相关工作。

2.1 基于复数谱映射的语音增强

基于复数谱的语音增强模型，顾名思义，就是将复数谱的实部和虚部成分作为网络输入和输出。实现起来也可以非常简单，只需将基于幅度谱的语音增强模型的输入从一个通道的幅度谱改为两个通道的复数谱(实部和虚部)，并将网络的输出从利用激活函数保证非负性的单通道的幅度变成不再使用任何激活函数的双通道的复数谱即可。一个合理的疑问可能会出现在大多数读者心中：直接将实部和虚部沿特征通道拼接是否是合理的，换言之，实部和虚部成分是否应该单独的编码或预测。Tan在2019年通过实验比较了这几种网络架构的性能区别，不出意外地以上结构都可以实现复数谱映射，而Tan更推荐使用单独的编码器同时编码带噪谱的实部和虚部，而后使用两个解码器分别估计增强语谱的实部和虚部。该模型后来被成为GCRN，其卓越的性能

成为了后续很多语音增强工作的基础。因此，这里也采用GCRN作为基于复数谱映射语音增强模型的例子，通过其代码理解这类算法。

GCRN的模型框架和网络参数如图2.2所示。该工作延续了Tan之前的CRN-causal(见1.2)的因果卷积编解码器结合LSTM时序建模的架构。带噪语谱的实部和虚部沿特征通道维拼接后送入一个编码器，编码后的高维特征经过LSTM进行建模后作为两个解码器的输入，两个解码器分别估计得到增强语谱的实部和虚部。其中，卷积和反卷积层用其门控(gating)结构代替以提升语音增强性能，LSTM也引入了分组(grouped)机制减少模型的参数量和计算复杂度，这两个改进点至今仍被许多工作所采用。GCRN的模型前向处理流程为：

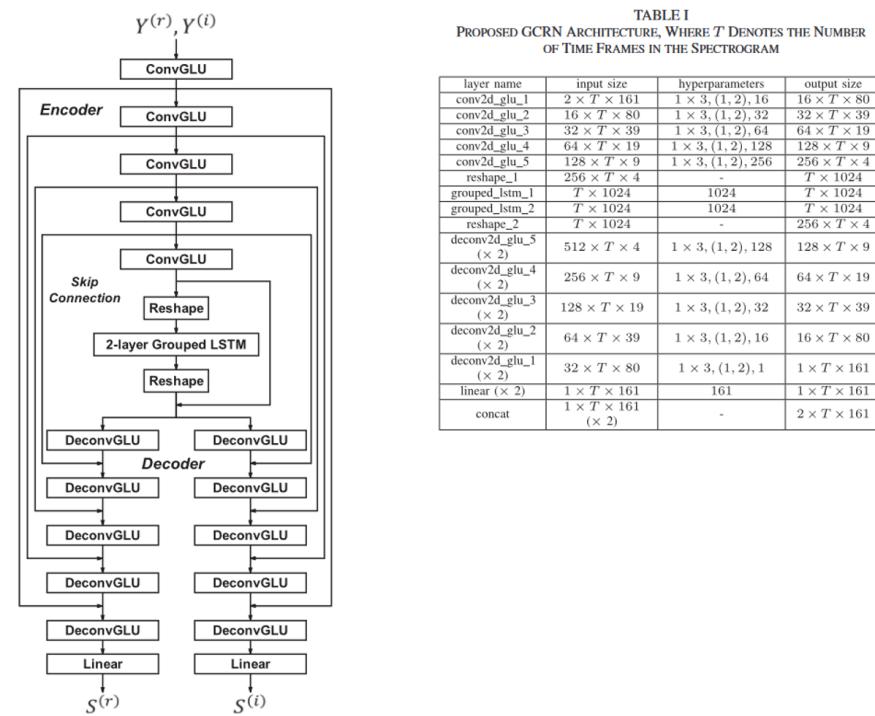


Fig. 2.2 GCRN框架[9]

```

1 # utils/networks.py lines 134–165
2 def forward(self, x):
3     out = x
4     e1 = self.elu(self.bn1(self.conv1(out)))
5     e2 = self.elu(self.bn2(self.conv2(e1)))
6     e3 = self.elu(self.bn3(self.conv3(e2)))
7     e4 = self.elu(self.bn4(self.conv4(e3)))
8     e5 = self.elu(self.bn5(self.conv5(e4)))

```

```

9     out = e5
10    out = self.glstm(out)
11    out = torch.cat((out, e5), dim=1)
12
13    d5_1 = self.elu(torch.cat((self.bn5_t_1(self.conv5_t_1(
14        out)), e4), dim=1))
15    d4_1 = self.elu(torch.cat((self.bn4_t_1(self.conv4_t_1(
16        d5_1)), e3), dim=1))
17    d3_1 = self.elu(torch.cat((self.bn3_t_1(self.conv3_t_1(
18        d4_1)), e2), dim=1))
19    d2_1 = self.elu(torch.cat((self.bn2_t_1(self.conv2_t_1(
20        d3_1)), e1), dim=1))
21    d1_1 = self.elu(self.bn1_t_1(self.conv1_t_1(d2_1)))
22
23    d5_2 = self.elu(torch.cat((self.bn5_t_2(self.conv5_t_2(
24        out)), e4), dim=1))
25    d4_2 = self.elu(torch.cat((self.bn4_t_2(self.conv4_t_2(
26        d5_2)), e3), dim=1))
27    d3_2 = self.elu(torch.cat((self.bn3_t_2(self.conv3_t_2(
28        d4_2)), e2), dim=1))
29    d2_2 = self.elu(torch.cat((self.bn2_t_2(self.conv2_t_2(
        d3_2)), e1), dim=1))
        d1_2 = self.elu(self.bn1_t_2(self.conv1_t_2(d2_2)))

30    out1 = self.fc1(d1_1)
31    out2 = self.fc2(d1_2)
32    out = torch.cat([out1, out2], dim=1)

33    return out

```

GCRN的训练流程如下:

```

1 # utils/models.py lines 145–174
2 for n_iter, egs in enumerate(tr_loader):
3     n_iter += start_iter
4     mix = egs['mix']
5     sph = egs['sph']
6     n_samples = egs['n_samples']
7
8     mix = mix.to(self.device)
9     sph = sph.to(self.device)
10    n_samples = n_samples.to(self.device)
11    n_frames = countFrames(n_samples, self.win_size, self.
12                           hop_size)
13    ...
14    # prepare features and labels
15    feat, lbl = feeder(mix, sph)
16    loss_mask = lossMask(shape=lbl.shape, n_frames=n_frames,
17                          device=self.device)
18    # forward + backward + optimize
19    optimizer.zero_grad()
20    with torch.enable_grad():
21        est = net(feat)
22    loss = criterion(est, lbl, loss_mask, n_frames)

```

```

21     loss.backward()
22     if self.clip_norm >= 0.0:
23         clip_grad_norm_(net.parameters(), self.clip_norm)
24     optimizer.step()
25     # calculate loss
26     running_loss = loss.data.item()
27     accu_tr_loss += running_loss * sum(n_frames)
28     accu_n_frames += sum(n_frames)

```

和CRN-causal一样，feeder是用于特征提取的对象，提取了带噪语音和纯净语音的复数谱。

2.2 基于笛卡尔坐标系的复值掩蔽方法

和基于幅度谱的语音增强方法类似，基于复数谱的语音增强方法除了基于谱映射的形式之外也包括使用掩蔽滤波的方式。网络的输出结果不再是复数谱的实部和虚部成分而是时频掩蔽。复值理想比值掩蔽(complex-valued ideal ratio mask, cIRM)于2016年被提出，处理过程如图2.3-1所示。它将复数域分解为实部和虚部的组合形式，即 $M = M_r + j \cdot M_i \in \mathbb{C}$ 。利用神经网络直接估计实部掩蔽和虚部掩蔽，而后将估计的掩蔽与带噪复数谱按复数乘法相乘进行滤波得到增强语谱：

$$\begin{aligned}\widehat{S}_r &= \widehat{M}_r \cdot X_r - \widehat{M}_i \cdot X_i, \\ \widehat{S}_i &= \widehat{M}_r \cdot X_i + \widehat{M}_i \cdot X_r,\end{aligned}\tag{2.1}$$

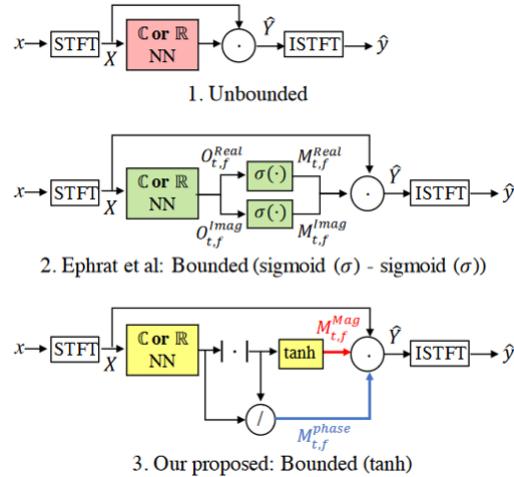


Fig. 2.3 基于复数谱的掩蔽示意图[10]

如果采用MA的方式计算损失函数，则需要按如下方式计算cIRM作为标签：

$$\begin{aligned} M_r &= \frac{X_r \cdot S_r + X_i \cdot S_i}{X_r^2 + X_i^2}, \\ M_i &= \frac{X_r \cdot S_i - X_i \cdot S_r}{X_r^2 + X_i^2} \end{aligned} \quad (2.2)$$

为了与2.3中介绍的掩蔽区分，有时也将cIRM成为基于笛卡尔坐标系的复值掩蔽。需要注意的是，为了便于网络优化，cIRM最初采用有界的形式通过tanh函数将范围压缩至某个对称的范围内。然而后续的工作表明至少采用SA的方式采用无界的cIRM并没有出现优化问题，DPCRN展示了掩蔽的滤波过程：

```

1 # main.py lines 232–246
2 def mk_mask(self, x):
3     ...
4     enh_real = noisy_real * mask_real - noisy_imag *
5         mask_imag
6     enh_imag = noisy_real * mask_imag + noisy_imag *
7         mask_real
8
9     return [enh_real, enh_imag]
```

本节余下部分将简要介绍DPCRN的源码，首先DPCRN的框架如图2.4所示 可以看到，与GCRN相似，网络架构由编码器、DPRNN构成的时序建

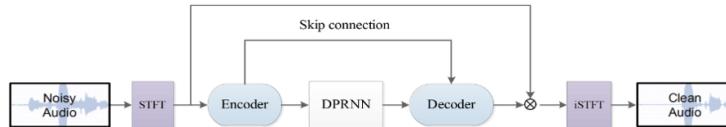


Fig. 2.4 DPCRN框架[11]

模模块和解码器组成，编解码器与GCRN相似——五层由二维因果卷积层+BN层+PReLU层构成的编码层和五层由二维因果转置卷积+BN层+PReLU层构成的解码层(为了保证输出无界，最后一个转置卷积不级联BN层和PReLU)，只是舍弃了门控卷积并且只使用了一个解码器。时序建模模块的DPRNN源自[?], 为了发挥DPRNN的作用，编码器输出的特征维度设为较大的128。编码器的输入维度设为2以使带噪复数谱按实部、虚部沿特征通道维度拼接后输入，对应的，解码器的输出维度也设为2从而得到cIRM的实部和虚部。值得注意的是一种instant layer normalization(iLN)以应对输入语音动态范围过大问题。DPCRN在谱域上采用SA的损失函数未对掩蔽约束至有界。模型代码如下：

```

1 # main.py lines 248–341
2 def build_DPCRN_model(self, ...):
3     ...
```

```

4     ''' encoder '''
5     ...
6     input_complex_spec = LayerNormalization(...)(
7         input_complex_spec)
8     ...
9     conv_1 = Conv2D(32, (2,5),(1,2),...)(input_complex_spec)
10    bn_1 = BatchNormalization(...)(conv_1)
11    out_1 = PReLU(shared_axes=[1,2])(bn_1)
12    ...
13    conv_5 = Conv2D(128, (2,3),(1,1),...)(out_4)
14    bn_5 = BatchNormalization(...)(conv_5)
15    out_5 = PReLU(shared_axes=[1,2])(bn_5)
16
17    dp_in = out_5
18    for i in range(self.numDP):
19        dp_in = DprnnBlock(...)(dp_in)
20        dp_out = dp_in
21
22    ''' decoder '''
23    skipcon_1 = Concatenate(axis = -1)([out_5, dp_out])
24    deconv_1 = Conv2DTranspose(64,(2,3),(1,1),...)(skipcon_1)
25    dbn_1 = BatchNormalization(...)(deconv_1)
26    dout_1 = PReLU(shared_axes=[1,2])(dbn_1)
27    ...
28    skipcon_5 = Concatenate(axis = -1)([out_1, dout_4])
29    deconv_5 = Conv2DTranspose(2,(2,5),(1,2),...)(skipcon_5)
30    ...
31    output_mask = deconv_5
32
33    enh_spec = Lambda(self.mk_mask)([real, imag, output_mask])
34

```

除此之外，这里不得不提及S. Braun等人于2021年提出的CRUSE模型，其作为一个参数量和计算量较低的实时语音增强模型在ICASSP2021的DNS Challenge中取得了极其优异的成绩。网络架构如图2.5所示，同样是编解码器结构，和DPCRN一样(但它比DPCRN更早提出)，门控卷积是被舍弃的，解码器也减至一个。此外GCRN的分组机制被保留只是RNN从LSTM变为计算量和参数量更少的GRU，skip connection也不再以拼接的方式而是以相加的方式(如图2.6)将编码器的信息送入解码器中，进一步降低了参数量和计算量。

CRUSE的特征采用了幂律压缩的复数谱($X_{cprs} = \frac{X(k,n)}{|X(k,n)|} |X(k,n)|^{0.3}$)，网络输出STFT域的cIRM，对带噪复谱进行滤波后采用能量级不变的幂律压缩RI+Mag损失函数结合STFT一致性约束进行网络优化。以上训练流程如图2.7所示 损失函数定义为

$$\mathcal{L} = \frac{1}{\sigma_S^c} (\lambda \sum_{k,n} |S^c - \hat{S}^c|^2 + (1 - \lambda) (\sum_{k,n} ||S|^c - |\hat{S}|^c|^2)), \quad (2.3)$$

其中 σ_S 是纯净语音有声段的能量， c 设为0.3。

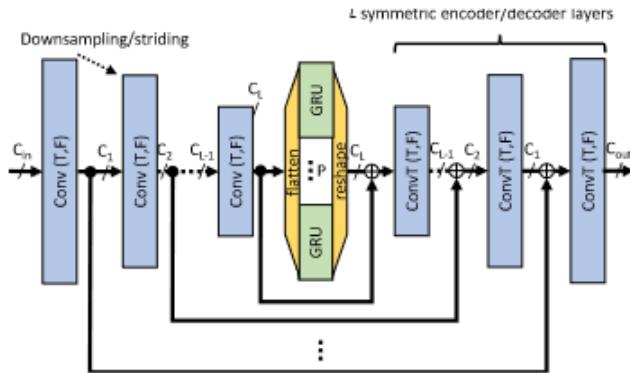


Fig. 2.5 CRUSE框架[12]

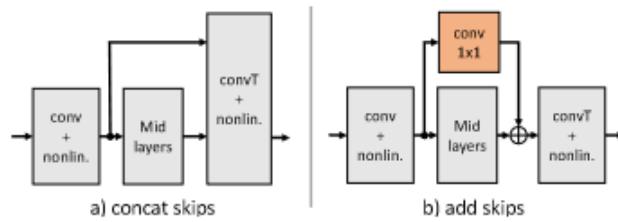


Fig. 2.6 skip connection的改进[12]

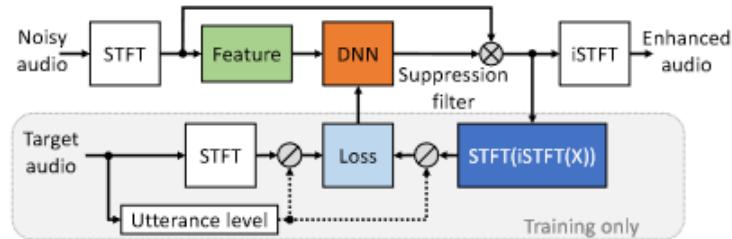


Fig. 2.7 CRUSE训练流程[12]

处理针对当前时频点的复值掩蔽，利用当前时频点相邻时频点进行滤波求和计算得到当前时频点谱估计的深度滤波(deep filtering)方法近年来也开始被使用。深度滤波的形式可写作：

$$\widehat{S}(t, f) = \sum_{k=-K}^K \sum_{i=0}^N \widehat{G}(l, k) \odot X(t - i + l, f - k). \quad (2.4)$$

式中 K 和 N 分别为与当前时频点相邻的频点数和帧数, l 为前看(look-ahead)偏移, 若为因果模型则 l 取0。有时 K 取0, 即指利用相同频率的当前帧和历史帧信息进行滤波, 而不使用相邻频点信息。 \odot 是指复数乘法。

2.3 基于极坐标系的复值掩蔽方法

区别于基于笛卡尔坐标系下的复值掩蔽, 基于极坐标系的复值掩蔽将掩蔽对带噪复谱的滤波作用按如下方式理解:

$$\widehat{S} = \widehat{M} \cdot X = |\widehat{M}| \cdot X \cdot e^{j(\angle \widehat{M} + \angle X)} \quad (2.5)$$

复值掩蔽可视为对每个时频点幅度的放缩和相位的旋转。为了方便网络优化, 作用在幅度谱上的掩蔽通过 $tanh(\cdot)$ 函数将无界范围非线性映射到复平面的单位圆内。基于极坐标系的复值掩蔽计算方法如图2.3(b)-3所示, 网络隐式的估计该掩蔽的实部和虚部成分后, 分别计算得到无界的幅度掩蔽分量和相位掩蔽分量, 无界的幅度掩蔽分量通过 $tanh(\cdot)$ 激活函数得到有界的幅度掩蔽分量。而后将估计得到的掩蔽按式2.5对带噪复谱进行滤波, 从而得到增强语音的复谱。H. S. Choi等通过实验分析了基于笛卡尔坐标系和极坐标系的复值掩蔽在谱域和时域损失函数下处理的结果, 通过波形对比表明了基于极坐标系的复值掩蔽在时域损失函数下处理结果与纯净语音波形更相似, 而谱域损失函数并不利于相位重构。不过需要注意的是, 该结论是基于波形相似度得出的, 在听感上该结论是否成立是还需验证的。

这里采用DCCRN的代码进行举例。该模型输入是将实部和虚部沿特征通道维度拼接的带噪的复数谱, 网络输出的是基于极坐标系的复值掩蔽, 而后复值掩蔽对带噪语音的幅度和相位谱进行调整, 并通过iSTFT变换回时域的增强波形。其中网络采用了编码器-时序建模-解码器架构, 与之前提出方法不同是, 各模块都采用了复值网络。复值网络简单来说就是将特征沿特征通道维对半分成“实部” E_r 和“虚部” E_i , 二者分别经过两个参数不共享的网络层 $f_r(\cdot)$ 和 $f_i(\cdot)$, 得到输出的 $f_r(E_r)$ 、 $f_i(E_r)$ 、 $f_r(E_i)$ 和 $f_i(E_i)$, 而后这四个分量模仿复数乘法的计算规则得到最终结果 $(f_r(E_r) - f_i(E_i)) + j(f_r(E_i) + f_i(E_r))$ 。

```

1 # dc_crn.py lines 150-234
2 def forward(self, inputs, lens=None):
3     # 特征提取部分
4     specs = self.stft(inputs)
5     real = specs[:, :self.fft_len//2+1]
6     imag = specs[:, self.fft_len//2+1:]
7     spec_mags = torch.sqrt(real**2+imag**2+1e-8)
8     spec_phase = torch.atan2(imag, real)
9     cspecs = torch.stack([real, imag], 1)
10    cspecs = cspecs[:, :, 1:]
11    out = cspecs
12
13    # 网络部分
14    encoder_out = []

```

```

15     for idx, layer in enumerate(self.encoder):
16         out = layer(out)
17         encoder_out.append(out)
18
19         batch_size, channels, dims, lengths = out.size()
20         out = out.permute(3, 0, 1, 2)
21         if self.use_clstm:
22             r_rnn_in = out[:, :, :channels//2]
23             i_rnn_in = out[:, :, channels//2:]
24             r_rnn_in = torch.reshape(r_rnn_in, [lengths, batch_size,
25                                           channels//2*dims])
26             i_rnn_in = torch.reshape(i_rnn_in, [lengths, batch_size,
27                                           channels//2*dims])
28             r_rnn_in, i_rnn_in = self.enhance([r_rnn_in, i_rnn_in])
29             r_rnn_in = torch.reshape(r_rnn_in, [lengths, batch_size,
30                                           channels//2, dims])
31             i_rnn_in = torch.reshape(i_rnn_in, [lengths, batch_size,
32                                           channels//2, dims])
33             out = torch.cat([r_rnn_in, i_rnn_in], 2)
34             ...
35             out = out.permute(1, 2, 3, 0)
36
37     for idx in range(len(self.decoder)):
38         out = complex_cat([out, encoder_out[-1 - idx]], 1)
39         out = self.decoder[idx](out)
40         out = out[..., 1:]
41
42         mask_real = out[:, 0]
43         mask_imag = out[:, 1]
44         mask_real = F.pad(mask_real, [0, 0, 1, 0])
45         mask_imag = F.pad(mask_imag, [0, 0, 1, 0])
46
47         # 极坐标系复值掩蔽的计算和滤波
48         if self.masking_mode == 'E':
49             mask_mags = (mask_real**2+mask_imag**2)**0.5
50             real_phase = mask_real/(mask_mags+1e-8)
51             imag_phase = mask_imag/(mask_mags+1e-8)
52             mask_phase = torch.atan2(imag_phase, real_phase)
53             mask_mags = torch.tanh(mask_mags)
54             est_mags = mask_mags*spec_mags
55             est_phase = spec_phase + mask_phase
56             real = est_mags*torch.cos(est_phase)
57             imag = est_mags*torch.sin(est_phase)
58             ...
59
60             out_spec = torch.cat([real, imag], 1)
61
62             # 波形合成部分
63             out_wav = self.istft(out_spec)
64             out_wav = torch.squeeze(out_wav, 1)
65             out_wav = torch.clamp_(out_wav, -1, 1)
66             return out_spec, out_wav

```

代码通过一维卷积实现STFT和iSTFT操作，幅度和相位可以通过实部和虚部计算得到，即 $|X| = \sqrt{X_r^2 + X_i^2}$, $\angle X = \arctan(\frac{X_i}{X_r})$ 。为了数值稳定， $\sqrt{(\cdot)}$ 内加了小的正值。由于DCCRN舍弃了直流量，因此在送入网络前特征的直流成分被截断，网络输出结果在直流频点用0填充。

2.4 基于幅度相位分量估计的语音增强

由于基于幅度谱的语音增强算法的成功应用，直接估计幅度和相位成分是一个极其自然的想法。尽管前文提到的估计实部和虚部的方式理论上完成了相同任务的同时还规避了相位优化的问题，然而训练目标理论上的等价并不代表网络的优化会符合我们的预期，D. Yin等就指出Ephrat等将cIRM作为目标的算法(如图2.3-2)估计的掩蔽虚部分量几乎全为0[13]，因此仍有一些工作在相位恢复问题上继续努力。

需要注意的是，目前还缺乏相关文献表明更一般的cIRM以及基于极坐标系下的复值掩蔽存在相似的问题。另外，除了训练目标外，损失函数同样对网络增强结果存在影响。只能说目前关于复数谱恢复的问题尚存在提升的空间，在某些复值掩蔽和损失函数的作用下存在和预期相距较大的情况。但更一般的结论仍需等待更多的实验结果。

2018年，一些相位估计网络被提出用于解决相位问题，如N. Takahashi等人提出将相位离散化作为一个分类问题，S. Takamichi等人则在增强的幅度谱的基础上用一个von-mises分布全连接网络在相位距离和群时延距离的损失函数下估计相位谱。

解决相位无清晰结构的另一种直观的想法就是寻找能够表征相位信息的有清晰结构模式的物理量。2019年，Z. Wang等和N. Zheng等分别提出同时估计幅度谱和群时延/瞬时频率导数，成为真正意义上基于深度学习的幅度相位分量估计算法。群时延和瞬时频率导数等物理量其实早在传统语音增强算法中就被提出用于代替相位谱的估计，通过图2.8也可以发现，两个物理量的结构与幅度谱结构也具有极强的相关性。两个工作的网络结构十分类似，如图2.9所示，将基于幅度谱的语音增强模型(LSTM/DNN)的最后一层拆分成两个分支，分别估计幅度和与相位相关的物理量。

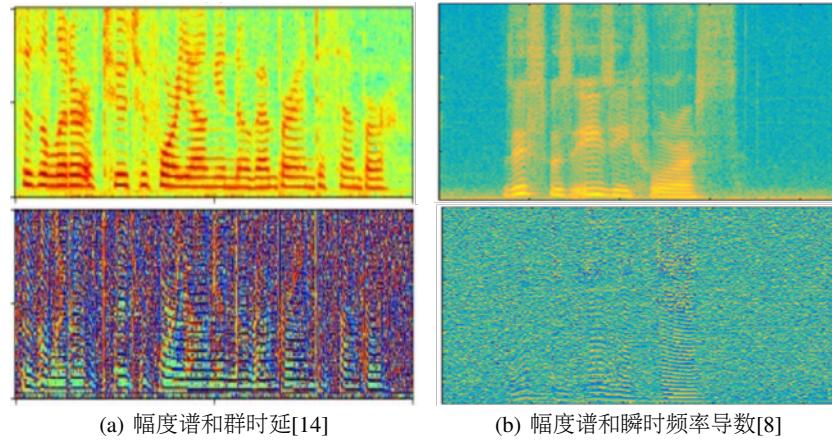


Fig. 2.8 群时延和瞬时频率导数示意图

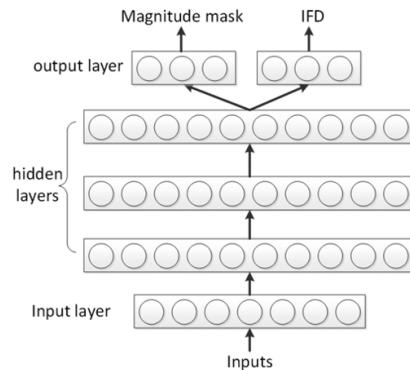


Fig. 2.9 基于幅度相位分量估计的语音增强模型架构[?]

2020年，PHASEN按如下方式表征语音增强过程：

$$\widehat{S} = |X| \cdot \widehat{M} \cdot \Psi. \quad (2.6)$$

其采用双流结构将幅度谱掩蔽和复数谱映射的方式结合起来，一个分支用于和基于幅度谱的语音增强算法一样估计幅度掩蔽 \widehat{M} ，另一个分支则只映射与相位相关的复谱 Ψ ，与基于复谱映射的方法一样， Ψ 被分解为实部 Ψ_r 和虚部 Ψ_i 两部分，通过两个分支之间的信息交互实现相位的预测。

2.5 小结

基于复数谱的语音增强方法进一步提升了基于幅度谱的语音增强算法性能，并成为目前语音增强研究的重点。这一阶段编解码结构的性能逐渐成为多数人的共识，许多高效的网络模块相继被提出，双流结构等新的结构也开始出现。严格地说，后续的第5章中的大部分方法仍属于基于复数谱的语音增强算法。然而由于其重框架轻结构，因此单列一章。

Chapter 3

混合式和生成式语音增强模型

3.1 混合式语音增强模型

神经网络除了用于直接估计滤波器系数或语谱，也常作为语音增强系统的参数估计器，我们将这类算法称为混合式语音增强(Hybrid speech enhancement)模型。

混合式语音增强模型主要建立在传统的基于统计模型的语音增强算法的基础上，利用神经网络实现先验信噪比的估计。这是由于，基于统计模型的语音增强算法的增益因子往往由先验和后验信噪比决定，而后验信噪比可以通过判决引导(Decision-Directed, DD)算法从先验信噪比中估计得到。因此，先验信噪比的估计是这类算法的核心。由于传统先验信噪比估计算法面对高度非平稳噪声缺乏良好的跟踪能力，神经网络被用于代替对应的传统模块以实现不对噪声和语音特性进行假设的、能够迅速响应噪声特性变化的先验信噪比估计。早在2011年，利用神经网络估计先验信噪比的算法就已被提出[15]用于校正传统算法估计的先验信噪比。2018年，Xia等人利用RNN辅助DD算法估计先验信噪比。2019年和2020年，DeepXi和DeepMMSE分别被提出利用网络估计先验信噪比的累计分布函数以改善网络收敛性能。

这里以DeepXi为例介绍一下基于先验信噪比估计的混合式语音增强模型的大致流程。DeepXi建立在传统的MMSE-LSA算法之上，MMSE-LSA的增益函数表示为：

$$G(t, f) = \frac{\xi(t, f)}{\xi(t, f) + 1} \exp\left(\frac{1}{2} \int_{v(t, f)}^{\infty} \frac{e^{-t}}{t} dt\right), \quad (3.1)$$

式中 $v(t, f) = \frac{\xi(t, f)}{\xi(t, f) + 1} \gamma(t, f)$ ，取决于先验信噪比 $\xi(t, f)$ 和后验信噪比 $\gamma(t, f) = \xi + 1$ 。可以看到，增益因子中所有参数均取决于先验信噪比 ξ 。关于算法增益函数的推导这里不做展开，感兴趣的读者可参看[16]。

DeepXi采用ResLSTM模型训练带噪幅度谱和先验信噪比的累计分布函数 $\bar{\xi} = \frac{1}{2} \left[1 + \operatorname{erf}\left(\frac{\xi_{dB} - \mu}{\sqrt{2}\sigma}\right) \right]$ 之间的映射。在推理阶段，估计的先验信噪比 $\hat{\xi}$ 由网络输出 $\bar{\xi}$ 经过 $\hat{\xi}_{dB} = \sqrt{2}\sigma \operatorname{erf}^{-1}(2\hat{\xi} - 1) + \mu$ 及 $\hat{\xi} = 10^{(\hat{\xi}_{dB}/10)}$ 得到后，

代入MMSE-LSA增益函数公式3.1即可对带噪幅度谱进行滤波得到增强语谱 $\hat{S} = G \cdot |X| \cdot e^{\angle X}$ 。

另一类混合式语音增强算法严格意义上并不仅在幅度谱上进行，也并非源于语音增强领域的传统算法。RNNoise是其中最著名的代表之一，其改进版PercepNet更是取得了令人瞩目的性能。他们明显受到了语音编码算法的影响，按照CELT编码器的思路，将语谱分解为语谱包络和语谱细节(激励部分)两部分。根据源-滤波器模型(source-filter model)，语音的包络由声道形状的变化得到，语谱细节则是由声带的准周期性震动产生，主要是基频及其谐波。根据人耳听觉感知机制，语谱包络可以在某种听觉感知相关的低分辨率频带(band)上进行操作。CELT中“应当保证重构的语谱包络能量在频带上与纯净语谱包络相等”的思想启发我们可以通过计算增益函数得到增强语音的语谱包络，此时的语谱包络在浊音段仍是粗糙的，而在其他段则与纯净语谱听感上相似。而语谱细节——也就是基频及其谐波——可以通过基频滤波进行修正，抑制掉谐波间的噪声，此时应在高分辨率的STFT谱或时域上进行，主要包含两步：首先利用梳状滤波器得到滤波后的语谱 \hat{P} ，然后根据网络估计的基频滤波强度 r 加权组合带噪谱成分和滤波信号谱成分以增加语音的自然度 $Z = (1 - r) \cdot X + r \cdot \hat{P}$ 。图3.1展示了上述过程，其中特征提取器feature extraction、神经网络DNN model和包络后滤波envelope postfilter承担了语谱包络估计任务(分别用于变换到低分辨率的ERB频带、估计语谱包络增益函数和通过增益补偿和加混响改善听感)；而神经网络DNN model和基频滤波pitch filtering则承担了语谱细节估计任务(分别用于估计基频滤波强度和基频滤波)。毫无疑问神经网络并非混合式语音增强模型的重点，因此这里一笔掠过：RNNoise试图按照谱减法框架设计RNN网络完成语谱包络修正，PercepNet则直接使用多任务学习的CRN同时估计语谱包络增益因子和基频滤波强度系数。

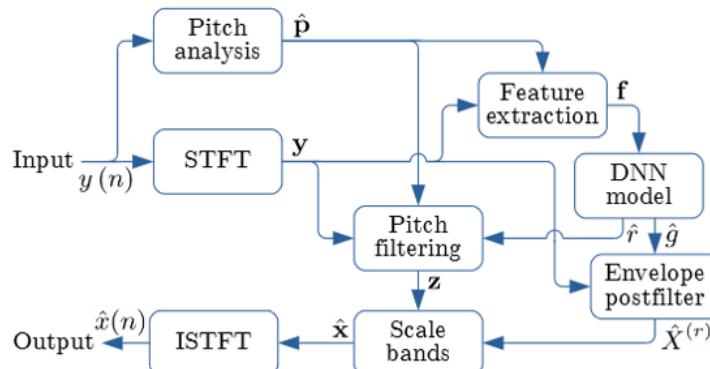


Fig. 3.1 PercepNet框图[?]

值得注意的是，关于RNNoise和PercepNet的讨论并未结束，其将语音分解为包络和细节的思想被后续工作完全神经网络化，这种思想催生出的模型在

第5章中详细介绍。此外，随着超宽带和全带语音增强的任务得到重视，其在低分辨率听觉感知频带的处理思路仍具参考价值。

除了以上两类算法之外，还有一种时域混合式语音增强算法。这类算法将神经网络与Kalman滤波的结合，将语音生成视为一种自回归(auto-aggressive, AR)过程(这同样在语音编码和生成领域被广泛应用)，使用基于MMSE准则的Kalman滤波器增强语音波形，从而隐式地修正带噪语音的幅度和相位。

这一过程中，Kalman滤波器的参数主要是线性预测系数(linear prediction coefficients, LPCs)以及激励噪声和测量噪声的统计特性。神经网络被用于估计Kalman滤波器的三个参数(也可能只估计LPCs，其他参数利用传统算法估计)，这些参数被带入Kalman滤波器的“预测-更新”公式，从而得到滤波器系数。语音增强任务的“预测-更新”公式如下，同样，更详细的公式化过程可参看[17]：

$$\begin{aligned}\mathbf{K}(n) &= \mathbf{P}(n|n-1)(\mathbf{R}_w + \mathbf{P}(n|n-1))^{-1}, \\ \hat{\mathbf{s}}(n|n) &= \hat{\mathbf{s}}(n|n-1) + \mathbf{K}(n)(\mathbf{x}(n) - \hat{\mathbf{s}}(n|n-1)), \\ \mathbf{P}(n|n) &= (\mathbf{I} - \mathbf{K}(n))\mathbf{P}(n|n-1), \\ \hat{\mathbf{u}}(n+1|n) &= \mathbf{F}\hat{\mathbf{u}}(n|n), \\ \mathbf{P}(n+1|n) &= \mathbf{F}\mathbf{P}(n|n)\mathbf{F}^T + \sigma_v^2 \mathbf{G}\mathbf{G}^T,\end{aligned}\quad (3.2)$$

式中 $\mathbf{s}(n) = [s(n-p+1), \dots, s(n)]^T$, $\mathbf{x}(n) = [x(n-p+1), \dots, x(n)]^T$, $\mathbf{G} = [0, \dots, 0, 1]^T$, p 是对纯净语音进行AR建模的阶数。 \mathbf{F} 、 σ_v^2 和 \mathbf{R}_w 分别代表包含LPCs的变换矩阵、AR建模时的激励噪声方差和kalman滤波建模时的测量噪声协方差矩阵。

通过观察 $\mathbf{s}(n)$ 的定义不难得出，采样点 n 的增强语音 $\hat{\mathbf{s}}(n)$ 可通过 $\mathbf{G}\hat{\mathbf{s}}(n|n)$ 得到。

上面介绍了最简单的基于Kalman滤波的混合式语音增强模型，利用线谱频率(line spectrum frequencies, LSFs)代替LPCs作为网络映射目标、采用子带Kalman滤波和Colored-Noise Kalman滤波等Kalman滤波器变体等改进也相继被提出。图3.2展示了这类算法的训练和推理流程。

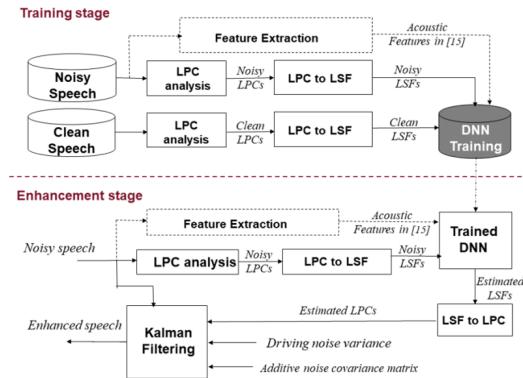


Fig. 3.2 基于Kalman滤波的混合式语音增强框图[?]

3.2 生成式语音增强模型

与前文提及的所有建立从带噪语音到纯净语音/掩蔽函数/信噪比映射关系的语音增强方法不同，生成式语音增强模型(generative models for speech enhancement)强调对纯净语音先验分布的学习，以学习语音时序或谱频结构特性为目的。这类算法在语音生成领域的成功应用使得研究者开始尝试生成式模型在语音增强任务上的可能，毕竟语音增强除了被视为滤波(分离)问题也同样可以被视为生成问题。

生成对抗网络(Generative Adversarial Network, GAN)无疑是近年来最引人瞩目的生成式模型，其在计算机视觉领域展示出的以假乱真的能力引起了整个深度学习领域的关注。最早将GAN引入语音增强领域的是Pascual等人提出的SEGAN[46]，其使用GAN的动机大抵是由于非GAN模型的增强语音受限于其采用的损失函数，确实时至今日仍未出现一种损失函数能完美地衡量听感上的相似度，这也不可避免地使语音增强模型陷入某种上界的约束。而使用一个鉴别器去基于数据驱动地判别增强语音和纯净语音的某种相似度，确实可能达到对目前损失函数带来的缺陷的补偿。而这种补偿至少不会陷入在利用语音增强评价指标作为损失函数优化网络时所带来的仅有对应的指标明显提升的困境之中。于是，其生成器 $G(\cdot)$ 采用了时域全卷积编解码器结构，不过在bottleneck中引入了高斯分布的噪声向量 z ，这是源自GAN的历史遗留，不过后续实验发现其带来的贡献并不明显[47]，因此后续很多工作放弃了 z 的引入。正如前面所言，可以将GAN的语音增强视为对损失函数的改进，生成器的损失函数可以看作原本时域语音增强中常用的 \mathcal{L}_1 距离损失外加一个对抗损失函数。对抗损失函数和鉴别器 $D(\cdot)$ 的损失函数均采用最小二乘GAN(LSGAN)的损失，损失函数表征如下：

$$\mathcal{L}_G = \frac{1}{2} \mathbb{E}_{z,x} [(D(G(z,x), x) - 1)^2] + \lambda \|G(z,x) - s\|_1, \quad (3.3)$$

$$\mathcal{L}_D = \frac{1}{2} \mathbb{E}_{s,x} [(D(s,x) - 1)^2] + \frac{1}{2} \mathbb{E}_{z,x} [(D(G(z,x), x))^2]. \quad (3.4)$$

基于GAN的训练方式大多如图3.3所示，首先利用纯净语音-带噪语音对和增强语音-带噪语音对训练鉴别器使鉴别器具有分类纯净语音和增强语音的能力，而后固定鉴别器参数后训练生成器，使生成器尽可能骗过鉴别器。

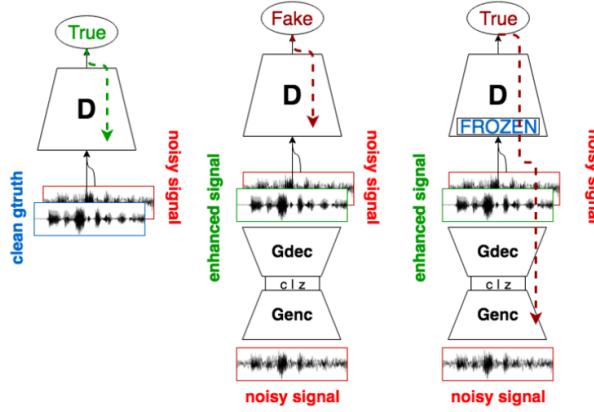


Fig. 3.3 基于GAN的语音增强训练流程[46]

同年，结合条件GAN(cGAN)的谱域生成器也被提出，随后基于GAN的语音增强算法大多是将第1、2、4章中提到的增强模型引入生成器、改进Normalization、或是引入更先进的GAN(如Wasserstein GAN、Relativistic GAN等)以及鉴别器损失(上下文金字塔、多尺度、多周期等)以提升性能，本节并不打算对这类工作展开介绍，感兴趣的读者可参考[48]了解。值得一提的是，MetricGAN如前文所述将cGAN与基于语音评价指标的损失函数设计思想紧密结合，鉴别器不再只判别真假，而是估计增强语音的纯净程度。

总的来说，由于基于GAN的语音增强为了方便比较结果，大多采用统一的但规模较小且信噪比偏高的数据集(Voice-Bank)；同时为了与前人的工作进行比较大多数采用非因果模型；此外，或许带噪语音和纯净语音的高度相似性也是GAN在语音增强领域没有再现其在图像生成、语音合成等领域大放异彩的原因。总之，这类工作目前看来颇有孤芳自赏之感，少有“出圈”的工作与其他语音增强模型进行公平的性能比较。另外由于其更像是一种提升生成器性能的策略，因此难免大多数工作更专注于生成器本身的发展，这点在诸届DNS Challenge中体现明显。

本节以HiFi-GAN为例展示基于GAN的语音增强算法的实现。

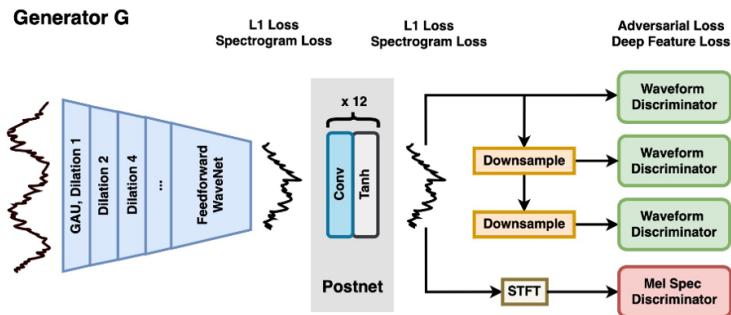


Fig. 3.4 HiFi-GAN网络架构[49]

HiFi-GAN[49]的网络框架如图3.4所示，训练阶段网络由两部分组成，分别是WaveNet和Postnet组成的生成器，以及由时域多尺度鉴别器和频域鉴别器组成的一组鉴别器。其中WaveNet是常见的时域降噪网络，它被以0.001的学习率单独训练多轮以保证其具有提取波形模式的能力，而后其与Postnet一同以更小的0.0001的学习率训练，对应下面代码的23-24行。最后鉴别器也加入训练，以较大的0.001为学习率并且每步更新两次参数，而此时生成器以较小的0.0001作为学习率更新参数。其中学习率和不平衡训练主要是为了使生成器和鉴别器之间能够达到动态平衡的博弈，其设置在不同文章中也有所区别。这里并不试图使读者仅通过本例就了解GAN的各种不同的训练技巧，也不会展开介绍网络架构。仅希望以此为例展示一种常见的GAN的训练流程，需要注意的是这里提及的训练流程也只是其中一种。不过相比于GAN的训练技巧，其训练流程相对更加简单，感兴趣的读者可以参考此博客。

```

1 # train.py lines 32-265
2 generator = Generator(hp.model.in_channels, hp.model.out_channels
3     ).to(device)
4 specd = SpecDiscriminator().to(device)
5 msd = MultiScaleDiscriminator().to(device)
6 stft_loss = MultiResolutionSTFTLoss()
7 ...
8 generator.train()
9 specd.train()
10 msd.train()
11 with_postnet = False
12 for epoch in range(max(0, last_epoch), args.training_epochs):
13     ...
14     for i, batch in enumerate(train_loader):
15         ...
16         x, y, file, _, y_mel_loss = batch
17         x = torch.autograd.Variable(x.to(device, ...))
18         y = torch.autograd.Variable(y.to(device, ...))
19         y_mel_loss = torch.autograd.Variable(y_mel_loss.to(device
20             , ...))
21         x = x.unsqueeze(1)
22         y = y.unsqueeze(1)
23         before_y_g_hat, y_g_hat = generator(x, with_postnet)
24         ...
25         if y_g_hat is not None:
26             y_g_hat_mel = mel_spectrogram(y_g_hat.squeeze(1),
27                 ...)

28         if steps > hp.train.discriminator_train_start_steps:
29             for _ in range(hp.train.rep_discriminator):
30                 optim_d.zero_grad()
31                 # SpecD
32                 y_df_hat_r, y_df_hat_g, _, _ = specd(y_mel_loss,
33                     y_g_hat_mel.detach())
34                 loss_disc_f, losses_disc_f_r, losses_disc_f_g =
35                     discriminator_loss(y_df_hat_r, y_df_hat_g)
36                 # MSD
37

```

```

33     y_ds_hat_r, y_ds_hat_g, _, _ = msd(y, y_g_hat.
34         detach())
35     loss_disc_s, losses_disc_s_r, losses_disc_s_g =
36         discriminator_loss(y_ds_hat_r, y_ds_hat_g)
37     loss_disc_all = loss_disc_s + loss_disc_f
38
39     loss_disc_all.backward()
40     optim_d.step()
41
42     before_y_g_hat_mel = mel_spectrogram(before_y_g_hat.
43         squeeze(1), ...)
44     optim_g.zero_grad()
45     # L1 Mel-Spectrogram Loss
46     sc_loss, mag_loss = stft_loss(before_y_g_hat[:, :, :y.
47         size(2)].squeeze(1), y.squeeze(1))
48     before_loss_mel = sc_loss + mag_loss
49     # L1 Sample Loss
50     before_loss_sample = F.l1_loss(y, before_y_g_hat)
51     loss_gen_all = before_loss_mel + before_loss_sample
52     ...
53     if steps > hp.train.discriminator_train_start_steps:
54         y_df_hat_r, y_df_hat_g, fmap_f_r, fmap_f_g = specd(
55             y_mel_loss, y_g_hat_mel)
56         y_ds_hat_r, y_ds_hat_g, fmap_s_r, fmap_s_g = msd(y,
57             y_g_hat)
58         loss_fm_f = feature_loss(fmap_f_r, fmap_f_g)
59         loss_fm_s = feature_loss(fmap_s_r, fmap_s_g)
60         loss_gen_f, losses_gen_f = generator_loss(y_df_hat_g)
61         loss_gen_s, losses_gen_s = generator_loss(y_ds_hat_g)
62         loss_gen_all += hp.model.lambda_adv * (loss_gen_s +
63             loss_gen_f + loss_fm_s + loss_fm_f)

64     loss_gen_all.backward()
65     optim_g.step()
66     ...

```

仅考虑完整的GAN训练过程(即示例代码中所有if语句均为True时)，可以看出其训练思路与图3.3类似，首先训练鉴别器区分纯净语音和增强语音，而后训练生成器生成更真实的增强结果。具体而言，首先模型经过生成器后将输出变量的detach()作为specd和msd两个判别器的输入，计算得到鉴别器的损失函数，损失函数反向传播得到生成器和鉴别器的梯度(其中鉴别器梯度已完成清零)，最后对鉴别器的权重进行更新；而后计算生成器的Mel谱距离、时域采样点距离损失和对抗损失函数(包括specd、msd和深度特征匹配损失)，计算得到生成器和此时鉴别器的梯度(其中生成器梯度已完成清零)，最后对生成器的权重进行更新。以上过程实现鉴别器和生成器的交替训练。

在推理过程中，仅使用WaveNet即可完成语音增强，代码如下：

```

1 # inference.py
2 wav, sr = load_wav(os.path.join(a.input_wavs_dir, filename))
3 wav = wav / MAX_WAV_VALUE
4 wav = normalize(wav) * 0.95
5 wav = torch.FloatTensor(wav)
6 wav = wav.reshape((1, 1, wav.shape[0],)).to(device)

```

```

7 before_y_g_hat, y_g_hat = generator(wav, with_postnet)
8 audio = before_y_g_hat.reshape((before_y_g_hat.shape[2],))
9 audio = audio * MAX_WAV_VALUE
10 audio = audio.cpu().numpy().astype('int16')
11 ...
12 write(output_file, hp.audio.sampling_rate, audio)

```

由于HiFi-GAN是非因果模型，因此在推断阶段进行了波形的音量归一化(代码3-4行)，同样的操作也设置在训练阶段。

除了GAN之外，变分自编码器(VAE)、流(Flow)模型和扩散模型(diffusion model)等生成模型同样被应用在语音增强任务中。如VAE将纯净语音表征压缩至低维潜在变量(latent variable)用于提取纯净语音的先验分布，而后联合非负矩阵分解(NMF)等方法建模的噪声模型从而为降噪算法提供语音和噪声方差信息。这类算法在无监督学习中更为重视，由于最初其指利用纯净语音进行训练，因此VAE对带噪信号的鲁棒性问题是其重点研究的方向之一。与GAN和VAE不同，流模型利用可逆的神经网络直接建模输入数据和生成数据之间一对一的确定关系，直接(tractable)建模而非近似推断出样本的概率密度，其典型代表是利用标准化流(Normalizing Flow, NF)。通过最大化带噪语音条件下得到纯净语音的概率，训练阶段纯净语音和带噪语音通过一系列可逆变换(流模型)映射到高斯分布，在推理过程将该高斯分布和带噪信号通过流模型的逆过程即可得到增强的语音。而扩散模型受非平衡热力学启发，训练阶段作为一个前向扩散过程，在时域或频域逐步向纯净语音中注入随机噪声得到带噪信号；而后通过逆扩散过程生成增强语音。

3.3 小结

本章介绍了混合式语音增强算法的思路，信号处理和深度学习相结合的语音增强算法无疑在低复杂度和强鲁棒性上具有潜在的优势。如何巧妙地将二者融合以达到可观的性能仍值得进一步探究。

生成式模型则从建模纯净语音分布的角度重新审视了语音增强任务，尽管这类算法大多从语音合成领域迁移而来且大多数发展缓慢，但也许其在无监督、通用阵型和极低复杂度限制等早期语音增强未关注的方面的表现尤可期待。

Chapter 4

基于时域的语音增强

4.1 基于波形映射的时域语音增强模型

基于时域的语音增强算法的目的同样在于规避相位估计困难的问题，尽管早在1988年语音增强模型就开始了直接对语音时域波形处理的尝试，然而目前流行的时域语音增强算法更多依赖于语音合成和语音分离算法的影响。如在3提及的SEGAN的生成器采用了语音合成中提出的时域模型WaveNet：一维卷积堆叠而成的编码器和一维反卷积堆叠的解码器组成一个U-Net架构分别用于提取波形特征和直接映射恢复波形。自然地，之前在1、2中提及的GLU和encoder-LSTM-decoder架构可以很自然的应用其中，无非是将二维卷积替换为一位卷积而已，DEMUCS便是其中的代表[?]。其架构由“一维卷积-ReLU-点卷积(point convolution)-门控单元”和“点卷积-门控单元-一维反卷积”分别组成编码层和解码层，若干编码层组成的编码器和若干解码层组成的解码器之间是两层LSTM层用于时序建模，skip connection连接对应的编码层和解码层以缓解梯度消失问题，其网络架构如图4.1所示：其代码见

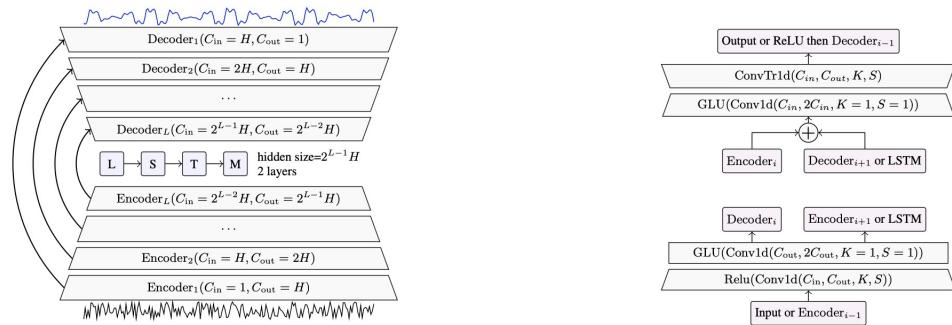


Fig. 4.1 DEMUCS框图[?]

于DEMUCS，这里只对其网络架构部分进行展示：

```

# demucs.py lines 49-195
class Demucs(nn.Module):
    @capture_init
    def __init__(self, ...):
        ...
        self.encoder = nn.ModuleList()
        self.decoder = nn.ModuleList()
        activation = nn.GLU(1) if glu else nn.ReLU()
        ch_scale = 2 if glu else 1

    for index in range(depth):
        encode = []
        encode += [
            nn.Conv1d(chin, hidden, kernel_size, stride),
            nn.ReLU(),
            nn.Conv1d(hidden, hidden * ch_scale, 1),
            activation,
        ]
        self.encoder.append(nn.Sequential(*encode))

        decode = []
        decode += [
            nn.Conv1d(hidden, ch_scale * hidden, 1),
            activation,
            nn.ConvTranspose1d(hidden, chout, kernel_size,
                             stride),
        ]
        if index > 0:
            decode.append(nn.ReLU())
        self.decoder.insert(0, nn.Sequential(*decode))
        ...

        self.lstm = BLSTM(chin, bi=not causal)
        if rescale:
            rescale_module(self, reference=rescale)

    def forward(self, mix):
        if mix.dim() == 2:
            mix = mix.unsqueeze(1)

        if self.normalize:
            mono = mix.mean(dim=1, keepdim=True)
            std = mono.std(dim=-1, keepdim=True)
            mix = mix / (self.floor + std)
        else:
            std = 1
        length = mix.shape[-1]
        x = mix
        x = F.pad(x, (0, self.valid_length(length) - length))
        if self.resample == 2:
            x = upsample2(x)
        elif self.resample == 4:

```

```

50     x = upsample2(x)
51     x = upsample2(x)
52     skips = []
53     for encode in self.encoder:
54         x = encode(x)
55         skips.append(x)
56         x = x.permute(2, 0, 1)
57         x, _ = self.lstm(x)
58         x = x.permute(1, 2, 0)
59     for decode in self.decoder:
60         skip = skips.pop(-1)
61         x = x + skip[..., :x.shape[-1]]
62         x = decode(x)
63     if self.resample == 2:
64         x = downsample2(x)
65     elif self.resample == 4:
66         x = downsample2(x)
67         x = downsample2(x)
68
69     x = x[..., :length]
70     return std * x

```

其中`rescale_module(...)`是对卷积层权重的均值方差归一化初始化，时域波形在进入网络前，可以看到进行了标准化以及sinc插值重采样，而后就是标准的编码器-LSTM-解码器逐层运行的部分。如果一定要与之前的频域算法对应，可以讲这一类算法视为基于时域波形映射的方法。

4.2 基于变换域掩蔽的时域增强模型

另一类算法则将时域增强过程与基于时频掩蔽的增强方法对齐，分为：将波形转换到变换域、在变换域估计掩蔽以及将变换域特征转换回时域波形三个过程。按照上述过程，基于时频掩蔽的增强方法可以视为该种方法的特例——变换基函数选用傅立叶变换基的情况。

这类算法中最具代表性的毋庸置疑是Conv-TasNet[?]。尽管该算法最初提出时主要针对语音分离问题，由于相当长的一段时间该类算法在分离任务中相比时频掩蔽算法的显著优势，这类算法自然被人验证其在语音增强任务上的可行性[50]。

如前文所述，Conv-TasNet包含三个过程：将波形转换到变换域、在变换域估计掩蔽以及将变换域特征转换回时域波形，这三个过程分别被称为编码器、分离器和解码器，如图4.2所示。其中编码器和解码器都由一位卷积(之后发展成多层一维卷积)组成，分离器则可以是之前成功应用于语音增强和分离任务上的模块，比如这里的时域卷积网络(TCN)。

```

1 # conv_tasnet.py lines 10–81
2 class TasNet(nn.Module):
3     def __init__(self, ...):
4         super(TasNet, self).__init__()
5         ...

```

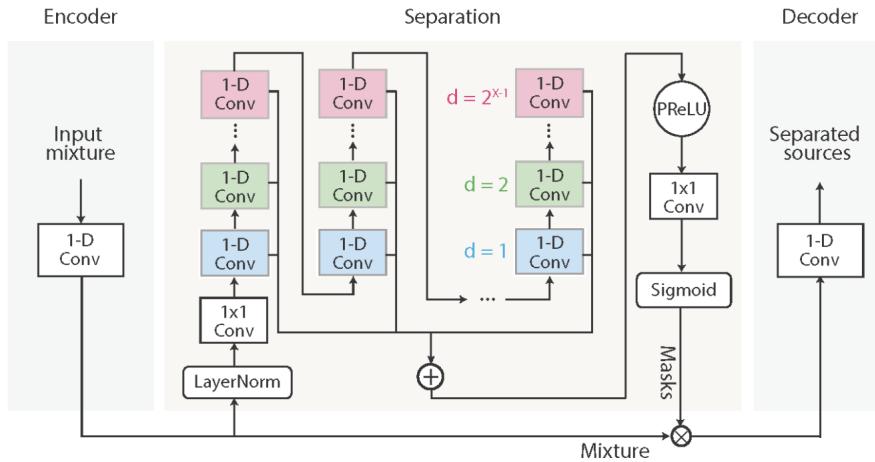


Fig. 4.2 Conv-TasNet框图[?]

```

6   # input encoder
7   self.encoder = nn.Conv1d(1, self.enc_dim, self.win, bias=
8     False, stride=self.stride)
9
10  # TCN separator
11  self.TCN = models.TCN(self.enc_dim, self.enc_dim*self.
12    num_spk, self.feature_dim, self.feature_dim*4,
13      self.layer, self.stack, self.kernel
14      , causal=self.causal)
15
16  self.receptive_field = self.TCN.receptive_field
17
18  # output decoder
19  self.decoder = nn.ConvTranspose1d(self.enc_dim, 1, self.
20    win, bias=False, stride=self.stride)
21
22  def pad_signal(self, input):
23    ...
24
25  def forward(self, input):
26    # padding
27    output, rest = self.pad_signal(input)
28    batch_size = output.size(0)
29
30    # waveform encoder
31    enc_output = self.encoder(output) # B, N, L
32
33    # generate masks
34    masks = torch.sigmoid(self.TCN(enc_output)).view(
35      batch_size, self.num_spk, self.enc_dim, -1) # B, C,
36      N, L

```

```
31     masked_output = enc_output.unsqueeze(1) * masks # B, C,
32             N, L
33
34     # waveform decoder
35     output = self.decoder(masked_output.view(batch_size*self.
36         num_spk, self.enc_dim, -1)) # B*C, 1, L
37     output = output[:, :, self.stride:-(rest+self.stride)].
38         contiguous() # B*C, 1, L
39     output = output.view(batch_size, self.num_spk, -1) # B,
40             C, T
41
42     return output
```

这类网络在语音分离任务提出时展示了一下三大优势：首先，时域增强避免了与信号在频域之间的转换相关的计算。其次，由于底层的DNN是从原始样本中训练出来的，它有可能学会提取更适合语音增强特定任务的特征。最后，基于T-F表示的短时处理要求帧大小大于某些阈值以获得足够的光谱分辨率，而在时域处理中帧大小可以设置为任意值。不过在语音增强任务中噪声的丰富多变以及混响等乘性噪声对于基于数据驱动的变换器(编码器和解码器)的性能(尤其在听感上)和鲁棒性上无疑极具挑战，这可能也是历届DNS Challenge排名靠前的方案大多选用频域算法的原因。但不可否认的是，该类算法的出现无疑促进了低延时语音增强算法[51, 52]和短帧长STFT语音增强算法的研究[53](以上两个问题非常有趣，限于篇幅这里不会展开，但推荐感兴趣的读者查阅对应的参考文献)。

Chapter 5

基于解耦的语音增强

基于解耦的语音增强技术严格来说仍可归类于基于幅度谱、复数谱、时域或者生成式的语音增强方法，然而由于其近年来的迅猛发展和卓越性能，这里将其单独作为一个章节进行介绍。其受到关注的时间较短，以至于目前对其命名以及如何分类、是否应当单作一类都尚未成为共识。

既然决定单列一章，那这里不得不斗胆厘清这类算法的命名、定义以及分类，以上几点均一家之言，不过涉及的工作却是确确实实可以相信的。由于本章介绍的算法无一不是将语音增强任务分解为多个部分“各个击破”，因此这里将其命名为基于解耦的语音增强算法。区别于之前介绍的语音增强算法，这里将把语音增强任务按照某种概念进行解耦、分别用不同网络模块完成每个解耦后子任务的语音增强方法定义为基于任务解耦的语音增强算法。这里的概念可以是某种语音质量评价指标(如SNR)、某种对语音成分的理解(如语音包含幅度谱和复数谱残差)以及对语音增强任务的理解(如语音增强可以理解为去除带噪的时频点和再生失真的时频点)等。根据其依据的概念将其进行分类，并分别在以下各节进行介绍。

5.1 基于评价指标解耦的语音增强算法

传统语音增强算法常会为增益函数设置下限以减轻过度的语音失真，这是用过降低目标语音的信噪比从而达到语音失真和噪声抑制能力的折衷。最初基于幅度谱的深度学习语音增强研究同样遇到了类似的困境，模型能够带来的信噪比增益是有限的，当触及过低信噪比的带噪语音时降噪效果亟需改善。认知科学中的程式化学习概念似乎为改善这一问题带来了可能，其思想是通过一个一个的、简单的任务开始而后逐渐增加难度，子任务的解决将为克服更难的任务带来帮助，相比直接去克服一个高难度的任务更加容易。基于此，Gao等人提出了一种基于信噪比的渐进学习框架，将直接从带噪语音映射到纯净语音的过程分解为多个阶段，以逐步改善目标的信噪比。思路如图5.1所示。

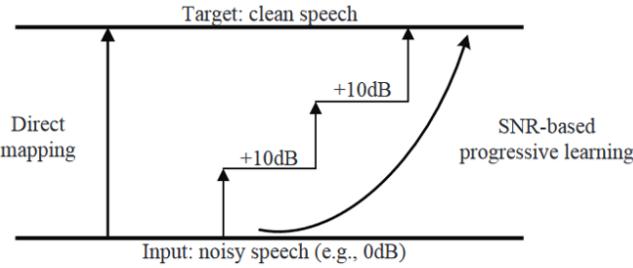


Fig. 5.1 基于信噪比的渐进学习框架思路[18]

其网络架构如图5.2所示，输入特征采用带噪对数功率谱特征，对应的信噪比分别为10 dB、20 dB和inf(纯净语音)的混合语音对数功率谱分别作为三个阶段的映射目标。在训练过程中，反向传播的梯度会影响从对应映射目标到第一层之间所有神经元的参数更新，梯度定义为 $\nabla = \frac{\partial \mathcal{L}_3}{\partial (\mathbf{W}^l, \mathbf{b}^l)} \Big|_{1 \leq l \leq L_3+1} + \alpha_2 \frac{\partial \mathcal{L}_2}{\partial (\mathbf{W}^l, \mathbf{b}^l)} \Big|_{1 \leq l \leq L_2+1} + \alpha_1 \frac{\partial \mathcal{L}_1}{\partial (\mathbf{W}^l, \mathbf{b}^l)} \Big|_{1 \leq l \leq L_1+1}$ ，其中损失函数 \mathcal{L}_i 被设为MSE谱距离损失。最后系统将模型预测的三个结果平均作为最终增强语音。

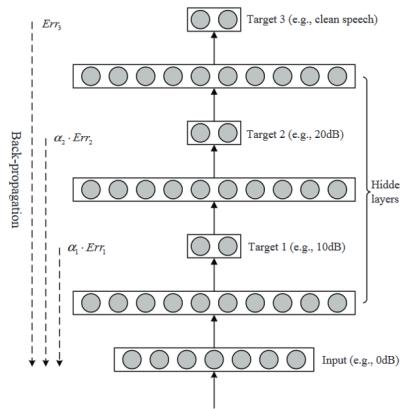


Fig. 5.2 基于信噪比的渐进学习网络结构[18]

随后，该框架中的全连接层被LSTM层代替，同时密集连接用于传递每个阶段间的信息，每个阶段的输入特征和估计结果被拼接在一起作为下一个阶段的输入。这一结构也成为后续基于任务解耦的多阶段语音增强框架的常用结构，如图5.3所示。

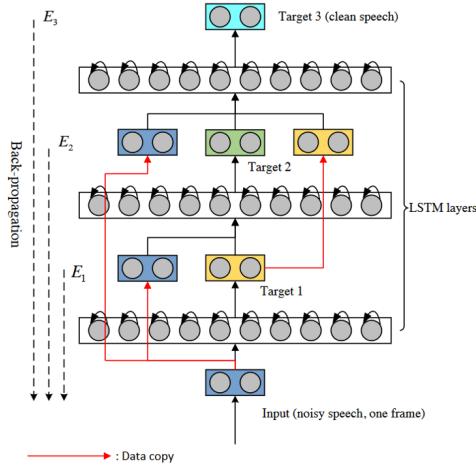


Fig. 5.3 基于信噪比的渐进学习网络结构[19]

相比于显式定义各阶段映射目标的信噪比，J. Li等人[20]认为参数量和计算量较小的增强模型本身的增强性能就十分有限，即使每个阶段都采用纯净语谱作为目标，渐进学习依旧可以完成由粗到细的增强过程，随着阶段数的增加，残留噪声和语音失真得到逐步修复。此外，该框架沿用GCRN估计cIRM进行复数谱增强，输入频谱被拆解为多个子带后沿特征通道维拼接以解决语谱高低频之间能量动态范围过大而导致的高频弱能量区域语音失真问题。

5.2 基于语音成分解耦的语音增强算法

5.2.1 基于幅度相位补偿的语音增强算法

在第2章中就可以发现，基于复数谱的语音增强算法穷尽努力只为解决除幅度谱外的其他部分。传统的相位重构算法同样启示着我们幅度谱信息如果作为先验将有助于相位信息的重构。按照启发式学习的思路，幅度谱估计作为相对简单的任务可以优先完成，估计的幅度谱信息同样将对后续整体增强语音的预测有帮助，因此可以将第二阶段定义为从估计幅度谱到最终纯净语音的估计过程(即除幅度谱外的估计过程，由于第一阶段的幅度谱估计难免有误差，因此第二阶段不仅是对相位谱的修正，还包括对幅度谱误差的修正)。参考图5.1，可以将上述过程表示为图5.4。

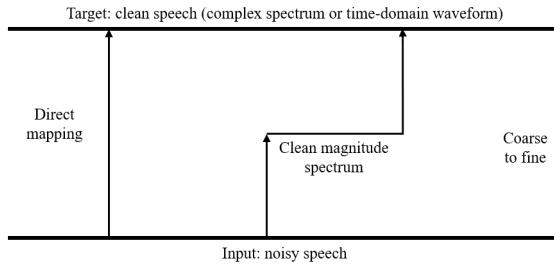


Fig. 5.4 基于语音成分解耦的语音增强算法框架

由于幅度谱估计技术相对成熟且鲁棒性较好，因此这类算法首先完成幅度谱的估计。第二阶段整体上都是基于在幅度谱估计的基础上进行最终校正的目的设计的(基于相位估计的语音增强算法似乎证明：良好估计的幅度谱对相位估计的是有利的)，只是实现方式上存在区别——采用时域模型还是复数谱模型。无论实现方式如何，其核心目的都是为了解决直接进行复数谱语音增强导致的幅度谱优化补偿问题[24]。这类算法通过首先完成幅度谱估计从而限制网络的求解空间，从而减轻这种补偿作用，如图5.6。

幅度谱优化补偿问题是在分析使用实部虚部(RI)MSE损失函数优化语音模型时发现的幅度-相位补偿现象，如图5.5所示。由于相位估计的难度较大，因此估计的结果与纯净语音之间仍存在一定的相位差。假设相位差一定，则此时根据RI-MSE最优的估计是纯净语音在指定相位差方向上的投影 \hat{S}^{RI} ，如图5.5(a)，此时幅度估计的误差是 \hat{S}^{RI} 端点到圆周的距离。而如果只对幅度谱进行MSE损失优化，此时最优的估计 \hat{S}^{mag} 则与纯净语音的幅度相同，这便造成了在复数谱优化过程中为了优化相位导致幅度谱估计的误差。更严重的结果发生在增强语音和纯净语音之间的相位差超过 $\frac{\pi}{2}$ 的时频点上，此时的最优幅度估计将退化为0，如5.5(b)。

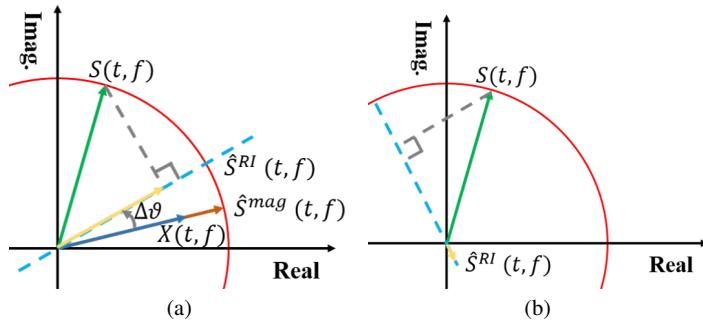


Fig. 5.5 幅度相位补偿问题

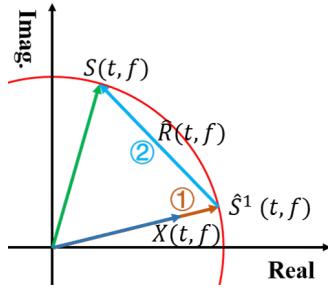


Fig. 5.6 基于语音成分解耦的语音增强算法思路

2020年N. L. Westhausen等人[21]提出DTLN利用时域增强的方法在第二阶段隐式地完成相位的恢复(毫无疑问幅度谱也会受到一定的调整), 其在两个阶段分别利用两种互补的信号变换方式——固定基函数的STFT变换和可学习基函数的编码变换, 从而完成语音增强。整个过程可以看做是1.1和??的有机结合, 如图5.7所示并可表示如下:

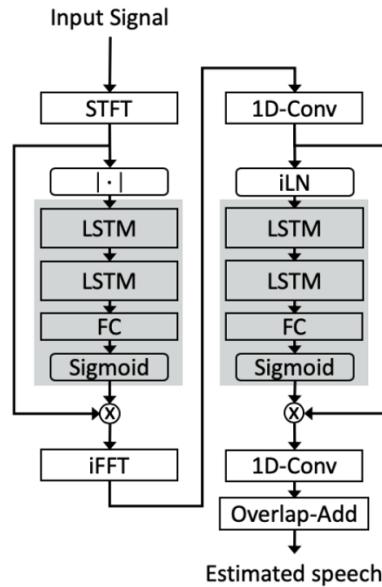


Fig. 5.7 DTLN算法框架[21]

$$\begin{aligned}\hat{s}^1 &= \mathcal{F}^{-1}(M \cdot |\mathcal{F}(x)| \cdot e^{\angle \mathcal{F}(x)}), \\ \hat{s} &= \mathcal{D}(m \cdot \mathcal{E}(\hat{s}^1)).\end{aligned}\quad (5.1)$$

式中， \mathcal{F} 和 \mathcal{F}^{-1} 为STFT变换及其逆变换， \mathcal{E} 和 \mathcal{D} 分别是利用一维卷积可学习编码器和解码器。式5.1中的 M 和 m 分别是由两个“LSTM-LSTM-FC (Sigmoid)”网络估计得到的滤波器。两个阶段的网络结构相同，定义在seperation_kernel方法中：

```

1 # DTLN_model_lines_264-285
2 def seperation_kernel(self, num_layer, mask_size, x,
3     stateful=False):
4     for idx in range(num_layer):
5         x = LSTM(self.numUnits, return_sequences=True, stateful=
6             stateful)(x)
7         if idx < (num_layer - 1):
8             x = Dropout(self.dropout)(x)
9         mask = Dense(mask_size)(x)
10        mask = Activation(self.activation)(mask)
11    return mask

```

式5.1所示的过程的代码表示如下，STFT, iSTFT和OLA过程分别被定义在同一个类的stftLayer、ifftLayer和overlapAddLayer方法中。另外在第二阶段将波形变换到可学习的变换域后，采用了instant layer normalization(iLN)¹进行标准化：

```

1 # DTLN_model.py_lines_322-366
2 def build_DTLN_model(self, norm_stft=False):
3     # input layer for time signal
4     time_dat = Input(batch_shape=(None, None))
5     # calculate STFT
6     mag, angle = Lambda(self.stftLayer)(time_dat)
7     ...
8     mag_norm = mag
9     # predicting mask with separation kernel
10    mask_1 = self.seperation_kernel(self.numLayer, (self.blockLen
11        //2+1), mag_norm)
12    # multiply mask with magnitude
13    estimated_mag = Multiply()([mag, mask_1])
14    # transform frames back to time domain
15    estimated_frames_1 = Lambda(self.ifftLayer)([estimated_mag,
16        angle])
17    # encode time domain frames to feature domain
18    encoded_frames = Conv1D(self.encoder_size, 1, strides=1,
19        use_bias=False)(estimated_frames_1)
20    # normalize the input to the separation kernel
21    encoded_frames_norm = InstantLayerNormalization()(encoded_frames)
22    # predict mask based on the normalized feature frames
23    mask_2 = self.seperation_kernel(self.numLayer, self.
24        encoder_size, encoded_frames_norm)
25    # multiply encoded frames with the mask
26    estimated = Multiply()([encoded_frames, mask_2])
27    # decode the frames back to time domain
28    decoded_frames = Conv1D(self.blockLen, 1, padding='causal',
29        use_bias=False)(estimated)

```

¹ 代码实现可参考https://github.com/breizhn/DTLN/blob/master/DTLN_model.py#L586

```

25     # create waveform with overlap and add procedure
26     estimated_sig = Lambda(self.overlapAddLayer)(decoded_frames)
27     # create the model
28     self.model = Model(inputs=time_dat, outputs=estimated_sig)

```

其中stftLayer、ifftLayer和overlapAddLayer方法的代码实现分别为：

```

1 # DTLN_model.py lines 203–218
2 def stftLayer(self, x):
3     # creating frames from the continuous waveform
4     frames = tf.signal.frame(x, self.blockLen, self.blockShift)
5     # calculating the fft over the time frames. rfft returns NFFT
6     # /2+1 bins.
7     stft_dat = tf.signal.rfft(frames)
8     # calculating magnitude and phase from the complex signal
9     mag = tf.abs(stft_dat)
10    phase = tf.math.angle(stft_dat)
11    return [mag, phase]
12
13 # DTLN_model.py lines 239–250
14 def ifftLayer(self, x):
15     s1_stft = (tf.cast(x[0], tf.complex64) * tf.exp((1j * tf.
16         cast(x[1], tf.complex64))))
17     return tf.signal.irfft(s1_stft)
18
19 # DTLN_model.py lines 253–260
20 def overlapAddLayer(self, x):
21     return tf.signal.overlap_and_add(x, self.blockShift)

```

除了通过时域模型实现在幅度谱估计的基础上完成相位恢复外，A. Li等提出了一种完全在时频域上实现的上述两阶段框架并命名为CTSNet，从而规避了时域信号模态不清晰且对混响和噪声泛化能力有限的问题[25]。与DTLN[21]的第一阶段相似，幅度谱首先被增强，增强的幅度谱 $|\widehat{S}^1(t, f)|$ 与带噪语音的相位 $\angle X(t, f)$ 耦合得到增强语音的粗估计 $\widehat{S}^1(t, f) = |\widehat{S}^1(t, f)| \cdot e^{j\angle X(t, f)}$ 。接下来还需要进行修正的部分，是一个复数谱残差 $R(t, f) = S(t, f) - \widehat{S}^1(t, f)$ 。由于该残差实部和虚部的模式与语音复数谱的实虚部模式十分相似，这使得2.1中基于复数谱映射的模型能够成功完成对残差量的映射。此外，与[19]类似，在第二阶段的输入部分，包含了带噪复数谱的data copy以及第一阶段得到的增强语音粗估计。上述算法执行过程如图5.8所示并可表示为如下形式：

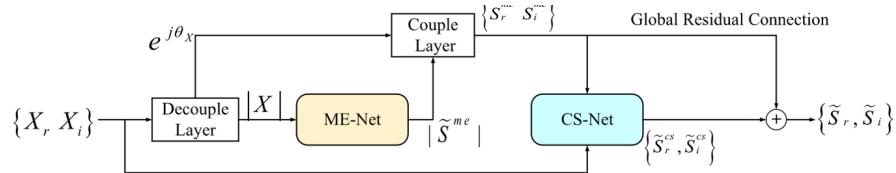


Fig. 5.8 CTSNet算法框架[25]

$$\begin{aligned}\widehat{S}^1 &= \mathcal{G}_1(|X|) \cdot e^{j\angle X}, \\ \widehat{S} &= \mathcal{G}_2([\widehat{S}^1; X]) + \widehat{S}^1.\end{aligned}\quad (5.2)$$

随后，在CTSNet的基础上又发展出一种联合去噪去混响的解耦框架[26]，其中值得注意的一点是一种压缩谱特征的设计，该框架对特征和映射目标的幅度谱进行了幂律压缩，即 $X' = |X|^c \cdot e^{j\angle X}$ ，并取得了显著的性能提升。这种设计方式与2.2中提及的[12]所使用的损失函数作用类似。

近期，上述基于语音成分解耦的多阶段方式开始逐渐被一种单阶段并行网络代替，这些方法试图揭示基于语音成分解耦的有效性并不在于多阶段网络的结构，而在于对语音成分的解耦。另外，这类网络只需进行一次训练，在训练阶段更加快捷方便。[27]作为CTSNet的单阶段网络形式被提出，尽管它改用了[26]中提出的压缩谱特征。相比于CTSNet，该算法采用了更先进的U²-Net编码器去提取复数谱特征，而后利用两个时序卷积网络(Temporal Convolutional Network, TCN)分别进行幅度谱和复数谱残差的估计，最后显式地将二者相加后得到估计结果。

根据[24]中分析的幅度相位补偿效应，除了先约束幅度再调节相位，先在复数谱上逼近后再调节幅度上的补偿效应也未尝不可。[28]和[29]可以分别看做这一思路的多阶段和单阶段并行版本，尽管二者基于不同的出发点几乎同时提出，网络结构上也没有如[25]和[27]一般的连续性。第一阶段采用基于极坐标系复值掩蔽的语音增强方法得到复值掩蔽 \widehat{M}_C^{mag} 和 \widehat{M}_C^{phase} ，而后第二阶段估计得到一个用于幅度补偿的掩蔽 \widehat{M}_R 进一步修正幅度谱得到最终的增强语音 $\widehat{S} = \mathcal{G}(\widehat{M}_R, \widehat{M}_C^{mag}) \cdot |X| \cdot e^{j(\angle X + \widehat{M}_C^{phase})}$ 。其中，[24]和[29]对 $\mathcal{G}(\cdot, \cdot)$ 的定义略有不同，分别是 $\mathcal{G}(\widehat{M}_R, \widehat{M}_C^{mag}) = \widehat{M}_C^{mag} + \widehat{M}_R \cdot \widehat{M}_C^{mag}$ 和 $\mathcal{G}(\widehat{M}_R, \widehat{M}_C^{mag}) = 0.5 \cdot (\widehat{M}_R + \widehat{M}_C^{mag})$ 。

5.2.2 基于谱包络-周期性语音成分解耦的语音增强方法

除此之外，继承自3.1中RNNoise类的算法得到进一步发展。在遵循将语谱分解为包络和细节两部分的解耦原则下，[30]将用于修复谱细节以增强谐波成分的基频滤波阶段通过神经网络实现。与PercepNet类似，首先利用神经网络估计ERB尺度的增益函数，经过插值后对频域的谱包络进行滤波。而后，第二阶段通过网络估计低频部分的复值滤波器系数，通过deep filtering操作实现对低频周期性语音成分的谐波进行增强的目的，deep filtering定义在2.4， K 取0。同时估计一个权重因子 α 去控制基频滤波仅作用在浊音段 $\widehat{S}(t, f) = \alpha(f) \cdot \widehat{S}^2(t, f) + (1 - \alpha(f)) \cdot \widehat{S}^1(t, f)$ 。上述过程如图5.9所示。

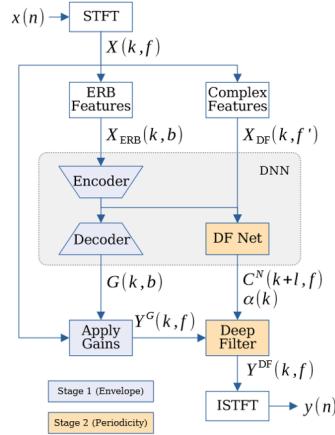


Fig. 5.9 DeepFilterNet算法框架[30]

和[27]类似，一种双分支版本随后被提出[31]。带有信息融合的双编码器双解码器结构分别编码ERB尺度特征和低频复数谱，并生成ERB尺度的增益因子和仅作用于低频的deep filtering系数。通过这种联合优化方式，损失函数能够只作用于最终增强的语谱，从而避免了[30]中对浊音段权重因子 $\alpha(f)$ 的估计。整个过程可表示为：

$$\begin{aligned} U(t) &= \mathcal{E}_{en}(\mathcal{F}_{erb}(|X|(t, f)), \mathcal{F}(X(t, f_{low}))), \\ \widehat{S}^1(t, f) &= \text{interp}(\mathcal{D}_{erb}(U(t))) \cdot |X|(t, f) \cdot e^{j\angle X(t, f)}, \\ \widehat{S}(t, f) &= \text{DeepFilter}(\mathcal{D}(U(t)), \widehat{S}^1(t, f)). \end{aligned} \quad (5.3)$$

5.2.3 基于子带分解的语音增强算法

子带(sub-band)分解思想被广泛应用在传统语音增强算法当中[43]，基于深度学习的子带分解在前文提及的[20]中即有体现，不过近年来使子带分解思想再度得到重视的是[44]——神经网络被用于基于信号统计性和局部谱特性建立从子带幅度谱到中心频点cIRM的映射函数，该映射函数通过子带内语音的统计特性一般相对噪声变化更快的特性区分语音和噪声。由于在推断阶段所有子带共享同一个网络模型，从而节省了模型的参数量。随后，为了解决子带模型对全局谱特性建模缺失问题，FullsubNet[45]融合全带(这里的全带指的是未分子带前的频带，是相对于子带而言的概念)和子带各自的优势——即谱特性的全局建模和局部建模，实现全带增强和子带增强的互补。具体而言，当前帧的幅度谱经过全带模型进行全带增强后与上下文的带噪幅度谱形成的子带拼接，这组拼接后的特征作为[44]提出的子带模型的输入，最终获得对应子带中心频率的cIRM。整个过程如图5.10所示。

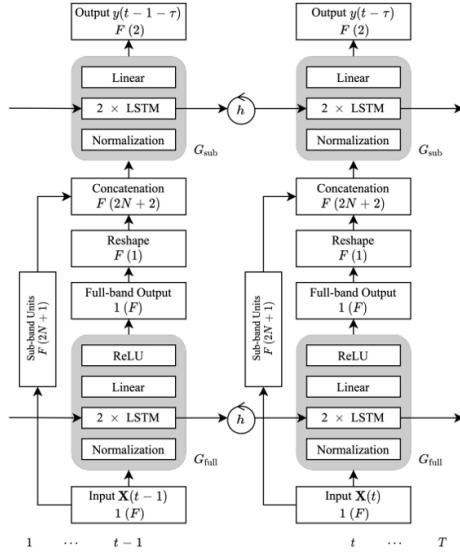


Fig. 5.10 FullsubNet框图[45]

尽管子带分解的语音增强方法在DNS Challenge中与前几名的算法在性能上尚存在差距，但由于其结构适合对高采样率的音频进行降噪，因此近年来在全带(48 kHz采样率)语音增强上得到了继承和发展。

5.3 基于语音增强任务理解解耦的语音增强算法

除了前文提及的两种方式，还出现了一类根据其对语音增强理解对其进行解耦的算法，这类算法主要包括将语音增强过程分解为噪声抑制和语音恢复两个阶段，和将语音增强过程分解为提取纯净语音声学参数和利用声码器基于参数合成纯净语音波形两个阶段。

5.3.1 基于噪声抑制和语音恢复的语音增强算法

“噪声抑制”和(狭义上的)“语音增强”都用于描述从带噪信号中消除噪声成分并提取语音成分的过程，然而他们的命名却显示了两种不同的思路(尽管这两种思路理想情况下的结果应该相同)——从频域掩蔽的角度分别可以看作——找到噪声分布的时频点并将其置为零；以及找到语音分布的时频点并将其抠出。不幸的是，语音和噪声的分布不可避免地存在交集。执行前者则会带来语音的失真，而执行后者则会带来噪声的残留。因此，自然地产生了这样一种思路：先将噪声成分去除，在这一过程中实数增益函数必然会带来

该时频点上语音成分的抑制，复数增益函数也由于估计误差不可避免地会带来该时频点上语音的畸变；因此接下来对这些失真的语音根据时频上下文信息重新绘制出在这些时频点上最可能的语音频谱，从而得到质量更高的处理结果。值得注意的是，这两种思想在传统语音增强技术中都得到了发展：前者包括最常见的谱减法、维纳滤波和基于统计模型的方法等，这类算法估计了一个值域在0到1之间增益函数对语谱进行滤波，由信噪比确定的增益函数势必在低信噪比的时频点加强抑制，如果这些时频点上包含语音成分，失真则是必然[32]；后者的代表是谐波再生方法[33]，这类算法利用非线性函数重构的方法解决了前者造成的谐波部分的语音失真问题。图5.11展示了这两类方法的处理过程。

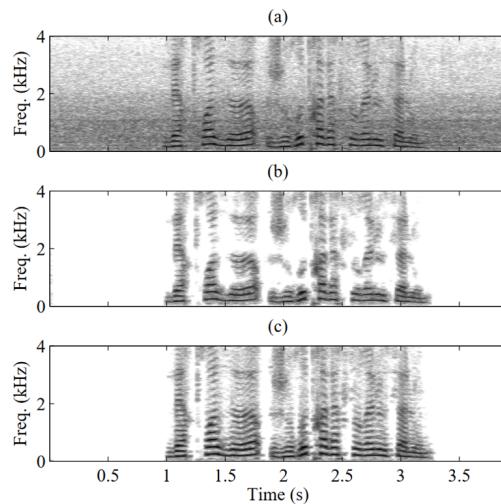


Fig. 5.11 (a) 带噪语谱; (b) “噪声抑制” (c) “语音增强” [33]

2019年这种思路被神经网络化：激进的噪声抑制和语音恢复再生的架构被提出，前者采用基于时频掩蔽的LSTM，第二阶段则采用在图像恢复中常用的直接映射的CED，整个框架完全在复数谱上进行。两个阶段分别采用时频掩蔽和谱映射的原因大约如1提及的那样，分别对应激进的降噪和语谱的脑补。2020年提出的“先掩蔽再补全(masking and inpainting)”框架[4]更是完美践行了上述思路：对于低信噪比的时频点，与其试图从中寻找语音的结构不如完全放弃，而是利用与之相关的谱特性补全缺失的谱，从而降低网络学习的难度。整个过程如图5.12所示，掩蔽阶段直接采用IBM以完全去除噪声，之后的补全阶段网络根据上下文信息将千疮百孔的语谱补全。

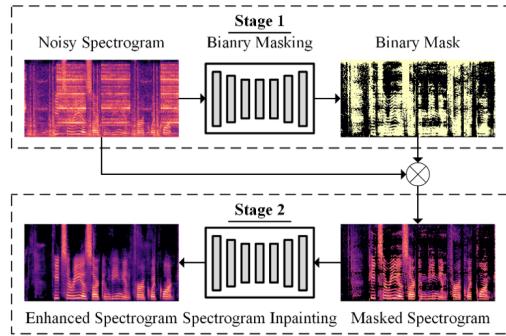


Fig. 5.12 “先掩蔽再补全”框架 [4]

然而，上述算法仍旧停留在对语音模式的建模(抑或对信噪比的建模)中。既然谈到噪声抑制，显式的噪声信息对语音增强的帮助势必应该得到讨论。在语音增强早期，噪声谱估计就是主要任务，谱减法即是如此。由于噪声种类之多样，完全准确估计噪声谱并将其从带噪语谱中去除几乎是不现实的。噪声谱应该发挥的作用，似乎应该作为一种信息以辅助模型进行语音增强。这是因为高信噪比时可以轻易获得语音的谱结构进行增强，而低信噪比时利用噪声谱特性进行降噪则更加容易。2020年，R. Xu等[34]显式地估计了噪声谱，估计的噪声谱与带噪谱一同作为网络的输入，估计得到降噪的语音。而噪声估计的方式，可以看作是一种“先掩蔽再补全”的过程。这一过程参考了传统噪声谱估计的一种方法：通过VAD得到静音段，利用静音段的谱特性补全整个语句的噪声谱。明显地，这种利用时间信息作为上下文补全谱的方式不得不设计为非因果模型，而单凭静音段的噪声谱很难估计长时间语音段内的高度非平稳噪声谱。但至少该工作展示了噪声谱估计的可能，以及噪声谱对于模型增强的有效性。2021年，W. Liu等[35]则联合了5.2.1的幅度-相位补偿思想和5.3.1的噪声抑制-语音恢复思想。如图5.13所示，首先采用多任务学习的方式同时估计噪声和语音的幅度谱，利用语音和噪声的互补性估计噪声的方式避免了上文提及的噪声谱估计问题；而后，和[25]一样增强幅度谱以及带噪语谱共同作为下一阶段的输入，和[34]一样估计的噪声谱也作为下一阶段的输入，最终预测得到增强的复数谱。第二阶段中，同时完成了噪声信息的引入指导和幅度谱基础上复数谱残差的修正。同年，C. Zheng等[36]在噪声估计阶段采用继承自PHASEN[13]的双分支结构同时估计语音谱和噪声谱，之间利用信息交互促进谱估计的准确性；第二阶段则并未如[34]和[35]类似将噪声估计作为一种辅助信息(或者说，噪声谱信息的辅助作用是在第一阶段利用信息交互体现的)，而是直接将二者的结果进行线性加权，是基于深度学习的语音增强模型和噪声谱估计更精确的谱减法结果的线性组合。

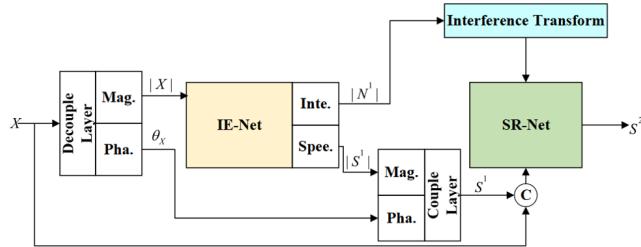


Fig. 5.13 联合幅度-相位补偿和噪声抑制-语音恢复思想的框架 [35]

5.3.2 基于参数再合成的语音增强方法

基于参数再合成的语音增强方法可以看作生成式语音增强模型的任务解耦版本。语音增强过程除了视为一种滤波任务外，也可以利用语音合成的方法生成纯净语音。因此，可以将该过程解耦为从带噪语音中提取纯净语音的声学参数，和根据声学参数生成纯净语音两个子任务，后者即是语音合成中的声码器(vocoder)；而整个过程被命名为参数再合成(parametric resynthesis)。

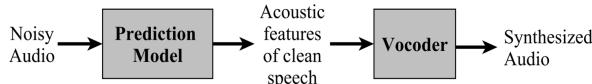


Fig. 5.14 基于参数再合成的语音增强框架 [37]

2019年，S. Maiti等[37]利用神经网络从带噪信号的对数Mel谱中预测声码器输入所需的谱包络、对数基频等参数，而后采用传统的WORLD声码器将声学参数转换为纯净语音波形。他们将这两个阶段分别称之为预测模型和声码器，整个过程如图5.14所示。而后，基于神经网络的声码器：WaveGlow、WaveNet以及LPCNet等相继被应用，预测模型输出的声学参数也随声码器的不同而有所调整(纯净语音的Mel谱、BFCC和基音周期以及相关性等)，在说话人泛化能力上得到了提升[38]。Du等[39]也提出了相似的框架，由于其采用的声码器的输入主要是纯净语音的Mel谱，因此将第一阶段称为语音增强模型并采用1中提及的编码器-时序建模-解码器架构对Mel谱进行降噪。而后采用Flowavenet声码器根据纯净语音Mel谱和高斯噪声合成出纯净语音波形，从而完成语音增强，如图所示。VoiceFixer框架[40]则将两个阶段分别命名为分析和合成模块，并证明了该框架在降噪、去混响、低分辨率和截断失真等场景下的有效性。

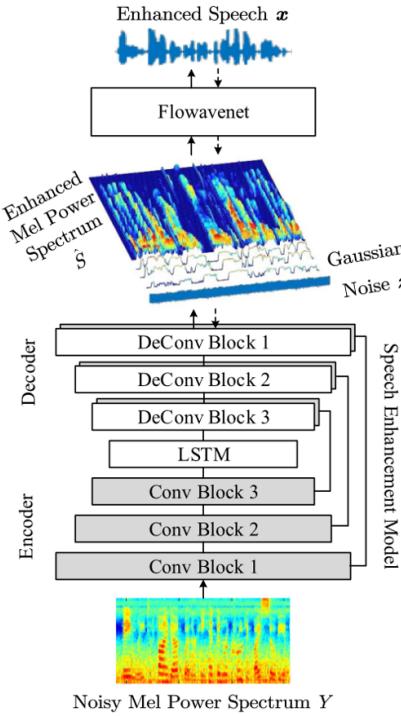


Fig. 5.15 联合语音增强和语音合成框架 [39]

5.4 小结

基于解耦的语音增强方法或受启发于课程式学习由易到难的解决思路，或继承传统的信号分解方式。纵观下来可以发现，其中很多框架的设计思路实则在多年前便被提出。只因深度学习过度亮眼的表现将人们早期的关注点集中在网络模块的设计和提出上。近年来，过往的分解方式正在再次被重视，并与成熟的深度学习语音增强技术有机结合。尽管目前尚不能断言这类算法一定相比之前单阶段的语音增强算法具有更强的性能优势，但无疑与混合式语音增强算法类似，一种成功的解耦方式——即确实将复杂的任务转化为两个或多个更简单的任务，而非转化的任务实则优化难度并未降低——将为更轻量的增强网络设计提供一种可能。另外私人的偏爱在于，放眼技术发展的长河，尽可能脱离具体网络模块细节的框架设计更可能对未来的统计学习技术提供借鉴。基于解耦的语音增强算法目前仍在发展，并在全带语音增强上也带来了帮助[41, 42]。旧有分解思路神经网络化的成功固然令人欣喜，依托数据驱动的全新解耦方式更值得期待。

Chapter 6

语音增强算法的因果性

语音增强技术根据其应用的不同可以分为实时系统和非实时系统两大类。助听辅听、耳机手机通话等是典型的实时应用场景，其中的语音增强算法必须在使用较少的未来信息甚至不使用未来信息的情况下(即模型是因果的)下完成处理，同时，算法在相应的硬件设备上的计算时间也应满足实时要求。需要注意的是，这里的实时是一个相对的概念，绝大多数情况的语音增强并不要求逐采样点的实时处理，而是指在较短延时内完成处理，而对于不同的应用，能容忍的延时也各不相同。由于不同硬件平台的算力不同，更常见的是用模型的因果性代替对其实时性的评价，并使用参数量和计算量作为是否能够在某个硬件上实时处理的参考指标。这是由于语音增强的应用场景是如此广泛、模型压缩技术仍有对模型效率改进的可能、以及芯片等计算资源的迅猛发展，因此即使某个提出的语音增强模型无法应用在某个平台上，该模型可能仍不失其价值。

语音增强处理延时的定义并不止一种，这里给出一种常见的定义：对于采用重叠相加(overlap-add)方法重构的频域语音增强算法，时延由OLA带来的时延(合成窗长)、频域变换带来的时延(帧移)以及算法接收的未来信息带来的时延三部分组成，另外算法的处理时间应不大于帧移。

为了保证模型比较的公平性以确定模型中每个模块的有效性，模型因果与否是应该明确的。然而，很遗憾的是，社区内常出现工作有意无意地忽视了这一问题，从而误导了社区对于某种特征/模块/损失函数的认识。因此，作为一本语音增强导读，有必要将这部分单列一节进行说明。

这一话题无疑是必要但敏感的，因此这里不得不表明个人态度。其一，本节的立意是尽量澄清一些社区内的误区、尽量减少初学者所走的弯路、尽量提升初学者对文章和开源代码的判断能力，而非否定以往的和未来的任何工作。其二，尽管一些工作并不严格满足因果处理，但与因果处理保持一致的baselines比较时大约是能够保证其创新部

分的有效性的。另一方面，尽管个人由于不希望看到语音前端陷入到和计算机视觉一样门槛大幅度降低、滥竽充数者比例渐高、工业界多方面压制学术界的状况，因此对开源并不持支持态度。但无疑开源者是对社区发展有着推动和帮助作用的，他们的贡献是巨大的。窃以为不宜以这种很可能无意的失误而使贡献者受到责难。我也反对在任何高速发展的学科因为小的技术失误而全面否定整个工作的行为，这对学科和个人的发展是有负面作用的(很不幸近年来这种趋势似乎是明显的)。更何况实时语音增强中大量工作并不因这一问题而失去其价值，而且语音增强在非实时场景同样也有广泛应用。

6.1 语音音量的归一化

由于说话人本身音量以及说话人距传声器距离远近的差异，语音音量大小的动态范围通常很大。为了保证语音增强模型能够处理不同音量的语音，对训练数据的处理通常是重要的。社区内对此常见的两种非因果操作是对训练和测试过程中的带噪语音和纯净语音进行能量均方根归一化(即假设语音经过了理想的自动增益控制(AGC))、以及利用整段带噪音频计算均值和方差后用作均值方差归一化(这种方式既有利用训练集的均值方差对测试语料做归一化的，又有直接测试语料计算均值和方差做归一化的，显然后者是非因果的)。以这种方式压缩了数据的动态范围，方便网络优化。替代这类非因果操作的方式有两种：一种是对输入特征进行在线归一化、另一种是在损失端进行归一化。

前者常见的有在线频率依赖的均值方差归一化，当前帧的均值方差均以指数衰减平滑的形式估计得到：

$$\begin{aligned}\mu(t, f) &= \alpha\mu(t-1, f) + (1-\alpha)|X(t, f)|, \\ \sigma^2(t, f) &= \alpha\sigma^2(t-1, f) + (1-\alpha)|X(t, f)|^2, \\ |X^{norm}(t, f)| &= \frac{|X(t, f)| - \mu(t, f)}{\sqrt{\sigma^2(t, f) - \mu^2(t, f)}}.\end{aligned}\tag{6.1}$$

上述关系式代码为：

```

1 # https://github.com/GuillaumeVW/NSNet/blob/master/
2     dataloader/wav_dataset.py lines 85 – 102
3 # mean normalization
4 frames = []
5 x_lps = x_lps.transpose(0, 1)
6 n_init_frames = self.n_init_frames
7 alpha_feat_init = self.alpha_feat_init
8 alpha_feat = self.alpha_feat
9 for frame_counter, frame_feature in enumerate(x_lps):
10 if frame_counter < n_init_frames:
11     alpha = alpha_feat_init
12 else:
```

```

12     alpha = alpha_feat
13     if frame_counter == 0:
14         mu = frame_feature
15         sigmasquare = frame_feature.pow(2)
16         mu = alpha * mu + (1 - alpha) * frame_feature
17         sigmasquare = alpha * sigmasquare + (1 - alpha) *
18             frame_feature.pow(2)
19         sigma = torch.sqrt(torch.clamp(sigmasquare - mu.pow(2),
20                                         min=1e-12)) # limit for sqrt
21         norm_feature = (frame_feature - mu) / sigma
22         frames.append(norm_feature)

```

此外，其他累积归一化方式和可学习归一化方式也被探索。

后者则是将语音能量均方根归一化的操作转移到损失端，整句语音的能量仅在训练阶段被使用，因此模型是因果的。这类方法被用于复数域语音增强模型，其损失函数被定义为[7]：

$$\mathcal{L} = \frac{1}{\sigma} (\alpha \sum_{t,f} |S - \hat{S}|^2 + (1 - \alpha) \sum_{t,f} ||S| - |\hat{S}||^2), \quad (6.2)$$

其中 σ 是指经过语音活动检测(VAD)后的语音能量。一般同时还会在训练阶段对语料音量以某种随机分布随机缩放，如在训练阶段将纯净语音和噪声的音量都服从均值为-26 dBFS，方差为10 dB的高斯分布进行随机缩放[7]，类似操作的代码见于DPCRN3：

```

1 # data_loader.py lines 163 - 192
2 gain = np.random.normal(loc=-5, scale=10)
3 gain = 10**((gain/10))
4 gain = min(gain, 3)
5 gain = max(gain, 0.01)
6 ...
7 batch_clean[i, :] = clean_s * gain
8 batch_noisy[i, :] = noisy_s * gain

```

6.2 因果模块讨论

网络层面的因果性相对更易察觉，不过这有时要求对已有的深度学习API有足够的认知。目前基于深度学习的语音增强模型常用层等。本节以Pytorch框架为例简要地对这些模块进行讨论。这里默认读者已熟悉这些模块并熟悉相关的Pytorch函数，使用其他框架的读者可做参考，结合所使用框架的文档进行核对。

1. LSTM层和GRU层：torch.nn.LSTM和torch.nn.gru中的参数bidirectional设为False(默认)即只使用历史信息；
2. 二维卷积层：需对torch.nn.Conv2d/Conv1d输入特征的时间帧维度向历史帧方向补零(时间帧方向卷积核不为1时)；

3. 二维转置卷积：需对`torch.nn.Transpose2d`输出结果的时间帧维度进行在未来帧方向截断(时间帧方向卷积核不为1时);
4. 注意力机制模块：需对点积相似度公式中的`Softmax`函数内加掩蔽项，这样虽然是因果的，但在推理时内存和运行时间仍会随帧数增加而增长。基于`chunk`或类`TransformerXL`的`attention`才有可能满足实时处理要求。
5. 标准化层：`Batch Normalization`应在推理时将模型设为`eval()`模式；`Instance Normalization`应将参数`track_running_stats`设置为`True`。
6. 池化层：因果模型中不应对时间帧维度进行池化。

References

1. 刘文举等: 基于深度学习语音分离技术的研究现状与进展, 自动化学报, 42(6), pp. 819-833, 2016.
2. Wang, et, al. *supervised speech separation based on deep learning: an overview*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 26(10), pp.1702-1726, 2018.
3. Y. Xia, et, al. *Weighted Speech Distortion Losses for Neural-network-based Real-time Speech Enhancement*, ICASSP 2020.
4. X. Hao, et, al. *Masking and Inpainting: A Two-Stage Speech Enhancement Approach for Low SNR and Non-Stationary Noise*, ICASSP 2020, pp. 6959–6963. doi: 10.1109/I-CASSP40776.2020.9053188.
5. Y. Xu, et, al. *An Experimental Study on Speech Enhancement BasedonDeepNeuralNetworks*, IEEE Signal Processing Letters, 21(1), pp. 65-68, 2014.
6. K. Tan, et, al. *A convolutional recurrent neural network for real-time speech enhancement*, INTERSPEECH 2018, pp. 3229-3233.
7. S. Braun, et, al. *Effect of noise suppression losses on speech distortion and ASR performance*, arXiv:2111.11606.
8. N. Zheng, et, al. *Phase-Aware Speech Enhancement Based on Deep Neural Networks*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 27(1), pp.63-76, 2019.
9. K. Tan, et, al. *Learning complex spectral mapping with gated convolutional recurrent networks for monaural speech enhancement*, 28, pp.380-390, 2020.
10. H. Choi, et, al. *Phase-aware speech enhancement with deep complex U-Net*, arXiv:1903.03107.
11. X. Le, et, al. *DPCRN: Dual-Path Convolution Recurrent Network for Single Channel Speech Enhancement*, Interspeech 2021, pp. 2811-2815.
12. S. Braun, et, al. *Towards efficient models for real-time deep noise suppression*, arXiv:2101.09249.
13. D. Yin, et, al. *PHASEN: a phase-and-harmonics-aware speech enhancement network*, Proceedings of the AAAI Conference on Artificial Intelligence, 34(5), pp. 9458-9465, 2020.
14. Z. Wang, et, al. *Deep Learning Based Phase Reconstruction for Speaker Separation: A Trigonometric Perspective*, ICASSP 2019, pp. 71-75.
15. S. Suhadi, et, al. *A data-driven approach to a priori SNR estimation*, IEEE Transactions on Audio, Speech and Language Processing, vol. 19, no. 1, pp. 186–195, 2011.
16. Y. Ephraim and D. Malah. *Speech enhancement using a minimum mean-square error log-spectral amplitude estimator*, IEEE Trans. Acoust. Speech Signal Process. 33 (2), pp. 443–445. 1985.
17. Y. Xia, et, al. *Low-dimensional recurrent neural network-based Kalman filter for speech enhancement*, Neural Networks, 67, pp. 131-139, 2015.
18. T. Gao, et, al. (2016). *SNR-Based Progressive Learning of Deep Neural Network for Speech Enhancement*, Interspeech 2016, pp. 3713-3717.
19. T. Gao, J. Du, L.-R. Dai, and C.-H. Lee, “Densely Connected Progressive Learning for LSTM-Based Speech Enhancement,” in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Apr. 2018, pp. 5054–5058. doi: 10.1109/I-CASSP.2018.8461861.
20. J. Li et al., “Densely Connected Multi-Stage Model with Channel Wise Subband Feature for Real-Time Speech Enhancement,” in ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, Jun. 2021, pp. 6638–6642. doi: 10.1109/ICASSP39728.2021.9413967.
21. N. L. Westhausen and B. T. Meyer, “Dual-Signal Transformation LSTM Network for Real-Time Noise Suppression,” in Interspeech 2020, Oct. 2020, pp. 2477–2481. doi: 10.21437/Interspeech.2020-2631.
22. Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, “a regression approach to speech enhancement based on deep neural networks,” IEEE/ACM Trans. Audio Speech Lang. Process., vol. 23, no. 1, Art. no. 1, Jan. 2015, doi: 10.1109/TASLP.2014.2364452.

23. Z. Du, X. Zhang, and J. Han, “A Joint Framework of Denoising Autoencoder and Generative Vocoder for Monaural Speech Enhancement,” IEEE/ACM Trans. Audio Speech Lang. Process., vol. 28, pp. 1493–1505, 2020, doi: 10.1109/TASLP.2020.2991537.
24. Z.-Q. Wang, G. Wichern, and J. Le Roux, “On the Compensation Between Magnitude and Phase in Speech Separation,” IEEE Signal Processing Letters, vol. 28, pp. 2018–2022, 2021, doi: 10.1109/LSP.2021.3116502.
25. A. Li, W. Liu, C. Zheng, C. Fan, and X. Li, “Two Heads are Better Than One: A Two-Stage Complex Spectral Mapping Approach for Monaural Speech Enhancement,” IEEE/ACM Trans. Audio Speech Lang. Process., vol. 29, pp. 1829–1843, 2021, doi: 10.1109/TASLP.2021.3079813.
26. A. Li, W. Liu, X. Luo, G. Yu, C. Zheng, and X. Li, “A Simultaneous Denoising and Dereverberation Framework with Target Decoupling,” p. 5.
27. A. Li, C. Zheng, L. Zhang, and X. Li, “Glance and gaze: A collaborative learning framework for single-channel speech enhancement,” Applied Acoustics, vol. 187, p. 108499, Feb. 2022, doi: 10.1016/j.apacoust.2021.108499.
28. T. Wang, W. Zhu, Y. Gao, J. Feng, and S. Zhang, “HGNCN: harmonic gated compensation network for speech enhancement,” arXiv:2201.12755 [cs, eess], Jan. 2022, Accessed: Feb. 10, 2022. [Online]. Available: <http://arxiv.org/abs/2201.12755>
29. Y. Fu et al., “Uformer: A Unet based dilated complex & real dual-path conformer network for simultaneous speech enhancement and dereverberation,” arXiv:2111.06015 [cs, eess], Nov. 2021, Accessed: Dec. 20, 2021. [Online]. Available: <http://arxiv.org/abs/2111.06015>
30. H. Schröter, A. N. Escalante-B., T. Rosenkranz, and A. Maier, “DeepFilterNet: A Low Complexity Speech Enhancement Framework for Full-Band Audio based on Deep Filtering.” arXiv, Feb. 01, 2022. Accessed: Jul. 08, 2022. [Online]. Available: <http://arxiv.org/abs/2110.05588>
31. H. Schröter, A. N. Escalante-B., T. Rosenkranz, and A. Maier, “DeepFilterNet2: Towards Real-Time Speech Enhancement on Embedded Devices for Full-Band Audio.” arXiv, May 11, 2022. Accessed: May 19, 2022. [Online]. Available: <http://arxiv.org/abs/2205.05474>
32. P.C. Loizou, “Speech Enhancement: Theory and Practice (2nd ed.),” CRC Press. 2013, <https://doi.org/10.1201/b14529>.
33. C. Plapous, C. Marro, and P. Scalart, “Speech enhancement using harmonic regeneration,” IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP’2005), Mar 2005, Philadelphia, United States.
34. R. Xu, R. Wu, Y. Ishiwaka, C. Vondrick, and C. Zheng, “Listening to Sounds of Silence for Speech Denoising,” arXiv:2010.12013 [cs, eess], Oct. 2020, Accessed: Dec. 20, 2021. [Online]. Available: <http://arxiv.org/abs/2010.12013>
35. W. Liu, A. Li, Y. Ke, C. Zheng, and X. Li, “know your enemy, know yourself: A unified two-stage framework for speech enhancement,” in Interspeech 2021, Aug. 2021, pp. 186–190. doi: 10.21437/Interspeech.2021-238.
36. C. Zheng, X. Peng, Y. Zhang, S. Srinivasan, and Y. Lu, “Interactive Speech and Noise Modeling for Speech Enhancement,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 16, no. 35, pp. 14549–14557, 2021.
37. Soumi Maiti and Michael I Mandel, “Speech denoising by parametric resynthesis,” in Proc. IEEE ICASSP. IEEE, 2019, pp. 6995–6999.
38. Soumi Maiti and Michael I Mandel, “Speaker independence of neural vocoders and their effect on parametric resynthesis speech enhancement,” in Proc. IEEE ICASSP. IEEE, 2020, pp. 206–210.
39. Zhihao Du, Xueliang Zhang, and Jiqing Han, “A joint framework of denoising autoencoder and generative vocoder for monaural speech enhancement,” IEEE/ACM Trans. ASLP., vol. 28, pp. 1493–1505, 2020.
40. H. Liu et al., “VoiceFixer: Toward General Speech Restoration with Neural Vocoder.” arXiv, Oct. 05, 2021. Accessed: Jul. 06, 2022. [Online]. Available: <http://arxiv.org/abs/2109.13731>
41. X. Zhang et al., “A two-step backward compatible fullband speech enhancement system.” arXiv, Jan. 27, 2022. Accessed: Jul. 17, 2022. [Online]. Available: <http://arxiv.org/abs/2201.10809>

42. G. Yu, A. Li, W. Liu, C. Zheng, Y. Wang, and H. Wang, “Optimizing Shoulder to Shoulder: A Coordinated Sub-Band Fusion Model for Real-Time Full-Band Speech Enhancement.” arXiv, Jun. 15, 2022. Accessed: Jul. 17, 2022. [Online]. Available: <http://arxiv.org/abs/2203.16033>
43. E. J. Diethorn, “Subband Noise Reduction Methods for Speech Enhancement,” *Acoustic Signal Processing for Telecommunication*. Springer US, Boston, MA, pp. 155–178, 2000. [Online]. Available: https://doi.org/10.1007/978-1-4419-8644-3_9.
44. X. Li and R. Horaud, “Online Monaural Speech Enhancement Using Delayed Subband LSTM,” in *Interspeech 2020*, Oct. 2020, pp. 2462–2466. doi: 10.21437/Interspeech.2020-2091.
45. X. Hao, X. Su, R. Horaud and X. Li, ”Fullsubnet: A Full-Band and Sub-Band Fusion Model for Real-Time Single-Channel Speech Enhancement,” *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 6633-6637, doi: 10.1109/ICASSP39728.2021.9414177.
46. S. Pascual, A. Bonafonte, and J. Serrà, “SEGAN: Speech Enhancement Generative Adversarial Network,” arXiv:1703.09452 [cs], Jun. 2017, Accessed: Dec. 20, 2021. [Online]. Available: <http://arxiv.org/abs/1703.09452>
47. A. Pandey and D. Wang, “On Adversarial Training and Loss Functions for Speech Enhancement,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, Apr. 2018, pp. 5414–5418. doi: 10.1109/ICASSP.2018.8462614.
48. A. Wali, “Generative adversarial networks for speech processing: A review,” *Computer Speech*, p. 24, 2022.
49. J. Su, Z. Jin, and A. Finkelstein, “HiFi-GAN: High-Fidelity Denoising and Dereverberation Based on Speech Deep Features in Adversarial Networks,” arXiv:2006.05694 [cs, eess], Sep. 2020, Accessed: Dec. 20, 2021. [Online]. Available: <http://arxiv.org/abs/2006.05694>
50. Kolbæk, Morten et al. “On TasNet for Low-Latency Single-Speaker Speech Enhancement.” ArXiv abs/2103.14882 (2021): n. pag.
51. Zehai Tu, Jisi Zhang, Ning Ma, Jon Barker. A Two-Stage End-to-End System for Speech-in-Noise Hearing Aid Processing, 2021.
52. Chengshi Zheng, Wenzhe Liu, Andong Li and Xiaodong Li, Low-latency Monaural Speech Enhancement with Deep Filter-bank Equalizer, *Jasa*, 2022.
53. Tal Peer and Timo Gerkmann, Phase-Aware Deep Speech Enhancement: It’s All About The Frame Length, arXiv:2203.16222.

Part II

基于深度学习的阵列信号处理

阵列信号处理，这里特指麦克风(传声器)阵列相关的信号处理，包括降噪、去混响、分离和定位等任务。