

语音识别基本法

清华大学语音和语言技术中心

© 清华大学语音和语言技术中心

<http://cslt.org>

有一天，王叔说，我们要写一本书，于是便有了这本“书”。

物有本末，事有终始。一件事情，从头到尾反复地经历与琢磨下来，心中的大图越来越细致完整，慢慢地形成了一套自己的行事法则，这就是经验了。任何一个新的领域，都有经验可言。

我们从实习生开始，摸索着起来，趟过了很多语音领域的坑坑洼洼。后来，与新的实习生一起工作学习，发现有些经验是固定不变的，常限于口口相传，费时耗力又难成体系，不如写在纸上。于是，我们决定把一些经验记录下来，与后来的小伙伴们交流互进，并不断充实完善，也起到一点知识传承的作用。

经验的主观性使得本书并不如论文一样义正辞严，不连贯性也使得本书不能像教材一样循循善诱。本书是直觉的首先流露，多重于实践上的“是什么”，让新来者尽快“感受”到整个语音识别系统的大体模样，做到心中有谱，会使用 Kaldi 等工具调配常见的系统结构，理论上的“为什么”待以后点点积淀。内外兼修，方能见长，实验室王叔（wangd.csalt.org）写的《机器学习》恰可作为理论上的支撑。

本书涉及的代码放在 <https://github.com/tzyll/kaldi> 中，该分支定期更新至最新的 Kaldi 状态。本书牵扯的事，乃至无关的事，都可往来讨论，邮件可至 tangzy@csalt.org。尽信书不如无书，行色匆匆，以此免责：)

此外

风声雨声读书声，声声入耳。我们关心所有的声音，并渴望用手中的技术发掘其中的意义，使之广为流传，从而创造更多的价值，影响更多的人。

进入这个实验室时，我们与语音的缘分便注定了。我们的生活和工作与语音技术分不开了，我们的理想和抱负也惯用了语音领域的思维，这是我们受益的地方，更是技术分子的归属感：更广阔的声音、更深远的智慧，始于语音，始于初次听见。

实验室期待拥有同一个目标、能够一起做事的人。博士后、工程师、联合培养学生和实习生，构成了实验室的生物多样性，更是语音领域的源头活水。

目 录

I	语音识别基础	
1	语音是什么	9
2	语音识别方法	13
3	语音识别工具	19
II	语音识别基本流程	
4	实验先行	23
5	前端处理	31
6	训练与解码	39
III	语音识别实际问题	
7	说话人自适应	47
8	噪声对抗与环境鲁棒性	49

9	新词处理与领域泛化	51
10	小语种识别	53
11	关键词唤醒与嵌入式系统	55

IV

前沿课题

12	说话人识别	59
13	语种识别	61
14	情绪识别	63
15	语音合成	65

V

Roman Forum

16	技术收藏夹	69
17	Q & A	71
	参考文献	73
	索引	75



语音识别基础

1	语音是什么	9
1.1	大音希声	
1.2	看见语音	
2	语音识别方法	13
2.1	总体思路	
2.2	实现方法	
3	语音识别工具	19
3.1	Kaldi	
3.2	深度学习平台	

1. 语音是什么

by 阿汤

从最起初的一声巨响，到梵音天籁，到耳旁的窃窃私语，到妈妈喊我回家吃饭，总离不开声音。声音是这个世界存在并运动着的证据。

1.1 大音希声

假设我们已经知道了声音是什么。

我们可以找到很多描述声音的词语，如“抑扬顿挫”、“余音绕梁”。当我们在脑海中搜刮这类词语时，描述对象总绕不过这两个：人的声音和物的声音。人的声音，就是语音；物的声音，多数想到的是音乐。这样的选择源于人的先验预期：语音和音乐才最可能有意义，有意义的才去关注。估计不会有人乐于用丰富的辞藻来描述毫无意义的声音。所以，语音研究的意义在于语音本身所传递的意义是什么，以及语音为什么能够传递意义。

声音有很多，每时每刻每次振动都能产生声音，可是有意义的声音实在不多。我们可以使用机器随机生成一段声音，心想着也许这段声音可以产生一些文字内涵。这个想法与很多年前就开始忙不迭地敲打莎士比亚巨著的大猩猩没有差别。不管重复多少次，这些随机的声音听起来都是杂音，没意思。很显然，在这样一个庞大的声音空间中，有意义的语音和音乐只是其中极微小的一点，这也是“大音希声”的一种解释吧。偏偏人类就能毫不费力找到那个点，并且能说会道，这种搜索能力也是千百年来才积攒下来的。不过就算是这么一个小点，古往今来的文学和音乐经典也并未占据多少地盘，这也使得语音语言的研究、文学音乐的创作有着广阔的发挥空间。

从大音希声中，我们可以得到以下一些启示：语言是高度概括和规范化的产物，它的熵值（简单理解为系统的混乱程度）极低，所以语言本身反映了一种思维方式，比如不同

语言对“过去时”、“现在时”、“将来时”的处理方式体现了对时间的不同感受，不同语言对主谓宾的排序体现了对空间层次的不同感知；还有，语音在声音空间中是高度集中的，这使得我们在解析一段语音时不用搜索整个声音空间，少了一些盲目性（不过语言本身的博大精深已算是无垠了）。

1.2 看见语音

语音是用来听的，看不见，摸不着，但是我们可以看看语音的保存形式。自然存在的语音是连续的波动，具有波的所有属性，因而声波可以保存成离散的数字，即模数转换（Analog to Digital Conversion, ADC），所以，我们之后所研究的语音并不是声音的最原始形态，甚至都不叫声音，一串数字而已，但这些数字却达到了它的目的：再现声音，且需要传递的信息不丢失。音乐可以做得更彻底，直接将声音记录在一纸没有动静的乐谱上。除了声音，光线也是自然存在的现象，同样地，我们也可以将它数字化，保存成图片或视频。机器学习中注重表征学习（Representation Learning），不管是声音还是光影，它们的数字化保存形式已经是一种表征方法了。对文本的处理显得直来直去一些，因为文字是人类发明出来的，发明文字的目的就是为了保存，如音符一样，它也是一种离散的可记录、传播的符号，它的形态就是它的保存形式，所以文字本身就是文本处理的原始表征方法。



图 1.1: 语音文件的波形图（Adobe Audition 生成）

语音的基本保存形式可用波形图（Waveform）展现出来，如图 1.1 所示，可以简单地看作是一串上下摆动的数字，比如，每 1 秒的音频可以用 16000 个电压数值表示，即采样率为 16 kHz 。进一步聚焦放大波形图，可以清晰地看到每个采样点，如图 1.2 所示。真正的语音不需要额外的注解，但对于数字化的语音来说，还需要额外的信息对文件格式进行说明，如信道、采样率、精度、时长等，并有文件大小 = 格式信息 + 信道数 * 采样率 * 精度 * 时长。可以用 `soxi` 查看文件信息，如图 1.3 所示。

语音，是包含时序信息的序列，是时域表征。离散傅里叶变换（Discrete Fourier Transform, DFT）使得语音的频域分析成为可能，图 1.1 的语音可以变成图 1.4 的频谱图（Spectrogram）模样，图中可以清楚地看到“层峦叠嶂”，原始音频里的信息又以另一种表征方法释放出来了，颜色明暗表示数值大小，较亮的条纹即是共振峰（Formant）。整

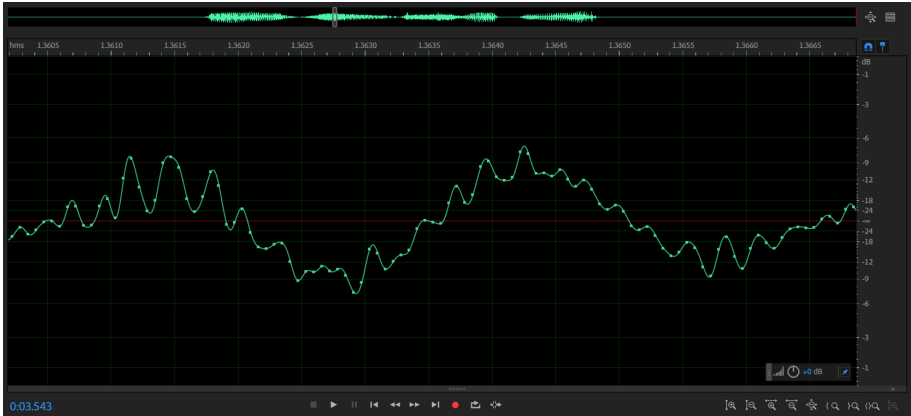


图 1.2: 语音文件的采样点 (Adobe Audition 生成)

```
[tangzy@csl.t.org ~]$ soxi 1a_1.wav

Input File      : '1a_1.wav'
Channels        : 1
Sample Rate     : 16000
Precision       : 16-bit
Duration        : 00:00:03.54 = 56703 samples ~ 265.795 CDDA sectors
File Size       : 119k
Bit Rate        : 268k
Sample Encoding : 16-bit Signed Integer PCM
```

图 1.3: 语音文件的格式信息 (Linux 系统)

这个过程就好比一双好耳朵听到了一首随时间流动的曲子，随即写出了它的谱子，看着谱，曲子又随即可以复现出来。时域与频域之间是一一对应的，可以代表彼此。从一种表征到另一种表征，包含的意义都在，只是有些藏得深，挖掘不到，有些露得浅，一目了然，后者才是适合机器的，所以机器学习领域常常脱不开表征学习的本质。

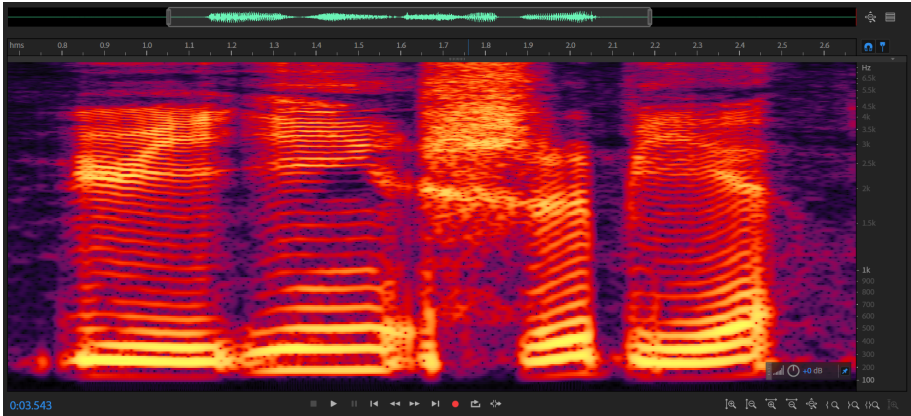


图 1.4: 语音文件的频谱图 (Adobe Audition 生成)

2. 语音识别方法

by 阿汤

语音识别的全称是自动语音识别（Automatic Speech Recognition, ASR），说得多了，就把“自动”省去了，认为“自动”是理所当然的了。语音识别属于序列转换技术，它将语音序列转换为文本序列。大体来说，这是一次搬运，是把一段话的表现形式从语音变成了文本，至于文本想要表达的深层含义（自然语言理解）、倾诉的感情（情感识别）、说话的主体（说话人识别），就需要其他的系统来处理，所以语音应用开始时是分工明确的，但这显然不符合人类的感知和认知，所以后来的技术也有了不同程度的整合和联合学习。

如何实现有效的语音识别，无非是，先确定问题，然后找个模型，最后训好它。

2.1 总体思路

已知一段语音信号，处理成声学特征向量（Feature Vector，而不是 Eigenvector）后表示为 $X = [x_1, x_2, x_3, \dots]$ ，其中 x_i 表示一帧（Frame）特征向量，可能的文本序列表示为 $W = [w_1, w_2, w_3, \dots]$ ，其中 w_i 表示一个词，求 $W^* = \operatorname{argmax}_W P(W|X)$ ，这是语音识别的基本出发点。可知

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)} \quad (2.1)$$

$$\propto P(X|W)P(W) \quad (2.2)$$

其中， $P(X|W)$ 称之为声学模型（Acoustic Model）， $P(W)$ 称之为语言模型（Language Model），二者对语音语言现象刻画得越深刻，识别结果越准确。化整为零，逐个击破，很符合逻辑惯性，所以大多数研究都把语音识别分作声学模型和语言模型两部分，并把很多精力放在声学模型的改进上。后来，基于深度学习和大数据的端对端（End-to-End）方法发展起来，它直接计算 $P(W|X)$ ，把声学模型和语言模型融为了一体。

T 对于不同的候选文本来说，待解码语音的概率保持不变，是各文本之间的不变量，所以公式 2.1 中的 $P(X)$ 可以省去不算。

2.2 实现方法

一段语音，经历什么才能变成它所对应的文本呢？语音作为输入，文本作为输出，第一反应是该有一个函数，自变量一代入，结果就出来了。可是，由于各种因素（如环境、说话人等）的影响，同一段文本，读一千遍就有一千个模样，语音的数字化存储也因之而不同，长短不一，幅度不一，就是一大堆数字的组合爆炸，想要找到一个万全的规则将这些语音唯一地对应到同一段文本，这是演算逻辑所为难的；而常用的词汇量也很庞大，能够拼成的语句不计其数，面对一段语音，遍历搜寻所有可能的文本序列必然无法负担。这样，定义域和值域都是汪洋大海，难以通过一个函数一步到位地映射起来。

如果我们能够找到语音和文本的基本组成单位，并且这些单位是精确的、规整的、可控的，那么二者之间的映射关系会单纯一些。语音，选择的基本单位是帧（Frame），帧是一个向量，整条语音可以整理为以帧为单位的向量组，每帧维度固定不变。一帧数据是由一小段语音经由 ASR 前端数字信号处理产生，涉及的主要技术包括离散傅里叶变换和梅尔滤波器组（Mel Filter Bank）等。一帧的跨度是可调的，以适应不同的文本单位。对于文本，字（或字母）组成词，词组成句子，字词是首先想到的组成单位。

至此，语音的基本组成单位有了统一的格式，文本的基本组成单位又是有限的，问题便在于如何将二者对应起来，如图 2.1 归纳了当下常用的路数，差异多体现在声学模型上，大体说来有以下几个关键结构。

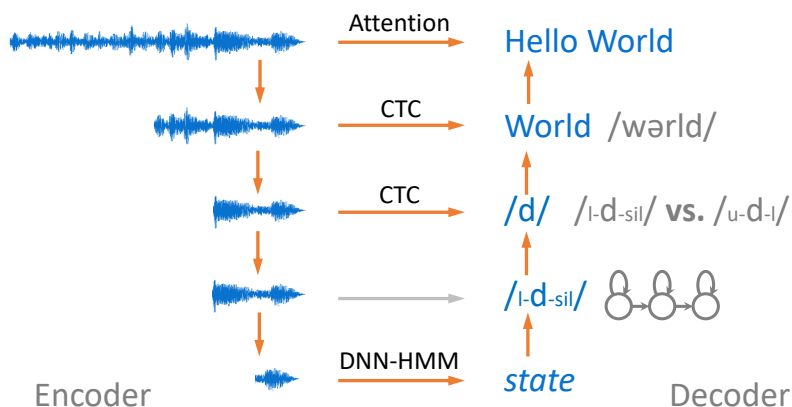


图 2.1: 语音识别的基本途径

2.2.1 HMM

一个首要问题是，语音和文本的不定长关系，使得二者的序列之间无法一一对应，常规的概率公式演算就不适宜了。隐马尔可夫模型（Hidden Markov Model, HMM）正好可以解决这个问题。比如 $P(X|W) = P(x_1, x_2, x_3 | w_1, w_2)$ 可以表示成如图 2.2 隐马尔科夫链的

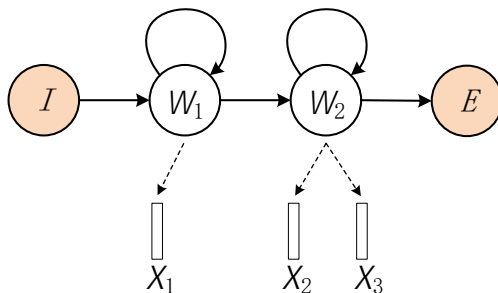


图 2.2: 隐马尔科夫模型

形式，图中 w 是 HMM 的隐含状态， x 是 HMM 的观测值，隐含状态数与观测值数目不受彼此约束，这便解决了输入输出的不定长问题，并有

$$P(X|W) = P(w_1)P(x_1|w_1)P(w_2|w_1)P(x_2|w_2)P(w_2|w_2)P(x_3|w_2) \quad (2.3)$$

其中，初始状态概率 ($P(w_1)$) 和状态转移概率 ($P(w_2|w_1)$ 、 $P(w_2|w_2)$) 可以用常规的统计方法从样本中计算出来，主要的难点在于发射概率 ($P(x_1|w_1)$ 、 $P(x_2|w_2)$ 、 $P(x_3|w_2)$) 的计算，所以声学模型问题进一步细化到发射概率 (Emission Probability) 的学习上，可以通过下文的生成式模型 (GMM) 或判别式模型 (DNN) 求解。

另一个问题是，基本单位的粒度大小。对于语音，帧的粒度可通过调节处理窗口的宽窄来控制。对于文本，字词级别的粒度过于宽泛笼统，于是我们往下分解，如图 2.1 所示：字词是由音素 (Phone) 组成的；音素的上下文不同，同一个音素就有了不同的变体，比如 /l-d-sil/ 与 /u-d-l/ 是一对亲兄弟却是两家子，记为三音素 (Triphone)；每个三音素又可以用一个独立的三状态 HMM 表示，这样，文本方面的基本单位降解为微小的 HMM 状态。由于很多三音素并未在语料中出现或数量不多，并且可以通过决策树 (Decision Tree) 共享三音素的状态，所以对于共有 N 个音素的语言，最终保留下来的三音素状态数量远小于 $3N^3$ ，一般为几千，并把他们叫做 Senones，此时，发射概率记为 $P(x_i|s_j)$ ，其中 s_j 表示第 j 个 Senone，与之对应的帧 (x_i) 的跨度通常取为 25 ms，帧间步移常取为 10 ms，由于跨度大于步移，相邻帧的信息是冗余的，这是跳帧训练和解码的一个出发点。

逐层分解一事物直至根本，把握住每个关键节点之后，拼装回去，整体复又呈现在眼前，只是理解得更透彻了，这正是还原论 (Reductionism) 和第一性原理思考法 (First Principles Thinking) 的思想。但凡问题，都有其最原始的症结；但凡事物，都有其最基本的要素。语音识别系统的设计和训练正是一个从大到小、从宏观到微观的拆解过程，而语音识别系统的解码是要回去的：从 Frame 到 Senone，从 Senone 到 Triphone，再到 Phone，最后到 Word 直至 Sentence。

T HMM 涉及的主要内容有，两组序列 (隐含状态和观测值)，三种概率 (初始状态概率，状态转移概率，发射概率)，和三个基本问题 (产生观测序列的概率计算，最佳隐含状态序列的解码，模型本身的训练)，以及这三个问题的常用算法 (前向或后向算法，Viterbi 算法，EM 算法)。语音识别的最终应用对应的是解码问题，所以对语音识别系统的评估也叫做解码 (Decoding)。

2.2.2 GMM-HMM

HMM 确定了语音识别的整体框架，其中发射概率的求取直接取决于声学模型的好坏，也是研究者探索最多的地方。

高斯混合模型（Gaussian Mixture Model, GMM）是最常用的统计模型，给定充分的子高斯数，GMM 可以拟合任意的概率分布，自我迭代式的 EM 算法使得 GMM 的训练较为容易实现，所以 GMM 成为首选的发射概率模型。每个 GMM 对应一个 Senone，并用各自的概率密度函数（Probability Density Function, PDF）表示。GMM 把每帧看成空间中一个孤立的点，点与点之间没有依赖关系，所以 GMM 忽略了语音信号中的时序信息，并且习惯使用帧间相关性较小的 MFCC（Mel Frequency Cepstral Coefficient）特征。


GMM 训练完成后，通过比对每个 PDF，可以求出每个发射概率 $P(x_i|s_j)$ ，然后往上回溯，直到得到句子，这其中会有一系列条件限制，比如，这一串 Senones 能否组成 Triphone，这一串 Triphones 能否组成 Phone，这一串 Phones 能否组成 Word，这一串 Words 能否组成 Sentence，以及组合过程当中，这种选择是否是当下最优的，这些问题可借助加权有限状态转换器（Weighted Finite State Transducer, WFST）统一进行最优路径搜索 [1]。

2.2.3 DNN-HMM

GMM 是生成式模型（Generative Model），着重刻画数据的内在分布，可直接求解 $P(x_i|s_j)$ ，而 $P(x_i|s_j) = P(s_i|x_j)P(x_j)/P(s_j)$ ，因 $P(x_j)$ 省去不算， $P(s_j)$ 可通过常规统计方法求出，问题进一步归结为求取 $P(s_i|x_j)$ ，这是典型的分类（Classification）问题，也是判别式模型（Discriminative Model）所擅长的，其中深度神经网络（Deep Neural Network, DNN）的研究在当下很是繁荣。上述各项也有各自的叫法， $P(x_i|s_j)$ 是似然（Likelihood）， $P(s_j)$ 是先验概率（Prior Probability）， $P(s_i|x_j)$ 是后验概率（Posterior Probability）。

DNN 用于分类问题，是有监督学习（Supervised Learning），标签（Label）的准备是必不可少的。由于训练集中只提供了整条语音与整条文本之间的对应关系，并未明确指出帧级别的标签，所以还需要额外的算法对数据集进行打标签，选择的方法是上文的 GMM。作为生成式模型的 GMM 擅长捕捉已知数据中的内在关系，能够很好地刻画数据的分布，打出的标签具有较高的可信度，但对于未知数据的分类，判别式模型的 DNN 有着更强的泛化能力。通俗点来说，GMM 善于就已有资源进行最大化的开发（Exploitation），而 DNN 擅长举一反三，具有探索精神（Exploration），DNN-HMM 能够超越 GMM-HMM 正是两大态度的强强结合，所以青（DNN）出于蓝（GMM）也就不足为奇了。

相较于 GMM-HMM 结构，DNN-HMM 与之唯一的不同是结构中的发射概率是由 DNN 而非 GMM 求出的，即二者的区别在于 GMM 与 DNN 之间的相互替代。此外，GMM 和 DNN 中的前向神经网络（Feedforward Neural Network），是独立对待各帧的，即上一帧计算的结果不会影响下一帧的计算，忽略了帧与帧之间的时序信息。DNN 起用循环神经网络（Recurrent Neural Network, RNN）时，便可以考虑时序信息了。

 贝叶斯定理（Bayes' theorem）已被用到两次，宏观的一次是分出了声学模型和语言模型，微观的一次是构造了 HMM 发射概率的判别式求法。

2.2.4 End-to-End

由于语音与文本的多变性，刚开始的时候我们否决了从语音到文本端到端映射的想法，经过了抽丝剥茧、以小见大，再回过头来看这个问题。假设输入是一整段语音（以帧为基本单位），输出是对应的文本（以音素或字词为基本单位），两端数据都处理成规整的数学表示形式了，只要数据是足够的，选的算法是合适的，兴许能训练出一个好的端对端模型，于是所有的压力就转移到模型上来了，怎样选择一个内心强大的模型是关键。深度学习方法是端对端学习的主要途径。

端对端学习需要考虑的首要问题也是输入输出的不定长问题。

对于输入，可以考虑将不同长度的数据转化为固定维度的向量。如果输入一股脑地进入模型，可以选择使用卷积神经网络（Convolutional Neural Network, CNN）进行转换，CNN 通过控制池化层（Pooling Layer）的尺度来保证不同的输入转换后的维度相同；如果输入分帧逐次进入模型，可以使用 RNN，虽然输入是分开进入的，但 RNN 可以将积累的历史信息在最后以固定维度一次性输出。这两个方法常常用于基于注意力（Attention）的网络结构 [2, 3]。

对于输出，往往要参照输入的处理。先考虑输入长度不做处理的情况，此时输出的长度需要与输入保持匹配。因为语音识别中，真实输出的长度远小于输入的长度，可以引入空白标签充数，这是 CTC（Connectionist Temporal Classification）损失函数 [4] 常用的技巧，如果真实输出的长度大于输入的长度，常规 CTC 就不适宜了；另一个情况是，输入只传来一个向量，这正是上段对输入的处理，也正是注意力模型的手段，它根据这个向量解码出一个文本序列（真正实现时，不同时步传来的向量因着当时的注意力权重有所差异和偏重），此时输出的长度便没有了参照，则需要其他机制来判断是否结束输出，比如引入结束符标签，当输出该标签时便结束输出。

当仔细斟酌了输入输出的不定长问题，目前最主流的两个端对端方法也呼之欲出，即上文提到的基于 CTC 损失函数和注意力网络结构的深度学习方法，且二者可以合用。端对端方法将声学模型和语言模型融为一体，简单明了，实施便捷，是当下语音识别的主要方向之一。随着数据量和计算力的增加，端对端方法行之愈加有效，然而这里仍将语音识别系统拆解开来、逐一透视，只因这是真正理解语音识别的必经之路。

上面简述了声学模型各个层次可以使用的数学模型，而数学模型的参数该如何确定才能物尽其用，就是训练的事了，具体实施细节会在后面结合实验详述。语言模型方面没有太多的枝节，常用的方法基于 n 元语法（N-gram Grammar）或 RNN。

3. 语音识别工具

by 阿汤

开源社区大大加速了计算机科学的研究进展，语音识别领域更是如此，其中广泛使用的语音识别开源工具有 CMUSphinx¹、HTK²、Kaldi³等，更多语音识别工具可参考维基百科⁴。本书以 Kaldi 的应用为主。

3.1 Kaldi

Kaldi [5] 是语音识别工具中的后起之秀，年轻正是它的优势之一，可以直接汲取前人的经验，吸收当下已成熟的语音技术，没有历史中的摸爬滚打，避免了积重难返的尴尬。清晰的代码结构，完整的 GMM、WFST 实现，大量适合新手的案例教程，繁荣的开源社区，以及更开放的代码许可，使得 Kaldi 吸引了大批用户。

深度学习广泛应用于语音识别后，Kaldi 先后设计了不同的神经网络构架（nnet1、nnet2、nnet3），其中 nnet3 越来越被研究者所接受，相较于其他两种构架，nnet3 采用计算图（Computational Graph）的思路，可以更容易地设计各种类型的网络结构，并支持多任务并行计算，大大缩短训练时间。

后文将利用 Kaldi 部署实验，并对实验中所涉及的相关概念和技术进行梳理。

¹<https://cmusphinx.github.io>

²<http://htk.eng.cam.ac.uk>

³<http://kaldi-asr.org>

⁴https://en.wikipedia.org/wiki/List_of_speech_recognition_software

3.2 深度学习平台

随着深度学习的发展，更先进的计算平台层出不穷⁵。比起 Kaldi 等术业专攻的平台，通用深度学习框架提供各种深度学习技术，并可拓展应用于多种任务，比如语音识别、计算机视觉、自然语言处理等，所以语音识别系统的建立并不局限于某个平台。最为流行的深度学习框架有

TensorFlow⁶（Google 首先开发并使用），
PyTorch⁷（Facebook 首先开发并使用），
Caffe2⁸（Facebook 首先开发并使用，已集成至 PyTorch），
CNTK⁹（Microsoft 首先开发并使用），
MXNet¹⁰（Amazon 等使用），

以及对已有框架的进一步封装，比如 Keras¹¹（TensorFlow 已开发相关 API），其他的不一列举，其中 PyTorch 等使用动态计算图，较适合快速的研究探索，TensorFlow、Caffe2 等较适合高效的产品部署（包括移动端）。

通用深度学习框架的内核语言多为 C++，前端接口语言多支持 Python，这样的搭配使得保持灵活性的同时又不失计算速度。开源工具的更新换代也很快，比如 Theano¹² 已停止维护，有些功成身退的意味，而面对执着于 Lua 语言的 Torch¹³，更多人选择或转移到了 PyTorch。

面对林林总总的深度学习框架，Microsoft 与 Facebook 发起推出 ONNX¹⁴，让用户可以在不同框架之间转换模型。

语音识别系统有着长久的积淀，并形成了完整的流程（从前端语音信号处理，到声学模型和语言模型的训练，再到后端的解码），而深度学习方法较多地作用于声学模型和语言模型部分（或者端对端模型），所以，深度学习框架常与专有的语音识别工具相结合，各取所长，相互弥补，以减少重复劳动、提高研发效率。

各种开源工具应接不暇，然而善假于物而不囿于物，通晓原理，仍是使用工具的基本原则。

⁵https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

⁶<https://www.tensorflow.org>

⁷<http://pytorch.org>

⁸<https://caffe2.ai>

⁹<https://www.microsoft.com/en-us/cognitive-toolkit>

¹⁰<https://mxnet.incubator.apache.org>

¹¹<https://keras.io>

¹²<http://deeplearning.net/software/theano>

¹³<http://torch.ch>

¹⁴<http://onnx.ai>



语音识别基本流程

4	实验先行	23
4.1	代码	
4.2	运行	
4.3	其他案例	
5	前端处理	31
5.1	数据准备	
5.2	声学特征提取	
6	训练与解码	39
6.1	GMM-HMM	
6.2	DNN-HMM	

4. 实验先行

by 阿汤

对语音识别有了大体印象，或许技术流程的层次、细节、原理仍不是太清晰，此时大可以花些时间先行将每个知识点逐一弄明白，等到面面俱到了，这才出山，并期望有了一整套理论支撑便能立马上手一个语音识别系统，而后畅通无阻。这样的线性思维适合一些考试，只要把教材的知识点摸得烂熟，一鼓作气，考个高分就很容易了。只是这种方式并不适用于技术，终究是绝知此事要躬行。有些细节非得实践才能明了，有些经验并非总能见诸文字，更何况技术日新月异，有过时的，有更新的，鱼龙混杂，一劳永逸的方法不可得。显然，对技术该有开放包容的态度和敏锐的感知，时刻准备着更进一层，不断扩大自己的知识领地，所以好的学习策略应是尽快有个自己的知识框架，无论粗糙、精致，首先保证骨架的完整性，然后不断添砖加瓦，一层一层地铺设，每一次都有新面貌、新视野。如果一个方寸一个方寸逐个修葺，这边没完那边便不动工，就无法有的放矢。第二部分从一个完整的语音识别实验开始，走过这一遭，语音识别的基本步骤已于心中有数，随后才是局部的深化和装点。

使用 Thchs30 数据库，可用 Kaldi 快速实现一个基于 DNN-HMM 结构的语音识别系统。本章先行实现该系统的训练和解码，接下来几章简要介绍几大步骤以及每步生成的文件构成。书中实验代码放于 <https://github.com/tzyll/kaldi> 下的 `egs/cs1t_cases`，下文的文件引用将省去这些路径，各子目录中的 `steps` 和 `utils` 软链接的是各个案例公用的标准目录，放于 `egs/wsj`，`steps` 包含的文件与系统训练和解码步骤直接相关，`utils` 包含一些可能会用到的实用工具，下文以 `steps` 和 `utils` 为开头的文件引用都来自这两个目录，不再表明相对出处。

4.1 代码

参照 README, Linux 下安装好 Kaldi, 进入 asr_baseline (第二部分的实验多以此为当前目录), 其中上层脚本 run.sh 包含了 DNN-HMM 语音识别系统从前期数据准备到最后解码的整个过程, 该脚本是语音识别各个步骤的封装, 每个步骤的脚本又是另一些细枝末节的包裹, 归根结底, 大部分命令会寻至 Kaldi 编译出来的 C++ 可执行程序, 这些 C++ 可执行程序基本放在 kaldi/src 中以 bin 结尾的文件夹中, 其名可望文生义, 从而快速了解功能, 代码本身和 Kaldi 文档¹则有详细介绍。通过 run.sh 中的代码注释可速览整个语音识别的流程, 代码如下:

```
1  #!/bin/bash
2
3  # Copyright 2016  Tsinghua University (Author: Dong Wang,
   #           Xuewei Zhang)
4  #           2018  Tsinghua University (Author: Zhiyuan
   #           Tang)
5  # Apache 2.0.
6
7  . ./cmd.sh ## You'll want to change cmd.sh to something
   #           that will work on your system.
8  #           ## This relates to the queue.
9  . ./path.sh
10
11
12  n=8 # parallel jobs
13
14  set -euo pipefail
15
16
17  ##### Bookmark: basic preparation #####
18
19  # corpus and trans directory
20  thchs=/nfs/public/materials/data/thchs30-openslr
21
22  # you can obtain the database by uncommting the following
   #           lines
23  # [ -d $thchs ] || mkdir -p $thchs
24  # echo "downloading THCHS30 at $thchs ..."
```

¹<http://kaldi-asr.org/doc>


```
25 # local/download_and_untar.sh $thchs
    http://www.openslr.org/resources/18 data_thchs30
26 # local/download_and_untar.sh $thchs
    http://www.openslr.org/resources/18 resource
27 # local/download_and_untar.sh $thchs
    http://www.openslr.org/resources/18 test-noise
28
29 # generate text, wav.scp, utt2pk, spk2utt in
    data/{train,test}
30 local/thchs-30_data_prep.sh $thchs/data_thchs30
31
32
33 ##### Bookmark: language preparation #####
34
35 # prepare lexicon.txt, extra_questions.txt,
    nonsilence_phones.txt, optional_silence.txt,
    silence_phones.txt
36 # build a large lexicon that involves words in both the
    training and decoding, all in data/dict
37 mkdir -p data/dict;
38 cp $thchs/resource/dict/{extra_questions.txt,
    nonsilence_phones.txt, optional_silence.txt,
    silence_phones.txt} data/dict && \
39 cat $thchs/resource/dict/lexicon.txt
    $thchs/data_thchs30/lm_word/lexicon.txt | \
40 grep -v '<s>' | grep -v '</s>' | sort -u >
    data/dict/lexicon.txt
41
42
43 ##### Bookmark: language processing #####
44
45 # generate language stuff used for training
46 # also lexicon to L_disambig.fst for graph making in
    local/thchs-30_decode.sh
47 mkdir -p data/lang;
48 utils/prepare_lang.sh --position_dependent_phones false
    data/dict "<SPOKEN_NOISE>" data/local/lang data/lang
49
```

```

50 # format trained or provided language model to G.fst
51 # prepare things for graph making in
    local/thchs-30_decode.sh, not necessary for training
52 (
53     mkdir -p data/graph;
54     gzip -c $thchs/data_thchs30/lm_word/word.3gram.lm >
        data/graph/word.3gram.lm.gz
55     utils/format_lm.sh data/lang
        data/graph/word.3gram.lm.gz
        $thchs/data_thchs30/lm_word/lexicon.txt
        data/graph/lang
56 )
57
58
59 ##### Bookmark: feature extraction #####
60
61 # produce MFCC and Fbank features in
    data/{mfcc,fbank}/{train,test}
62 rm -rf data/mfcc && mkdir -p data/mfcc && cp -r
    data/{train,test} data/mfcc
63 rm -rf data/fbank && mkdir -p data/fbank && cp -r
    data/{train,test} data/fbank
64 for x in train test; do
65     # make mfcc and fbank
66     steps/make_mfcc.sh --nj $n --cmd "$train_cmd"
        data/mfcc/$x
67     steps/make_fbank.sh --nj $n --cmd "$train_cmd"
        data/fbank/$x
68     # compute cmvn
69     steps/compute_cmvn_stats.sh data/mfcc/$x
70     steps/compute_cmvn_stats.sh data/fbank/$x
71 done
72
73
74 ##### Bookmark: GMM-HMM training & decoding #####
75
76 # monophone
77 steps/train_mono.sh --boost-silence 1.25 --nj $n --cmd

```

```
    "$train_cmd" data/mfcc/train data/lang exp/mono
78 # test monophone model
79 local/thchs-30_decode.sh --nj $n "steps/decode.sh"
    exp/mono data/mfcc &
80 # monophone ali
81 steps/align_si.sh --boost-silence 1.25 --nj $n --cmd
    "$train_cmd" data/mfcc/train data/lang exp/mono
    exp/mono_ali
82
83 # triphone
84 steps/train_deltas.sh --boost-silence 1.25 --cmd
    "$train_cmd" 2000 10000 data/mfcc/train data/lang
    exp/mono_ali exp/tri1
85 # test tri1 model
86 local/thchs-30_decode.sh --nj $n "steps/decode.sh"
    exp/tri1 data/mfcc &
87 # triphone_ali
88 steps/align_si.sh --nj $n --cmd "$train_cmd"
    data/mfcc/train data/lang exp/tri1 exp/tri1_ali
89
90 # lda_mllt
91 steps/train_lda_mllt.sh --cmd "$train_cmd" --splice-opts
    "--left-context=3 --right-context=3" 2500 15000
    data/mfcc/train data/lang exp/tri1_ali exp/tri2b
92 # test tri2b model
93 local/thchs-30_decode.sh --nj $n "steps/decode.sh"
    exp/tri2b data/mfcc &
94 # lda_mllt_ali
95 steps/align_si.sh --nj $n --cmd "$train_cmd"
    --use-graphs true data/mfcc/train data/lang exp/tri2b
    exp/tri2b_ali
96
97 # sat
98 steps/train_sat.sh --cmd "$train_cmd" 2500 15000
    data/mfcc/train data/lang exp/tri2b_ali exp/tri3b
99 # test tri3b model
100 local/thchs-30_decode.sh --nj $n "steps/decode_fmllr.sh"
    exp/tri3b data/mfcc &
```

```
101 # sat_ali
102 steps/align_fmllr.sh --nj $n --cmd "$train_cmd"
    data/mfcc/train data/lang exp/tri3b exp/tri3b_ali
103
104 # quick
105 steps/train_quick.sh --cmd "$train_cmd" 4200 40000
    data/mfcc/train data/lang exp/tri3b_ali exp/tri4b
106 # test tri4b model
107 local/thchs-30_decode.sh --nj $n "steps/decode_fmllr.sh"
    exp/tri4b data/mfcc &
108 # quick_ali
109 steps/align_fmllr.sh --nj $n --cmd "$train_cmd"
    data/mfcc/train data/lang exp/tri4b exp/tri4b_ali
110
111
112 ##### Bookmark: DNN training & decoding #####
113
114 # train tdnn model
115 tdnn_dir=exp/nnet3/tdnn
116 local/nnet3/run_tdnn.sh data/fbank/train exp/tri4b_ali
    $tdnn_dir
117
118 # decoding
119 graph_dir=exp/tri4b/graph_word # the same as gmm
120 steps/nnet3/decode.sh --nj $n --cmd "$decode_cmd"
    $graph_dir data/fbank/test $tdnn_dir/decode_test_word
121
122
123 ##### Bookmark: discriminative training & decoding #####
124
125 # mmi training
126 criterion=mmi # mmi, mpfe or smbr
127 local/nnet3/run_tdnn_discriminative.sh --criterion
    $criterion $tdnn_dir data/fbank/train
128
129 # decoding
130 steps/nnet3/decode.sh --nj $n --cmd "$decode_cmd"
    $graph_dir data/fbank/test
```

```
    ${tdnn_dir}_${criterion}/decode_test_word  
131  
132  
133 exit 0
```

4.2 运行

根据服务器配置，设置 `cmd.sh`，其中“`train_cmd`”、“`decode_cmd`”、“`cuda_cmd`”分别表示 CPU 训练任务、解码任务、GPU 训练任务可以调用的机器或机器集群。具体地，“`run.pl`”表示本机运行，“`queue.pl`”表示使用 Grid Engine²集群，如 OGS/GE³，并可通过运行 `qconf -sql` 查看服务器已有的集群配置文件，并将其作为 `queue.pl` 的参数。

执行 `run.sh`（通常后台形式，如 `nohup ./run.sh > run_asr.log &`），可以从 0 到 1 实现整个语音识别系统的训练和解码，根据日志文件查看实验进度。日志是程序开发的重要文件，可通过阅读日志跟踪、了解系统的运行内容，并可根据日志快速定位、调试错误。Kaldi 每一个关键步骤都会生产日志，存放于输出目录中的 `log` 文件夹或以 `.log` 结尾的文件中，这些日志文件是学习和使用 Kaldi 必不可少的材料。

T 执行 `./path.sh`，使得当前 shell 下可以直接调用 Kaldi 编译出来的 C++ 可执行程序和相关脚本，方便进一步分析和使用，比如将中断的语句单独取出，并于命令行运行，可加快调试。当不加任何参数直接运行这些程序或脚本时，可打印使用方法。

4.3 其他案例

语音相关的常规任务包括语种识别（Language Recognition）、说话人识别（Speaker Recognition）等，相关案例参见 `cs1t_cases/{lre_baseline,sre_dvector,sre_ivector}`。此外，Kaldi 针对说话人识别提供了 `x-vector`（`egs/sre16`）系统。

关于语音识别的端对端学习，相关案例可参见 DeepSpeech⁴、Eesen⁵、ESPnet⁶、wav2letter++⁷ 等。

²<http://wiki.gridengine.info/wiki>

³<http://gridscheduler.sourceforge.net>

⁴<https://github.com/mozilla/DeepSpeech>

⁵<https://github.com/srvk/eesen>

⁶<https://github.com/espnet/espnet>

⁷<https://github.com/facebookresearch/wav2letter>

5. 前端处理

by 阿汤

模型的训练由数据驱动，有些数据是必须要准备的。为了让机器学会将语音转换为文本，首先需要给它提供大量的例子，即语音及其对应的文本，这是原始素材，也最能反映学习目的。这些数据的符号化和结构化则需要一些人类其他知识，包括语言知识和数字信号处理知识。

5.1 数据准备

1) 基本数据

run.sh 中的 `Bookmark: basic preparation` 对 Thchs30 原始数据进行了形式上的处理，以适应 Kaldi 的需要。一个常规的有监督语音识别数据集必然包括一一对应的语音和文本，说话人的信息有好处但非必要。Thchs30 经过初步处理后得到四种文本文件，可以直接打开查看（比如训练集则放在 `data/train` 下），这四个文件也是 Kaldi 的必需文件：

- wav.scp，每条语音的 ID 及其存储地址；
- text，每条语音的 ID 及其对应文本；
- utt2spk，每条语音的 ID 及其说话人 ID；
- spk2utt，每个说话人的 ID 及其所说语音的所有 ID，使用 `utils/spk2utt_to_utt2spk.pl` 或 `utils/utt2spk_to_spk2utt.pl` 将其与 `utt2spk` 进行相互转换。

对于不同数据源或任务，可能需要另外准备一些文件，比如 `segments` 文件标记每条小语音属于某条大语音的哪一部分，文件格式形如 “<segment-id> <recording-id> <start-time> <end-time>”，时间以秒计，`extract-segments` 读取此文件对音频进行批量剪切并

保存为 Kaldi 支持的格式（`sox` 也可逐条切割音频）；`spk2gender` 文件标明每个说话人的性别，用于性别识别；`utt2lang` 文件标明每条语音 ID 对应的语种 ID，用于语种识别。由于不同的数据集有着不同的编排，并没有统一的工具提取出以上文件。当根据某个数据集自行生成以上类别的文件并用 `sort` 排序后，可以使用 `utils/validate_data_dir.sh` 校验是否满足 Kaldi 需求，并使用 `utils/fix_data_dir.sh` 进行修复。根据数据校验和修复脚本也可侧面了解 Kaldi 支持的文件类型和格式。

T `utt2spk` 和 `spk2utt` 是 Kaldi 处理所必须的，有时候如果不能及时提供说话人信息，可以“伪造”，比如每条语音的说话人 ID 直接使用这条语音的 ID，这对语音识别性能影响不大，但在做说话人识别任务时，显然务必要提供真实的说话人信息。此外，两个文件都需要排序，为保证二者顺序的总体一致性，通常句子 ID 的前端设置为说话人 ID。

2) 语言资料

语言知识方面，Kaldi 至少需要以下文件，见 `Bookmark: language preparation`，存放于 `data/dict` 下：

- `lexicon.txt`，发音词典，即每个词与其所对应的音素串，格式为“word phone1 phone2 phone3 ...”；
- `lexiconp.txt`，与 `lexicon.txt` 作用相同，多了发音概率，格式为“word pronunciation-probability phone1 ...”，可由系统通过 `lexicon.txt` 自动生成（此时所有词的概率相同），二者提供一个即可，`lexiconp.txt` 优先使用；
- `silence_phones.txt`，静音类音素，包括静音（`sil` 或者 `SIL`）、噪音、笑声等非语言直接相关的伪音素，同一行的音素是某一个音素的不同变体（重音、音调方面），故可共享决策树根；
- `nonsilence_phones.txt`，语言直接相关的真实音素，同一行的音素是某一个音素的不同变体（重音、音调方面），故可共享决策树根；
- `optional_silence.txt`，备用的静音类音素，一般直接来自 `silence_phones.txt` 中的 `sil` 或者 `SIL`；
- `extra_questions.txt`，可为空，同一行的音素有着相同的重音或音调，与 GMM 训练中自动生成的“questions”一同用于决策树的生成。

当针对同一种语言，基于新的数据集另起炉灶时，这些文件都可以直接移植过去复用。运行 `egs/wsj/s5/local/wsj_prepare_dict.sh` 可以瞥见如何利用 CMU 英语发音词典¹构建出其他所需文件，`Thchs30` 中文数据集提供了现成的结果。发音词典应尽可能覆盖训练语料，且基于已有的音素表，可更改或扩充发音词典，以适用于不同的领域或场景。到此为止，Kaldi 用于语音识别系统训练的数据都齐全了，后来的事便是 Kaldi 对这些数据的自动处理和使用。

¹<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

- T** 在决策树的生成当中, `nonsilence_phones.txt` 倡“合”, `extra_questions.txt` 促“分”, 然而前者中的一行如果由同一基音素衍生出来, 具有不同的重音或音调, 则在后者中常常处于不同的行, 这时保留根源性, 即以前者为准。

Bookmark: `language processing` 中 `utils/prepare_lang.sh` 对 `data/dict` 进行了处理, 得到 `data/lang`。选项 “`position_dependent_phones`” 指明是否使用位置相关的音素, 即是否根据一个音素在词中的位置将其加上不同的后缀: “_B” (开头)、“_E” (结尾)、“_I” (中间)、“_S” (独立成词)。参数 “`<SPOKEN_NOISE>`” 取自 `lexicon.txt`, 后续处理中所有集外词 (Out Of Vocabulary, OOV) 都用它来代替。lang 中生成的文件:

- `phones.txt`, 将所有音素一一映射为自然数, 即音素 ID, 引入 “`<eps>`” (epsilon)、消歧 (Disambiguation) 符号 “`#n`” (n 为自然数), 便于 FST 处理;
- `words.txt`, 将词一一映射为自然数, 即词 ID, 引入 “`<eps>`” (epsilon)、消歧符号 “`#0`”、“`<s>`” (句子起始处)、“`</s>`” (句子结尾处), 便于 FST 处理;
- `oov.txt`, `oov.int`, 集外词的替代者 (此处为 `<SPOKEN_NOISE>`) 及其在 `words.txt` 中的 ID;
- `topo`, 各个音素 HMM 模型的拓扑图, 第二章提过将一个音素 (或三音素) 表示成一个 HMM, 此文件确定了每个音素使用的 HMM 状态数以及转移概率, 用于初始化单音素 GMM-HMM, 可根据需要自行进行修改 (并用 `utils/validate_lang.pl` 校验), 实验中静音音素用了 5 个状态, 其他音素用了 3 个状态。
- `L.fst`, `L_disambig.fst`, 发音词典转换成的 FST, 即输入是音素, 输出是词, 两个 FST 的区别在于后者考虑了消歧。
- `phones/`, 是 `dict/` 的拓展, 内部文件均可以文本形式打开查看, 后缀为 `txt/int/csl` 的同名文件之间是相互转换的, 其中 `context_indep.txt` 标明了上下文无关建模的音素, 通常为静音音素, `wdisambig.txt/wdisambig_phones.int/wdisambig_words.int` 分别标明了 `words.txt` 引入的消歧符号 (`#0`) 及其在 `phones.txt` 和 `words.txt` 中的 ID, `roots.txt` 定义了同一行音素的各个状态是否共享决策树的根以及是否拆分, 对应的音素集则存放于 `sets.txt`。

- T** 消歧是为了确保发音词典能够得到一个确定性的 (Deterministic) WFST。如果有些词的音素串是另一些词的前缀, 则需要引入消歧音素加在前者后面。

语言模型方面, 可以单独提供 ARPA 格式的统计语言模型, 也可以由现有文本训练出来 (如使用 Kaldi LM 工具或 SRILM 工具包的 `ngram-count`, 具体训练方法可参照 `egs/fisher_swbd/s5/local/fisher_train_lms.sh`), `utils/format_lm.sh` 将该语言模型转换为 G.fst, 即输入是词, 输出也是词, 与 `data/lang` 中的文件一同放在 `data/graph/lang` 下, 用于后面制作解码图, 与模型的训练无关。

5.2 声学特征提取

原始音频信号可以直接作为模型的输入, 只是在保守情况下, 如数据不足、计算力薄

弱时，更讨好的做法是先将其由时域信号转换为频域信号，借鉴人耳的处理机制，最终产生声学特征（Acoustic Feature）。声学特征提取使得语音信息更容易暴露，大大降低算法优化的压力，某种程度上也起到降维的效果，提高计算效率，比如 16 kHz 下的 25 ms 共 400 个数值可转换为 40 维的声学特征。

Bookmark: feature extraction 分别提取音频的 MFCC（Mel Frequency Cepstral Coefficient，梅尔频率倒谱系数）和 FBANK（Mel Filter Bank，梅尔滤波器组）两种声学特征，并计算二者关于说话人的倒谱均值和方差统计量，用于 CMVN（Cepstral Mean and Variance Normalization）标准化。

计算 MFCC 和 FBANK 之前需要通过 `conf/{mfcc.conf,fbank.conf}` 设置相关选项，可通过命令行运行 `compute-mfcc-feats` 和 `compute-fbank-feats` 得知可设置哪些选项，每个选项基本都有合理的默认值。原理上 MFCC 是基于 FBANK 生成的，Kaldi 则将两种特征的计算过程分别包装成两个命令。MFCC 特征各维度之间具有较弱的相关性，适合 GMM 的训练，FBANK 特征用于 DNN 的训练则更有优势。两种特征需要注意的选项有：

- `sample-frequency`，音频的采样频率，默认为 16000（16 kHz），如果真实数据与此不同需要指明；
- `num-mel-bins`，梅尔滤波器个数，两种特征兼有，对于 FBANK 来说也是特征维度（通常设为 40）；
- `num-ceps`，MFCC 特征专有，倒谱个数，也是特征的维度，须不大于 `num-mel-bins`，通常使用默认值 13，首维较为特殊，为第 0 个倒谱系数，即 C0，如果 `use-energy` 设为 `true`，则首维替换为能量值（对数形式）；
- `use-energy`，对于 MFCC 来说，表明计算时是否使用“energy”，GMM-HMM 的训练通常将其设置为“false”，对于 FBANK 来说，表明计算时是否在特征首部多加一维能量值，`compute-vad` 使用 MFCC 或 FBANK 中的能量值用于语音活动检测（Voice Activity Detection，VAD）时，此选项须设为“true”。

实验中对于训练集，MFCC 和 FBANK 以及它们的均值和方差统计量分别存放在 `data/mfcc/train` 和 `data/fbank/train` 的 `data` 文件夹下，以“.ark”（archive，存档文件）和“.scp”（script，前者的索引文件）格式同步存在，最终将所有索引文件合并放入上一级目录中，即 `feats.scp` 和 `cmvn.scp`。通过索引文件可以找到每条语音特征数据的存放位置，并可通过 `copy-feats` 以文本形式打印出来，每条语音的声学特征都以矩阵形式存储，每一行为一帧，宽度即为特征维度，可用 `feat-to-dim` 查看，行数即为帧数，不同长度的语音有着不同的帧数，可用 `feat-to-len` 查看每条或所有语音的总帧数（考虑到每帧的跨度和步移，有助于估算语音的总时长，大体等于总帧数与步移之积）。

T Kaldi 以 ark 和 scp 两种格式同步存储文件，相关命令对这些数据进行读写（I/O）时需要特别注明，并可添加相关选项，下面是基于 `copy-feats` 的几个例子。

1. 读 ark，写文本 ark：

```
copy-feats ark:raw_mfcc_train.1.ark ark,t:tmp.ark
```

2. 读 scp，写文本 ark：

```
copy-feats scp:feats.scp ark,t:tmp.ark
```

3. 读 scp, 写二进制 ark:

```
copy-feats scp:feats.scp ark:tmp.ark
```

4. 读 ark, 写二进制 ark、scp:

```
copy-feats ark:raw_mfcc_train.1.ark ark,scp:tmp.ark,tmp.scp
```

5. 读写 scp 时, 忽略数据丢失错误:

```
copy-feats scp,p:feats.scp ark,scp,p:tmp.ark,tmp.scp
```

另外, 同时写 ark 和 scp 时 ark 须放在 scp 前面, scp 不能单独写入; 选项相对于 ark/scp 的位置不分先后; 多种选项可以同时存在。

提取声学特征的目的是在保证音素可辨的情况下, 增强信号对说话人、噪声、信道等的鲁棒性, 常用的声学特征有 FBANK、MFCC、PLP (Perceptual Linear Prediction) 等。Kaldi 中使用 `compute-plp-feats` 提取 PLP 特征。下面以 MFCC 特征为例说明其产生的主要步骤。声学特征提取需要预先对音频做一些处理, 并将其转换至频域, 进一步产生 FBANK 和 MFCC 特征, 综合起来, 大体上主要有如图 5.1 所示的流程。

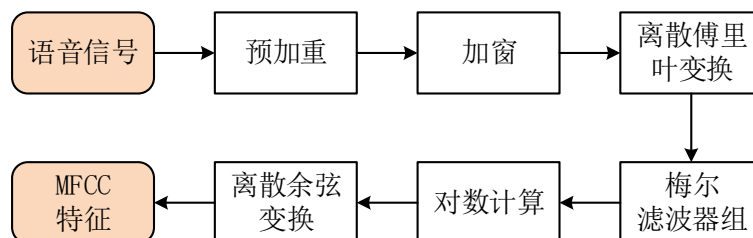


图 5.1: MFCC 特征的提取流程

5.2.1 预加重 (Pre-emphasis)

语音中有频谱倾斜 (Spectral Tilt) 现象, 即低频具有较高能量, 因此, 需要加重高频语音的能量, 使得高频信息凸显出来, 其计算方法为

$$x'[t] = x[t] - \alpha x[t-1] \quad (5.1)$$

其中, $x[t]$ 表示音频数据 (可以看成是一个向量) 的第 t 个数, α 通常取值范围是 (0.95, 0.99)。

预加重之前也可先对音频进行抖动 (Dithering), 抖动是信号处理常用的手段, 它将信号加入低剂量随机噪音, 可有效降低录制音频时模数转换产生的量化误差 (Quantization Error), 似一种以其人之道还治其人之身的方法。

5.2.2 加窗 (Windowing)

特征提取时, 如果每次取出 25 ms (跨度) 的语音, 进行离散傅里叶变换计算出一帧, 接着步移 10 ms 继续计算下一帧, 这种基本做法就是矩形窗 (Rectangular Window), 棱角分明的窗容易造成频谱泄露 (Spectral Leakage), 可以选择使用钟形窗, 如海明窗 (Hamming Window)、汉宁窗 (Hanning Window) 等。加窗的计算方法为

$$x'[n] = w[n]x[n] \quad (5.2)$$

其中 $x[n]$ 是所取窗口（窗长为 N ）之内的第 n 个数， $w[n]$ 是与之对应的权重，不同的加窗方式则体现在 w 的取值上，以 $N = 400$ ($16 \text{ kHz} * 25 \text{ ms}$) 为例，图 5.2 展示了不同窗口的形状。本质上，加窗计算也是卷积（Convolution）。

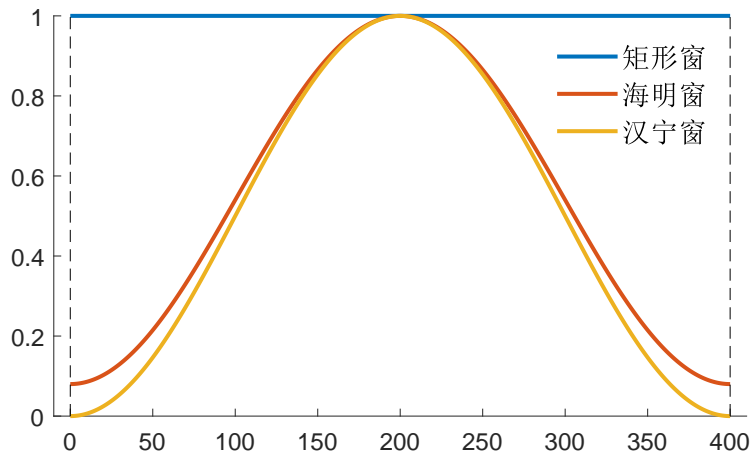


图 5.2: 不同窗口的形状（从上到下分别为矩形窗、海明窗、汉宁窗）

5.2.3 离散傅里叶变换（DFT）

DFT 从每一段加窗后的音频中分别提取出频域信息，计算方式为

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp(-j \frac{2\pi}{N} kn), \quad 0 \leq k < N \quad (5.3)$$

通过复数 $X[k]$ 可计算第 k 个频段的幅度（Magnitude）和相位（Phase），幅度之于频率的坐标图即是频谱（Spectrum），一段语音的所有频谱按时间顺序横排在一起便是这段语音的频谱图（Spectrogram），如之前的图 1.4，每一列都是一个频谱（颜色明暗表示数值大小）。

DFT 的一个实现方法是快速傅里叶变换（Fast Fourier Transform, FFT），可将时间复杂度从 $O(N^2)$ 降为 $O(N \log_2 N)$ ，但需要保证窗长 N 是 2 的指数，如果原窗长不满足此条件，一般音频信号 x 补充尾零，如 400 的窗长可扩展为 512。

频谱的具体使用中，通常用 $|X[k]|^2$ 表示第 k 个频段的能量值（忽略了相位信息），记为 Power Spectrum（既是能量频谱，也是频谱的幂），并根据奈奎斯特频率（Nyquist Frequency），只取其前半段（比如 512 的频数，取其前 $512 * 1/2 + 1$ ）作为最终输出结果。声学特征多基于频谱提取出来，甚至就使用频谱本身，`compute-spectrogram-feats` 可用于频谱特征的提取。

5.2.4 FBANK 特征

人耳对不同频率的感知程度不一样，频率越高，敏感度较低，所以人耳的频域感知是非线性的，梅尔刻度（Mel Scale）正是刻画这种规律的，它反映了人耳线性感知的梅尔频率（Mel Frequency）与一般频率之间的关系，梅尔频率 $Mel(f)$ 与一般频率 f 的转换公

式为

$$Mel(f) = 1127 \ln(1 + \frac{f}{700}) \quad (5.4)$$

将频谱规划到梅尔刻度上，能有效促进语音识别系统的性能，实现方法是梅尔滤波器组（Mel Filter Bank）。具体地，将上一节输出的能量频谱通过如图 5.3 所示的三角滤波器组（Triangular Filter Bank）得到梅尔频谱，计算方式与加窗类似，越往高频，滤波器窗口越大，窗口扩大的量级则与梅尔刻度一致。滤波器的个数就是梅尔频段的总数目，通常取为几十。梅尔频谱的能量数值取对数，最终得到的结果就是常说的 FBANK 特征。对数计算增强了特征的鲁棒性，且人类对能量强弱的感知是符合对数关系的。用于 DNN 训练时，FBANK 的维度就是梅尔滤波器的个数，常取 20 到 40 之间。

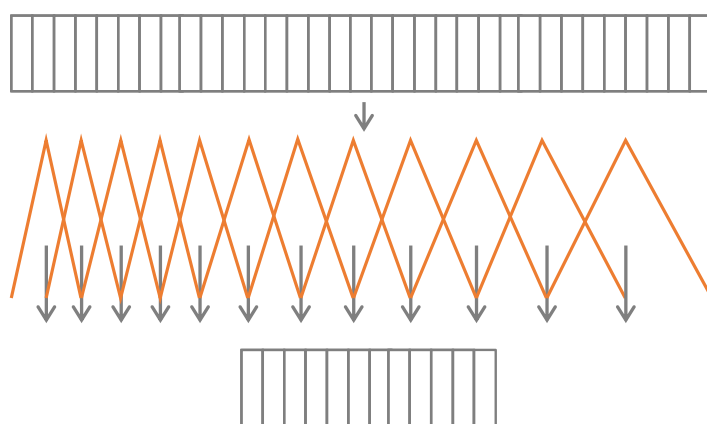


图 5.3: 三角滤波器组的工作方式

T 梅尔滤波器的计算方式与加窗类似，只是窗口跨度依次变大，也属于卷积运算，故可使用卷积神经网络自动学习更加合理的滤波器组。

5.2.5 MFCC 特征

1) 倒谱（Cepstrum）

FBANK 中含有基频（Fundamental Frequency, F0）的谐波（Harmonic），可简单理解为频谱中的毛刺，不利于整体轮廓即包络（Envelope）的显现，且各维度之间具有较高的相关性，不适宜 GMM 学习。生理上，谐波源于声带（Vocal Cord），而真正对具体音素进行调节的是声道（Vocal Tract），MFCC 的目的便是消除与音素判别关系不大的谐波，保留包络信息。FBANK 特征进行离散傅里叶反变换（Inverse Discrete Fourier Transform, IDFT）可以将包络与谐波分开，具体等价于离散余弦变换（Discrete Cosine Transform, DCT），生成结果记为倒谱（Cepstrum 由 Spectrum 字母倒换而来），倒谱的低段位系数（通常前 13 个）可以描述频谱包络，即梅尔频率倒谱系数。Spectrum 是频域，相对地 Cepstrum 则是时域。

如第 5.2 小结所述，使用 `compute-mfcc-feats` 计算 MFCC 时，如需添加一维能量（Energy）值，则由该帧下所有音频采样点取值的平方和计算而来（常取对数），并替换 MFCC 的第一个系数（C0），这样 MFCC 特征则是 Energy + 12 MFCCs。

T 声学特征提取中多次使用到对数计算，对数计算在其他地方也有很多好处，比如它可以一定程度地增加非线性，平滑数据；缩小数据范围，防止溢出；将乘（除）变成加法，方便计算；与 Softmax（神经网络的一种激活函数）合用便于梯度计算和传递等。

2) 动态特征（Dynamic Feature）

语音是时序信号，故声学特征的帧与帧之间并不是孤立的，是连续变化的，前后的变化往往包含一些声音线索，动态特性可以显示特征随时间变化的程度，常采用一阶导数、二阶导数，一阶导数的简单估算方法为

$$d[t] = \frac{c[t+1] - c[t-1]}{2} \quad (5.5)$$

其中， $c[t]$ 表示第 t 帧 MFCC 特征，二阶偏导也照此法，`add-deltas` 可以完成计算。所以通常用来训练 GMM 的声学特征共 39 维（ Δ 表示 delta）：

12 MFCCs + Energy;
12 Δ MFCCs + Δ Energy;
12 Δ^2 MFCCs + Δ^2 Energy。

最后，Kaldi 通过 `compute-fbank-feats` 和 `compute-mfcc-feats` 综合以上过程分别计算 FBANK 和 MFCC 特征，并通过设置相关选项调节各个步骤的计算，比如“`preemphasis-coefficient`”选项设置预加重系数，“`window-type`”选项设置加窗类型。

6. 训练与解码

by 阿汤

上文花了一些笔墨来介绍前端数据准备的过程，其一是为了尽快满足 Kaldi 入口的需求，以促成一个语音识别系统；其二，当 Kaldi 与其他深度学习平台协作用于语音识别时，前端声学特征提取常常依赖于 Kaldi；再者，即使不使用 Kaldi 来搭建语音识别系统，常规声学特征（如 FBANK）提取的基本流程是大同小异的，所以，首先掌握语音数据的前端处理，将语音信号转换为标准数学形式（即向量、矩阵）的声学特征，并把主要精力集中在模型的设计和训练上，必然有助于语音识别系统的快速迭代。

语音识别研究的重心终究是模型的设计、训练和解码。无论是 GMM-HMM 还是 DNN-HMM，都是枝繁叶茂，均可独立成册，而且 DNN 本身即是一大方向，无法简单概括，所以以下只简要介绍 Kaldi 中模型训练和解码的基本实施过程。

6.1 GMM-HMM

帧级别的声学特征准备好了，对应的音素串标签可通过查询发音词典将文本转换而来，接下来便是 GMM-HMM 的训练。

6.1.1 训练

GMM-HMM 训练中 HMM 的转移概率也可更新，但参数量和计算量远小于 GMM，所以接下来的 [Bookmark: GMM-HMM training & decoding](#) 主要进行 GMM 参数的训练。第 2.2 小节提过，每个音素（或三音素）用一个 HMM 建模，每个 HMM 状态的发射概率对应一个 GMM，所以，通俗点说，GMM-HMM 的目的即是找到每一帧属于哪个音素的哪个状态。GMM-HMM 的训练使用自我迭代式的 EM 算法，每一次 EM 后都比自己进步一点点；接下来新一代 GMM-HMM 继承父辈的成果（上一代 GMM-HMM 标记的数据），从

头开始学习，青出于蓝，每一代都比上一代进步一大截。这是 GMM 的进化史，每一代都发挥自己的最大潜力，然后把基业交给更具潜力的下一代：训练模型，测试模型性能（对训练来说不是必须的），用模型标记数据，完成一个生命周期，再用标记的数据训练新的模型，循环往复下去。所谓“标记”，是指现阶段模型自以为是的哪一帧属于哪一个音素的哪个状态。实验中经历了 5 个回合。

模型训练的主要目标是调整每个 GMM 的参数，使其能够更好地刻画它所对应的音素的某一状态，所以需要先获取属于该音素状态的所有语音特征，EM 算法基于这些数据做最大似然估计（Maximum Likelihood Estimation, MLE），亦步亦趋地优化 GMM 的 PDF。不同的 GMM 之间是分开进行的，但可使用高斯模型共享等参数共享方法。因为只知道整段语音对应哪个音素串，小粒度上的对应无从知晓，问题便归结为如何获取属于每个 PDF 的所有语音特征。对于第一个开荒者（即 monophone GMM）来说，采用平启动（Flat Start）的方法，即对于一条语音来说，按对应音素串所有的 PDF 数平分语音特征，每个片段的特征归为对应位置的 PDF 所有，当然也得考虑静音类音素。对于继承者来说，直接使用上一个 GMM-HMM 系统强制对齐（Force Alignment）的结果，即标记。强制对齐时，每帧语音特征与每个 PDF 比对，可知它属于每个音素状态的可能性，每帧数据大可自顾自地选择相对概率最大的那个，但须保证有意义，即最终能够通过 HMM 串联回到参照文本，满足这个“强制”条件的情况下，虽然某些帧选择的不是概率最大的 PDF，但总体上仍可保证有个尽可能高的分数，此时便是关于参考文本的似然最大。

不同时期的 GMM 叠加了不同的特性，“mono”基于单音素，“mono”是“monophone”，“tri1”引入三音素，“tri”是“triphone”，“tri2b”对 MFCC 特征进行了基于线性判别分析（Linear Discriminant Analysis, LDA）和最大似然线性变换（Maximum Likelihood Linear Transform, MLLT）的转换，“tri3b”进行了说话人适应（Speaker Adaptation），“tri4b”返璞归真，再将训练好好来过一次。可以使用 `gmm-info` 查看不同时期 GMM-HMM 系统的具体信息。

6.1.2 解码

如第 2.2 所述，GMM 训练完成后，通过比对每个 PDF，可以求出每个发射概率 $P(x_i|s_j)$ ，然后往上回溯，直到得到句子，这其中会有一系列条件限制，比如，这一串 Senones 能否组成 Triphone，这一串 Triphones 能否组成 Phone，这一串 Phones 能否组成 Word，这一串 Words 能否组成 Sentence，以及组合过程当中，这种选择是否是当下最优的，这些问题可借助加权有限状态转换器（Weighted Finite State Transducer, WFST）统一进行最优路径搜索 [1]。

测试模型性能就是看其解码效果，并由识别文本相对于参照文本的词错误率（Word Error Rate, WER）来衡量，其计算方式是 $WER = \frac{S+D+I}{N}$ ，其中 $S+D+I$ 是编辑距离，包括三种错误：替换（Substitution）、删除（Deletion）和插入（Insertion）， N 是参照文本的总词数。解码需要解码图，即加权有限状态转换器，`local/thchs-30_decode.sh` 中 `Bookmark: graph making` 生成 HCLG.fst，具体放在 GMM 生成目录下的图目录里（本实验是 `graph_word`），接着可以看到，`Bookmark: GMM decoding` 调用 HCLG.fst 进行最优路

径搜索，即解码。因为解码脚本默认在解码生成结果所在目录的上一层目录寻找 GMM 模型，所以解码目录有必要放在 GMM 目录之下（下午的 DNN 解码亦是如此）。HCLG.fst 中，H（H.fst）的输入是 transition-id（Kaldi 所定义，PDF 及相关转移操作所对应的 ID），输出是三音素，包含了 HMM 信息，C（C.fst）的输入是三音素，输出是音素，包好了音素的上下文关系，L（L.fst）、G（G.fst）已在第 5.1 小节准备妥当，分别包含了词典和语法信息。解码图可以完成从 Senone 到 Triphone，再到 Phone，最后到 Word 直至 Sentence 的最优路径搜索。

6.1.3 强制对齐

用模型标记数据就是上面说的强制对齐，即每帧对应哪个 PDF。对齐的结果存放在 exp/*_ali 中，以 mono_ali 为例，用 `gunzip -c ali.1.gz | less` 打印出来，每句话都有一串整数，每个整数是相应位置的帧所属 transition-id，为了将每帧对应到 pdf-id，可以通过 `gunzip -c ali.1.gz | ali-to-pdf final.mdl ark:- ark,t: | less`，并可通过 `gunzip -c ali.1.gz | ali-to-phones final.mdl ark:- ark,t: | less` 将每帧对应到 phone-id。

- T** Kaldi 定义了一些操作对象，并用整数表示其 ID，这是最基本的符号化：
1. phone-id: 音素的 ID，参见 data/lang/phones.txt，强制对齐的结果不含 0（表示<eps>）和消歧符 ID；
 2. hmm-state-id, 单个 HMM 的状态 ID，从 0 开始的几个数，参见 data/lang/topo；
 3. pdf-id, GMM 的 ID，从 0 开始，总数确定了 DNN 输出节点数，通常有数千个；
 4. transition-index, 标识一个状态的不同转移，从 0 开始的几个数；
 5. transition-id, 上面四项的组合 (phone-id,hmm-state-id,pdf-id,transition-index)，可以涵盖所有可能动作，表示哪个 phone 的哪个 state 的哪个 transition 以及这个 state 对应的 pdf 和这个 transition 的概率，其中元组 (phone-id,hmm-state-id,pdf-id) 单独拿出来，叫 transition-state，与 transition-id 都从 1 开始计数。

GMM-HMM 是上一代语音识别系统结构，已有几十年积淀，技术较为成熟和稳定，Kaldi 对其做了极好的流程化整合，所以在后期研究中多对 GMM-HMM 做着老实的搬运，不花多少心思，但这套算法传达的思想源远流长，将影响语音领域多年。

6.2 DNN-HMM

一个复杂的功能指望一个复杂的函数来实现，可以找张纸洋洋洒洒地写上一长串复杂的多项式计算，只是这加减乘除、这对数指数、这括号该如何排列组合不好盘算，还不如画个图来得直截了当，于是 DNN 某个层面的基本思想就是将各种嵌套、组合的多项式计算以图的形式可视化出来，记为“计算图”（Computational Graph）。所见即所得，计算图更契合人的认知感受，想象空间里也多了质地感。从公式到图，像是从算盘到画布，从线性到立体，换了个视角，所以“图”的作用也只是增加了函数设计的深度和广度，降低了难度，本质上的数学计算从未变过。而且，DNN 的基本算子很普通：权重连接（Weight）、

激活函数（Activation），好比是庞大的计算机体系最原始的形态只是简单地 0/1，小的行为相互影响、触动，众志成城，积沙成塔，才涌现了 DNN 的“大智慧”。

6.2.1 主要过程

最后一代的 GMM-HMM 将数据打完标签后，递交给 DNN 进行有监督学习，训练过程是典型的分类器（Classifier）学习。

Bookmark: DNN training & decoding 中训练了一个时延神经网络（Time Delay Neural Network, TDNN）[6]。DNN 的作用是替代了 GMM 来计算 HMM 的发射概率，并不影响 HMM（H.fst），更不用说音素（C.fst）、发音词典（L.fst）和语言模型（G.fst），所以解码时仍然沿用了 GMM-HMM 的 HCLG.fst。

可在 conf/decode_dnn.config 中配置解码参数，其中常用的选项为 beam 和 lattice-beam。beam 是解码时集束搜索（Beam Search）的宽度，宽度越大，搜索结果越准确，相应地搜索时间越长；lattice-beam 是生成 Lattice（简单理解为包含了很多可能的解码路径，留待进一步择优）的宽度，宽度越大，Lattice 越大，相应地搜索时间越长。

语音识别系统的最终衡量标准是 WER，是序列（Sequence）上的尺度上，而 GMM 以最大似然为目标，DNN 以最小化交叉熵（Cross Entropy, CE）或均方误差（Mean Squared Error, MSE）等为损失函数，二者训练时的目标都在帧的尺度上，造成训练与推断（Reference）的不一致，所以可以引入更靠近序列层面的训练准则，也就是序列判别训练（Sequence-Discriminative Training, SDT），常用的准则有：MMI（Maximum Mutual Information）和 BMMI（Boosted MMI），MPE（Minimum Phone Error），sMBR(state Minimum Bayes Risk)。SDT 最开始用于 GMM-HMM，后来移植到 DNN-HMM 上，**Bookmark: discriminative training & decoding** 使用了 MMI，由于模型本身结构不变，只改变了训练的准则，解码方式仍与之前相同。

6.2.2 nnet3 配置

Kaldi nnet3 的神经网络结构是通过读取包含相关组件信息的配置文件生成的。配置文件由实验人员直接编写，它描述了网络中各个组件的拓扑关系。以下简介了 nnet3 配置文件的基本语法，为“绘制”不同的神经网络结构作参考。

Kaldi nnet3 参照的仍是计算图的思想，即将神经网络看成是由多个不同计算单元或步骤按特定顺序连接起来的图，并对该计算图进行编译和执行。nnet3 的配置文件则是对图的构造进行详细的描述，用于网络的初始化，比如含有一层隐含层（激活函数为 Rectifier）的前向神经网络可以描述为：

```
# First the components
component name=affine1 type=AffineComponent input-dim=30 output-dim=1000
component name=relu1 type=RectifiedLinearComponent dim=1000
component name=affine2 type=AffineComponent input-dim=1000 output-dim=800
component name=logsoftmax type=LogSoftmaxComponent dim=800
# Next the component-nodes
```

```

input-node name=input dim=10
component-node name=affine1_node component=affine1 input=Append( Offset(input, -2),
Offset(input, 0), Offset(input, 1))
component-node name=nonlin1 component=relu1 input=affine1_node
component-node name=affine2_node component=affine2 input=nonlin1
component-node name=output_nonlin component=logsoftmax input=affine2_node
output-node name=output input=output_nonlin objective=quadratic

```

1) component 和 component-node

上例中可以看出，配置文件分为两大部分：**component** 和 **component-node**，它们之间的关系可以类比为“类的实例化”。**component** 可以看作是类，它定义了网络中所有情形的组件，每个类（**component**）都有独一无二的名称，属性的设置相互不受干扰，属性包括类型（权重连接方式和不同的激活函数）、输入输出的维度，以及该类的特性。**component-node** 则是对 **component** 的实例化，在对它进行定义时，它所属的类由后面的 **component** 选项给出，并且需要明确它的输入（另一个 **component-node**）。比如上例中，**component-node affine2_node** 实例化了 **component affine2**，并且指定了它的输入为 **component-node nonlin1**，**nonlin1** 的维度与 **affine2** 所要求的输入维度是相同的。

input-node 和 **output-node** 是特殊的标识符，不需要指定类（**component**），它们分别代表神经网络的输入和输出节点，可以有多个（名称不同）。**output-node** 中可以通过 **objective** 选项指定不同的损失函数，如 **linear**（交叉熵，不指定 **objective** 选项时的默认设置）、**quadratic**（均方误差）。还有一个不需要类的特殊标识符 **dim-range-node**，它用于截取一个节点的部分作为输入，比如我们可以通过以下语句截取 **node1** 传入值（向量）的前100维，并赋值给 **node2**：

```
dim-range-node name=node2 input-node=node1 dim-offset=0 dim=100
```

一个 **component** 可以对应多个不同名 **component-node**，此时，这些 **component-node** 也将共享相同的参数。**component** 可以不被实例化，**component-node** 必须要有定义了的 **component**。

2) 属性与描述符

Kaldi 中实现了多种常见的和定制化的 **component**，具体名称、作用和属性，可参考源文件 `nnet3/nnet-{simple,normalize,convolutional,attention,combined,general}-component.h`，其中的常规方法 `InitFromConfig` 表示了网络在初始化过程中需要从配置文件中读取的信息，常见的属性有 **type**（直接使用类名，如激活函数 **SigmoidComponent**、**RectifiedLinearComponent**，全连接 **AffineComponent**、**NaturalGradientAffineComponent** 等），**input-dim** 和 **output-dim**（对于输入输出维度相同的组件来说，直接用 **dim** 表示二者），还有一些组件有自己特定的属性，比如卷积层 **ConvolutionComponent** 需要指定滤镜的大小、步长和个数。

描述符	功能
node1	最基本形式，没有修饰语，node1 的值直接传入。
Append(node1,node2,...)	将 node1、node2 等组件传入的值拼接起来，结果的维度是各组件维度之和。
Sum(node1,node2)	将 node1、node2 传入的值加起来，node1、node2 以及最终结果的维度相同。
Failover(node1,node2)	如果 node1 是不可计算的，则使用 node2。
IfDefined(node1)	如果 node1 目前还没有计算出来，先以“0”值代替。
Offset(node1,value)	node1 传过来的值是相对于当前时标偏离 value（可正可负）时步时的结果。
Switch(node1,node2,...)	按顺序，每个时步使用其中一个 node，该 node 的序号是当前时步对 node 总数取模的结果。
Scale(value,node1)	node1 传入的值乘以系数 value。
Const(value,dim)	维度为 dim、各元素取值为 value 的常值向量。
Round(node1,value)	每个时步，将 node1 的时标（time-index）设置为小于当前时步且又是 value 的最大倍数。
ReplaceIndex(node1,t,value)	每个时步，node1 的时标保持不变，恒为 value。t 也可换为其他变量。

表 6.1: Kaldi nnet3 配置文件中的描述符

上例中出现的其他关键字（Append、Offset）是 nnet3 的描述符（Descriptor），出现于 component-node 的声明中，由源文件 nnet3/nnet-descriptor.h 定义。各个组件相互粘合时，描述符常常用于 input 选项的量化说明，比如上例中 affine1_node 的输入是由 3 个不同时步（time-step）的原始输入（input-node）拼接（Append）而成的，而 Offset 则表示了某个时步的输入与当前时步的相对时间偏移。如果上例用于语音识别任务，affine1_node 的输入则是由当前帧的上上帧、当前帧以及当前帧的下一帧拼接而成，共有 $10 \times 3 = 30$ 维。描述符的使用方法为：input=[Descriptor]，表 6.1 总结了 nnet3 中提供的描述符，描述符是可以嵌套使用的。

最后，nnet3 通过以上语法规则将不同的组件拼装起来并初始化，在设计网络结构时，可以通过 nnet3-init 校验配置文件能否初始化成功。



语音识别实际问题

7	说话人自适应	47
8	噪声对抗与环境鲁棒性	49
9	新词处理与领域泛化	51
10	小语种识别	53
11	关键词唤醒与嵌入式系统	55

7. 说话人自适应

8. 噪声对抗与环境鲁棒性

9. 新词处理与领域泛化

10. 小语种识别

11. 关键词唤醒与嵌入式系统

IV

前沿课题

12	说话人识别	59
13	语种识别	61
14	情绪识别	63
15	语音合成	65

12. 说话人识别

13. 语种识别

14. 情绪识别

15. 语音合成



Roman Forum

16	技术收藏夹	69
17	Q & A	71
	参考文献	73
	索引	75

16. 技术收藏夹

17. Q & A

参考文献

- [1] Mehryar Mohri, Fernando Pereira, and Michael Riley. “Weighted finite-state transducers in speech recognition”. In: *Computer Speech & Language* 16.1 (2002), pages 69–88 (cited on pages 16, 40).
- [2] Jan K Chorowski et al. “Attention-based models for speech recognition”. In: *Advances in Neural Information Processing Systems*. 2015, pages 577–585 (cited on page 17).
- [3] Dzmitry Bahdanau et al. “End-to-end attention-based large vocabulary speech recognition”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016, pages 4945–4949 (cited on page 17).
- [4] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE. 2013, pages 6645–6649 (cited on page 17).
- [5] Daniel Povey et al. “The Kaldi speech recognition toolkit”. In: *IEEE 2011 workshop on automatic speech recognition and understanding*. EPFL-CONF-192584. IEEE Signal Processing Society. 2011 (cited on page 19).
- [6] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. “A time delay neural network architecture for efficient modeling of long temporal contexts”. In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015 (cited on page 42).

索引

A

Acoustic Feature, 声学特征 34
Acoustic Model, 声学模型 13
Activation, 激活函数 42
Analog to Digital Conversion, 模数转换 .. 10
Attention, 注意力 17

B

Beam Search, 集束搜索 42

C

Cepstral Mean and Variance Normalization 34
Cepstrum, 倒谱 37
Classification, 分类 16
Classifier, 分类器 42
Computational Graph, 计算图 19, 41
Connectionist Temporal Classification 17
Convolution, 卷积 36
Convolutional Neural Network, 卷积神经网络 17
Cross Entropy, 交叉熵 42

D

Decision Tree, 决策树 15
Decoding, 解码 15
Deep Neural Network, 深度神经网络 16
Deterministic, 确定性 33
Disambiguation, 消歧 33
Discrete Cosine Transform, DCT, 离散余弦变换 37
Discrete Fourier Transform, 离散傅里叶变换
10
Discriminative Model, 判别式模型 16
Dithering, 抖动 35
Dynamic Feature, 动态特征 38

E

Emission Probability, 发射概率 15
Energy, 能量 38
Envelope, 包络 37

F

Fast Fourier Transform, FFT, 快速傅里叶变换 36

Feedforward Neural Network, 前向神经网络
16
Force Alignment 40
Formant, 共振峰 10
Frame, 帧 14
Fundamental Frequency, 基频 37

G

Gaussian Mixture Model, 高斯混合模型 .. 16
Generative Model, 生成式模型 16

H

Hamming Window, 海明窗 35
Hanning Window, 汉宁窗 35
Harmonic, 谐波 37
Hidden Markov Model, 隐马尔可夫模型 . 14

I

Inverse Discrete Fourier Transform, 离散傅里
叶反变换 37

L

Label, 标签 16
Language Model, 语言模型 13
Language Recognition, 语种识别 29
Likelihood, 似然 16
Linear Discriminant Analysis, 线性判别分析
40

M

Maximum Likelihood Estimation, 最大似然
估计 40
Maximum Likelihood Linear Transform, 最大
似然线性变换 40
Maximum Mutual Information, MMI 42
Mean Squared Error, 均方误差 42
Mel Filter Bank, 梅尔滤波器组 34, 37

Mel Frequency Cepstral Coefficient, 梅尔频率
倒谱系数 34

Mel Frequency, 梅尔频率 36
Mel scale 36
Minimum Phone Error, MPE 42

N

N-gram Grammar, n 元语法 17
Nyquist frequency, 奈奎斯特频率 36

O

Out Of Vocabulary, 集外词 33

P

Perceptual Linear Prediction, PLP 35
Phone, 音素 15
Posterior Probability, 后验概率 16
Pre-emphasis, 预加重 35
Prior Probability, 先验概率 16
Probability Density Function, 概率密度函数
16

Q

Quantization Error, 量化误差 35

R

Rectangular Window, 矩形窗 35
Recurrent Neural Network, 循环神经网络 16
Reference, 推断 42
Representation Learning, 表征学习 10

S

Senones 15
Sequence-Discriminative Training, 序列判别
训练 42
Speaker Recognition, 说话人识别 29
Spectral Leakage, 频谱泄漏 35
Spectral Tilt, 频谱倾斜 35

Spectrogram, 频谱图 10, 36
Spectrum, 频谱 36
Speech Recognition, 语音识别 13
state Minimum Bayes Risk, sMBR 42
Supervised Learning, 有监督学习 16

T

Time Delay Neural Network, 时延神经网络
42
Triangular Filter Bank, 三角滤波器组 37
Triphone, 三音素 15

V

Vocal Cord, 声带 37
Vocal Tract, 声道 37
Voice Activity Detection, 语音活性检测 .. 34

W

Waveform, 波形图 10
Weight, 权重 42
Weighted Finite State Transducer, 加权有限状态转换器 16, 40
Windowing, 加窗 35
Word Error Rate, WER, 词错误率 40