

中科院声学所DNNVAD引擎

详细设计说明

目录

[1.概述 5](#_Toc80365261)

[1.1简介 5](#_Toc80365262)

[1.2 读者对象 5](#_Toc80365263)

[1.3术语定义 6](#_Toc80365264)

[2.总体设计 7](#_Toc80365265)

[3. 数据结构设计 9](#_Toc80365266)

[4.模块设计 11](#_Toc80365267)

[4.1 特征提取模块 11](#_Toc80365268)

[4.2 DNN模型概率计算 12](#_Toc80365269)

[4.3维特比算法 14](#_Toc80365270)

[4.4起尾点判断 17](#_Toc80365271)

[5. 接口设计 18](#_Toc80365272)

[5.1 DNNVAD调用流程 18](#_Toc80365273)

[5.2初始化引擎 18](#_Toc80365274)

[5.3VAD切分 19](#_Toc80365275)

[5.4设置回调函数 20](#_Toc80365276)

[6可靠性设计 21](#_Toc80365277)

[7. 扩展性设计 22](#_Toc80365278)

[8.维护性设计 23](#_Toc80365279)

[9.易用性设计 24](#_Toc80365280)

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 1.概述

1

1.1简介

DNNVAD的主要功能：前端功能，对输入的原始语音进行有效音截取，使得输入到识别引擎中的语音为有效音从而减少噪音对识别引擎的干扰。本引擎主要基于DNN模型的VAD算法。对原始语音进行特征提取并输入DNN模型，根据DNN模型的输出概率值并结合维特比算法，对原始语音逐帧判别是否为有效语音，最后根据起尾点判别算法将原始语音中的有效语音截取出来。

## 1.2 读者对象

本文档是对DNNVAD引擎的内部代码流程和相关技术点的整体说明，因此读者最好满足如下几点条件后再读此文档：

* 有一定的C和C++语言基础
* 有基本的音频信号处理常识

## 1.3术语定义

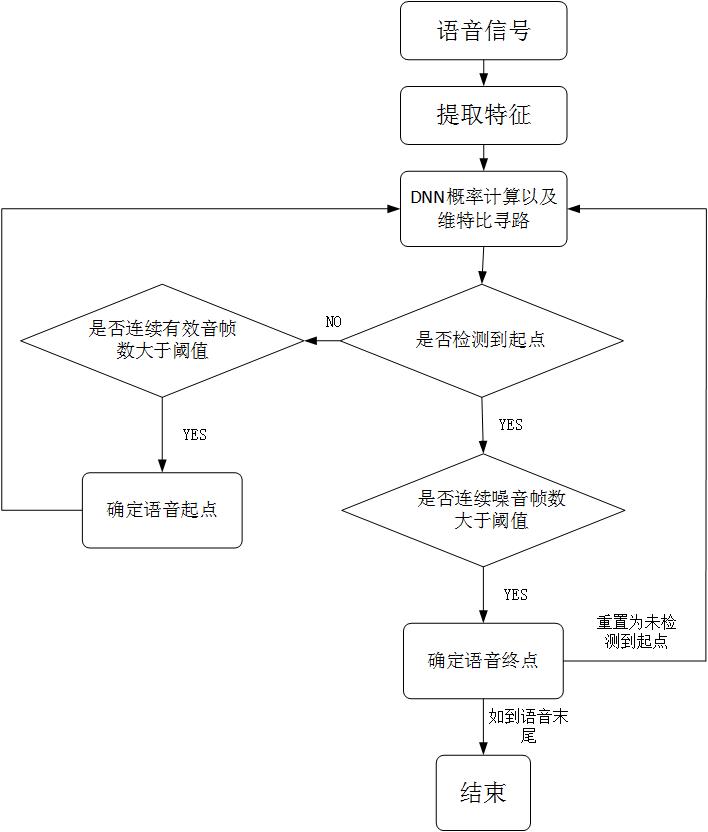
DNN(Deep Neural Networks）：深度神经网络

VAD（Voice Activity Detection）：有效音检测

# 2.总体设计

2

DNNVAD主要模块主要流程包括特征提取、DNN模型推理、维特比寻址、起尾点判断，大致流程图如下：



语音数据送入引擎后，经过“特征提取->DNN概率计算->维特比寻路->结果判决”，最终得到是否唤醒的结果。

* 特征提取：输入的语音数据分别经过预加重、分帧、加窗、FFT、梅尔滤波、对数运算、DCT后形成每帧40维的MFCC语音特征，语音特征数据将被送入DNN概率计算模块；
* DNN概率计算：在引擎初始化阶段加载DNNVAD模型，构建神经网络；将特征送入神经网络中，计算出后验概率，送入结果判决模块；
* 维特比寻路:利用神经网络输出的后验概率，进行维特比寻路，判别输入的语音帧是语音或者非语音；
* 结果判决：根据连续的语音帧或者非语音帧是否超过阈值，来判断语音的开始以及结束。

# 3. 数据结构设计

3

本节主要介绍DNNVAD的核心类，主要功能就是利用神经网络来对送入的语音特征进行分类，从原始语音中寻找有效语音的起尾时间点。

* 实现源文件：DSpSingle.h DSpSingle.cpp
* 类名：DSpSingle
* 模块接口函数说明：

（1）bool initDnn(const char \*\_sDnnModelName, int \_nLeft, int \_nRight, int \_nDim, float g\_backThreshold, float g\_musicThreshold);

功能：

初始化神经网络计算部分，载入模型。

形参：

const char \*\_sDnnModelName：模型名称

int \_nLeft：左拼帧数

int \_nRight：右拼帧数

int \_nDim：输入特征每帧维度

float g\_backThreshold：噪音阈值

float g\_musicThreshold：彩铃阈值

返回值：1

（2）bool procFeat( float \*Feature\_in, int FrameNum, int \*breakpoint, int \*begin, int \*num, void\* Pt, DSpSingle \* CaiLingDspSingle, DSpSingle \*backDspSingle);

功能：

VAD分段函数，通过回调函数的形式返回VAD分段结果。

形参：

float \*Feature\_in：输入语音

int FrameNum：输入语音帧数

int \*breakpoint，int \*begin，int \*num：占位指针，默认为0

void\* Pt： this指针，需要将接口函数中的一些变量送入

DSpSingle \* CaiLingDspSingle：彩铃实例

DSpSingle \*backDspSingle：噪音判断实例

返回值：1

（3）FPNeuralNet \*m\_pDnnModel;

功能：神经网络类，FPNeuralNet为基于kaldi整合的神经网络前向计算类，包括读取神经网络模型以及前向计算得到每帧的分类概率，详情可参考章节4.2 DNN模型概率计算。

# 4.模块设计

4

## 4.1 特征提取模块

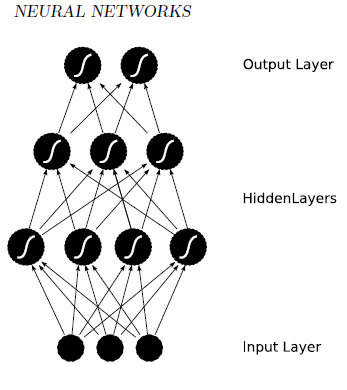
引擎使用梅尔频率倒谱系数（MFCC）作为声学特征，梅尔刻度是一种基于人耳对等距音高变化的感官判断而定的非线性频率刻度，与人耳听觉相符，是语音识别中常用的声学特征。MFCC特征提取模块主要流程图如下，语音输入后分别经过预加重、分帧、加窗、FFT、梅尔滤波、对数运算、DCT后形成每帧40维的MFCC语音特征，特征提取流程图如下：



* 实现：VAD模块直接接受外部送入的语音特征数据，本身并不提取语音特征，以上仅做算法简述。

## 4.2 DNN模型概率计算

提取特征之后，输入DNN模型中的，模块中的DNN模型执行了一个二分类的任务，即输出某一原始语音帧为有效音帧概率和为噪音帧的概率。我们目前采用全连接的深度神经网络（之后会考虑替换其他结构的神经网络）来得到每帧的概率输出。下图是一个典型的神经网络结构示意图，语音特征先后通过输入层、隐含层、输出层，与有效音帧概率与噪音帧之间建立复杂的映射关系。在实际运行中，将每一帧语音特征联合其前后数帧的特征作为输入矢量，检测该帧的所属的类别，之后将每帧的输出概率做为一个二维向量存储下来。



* 实现源文件：NeuralNet.h NeuralNet.cpp
* 类名：FPNeuralNet
* 模块接口函数说明：

（1）bool ParseNNModel(const char\* DNNModel,const char\* prior,const INT nStateNum, const INT nLeftFrm,const INT nRightFrm,const INT nTimeShift,const INT nVecSize);

功能：

模型读取函数，用于读取神经网络模型。

形参：

const char\* DNNModel：神经网络名称

const char\* prior：模型先验概率，一般为NULL

const INT nStateNum：分类数

const INT nLeftFrm，const INT nRightFrm：模型左右拼帧数

const INT nTimeShift：增加语音帧数

const INT nVecSize：输入特征维度

返回值：1

（2）void Propagate(FLOAT\* pFeatData, int nFrmNum);

功能：

神经网络计算函数。

形参：

FLOAT\* pFeatData：输入语音

int nFrmNum：输入语音帧数

返回值：空

（3）Matrix<FLOAT>\* GetResult()；

功能：

得到神经网络计算结果，为一个M \* N的矩阵，其中M为分类数，N为语音帧数。

形参：空

返回值：

神经网络结果矩阵

## 4.3维特比算法

维特比算法是一种[动态规划](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92)算法用于寻找最有可能产生观测事件序列的-维特比路径-隐含状态序列。假设给定隐式马尔可夫模型（HMM）状态空间 S，共有k个状态，初始状态i的概率为  ，从状态 i 到状态 j 的转移概率为 。 令观察到的输出为  。 产生观察结果的最有可能的状态序列



由递推关系给出：



此处  是前 t 个最终状态为 k 的观测结果最有可能对应的状态序列的概率。 通过保存向后指针记住在第二个等式中用到的状态 x 可以获得维特比路径。声明一个函数  ，它返回若  时计算  用到的 x 值 或若  时的 k，这样:



这里我们使用了arg max的标准定义 算法复杂度为  。



维特比算法的基础可以概括成下面三点：

1.如果概率最大的路径p(或者说最短路径)经过某个点，比如途中的X22，那么这条路径上的起始点S到X22的这段子路径Q，一定是S到X22之间的最短路径。否则，用S到X22的最短路径R替代Q，便构成一条比P更短的路径，这显然是矛盾的。证明了满足最优性原理。

2.从S到E的路径必定经过第i个时刻的某个状态，假定第i个时刻有k个状态，那么如果记录了从S到第i个状态的所有k个节点的最短路径，最终的最短路径必经过其中一条，这样，在任意时刻，只要考虑非常有限的最短路即可。

3. 结合以上两点，假定当我们从状态i进入状态i+1时，从S到状态i上各个节的最短路径已经找到，并且记录在这些节点上，那么在计算从起点S到第i+1状态的某个节点Xi+1的最短路径时，只要考虑从S到前一个状态i所有的k个节点的最短路径，以及从这个节点到Xi+1，j的距离即可。

根据DNN模型的输出概率，使用维特比算法，其中每帧的状态数总计有两个，即有效音与噪音，其生成概率为上一步DNN模型的输出结果，转移概率为经验值。之后通过维特比算法，将DNN输出的二维概率序列变为一个由有效音帧以及噪音帧组成的一维序列。

代码中维特比实现：

for(int k=0,n=0;k<ROUTENUM;k++,n++) //Choose ten Contours with largest probs.

{

max=INT\_MIN;//the 0th contour,

pre=0;

back=0;

for(int m=0;m<ROUTENUM;m++)

{

if(probability[m][n]>max)

{

max=probability[m][n];

pre=m;

back=n;

}

}//寻找转移到两种状态上各自累计概率最大的前置状态以及累计概率值

M[k]=max;//store Probs. of the contours until current frame

if(back==0)

{

route[pre][i]=0; //i-th frame is Speech

prior[k]=0; //the content of k-th contour at current frame

}

else

{

route[pre][i]=1;

prior[k]=1;

}//记录下目前状态序列当前的状态

if((float)EPD->DELAY < 0.001)//offline

memcpy(R[k],route[pre],sizeof(int)\*(i+1));

else

{

if( i > EPD->DELAY )

memcpy(R[k]+i+1-EPD->DELAY,route[pre]+i+1-EPD->DELAY,sizeof(int)\*EPD->DELAY);

else

memcpy(R[k],route[pre],sizeof(int)\*(i+1));

}//更新当前的状态序列

probability[pre][back]=-3.40e+30; //Drop the largest Prob.

}

## 4.4起尾点判断

利用DNN输出概率以及维特比算法之后，原始输入语音变为了一个有效音与噪音的一维序列。之后根据起尾点寻找算法来判断该序列中有效音的起点与尾点，其中步骤主要有：

起点寻找算法：当某一有效音帧之后连续的有效音帧超过阈值后，则认为该帧为有效音段的起点；

尾点寻找算法：当某一噪音帧之后连续的噪音帧超过阈值后，则认为该帧为有效音段的尾点；

静音毛刺去除：当某一噪音段的帧数长度低于阈值时，认为该段噪音短为有效音段；。

# 5. 接口设计

5

### 5.1 DNNVAD调用流程

DNNVAD引擎的流程如下:

1. 初始化引擎
2. 读入语音数据，并进行DNNVAD切分，以回调函数的方式返回语音起尾点；

### 5.2初始化引擎

|  |  |
| --- | --- |
| 函数原型 | **DNN\_VAD(char \*config);** |
| 功能 | 加载配置文件 |
| 参数说明 | **char \*config:配置文件位置** |
|  |
| 返回值 |  |
| 说明 |  |

### 5.3VAD切分

|  |  |
| --- | --- |
| 函数原型 | **int doVAD(float \*Feature\_in, int FrameNum, int IsFinished);** |
| 功能 | 接收数据并获取分段结果 |
| 参数说明 | **float \*Feature\_in：语音特征** |
| **int FrameNum：语音特征帧数** |
| **int IsFinished：是否送入完毕，默认1** |
|  |
|  |
| 返回值 | 正常为0，异常为-1 |
| 说明 |  |

### 5.4设置回调函数

|  |  |
| --- | --- |
| 函数原型 | **int SetVadCallback(Dnn\_vad\_callback pFunc, void\* pObj);** |
| 功能 | 将VAD输出的结果通过回调函数的形式返回出来 |
| 参数说明 | **Dnn\_vad\_callback pFunc：定义的回调函数句柄** |
| **void\* pObj：占位空指针，为识别引擎服务** |
| 返回值 | 正常为0 |
| 说明 |  |

# 6可靠性设计

6

DNNAVD引擎设计考虑了应用场景的复杂性和输入的多样性的情况，内部加入了大部分实际情况中会出现的异常情况的处理方案，例如超长静音、超长连续语音等异常事件的处理；并且DNNAVD引擎测试按照严格的7\*24小时压力测试标准进行测试，同时也针对所有异常数据进行异常情况测试，保证引擎的稳定性和可靠性。

# 7. 扩展性设计

7

DNNAVD引擎支持多路并发，保证了不同cpu数和内存大小的机器拓展需求；同时支持多种配置参数，以适应不同的应用场景；引擎本身遵循模块化设计，可以随时添加定制化功能，满足不同应用场景的需要。

# 8.维护性设计

8

DNNAVD引擎为了方便后期的引擎维护，采用标准c/c++函数同时在较低编译器上进行编译。

DNNAVD引擎开放了一些内部参数，保证了引擎的可配性和可维护性。常见参数如下：

SpeechMiniLen 语音最小长度，如果连续语音帧小于这个长度则不返回

SilenceMinLen 静音最小长度，如果连续静音帧大于这个长度则认为语音结束

SpeechMaxLen 语音的最大长度，当语音长度大于最大长度会强制输出语音尾点

SNRBias 静音增益，可以设置放大或者缩小静音输出概率

# 9.易用性设计

9

DNNVAD引擎采用c/c++接口，接口方便灵活，支持c/c++，java，python等变成语言调用，只需掌握基础的语言学习知识，即可自主设计功能模块和功能的实现。