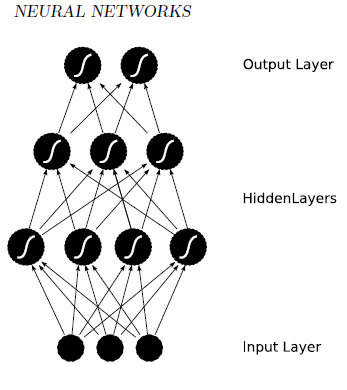
# DNNVAD源代码概要设计

## 1 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

## 2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.

}