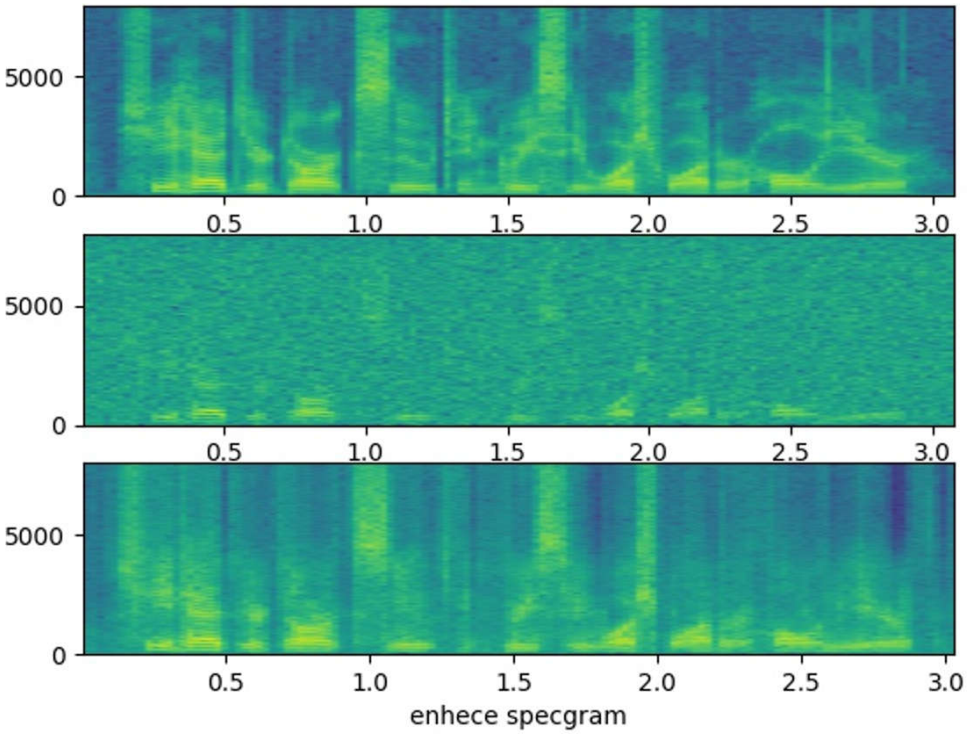
语音增强-DNN-IRM学习

Speech Enhancement- DNN based IRM Learning



于泓
鲁东大学
信息与电气工程学院
2021.8.7

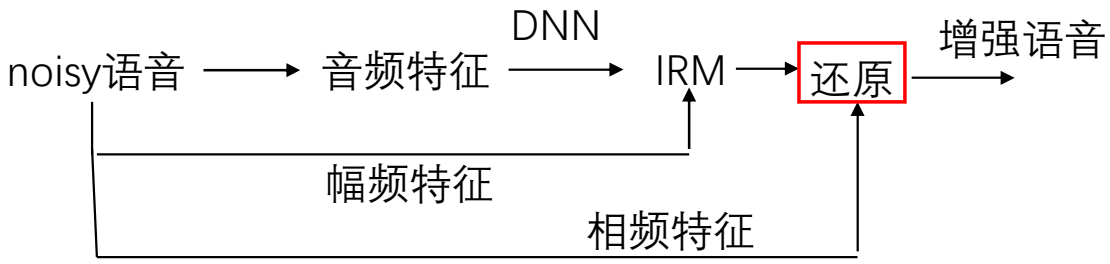
DNN IRM学习（DNN based IRM Learning）

IRM： Ideal Ratio Mask

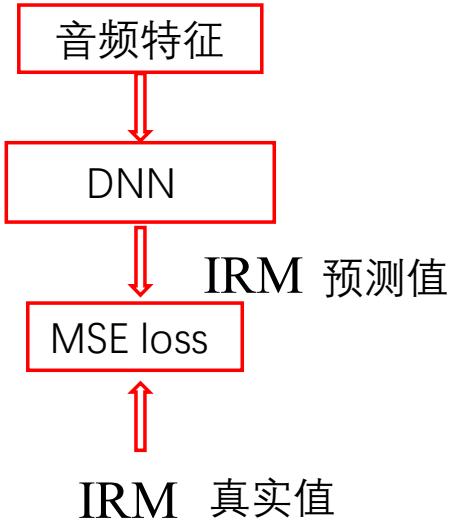
类似 维纳滤波中获取的滤波器，IRM的定义为：

$$\mathrm{IRM}_{t,f}=\left(\frac{Px_{t,f}}{Px_{t,f}+Pn_{t,f}}\right)^{\beta}$$

测试



将IRM 作为训练目标，利用DNN学习一个IRM提取器。



数据准备： 和DNN-Mapping 相同利用TIMIT数据库和Noise-92 生成 干净-含噪语音对。

Dataset： dataset.py

```
def feature_stft(wav,para):
    spec = librosa.stft(wav,
                        n_fft=para["N_fft"],
                        win_length = para["win_length"],
                        hop_length = para["hop_length"],
                        window =para["window"])

    mag =   np.abs(spec)
    phase = np.angle(spec)

    return mag.T, phase.T     #  T x D
```

计算IRM mask

```
def get_mask(clean,noisy,para):
    noise = noisy-clean

    clean_mag,_ =  feature_stft(clean,para)
    noisy_mag,_ =  feature_stft(noisy,para)
    noise_mag,_ =  feature_stft(noise,para)

    mask =   (clean_mag ** 2 / (clean_mag ** 2 + noise_mag ** 2))**(0.5)
    return clean_mag,noisy_mag,mask
```

```python
class TIMIT_Dataset(Dataset):

    def __init__(self,para):

        self.file_scp = para.file_scp
        self.para_stft = para.para_stft
        self.n_expand = para.n_expand

        files = np.loadtxt(self.file_scp,dtype = 'str')
        self.clean_files = files[:,1].tolist()
        self.noisy_files = files[:,0].tolist()

        print(len(self.clean_files))

    def __len__(self):
        return len(self.clean_files)
```

```python
def my_collect(batch):
    batch_X = [item[0] for item in batch]
    batch_Y = [item[1] for item in batch]
    batch_X = torch.cat(batch_X,0)
    batch_Y = torch.cat(batch_Y,0)
    return[batch_X.float(),batch_Y.float()]
```

```python
def __getitem__(self,idx):

    # 读取干净语音
    clean_wav,fs = sf.read(self.clean_files[idx],dtype = 'float32')
    clean_wav = clean_wav.astype('float32')

    #  读取含噪语音
    noisy_wav,fs = sf.read(self.noisy_files[idx],dtype = 'float32')
    noisy_wav = noisy_wav.astype('float32')

    # 进行 特征提取
    clean_mag,noisy_mag,mask = get_mask(clean_wav,noisy_wav,self.para_stft)

    # 转为torch格式
    X_train = torch.from_numpy(np.log(noisy_mag**2))
    Y_train = torch.from_numpy(mask)

    # 拼帧
    X_train = feature_contex(X_train,self.n_expand)
    Y_train = Y_train[self.n_expand:-self.n_expand,:]
    return X_train, Y_train
```

LPS 特征

model_IRM.py

```
class DNN_IRM(nn.Module):
    def __init__(self,para):
        super(DNN_IRM,self).__init__()
        self.dim_in = para.dim_in
        self.dim_out = para.dim_out
        self.dim_embeding = para.dim_embeding
        self.dropout = para.dropout
        self.negative_slope = para.negative_slope
```

```
self.model = nn.Sequential(
                nn.BatchNorm1d(self.dim_in),
                # 第一层
                nn.Linear(self.dim_in, self.dim_embeding),
                nn.BatchNorm1d(self.dim_embeding),
                nn.LeakyReLU(self.negative_slope),
                nn.Dropout(self.dropout),

                # 第二层
                nn.Linear(self.dim_embeding, self.dim_embeding),
                nn.BatchNorm1d(self.dim_embeding),
                nn.LeakyReLU(self.negative_slope),
                nn.Dropout(self.dropout),

                # 第三层
                nn.Linear(self.dim_embeding, self.dim_embeding),
                nn.BatchNorm1d(self.dim_embeding),
                nn.LeakyReLU(self.negative_slope),
                nn.Dropout(self.dropout),

                # 第四层
                nn.Linear(self.dim_embeding, self.dim_out),
                nn.BatchNorm1d(self.dim_out),
                nn.LeakyReLU(self.negative_slope),
                nn.Sigmoid()
                )

    for m in self.modules():
        if isinstance(m, nn.Linear):
            nn.init.xavier_normal_(m.weight.data)
```

```
def forward(self,x):
    out_mask = self.model(x)
    return out_mask
```

```python
import torch
import torch.nn as nn
from hparams import hparams
from torch.utils.data import Dataset,DataLoader
from dataset import TIMIT_Dataset,my_collect
from model_IRM import DNN_IRM
import os
```

```python
if __name__ == "__main__":
    # 定义device
    device = torch.device("cuda:0")

    # 获取模型参数
    para = hparams()

    # 定义模型
    m_model = DNN_IRM(para)
    m_model = m_model.to(device)
    m_model.train()

    # 定义损失函数
    loss_fun = nn.MSELoss()
    # loss_fun = nn.L1Loss()
    loss_fun = loss_fun.to(device)

    # 定义优化器
    optimizer = torch.optim.Adam(
        params=m_model.parameters(),
        lr=para.learning_rate)

    # 定义数据集
    m_Dataset= TIMIT_Dataset(para)
    m_DataLoader = DataLoader(m_Dataset,batch_size = para.batch_size,shuffle = True, num_workers = 4, collate_fn = my_collect)
```

```
# 定义训练的轮次
n_epoch = 100
n_step = 0
loss_total = 0
for epoch in range(n_epoch):
    # 遍历dataset中的数据
    for i_batch, sample_batch in enumerate(m_DataLoader):
        train_X = sample_batch[0]
        train_Y = sample_batch[1]

        train_X = train_X.to(device)
        train_Y = train_Y.to(device)

        # 得到网络输出
        output_mask = m_model(x=train_X)

        # 计算损失函数
        loss = loss_fun(train_Y,output_mask)

        # 误差反向传播
        optimizer.zero_grad()
        loss.backward()

        # 进行参数更新

        optimizer.step()

        n_step = n_step+1
        loss_total = loss_total+loss

        # 每100 step 输出一次中间结果
        if n_step %100 == 0:
            print("epoch = %02d  step = %04d  loss = %.4f"%(epoch,n_step,loss))

    # 训练结束一个epoch 计算一次平均结果
    loss_mean = loss_total/n_step
    print("epoch = %02d mean_loss = %f"%(epoch,loss_mean))
    loss_total = 0
    n_step =0

    # 进行模型保存
    save_name = os.path.join('save2','model_%d_%.4f.pth'%(epoch,loss_mean))
    torch.save(m_model,save_name)
```

模型参数

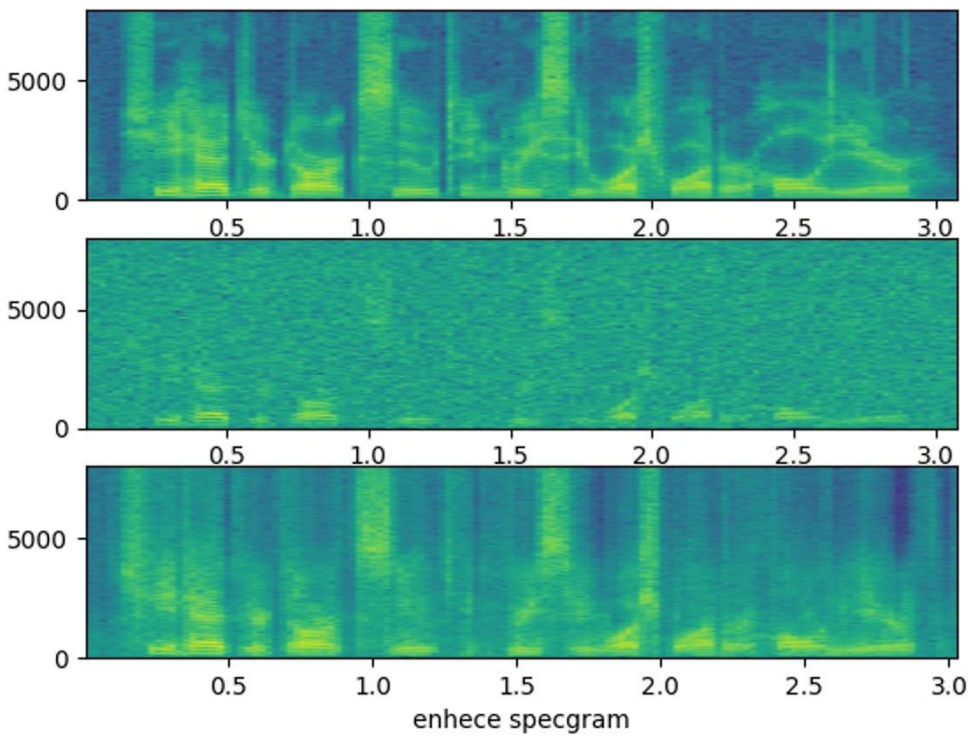
```
import torch
class hparams():
    def __init__(self):

        self.root_path = ""
        self.file_scp = "scp/train_DNN_enh.scp"
        self.snrs = [0,5,10,15,20]

        self.para_stft = {}
        self.para_stft["N_fft"] = 512
        self.para_stft["win_length"] = 512
        self.para_stft["hop_length"] = 128
        self.para_stft["window"] = 'hamming'

        self.n_expand = 3
        self.dim_in = int((self.para_stft["N_fft"]/2 +1)*(2*self.n_expand+1))
        self.dim_out = int((self.para_stft["N_fft"]/2 +1))
        self.dim_embeding = 2048
        self.learning_rate = 1e-2
        self.batch_size = 32
        self.negative_slope = 1e-1
        self.dropout = 0.1
```

测试部分：

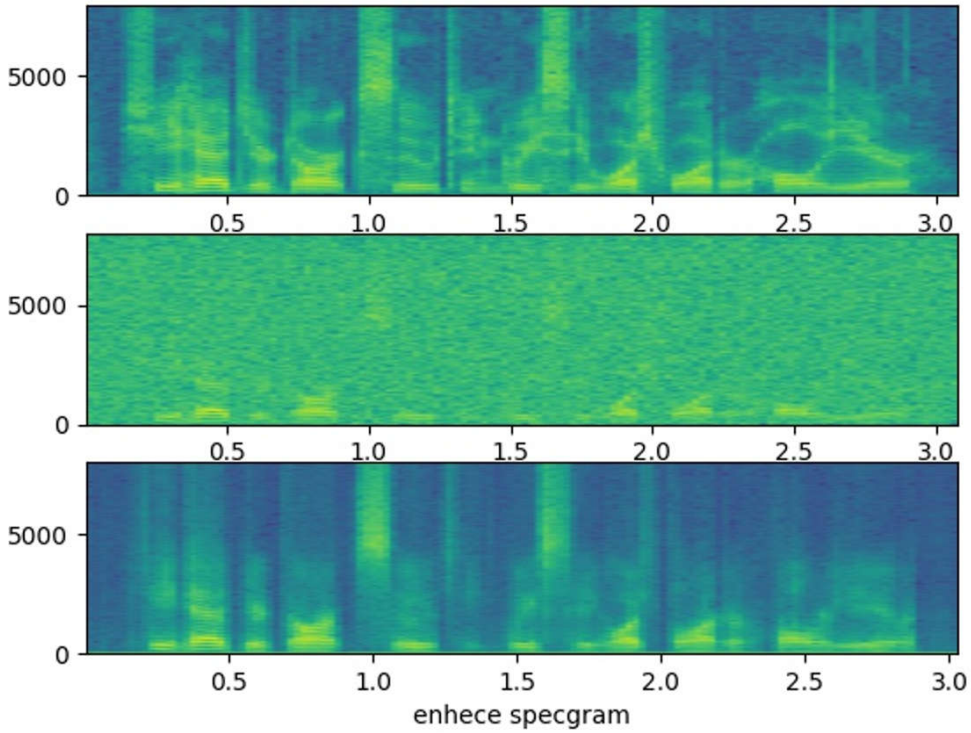
```
def eval_file_IRM(wav_file,model,para):

    # 读取noisy 的音频文件
    noisy_wav,fs = sf.read(wav_file,dtype = 'float32')
    noisy_wav = noisy_wav.astype('float32')

    # 提取LPS特征
    noisy_mag,noisy_phase = feature_stft(noisy_wav,para.para_stft)

    # 转为torch格式
    noisy_LPS = torch.from_numpy(np.log(noisy_mag**2))

    # 进行拼帧
    noisy_LPS_expand = feature_contex(noisy_LPS,para.n_expand)

    # 利用DNN进行mask计算
    model.eval()
    with torch.no_grad():
        enh_mask = model(x = noisy_LPS_expand)
    # 转为numpy格式
    enh_mask = enh_mask.numpy()

    enh_pahse = noisy_phase[para.n_expand:-para.n_expand,:].T
    enh_mag = (noisy_mag[para.n_expand:-para.n_expand,:]*enh_mask).T

    enh_spec = enh_mag*np.exp(1j*enh_pahse)

    # istft
    enh_wav = librosa.istft(enh_spec, hop_length=para.para_stft["hop_length"], win_length=para.para_stft["win_length"])
    return enh_wav
```

```python
if __name__ == "__main__":

    para = hparams()

    # 读取训练好的模型
    model_name = "save2/model_68_0.0337.pth"
    m_model = torch.load(model_name,map_location = torch.device('cpu'))

    snrs = [0,5]
    noise_path = '/home/sdh/dataset/noise/'
    noises = ['white']

    test_clean_files = np.loadtxt('scp/test_small.scp',dtype = 'str').tolist()
    test_clean_files = test_clean_files[:2]
    path_eval = 'eval'
    os.makedirs(path_eval,exist_ok=True)
    clean_path = '/home/sdh/dataset/TIMIT'

    for noise in noises:
        print(noise)
        noise_file = os.path.join(noise_path,noise+'.wav')
        noise_data,fs = sf.read(noise_file,dtype = 'int16')

        for clean_wav in test_clean_files:

            # 读取干净语音并保存
            clean_file = os.path.join(clean_path,clean_wav)
            clean_data,fs = sf.read(clean_file,dtype = 'int16')
            id = os.path.split(clean_file)[-1]
            sf.write(os.path.join(path_eval,id),clean_data,fs)
```

```
for snr in snrs:
    # 生成noisy文件
    noisy_file = os.path.join(path_eval,noise+'-'+str(snr)+'-'+id)
    mix = signal_by_db(clean_data,noise_data,snr)
    noisy_data = np.asarray(mix,dtype= np.int16)
    sf.write(noisy_file,noisy_data,fs)

    # 进行增强
    print("enhancement file %s"%(noisy_file))
    enh_data = eval_file_IRM(noisy_file,m_model,para)

    # 信号正则
    # max_ = np.max(enh_data)
    # min_ = np.min(enh_data)
    # enh_data = enh_data*(2/(max_ - min_)) - (max_+min_)/(max_-min_)
    enh_file = os.path.join(path_eval,noise+'-'+str(snr)+'-'+'enh'+'-'+id)
    sf.write(enh_file,enh_data,fs)

    # 绘图
    fig_name = os.path.join(path_eval,noise+'-'+str(snr)+'-'+id[:-3]+'jpg')

    plt.subplot(3,1,1)
    plt.specgram(clean_data,NFFT=512,Fs=fs)
    plt.xlabel("clean specgram")
    plt.subplot(3,1,2)
    plt.specgram(noisy_data,NFFT=512,Fs=fs)
    plt.xlabel("noisy specgram")
    plt.subplot(3,1,3)
    plt.specgram(enh_data,NFFT=512,Fs=fs)
    plt.xlabel("enhece specgram")
    plt.savefig(fig_name)
```



IRM



DNN-Mapping

其他:

$$\mathrm{CRM}_{t,f}=\left(\frac{Px_{t,f}}{Py_{t,f}}\right)^{0.5}$$

$$\mathrm{PSM}_{t,f}=\frac{Ax_{t,f}\cdot\cos\left(\theta y_{t,f}-\theta x_{t,f}\right)}{Ay_{t,f}}$$

$$\mathrm{IBM}_{t,f}=\begin{cases}1 & Px_{t,f}>Pn_{tf}\\ 0 & Px_{t,f}<Pn_{tf}\end{cases}$$

实部，虚部分开 分别计算CRM

其他: