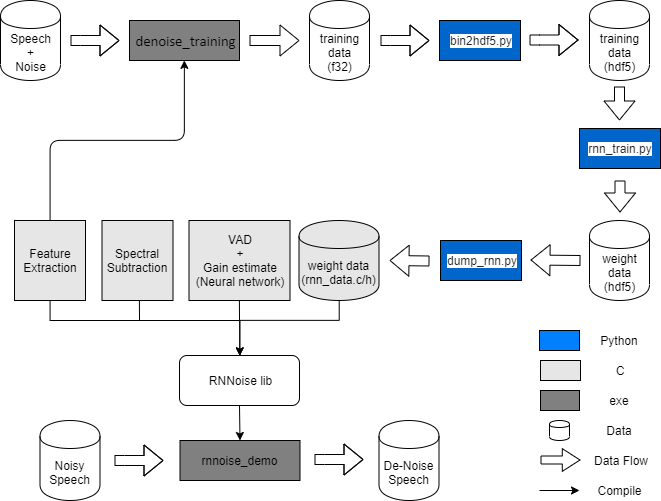
代码来自: <https://github.com/xiph/rnnoise>

# 1. 代码包含了什么?

如下图所示, RNNoise代码库的可以总结为两个部分:

* 对神经网络进行训练的流程(缺少训练数据生成/构造部分)
* 包含训练权重的RNNoise库和降噪demo



# 2. 工程的目录结构

该工程只有两层目录. 这里列出了全部目录和文件名, 并稍作了注释.

\

│ AUTHORS 作者信息

│ COPYING license信息

│ README 简介和操作命令

│ TRAINING 训练的操作命令

│ autogen.sh 以下为编译相关部分

│ configure.ac

│ Makefile.am

│ update\_version

│ rnnoise-uninstalled.pc.in

│ rnnoise.pc.in

│

├───doc

│ Doxyfile.in 用于生成文档的配置

│

├───examples

│ rnnoise\_demo.c 使用库, 对输入的语音做降噪处理

│

├───include

│ rnnoise.h 库函数接口

│

├───m4

│ attributes.m4 宏，用于检查通用（非类型）符号的存在

│

├───src

│ compile.sh denoise\_training的编译脚本

│ denoise.c 库的相关接口函数, 以及denoise\_training的main函数

│ rnn.c/h 神经网络的计算函数, 包括GRU和全连接层(Dense)

│ rnn\_data.c/h 生成的权重数据, 以权重->层(layer)->模型的架构来组织的

│ rnn\_reader.c 载入和释放模型数据的函数

│ celt\_lpc.c/h CELT相关代码

│ kiss\_fft.c/h, \_kiss\_fft\_guts.h [kiss fft](http://kissfft.sourceforge.net/)的相关代码

│ pitch.c/h 基音(pitch)相关代码

│ tansig\_table.h 双曲正切S型函数的速查数据

│ arch.h, opus\_types.h 架构和平台相关的定义和宏

│ common.h 内存操作相关的宏/接

│ rnn\_train.py 另外一个版本的训练脚本

└───training

bin2hdf5.py 转换训练数据的格式: f32-> [hdf5](https://www.h5py.org/), 需要指定矩阵维度

dump\_rnn.py 把权重转换为c代码, 到src\rnn\_data.h|c

rnn\_train.py 训练脚本, 包括构造模型, 载入和分配数据, 训练, 保存权重

# 3. 对Windows的支持

这个工程应该主要是考虑了linux平台的用户,

在windows平台上使用有些小问题, 怎么在windows上使用? 修改? 相关环境?

* Windows上下载会存在问题, 根目录下的TRAINING和training目录会冲突, 因为Windows上的文件名是大小写不敏感的, 不能允许这两个文件和目录同时存在.解决方法: 手动从github下载缺失的部分, 或者从解压包解压缺失的部分, 并给他们不同的名字, 比如把TRAINING文件改为TRAINING.txt.
* 编译方面会有些问题, 有人给出了修改 <https://github.com/xiph/rnnoise/pull/88> 和 <https://github.com/mumble-voip/rnnoise/commit/b30f2bb8049601c3b7253d47f75dc6a68c5f32fd>

# 4. 搭建Windows下的训练环境

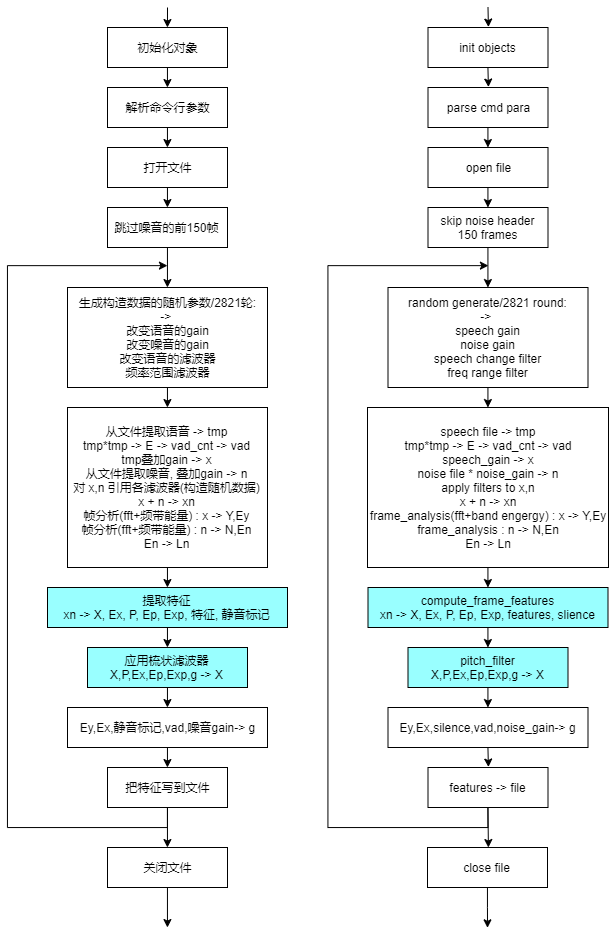
# 5. 代码解读

我们关注训练逻辑和谱减算法这两部分的代码, 后者还可以细分为训练数据构造, 特征提取, 增益计算(使用神经网络计算每个频带的增益), 以及谱减操作. 重头戏是特征提取.

## 3.1. 训练数据构造和特征提取

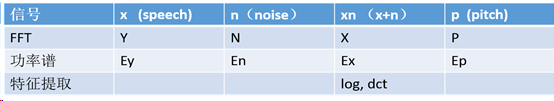
### 程序分支

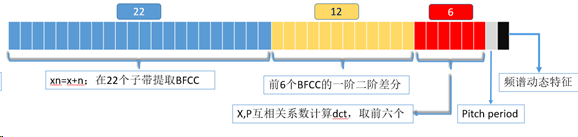
数据的构造和特征提取被放在一起实现, 其流程图如下:



### 数据总结

博客<https://blog.csdn.net/dakeboy/article/details/88039977> 提供了非常好的数据总结, 我们借用了博主dakeboy的成果如下:





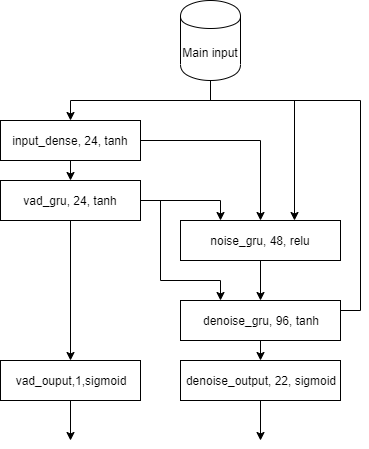
### 重要函数

## 3.3. 训练逻辑

训练逻辑的代码在rnn\_train.py, 完全基于keras实现, 包括四个部分: 模型构造, 数据载入和组织, 训练和模型保存. 这里可以参考keras的[文档](https://keras.io/).

### 模型构造

* 参数constraint = WeightClip(0.499), 将权重(weight)和偏移(bias)的值约束在-0.499到0.499之间.
* 所有的GRU的循环时间步的激活函数采用sigmod, 而不是默认的hard sigmoid(这是什么?). 图上并未标出.
* 激活函数的使用则有三种之多, 与论文中给出的不一致(有何差别?).
* GRU的参数return\_sequences = true (return\_sequences决定在输出序列中，返回最后一个输出值 (hidden state) 还是返回全部time step 的输出(hidden state)值。 False 返回单个， true 返回全部。注意区别于return\_state, 后者决定除了输出(hidden state)之外是否返回最后一个状态(cell state), 更多的细节可以参考<https://blog.csdn.net/u011327333/article/details/78501054> 和 <https://huhuhang.com/post/machine-learning/lstm-return-sequences-state>)
* GRU的kernel\_regularizer(施加在权重上的正则项)和recurrent\_regularizer(施加在循环核上的正则项)都是regularizers.l2(0.000001)
* 优化器为adam
* 损失函数是模型想要最小化的函数. 对应于输出的两个部分denoise\_output和vad\_output, 采用两个不同的损失函数, 分别是自定义的mycost和my\_crossentropy, 他们的权重关系是10:0.5.
  + Mycost比较复杂,
    - 首先会乘一个mymask这个函数, 他的功能是对标记增益作+1后再限1操作, (这里增益是什么单位?线性?, 需要从特征提取部分找答案) (为什么这么做?)
    - 然后令d=(sqrt(预测增益)-sqrt(标记增益))2,核心部分为10\*d2+d+0.01\*标准的交叉熵损失函数.(为什么这么做?)
  + 而my\_crossentropy也并不是如论文所说的是标准的交叉熵损失函数, 而是乘了一个标记值和中位值0.5的距离的两倍这么一个值(这应该是为了应对标记数据不自信的情况吧? 需要从特征提取部分找答案), 还做了求平均动作(对什么维度求平均?),
* Metrics: 用来评价模型的函数, 该评价结果不在训练模型时中使用. 这里用的是是自定义的mmse,(它的实现是mymask\*mycost里面的d),这里没有两个函数, 也没有使用权重, 应该是两个输出都是用了标准吧(keras实验来回答)



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

main\_input (InputLayer) (None, None, 42) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

input\_dense (Dense) (None, None, 24) 1032 main\_input[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

vad\_gru (GRU) (None, None, 24) 3528 input\_dense[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate\_1 (Concatenate) (None, None, 90) 0 input\_dense[0][0]

vad\_gru[0][0]

main\_input[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

noise\_gru (GRU) (None, None, 48) 20016 concatenate\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate\_2 (Concatenate) (None, None, 114) 0 vad\_gru[0][0]

noise\_gru[0][0]

main\_input[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

denoise\_gru (GRU) (None, None, 96) 60768 concatenate\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

denoise\_output (Dense) (None, None, 22) 2134 denoise\_gru[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

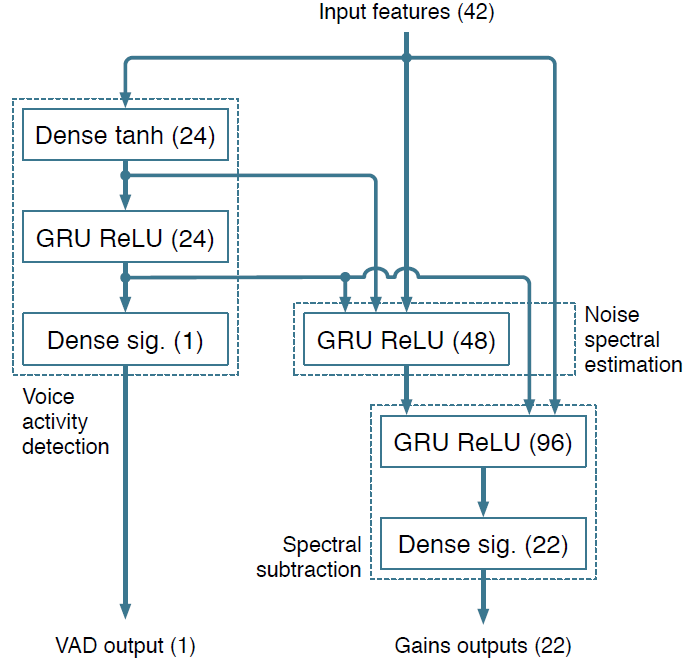
vad\_output (Dense) (None, None, 1) 25 vad\_gru[0][0]

==================================================================================================

Total params: 87,503

Trainable params: 87,503

Non-trainable params: 0



### 数据载入和组织

载入数据的过程是通过h5py打开训练文件, 然后把data字段中的所有数据塞到all\_data里面.

with h5py.File('training.h5', 'r') as hf:

    all\_data = hf['data'][:]

这里得到的all\_data应该是一个nx87的张量, 其数据内容如下:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据下标 | 0…41 | 42…63 | 64…85 | 86 |
| 数据大小 | 42 | 22 | 22 | 1 |
| 数据名称 | 输入数据 | 目标增益 | 噪音数据 | Vad数据 |
| 备注 |  |  | 未使用 |  |

这里设定了一个变量

window\_size = 2000

把数据从时域上分割成长度为2000的小片, 这个应该是和采样率有关的, 所以调整采样率的时候, 这个只应该调整, 在48k采样率下, 2000个sample(一组特征对应一个sample? 还是一帧?如果是一个480采样的帧(10ms), 那应该是2000个10ms的帧, 则是相当于20秒, 这个需要了解前面的部分)相当于1/24秒, 大概是40ms.

我们根据window\_size得到一个nb\_sequence, 这实际上是样本的数目, 我们会看到x\_train等变量的shape是(nb\_sequence, window\_size, 42), 当数据整列是500000x87时, 实际上样本数只有250个40ms的音频片段.

### 训练

调用[model.fit()](https://keras.io/zh/models/model/" \l "fit)进行训练

model.fit(x\_train,

          [y\_train, vad\_train], # 使用目标增益和vad标志作为训练目标

          batch\_size=batch\_size, # 32, 每次梯度更新之间使用的样本数

          epochs=120, #模型训练迭代120轮

          validation\_split=0.1) # 用作验证集的数据比例

比较重要的默认参数:

# shuffle=True, 是否在每轮迭代之前混洗数据

# initial\_epoch=0, 开始训练的轮次

# steps\_per\_epoch=None, 在声明一个轮次完成并开始下一个轮次之前的总步数, 使用 TensorFlow 数据张量等输入张量进行训练时，默认值 None 等于数据集中样本的数量除以 batch 的大小，如果无法确定，则为 1

# validation\_steps=None, 停止前要验证的总步数（批次样本）, 只有在指定了 steps\_per\_epoch 时才有用。

# validation\_freq=1,整数或列表/元组/集合。如果为整数，则指定在执行新的验证运行之前要运行多少个训练时期，例如 validation\_freq=2，每2个时期运行一次验证。如果列表，元组或集合指定了要运行验证的时期，例如validation\_freq=[1, 2, 10]在第一个，第二个和第十个时期的末尾运行验证。仅在提供验证数据时才相关。

这里具体的训练是怎么进行的呢? 需要进一步了解.

数据送进去, 是作为一段语音还是每一采样? 考虑到应用RNN, 不应该在时域上洗牌, 而且要保证语音的连续性

怎么样计算loss, 针对整个语音段? 怎样优化?

分成5000, 那前后相关性如何? Gru状态是否会丢失?

批次的意义又在哪里?

轮呢?

如果我们不指定轮会怎样?

### 模型保存

调用model.save(filepath)可以把Keras模型保存到HDF5文件, 其中包含了以下信息:

- 模型架构信息

- 模型权重

- 训练配置(损失函数, 优化器)

- 优化器的状态(用于从训练结束的地方继续训练)

### 数据可视化

<https://keras.io/visualization/#model-visualization>

## 3.4. 增益计算

## 3.5. 谱减操作

# 6. 文档, 交流和论坛

此项目的可以自动生成一些文档

此项目在github上的issure区有不少交流信息 <https://github.com/xiph/rnnoise/issues>

在作者的博客上也有一些交流信息 <https://jmvalin.dreamwidth.org/15210.html>