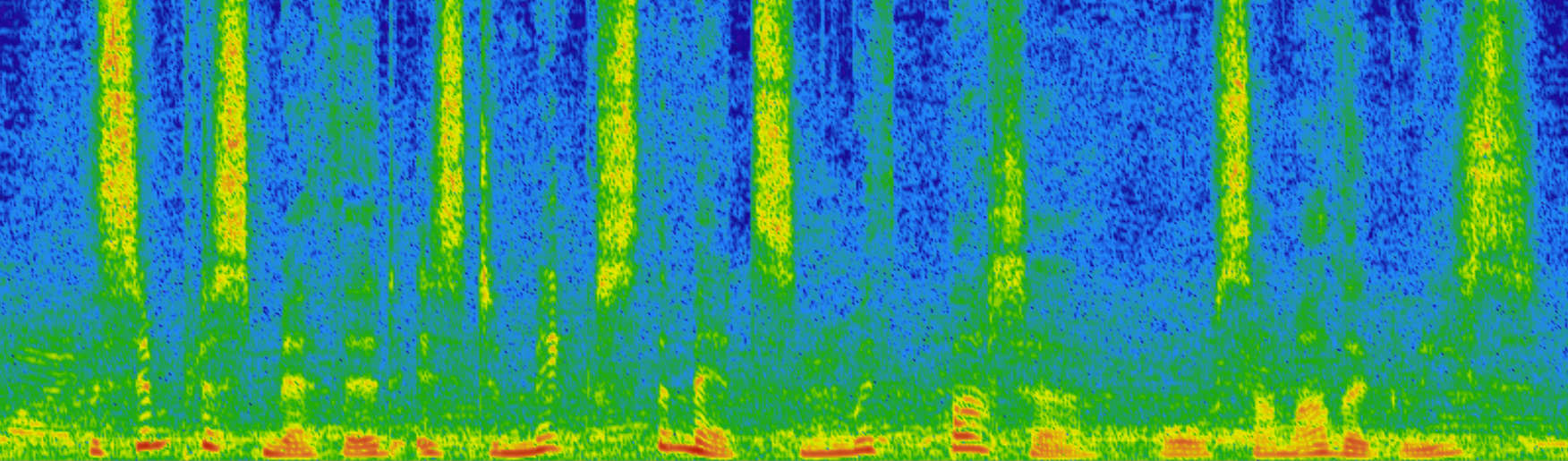
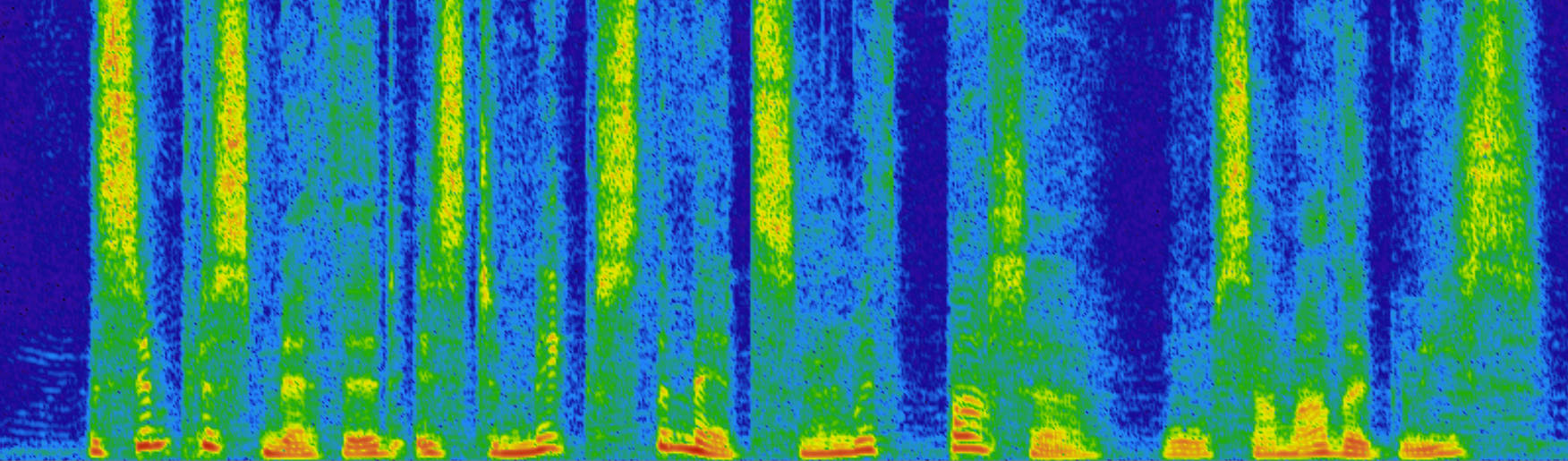
RNNoise：学习噪声抑制

<https://people.xiph.org/~jm/demo/rnnoise/>

[（Monty的演示页面）](https://people.xiph.org/~xiphmont/demo/index.html)





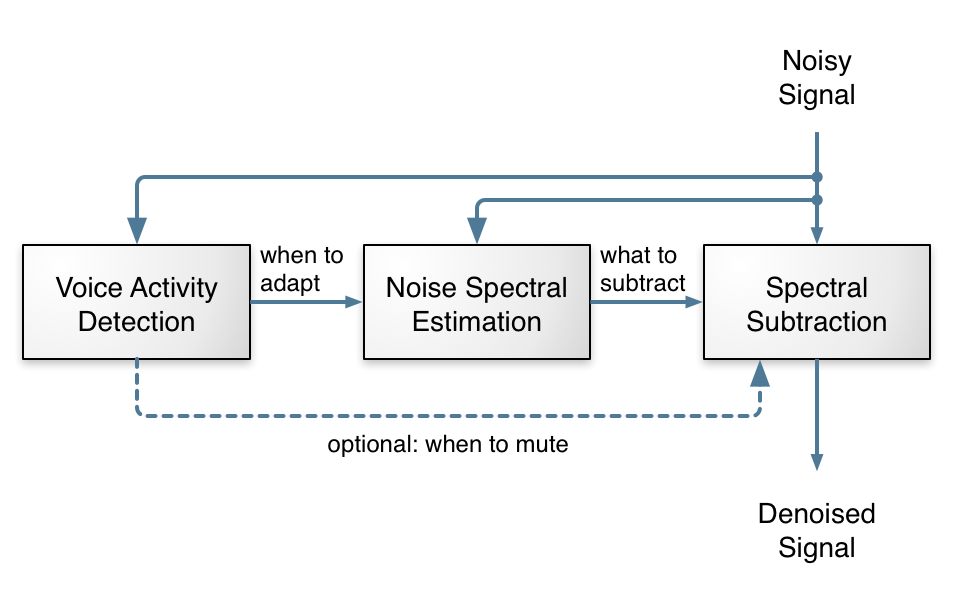
上图显示了噪声抑制前后（上图为原语音, 下图为处理后）音频的声谱图。

# 这是RNNoise

该演示演示了RNNoise项目，展示了如何将深度学习应用于噪声抑制。主要思想是将经典信号处理与深度学习相结合，以创建一种小型且**快速**的实时噪声抑制算法。无需昂贵的GPU, 它可以在树莓派(Raspberry Pi)上轻松运行。它比传统的噪声抑制系统更简单（更容易调音），并且听起来更好。

# 噪声抑制

噪声抑制是语音处理中的一个相当老的话题 [，至少可以追溯到70年代](http://ieeexplore.ieee.org/document/1163209/)。顾名思义，该想法是获取噪声信号并消除尽可能多的噪声，同时使目标语音的失真最小。



这是常规噪声抑制算法的概念图。语音活动检测（VAD）模块可以**检测信号何时包含语音以及何时只是噪音**。噪声频谱评估模块(NSE) 使用它来**计算噪声的频谱特性**（每个频率多少功率）。然后，知道了噪声的样子，就可以谱减模块(SS)从输入音频中“减去”噪声（听起来并不简单）。

从上图可以看出，噪声抑制看起来很简单：只有三个概念上简单的任务就完成了，对吧？对错参半吧！随便一个电子工程专业的本科生都可以编写一种有效的噪声抑制算法，有时……。有难度的是要在各种噪音条件下始终保持良好的性能。这就需要对许多特殊情况和奇怪信号, 非常仔细地调整算法中的每个旋钮，并进行大量测试。总会有一些奇怪的信号会引起问题并需要更多的调整，而且总是这种修复总是没完没了。这是50％的科学，50％的艺术。我以前曾在[speexdsp库中](https://speex.org/)也使用过噪声抑制器。它也能用，但是效果不是很好。

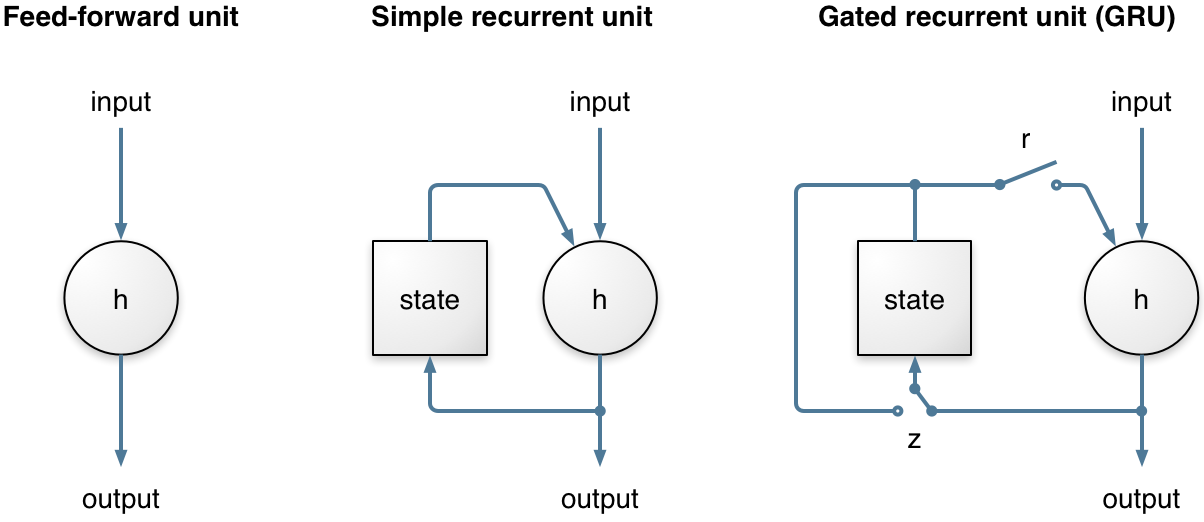
# 深度学习和递归神经网络

深度学习是人工神经网络的新版本。尽管这些技术自60年代就已经存在，但近年来发生了一些新变化：

1. 现在我们知道如何超过两个隐藏层的深度网络
2. 我们知道如何让递归网络记住过去的模式
3. 我们有计算资源来实际训练他们

递归神经网络（RNN）在这里非常重要，因为它们使建模时间序列成为可能，而不仅仅是独立考虑输入和输出帧。这对于抑制噪声尤为重要，因为我们需要时间来很好地估计噪声。长期以来，RNN的能力受到严重限制，因为它们无法长时间保存信息，并且由于在时间上反向传播时涉及的梯度下降过程效率非常低（梯度消失问题）。门控单元的发明解决了这两个问题，它有许多变体, 例如长短期记忆（LSTM），门控循环单元（GRU）等。

RNNoise使用门控循环单元（GRU），因为它在此任务上的性能比LSTM稍好，并且需要较少的资源（包括CPU和保存权重的内存）。与简单的循环单元相比，GRU具有两个额外的门。重置门(Reset gate) 控制是否使用历史状态 (state) 来计算新状态，而更新门 (update gate) 控制新输入将在多大程度上改变存储的状态值 (state) 。这个更新门（关闭时）使GRU可以（很容易）长时间记住某个信息，这就是GRU（和LSTM）的性能比简单循环单元好得多的原因。



将简单的循环单元与GRU进行比较。区别在于GRU的r和z 门，这使得学习长期模式成为可能。两者都是基于整个层的历史状态和输入计算出来的软开关（介于0和1之间的值），具有S型激活函数。当更新门Z在左侧时，状态可以在很长一段时间内保持恒定-直到某种情况导致z切换到右侧。

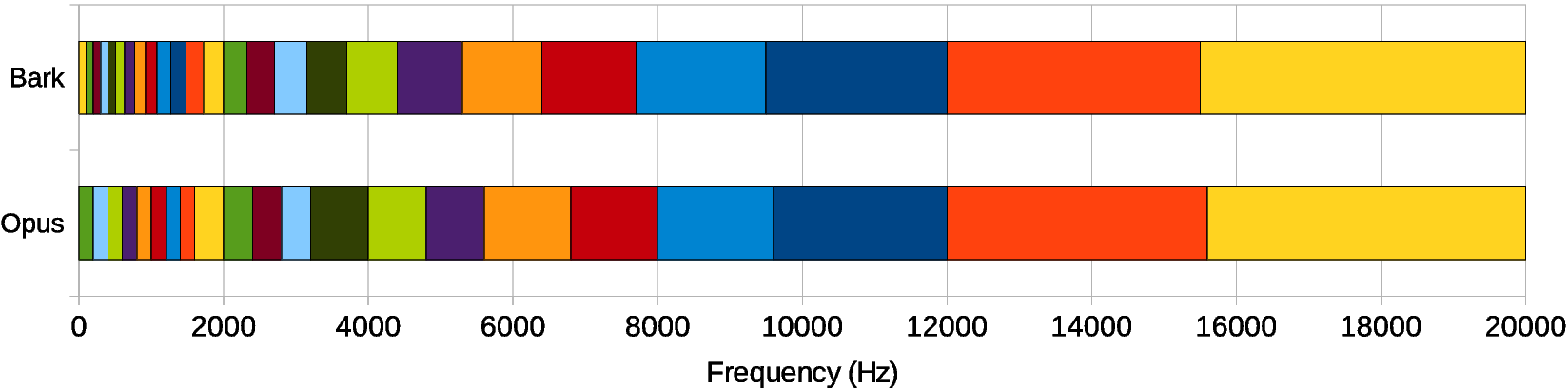
# 混合方案

得益于深度学习的成功，现在流行将深度神经网络来解决整个问题上。这些方法称为端到端 –只有神经元 (没有其他处理模块) 。端到端方法已应用于[语音识别](https://arxiv.org/pdf/1412.5567.pdf)和[语音合成](https://deepmind.com/blog/wavenet-generative-model-raw-audio/)方面，这些端到端系统已经证明了深度神经网络的强大功能。另一方面，这些系统有时可能不够理想，而且浪费资源。例如，某些噪声抑制方法使用具有数千个神经元的层以及数以千万计的权重来执行噪声抑制。缺点不仅是运行网络的计算成本，还在于规模, 因为现在是一千行代码以及数十兆字节的神经元权重值（搞不好还更多）。

这就是为什么我们在这里采用了不同的方法的原因：始终保留所有必需的基本信号处理（没有神经网络尝试对其进行仿真），但是让神经网络学习所有需要在信号处理旁边进行无休止调整的棘手部分。和那些现有的噪声抑制的深度学习实现不同, 我们的目标是实时通信, 而不是语音识别, 我们不能预读太多的数据（这次是10毫秒） 。

## 定义问题

为了避免产生大量输出，以及避免使用大量神经元，我们决定不直接使用采样数据或频谱。相反，我们使用基于巴克刻度(Bark scale)划分的频率带(frequency bands)，这种频率刻度符合我们感知声音的方式。我们总共使用22个频段，而不是480个频谱值（复杂）。



Opus频带 (bands) 的分布v.s.巴克刻度。对于RNNoise，我们使用与Opus相同的基本布局。由于频带间存在重叠，因此Opus频带之间的边界成为重叠的RNNoise频带的中心。在较高频率下, 由于耳朵对频率的分辨精度较差，因此频带划分的比较宽。在低频下，频带较窄，但比巴克刻度所给定的宽，否则我们将没有足够的数据来进行良好的估算。

当然，我们不能仅从22个频带的能量中重建音频。但是，我们可以为每个频带计算一个增益，并用在信号之上。您可以将他视作一个22段的EQ (均衡器) 并快速更改每个频带的电平以衰减噪声，同时让信号通过。

计算频带增益有几个优点。首先，由于要计算的频带较少，因此模型更简单。其次，它不可能产生所谓的音乐噪音伪像（musical noise artifacts），因为其相邻声音会被衰减, 只有单音会通过。这些伪像在噪声抑制中很常见，而且很难处理。频带足够宽时，我们要么让整个频带通过，要么都剪掉。第三个优势来自我们如何优化模型。由于增益总是限制在0和1之间，只需使用S型(sigmoid)激活函数（其输出也在0和1之间）来计算它们，就可以确保计算正确，首先是不引入噪声。

对于输出，我们还可以选择ReLU(整流线性激活函数, rectified linear activation function)来表示0到无穷大之间的衰减(dB)。为了在训练过程中更好地优化增益，损失函数(loss function)采用pow(增益, α)的均方误差（MSE）。到目前为止，我们发现α=0.5时产生的结果在感知上是最佳的。使用α→0等效于最小化对数谱间距 (LSD, Log Spectral Distance) ，这是有问题的，因为最佳增益可能非常接近零。

分频带计算的主要缺点是, 在分辨率太低时, 不能有效地抑制音高谐波之间的噪声。还在这并不是那么重要，有一个简单的技巧可以处理（请参见下面的音高滤波部分）。

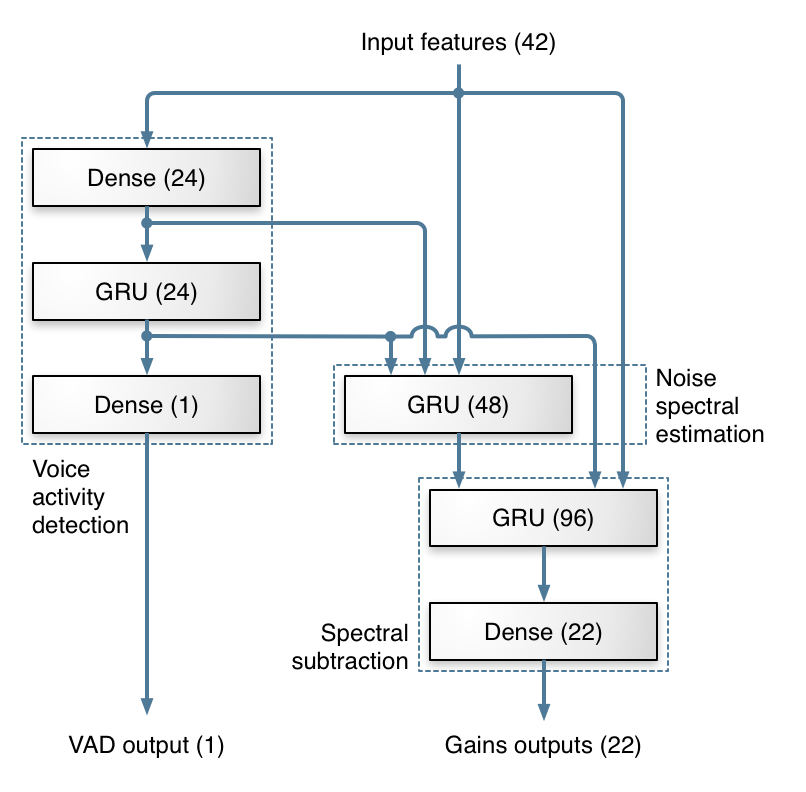
由于我们计算输出是基于22个频带的，因此在输入上使用更高的频率分辨率没什么意义，因此我们使用相同的22个频带将频谱信息输入神经网络。因为音频具有**很大的**动态范围，所以计算能量的对数值要好于直接提供能量值。然后, 对该数值执行DCT (离散余弦变换) 对特征进行去相关 (Decorrelate) 。所得数据是基于巴克刻度的倒谱(BFCC)，它与语音识别中常用的梅尔倒谱系数（MFCC）密切相关。

除倒谱系数外，我们还使用了以下数据作为神经网络的输入特征(22+6x2+1+6+1 = 42)：

* 前6个BFCC系数的一阶和二阶导数
* 基音周期（1 /基频）
* 6个频段的基音增益（发声强度）
* 一个特殊的非平稳值，可用于检测语音（超出了本Demo的范围）

## 深度学习架构

我们网络架构灵感来自传统的噪声抑制方法。大部分工作由3个GRU层完成。下图显示了我们用于计算频带增益的层，以及该体系结构如何映射到噪声抑制的传统步骤。当然，与其他神经网络一样，我们没有实际的证据证明网络正在按预期那样运作，但是该结构比我们尝试的其他结构都工作得更好, 这让我们有理由认为它符合我们设计预期。



本项目中使用的神经网络结构。每个方框代表一层神经元，括号中标出单位数。致密层(Dense) 是完全连接的非循环层。网络的输出之一是一组应用于不同频率的增益。另一项输出是VAD概率，该输出虽未用于噪声抑制，也非常有用的。

## 关于数据

深度神经网络有时也可能很笨。他们擅长于已学的知识，但是在面对差别很大的新知识时, 容易犯下非常严重的错误。更糟糕的是，他们真的是懒惰的学生。如果他们可以在训练过程中使用任何漏洞来偷懒，他们一定会那么做。这就是为什么训练数据的质量至关重要的原因。

一个广为流传的故事是，很久以前，一些军队研究人员正试图训练一个神经网络来识别树木伪装的坦克。他们为有和没有坦克的树木拍照，然后训练了神经网络来识别有坦克的树木。网络成功超出预期！只有一个问题。由于带有坦克的照片是在阴天拍摄的，而没有坦克的照片是在晴天拍摄的，因此网络真正学到的是晴天和阴天有何不同。尽管研究人员现在已经意识到了这个问题并避免了此类明显的错误，但它仍然可以以更难察觉的方式存在（并且过去确实发生在您的身上）。

在噪声抑制的情况下，我们不能仅仅收集可用于监督学习的输入/输出数据，因为我们很少同时获得干净的语音**和**嘈杂的语音。我们必须根据清晰的语音和噪音录音来人为地创建该数据。棘手的部分是获取各种各样的噪声数据以添加到语音中。我们还必须确保覆盖所有录音条件。例如，当音频以8 kHz进行低通滤波时，仅接受全频带音频（0-20 kHz）训练的早期版本就会失败。

与语音识别不同，我们选择不对特征进行倒谱平均归一化，并且保留表示能量的第一个倒谱系数。因为我们必须确保数据包括所有现实音量的音频。我们还对音频使用随机滤波器，使系统对各种麦克风频率响应具有鲁棒性（通常由倒谱均值归一化处理）。

## 基音(pitch)滤波

由于我们频带的频率分辨率太粗糙，无法滤掉基音谐波之间的噪声，因此我们使用基本信号处理来做到这一点。这是混合方案的另一部分。当对同一变量进行多次测量时，提高准确性（降低噪声）的最简单方法就是计算平均值。显然，只计算相邻音频样本的平均值是不合适的，因为这相当于低通滤波。但是，当信号是周期性的（例如浊音）时，我们可以计算出偏离基音周期的样本平均值(we can compute the average of samples offset by the pitch period)。这样就得到一个梳状滤波器, 使谐波可以通过，同时衰减它们之间的频率成分 (噪声) 。为避免信号失真，梳状滤波器将独立应用于每个频带，其滤波器强度既取决于基音相关性，也和神经网络计算出的频带增益有关。

目前，我们使用FIR滤波器进行基音滤波，但也可以（在TODO列表中）使用IIR滤波器，如果强度过高，则会导致更大的噪声衰减，因为有更大失真的风险。

## 从Python到C

本神经网络的所有设计和训练都是用强大的[Keras](https://keras.io/)深度学习库和python完成的。由于Python通常不是实时系统的首选语言，因此我们必须以C语言实现运行时代码。幸运的是，运行神经网络远比训练一个神经网络容易，我们所要做的就是实前向传播和GRU层。为了将精简模型，我们在训练过程中将权重的大小限制为+/-0.5，以便可以用8bit值来存储它们。生成的模型精简到了85kB（如果用32位浮点数存储则是340kB）。

C代码可以在BSD许可下[可用](https://git.xiph.org/?p=rnnoise.git;a=summary)。尽管在编写本演示时，代码尚未优化，但在x86 CPU上，它的运行速度比实时运行快60倍。它甚至比Raspberry Pi 3上的实时运行速度快7倍左右。借助良好的矢量化（SSE / AVX），应该可以使其比当前速度快4倍。

# 给我看例子！

好的，这很好，但是实际上**听起来**如何？[原文网站](https://people.xiph.org/~jm/demo/rnnoise/)上有一些实际的RNNoise示例，可消除三种不同类型的噪声。这些噪音和干净的语音都没有在训练期间使用过。

译注:请访问[原文网站](https://people.xiph.org/~jm/demo/rnnoise/)体验算法效果, 其中包含了:

* 无降噪/RNNoise/Speexdsp三种算法状态
* 0~10dB的SNR, 以及干净语音
* 三种噪音类型: 嘈杂噪声(Babble Noise), 汽车噪音(Car noise), 街道噪音(Street Noise)
* 相关语音也可以从<https://people.xiph.org/~jm/demo/rnnoise/samples/>下载

与无抑制和与Speexdsp噪声抑制器相比，评估RNNoise的效果。尽管提供的SNR低至0 dB，但我们针对的大多数应用（例如WebRTC呼叫）的SNR往往更接近20 dB，而不是0 dB。

那么，无论如何，您应该听什么？也许我的说法很奇怪，你**不**应该期待声音变得更清晰(intelligibility)。人类非常擅长于理解噪声中的语音，而增强算法只能破坏信息（尤其是不能在降噪之前预读语音的情况下）。那么，为什么我们要还要这样做呢？答案是为了质量(Quality), 让处理后的语音听起来不那么嘈杂，减少听觉疲劳。

实际上，在有些场景下它可以提高清晰度。第一个是有多个扬声器输入混合在一起视频会议。这时噪声抑制可防止来自所有非活跃扬声器的噪声与活跃扬声器混合在一起，从而提高质量和清晰度。第二种情况是语音需要通过低比特率编解码器。与干净的语音相比，嘈杂的语音在此情况下会变得更差，因此消除噪声可使编解码器效果更好。

## 试一下您的声音！

对以上样本不满意？实际上，(在[原文网站](https://people.xiph.org/~jm/demo/rnnoise/)上)您可以通过麦克风进行录制，并（近乎）实时地对音频进行降噪。如果单击下面的按钮，RNNoise将在浏览器中的Javascript中执行噪声抑制。该算法是实时运行的，但我们故意**将其延迟了几秒钟**，以使其更容易听到降噪后的输出。 **确保戴上耳机，否则您会听到反馈循环。**要开始演示，请选择“无抑制”或“ RNNoise”。您可以在两者之间切换以查看抑制效果。如果您的输入没有足够的噪声，则可以通过单击“白噪声”按钮来人为添加一些噪声。

## 将您的噪音捐赠给科学

~~如果您认为这项工作有用，有一种简单的方法可以使它变得更好！只需要一分钟的时间。单击下面的链接，让我们记录您所在位置的一分钟噪音。该噪声可用于改善神经网络的训练。作为附带的好处，这意味着网络将知道您所拥有的噪声类型，并且在将其用于视频会议时（例如在WebRTC中）可能会做得更好。我们对您可能通过语音进行交流的任何环境中的噪音感兴趣。可以是您的办公室，汽车，街道，或可能使用手机或计算机的任何地方。~~

感谢所有捐赠他们的噪音的人。现在[可以免费下载数据](https://people.xiph.org/~jm/demo/rnnoise/rnnoise_contributions.tar.gz)（6.4 GB）。有关更多详细信息，请参见随附的自述文件。

# 从哪里来？

如果您想进一步了解RNNoise的技术细节，请参阅[本文](https://arxiv.org/pdf/1709.08243.pdf)（尚未提交）。该代码仍在积极开发中（没有冻结的API），但已在应用程序中使用。它当前针对VoIP/视频会议应用程序，但是经过一些调整，它可能可以应用于许多其他任务。一个明显的目标是自动语音识别（ASR），尽管我们可以对嘈杂的语音进行降噪并将输出发送到ASR，但这不是最佳选择，因为它会丢弃有关过程固有不确定性的有用信息。当ASR不仅知道最可能的纯净语音，而且知道它可以在多大程度上依赖该估计时，它会有用得多。另一种可能的“定位”是实现一个更智能化的电子乐器的噪音抑制器(noise gate)。它需要做的就是提供良好的训练数据，并对代码进行一些更改，以将Raspberry Pi变成真正的吉他噪声抑制器。有没有人想试试？可能还有许多我们还没考虑到的潜在应用。

如果您想对此Demo发表评论，可以在[此处进行评论](https://jmvalin.dreamwidth.org/15210.html)。

— Jean-Marc Valin（[jmvalin@jmvalin.ca](mailto:jmvalin@jmvalin.ca)）2017年9月27日

# 其他资源

1. 代码：[RNNoise Git存储库](https://git.xiph.org/?p=rnnoise.git;a=summary) （[Github镜像](https://github.com/xiph/rnnoise/)）
2. J.-M. Valin，[一种用于实时全频带语音增强的混合DSP /深度学习方法](https://jmvalin.ca/papers/rnnoise_mmsp2018.pdf)，国际多媒体信号处理研讨会，2018年。（arXiv：[1709.08243](https://arxiv.org/abs/1709.08243)）
3. Jean-Marc的[博客文章](https://jmvalin.dreamwidth.org/15210.html)以发表评论

# 致谢

特别感谢Michael Bebenita，Thomas Daede和Yury Delendik的帮助，帮助他们组织了这个Demo。感谢Reuben Morais对Keras的帮助。