UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence

Master Thesis

Unsupervised musical source separation

by

Maurice Frank

1165765

August 17, 2020

48 ECTS

November 2019 - July 2020

Supervisor:
Maximilian Ilse

Assessor:
dhr. dr. Efstratios Gavves

UNIVERSITY AMSTERDAM

Contents

Abstract

Research Question

Source separation is the task of finding a set of latent sources $\boldsymbol{s} = [\boldsymbol{s}_1, \dots \boldsymbol{s}_k, \dots, \boldsymbol{s}_N]^T \in \mathbb{R}^{N \times T}$ to an observed mix of those sources $\boldsymbol{m} \in \mathbb{R}^{1 \times T}$. It exists an implied but unknown mixing function $\boldsymbol{m} = f(\boldsymbol{s})$. The task is to find an approximate inverse model $g(\cdot)$ which retrieves $\boldsymbol{s}$:

$$\boldsymbol{m} = f(\boldsymbol{s}) \tag{1}$$

$$g(\boldsymbol{m}) \approx \boldsymbol{s} \tag{2}$$

In the case of musical source separation, the sources are different instruments [1] or their separate voices. The mix is mixed and mastered mono or stereo recording of the musical piece something is missing here. The reversal of this operation is strongly unidentifiable. As instruments can make similar sounds extracting the instruments' voices is not always decidable not very clear. Therefore any approach to this problem is approximate in expectation.

Learning this operator $g(\cdot)$ can be a relieved don't know what can be a relieved means with three sources of supervision: First, by supplying the number of sources N in a given mix. Second, by supplying the distinct source classes for each contained source. And lastly, by actually providing tuples $(\boldsymbol{m}, \boldsymbol{s})$ of associated mixes and sources. examples like in footnote 1 for all three would be good. Maybe you start with an example and then explain the three options. The most costly supervision is the last one, as there exists no large source of such data. Therefore when we refer to our method as unsupervised we refer to the relaxation from this source of supervision. This sentence is unnecessary complex

Therefore our research question summarizes to:

> Can we learn a musical source separation model $g(\cdot)$ without relying on examples of the mixing process but only using samples of unrelated sources $\{\boldsymbol{s}_k\}$ and mixes $\boldsymbol{m}$. I find that hard to understand. Can you not just say that you don't need tuples of all signals and the mix?

[1] Think of the classes being {*guitar*,*piano*,*voice*,...}

Background

In this chapter, we introduce the minimally necessary background concepts our work is building upon. Our work marries two fields of research: on the one side generative, graphical models. With those one tries to model the data distribution of an observed variable by inducing dependencies of latent (unobserved) variables generating the observed variables. Modeling those dependencies, the meaningful latent variables allow us to build an intuitive generative story for the probabilistic model of the observed data. On the other hand, we introduce the recent body of work on how to practically model sound signals, in particular how this compares differently to modeling images.

First, we introduce the concept of deep latent-variable models, which important classes of models exist and how to train them. Second, we will shortly draw up the difficulties of modeling sound and which recent solutions exist and how they differ.

Deep Latent-Variable Models

In this section, we introduce the idea of latent variable and the idea of finding models of data using those latent variables. Further, we discuss what practical ways are currently successful in estimating the model parameters and how they are trained.

For our process, we have observations from the data space $\boldsymbol{x} \in \mathcal{D}$ for which there exists an unknown data probability distribution $p^*(\mathcal{D})$. We collect a data set $\{\boldsymbol{x}_1 \dots \boldsymbol{x}_N\}$ with N samples. Further we introduce an approximate model with the density[2] $p_{\boldsymbol{\theta}}(\mathcal{D})$ and model parameters $\boldsymbol{\theta}$. Learning or modeling means finding the values for $\boldsymbol{\theta}$ which will give the closest approximation of the true underlying process:

[2] We write density and distribution interchangeably to denote a probability function.

$$p_{\boldsymbol{\theta}}(\mathcal{D}) \approx p^*(\mathcal{D}) \tag{3}$$

The model $p_{\boldsymbol{\theta}}$ has to be complex enough to be able to fit the data density while little enough parameters to be learned. Every choice for the form of the model will *induce* biases[3] about what density we can model, even before we maximize a learning objective using the parameters $\boldsymbol{\theta}$.

[3] called *inductive biases*

In the following described models we assume the sampled data points $\boldsymbol{x}$ to be drawn from $\mathcal{D}$ *independent and identically distributed*[4].

[4] meaning the sample of one datum does not depend on the other data points and we all draw them the same way

Therefore we can write the data log-likelihood as:

$$\log p_{\theta}(\mathcal{D}) = \sum_{x \in \mathcal{D}} \log p_{\theta}(x) \tag{4}$$

The maximum likelihood estimation of our model parameters maximizes this objective.

To form a latent-variable model we introduce a *latent variable*[5]. This latent variable can be any random variable underlying the generation of a data sample in $\mathcal{D}$. For example, if we look at the generative story of a sound sample, a latent variable could be the sound source (the *instrument*). Instead of modeling the distribution of sound samples generated by all possible instruments in one joint distribution we can introduce the instrument as the latent variable. This categorical variable z then implies the independent modeling process of each instrument's possible sound.

[5] Latent variables are part of the directed graphical model but not observed.

Now the data likelihood is the marginal density of the joint latent density. With one latent variable we get:

$$p_{\theta}(x) = \int p_{\theta}(x, z) dz \tag{5}$$

Typically we introduce a factorization of the joint. The most common one and also the one from our previous example is:

$$p_{\theta}(x) = \int p_{\theta}(x|z) p(z) dz \tag{6}$$

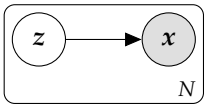


Figure 1: The graphical model with a introduced latent variable z. Observed variables are shaded.

This corresponds to the graphical model in which z is generative parent node of the observed x, see Figure 1. The density $p(z)$ is called the *prior distribution*.

If the latent is small, discrete, it might be possible to directly marginalize over it. If for example, z is a discrete random variable and the conditional $p_{\theta}(x|z)$ is a Gaussian distribution than the data model density $p_{\theta}(x)$ becomes a mixture-of-Gaussians, which we can directly estimate by maximum likelihood estimation of the data likelihood.

For more complicated models the data likelihood $p_{\theta}(x)$ as well as the model posterior $p_{\theta}(z|x)$ are intractable because of the integration over the latent z in Equation (6).

To formalize the search for an intractable posterior into a tractable optimization problem we can follow the *variational principle*[6] which introduces an approximate posterior distribution $q_{\phi}(z|x)$, also called the *inference model*. Again the choice of the model here carries inductive biases, as such that even in asymptotic expectation we can not obtain the true posterior.

[6] Jordan et al., "An Introduction to Variational Methods for Graphical Models", 1999.

Following the derivation in Kingma and Welling[7] we introduce the inference model into the data likelihood [8]:

[7] Kingma and Welling, "An Introduction to Variational Autoencoders", 2019.

[8] The first step is valid as q_{θ} is a valid density function and thus integrates to one.

$$\log p_{\theta}(\boldsymbol{x}) = \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log p_{\theta}(\boldsymbol{x})\right] \tag{7}$$

$$= \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x},z)}{p_{\theta}(z|\boldsymbol{x})}\right] \tag{8}$$

$$= \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x},z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\frac{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}{p_{\theta}(z|\boldsymbol{x})}\right] \tag{9}$$

$$= \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x},z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\right] + \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}{p_{\theta}(z|\boldsymbol{x})}\right] \tag{10}$$

$$= \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x},z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\right] + \mathbb{D}_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(z|\boldsymbol{x})\|p_{\theta}(z|\boldsymbol{x})] \tag{11}$$

Note that we separated the likelihood into two parts. The second part is the (positive) Kullback-Leibler divergence of the approximate posterior from the true intractable posterior. This unknown divergence states the 'correctness' of our approximation [9].

Therefore the first term becomes a lower bound for the likelihood of the model. Also called the *variational free energy* or *evidence lower bound* (ELBO):

$$\mathrm{ELBO}_{\theta,\boldsymbol{\phi}}(\boldsymbol{x}) = \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x},z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\right] \tag{12}$$

We can introduce the same factorization as in Equation (6):

$$\mathrm{ELBO}_{\theta,\boldsymbol{\phi}}(\boldsymbol{x}) = \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p_{\theta}(\boldsymbol{x}|z)p(z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\right] \tag{13}$$

$$= \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log \frac{p(z)}{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\right] + \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log p_{\theta}(\boldsymbol{x}|z)\right] \tag{14}$$

$$= -\mathbb{D}_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(z|\boldsymbol{x})\|p(z)] + \mathbb{E}_{q_{\theta}(z|\boldsymbol{x})}\left[\log p_{\theta}(\boldsymbol{x}|z)\right] \tag{15}$$

Under this factorization, we separated the lower bound into two parts. First, the divergence of the approximate posterior from the latent prior distribution and second the data posterior likelihood from the latent [10].

The optimization of the $\mathrm{ELBO}_{\theta,\boldsymbol{\phi}}$ allows us to jointly optimize the parameter sets $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. The gradient with respect to $\boldsymbol{\theta}$ can be estimated with an unbiased Monte Carlo estimate using data samples [11]. Though we can *not* do the same for the variational parameters $\boldsymbol{\phi}$, as the expectation of the ELBO is over the approximate posterior which depends on $\boldsymbol{\phi}$. By a change of variable of the latent variable we can make this gradient tractable, the so called *reparameterization trick*[12]. We express the $z \sim q_{\theta}$ as an random sample from a unparametrized source of entropy $\boldsymbol{\epsilon}$ and a parametrized transformation:

$$\boldsymbol{z} = f_{\eta}(\boldsymbol{\epsilon}) \tag{16}$$

For example for a Gaussian distribution we can express $z \sim \mathcal{N}(\mu,\sigma)$ as $z = \mu + \sigma \cdot \epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$ and $\eta = \{\mu,\sigma\}$.

[9] More specifically the divergence marries two errors of our approximate model. First, it gives the error of our posterior estimation from the true posterior, by definition of divergence. Second, it specifies the error of our complete model likelihood from the marginal likelihood. This is called the *tightness* of the bound.

[10] This will later be the reconstruction error. How well can we return to the data density from latent space?

[11] $\nabla_{\theta}\,\mathrm{ELBO}_{\theta,\boldsymbol{\phi}} \approxeq \nabla_{\theta} \log p_{\theta}(\boldsymbol{x},\boldsymbol{z})$

[12] Kingma and Welling, "Auto-Encoding Variational Bayes", 2014.

The VAE framework

The variational autoencoder (VAE) [12][13] presents a generative model using the above-introduced modeling of the joint distribution between an observed and latent variable. The model is split into two parts the encoder network and the decoder network. The encoder network parametrizes the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ by computing the parameters $\boldsymbol{\eta}$ of the latent distribution. The decoder maps samples $\boldsymbol{z} \sim q_{\boldsymbol{\theta}}(\boldsymbol{\eta})$ back into the data space. The optimization of the parameters of the two networks $\{\boldsymbol{\phi}, \boldsymbol{\theta}\}$ is done by maximizing the ELBO (Equation (15)).

End-to-end learning is made possible with the reparameterization trick and the algorithm looks like:

[13] Rezende et al., "Stochastic Backpropagation and Approximate Inference in Deep Generative Models", 2014.

Algorithm 1: Training's procedure for a variational autoencoder

while Training is not converged **do**
  $\boldsymbol{X} \sim \mathcal{D}$ ▷ Sample random mini-batch
  $\eta \leftarrow \text{EncoderNN}(\boldsymbol{X})$
  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$
  $\boldsymbol{z} \leftarrow f_{\boldsymbol{\eta}}(\boldsymbol{\epsilon})$
  $\boldsymbol{X}' \leftarrow \text{DecoderNN}(\boldsymbol{z})$
end while

The β-VAE[14] extends the VAE objective with an β hyperparameter in front of the KL divergence. The value β gives a constraint on the latent space, controlling the capacity of it. Adapting β gives a trade-off between the reconstruction quality of the autoencoder and the simplicity of the latent representations[14]. Using such a constraint is similar to the use of the information bottleneck[15].

[14] Higgins et al., "Beta-VAE", 2016.

[15] Burgess et al., "Understanding Disentangling in Beta-VAE", 2018.

Flow based models

The VAE and other related generative models, define the joint distribution between the latent and the observed variables. Another way of attacking the problem is a *change of variable* where we learn a function that changes the observed variable into the latent one, also known as *normalizing* flows[16]. A normalizing flow is a function $f : \mathcal{X} \to \mathcal{Z}$ that maps the input density to a fixed, prescribed density $p(\epsilon) = p(f(\boldsymbol{x}))$, in that normalizing the density [17]. With $f(\cdot)$ we can then integrate over $\boldsymbol{x}$ and $\boldsymbol{z}$:

[16] Tabak and Turner, "A family of non-parametric density estimation algorithms", 2013.

[17] The extreme of this idea is an infinitesimal, continuous-time flow with a velocity field.

$$\int_{\mathcal{Z}} p(\boldsymbol{z})d\boldsymbol{z} = \int_{\mathcal{X}} p(f(\boldsymbol{x}))\left|\frac{\partial f}{\partial \boldsymbol{x}}\right|d\boldsymbol{x} \tag{17}$$

$$= \int_{\mathcal{X}} p(\boldsymbol{x})dx = \int_{\mathcal{Z}} p(f^{-1}(\boldsymbol{z}))\left|\frac{\partial f^{-1}}{\partial \boldsymbol{z}}\right|d\boldsymbol{z} \tag{18}$$

The terms $\left|\frac{\partial f}{\partial \boldsymbol{x}}\right|$ and $\left|\frac{\partial f^{-1}}{\partial \boldsymbol{z}}\right|$ adapt the volumes under the measure. They are the determinant of the Jacobian matrix of the function f.

With this setup, we can choose a fixed distribution $p_z(\boldsymbol{z})$ and can optimize the model only in terms of the transformation into this

auxiliary variable:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{n}^{N} \log p_z(f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)) + \log \left| \frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{x}_n} \right| \tag{19}$$

For a finite normalizing flow, we consider a chain of invertible, smooth mappings making the composite f. The difficulty arises, from the condition, that we need to be able to compute the log-determinant of each flow function's Jacobian. Careful design of the flow functions is necessary to assure a cheap computation of the log-determinant while allowing for complex models to be learned.

The first deep model to successfully do so is NICE[18] which introduced coupling layers to parametrize the flow steps. In the coupling layer the input $\boldsymbol{x}$ is split in two equally sized parts $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. Then the first split is transformed by translation factors computed from the second split:

[18] Dinh, Krueger, et al., "NICE", 2015.

$$\mathbf{y}_a = \boldsymbol{x}_a + t(\boldsymbol{x}_b) \tag{20}$$

$$\mathbf{y}_b = \boldsymbol{x}_b \tag{21}$$

The translation function $t(\cdot)$ can be implemented as any complex operation. These additive coupling layers are volume-preserving and as such have their Jacobian's determinant is 1.

RealNVP[19] builds on top of NICE creating a non-volume preserving, normalizing flow. By extending the coupling layers to affine transformations:

[19] Dinh, Sohl-Dickstein, et al., "Density Estimation Using Real NVP", 2017.

$$\mathbf{y}_a = \exp\left(s(\boldsymbol{x}_b)\right) \odot \boldsymbol{x}_a + t(\boldsymbol{x}_b) \tag{22}$$

$$\mathbf{y}_b = \boldsymbol{x}_b \tag{23}$$

Again the scale $s(\cdot)$ and translation $t(\cdot)$ functions can be implemented with complex functions such as deep neural networks. As these transformations are non-volume preserving the Jacobian's determinant needs to be computed. Because of the design of the coupling layers the Jacobian reduces to:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T} = \begin{bmatrix} \mathbb{1} & 0 \\ \frac{\partial \boldsymbol{y}_a}{\partial \boldsymbol{x}_b^T} & \operatorname{diag}\left(\exp\left(s(\boldsymbol{x}_b)\right)\right) \end{bmatrix} \tag{24}$$

Which means the efficiently compute the determinant with:

$$\left| \frac{\partial f}{\partial \boldsymbol{x}} \right| = \exp\left[\sum_i s(\boldsymbol{x}_b)_i \right] \tag{25}$$

The authors provide multiple ideas on how to split an input tensor with spacial masking using repeating checkerboard patterns. Further, it is also important that the masking of the two splits is changed throughout a combination of multiple coupling layers so that the split into the transformer and transformed part is not fixed.

Glow[20] extended the RealNVP by introducing invertible 1×1-

[20] Kingma and Dhariwal, "Glow", 2018.

convolutions. Instead of having fixed masks and permutations for the computations of the affine parameters in the coupling layer, Glow learns a rotation matrix which mixes the channels. After mixing the input can always be split into the same two parts for the affine transformation. Further, the authors showed that training can be helped by initializing the last layer of each affine parameter network with zeros. This ensures that at the beginning without weight update each coupling layer behaves as an identity.

Langevin dynamics

Another topic we are gonna touch on in our research is sampling techniques, more specifically Langevin dynamics.

Let's take a step back and look at the typical procedure in a stochastic optimization[21] process. Let $\boldsymbol{\theta}$ denote a parameter vector for which we want to find a maximum-a-posteriori solution given the data $\boldsymbol{X} = \{\boldsymbol{x}_i\}_i^N$. In stochastic optimization, we update the parameters $\boldsymbol{\theta}$ with the gradient of the posterior using a subset of samples from our data set:

[21] Robbins and Monro, "A Stochastic Approximation Method", 1951.

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon_t}{2}\{(\nabla \log p(\boldsymbol{\theta}_t)) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\boldsymbol{\theta}_t)\} \tag{26}$$

With a prior distribution for the parameters $p(\boldsymbol{\theta})$. By taking the gradient of the posterior only under a subset of the data will give us a noisy estimate of the real gradient, but is computationally cheaper. The idea is, that the noise of the noisy gradients smoothes out while sampling different subsets. When using a decaying step size ϵ_t the stochastic optimization is guaranteed to converge to a local maximum. The problem with this approach is though, that we are not considering any uncertainty of the parameter updates.

To remedy this risk Welling and Teh[22] introduced the ideas of Langevin dynamics[23] into the update step to form *Stochastic Gradient Langevin Dynamics* (SGLD). The idea is that we introduce Gaussian noise in every update step of the optimization where the noise is scaled by the decreasing learning rate ϵ_t:

[22] Welling and Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics", 2011.

[23] Neal, "MCMC Using Hamiltonian Dynamics", 2012.

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon}{2}\{(\nabla \log p(\boldsymbol{\theta}_t) + \frac{N}{n}\sum_{i=1}^{N} \nabla \log p(x_i|\boldsymbol{\theta}_i))\} + \eta_t \tag{27}$$

$$\eta_t \sim \mathcal{N}(0, \epsilon) \tag{28}$$

Formally we want to take the updated proposal distribution and form a Markov chain Monte Carlo (MCMC) with Metropolis-Hastings to accept or reject the proposal. Because we are scaling the injected noise with decaying learning rate Welling & Teh[22] showed that the validation of the proposal distribution can be ignored. The rejection rate goes to zero.

Thus SGLD gives a computationally cheap method to do stochastic weight updates while capturing the uncertainty of the parameters.

Modeling audio

Lastly, we introduce the difficulties and approaches to modeling audio or sound signals.

In the physical world, a sound signal is a density change of the carrier air over time. As such it is a one-dimensional temporal signal[24]. Humans sound perception is *stereo* as we have two ears and sense the changes in pressure at two different points in space simultaneously.

[24] Sound waves in gases are purely compressive, therefore they cannot be polarized which would introduce a higher complexity. In solids a sound signal can have polarization, think of an earthquake with shear and pressure components.

Representations of audio

The simplest form of digitally representing a sound signal is recording the amplitude at fixed intervals at one or multiple points in space with a microphone. This method is called Pulse-Code modulation (PCM). It introduces two parameters which will bias the result of the record. First we have to pick a temporal frequency with which the pressure samples are taken, the so-called sample rate. The Nyquist-Shannon theorem[25] tells us that if the highest frequency in the true signal is B Hz than with a sample rate of $2 \cdot B$ Hz we will capture the complete signal with all frequencies. At a lower sample rate, we might introduce aliasing effects. Second, we have to represent each amplitude sample as a digital value. Most commonly the sample is encoded into a 8 bit or 16 bit integer. The microphone has a range of air pressure which w.l.o.g. we set to $[-1, 1]$, where a value of 0 is no signal, 1 shows maximum compression and -1 maximum decompression of the carrier air. The simplest form of encoding is Linear PCM.In Linear PCM, amplitudes are quantized on an evenly spaced grid over the possible value range. More advanced quantization include μ-law encoding[26] in which the quantization intervals are varied with the amplitude[27]. As the human perception of loudness is logarithmic this encoding also quantizes the amplitude values on a logarithmic response curve. Simply speaking the lower amplitudes will be quantized on a finer grid than higher amplitudes. This makes it possible to achieve a higher signal-to-noise ratio at smaller encoding sizes for sound signals.

[25] Kotelnikov, "On the Carrying Capacity of the Ether and Wire in Telecommunications", 1933.

[26] *Pulse Code Modulation (PCM) of Voice Frequencies*, 1972.

[27] The actual formula for calculating the μ-law quantization is:

$$\mu\text{-law}(x) = \operatorname{sgn}(x) \cdot \frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)}$$

As an example, if encoding a sound signal with 8 kHz 16 bit Linear PCM we sample an amplitude value every $\frac{1}{8000}$s and quantize those into $2^{16} = 65536$ bins. A sample with 100 values is only 12.5 ms long!

A standard full piano with 88 keys tuned to 440 Hz at the A_4[28] has a frequency range of $[27.5\,\text{Hz}, 4186\,\text{Hz}]$ from the A_0 to the C_8, respectively. Therefore just for the recording of a standard piano we need a sampling rate of more than 8 kHz. Beyond the piano, many instruments exhibit even larger frequency content which leads to the conclusion that not just for recording but also for our goal of modeling musical sounds a reasonably high sampling rate is crucial.

[28] ISO/TC 43 Acoustics, *ISO 16 – Acoustics — Standard Tuning Frequency*, 1975.

Looking at the temporal scales of musical information, on the small end we have the modeling of pitch as discussed before which

happens at scales of $[10^{-1}\,\text{ms}, 10^{2}\,\text{ms}]$ over single notes happening at scales of $[10^{2}\,\text{ms}, 10^{3}\,\text{ms}]$ to the structure of melodies and full songs happening at $[10^{4}\,\text{ms}, 10^{6}\,\text{ms}]$. The modeling of music happens over 7 magnitudes of time! In *Modeling raw audio* we will present common solutions to this difficulty.

Spectrograms

Commonly used alternative representations of signals in sound processing are spectrograms. A spectrogram is a mapping of the time-domain wave signal into a frequency domain. With a Fourier transform[29] it is possible to represent any function as an (at worst) infinite sum of harmonics, or trigonometrical signals. The Fourier transform is a complex-valued function where the absolute value gives the amount a given cosine with that frequency is present in the signal.

$$\hat{f}(\nu) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \nu} dx \tag{29}$$

Where $f(x)$ is the original time-domain signal and ν is a frequency of interest.

The idea for signal/music processing is that with a transformation into frequency space we can handcraft a response feature similar to how the hairs in the cochleae of the human inner ear react to different frequencies of the pressure wave. A Fourier transform gives the frequency content of the whole signal but we are interested in the change of frequencies present in a given signal over time. For this, the most common transformation is the Discrete Short-time Fourier transform (STFT)[30]. For the STFT the Fourier transformation is applied to overlapping windows sliding over the signal. As our signal is a discrete sample of the true signal, a discrete implementation is needed.

An unmodified Fourier transform is linear in frequency. Humans do not perceive changing frequency (pitch) in this way. First, there is an individual lower and upper bound to human perception of frequency. Second, changing frequencies are not perceived linearly in their pitch[31]. Pitch perception is sublinear[32]. To allow a spectrogram to reflect this perceived importance Mel-scales[33] are commonly applied. The Mel scale compresses the frequency range with a logarithm which parameters are determined from perceptional experiments. A common transformation is[34]:

$$\hat{\nu} = 2595 \cdot \log_{10}\left(1 + \frac{\nu}{700}\right) \tag{30}$$

When using a spectrogram for signal processing we have to choose the form of the spectrogram being used. First, most commonly one would use the magnitude of the complex spectrogram to access the actual frequency contents. By that, we are disregarding the phase of these harmonics though. Trying to model the phase separately to the frequency is futile, see Figure 2. Alternatively one

[29] Fourier, *Théorie analytique de la chaleur*, 1822.

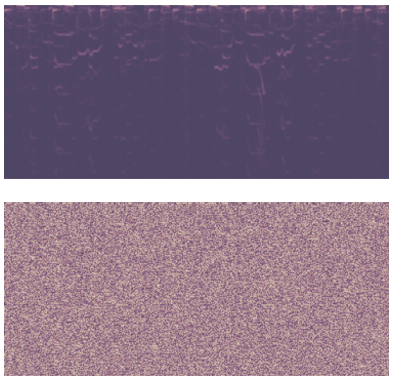


Figure 2: The STFT magnitude and the corresponding phases for each sinusoidal. The magnitude spectrogram exhibits clear structure while the corresponding phase strongly unstructured.

[30] Gröchenig, *Foundations of Time-Frequency Analysis*, 2001.

[31] Pitch is the human perception of a sound being *high* or *low*.

[32] Stevens et al., "A Scale for the Measurement of the Psychological Magnitude Pitch", 1937.

[33] Mel(ody)

[34] Douglas and Shaughnessy, "Speech Communications", 2000.

can use an approximate vocoder to reverse the destructive reduction of the complex spectrogram (e.g.[35]) or directly model the complex spectrogram which has been used for some tasks with success[36]. Many recent deep learning approaches to sound processing model a time-frequency map which is used to filter the original spectrogram[37]. As such the phase from the input signal is used in the resulting output signal. While often these approximate approaches are successful in practice the usage of magnitude spectrograms introduces undefined limits into, which might inhibit performance[38].

[35] Chandna, Blaauw, et al., "A Vocoder Based Method For Singing Voice Extraction", 2019.

[36] Le Roux et al., "Phasebook and Friends", 2019; Liu et al., "Supervised Speech Enhancement with Real Spectrum Approximation", 2019; Tan and Wang, "Complex Spectral Mapping with a Convolutional Recurrent Network for Monaural Speech Enhancement", 2019.

[37] Andreas Jansson et al., "Singing Voice Separation with Deep U-Net Convolutional Networks.", 2017; Chandna, Miron, et al., "Monoaural Audio Source Separation Using Deep Convolutional Neural Networks", 2017.

[38] Lluís et al., "End-to-End Music Source Separation", 2019.

Modeling raw audio

Deep learning models as used for image applications are unsuitable for raw audio signals (signals in *time-domain*). Digital audio is sampled at high sample rates, commonly 16kHz up to 44kHz. The features of interest lie at scales of strongly different magnitudes. Recognizing the local-structure of a signal, like frequency and timbre, might require features at short intervals ($\approx$ tens of milliseconds) but modeling of speech or music features happens at the scale of seconds to minutes. As such a generative model for this domain has to model at these different scales.

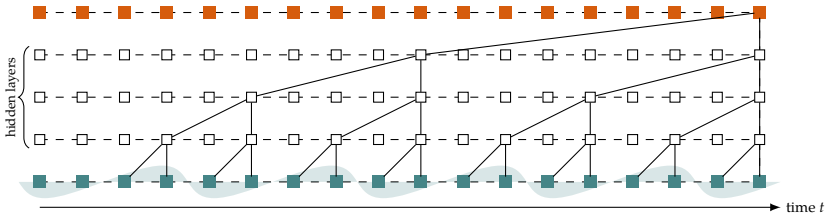


Figure 3: An example of how dilated convolutions are used in the WaveNet. We see three hidden layers with each a kernel size of two. By using the dilations the prediction of the new output element has a receptive field of 18. This convolution is *causal* as the prediction depends only on previous input values. Causality is enforced through asymmetric padding.

The **WaveNet**[39] introduced an autoregressive generative model for raw audio. It is build upon the similar PixelCNN[40] but adapted for the audio domain. The WaveNet accomplishes this by using dilated causal convolutions. Using a stack of dilated convolutions increases the receptive field of the deep features without increasing the computational complexity, see Figure 3. Further, the convolutions are gated and the output is constructed from skip connections. For the structure of a hidden layer refer to Figure 4. A gated feature, as known from the LSTM[41], computes two outputs: one put through an sigmoid $\sigma(\cdot)$ activation and one through an $\tanh(\cdot)$ activation. The idea being that the sigmoid (with an output range of $[0, 1]$) regulates the amount of information, thereby gating it, while the tanh (with a range of $[-1, 1]$) gives the magnitude of the feature. The output of the WaveNet is the sum of outflowing skip connections added after each (gated) hidden convolution. This helps fusing information from multiple time-scales (*low-level* to *high-level*) and makes training easier[42]. The original authors tested the model on multiple audio generation tasks. They used a μ-law encoding[43] which discretizes the range $[-1, 1]$ to allow a set of μtargets and a multi-class cross-entropy training objective. While being quite unnatural this is

[39] Van den Oord, Dieleman, et al., "WaveNet", 2016.

[40] Van den Oord, Kalchbrenner, et al., "Conditional Image Generation with PixelCNN Decoders", 2016,

[41] Hochreiter and Schmidhuber, "Long Short-Term Memory", 1997.

[42] Szegedy et al., "Going Deeper with Convolutions", 2014.

[43] *Pulse Code Modulation (PCM) of Voice Frequencies*, 1972.

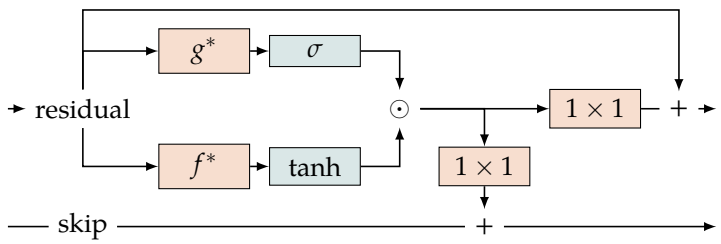


Figure 4: A hidden layer as in the WaveNet. We have the four convolutions, with the filter f^* and gate g^* being dilated and the two-channel mixing filters. Information flows along the skip connection that gets added up and the residual which receives the original input.

done to avoid making any assumptions about the target distribution. Sound generation with a WaveNet is slow as the autoregressiveness requires the generation value by value. The original WaveNet setting is generating waveforms, by giving the previously generated values as the input and conditioning the process on target classes, $p(x_t|x_{1:t}, c_t)$. Therefore the generation has to happen value-by-value (autoregressive). This can be alleviated by keeping intermediate hidden activations cached[44]. The WaveNet can be conditioned by adding the weighted conditionals in the gate and feature activations of the gated convolutions[45].

One of the first latent generative models to employ a WaveNet is NSynth[46]. They construct an autoencoder in which both encoder and decoder are WaveNet-like. The *non-causal temporal encoder* uses a stack of dilated residual non-causal convolutions. The convolutions are not gated and no skip-connections are used. The decoder is a WaveNet taking the original input chunk as an input and predicts the next value of the sound sample, while being conditioned on the latent variable. The authors use this model to learn latent temporal codes from a new large set of notes played by natural and synthesized instruments. The latent of the VAE is conditioned on the pitch of these notes. While the model is difficult to train, they show great improvement of the WaveNet-based VAE compared to a spectral-based autoencoder.

Chorowski et al.[47] presents a WaveNet-based VAE.They are learning speech representations, unsupervised. The encoder is a residual convolutional network and takes a Mel-scaled spectrogram of the signal as its input. As the bottleneck they found a VQ codebook[48] to be most successful. The decoder is an autoregressive WaveNet conditioned on the latent features. While the input to the encoder is a spectrogram the decoder reconstructs the waveform of the signal.

Multiple works have combined the WaveNet architecture with the *Flow based models*. In WaveGlow[49] the authors parametrize the affine coupling layers of a Glow model with WaveNets and keeping the rest of the architecture similar to the original Glow with the 1×1-convolutions. The authors apply this model to the task of speech synthesis where Mel-spectrograms are generated from the true audio. Then all WaveNets in the coupling layers are conditioned on the spectrogram. The model therefore learns the waveform generation task given spectral features (a vocoder). WaveGlow is outperforming similar approaches and the needlessness of autoregressiveness

[44] Paine et al., "Fast Wavenet Generation Algorithm", 2016.

[45] Van den Oord, Kalchbrenner, et al., "Conditional Image Generation with PixelCNN Decoders", 2016.

[46] Kalchbrenner et al., "Efficient Neural Audio Synthesis", 2018.

[47] Chorowski et al., "Unsupervised Speech Representation Learning Using WaveNet Autoencoders", 2019.

[48] Van den Oord, Vinyals, et al., "Neural Discrete Representation Learning", 2017.

[49] Prenger et al., "WaveGlow", 2018.

gives the model fast performance.

The FloWaveNet Kim et al.[50] is a similar model building a flow from WaveNets. As in WaveGlow, the affine coupling layers are parametrized by WaveNets. The model is more similar to the RealNVP in that the flexibility of the flows is achieved by flipping the channels of the data after a flow operation instead of learning a channel mixing convolution. Each coupling flow layer is preceded by an Activation Normalizing operation as in Glow.

[50] Kim et al., "FloWaveNet", 2019.

Related work

In this chapter, we give an overview of recent approaches to sound source separation which are related to our approach either in the chosen model choices or implicit ideas. We will introduce source separation models based on deep models in time and frequency domain. As the body of research into this task, especially in practical application, is vast and diverse, this overview can only be a small insight into tangentially related and recent research work. Many practical approaches to musical source separation work with handcrafted features and algorithms informed by musicological and perceptional research. To the opposite most of the methods presented here assume very little about the behaviour of the sound sources, enabling an end-to-end learning from ground data.

Source separation

All here presented model maximize $p(\boldsymbol{s}|\boldsymbol{m})$ for one source (e.g. extracting only the singing voice out of a mix) or $p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N|\boldsymbol{m})$ for multiple sources. Note that in the second case the conditional likelihood is not factorized, a shared model for all sources.

As such these methods employ a reconstruction loss that boils down to:

$$\arg\min_{\boldsymbol{\theta}} \sum_{k}^{N} \|\boldsymbol{s}_k - f_{\boldsymbol{\theta}}(\boldsymbol{m})_k\|^l \tag{31}$$

$$l \in \{1, 2\} \tag{32}$$

Where the parameters $\boldsymbol{\theta}$ of the model f are optimized with either the L1 or L2 loss to the true signals $\{\boldsymbol{s}_k\}$.

Deep clustering

One explored approach to musical source separation is called Deep clustering introduced by Hershey et al.[51]. In their work the authors train two BiLSTM modules to predict an embedding vector per time-frequency bin of the spectrogram. Then a clustering step is applied to these embeddings, clustering them into as many clusters as sources. This results in a mapping of the spectrogram bins to the different sources. The signals are reconstructed by applying the

[51] Hershey et al., "Deep Clustering", 2015.

maps to the original input spectrogram which then is reversed. The authors use k-means clustering. Isik et al.[52] changes that by learning the clustering step through a deep neural network and adding some additional architectural changes and L. Li and Kameoka[53] improves the model further by replacing the LSTM modules with Gated Linear Units[54].

Huang et al.[55] builds a recurrent deep neural network for source separation on spectrograms. The input to the network is a magnitude spectrogram and the network predicts a frequency filter map which together with the phase information from the spectrogram of the mix is used to recover the sources. The model is used for singing voice separation with an L2 loss on both the separated voice and music.

U-Net based

Jansson et al.[56] were the first to use an U-Net architecture (see Figure 5) for musical source separation. They use a convolutional U-Net that takes magnitude spectrograms as the input and predicts a filter map of the singing voice in frequency space. The voice signal is reconstructed by multiplying the input magnitude with the predicted mask and using the original phase information from the mix, unaltered. The training objective is the L_1 loss between the masked input spectrogram and the target signal.

The Wave-U-Net[57] brings this architecture into the time-domain. The downstreaming and upstreaming layers are replaced with 1D convolutional layers. The upstreaming convolutions are not implemented as deconvolutional layers[58] but by upsampling the features with a bicubic kernel and then applying a normal convolution. Further here the model is predicting multiple source signals instead of only extracting the voice. In the singing voice separation task the model achieves similar results compared to the above-introduced spectrogram based U-Net.

Slizovskaia et al.[59] extends the Wave-U-Net by conditioning the filter activations at the bottleneck with the instrument class label. The additional information is improving the separation quality.

Cohen-Hadria et al.[60] show improvements of training a Wave-U-Net when using additional data augmentation. They apply a hand-crafted set of auditory transformations (pitch-shifting, time-stretching, transforming the spectral envelope of the singing voice) to the training set of the musdb18 dataset.

The Hydranet[61] adds an bidirectional LSTM at the bottleneck of the Wave-U-Net. The BiLSTM is capable of select and keep information at the bottleneck over time and combine this memory with the new encoded information. This makes it possible for the network to keep information about the source signal at scales larger than the actual receptive field of the U-Net. The research shows a significant improvement over previous U-Net based approaches.

Narayanaswamy et al.[62] adapts the Wave-U-Net by exchanging

[52] Isik et al., "Single-Channel Multi-Speaker Separation Using Deep Clustering", 2016.

[53] L. Li and Kameoka, "Deep Clustering with Gated Convolutional Networks", 2018.

[54] Dauphin et al., "Language Modeling with Gated Convolutional Networks", 2017.

[55] Huang et al., "Singing-Voice Separation from Monaural Recordings Using Deep Recurrent Neural Networks", 2014.

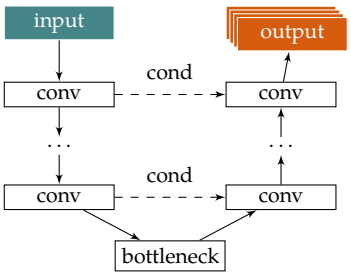


Figure 5: U-Net: the input gets projected into a bottleneck through some form of convolutional architecture. A set of upsampling or transposed convolutional layers predict the output from the bottleneck features. The upstream layers are conditioned on the activations of the according downstream features..

[56] Jansson et al., "Singing Voice Separation with Deep U-Net Convolutional Networks", 2017.

[57] Stoller et al., "Wave-U-Net", 2018.

[58] Dumoulin and Visin, "A Guide to Convolution Arithmetic for Deep Learning", 2018.

[59] Slizovskaia et al., "End-to-End Sound Source Separation Conditioned On Instrument Labels", 2019.

[60] Cohen-Hadria et al., "Improving Singing Voice Separation Using Deep U-Net and Wave-U-Net with Data Augmentation", 2019.

[61] Kaspersen, "HydraNet", 2019.

[62] Narayanaswamy et al., "Audio Source Separation via Multi-Scale Learning with Dilated Dense U-Nets", 2019.

the 1D convolutional layers with dilated convolutions as in the WaveNet. By doing so the context of the predictions can be increased. They also propose a modified architecture with in which the up- and downstream feature blocks become residuals, making optimization easier. Compared to the original Wave-U-Net they show a significant increase in performance.

Demucs[63] also introduces a BiLSTM at the bottleneck[64]. Additionally to the ideas from the HydraNet Demucs also adds data augmentation and makes additional internal changes to the network architecture. They simplify the layer blocks by using the simpler Gated Linear Units[65] and while the Wave-U-Net is using upsampling followed by a convolutional layer in the decoder, here the others directly use transposed convolutions to achieve the upsampling. Demucs is the first time-domain based end-to-end model that is achieving similar or better results compared to spectrogram based models.

[63] Défossez, Usunier, et al., "Demucs", 2019.

[64] Défossez, Zeghidour, et al., "SING", 2018.

[65] Dauphin et al., "Language Modeling with Gated Convolutional Networks", 2017.

WaveNet based

Rethage et al.[66] uses a WaveNet for speech denoising. Speech denoising is a special case of source separation as the observed mix is separated into true signal and noise. The authors make the WaveNet non-causal by making the padding symmetrical. Further, they used L_1-loss, instead of the multi-class μ-law objective. They show that for their case the real-valued predictions behave better.

In the setting of source separation or de-noising, the WaveNet can be non-causal because we are modeling a one-to-one translation from mixed or noised signal to the true source. Therefore the WaveNet is no longer autoregressive but also the receptive field of a prediction contains inputs from the past and future. For some practical use-cases that might be disadvantageous.

Lluís et al.[67] adapts a time-domain WaveNet for musical source separation. The non-causal WaveNet directly outputs the separated sources and is trained fully supervised. They show this simple setup performing well against spectrogram based baselines. Also, the experiments show a higher performance with a network that is deeper but less wide [68].

[66] Rethage et al., "A Wavenet for Speech Denoising", 2018.

[67] Lluís et al., "End-to-End Music Source Separation", 2019.

[68] Deepness of a network refers to the number of hidden layers. Wideness refers to the number of kernels/features of each of these hidden layers. Making the WaveNet deeper significantly increases its receptive field.

Auto-encoder based

Grais et al.[69] present the first work using an auto-encoder for multi-channel sound source separation. Their auto-encoder takes raw audio as the input is using multi-scale convolutional layers (transposed in the decoder) in the encoder and decoder, combining the activations of the different scales before the activation. (Similar to Inception networks[70]). While the network does generate multi-scaled features it only takes in a very shore context frame from the mixed signal. The many artifacts in the results might be explained by that.

The SEGAN[71] proposes a Generative Adversarial network[72] for

[69] Grais et al., "Raw Multi-Channel Audio Source Separation Using Multi-Resolution Convolutional Auto-Encoders", 2018.

[70] Szegedy et al., "Going Deeper with Convolutions", 2014.

[71] Pascual et al., "SEGAN", 2017.

[72] Goodfellow et al., "Generative Adversarial Networks", 2014.

Speech Enhancement. The network works non-causal and directly on the waveform. The Generator is has an auto-encoder structure. The input, the noisy speech, gets reduced to a smaller representation by a convolutional encoder. At the bottleneck, a noise sample is added and a convolutional decoder outputs the denoised frame. The Discriminator is optimized to distinguish noiseless from noised samples. Training is done with the standard mini-max objective.

The TasNet[73] proposes another distinct approach to source separation in the time-domain. The mixed signal is decomposed into a set of basis signals and weights. An LSTM predicts the weights for each source channel and basis signal in each frame. The original source signals can then be reconstructed by summing up the weighted basis signals for each source. The model is shown to work well for the task of voice separation but is specifically build with conceptual tradeoffs to increase computationally inexpensive.

[73] Luo and Mesgarani, "TasNet", 2018.

Method

In this chapter, we introduce the theoretical idea of our probability modelling of musical source separation. First, we explicitly state our chosen model of the problem, next derive a suitable optimization objective from it and then explain how we can optimize towards that.

We propose the graphical model as shown in Figure 6 as the true generative model of music tracks being generated from separate source tracks. For each source, a sample is taken from the latent source distribution. The observed mix is generated deterministically from the full set of sources. Without loss of generality, we fix this function to be the mean.

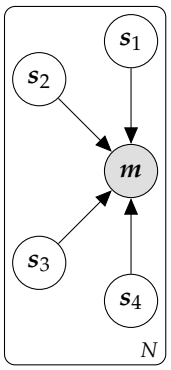


Figure 6: The used graphical model for the source separation task. We have the latent source channel variables s_k. Exemplary here, as in our data, we have four sources. The mix m is observed.

Our stated task in *Research Question* is to retreive the sources $\{s_1, \ldots, s_N\}$ from a given mix m. Our model is trained without using sample tuples

$$(m, \{s_1, \ldots, s_N\}) : f(\{s_1, \ldots, s_N\}) = m$$

which would demonstrate the relation between separate sources and mixes. The general idea is visualized in Figure 7.

The graphical model in Figure 6 implies the following factorization:

$$p(m) = \int^N p(s_1, \ldots, s_N, m)\, d^N s \tag{33}$$

$$= \int^N p(m|s_1, \ldots, s_N) \cdot p(s_1, \ldots, s_N)\, d^N s \tag{34}$$

$$= \int^N p(m|s_1, \ldots, s_N) \cdot p(s_1) \cdots p(s_N)\, d^N s \tag{35}$$

While the conditional $p(m|s_1, \ldots, s_N)$ is only a Dirac delta distribution, as the mix is generated deterministically with the mean of the sources, the model posterior $p(s_1, \ldots, s_N|m)$ is intractable and precisely what we are interested in. Again we want to make it clear that this setup changes the typical optimization target, as seen in other source separation approaches. We factorize the distribution $p(m)$ of mixed songs, only then to extract the most likely *latent* components that generate this mix, the sources. Therefore we are explicitly modelling the generative process of the mixed songs, and only implicitly the separation task.

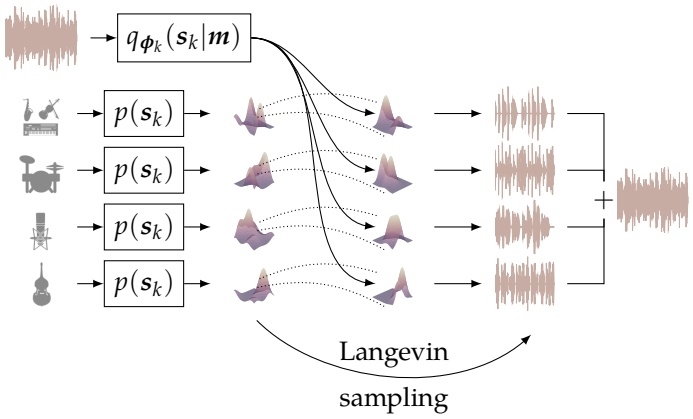


Figure 7: Our approach visualized. From each data source, we build a prior distribution. From there we either train a respective set of posteriors that are informed by the signal mix and that we can sample a set of separation solutions. Or we use Langevin dynamics to sample from the posteriors directly. Either approach happens under the constraint that the samples can construct the mix.

Already with the graphical model we introduce a fairly big assumption, namely that we can model the source distributions independently from each other when modeling the joint:

$$p(\boldsymbol{m}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) \equiv p(\boldsymbol{m}|\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) \cdot \prod_k^N p(\boldsymbol{s}_k) \tag{36}$$

Intuitively this assumption does not, in general, hold for the musical domain. We can expect a different behaviour for a *guitar* in a song involving a second source like a *percussion set*. The joint of those two models is different than their independent densities [74]. Nevertheless, this assumption is crucial to be able to model the prior instrumental models without needing the tuples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ of co-occurring stems.

[74] A practical counter-example: If learning the behaviour of drums, only from solo drum recordings, one will encounter complex rhythms and sounds. In many rock genres drums often exhibit simpler sounds, providing beat and tempo. These two distributions are not the same.

The general process is as follows:

First, because we assumed in our graphical model the latent sources to be independent, there exists a probability distribution $p(\boldsymbol{s}_k)$ for each source k. Thereout we need to choose a model which gives the parametrized approximated prior $p_{\boldsymbol{\theta}}(\boldsymbol{s}_k)$ with the parameters $\boldsymbol{\theta}$ which we optimize with the samples from $\mathcal{D}_k$.

Second, for each source, there exists a posterior given a mix $p(\boldsymbol{s}_k|\boldsymbol{m})$ from which we want to draw samples. Here two approaches are possible:

Drawing samples from the posterior

The two fundamental approaches to drawing samples from the posterior are to either model an approximate posterior distribution and draw samples from that or to sample from the posterior without ever modeling it.

In the first case, we propose an approximate posterior $q_{\phi_k}(\boldsymbol{s}_k|\boldsymbol{m})$, which parameters are optimized to minimize the divergence from the previously trained and fixed corresponding prior. This is the setting similar to a VAE.After training the approximate posteriors, the prior distributions are not needed anymore. The inference of a separation is one forward pass through the encoder networks

given a mix. From the resulting posterior we can sample separated sources.

In the second case, we propose using Stochastic Gradient Langevin Dynamics (SGLD)[75] to sample from the posterior. Here we never explicitly approximate the posterior distribution but by iteratively optimizing the samples under the pre-trained priors we can sample from the true posterior. In this case, each separation requires a MCMC sampling pass using the prior models, which is parameterless.

[75] Welling and Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics", 2011.

Both optimization, either training or sampling, happen under the mixing constraint:

$$\sum_{k=1}^{N} a_k \cdot \boldsymbol{s}_k = \boldsymbol{m} \tag{37}$$

Building blocks

Thinking in the common terms of the VAE we need to choose:

1. a parametrization $p_{\boldsymbol{\theta}}(\boldsymbol{s}_k)$ of the *prior* distribution
2. a parametrized approximate posterior distribution $q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})$ with parameters $\boldsymbol{\eta}$
3. a parametrized *encoder* which gives the parameters for the posterior distribution $\texttt{Encoder}_{\boldsymbol{\phi}}(\boldsymbol{m}) = \boldsymbol{\eta}$
4. a *decoder* which returns the input $\boldsymbol{m}$ from the latents $\texttt{Decoder}_{\boldsymbol{a}}(\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N\}) = \boldsymbol{a}^T \cdot \left[\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N\right]^T = \boldsymbol{m}$

The modeling of the decoder is implied because of the linear mixing assumption in Equation (37). The parameters of the decoder $\boldsymbol{a} = \left[a_1, \ldots, a_N\right]$ are the N mixing coefficients.

For the VAE-like approach, we have to construct the prior, approximate posterior and encoder, while for the sampling approach we only need to construct the prior models. Both methods happen under the mixing constraint.

To see the relationship between the prior models and the posterior models, we can look at the divergence between the the approximate posterior and the prior model that we will be using to train the VAE model:

$$\mathbb{D}_{\mathrm{KL}}[q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m}) \| p(\boldsymbol{s}_k)] = \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{s}_k)}{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\right] \tag{38}$$

The posterior is trained to minimize the divergence from the prior but informed by mixes.

Modeling the priors

The first step in the training procedure is the training of the independent source priors $p(\boldsymbol{s}_k)$. We have to choose a model for estimating

the density that results in smooth, tight densities but also is capable enough to capture the complex data space of audio data. We choose to estimate the priors, with flow models for which we gave an overview in *Flow based models*. The models estimate densities over the separate data sources. For both the sampling-based approach and the variational approach we need these densities to give us likelihoods of samples in that source's distribution.

We propose to adapt the FloWaveNet[76] architecture, as introduced in *Flow based models*. In this architecture, the data splitting for the coupling layers is done along the channel dimension. As we are using mono waveforms, the squeeze operation is necessary to increase the channel depth from the one input channel. The next section will introduce the architecture more thoughtfully.

[76] Kim et al., "FloWaveNet", 2019.

We make it clear that the success of this method depends strongly on the expressiveness and smoothness of the learned prior models. In the case of learning the posterior, variational learning will fit the posterior distribution under the prior distribution. If the prior does not contain enough variation of the actual data distribution, the de-mixing model cannot fit a conditional posterior similar to the sample in the mixed song. In the case of the sampling approach, this constraint is even more stressed. The mixed is generated by iteratively sampling from the set of prior distributions, any possible source sample, therefore, has to be found on the manifold of the prior. If any of the priors does not capture the full variations of the data distributions, neither method will be able to extract the correct separated sources.

More practically we again point out the high difficulty of modelling audio data. As laid out in *Representations of audio* audio consists of information at greatly different resolutions. A generative model of an instrument has to model the low-range components, e.g. pitch, timbre, tremolo and high-range components, e.g. notes and rhythm.

Prior architecture

See the visualization of the prior model's architecture in Figure 8. We construct normalizing flow models, following the description in Kim et al.[77].

[77] Kim et al., "FloWaveNet", 2019.

The flow network, at the highest level, consists of n_b context blocks, each of whom contain one squeeze operation and n_f flows. The squeeze operation halves the length of the signal by doubling the number of channels. By doing so, we effectively double the receptive field of the WaveNet operators in the affine couplings. One flow contains one affine coupling layer after which the masking for the coupling (*even*/*odd*) is inverted. Before the coupling layer, we apply activation normalization[78]. The affine coupling is applying an affine transformation on the *odd* set of the input, with scale $\boldsymbol{s}$ and translation $\boldsymbol{t}$ coming from the *even* input put through a *non-causal* WaveNet.

[78] Kingma and Dhariwal, "Glow", 2018.

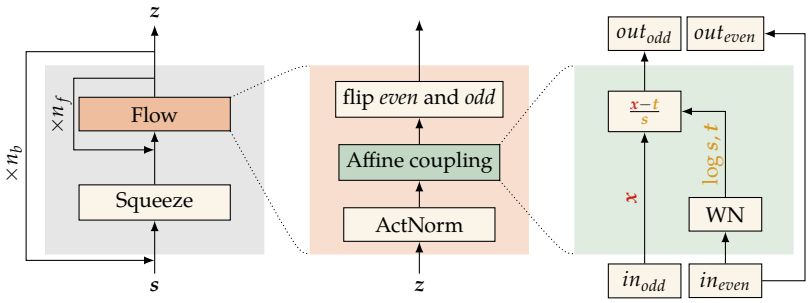


Figure 8: The building blocks for the prior model. The model consists of n_b blocks (left). Each block consists of n_f flows (middle). In each flow we apply activation normalization, followed by the affine coupling (right), after which the binary mask for the even/odd mapping is inverted. The affine coupling layer uses a WaveNet with the *even* set as the input to output scaling $\log s$ and translation t which which the *odd* set is transformed.

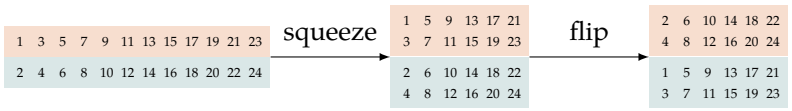


Figure 9: Flip and squeeze operation for the coupling layers. The squeeze operation doubles the channels and halves the length while the flip operation flips the split of channel dimension into odd and even.

The *even*/*odd* masking for the coupling layer is simply achieved by splitting the channel dimension into two equal parts. Flipping the masking exchanges the top half of channels with the bottom half. No channel transformation of the channels is learned as in Glow. Further observe that through the repeated squeeze operation the masking implicitly includes a checkerboard pattern on the temporal dimension, similar to the explicit pattern used in RealNVP.

As explained in *Flow based models* the invertible flow $f(\cdot)$ network directly maps the data space into a prior distribution $p(\boldsymbol{z})$. Through the invertibility we can directly optimize the log probability with the change of variable formula:

$$\log p(\boldsymbol{x}) = \log p(f(\boldsymbol{x})) + \log \det \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \tag{39}$$

Our network consists of n_b blocks with each n_f flows each with one activation normalization layer f_{AN} and an affine coupling layer f_{AC}. Therefore the log-determinant becomes:

$$\log \det \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) = \sum_{i}^{n_b \cdot n_f} \log \det \frac{\partial f_{\text{AN}}(\boldsymbol{x})}{\partial \boldsymbol{x}} + \log \det \frac{\partial f_{\text{AC}}(\boldsymbol{x})}{\partial \boldsymbol{x}} \tag{40}$$

The activation normalization layer f_{AN} learns an affine transformation per channel i:

$$f^i_{\text{AN}}(\boldsymbol{x}_i) = s_i \cdot \boldsymbol{x}_i + t_i \tag{41}$$

The inverse is: $f^{-1}_{\text{AN}}(\boldsymbol{y}_i) = \frac{\boldsymbol{y}_i - t_i}{s_i}$

The log-determinant is easily computed with[78]:

$$\log \det \frac{\partial f_{\text{AN}}(\boldsymbol{x})}{\partial \boldsymbol{x}} = T \cdot \sum_{i}^{C} \log |s_i| \tag{42}$$

where C is the number of channels and T is the length of the signal.

For the affine coupling layer, the log-determinant is made tractable by only transforming half the elements in each application. The coupling transforms the *odd* part as follows:

$$\log \boldsymbol{s}, \boldsymbol{t} = \mathrm{WN}(in_{odd}) \tag{43}$$

$$out_{odd} = f_{\mathrm{AC}}(in_{odd}) = \frac{in_{odd} - \boldsymbol{t}}{\boldsymbol{s}} \tag{44}$$

The inverse is:
$f_{\mathrm{AC}}^{-1}(out_{odd}) = out_{odd} \cdot \boldsymbol{s} + \boldsymbol{t}$

where $\mathrm{WN}(\cdot)$ is a WaveNet. The Jacobian of the coupling function is a lower triangular matrix[79] because of which the log-determinant becomes the product of the diagonal elements:

[79] Dinh, Sohl-Dickstein, et al., "Density Estimation Using Real NVP", 2017.

$$\log \det \frac{\partial f_{\mathrm{AC}}(\boldsymbol{x})}{\partial \boldsymbol{x}} = -\sum \boldsymbol{s} \tag{45}$$

The purpose of the invertible flow is it to project the complex data distribution into a tractable defined prior distribution. In most cases, as in this, the prior $p(\boldsymbol{z})$ is set to be a factorized Gaussian distribution with zero mean and variance of one. As such the likelihood of $f(\boldsymbol{x}) \sim p(\boldsymbol{z})$ can be evaluated with:

$$\log p(f(\boldsymbol{x})) = -\frac{1}{2}(\log(2 \cdot \pi)) + f(\boldsymbol{x})^2 \tag{46}$$

To train the model we maximize the likelihood in Equation (39) which we is the sum of the parts in Equations (42), (45) and (46). Optimization is done in mini-batches with the Adam optimzer[80] and a decaying learning rate scheme.

[80] Kingma and Ba, "Adam", 2017.

It is noted that the likelihood under the prior is computed "pixel"-wise[81]. While for the optimization the actual loss on which the optimizers performs update steps is the mean likelihood, we can use the trained prior flow to assign a likelihood map to an input signal where each log-likelihood value is depending on this values receptive field.

[81] time-points in our sound case

WaveNet

In the affine coupling layer we utilize a *non-causal* WaveNet for generating the scale $\log \boldsymbol{s}$ and translation $\boldsymbol{t}$ values. The WaveNet is non-causal because the filters are centred, as such the receptive field of an output value is symmetrically in the past and future input space. The kernel size is fixed to 3. The WaveNet is using the original gated convolution scheme described in *Modeling raw audio*. Appended to the last skip connection output are two additional one-dimensional convolutional layers, each preceded by a ReLU. The last convolutional layer is initialized to zero. By that we initialize the affine transformation of the coupling layer to the identity at the beginning of training, helping to stabilize optimization. The last layer outputs both scaling and translation factors. Learning those factors with weight sharing does not change the formulation of the coupling layer, as the log-determinant does not depend on the network. Learning them in the same network might increase efficiency, as well as, increasing implementation simplicity. The WaveNet completely consists of convolutional operations. As such it can be applied to one-dimensional signals of variable length.

Sampling from the posteriors

After having estimated the N prior distributions the first possibility for retrieving sample estimates $\boldsymbol{s}_k$ is sampling from the posterior using Langevin dynamics[82]. With Stochastic Gradient Langevin Dynamics (SGLD) we can sample from $\boldsymbol{s}_i \sim p(\cdot|\boldsymbol{m})$ without computing, explicitly, the posterior $p(\boldsymbol{s}_i|\boldsymbol{m})$. For a overview of SGLD see *Langevin dynamics*.

[82] Welling and Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics", 2011.

Starting with the update step in SGLD we reformulate the update with our problem constraint:

$$\boldsymbol{s}_k^{(t+1)} = \boldsymbol{s}_k^{(t+1)} + \eta \cdot \nabla_{\boldsymbol{s}_k} \log p(\boldsymbol{s}_k^{(t)}|\boldsymbol{m}) + 2\sqrt{\eta}\epsilon_t \tag{47}$$

$$= \boldsymbol{s}_k^{(t+1)} + \eta \cdot \nabla_{\boldsymbol{s}_k} \left[\log p(\boldsymbol{m}|\boldsymbol{s}_k) + \log p(\boldsymbol{s}_k) - \log p(\boldsymbol{m})\right] + 2\sqrt{\eta}\epsilon_t \tag{48}$$

$\log p(\boldsymbol{m})$ is independent from $\boldsymbol{s}_k$ therefore no gradient.

$$= \boldsymbol{s}_k^{(t+1)} + \eta \cdot \nabla_{\boldsymbol{s}_k} \left[\log p(\boldsymbol{m}|\boldsymbol{s}_k) + \log p(\boldsymbol{s}_k)\right] + 2\sqrt{\eta}\epsilon_t \tag{49}$$

$$= \boldsymbol{s}_k^{(t+1)} + \eta \cdot \left[\|\boldsymbol{m} - \frac{1}{N}\sum_j^N \boldsymbol{s}_j^{(t)}\| + \nabla_{\boldsymbol{s}_k} \log p(\boldsymbol{s}_k)\right] + 2\sqrt{\eta}\epsilon_t \tag{50}$$

The learning rate η is vanishing over time thereby collapsing the added noise $\epsilon_t \sim \mathcal{N}(0, \mathbb{1})$.

Note that the final formulation of the update step is simply a constrained greedy hill-climb under the prior model with added Gaussian noise. Intuitively the artificially noised update makes it harder for the greedy optimization to get stuck in local minima of the prior surface.

We give the pseudo-code of the optimization:

Algorithm 2: The Langevin sampling procedure for source separation is fairly straight forward. For a fixed number of steps T we sample we take a step into the direction of the gradient under the priors and the gradient of the mixing constraint while adding Gaussian noise ϵ_t.

for $t = 1 \dots T$ **do**
  for $k = 1 \dots N$ **do**
    $\epsilon_t \sim \mathcal{N}(0, \mathbb{1})$
    $\Delta \boldsymbol{s}_k^t \leftarrow \boldsymbol{s}^t + \eta \cdot \nabla \log p(\boldsymbol{s}^t) + 2\sqrt{\eta}\epsilon_t$
  end for
  for $k = 1 \dots N$ **do**
    $\boldsymbol{s}_k^{t+1} \leftarrow \Delta \boldsymbol{s}_k^t - \frac{\eta}{\sigma^2} \cdot [\boldsymbol{m} - \frac{1}{N}\sum_i^N \boldsymbol{s}_i^t]$
  end for
end for

The idea of using SGLD in combination with deep parametrized priors for source separation was, concurrently to this work, introduced in Jayaram and Thickstun[83]. In their work the authors learn flows on different image classes, at a small resolution. The mixes are generated by transparently overlaying two images. For their setup the Langevin separation is achieving better to a non-Bayesian baseline.

[83] Jayaram and Thickstun, "Source Separation with Deep Generative Priors", 2020.

Empirically they show that the gradient under the priors are proportional to the choosen noise level σ:

$$\mathbb{E}\left[\|\nabla_s \log p_\sigma(\boldsymbol{s})\|^2\right] \propto 1/\sigma^2$$

They argue that this surprising proportionality is stemming from the severe non-smoothness of the prior model. If the prior model, in the extreme case, exhibits a Dirac delta peak as the probability mass, then the estimation of the gradient with added Gaussian noise will it-self be proportional to the Gaussian. From there the authors model noise-conditioned prior distribution at the same noise levels that are later used in the sampling. The noise-conditioning strongly improves their results. We will examine the influence of noise to our models too.

Modeling the posteriors

Instead of formulating a sampling regime for approaching $\boldsymbol{s}_k$ we can also use the prior models to variationally train a model to estimate the parameters of an approximate posterior $q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})$. While estimating the true posterior $p(\boldsymbol{s}_k|\boldsymbol{m})$ is intractable, we choose a certain parametrized distribution as an approximation of the posterior and optimize the parameters to align with the true one.

We follow the same steps as previously shown generally for the latent variable models. First, we introduce an approximate posterior $q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})$ for each source channel. Next, we express the mix density as the expectation over those posteriors:

$$\log p(\boldsymbol{m}) = \mathbb{E}^N_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log p(\boldsymbol{m})\right] \tag{51}$$

From here we introduce the approximate posteriors in the expectation as before, just now for N separate priors:

$$\mathbb{E}^N_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log p(\boldsymbol{m})\right] = \mathbb{E}^N_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{m}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)}{p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N|\boldsymbol{m})}\right] \tag{52}$$

$$= \mathbb{E}^N_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{m}|\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) \cdot \prod_k^N p(\boldsymbol{s}_k)}{\prod_k^N q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})} + \log \frac{\prod_k^N q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}{p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N|\boldsymbol{m})}\right] \tag{53}$$

Note that again we use the assumption in Equation (36) to factorize the joint. This assumption is not being used for the independent approximate posteriors. While the true posterior certainly is the posterior of the joint source model, we choose our approximate posteriors to be independent. The expectation in (51) over those is still correct. The error arising from the independence assumption is captured in the thightness of the ELBO.

We arrive at the ELBO by leaving out the divergence of the approximate posterior to the true posterior:

$$\mathbb{E}^N_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log p(\boldsymbol{m})\right] \geq \sum_k^N \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{s}_k)}{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\right] + \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[p(\boldsymbol{m}|\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)\right] \tag{54}$$

The lower bound, as in the normal VAE, will be our optimization target for training the inference models that give the variational parameters $\boldsymbol{\phi}_k$. We can also formulate the objective for just one source, as the Encoders are trained independently[84]. Thus we come to:

[84] While the optimization of the Encoders is independent, the training is, of course, concurrent as the mixing condition depends on all N source samples.

$$\mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log p(\boldsymbol{m})\right] \geq \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{s}_k)}{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\right] + \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[p(\boldsymbol{m}|\boldsymbol{s}_1,\ldots,\boldsymbol{s}_N)\right] \tag{55}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\left[\log \frac{p(\boldsymbol{s}_k)}{q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})}\right] + \|\boldsymbol{m} - \frac{1}{N}\sum_{j}^{N}\boldsymbol{s}_j\| \tag{56}$$

The Kullback-Leibler divergence term is computationally expensive, therefore the divergence is the stochastic estimate using just one mini-batch.

As both the prior $p(\boldsymbol{s}_k)$ and the approximate posterior $q_{\boldsymbol{\phi}_k}(\boldsymbol{s}_k|\boldsymbol{m})$ are parametrized by deep neural networks, the gradient of Equation (56) is tractable with respect to the mini-batch.

Choosing a posterior

With the optimization regime set up, we have to choose a fitting posterior distribution and the deep neural network which computes its parameters from data. In many typical VAE settings, the prior density is a fixed probability distribution, typically a multivariate normal distribution and accordingly the approximate posterior also is set to be a normal distribution. As our prior is a greatly complex distribution estimated by the flow, we are not incentivized by this to choose the posterior accordingly. Nevertheless

In the case of the **real musical data** we can not construct any informed posterior distribution, as the signals do not follow any constrained prior form. For the lack of an informed decision and the computational ease of the reparametrization trick, we resort to a Gaussian posterior. More specifically we propose employing a **truncated normal distribution**. The latent sound values are restricted to $[-1, 1]$ and a with a truncated Gaussian we can restrict the support to this interval. The reparametrization trick is still applicable, for a concrete description of the sampling refer to *Reparametrization of a truncated Gaussian*. The normal distribution is parametrized by mean and variance $\boldsymbol{\eta} = \{\sigma, \mu\}$.

Choosing an Encoder

The inference model $\texttt{Encoder}_{\boldsymbol{\phi}}(\boldsymbol{m})$ infers the parameters for the chosen posterior distribution from data. The model takes the time-domain signals as an input, therefore we use the architectural design choices as with the prior networks and employ a non-causal WaveNet architecture. As laid out before this model is fully convolutional, not autoregressive, as the prediction of the parameters is a one-to-one mapping, where each time-point is mapped to a set of

parameters. As it is practice we use the same network head for both parameters and append after the last skip-connection output of the WaveNet two convolutional layers with the final one mapping to the two separate parameters. The WaveNet used here is fairly similar to the modules used in the prior flows. They employ the gated convolution and have fixed kernel sizes of 3.

We note that we fix the encoder to work in the time-domain. We propose this to have access to the phase information of the original signals during the mapping $\boldsymbol{m} \rightarrow \boldsymbol{s}_k$. This does not make it necessary, that the prior models are also working in the time-domain. The samples taken from the approximate posterior during the training of the inference model can be transformed with the same spectral transformation used to train the prior models. We can compute the gradient of the likelihood from the prior under the spectral transformation[85]. Because we are not using the generative priors to generate in this setting, the phase information lost through the transformation is not disadvantageous and still is accessible to the inference model.

As we have N different sources with each a trained prior network we will also train N inference networks each with a separate approximate posterior.

[85] The `torchaudio` library provides implementations of real-valued spectrograms and a mel-transform that allow for automatic differentiation.

Experiments

All code can be found under: `github.com/morris-frank/unnamed-source-separation/` (MIT)

As we have laid out in the previous chapter our method mainly depends on the quality of the learned source priors. Remember we learn the source priors independently for each source channel. In the VAE-style approach, we learn the posterior for each search with the KL-divergence from the respective prior. In the sampling approach, we optimize the separation solutions under the source priors. Therefore we need the prior models to fulfill some qualities. First, the learned distributions have to generalize, as in both approaches the separation predictions have to be findable in the prior distributions. Second, the priors have to be smooth, so that the priors are capable of assigning lower to higher likelihood to samples that are more true to in-distributions samples. And third, the prior models have to be discriminative. They are trained only on samples of their respective class, but in the separation process, they have to able to discriminate the other signals in the mix from their in-distribution.

Datasets

We will be working on two datasets. First, a simplified Toy dataset which is made of simple generated signals and second the `musdb18` dataset which is a widely-used test-set for musical source separation.

ToyData

We simplify the problem domain to create a toy-like dataset. We randomly generate waves from four simple oscillations: a sine wave, a sawtooth wave, a square wave and a triangle wave; see Figure 10. We select the period and phase of each signal in each sample independently and randomly, but fixed for the sample. The frequency is a uniformly picked form the frequency bounds of the 88 keys of an equal-temperament tuned piano (see *Representations of audio*). The period is also uniformly picked. Given a wave from each source, the mix is computed by simply taking the mean. In our experiments we are gonna model these sources with probability densities, looking especially at the square that will pose a problem, as those only consist of two unique values (-1 and 1). This distribution would simplify the problem too much, therefore we also vary the amplitude of the sampled signals in the uniform range $[0.8, 1.0]$. The signals are sampled with a sampling rate of 14.700Hz.

Figure 10: One period of each of the four toy sources: sinus, sawtooth, square and triangle wave.

musdb18

Further we use the `musdb18`[86] dataset published for the 2018 Signal Separation Evaluation Campaign[87]. The dataset consists of 150 full songs covering various artists and genres, split into train and test sets sized 100 and 50, respectively. Each track is separated into the four sources *drums*, *bass*, *vocals* and *others*. The *others* source channel contains any remaining set of instruments not categorized under the first three ones. The song files are provided in the Stem audio file format[88] and encoded at 44.1kHz. Stem here is terming the provided source channels, we use the terms interchangeably.

The dataset consists of the 100 tracks taken from the DSD100[89] dataset and 46 track out of the MedleyDB[90] dataset. Both dataset contain tracks from a wide range of (western) musical genres, Rock, Pop, Rap, Jazz and Metal. While the upstream data sources provide more fine-grained meta-data, `musdb18` brings all tracks into the same format and stem splitting as described above.

Next to the separated stems, the dataset provides the original (studio) mix for each song. This mix is not equivalent to the simple linear mixing which we get by taking the mean. Nevertheless, the provided mix diverges only insignificantly from an auto-generated mix, as the original sources are provided in their post-compression, post-mixed form. This means that we can use the original mix and assume it to be reasonably close to the canonical linear mix.

As the songs are real natural songs, they are of different lengths. Our models will different to other recent methods, not be auto-regressive. Thus we sample fixed-length cuts from the songs as training and test samples. For the `musdb18` data, no pre-processing is applied, as the data already contains the wanted level variability, it spanning different genres and artists.

It is noted that the `musdb18` dataset while providing a remarkable diverse 10hr of real-world music, is a rather small numbered set of samples. Any data modelling from this set of data will consequently be biased. The dataset specifically is created for the accompanying separation challenge and will not translate to general music modelling.

Also for the `musdb18` dataset we downsample the signals to 14.700Hz.

[86] Rafii et al., *MUSDB18 - a Corpus for Music Separation*, 2017.

[87] Stöter et al., "The 2018 Signal Separation Evaluation Campaign", 2018.

[88] Native Instruments, *Stem Audio Format*, n.d.

[89] Liutkus et al., "The 2016 Signal Separation Evaluation Campaign", 2017.

[90] Bittner et al., "MedleyDB 2.0", 2016.

Figure 11: The four source channels for the `musdb18` dataset: bass ,drums, vocals and *other*.

Prior architectures

We construct two distinct prior models for the two datasets. All models follow the flow architecture described in *Method*. Table 1 gives a summary of the networks. Remember the affine transformations are parametrized by WaveNets, consisting of multiple dilated one-dimensional convolutions. We use the gated convolution and skip-connection setup as described in *Modeling raw audio*. We append two extra convolutional layers after the last dilated block with kernel size one and preceding ReLU units, respectively. The last

convolutional layer is initialized to zero. This guarantees that at the beginning of training each coupling layer defaults to the identity. All other convolutional weights are initialized with He initalization[91]. The number of features in each convolutional layer is fixed throughout the whole network, as given in the table. All kernels are size 3.

[91] He et al., "Delving Deep into Rectifiers", 2015.

	blocks	flows	layers	kernel size	width
toy (time)	4	6	10	3	32
`musdb18` (time)	8	6	10	3	48

Table 1: The hyperparameters for the two network architectures. The block is the biggest unit, containing a squeeze operation and n_f flow blocks. Each flow block contains a activation normalization unit, a affine coupling layer (WaveNet) and a flip operation. The WaveNets cosist of 10 layers each with fixed kernel size and width.

Training the priors

Training of the priors is done in mini-batches with an Adam optimizer[92]. We take random fixed-length samples from the songs/signals in each mini-batch. Each sample is 16384 long with is around 1.1sec. As all models are fully convolutional the input size is in no way regimented by the architecture, only in so far that we are avoiding padding in the lower layers. The initial learning rate is set to $1e-4$. The learning rate is decreased with $\gamma = 0.6$ and a fixed five-step decrease schedule. The toy model is trained with a batch size of 5 and the `musdb18` model with a batch size of 2. We train the two prior models first without any added noise for each 150.000. The training curves are shown in Figure 12 and Figure 13. As also observed by prior work, the training of deep normalizing flow models behaves unstable.

[92] Kingma and Ba, "Adam", 2017.

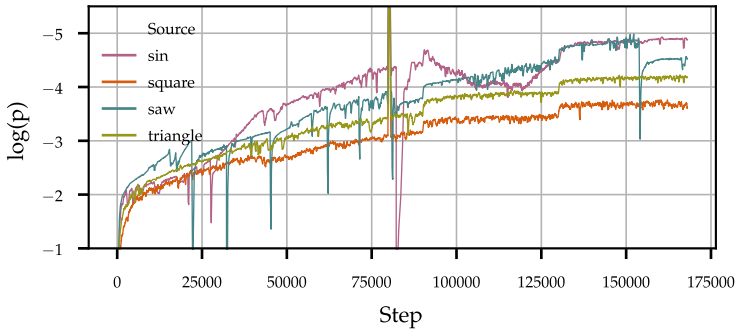


Figure 12: The negative mean log-likelihood during the training of each ToyData prior. We plot the likelihood and not the loss which include the sum of log-determinant. The instability of the training at the half comes from a restart of training.

For practicality, the priors for all source channels are contained in one model. The PyTorch framework allows for convolutional layers to be blocked into separate groups. We are not using any operations that are dependent over the batch dimension [93]. We implement the complete flow model in such a grouped way so that the training can be done in parallel on a single GPU while the inference and the gradients for the optimizer updates are independent over the grouped blocks. For that, we also implement the utility functions, squeeze and flip in the same blocked manner.

[93] That would e.g. be batch normalization

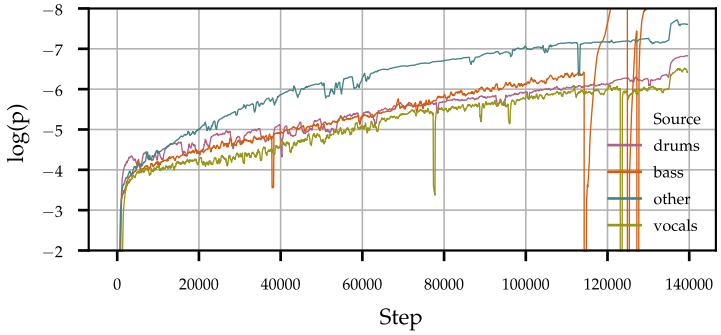


Figure 13: The negative mean log-likelihood during training of each `musdb18` prior.

Testing the priors

Training of the normalizing flow model combines two objectives: First, we maximize the log-determinant of the *non-volume preserving* affine coupling and the activation normalization layers. This guarantees us the invertibility of these transformations. Second, we maximize the likelihood of the projected samples under the chosen target distribution [94]. For the evaluation, we are interested in the mean average likelihood under the target distribution for different in- and out-of-distribution inputs. Remember that the flow gives a bijective mapping between the input and target space [95]. For an input of size $\boldsymbol{x} \in \mathbb{R}^{1\times L}$ the flow returns a latent $\boldsymbol{z} \in \mathbb{R}^{1\times L}$ of the same size. Each time-point latent $\boldsymbol{z}_t$ depends on the inputs from the receptive field of the flow before and after the respective input $\boldsymbol{x}_t$ but is evaluated under the independent margin $\boldsymbol{z}_t \sim \mathcal{N}(0,1)$. Because therefore the factorized multivariate Gaussian prior is equivalent to a set of one-dimensional Gaussians, the flow is also not size restricted. All transformations in the network are convolutional and use appropriate padding. L can be any chosen length. We want to set the input length larger than the receptive field of the network. The latents on either end side of the sample will be based on the larger amount of padded input, while the center values from the sample receive the full receptive field. The size of the receptive field is regimented by the depth and kernel sizes used in the convolutions of the coupling layers and the number of squeeze operations in the blocks (see Figure 9).

[94] The target distribution is the prior to our prior model. It is set to a factorized Gaussian.

[95] necessitated by the invertibility

Sampling from the priors

We show an example of sampling from the toy data priors in Figure 14. It should be unsurprising that the drawn samples are not returning a consistent curve. All the latents are from i.i.d. distributions that contribute to the generation of the data output in the bounds of its receptive field. The overlapping receptive fields generate locally consistent curves. This is not a hindrance as our method does not rely on sampling from the priors unconstrained.

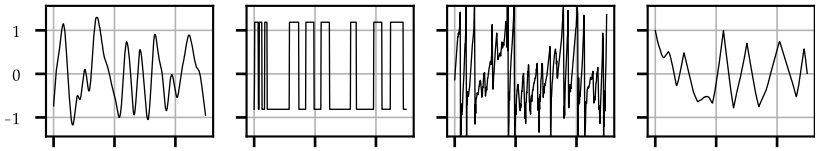
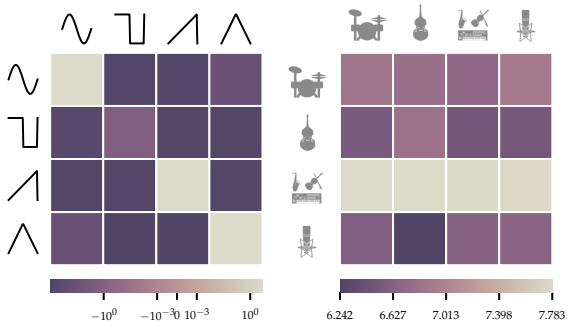


Figure 14: Samples from the prior trained on the toy data.

Testing cross-likelihood

In the full model, each prior is used to extract its source channel separately. Through their training, they explicitly contract the density for the positive, in-class examples. During separation, the priors therefore, encounter negative, out-of-distribution samples for the first time. To be useful for separation, the priors have to give a low likelihood to samples from the other classes of the dataset.



(a) ToyData (b) `musdb18`

Figure 15: The log-likelihood of each source under each prior for both sets of priors. Notice how for the ToyData we get the diagonal that we are expecting while for the real music the likelihood for every field is high and in the same range. The *other* prior assigns the highest likelihoods.

We test for this by calculating the mean log-likelihood of the test data set under each prior, for each source channel separately. What we anticipate is that the samples from the priors training source are of high likelihood while all other sources are of low likelihood. In Figure 15 (a) we show the source dependent likelihoods for the Toy Data. The in-distribution samples are all of high likelihood while all out-of distribution samples are highly un-likely. The Flow model, therefore, was able to detect out-of-distribution samples while only being trained on in-distribution samples. The estimated densities are discriminative.

When running the same experiment for the `musdb18` the discriminative power does not hold up, see Figure 15 (b). All signal sources are about equally likely under each prior density. We hypothesize this stems from the fact that the real musical data is severely more complicated compared to the Toy Data. The Flows model the appearance of sound in general, as the in-class variability of sounds is already high, without knowing anything about the *discriminative* differences between the instruments and does not infer the distribution from those features.

Even above that, the results exhibit an additional failing for the `musdb18` dataset. The third stem in the dataset files *other* is filled with a smorgasbord of different instruments. With our training regime, we are implying we can model the sound this set of unrelated instruments with only in-distribution samples with a resulting distribution which is discriminative against another set of unrelated instruments.

Intuitively this already fails to be reasonable. Ignoring this difficulty, the priors for `musdb18` still to fail to be discriminative in the remaining three classes.

We, therefore, continue the investigation of the flow models with the Toy Data.

Noise-conditioning

We fine-tune the noise-less flow models by continuing the training with added Gaussian noise on the input samples. Fine-tuning[96] the noise-less model instead of retraining the model is considerably cheaper and the noised distribution we want to approximate is close to the noise-free distribution of the initial training. We follow Jayaram and Thickstun[97] in evaluating the noised distribution at variances in 5 steps from 0 to 1, geometrically increasing. The signals with added noise are not capped to the range $[0, 1]$ as to not bias the signals towards the bounds of the interval.

We find that fine-tuning converges quickly at —-.

[96] Yosinski et al., "How Transferable Are Features in Deep Neural Networks?", 2014.

[97] Jayaram and Thickstun, "Source Separation with Deep Generative Priors", 2020.

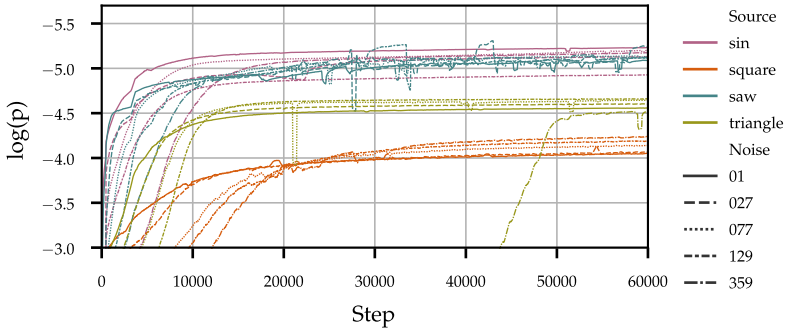
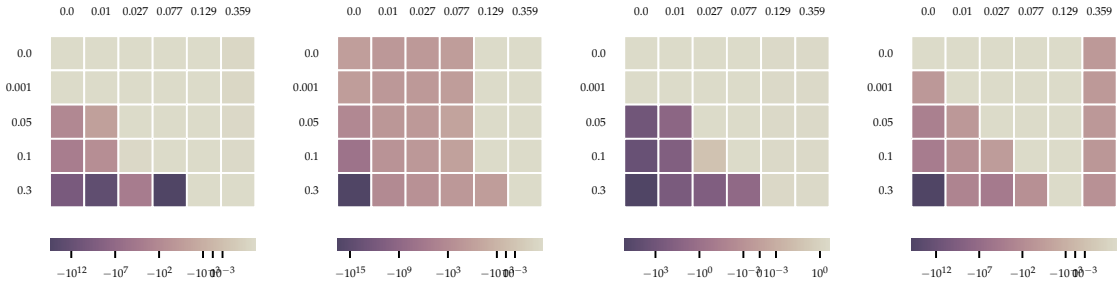


Figure 16: The negative mean log-likelihood during training of each `musdb18` prior.

Conditioning the prior distributions on noised samples is supposed to reduce local peaks in the latent space. Nalisnick et al.[98] points to the general problem that maximizing the Jacobian of the determinant in the objective of normalizing flows encourages the learned distribution to be sensitive to perturbations of inputs. They warn precisely of assuming anything about the density of a generative model outside of its training data-distribution. Jayaram and Thickstun[97] experimentally shows that for their source separation method the same problem can be alleviated by fine-tuning on noised samples. Figuratively speaking the Gaussian noise in data space translates to Gaussian smoothing of the peaks in the probability distribution. For our comparatively more complicated models, we want to assess the success of this step.

[98] Nalisnick et al., "Do Deep Generative Models Know What They Don't Know?", 2019.

In Figure 17 we show the mean average log-likelihood of the test data set with different levels of added noised under the priors fine-tuned with increasing levels of noise. We see that with the noiseless model the likelihood of noised samples quickly decreases with the increasing variance of the noise. Conditioning on a higher variance of noise levels out the likelihood for the lower levels of noise

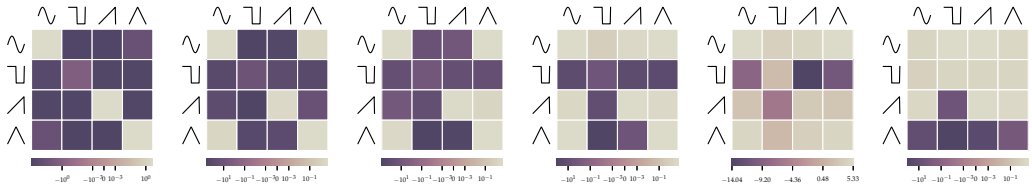


(a) sin (b) square (c) saw (d) triangle

Figure 17: The log likelihood of the test data set with different levels of noise $\mathcal{N}(0, \sigma)$ for the prior models fine-tuned on pertubated data with increasing levels of Gaussian noise. For the numerical data see Tables 10 to 13.

variances. This is not surprising as with the noise conditioning we explicitly model the noised distribution. Consider though the priors conditioned on the noise levels 0.027 or 0.077. The noised distribution is getting likely for noise levels beyond the conditioned level of noise. Now one might argue that the result, while unexpected, is not prohibitive as the whole point of noise-conditioning is too make 'close-to' in-distribution data around as likely as in-distribution data. We argue that our results do show that noise-conditioning the flow distribution is not a pertinent approach to reduce the contractiveness of the latent distribution. Conditioning the noiseless model on low levels of noised data quickly *destroys* the previously learned spiked latent distribution.

We show this failing more pronounced in Figure 18. Here we plot the mean average likelihood of the different source channels under each prior with widening noise-conditioning. The expectation is to preserve the discriminative power of the different source priors while reducing the sensitivity to noise. The results clearly show that trying to slowly decrease the sensitivity, quickly destroys any discriminative power the generative priors exhibit in the noiseless model.



(a) 0.0 (b) 0.01 (c) 0.027 (d) 0.077 (e) 0.129 (f) 0.359

Figure 18: The cross-likelihood of the toy source channels under each prior model after conditioning the distribution on different levels of noise.

With the noiseless model Figure 18 (a) the distribution is discriminative against the samples from the other sources. With increasing noise-conditioning the discriminative power is lost going to (f). The noising of the input data results in the latent distribution to flatten out to the point of not being able to differ between any of the signal channels.

Random and constant inputs

value	model	sin	square	saw	triangle
0.0	0.0	4.8e+00	-7.0e+02	4.4e+00	1.8e+00
	0.359	-5.0e-01	-3.1e+00	5.1e+00	-2.0e+11
1.0	0.0	-1.5e+01	-3.6e+03	-2.7e+06	-3.2e+02
	0.359	2.7e+00	4.5e+00	-2.8e+00	-1.1e+01

Table 2: The mean log likelihood of a full receptive field of contant inputs $\{0, 1\}$ for the noise-less and the widest noise-conditioned model.

Previous works[99,100,101] have pointed out that generative models tend to assign high likelihood to constant inputs. We examine the same on our Toy Data prior. Table 2 shows that for the noiseless model the mean data value 0 is indeed highly likely, except for the square wave, which we hypothesize stems from the square wave never having that value. Note further that for the noiseless model the maximum input 1 is highly unlikely. Noise-conditioning the distribution reverses both values in that the mean value becomes less likely and the maximum value becomes more likely.

[99] Sønderby et al., "Amortised MAP Inference for Image Super-Resolution", 2017.
[100] Van den Oord, Y. Li, et al., "Parallel WaveNet", 2017.
[101] Nalisnick et al., "Do Deep Generative Models Know What They Don't Know?", 2019.

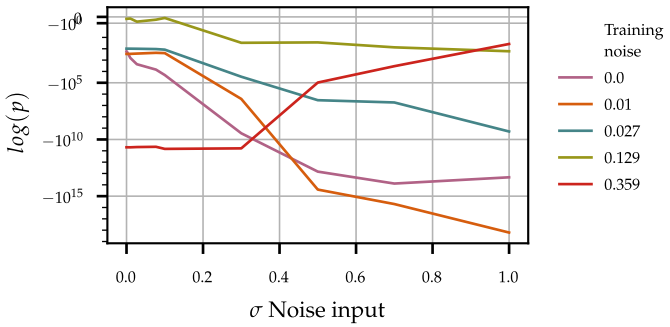


Figure 19: The likelihood of Gaussian noise under the Toy prior with increasing variance σ.

On the other hand purely random inputs should be unlikely and decreasing so with higher variance. We plot the relationship between the log-likelihood of Gaussian noise with varying variance under the noiseless and noise-conditioned priors in Figure 19. We see that for the noiseless model, any amount of noise is highly unlikely. But with wider noise-conditioning this decreases, until with a conditioning of 0.359 the effect even reverses.

We read these results to support the previous interpretation that even a small amount of noise fine-tuning can have severe effects on the estimated density. The noiseless model has sharp likelihood peaks around true data, in which even small amounts of noised data are highly unlikely. The noise-conditioning of the flow models flattens these peaks in so far that noise and out-of-distribution samples become highly likely, even at small amounts of added noise.

Langevin sampling

We tried source separation using the Toy dataset with random mixes and the trained flow priors. As already anticipated with the previous experiments, this is not successful. The noiseless model does not exhibit a smooth manifold which even with the noised gradients of SGLD is optimizable. The noised priors on the other hand, lost all the discriminative power and therefore it is not possible to modulate the out-of-distribution mix into a true signal by taking (noisy) steps on the prior.

Conlusion

In this research work, we introduce a new approach to unsupervised source separation. We describe theoretically that source separation can be achieved by modelling the generative process of the mixed signal from which then likely latent sources are extracted. Because we propose to model the data distribution of each signal source independently we remove the necessity of having mixed-unmixed training pairs. We sketch out two approaches to samples those sources, either by learning separate approximate posteriors or by directly sampling from the prior density under the mixing constraint.

Our experiments reinforce a suspicion that was also experimentally found in prior work under different circumstances. Current deep generative models do not learn a density that is discriminative against out-of-distribution samples.

Bibliography

Andreas Jansson, Eric J. Humphrey, Nicola Montecchio, Rachel M. Bittner, Aparna Kumar, and Tillman Weyde (2017). "Singing Voice Separation with Deep U-Net Convolutional Networks." (Suzhou, China) (cit. on p. 17).

Bittner, Rachel M., Julia Wilkins, Hanna Yip, and Juan P. Bello (2016). "MedleyDB 2.0: New Data and a System for Sustainable Data Collection". In: *ISMIR Late Breaking and Demo Papers* (cit. on p. 36).

Burgess, Christopher P., Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner (2018). "Understanding Disentangling in Beta-VAE". In: arXiv: 1804.03599 [cs, stat] (cit. on p. 12).

Chandna, Pritish, Merlijn Blaauw, Jordi Bonada, and Emilia Gomez (2019). "A Vocoder Based Method For Singing Voice Extraction". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 990–994. arXiv: 1903.07554 (cit. on p. 17).

Chandna, Pritish, Marius Miron, Jordi Janer, and Emilia Gómez (2017). "Monoaural Audio Source Separation Using Deep Convolutional Neural Networks". In: *Latent Variable Analysis and Signal Separation*. Ed. by Petr Tichavský, Massoud Babaie-Zadeh, Olivier J.J. Michel, and Nadège Thirion-Moreau. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 258–266 (cit. on p. 17).

Chorowski, Jan, Ron J. Weiss, Samy Bengio, and Aäron van den Oord (2019). "Unsupervised Speech Representation Learning Using WaveNet Autoencoders". In: *IEEE/ACM Trans. Audio Speech Lang. Process.* 27.12, pp. 2041–2053. arXiv: 1901.08810 (cit. on p. 18).

Cohen-Hadria, Alice, Axel Roebel, and Geoffroy Peeters (2019). "Improving Singing Voice Separation Using Deep U-Net and Wave-U-Net with Data Augmentation". In: arXiv: 1903.01415 [cs, eess] (cit. on p. 22).

Dauphin, Yann N., Angela Fan, Michael Auli, and David Grangier (2017). "Language Modeling with Gated Convolutional Networks". In: arXiv: 1612.08083 [cs] (cit. on pp. 22, 23).

Défossez, Alexandre, Nicolas Usunier, Léon Bottou, and Francis Bach (2019). "Demucs: Deep Extractor for Music Sources with Extra Unlabeled Data Remixed". In: arXiv: 1909.01174 [cs, eess, stat] (cit. on p. 23).

Défossez, Alexandre, Neil Zeghidour, Nicolas Usunier, Leon Bottou, and Francis Bach (2018). "SING: Symbol-to-Instrument Neural Generator". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 9041–9051 (cit. on p. 23).

Dinh, Laurent, David Krueger, and Yoshua Bengio (2015). "NICE: Non-Linear Independent Components Estimation". In: arXiv: 1410.8516 [cs] (cit. on p. 13).

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density Estimation Using Real NVP". In: arXiv: 1605.08803 [cs, stat] (cit. on pp. 13, 30).

Douglas, O. and O. Shaughnessy (2000). "Speech Communications: Human and Machine". In: *IEEE press, Newyork*, pp. 367–433 (cit. on p. 16).

Dumoulin, Vincent and Francesco Visin (2018). "A Guide to Convolution Arithmetic for Deep Learning". In: arXiv: 1603.07285 [cs, stat] (cit. on p. 22).

Fourier, Jean-Baptiste-Joseph (1822). *Théorie analytique de la chaleur*. Paris: F. Didot (cit. on p. 16).

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Networks". In: arXiv: 1406.2661 [cs, stat] (cit. on p. 23).

Grais, Emad M., Dominic Ward, and Mark D. Plumbley (2018). "Raw Multi-Channel Audio Source Separation Using Multi-Resolution Convolutional Auto-Encoders". In: arXiv: 1803.00702 [cs] (cit. on p. 23).

Gröchenig, Karlheinz (2001). *Foundations of Time-Frequency Analysis*. Applied and Numerical Harmonic Analysis. Birkhäuser Basel (cit. on p. 16).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: arXiv: 1502.01852 [cs] (cit. on p. 37).

Hershey, John R., Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe (2015). "Deep Clustering: Discriminative Embeddings for Segmentation and Separation". In: arXiv: 1508.04306 [cs, stat] (cit. on p. 21).

Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2016). "Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: (cit. on p. 12).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on p. 17).

Huang, Po-Sen, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis (2014). "Singing-Voice Separation from Monaural Recordings Using Deep Recurrent Neural Networks". In: *ISMIR* (cit. on p. 22).

Isik, Yusuf, Jonathan Le Roux, Zhuo Chen, Shinji Watanabe, and John R. Hershey (2016). "Single-Channel Multi-Speaker Separation Using Deep Clustering". In: arXiv: 1607.02173 [cs, stat] (cit. on p. 22).

ISO/TC 43 Acoustics (1975). *ISO 16 – Acoustics — Standard Tuning Frequency* (cit. on p. 15).

Jansson, Andreas, Eric J. Humphrey, Nicola Montecchio, Rachel M. Bittner, Aparna Kumar, and Tillman Weyde (2017). "Singing Voice Separation with Deep U-Net Convolutional Networks". In: *ISMIR* (cit. on p. 22).

Jayaram, Vivek and John Thickstun (2020). "Source Separation with Deep Generative Priors". In: arXiv: 2002.07942 [cs, stat] (cit. on pp. 31, 40).

Jordan, Michael I., Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul (1999). "An Introduction to Variational Methods for Graphical Models". In: *Machine Learning* 37.2, pp. 183–233 (cit. on p. 10).

Kalchbrenner, Nal, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu (2018). "Efficient Neural Audio Synthesis". In: arXiv: 1802.08435 [cs, eess] (cit. on p. 18).

Kaspersen, Esbern Torgard (2019). "HydraNet: A Network For Singing Voice Separation". In: (cit. on p. 22).

Kim, Sungwon, Sang-gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon (2019). "FloWaveNet : A Generative Flow for Raw Audio". In: arXiv: 1811.02155 [cs, eess] (cit. on pp. 19, 28).

Kingma, Diederik P. and Jimmy Ba (2017). "Adam: A Method for Stochastic Optimization". In: arXiv: 1412.6980 [cs] (cit. on pp. 30, 37).

Kingma, Diederik P. and Prafulla Dhariwal (2018). "Glow: Generative Flow with Invertible 1x1 Convolutions". In: arXiv: 1807.03039 [cs, stat] (cit. on pp. 13, 28).

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes". In: arXiv: 1312.6114 [cs, stat] (cit. on p. 11).

— (2019). "An Introduction to Variational Autoencoders". In: arXiv: 1906.02691 [cs, stat] (cit. on p. 10).

Kotelnikov, Vladimir (1933). "On the Carrying Capacity of the Ether and Wire in Telecommunications". In: *Material for the First All-Union Conference on Questions of Communication*. Moscow: Izd. Red. Upr. Svyazi RKKA (cit. on p. 15).

Le Roux, Jonathan, Gordon Wichern, Shinji Watanabe, Andy Sarroff, and John R. Hershey (2019). "Phasebook and Friends: Leveraging Discrete Representations for Source Separation". In: *IEEE Journal of Selected Topics in Signal Processing* 13.2, pp. 370–382 (cit. on p. 17).

Li, Li and Hirokazu Kameoka (2018). "Deep Clustering with Gated Convolutional Networks". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 16–20 (cit. on p. 22).

Liu, Yun, Hui Zhang, Xueliang Zhang, and Linju Yang (2019). "Supervised Speech Enhancement with Real Spectrum Approximation". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5746–5750 (cit. on p. 17).

Liutkus, Antoine, Fabian-Robert Stöter, Zafar Rafii, Daichi Kitamura, Bertrand Rivet, Nobutaka Ito, Nobutaka Ono, and Julie Fontecave (2017). "The 2016 Signal Separation Evaluation Campaign". In: *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*. Ed. by Petr Tichavský, Massoud Babaie-Zadeh, Olivier J.J. Michel, and Nadège Thirion-Moreau. Cham: Springer International Publishing, pp. 323–332 (cit. on p. 36).

Lluís, Francesc, Jordi Pons, and Xavier Serra (2019). "End-to-End Music Source Separation: Is It Possible in the Waveform Domain?" In: arXiv: 1810.12187 [cs, eess] (cit. on pp. 17, 23).

Luo, Yi and Nima Mesgarani (2018). "TasNet: Time-Domain Audio Separation Network for Real-Time, Single-Channel Speech Separation". In: arXiv: 1711.00541 [cs, eess] (cit. on p. 24).

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT press (cit. on p. 53).

Nalisnick, Eric, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan (2019). "Do Deep Generative Models Know What They Don't Know?" In: arXiv: 1810.09136 [cs, stat] (cit. on pp. 40, 41).

Narayanaswamy, Vivek Sivaraman, Sameeksha Katoch, Jayaraman J. Thiagarajan, Huan Song, and Andreas Spanias (2019). "Audio Source Separation via Multi-Scale Learning with Dilated Dense U-Nets". In: arXiv: 1904.04161 [cs, eess, stat] (cit. on p. 22).

Native Instruments (n.d.). *Stem Audio Format* (cit. on p. 36).

Neal, Radford M. (2012). "MCMC Using Hamiltonian Dynamics". In: arXiv: 1206.1901 [physics, stat] (cit. on p. 14).

Paine, Tom Le, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A. Hasegawa-Johnson, and Thomas S. Huang (2016). "Fast Wavenet Generation Algorithm". In: arXiv: 1611.09482 [cs] (cit. on p. 18).

Pascual, Santiago, Antonio Bonafonte, and Joan Serrà (2017). "SEGAN: Speech Enhancement Generative Adversarial Network". In: arXiv: 1703.09452 [cs] (cit. on p. 23).

Prenger, Ryan, Rafael Valle, and Bryan Catanzaro (2018). "WaveGlow: A Flow-Based Generative Network for Speech Synthesis". In: arXiv: 1811.00002 [cs, eess, stat] (cit. on p. 18).

Pulse Code Modulation (PCM) of Voice Frequencies (1972). G.711. ITU-T (cit. on pp. 15, 17).

Rafii, Zafar, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner (2017). *MUSDB18 - a Corpus for Music Separation*. Version 1.0.0 (cit. on p. 36).

Rethage, Dario, Jordi Pons, and Xavier Serra (2018). "A Wavenet for Speech Denoising". In: arXiv: 1706.07162 [cs] (cit. on p. 23).

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: arXiv: 1401.4082 [cs, stat] (cit. on p. 12).

Robbins, Herbert and Sutton Monro (1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. JSTOR: 2236626 (cit. on p. 14).

Slizovskaia, Olga, Leo Kim, Gloria Haro, and Emilia Gomez (2019). "End-to-End Sound Source Separation Conditioned On Instrument Labels". Version 2. In: arXiv: 1811.01850 [cs, eess] (cit. on p. 22).

Sønderby, Casper Kaae, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár (2017). "Amortised MAP Inference for Image Super-Resolution". In: arXiv: 1610.04490 [cs, stat] (cit. on p. 41).

Stevens, S. S., J. Volkmann, and E. B. Newman (1937). "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *The Journal of the Acoustical Society of America* 8.3, pp. 185–190 (cit. on p. 16).

Stoller, Daniel, Sebastian Ewert, and Simon Dixon (2018). "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation" (Paris, France) (cit. on p. 22).

Stöter, Fabian-Robert, Antoine Liutkus, and Nobutaka Ito (2018). "The 2018 Signal Separation Evaluation Campaign". In: arXiv: 1804.06267 [cs, eess] (cit. on p. 36).

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2014). "Going Deeper with Convolutions". In: arXiv: 1409.4842 [cs] (cit. on pp. 17, 23).

Tabak, Esteban and Cristina V. Turner (2013). "A family of nonparametric density estimation algorithms". In: *COMMUN. PURE & APPL. MATHS.* 66.2, pp. 145–164 (cit. on p. 12).

Tan, Ke and DeLiang Wang (2019). "Complex Spectral Mapping with a Convolutional Recurrent Network for Monaural Speech Enhancement". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 6865–6869 (cit. on p. 17).

Van den Oord, Aäron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "WaveNet: A Generative Model for Raw Audio". In: arXiv: 1609.03499 [cs] (cit. on p. 17).

Van den Oord, Aäron, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu (2016). "Conditional Image Generation with PixelCNN Decoders". In: arXiv: 1606.05328 [cs] (cit. on pp. 17, 18).

Van den Oord, Aäron, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, et al. (2017). "Parallel WaveNet:

Fast High-Fidelity Speech Synthesis". In: arXiv: 1711.10433 [cs] (cit. on p. 41).

Van den Oord, Aäron, Oriol Vinyals, and Koray Kavukcuoglu (2017). "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 6306–6315 (cit. on p. 18).

Welling, Max and Yee Whye Teh (2011). "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, pp. 681–688 (cit. on pp. 14, 27, 31).

Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson (2014). "How Transferable Are Features in Deep Neural Networks?" In: arXiv: 1411.1792 [cs] (cit. on p. 40).

Appendices

Reparametrization of a truncated Gaussian

We propose to model the approximate posterior distribution with a truncated Gaussian. To optimize the parameters of the encoder we need to take gradient of the samples with respect to the parameters of the distribution $\eta = \{\mu, \sigma\}$. For the normal distribution this can be cheaply done with the reparametrization trick. We do the same for the truncated distribution.

We can take samples from the normal distribution $\mathcal{N}(\mu, \sigma)$ truncated to the bounds $[a, b]$ with a inverse transform sampler[102]. We give an algorithm to take a sample from the truncated distribution and return its log-likelihood:

[102] Murphy, *Machine Learning*, 2012.

Algorithm 3: Sampling from a truncated Gaussian with given variance σ and mean μ in the bound $[a, b]$.

procedure SAMPLETRUNCATEDNORMAL(μ, σ, a, b)
  $a_{\text{cdf}} \leftarrow \Phi(\frac{a-\mu}{\sigma})$
  $b_{\text{cdf}} \leftarrow \Phi(\frac{b-\mu}{\sigma})$
  $\epsilon \sim U[a_{\text{cdf}}, b_{\text{cdf}}]$ ▷ $U[\cdot]$ gives a uniform sample.
  $\xi \leftarrow \Phi^{-1}(\epsilon)$
  $x \leftarrow \sigma \cdot \xi + \mu$
  $ll \leftarrow -\log(\sigma) - \frac{1}{2}(\xi^2 + \log(2\pi)) - \log(b_{\text{cdf}} - a_{\text{cdf}})$
  return x, ll
end procedure

The cumulative distribution function of a normal distribution and its inverse are:

$$\Phi(x) = \frac{1}{2} \cdot (1 + \text{erf}(\frac{x}{\sqrt{2}})) \tag{57}$$

$$\Phi^{-1}(x) = \text{erf}^{-1}(\sqrt{2} \cdot ((2x) - 1)) \tag{58}$$

where $\text{erf}(\cdot)$ is the error function.

Notice that we separated out the source of stochasticity to the uniform distribution. Both the error function $\text{erf}(\cdot)$ and its inverse $\text{erf}^{-1}(\cdot)$ are implemented in the PyTorch framework making the automatic computation of the gradients possible.

Cross-likelihood

	drums	bass	other	vocals
drums	6.9e+00	6.8e+00	6.7e+00	6.9e+00
bass	6.5e+00	6.8e+00	6.5e+00	6.5e+00
other	7.8e+00	7.8e+00	7.8e+00	7.8e+00
vocals	6.6e+00	6.2e+00	6.6e+00	6.7e+00

Table 3: The mean average log-likelihood of all source channels under the each prior for the `musdb18` data.

	sin	square	saw	triangle
sin	4.8e+00	-8.9e+02	-7.5e+02	-4.7e+01
square	-4.1e+02	-1.8e+00	-6.9e+02	-6.8e+02
saw	-6.8e+02	-8.2e+02	4.1e+00	-6.7e+02
triangle	-4.6e+01	-1.0e+03	-1.0e+03	4.0e+00

Table 4: The mean average log-likelihood of all source channels under the each prior for the noise level 0.0.

	sin	square	saw	triangle
sin	5.4e+00	-4.2e+02	-2.7e+02	2.6e+00
square	-1.5e+02	-1.3e+01	-8.0e+01	-1.2e+02
saw	-8.6e+01	-4.1e+02	3.9e+00	-1.9e+01
triangle	2.4e+00	-1.8e+02	-9.7e+01	4.8e+00

Table 5: The mean average log-likelihood of all source channels under the each prior for the noise level 0.01.

	sin	square	saw	triangle
sin	5.4e+00	-2.9e+01	-8.8e+00	4.8e+00
square	-5.3e+01	-9.3e+00	-5.0e+01	-5.1e+01
saw	-7.4e+00	-6.8e+01	4.1e+00	1.8e+00
triangle	3.5e+00	-8.0e+02	-5.7e+02	4.7e+00

Table 6: The mean average log-likelihood of all source channels under the each prior for the noise level 0.027.

	sin	square	saw	triangle
sin	4.9e+00	6.5e-01	3.5e+00	5.1e+00
square	-2.0e+01	-2.6e+00	-2.0e+01	-2.4e+01
saw	2.7e+00	-1.1e+01	4.5e+00	3.7e+00
triangle	4.7e+00	-1.0e+02	-2.9e+00	4.9e+00

Table 7: The mean average log-likelihood of all source channels under the each prior for the noise level 0.077.

	sin	square	saw	triangle
sin	5.3e+00	4.1e+00	4.8e+00	5.3e+00
square	-8.6e+00	1.8e+00	-1.4e+01	-1.1e+01
saw	2.5e+00	-6.1e+00	3.2e+00	2.8e+00
triangle	4.7e+00	1.4e+00	4.0e+00	4.5e+00

Table 8: The mean average log-likelihood of all source channels under the each prior for the noise level 0.129.

	sin	square	saw	triangle
sin	1.5e+00	4.0e+00	3.6e+00	2.0e+00
square	8.2e-01	1.5e+00	1.5e+00	1.6e+00
saw	2.3e+00	-3.4e+00	3.2e+00	2.9e+00
triangle	-1.3e+01	-1.1e+02	-3.5e+01	-1.3e+00

Table 9: The mean average log-likelihood of all source channels under the each prior for the noise level 0.359.

Noise-conditioning

	0.0	0.01	0.027	0.077	0.129	0.359
0.000	4.8e+00	5.4e+00	5.4e+00	4.9e+00	5.3e+00	1.5e+00
0.001	4.3e+00	5.4e+00	5.4e+00	4.9e+00	5.3e+00	1.5e+00
0.050	-1.7e+02	-1.1e-01	4.6e+00	5.3e+00	5.4e+00	2.9e+00
0.100	-7.3e+03	-2.2e+01	2.2e+00	4.9e+00	5.3e+00	3.2e+00
0.300	-7.5e+09	-3.7e+12	-1.1e+04	-2.7e+14	4.2e+00	3.8e+00

Table 10: The numerical data as visualized 17 for the source `sin`.

	0.0	0.01	0.027	0.077	0.129	0.359
0.000	-1.8e+00	-1.3e+01	-9.3e+00	-2.6e+00	1.8e+00	1.5e+00
0.001	-2.8e+00	-1.3e+01	-9.3e+00	-2.6e+00	1.8e+00	1.5e+00
0.050	-2.3e+03	-1.8e+01	-7.5e+00	-3.3e-01	1.8e+00	1.8e+00
0.100	-8.3e+06	-8.0e+01	-1.3e+01	-8.1e-01	1.5e+00	2.2e+00
0.300	-3.8e+16	-1.9e+03	-1.5e+02	-1.3e+01	-2.2e+00	2.3e+00

Table 11: The numerical data as visualized 17 for the source `square`.

	0.0	0.01	0.027	0.077	0.129	0.359
0.000	4.1e+00	3.9e+00	4.1e+00	4.5e+00	3.2e+00	3.2e+00
0.001	3.5e+00	3.9e+00	4.1e+00	4.5e+00	3.2e+00	3.2e+00
0.050	-8.9e+02	-1.2e+01	3.8e+00	4.5e+00	3.2e+00	3.2e+00
0.100	-3.5e+03	-6.4e+01	3.1e-02	4.3e+00	3.1e+00	3.2e+00
0.300	-1.4e+05	-2.3e+02	-8.5e+01	-6.2e+00	2.6e+00	3.2e+00

Table 12: The numerical data as visualized 17 for the source `saw`.

	0.0	0.01	0.027	0.077	0.129	0.359
0.000	4.0e+00	4.8e+00	4.7e+00	4.9e+00	4.5e+00	-1.2e+00
0.001	-1.8e+00	4.8e+00	4.7e+00	4.9e+00	4.5e+00	-1.2e+00
0.050	-5.5e+03	-1.9e+00	3.8e+00	4.8e+00	4.8e+00	-1.6e+00
0.100	-2.1e+04	-2.7e+01	-5.2e-01	4.4e+00	4.7e+00	-2.4e+00
0.300	-7.2e+14	-9.7e+02	-3.6e+04	-2.3e+01	3.6e+00	-1.5e+01

Table 13: The numerical data as visualized 17 for the source `triangle`.