University of California
Los Angeles

Cooperative Learning of Deep Generative Models with Application in Sound Synthesis

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Statistics

by

Ruiqi Zhong

2017

Abstract of the Thesis

Cooperative Learning of Deep Generative Models with Application in Sound Synthesis

by

Ruiqi Zhong
Master of Science in Statistics
University of California, Los Angeles, 2017
Professor Ying Nian Wu, Chair

Fires, rainstorms or insect swarms produce natural sounds made up of rapidly occurring acoustic events, which we call "sound textures". This kind of phenomena has been studied by computational audio research community [MS11] and neural science people for a long time. From previous studies, it has been verified that sound textures can be schematically synthesized from statistical models fairly well. Here we take a novel approach involving neural networks or deep learning methods. Specifically, we use cooperative training of a descriptor and a generator network, modeled as a convolutional neural network (ConvNet) and a deconvolutional neural network (DeconvNet) respectively. From several experiments, we prove that our framework can capture the essence of sound textures and synthesize identifiable natural sounds.

The thesis of Ruiqi Zhong is approved.

Hongquan Xu

Arash Ali Amini

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2017

To my parents for their unconditional love and support.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor Prof. Ying Nian Wu for his great support for this project. His precious guidance and encouragements helped a lot when O wrote this thesis. I also would like to thank Prof. Arash Ali Amini and Prof. Hongquan Xu for serving as my thesis committee members.

My sincere thanks to my lab-mates Dr. Jianwen Xie, Ruiqi Gao, Tian Han and Yang Lu. Their comments and advice were very instrumental to this project.

I also want to thank the mentors in my earlier academic life. They are Prof. Song-Chun Zhu, Prof. Gregory Sharkhnarovich and Dr. Han Hu.

Last but not least, thanks to Glenda Jones who helped clarify administrative stuff. And many thanks to all other staff members and students in the Statistics department.

CHAPTER 1

Introduction

In recent years, deep learning (DL)[GBC16] methods have achieved remarkable success in supervised learning or predicative learning on varieties of computer vision and natural language processing tasks. The current most prevailing architecture of neural networks - ConvNets [LBB98] - have proven to be very powerful in approximating highly non-linear mapping from a high-dimensional input, such as an image, to an output (either low-dimensional, such as a category, or high-dimensional, such as a 3D map). As for training, they can be learned fairly fast and well by simple training algorithms such as stochastic gradient descent that converges to local modes of the loss functions. Due to their empirical high performance and flexibility, DL methods are inspiring a revolution in artificial intelligence (AI) related industry.

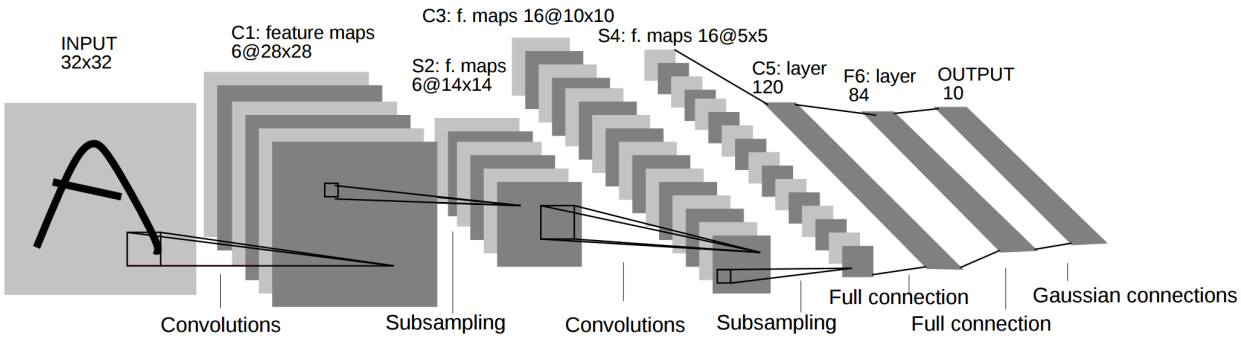


Figure 1.1: Architecture for ConvNet used by [LBB98] for digits recognition

In academia, however, we are also curious about learning how to represent the high-dimensional input itself by discovering hidden structures in the input or extracting features from the input. The mathematical form of such a representation is a statistical model that

defines a probability distribution defined on the space of the input signal. Learning such statistical models can be important for unsupervised or semi-supervised learning where the outputs are not available or otherwise very scarce. This is also what a statistician cares about — understanding and explaining the data more than just making predictions.

Sound textures, usually represented by one-dimensional temporal data, have been studied in traditional ways by biology research communities. From former studies [MOS09], it has been verified that sound texture perception may rely on statistical representations in human auditory systems. Some recent works have tried applying ConvNets to environmental sound classification tasks [Pic15] and achieved relatively high accuracy on public datasets. Here in this thesis we explore the possibility of directly encoding and synthesizing sound textures using highly non-linear deep generative models learnt by cooperative training of two ConvNets (CoopNets) [XLG16].

CHAPTER 2

Related Work

2.1 Sound Texture Synthesis via Statistical Methods

Many natural sounds, such as those produced by winds, storms, rivers, or animals at night, are made up of great numbers of rapidly reoccurring acoustic events, which we call "sound textures". In [MOS09], a synthesis methodology was proposed by combining individual frequency channel statistics and correlations between them, producing identifiable and natural sounds.

In order to successfully synthesize a vivid sound texture, they first obtained desired values of the statistics by calculating the model responses for a real-world sound. They then went through an iterative procedure to progressively modify a random noise signal, forcing it to have these desired statistic values. By starting from noise, they managed to generate a signal that had maximum entropy and is only constrained by the desired statistics. This study suggests that sound texture perception could rely on statistical measurements and that human auditory system may use these statistics to discriminate lots of natural sound textures. This insight lies the cornerstone of many later works.

2.2 Generative Adversarial Networks (GAN)

In GAN, the goal is to simultaneously train two opponent networks — the discriminator (D) and the generator (G). D outputs a scalar $D(\mathbf{x})$ representing the degree of belief that x is from data rather than from G. Whereas G works by taking a latent variable z sampled from

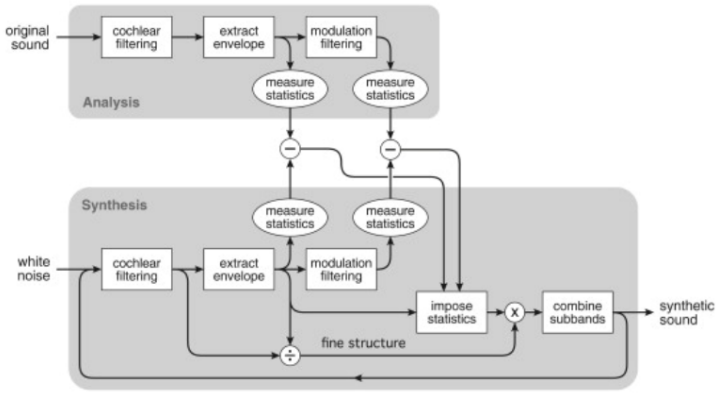


Figure 2.1: Schematic of synthesis procedure in [MOS09]

uniform distribution and then map it through several deconvolutional layers to an output in data space, thus implicitly defining a generative model. When training, D and G work against each other by playing a two-layer minimax game with value function V(G, D):

$$\min_G \max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}))] \tag{2.1}$$

In each iteration, they firstly update the weights of D by ascending its stochastic gradient, and then G by descending its stochastic gradient. Under certain conditions, the algorithm will make converge to the optimal situation where $p_{data} = p_g$ and $D(\mathbf{x}) = \frac{1}{2}$ almost everywhere. In that case, the discriminator will be unable to discriminate between from real data and generated data. In other words, G will generate too authentic or realistic data for D to discriminate, which signs the success of generator net G.

So far, GAN has already been successfully applied to lots of tasks including interactive image generation, image in-painting and image super resolution. Tons of attention from AI research community has been drawn to this promising training framework. We, however, adopt a different philosophy that advocates cooperation instead of adversary.

CHAPTER 3

Methodology: Cooperative Training of Descriptor and Generator Networks

Artificial neural networks are very good at approximating highly non-linear mapping from high-dimensional data to an output, usually defined as feature vector or probability scalar. So far, ConvNete have become one of the most common forms of network architectures adopted by machine learning researchers and practitioners. By inserting them into exponential family models, we obtain extremely complex and highly flexible probability distributions. In the following, this chapter will give you a detailed review of our methodology called CoopNets.

3.1 ConvNet: generalizing GLM and linear spline

By definition, a ConvNet can be viewed as a mapping $Y = f(X; W)$. Both the input X and the output Y can be multi-dimensional. Sometimes Y can also be one-dimensional. W denotes the parameters, or more specifically, weights of convolution filters. The function f is represented by a concatenation of L layers of liner transformations followed by element-wise non-linear transformations:

$$X^{(l)} = f_l(W_l X^{(l-1)} + b_l), \tag{3.1}$$

where $l = 1, ..., L$, $X^{(0)} = X$, $X^{(L)} = Y$, and $W = (W_l, b_l, l = 1, ..., L)$, where W_l is the weight matrix and b_l is the bias or intercept vector. f_l is element-wise transformation, i.e., for $X = (x_1, ..., x_d)^\top$, $f_l(X) = (f_l(x_1), ..., f_l(x_d))^\top$. $Y = f(X; W)$ generalizes two familiar structures in statistics.

(1) *Generalized linear model (GLM).* A GLM structure is a composition of a linear combination of the input variables and a non-linear link function. A ConvNet can be viewed as a recursion of this structure, where each component of $X^{(l)}$ is a GLM transformation of $X^{(l-1)}$, with f_l being the link function.

(2) *Linear spline.* The form of a one-dimensional linear spline is like

$$y = \beta_0 + \sum_{k=1}^{d} \beta_k \max(0, x - a_k), \tag{3.2}$$

where a_k are the knots. As is well known, the most commonly used link function in modern ConvNets is $f_l(x) = \max(0, x)$, which is called rectified linear unit (ReLU). The resulting rectified ConvNet can be viewed as a multi-dimensional linear spline. The number of linear pieces is exponential in the number of layers [PMB13]. Theoretically, such a structure is able to approximate any continuous non-linear function by a sufficiently large number of linear pieces.

Convolution. The input signal X can be defined on a spatial or temporal (or spatial-temporal) domain. Actually, by re-ordering the coordinates, we can always flatten X into a vector as we like. In ConvNets, it is assumed that the column vectors in W_l (as well as the elements in b_l) form different groups (or channels). Back in the spatial or temporal domain, the vectors in the same group are localized and translation invariant versions of each other, like wavelets. Each group of vectors can be called a filter, a kernel, or a wavelet basis function. For simplicity and narrative, we shall hide this technical detail in our review.

Back-propagation. In order to train a ConvNet, we need to calculate the gradient or partial derivatives $\partial f(X; W)/\partial W$ for updating parameters in W. Besides, $\partial f(X; W)/\partial X$ is needed for Langevin sampling of X. Both derivatives can be computed by the chain-rule back-propagation, and they share the computation of $\partial X^{(l)}/\partial X^{(l-1)} = f_l^{'}(W_l X^{(l-1)} + b_l)W_l$ in the chain rule, where $f_l^{'}$ is the Jacobian matrix of f_l. Because f_l is element-wise, $f_l^{'}$ is a diagonal.

3.2 Two classes of statistical models

Exponential family models can be written in the following form

$$P(Y;W) = \frac{1}{Z(W)} \exp\left[\sum_{k=1}^{d} w_k \phi_k(Y)\right] q(Y), \tag{3.3}$$

where Y is a D-dimensional vector, $W = (w_k, k = 1, ..., d)$ are the parameters, $(\phi_k(Y), k = 1, ..., d)$ are the sufficient statistics or features, $q(Y)$ is the reference distribution, and $Z(W) = E_q[\exp(\sum_{k=1}^{d} w_k \phi_k(Y))]$ is the normalizing constant.

The class of exponential family models also include Markov random field models (or the Gibbs distributions), such as Ising model, Potts model, etc., if the features $(\phi_k(Y))$ are locally defined. In machine learning, they are called energy-based models, or undirected graphical models.

Latent variable models are also called directed graphical models. The most prominent model in this class is the factor analysis model. Let Y be a D-dimensional observed data vector. Let X be the d-dimensional vector of latent factors, $X = (x_k, k = 1, ..., d)^\top$. The traditional factor analysis model is

$$X \sim N(0, I_d); Y = WX + \epsilon; \epsilon \sim N(0, \sigma^2 I_D); \epsilon \perp X, \tag{3.4}$$

where W is a $D \times d$ matrix, and ϵ is a D-dimensional error vector. I_d is d-dimensional identity matrix.

The factor analysis model is the prototype of many subsequent models that generalize the prior model of X. (1) Independent component analysis [HKO04]: $d = D$, $\epsilon = 0$, and x_k are assumed to be independent sources with heavy tailed distributions. (2) Sparse coding [OF97]: $d > D$, and X is assumed to be a redundant but sparse vector, i.e., only a small number of x_k are non-zero or significantly different from zero. (3) Non-negative matrix factorization [LS01]: it is assumed that $x_k \geq 0$. (4) Recommender system [KBV09]: X is a vector of a customer's desires in different aspects, and w_i (i-th row of W) is a vector of a product's desirabilities in these aspects.

3.3 Generator net: generalizing factor analysis

The model. The way we generalize factor analysis models is to replace the linear mapping WX to a non-linear mapping $g(X; W)$ parametrized by a ConvNet. Actually before us this form of model has already been proposed by [GPM14]. Mathematically, we can write it as a direct generalization of (3.4):

$$X \sim N(0, I_d); Y = g(X; W) + \epsilon; \epsilon \sim N(0, \sigma^2 I_D); \epsilon \perp X. \tag{3.5}$$

similar to $f(X; W)$, $g(X; W)$ can be viewed as a layer-wise recursion of factor analysis, with $X^{(l-1)} = g_l(W_l X^{(l)} + b_l)$ for $l = 1, ..., L$, and $W = (W_l, b_l, l = 1, ..., L)$. $X^{(0)} = g(X; W)$, and $X^{(L)} = X$. $X^{(l)}$ can be interpreted as factors at layer l. At the bottom layer that generates Y, we assume $g_1(x) = x$ (or $g_1(x) = \tanh(x)$ to make the elements of Y fall within [-1, 1]). For $l > 1$, if $g_l(x) = \max(0, x)$ or piecewise linear, then $g(X; W)$ is piecewise linear, and the whole model becomes a piecewise linear factor analysis.

Learning algorithm. In the literature, this model is trained by methods that involve learning extra networks [GPM14, KW14, RMW14, MG14]. In a recent work [HLZ16a], they recognize that the model is a non-linear generalization of factor analysis which can be learned by maximum likelihood using the EM algorithm [RT82], where both the E-step and the M-step are based on multivariate linear regressions. Inspired by this observation, we propose an alternating back-propagation algorithm for learning the generator network that iterates between the following two-steps:

(G1) *Inferential step*: Use langevin dynamics to infer the latent factors for each training example.

(G2) *Learning step*: Given the inferred latent factorsm, update the parameters by gradient descent.

We use back-propagation algorithm for the gradient computations of both steps.

Specifically, when we observe a training set of data vectors $\{Y_i, i = 1, ..., n\}$, each Y_i has a corresponding X_i, but all the Y_i share the same parametrization of the ConvNet W. We

can train the generator network by maximizing the observed-data log-likelihood $L(W) = \sum_{i=1}^{n} \log P(Y_i; W) = \sum_{i=1}^{n} \log \int P(Y_i, X_i; W) dX_i$.

According to the following well-known fact in EM, the gradient of $L(W)$ can be calculated as follows:

$$\begin{aligned} \frac{\partial}{\partial W} \log P(Y; W) &= \frac{1}{P(Y; W)} \int \left[\frac{\partial}{\partial W} \log P(Y, X; W) \right] P(Y, X; W) dX \\ &= E_{P(X|Y,W)} \left[\frac{\partial}{\partial W} \log P(Y, X; W) \right]. \end{aligned} \tag{3.6}$$

Because the expectation is analytically intractable, we approximate it by sampling from $P(X|Y, W)$ and then computing the Monte Carlo average.

The joint distribution

$$\log P(Y, X; W) = \log [P(X) P(Y|X, W)] \tag{3.7}$$

$$= -\frac{1}{2\sigma^2} \|Y - g(X; W)\|^2 - \frac{1}{2} \|X\|^2 + \text{const.} \tag{3.8}$$

According to Bayesian rule, the posterior distribution of the latent factors $P(X|Y, W) = P(Y, X; W)/P(Y; W) \propto P(Y, X; W)$. One step of the Langevin dynamics for sampling X from $p(X|Y, W)$ is

$$X_{\tau+1} = X_\tau + \frac{s^2}{2} \left[\frac{1}{\sigma^2} (Y - g(X_\tau; W)) \frac{\partial}{\partial X} g(X_\tau; W) - X_\tau \right] + sU_\tau, \tag{3.9}$$

where τ denotes the time step, s is the step size, and $U_\tau \sim \mathrm{N}(0, I_d)$. We can view the Langevin dynamics (3.9) as an explain-away process, where the latent factors in X compete to explain away the current residual $Y - g(X_\tau; W)$.

We can use stochastic gradient algorithm of [You99] for learning, where in each iteration, for each Y_i, only a single copy of X_i is sampled from $p(X_i|Y_i, W)$ by running a finite number of steps of Langevin dynamics starting from the current value of X_i (called warm start). With the sampled $\{X_i\}$, we can update the parameters W based on the gradient $L'(W)$, whose Monte Carlo approximation is:

$$L'(W) \approx \sum_{i=1}^{n} \frac{\partial}{\partial W} \log P(Y_i, X_i; W) = \sum_{i=1}^{n} \frac{1}{\sigma^2} (Y_i - g(X_i; W)) \frac{\partial}{\partial W} g(X_i; W). \tag{3.10}$$

It is actually a non-linear regression of Y_i on X_i.

Algorithm G. We call the learning algorithm the Algorithm G. To summarize it, after initializing W and $\{X_i\}$, Algorithm G iterates the following two steps. *Step G1*: starting from each current X_i, run l steps of Langevin dynamics to update X_i according to (3.9). *Step G2*: update $W^{(t+1)} = W^{(t)} + \gamma_t L'(W^{(t)})$, with $L'(W^{(t)})$ computed according to (3.10). γ_t is the learning rate. The convergence of this algorithm follows [You99].

Preliminary results. Our results show that the generator model is surprisingly expressive in that it can generate realistic images, sounds and videos. We adopt the structure of the generator network of [RMC15, DSB15], where the top-down network consists of multiple layers of deconvolution by linear superposition, ReLU non-linearity, and up-sampling, with tanh non-linearity at the bottom-layer [RMC15] to make the signals fall within $[-1, 1]$. We fix $\sigma = 0.3$ for the standard deviation of the noise vector ϵ. We use $l = 10$ or 30 steps of Langevin dynamics within each learning iteration, and the Langevin step size s is set at 0.1 or 0.3. We run $T = 600$ learning iterations, with learning rate .0001, and momentum .5.

Figure 3.1: Modeling texture patterns. The 224×224 images are the observed images. The 448×448 images are the images generated by the learned models $g(X; W)$ where X is a Gaussian white noise image.

Usefulness of generator: (1) Analysis: it disentangles the variations in the input data by independent factors. (2) Synthesis: it can synthesize new signals by direct sampling. (3) Embedding: it embeds the high-dimensional data vectors in the low-dimensional factor space (similar to but much more explicit than multi-dimensional scaling). (4) Interpolation: linear interpolation in factor space results in meaningful non-linear interpolation in signal space. (5) Indexing: the latent factors provide indexing of the signal, which can be used for other

tasks such as recognition.

3.4 Descriptor net: generalizing exponential family models

The model. We can generalize the exponential family model (3.3) using ConvNet. In our recent work [XLZ16], we assume the following model:

$$P(Y; W) = \frac{1}{Z(W)} \exp\left[f(Y; W)\right] q(Y), \tag{3.11}$$

where $q(Y)$ is the reference distribution like Gaussian white noise $q(Y) \propto \exp\left(-\|Y\|^2/2\sigma^2\right)$, $f(Y; W)$ is defined by a ConvNet whose parameters are denoted by W. This ConvNet is bottom-up because it maps the signal Y to a one-dimensional value. See the diagram in (3.3). $Z(W) = \int \exp\left[f(Y; W)\right] q(Y) dY = \mathrm{E}_q\{\exp[f(Y; W]\}$ is the normalizing constant. We show that this model can be derived from the commonly used ConvNet for multinomial logistic classification, if we assume $q(Y)$ to be a base category [XLZ16].

Relation with exponential family and Markov random fields. The model can be considered a recursive linear combination of features, which is the structure in exponential family models. Specifically, $Y^{(l)} = f_l(W_l Y^{(l-1)} + b_l)$, where $l = 1, ..., L$, $Y^{(0)} = Y$, $Y^{(L)} = f(Y; W)$. $Y^{(l)}$ can be interpreted as features or filter responses at layer l. f_l is element-wise non-linear transformation. At the top layer that gives us $f(Y; W)$, we assume $f_L(r) = r$ and $b_L = 0$, so that $f(Y; W) = W_L Y^{(L-1)}$. We can compare it with $\sum_{k=1}^{d} w_k \phi_k(Y)$ in the exponential family model (3.3), so that $W_L = (w_k, k = 1, ..., d)$ and $Y^{(L-1)} = (\phi_k(Y), k = 1, ..., d)^\top$, and the features $Y^{(L-1)} = (\phi_k(Y))$ themselves involve unknown parameters and are to be learned. In fact, our starting point in this line of work is to use existing ConvNet features $Y^{(L-1)}$ learned in classification task as $(\phi_k(Y))$ [LZW16]. The resulting model is a Markov random field model if the features are locally defined, and the features correspond to the clique functions of the Markov random field or the Gibbs distribution.

Auto-encoder revealed. We discover an interesting auto-encoding representational structure in the above model [XLZ16]. For the commonly used rectification $f_l(r) = \max(0, r) = 1(r > 0)r$ for $l = 1, ..., L-1$, where $1()$ is the indicator function, we can write $Y^{(l)} =$

$\delta_l(Y;W)(W_l Y^{(l-1)} + b_l)$, where $\delta_l(Y;W) = diag(1(W_l Y^{(l-1)} + b_l > 0))$, i.e., a diagonal matrix of binary indicators (the indicator function is applied element-wise) [PMB13]. Let $\delta = (\delta_l, l = 1, ..., L)$ consists of indicators at all the layers ($\delta_L = 1$ at the top layer), then $f(Y;W) = A_{\delta(Y;W)}Y + a_{\delta(Y;W)}$ is piecewise linear, where $A_\delta = \prod_{l=L}^{1} \delta_l W_l$ and a_δ can be similarly calculated. Assuming the standard deviation σ of the Gaussian reference $q(Y)$ is 1 for simplicity, then the energy function of the model, $\|Y\|^2/2 - f(Y;W)$, is piecewise quadratic, thus $P(Y;W)$ is piecewise Gaussian. On the piece $\{\delta(Y) = \delta\}$ with a fixed δ, the density $P(Y;W)$ is $\mathrm{N}(A_\delta, I_D)$ truncated to this piece. If the mean A_δ is also a local mode, then it satisfies an auto-encoder $Y = A_{\delta(Y;W)}$, which consists of a bottom-up computation of the indicators at different layers $\delta = \delta(Y;W)$, and a top-down computation of the reconstruction $Y = A_\delta$, where δ_l play the role of coefficients, and W_l play the role of basis functions in the top-down generation process. Unlike the generator network, the auto-encoder structure is explicit in both the bottom-up and top-down directions, whereas in the generator model, the inference is implicit. This auto-encoder should be approximately true for those Y that are close to the local modes.

Learning algorithm. The learning algorithm iterates the following two steps:

(D1) *Synthesis step*: Sampling synthesized examples from the current model by Langevin dynamics.

(D2) *Learning step*: Update the parameters by gradient descent to shift the density from the synthesized examples towards the observed examples.

Again, the gradient computations in both steps are powered by back-propagation.

Specifically, suppose we observe training examples $\{Y_i, i = 1, ..., n\}$. The maximum likelihood training seeks to maximize $L(W) = \frac{1}{n}\sum_{i=1}^{n} \log P(Y_i;W)$. The gradient of the $L(W)$ is

$$L'(W) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial W} f(Y_i;W) - \mathrm{E}_W\left[\frac{\partial}{\partial W} f(Y;W)\right], \tag{3.12}$$

where E_W denotes the expectation with respect to $P(Y;W)$.

The expectation in equation (3.12) is analytically intractable and has to be approximated

by MCMC, such as Langevin dynamics, which samples from $P(Y; W)$ by iterating the following step:

$$Y_{\tau+1} = Y_\tau - \frac{s^2}{2}\left[\frac{Y_\tau}{\sigma^2} - \frac{\partial}{\partial Y} f(Y_\tau; W)\right] + sU_\tau, \tag{3.13}$$

where τ indexes the time steps of the Langevin dynamics, s is the step size, and $U_\tau \sim \mathrm{N}(0, I_D)$ is the Gaussian white noise term. In the case of piecewise linear $f(Y; W) = A_{\delta(Y;W)}Y + a_{\delta(Y;W)}$, the above Langevin equation is driven by $Y/\sigma^2 - A_{\delta(Y;W)}$, which is the reconstruction error of the auto-encoder.

We can run $\tilde{n}$ parallel chains of Langevin dynamics according to (3.13) to obtain the synthesized examples $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$. The Monte Carlo approximation to $L'(W)$ is

$$L'(W) \approx \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial W} f(Y_i; W) - \frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}} \frac{\partial}{\partial W} f(\tilde{Y}_i; W), \tag{3.14}$$

which is the difference between the observed examples and the synthesized examples.

Algorithm D. We call the learning algorithm the Algorithm D. To summarize it, after initializing W and $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$, Algorithm D iterates the following two steps. *Step D1*: starting from each current $\tilde{Y}_i$ (called warm start), run l steps of Langevin dynamics to update $\tilde{Y}_i$ according to (3.13). *Step D2*: update $W^{(t+1)} = W^{(t)} + \gamma_t L'(W^{(t)})$, with $L'(W^{(t)})$ computed according to (3.14). γ_t is the learning rate. The convergence of this algorithm follows [You99].

Contrastive divergence and auto-encoder. A popular and more efficient alternative to the maximum likelihood method is contrastive divergence learning [Hin02], where in Step D1, instead of starting from the current synthesized images $\{\tilde{Y}_i\}$, we start the MCMC sampling from the observed images $\{Y_i\}$. We show in [XLZ16] that for the descriptor net, the contrastive divergence learning actually minimizes the average reconstruction error of the auto-encoder elucidated above, i.e., the average of $\|Y_i - A_{\delta(Y_i,W)}\|^2$.

Preliminary results. Our results show that the descriptor model is also surprisingly expressive. We use $\tilde{n} = 16$ parallel chains for Langevin sampling. The number of Langevin iterations between every two consecutive updates of parameters is $l = 10$. The training images are of size 224×224, whose intensities are within $[0, 255]$. We fix $\sigma^2 = 1$ in the reference

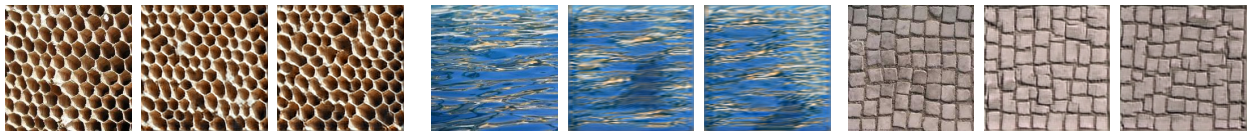


Figure 3.2: Generating texture patterns. For each category, the first image is the training image, and the rest are 2 of the images generated by the learning algorithm.

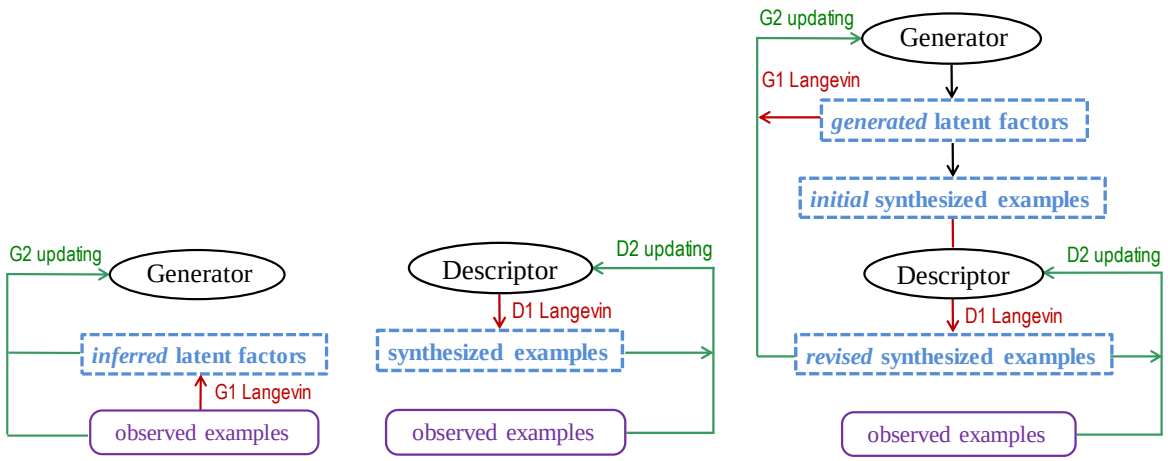
distribution q.

3.5 CoopNets: cooperative training of two models

Recently we discover that we can couple the maximum likelihood learning algorithms of the descriptor and generator nets into a cooperative training algorithm that we call the CoopNets algorithm [HLZ16b]. The main motivation is to use the generator net to generate the starting points to initialize the MCMC sampling of the descriptor net, because the generator can generate synthesized examples directly without MCMC. The reward for the generator is that it can learn from the synthesized data (instead of the observed data) with the known latent factors it has used to generate the initial synthesized data.

The contributions of CoopNets have four aspects. (1) It expands and enriches our precious collection of statistical models, and make the models more expressive and enable them to learn from big data. (2) It will lead to new learning and inference algorithms for these models, including a cooperative training algorithm that trains two complementary models together. (3) It will lead to a new learning-based method for Markov Chain Monte Carlo (MCMC), where a statistical model can accumulate the MCMC transitions and reproduce them in one step. (4) It will lead to non-linear generalizations of unsupervised learning methods such as matrix factorization and completion. By conducting experiments on audio signals, we further prove the versatility and effectiveness of CoopNets.

Fig. 3.3(a) and (b) display the flowcharts of Algorithm G and Algorithm D respectively. Fig. 3.3(c) displays the flowchart of the CoopNets algorithm. We now use $W_{\mathcal{G}}$ and $W_{\mathcal{D}}$ to denote the parameters of the generator net $g(X; W_{\mathcal{G}})$ and the descriptor net $f(Y; W_{\mathcal{D}})$



(a) Algorithm G (b) Algorithm D (c) CoopNets algorithm

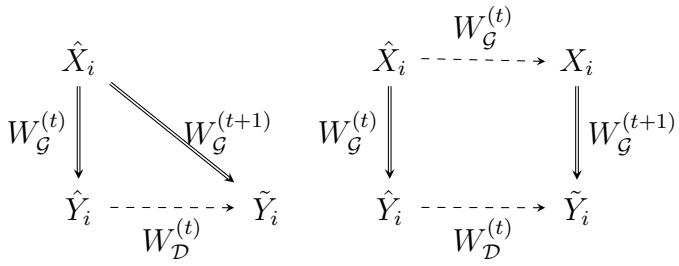
Figure 3.3: (a) Algorithm G involves sampling from the posterior distribution of the latent factors by Langevin dynamics. (b) Algorithm D involves sampling from the current model by Langevin dynamics. (c) CoopNets algorithm. The part of the flowchart for training the descriptor is similar to Algorithm D, except that the D1 Langevin sampling is initialized from the initial synthesized examples supplied by the generator. The part of the flowchart for training the generator can also be mapped to Algorithm G, except that the revised synthesized examples play the role of the observed examples, and the known generated latent factors can be used as inferred latent factors (or be used to initialize the G1 Langevin sampling of the latent factors).

respectively. In Step D1, we can initialize the synthesized examples by generating examples from the generator net. We first generate $\hat{X}_i \sim \text{N}(0, I_d)$, and then generate $\hat{Y}_i = g(\hat{X}_i; W_{\mathcal{G}}) + \epsilon_i$, for $i = 1, ..., \tilde{n}$. If the current generator $P_{\mathcal{G}}$ is close to the current descriptor $P_{\mathcal{D}}$, then the generated $\{\hat{Y}_i\}$ should be a good initialization for sampling from the descriptor net, i.e., starting from the $\{\hat{Y}_i, i = 1, ..., \tilde{n}\}$, we run Langevin dynamics in Step D1 for $l_{\mathcal{D}}$ steps to get $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$, which are revised versions of $\{\hat{Y}_i\}$. These $\{\tilde{Y}_i\}$ can be used as the synthesized examples from the descriptor net. We can then update $W_{\mathcal{D}}$ according to Step D2 of Algorithm D.

In order to update $W_{\mathcal{G}}$ of the generator net, we treat the $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$ produced by the above Step D1 as the training data for the generator. Since these $\{\tilde{Y}_i\}$ are obtained by

Figure 3.4: Generating texture patterns. For each category, the first image is the training image, and the rest are 3 of the images generated by the CoopNets algorithm.

the Langevin dynamics initialized from the $\{\hat{Y}_i, i = 1, ..., \tilde{n}\}$ produced by the generator net with known latent factors $\{\hat{X}_i, i = 1, ..., \tilde{n}\}$, we can update $W_{\mathcal{G}}$ by learning from $\{(\tilde{Y}_i, \hat{X}_i), i = 1, ..., \tilde{n}\}$ in Step G2, which is a supervised learning problem, or more specifically, a non-linear regression of $\tilde{Y}_i$ on $\hat{X}_i$. At $W_{\mathcal{G}}^{(t)}$, the latent factors $\hat{X}_i$ generates and thus reconstructs the initial example $\hat{Y}_i$. After updating $W_{\mathcal{G}}$, we want $\hat{X}_i$ to reconstruct the revised example $\tilde{Y}_i$. That is, we revise $W_{\mathcal{G}}$ to absorb the revision from $\hat{Y}_i$ to $\tilde{Y}_i$ made by the descriptor, so that the generator shifts its density from $\{\hat{Y}_i\}$ to $\{\tilde{Y}_i\}$. The left diagram in (3.15) illustrates the basic idea.



(3.15)

In the two diagrams in (3.15), the double-line arrows indicate generation and reconstruction by the generator net, while the dashed-line arrows indicate Langevin dynamics for revision and inference in the two nets. The diagram on the right in (3.15) illustrates a more rigorous method, where we initialize the Langevin inference of $\{X_i, i = 1, ..., \tilde{n}\}$ in Step G1 from $\{\hat{X}_i\}$, and then update $W_{\mathcal{G}}$ in Step G2 based on $\{(\tilde{Y}_i, X_i), i = 1, ..., \tilde{n}\}$. The diagram on the right shows how the two nets jumpstart each other's Langevin dynamics.

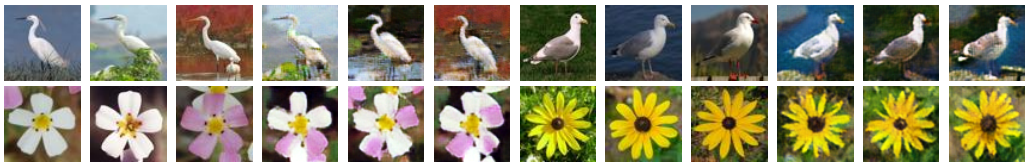


Figure 3.5: Generating object patterns. For each object category, the first 3 images are 3 of the training images, and the rest are 3 of the images generated by the CoopNets algorithm.

CHAPTER 4

Experiments

4.1 Implementation Details

For simplicity, We build our code on top of MatConvNet [VL15]. We try two sound textures obtained from on-line resources, one from bird chirping and one from dog barking, each enduring for about 5 seconds. More than synthesizing new chirping and barking, we also conduct experiments mixing these two sound textures. All experiments are run on NVIDIA GeForce GTX 750 Ti graphics card, which has 640 CUDA cores and 2048 MB memory. It takes about 10 minutes to train for 500 epochs. During experiments, we try different hyper parameters and ConvNet structures. In the following parts, we will present the results in different experiment settings.

	General		Generator	Descriptor
Number of Iterations	500	MCMC Step Size	0.001	0.1
Number of Markov Chains	2	Learning Momentum	0.1/0.2/0.3	1e-6
Steps of Langevin Dynamics	30/40/50	Noise Variance	0.03	1

Table 4.1: Hyper parameter settings

Generator Network The input to the generator network is a 1 * 60 vector sampled from a Gaussian noise distribution, and the output is a 1 * 60000 vector representing the generated audio. The network is made up of 3 deconvolution layers without usually added Rectified Linear Units (ReLU). Each deconvolution layer involves up-sampling by 10 times. Since we are dealing with textures, there is no need to add a fully connected layer on top of

layers of deconvolution. The network architecture is shown in Table 4.2.

Layer Name	Filter Size	Upsamplings Factor	Number of Filters	ReLU
DeConv1	25	10	256	×
DeConv2	25	10	128	×
DeConv3	25	10	1	×

Table 4.2: Generator network architecture.

Descriptor Network The descriptor network takes the 1 * 60000 audio signal as input and gives a feature vector. The network architecture is shown in Table 4.3.

Layer Name	Filter Size	Stride	Number of Filters	ReLU
Conv1	200	20/40/60	100	✓
Conv2	50	20	50	✓
Conv3	30	5	20	✓

Table 4.3: Descriptor network architecture

Calculation of Receptive Field (RF) for ConvNets The receptive field (RF) l_k of layer k is:

$$l_k = l_{k-1} + ((f_k - 1) * \prod_{i=1}^{k-1} s_i) \tag{4.1}$$

where l_{k-1} is the receptive field of layer $k-1$, f_k is the filter size (height or width, but assuming they are the same here), and s_i is the stride of layer i.

The formula above calculates receptive field from bottom up (from layer 1). Intuitively, RF in layer k covers $(f_k - 1) * s_{k-1}$ more pixels relative to layer $k-1$. However, the increment needs to be translated to the first layer, so the increments is a factorial a stride in layer $k-1$ corresponds to exponentially more strides in the lower layers.

When tuning the parameters we look at this variable for reference, which turn out to very useful.

Evaluation of Convergence We use the L2 norm of difference between sounds generated by G and the revised version by D to measure the degree of cooperation of two nets or the convergence of the algorithm.

4.2 Results and Analysis

4.2.1 Single Input

In this section, we present the results from models trained separately to synthesize a single sound texture signal - bird chirping or dog barking.

Results with Different Sizes of Receptive Fields We change the sizes of receptive fields by changing the stride of the 1st layer convolution of the descriptor network, which is directly connected to the input layer. According to Equation 4.1, the sizes of receptive fields of descriptor D in Figure 4.1 are 6390, 12680 and 18970 from left to right respectively.

We can see that the middle one presents much more authentic results than the other two. Consequently we may conclude that the quality of synthesized sounds does not change monotonically against the sizes of receptive fields. An appropriate setting of filter sizes and their strides is critical to the success of our algorithm.

Effects of Changing Learning Momentum We can see from Figure 4.2 that with larger momentum leading to larger learning rate, the learning path becomes less stable or the reconstruction error will even explode. To anneal this phenomenon, we enlarge the number of Langevin Dynamics steps.

Effects of Changing Steps of Langevin Dynamics Clearly from Figure 4.3, when we make more Langevin Dynamics steps in each iteration, the learning path returns normal. Next, we can compare the results under these two stable settings of learning momentum and number of Langevin dynamics.

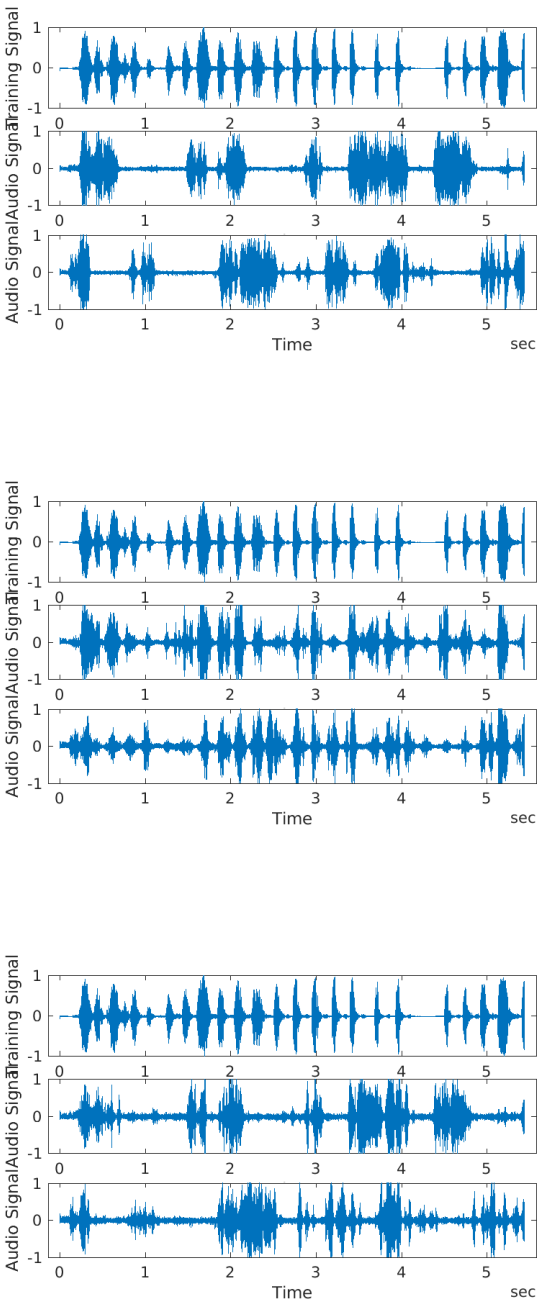


Figure 4.1: Synthesis results with different strides of the 1st layer of descriptor. Up: 20, middle: 40, down: 60.

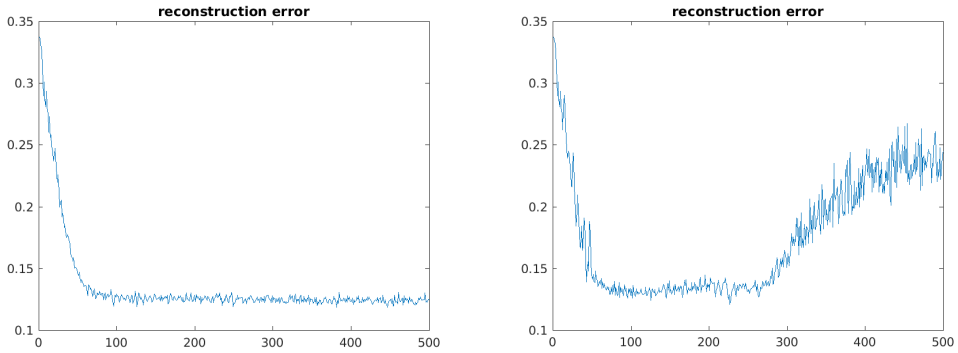


Figure 4.2: Reconstruction error with different learning Momentums. Left: 0.1, right: 0.2.

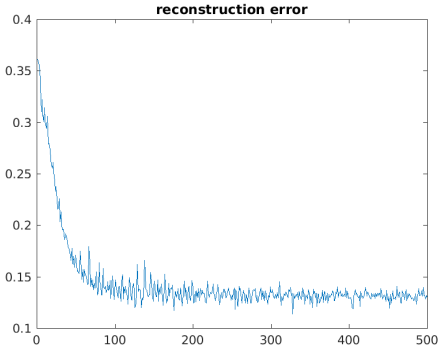


Figure 4.3: Reconstruction drror with learning momentum of 0.2 and 50 steps of Langevin dynamics.

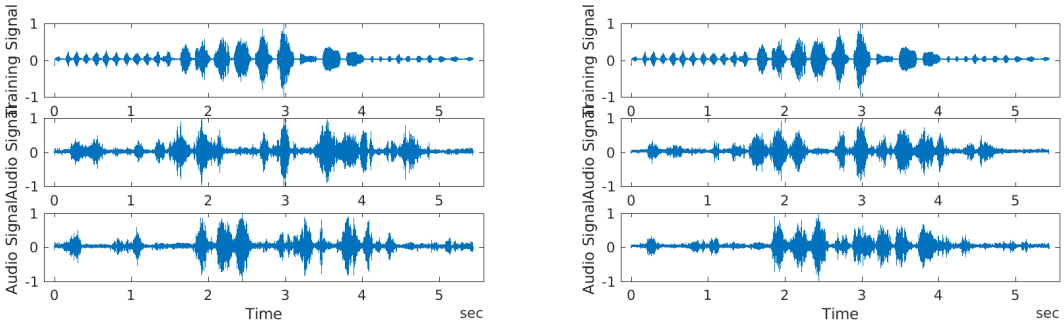


Figure 4.4: Synthesis results under two learning Settings. Left: momentum 0.1, number of Langevin dynamics: 40; right: momentum: 0.2, number of Langevin dynamics: 50.

In each subgraph of Figure 4.4, the first row is training signal, and the last two rows are synthesized signals. We can see that both settings can generate very authentic results while the right one sounds less noisy.

4.2.2 Multiple Inputs

By feeding the Descriptor with two sound signals, the CoopNets algorithm will synthesize signals that sound like a mixing of these two inputs. Here we give the result under one setting - steps of Langevin Dynamics: 40, learning momentum: 0.2, stride of Conv1 layer in D: 60.

When listening, you can hear both dog barking and bird chirping from the synthesized signals.

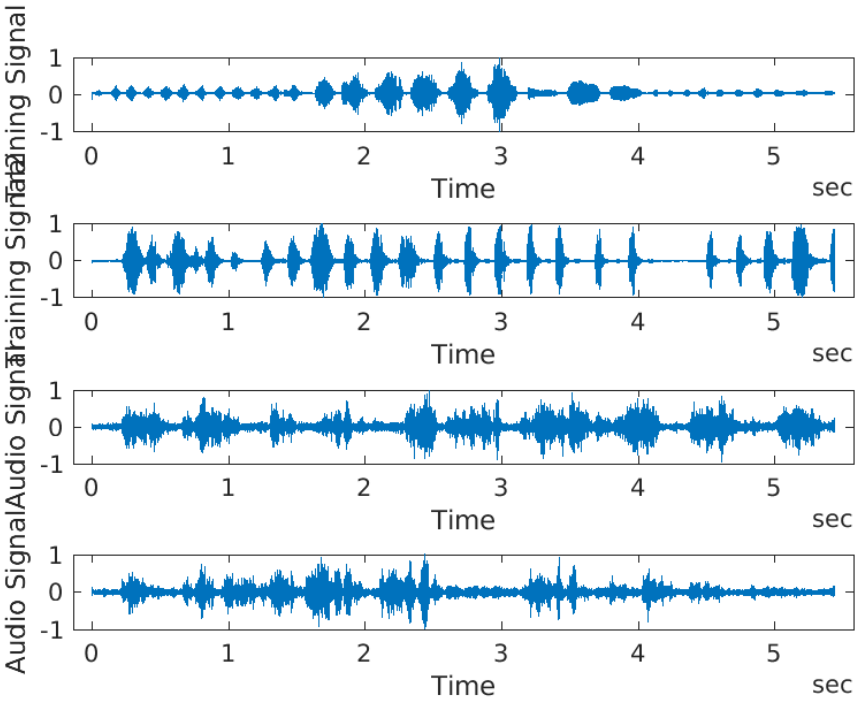


Figure 4.5: Synthesis results mixing two signals: bird chirping and dog barking.

CHAPTER 5

Discussion

In this thesis we innovatively apply a cooperative training framework called CoopNets to the task of sound synthesis. Contrary to traditional statistical methods, we embed two ConvNets - one bottom-up descriptor and one top-down generator - in our unified probabilistic model. The maximum likelihood estimations of these two ConvNets can be obtained by alternating back-propagation. By jump starting each other's Langevin Dynamics sampling process, these two algorithms manage to cooperate with each other. We do various experiments on sound textures in many different settings and compare their performances graphically. All of the original and generated audio files are included in supplementary materials in WAV format. When listening to the synthesized signals, we find them simulating the training examples quite well. By simulating we mean that they are different from the original examples but still very similar, indicating that our model has captured the essential knowledge of sound textures rather than simply memorizing the input audio signals.

For future work, we can try synthesizing more complicated sounds than natural sound textures by turning the models into inhomogeneous versions. Further more, we can apply this model on much larger scale data sets, which is the trend in current DL related research. We believe that such a useful and flexible framework as CoopNets will attract many more researchers to explore.

References

[DSB15] E Dosovitskiy, J. T. Springenberg, and T Brox. "Learning to Generate Chairs with Convolutional Neural Networks." In *CVPR*, 2015.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GPM14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *NIPS*, pp. 2672–2680, 2014.

[Hin02] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence." *Neural Computation*, **14**(8):1771–1800, 2002.

[HKO04] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[HLZ16a] Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. "Alternating Back-Propagation for generator network." *arXiv preprint arXiv:1606.08571*, 2016.

[HLZ16b] Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. "Cooperative Training of Descriptor and Generator Networks." *arXiv preprint arXiv:1609.09408*, 2016.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer*, (8):30–37, 2009.

[KW14] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes." *ICLR*, 2014.

[LBB98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11):2278–2324, 1998.

[LS01] Daniel D Lee and H Sebastian Seung. "Algorithms for non-negative matrix factorization." In *NIPS*, pp. 556–562, 2001.

[LZW16] Yang Lu, Song-Chun Zhu, and Ying Nian Wu. "Learning FRAME Models Using CNN Filters." In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[MG14] Andriy Mnih and Karol Gregor. "Neural Variational Inference and Learning in Belief Networks." In *ICML*, 2014.

[MOS09] Josh H McDermott, Andrew J Oxenham, and Eero P Simoncelli. "Sound texture synthesis via filter statistics." In *Applications of Signal Processing to Audio and Acoustics, 2009. WASPAA'09. IEEE Workshop on*, pp. 297–300. IEEE, 2009.

[MS11] Josh H McDermott and Eero P Simoncelli. "Sound texture perception via statistics of the auditory periphery: evidence from sound synthesis." *Neuron*, **71**(5):926–940, 2011.

[OF97] Bruno A Olshausen and David J Field. "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vision Research*, **37**(23):3311–3325, 1997.

[Pic15] Karol J Piczak. "Environmental sound classification with convolutional neural networks." In *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*, pp. 1–6. IEEE, 2015.

[PMB13] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. "On the number of response regions of deep feed forward networks with piece-wise linear activations." *arXiv preprint arXiv:1312.6098*, 2013.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." *arXiv preprint arXiv:1511.06434*, 2015.

[RMW14] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." In Tony Jebara and Eric P. Xing, editors, *NIPS*, pp. 1278–1286. JMLR Workshop and Conference Proceedings, 2014.

[RT82] Donald B Rubin and Dorothy T Thayer. "EM algorithms for ML factor analysis." *Psychometrika*, **47**(1):69–76, 1982.

[VL15] A. Vedaldi and K. Lenc. "MatConvNet – Convolutional Neural Networks for MATLAB." In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[XLG16] Jianwen Xie, Yang Lu, Ruiqi Gao, Song-Chun Zhu, and Ying Nian Wu. "Cooperative Training of Descriptor and Generator Networks.", 2016.

[XLZ16] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. "A theory of generative ConvNet." In *ICML*, 2016.

[You99] Laurent Younes. "On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates." *Stochastics: An International Journal of Probability and Stochastic Processes*, **65**(3-4):177–228, 1999.