Cooperative Training of Descriptor and Generator Networks

Jianwen Xie, Yang Lu, Ruiqi Gao, Song-Chun Zhu, *Fellow, IEEE,* and Ying Nian Wu

Abstract—This paper studies the cooperative training of two generative models for image modeling and synthesis. Both models are parametrized by convolutional neural networks (ConvNets). The first model is a deep energy-based model, whose energy function is defined by a bottom-up ConvNet, which maps the observed image to the energy. We call it the descriptor network. The second model is a generator network, which is a non-linear version of factor analysis. It is defined by a top-down ConvNet, which maps the latent factors to the observed image. The maximum likelihood learning algorithms of both models involve MCMC sampling such as Langevin dynamics. We observe that the two learning algorithms can be seamlessly interwoven into a cooperative learning algorithm that can train both models simultaneously. Specifically, within each iteration of the cooperative learning algorithm, the generator model generates initial synthesized examples to initialize a finite-step MCMC that samples and trains the energy-based descriptor model. After that, the generator model learns from how the MCMC changes its synthesized examples. That is, the descriptor model teaches the generator model by MCMC, so that the generator model accumulates the MCMC transitions and reproduces them by direct ancestral sampling. We call this scheme MCMC teaching. We show that the cooperative algorithm can learn highly realistic generative models.

Index Terms—Deep generative models; Energy-based models; Latent variable models; Bottom-up and top-down convolutional neural networks; Modified contrastive divergence; MCMC teaching

1 Introduction

Learning generative models of images is a fundamental problem in computer vision and machine learning. In this article, we propose a cooperative learning algorithm to train two important classes of generative models jointly for image modeling, representation and synthesis.

1.1 Two generative models

We begin with an analogy. A student writes up an initial draft of a paper. Her advisor then revises it. After that they submit the revised paper for review. The student then learns from her advisor's revision, while the advisor learns from the outside review. In this analogy, the advisor guides the student, but the student does most of the work.

This paper is about two generative models, and they play the roles of teacher and student as mentioned above. Both models are parametrized by convolutional neural networks (ConvNets or CNNs) [1], [2]. They are of opposite directions. One is bottom-up, and the other is top-down:

Bottom-up ConvNet	Top-down ConvNet
energy	**latent variables**
$\Uparrow$	$\Downarrow$
image	image
(a) Descriptor Network	(b) Generator Network
(teacher)	(student)

(1)

These two nets correspond to two major classes of probabilistic models. (a) The energy-based models [3], [4] or the Markov random field models [5], [6], where the probability distribution is defined by the feature statistics or the energy function computed from the image by a bottom-up process. (b) The latent variable models or the directed graphical models, where the image is assumed to be a transformation of the latent factors that follow a known prior distribution. The latent factors generate the image by a top-down process via direct ancestral sampling [7].

The two classes of models have been contrasted by [8], [9], [10], [11], [12]. Both classes of models can benefit from the high capacity of the ConvNets. (a) In the energy-based model, the energy function can be defined by a bottom-up ConvNet that maps the image to the energy [12], [13], [14], [15], [16], and the energy function is usually the sum or a linear combination of the features at the top layer. For ease of reference, we call the resulting model a descriptor network following [8], because it is built on descriptive feature statistics. (b) In the latent variable model or the directed graphical model, the transformation from the latent factors to the image can be defined by a top-down ConvNet [17], [18], which maps the latent factors to the image. We call the resulting model a generator network following [19].

The likelihoods of both models involve intractable integrals, and the gradients of both log-likelihoods involve intractable expectations that can be approximated by MCMC . We notice that the maximum likelihood algorithms for learning the two models can be interwoven into a cooperative learning algorithm , where each iteration consists of the following two steps: (1) Modified contrastive divergence for energy -based descriptor model : The learning of the descriptor model is based on the contrastive divergence [4], but the finite -step MCMC sampling of the model is initialized from the synthesized examples generated by the generator model instead of being initialized from the observed examples. (2) MCMC teaching of the latent variable generator model : The learning of the generator model is based on how the MCMC in (1) changes the initial synthesized examples generated by the generator model . That is, the descriptor model

• *J. Xie is with Hikvision Research Institute, Santa Clara, USA. Y. Lu is with Facebook, Menlo Park, California. R. Gao, S.-C. Zhu, and Y. N. Wu are with the Department of Statistics, University of California, Los Angeles.*

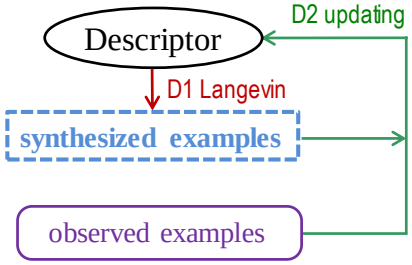


Fig. 1. The flow chart of Algorithm D for training the descriptor network. The updating in Step D2 is based on the difference between the observed examples and the synthesized examples. The Langevin sampling of the synthesized examples from the current model in Step D1 can be time consuming.

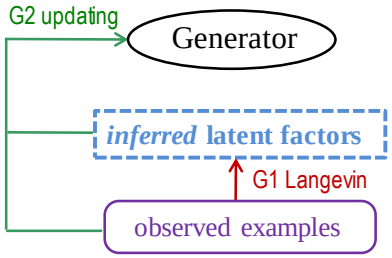


Fig. 2. The flow chart of Algorithm G for training the generator network. The updating in Step G2 is based on the observed examples and their inferred latent factors. The Langevin sampling of the latent factors from the current posterior distribution in Step G1 can be time consuming.

(teacher) distills its knowledge to the generator model (student) via MCMC, and we call it MCMC teaching. Our experiments show that the cooperative learning algorithm can learn realistic generative models of images.

1.2 Motivations and contributions

The main motivation for our work is that we find it very challenging to learn the two models separately, when the training images are highly varied. We find it much easier for the cooperative algorithm to learn highly realistic models from such data. Another motivation is to develop an alternative system to the generative adversarial networks (GAN) [19], [20], [21], where in our system both models are learned generatively. Our experiments suggest that the cooperative learning is stable and does not encounter mode collapsing issue.

The contributions of our work are as follows. We propose a cooperative learning algorithm to train the energy-based descriptor model and the latent variable generator model simultaneously, so that the learned models can synthesize highly realistic images. Our work connects the undirected model (descriptor) and the directed model (generator). It also connects the ancestral sampling (generator) and the MCMC sampling (descriptor).

In the following subsections, we shall further explain the basic idea of our paper, and review related work.

1.3 Two maximum likelihood algorithms

Both the energy-based model (the descriptor network) and the latent variable model (the generator network) can be learned from the training examples by maximum likelihood.

The training algorithm for the descriptor network alternates between the following two steps [15]. We call it Algorithm D. See Figure 1 for an illustration.

Step D1 for Langevin revision: Sampling synthesized examples from the current model by Langevin dynamics [22], [23], [24]. We call this step Langevin revision because it keeps revising the current synthesized examples.

Step D2 for density shifting: Updating the parameters of the descriptor network based on the difference between the observed examples and the synthesized examples obtained in D1 (Langevin revision). This step is to shift the high probability regions of the descriptor from the synthesized examples to the observed examples.

Both steps can be powered by back-propagation. The algorithm is thus an alternating back-propagation algorithm.

Intuitively, in Step D1 (Langevin revision), the descriptor network is dreaming by synthesizing examples from the current model. In Step D2 (density shifting), the descriptor updates its parameters to make the dream more realistic.

The training algorithm for the generator network alternates between the following two steps [25]. We call it Algorithm G. See Figure 2 for an illustration.

Step G1 for Langevin inference: For each observed example, sample the latent factors from the current posterior distribution by Langevin dynamics. We call this step Langevin inference because it infers the latent factors for each observed example, in order for the inferred latent factors to explain or reconstruct the observed examples.

Step G2 for reconstruction: Updating the parameters of the generator network based on the observed examples and their inferred latent factors obtained in G1 (Langevin inference), so that the inferred latent factors can better reconstruct the observed examples.

Again, both steps can be powered by back-propagation, and the algorithm is thus an alternating back-propagation algorithm.

The training algorithm for generator network is similar to the EM algorithm [26], where step G1 (Langevin inference) can be mapped to the E-step, and step G2 (reconstruction) can be mapped to the M-step. It is an unsupervised learning algorithm because the latent factors are unobserved.

Intuitively, in Step G1 (Langevin inference), the generator network is thinking about each observed example by inferring the latent factors that can reconstruct it. The thinking involves explaining-away reasoning: the latent factors compete with each other in the Langevin inference process to explain the example. In Step G2 (reconstruction), the generator network updates its parameters to make the thinking more accurate.

Compared to Step D2 (density shifting), Step G2 (reconstruction) is also a form of density shifting from the current reconstructed examples towards the observed examples, but it requires inferring the latent factors for each observed example.

1.4 Cooperative training via MCMC teaching

The two algorithms can operate separately on their own [15], [25]. But just like the case with the student and the advisor, they benefit from cooperating with each other, where the generator network plays the role of the student, and the descriptor network plays the role of the teacher.

The needs for cooperation stem from the fact that the Langevin sampling in Step D1 (Langevin revision) and Step G1 (Langevin

inference) can be time consuming. If the two algorithms cooperate, they can jumpstart each other's Langevin sampling in D1 (Langevin revision) and G1 (Langevin inference). The resulting algorithm seamlessly interweaves the steps in the two algorithms with minimal modifications.

Intuitively, while the descriptor needs to dream hard in Step D1 (Langevin revision) for synthesis, the generator needs to think hard in Step G1 (Langevin inference) for explaining-away reasoning. On the other hand, the generator is actually a much better dreamer because it can generate images by direct ancestral sampling without MCMC, while the descriptor does not need to think in order to learn.

cooperative learning algorithm:

Step G0 for initial generation: Generate the initial synthesized examples using the generator network. These initial examples can be obtained by direct ancestral sampling via a top-down process.

Step D1 for Langevin revision: Starting from the initial synthesized examples produced in Step G0 (initial generation), run Langevin revision dynamics for a finite number of steps to obtain the revised synthesized examples.

Step D2 for density shifting: The same as before, except that we use the revised synthesized examples produced by Step D1 (Langevin revision) to shift the density of the descriptor towards the observed examples.

Step G1 for Langevin inference: The generator network can learn from the revised synthesized examples produced by Step D1 (Langevin revision). For each revised synthesized example, we know the values of the latent factors that generate the corresponding initial synthesized example in Step G0 (initial generation), therefore we may simply infer the latent factors to be their known values given in Step G0 (initial generation), or initialize the Langevin inference dynamics in Step G1 (Langevin inference) from the known values.

Step G2 for reconstruction: The same as before, except that we use the revised synthesized examples and the inferred latent factors obtained in Step G1 (Langevin inference) to update the generator. The generator in Step G0 (initial generation) generates and thus reconstructs the initial synthesized examples. Step G2 (reconstruction) updates the generator to reconstruct the revised synthesized examples. The revision of the generator accounts for the revisions made by the Langevin revision dynamics in Step D1 (Langevin revision).

Figure 3 shows the flow chart of the cooperative algorithm. The generator is like the student. It generates the initial draft of the synthesized examples. The descriptor is like the teacher. It revises the initial draft by running a number of Langevin revisions. The descriptor learns from the outside review, which is in the form of the difference between the observed examples and the revised synthesized examples. This is a modified version of contrastive divergence, where the MCMC is initialized by the generator instead of being initialized from the observed examples. The generator learns from how the descriptor's MCMC revises the initial draft by reconstructing the revised draft. This is MCMC teaching.

The reason we let the generator network learn from the revised synthesized examples instead of the observed examples is that the generator does not know the latent factors that generate the observed examples, and it has to think hard to infer them by explaining-away reasoning. However, the generator knows the values of the latent factors that generate each initial synthesized example, and thus it essentially knows the latent factors when learning from the revised synthesized examples. By reconstructing the revised synthesized examples, the generator traces and accumulates the Langevin revisions made by the descriptor. This cooperation is thus beneficial to the generator by relieving it the burden of inferring the latent factors.

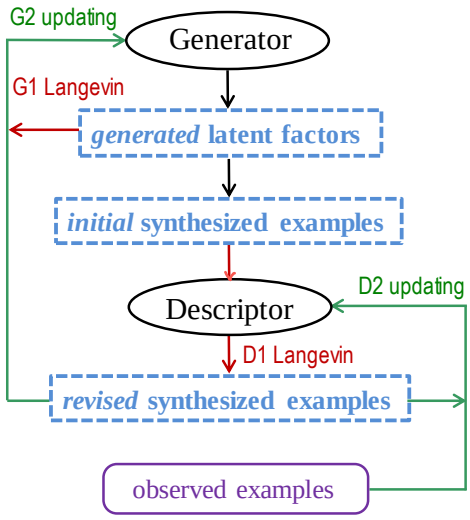


Fig. 3. The flow chart of the cooperative algorithm. The part of the flow chart for training the descriptor is similar to Algorithm D in Figure 1, except that the D1 Langevin sampling is initialized from the initial synthesized examples supplied by the generator. The part of the flow chart for training the generator can also be mapped to Algorithm G in Figure 2, 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).

While the generator may find it hard to learn from the observed examples directly, the descriptor has no problem learning from the observed examples because it only needs to compute the bottom-up features deterministically. However, it needs synthesized examples to find its way to shift its density, and they do not come by easily. The generator can provide unlimited number of examples, and in each learning iteration, the generator supplies a completely new batch of independent examples on demand. The descriptor only needs to revise the new batch of examples instead of generating a new batch by itself from scratch. The generator has memorized the cumulative effect of all the past Langevin revisions, so that it can produce new samples in one shot. This cooperation is thus beneficial to the descriptor by relieving it the burden of synthesizing examples from scratch.

In terms of density shifting, the descriptor shifts its density from the revised synthesized examples towards the observed examples in Step D2, and the generator shifts its density from the initial synthesized examples towards the revised synthesized examples in Step G2 by reconstructing the latter.

In terms of energy function, the descriptor shifts its low energy regions towards the observed examples, and it induces the generator to map the latent factors to its low energy regions. It achieves that by stochastically relaxing the synthesized examples towards low energy regions, and let the generator track the synthesized examples.

1.5 Related work

Our work is related to contrastive divergence [4] for training the energy-based model, such as deep Boltzmann machine [27] and deep belief network [28]. The contrastive divergence initializes the finite-step MCMC sampling from the observed examples. Our method initializes the MCMC sampling from the generator that seeks to approximate the descriptor, so that the learning is closer to maximum likelihood.

Our work is similar to the recent work of [29]. In [29], the generator learns from the energy-based model by minimizing the Kullback-Leibler divergence from the generator model to the energy-based model, which can be decomposed into an energy term and an entropy term. In our work, the energy-based descriptor model teaches the generator via MCMC teaching. Our method does not need to approximate the intractable entropy term.

Another method for training the generator network is variational auto-encoder (VAE) [30], [31], [32], which learns an inferential or recognition network. The MCMC teaching in our work avoids the challenging problem of inferring the latent variables from the observed examples.

Our work is related to knowledge distilling [33]. In our work, the descriptor distills its MCMC algorithm to the generator through MCMC teaching.

Building on the pioneering work of [34], recently [35], [36], [37] have developed an introspective learning method to learn the energy-based model, where the energy function is discriminatively learned, and the learned energy function is used to generate synthesized examples via Langevin dynamics. It can be interesting to combine introspective learning with the proposed cooperative learning method that recruits a generator to jumpstart the Langevin sampling.

Another recently proposed model is pixelCNN [38], which learns an autoregressive model. Unlike pixelCNN, the descriptor and generator networks model the joint distribution of the image pixels directly without factorizing it into a sequence of conditional distributions.

This paper is an expansion of our conference paper [39]. It is also related to our recent papers on 3D descriptor model [40] and the spatial-temporal descriptor model [41].

2 Two Models and Two Algorithms

(p_θ, q_α) notation. Let Y be the D-dimensional signal, such as an image. We use $p(Y;\theta)$ or p_θ to denote the probability distribution of the descriptor network (energy-based model), where θ denotes the parameters of the bottom-up ConvNet. We use $q(Y;\alpha)$ or q_α to denote the probability distribution of the generator network (latent variable model), where α denotes the parameters of the top-down ConvNet.

2.1 Energy-based model and maximum likelihood learning

As an energy-based model, the descriptor network is in the form of exponential tilting of a reference distribution [15]:

$$p(Y;\theta) = \frac{1}{Z(\theta)} \exp\left[f(Y;\theta)\right] p_0(Y). \tag{2}$$

$p_0(Y)$ is the reference distribution such as Gaussian white noise

$$p_0(Y) = \frac{1}{(2\pi s^2)^{D/2}} \exp\left[-\frac{\|Y\|^2}{2s^2}\right], \tag{3}$$

where D is the dimensionality of the image Y, and $Y \sim \mathrm{N}(0, s^2 I_D)$ under $p_0(Y)$ (I_D denotes the D-dimensional identity matrix). $f(Y;\theta)$ is a ConvNet whose parameters are denoted by θ. This ConvNet is bottom-up because it maps the image Y to the energy. See the diagram in (1). $Z(\theta) = \int \exp\left[f(Y;\theta)\right] p_0(Y) dY$ is the normalizing constant, and this integral is analytically intractable. The energy function of the model is

$$\mathcal{E}(Y;\theta) = \frac{1}{2s^2}\|Y\|^2 - f(Y;\theta). \tag{4}$$

p_0 can also be taken to be the uniform measure, and in that case, $\mathcal{E}(Y;\theta) = -f(Y;\theta)$.

Suppose we observe training examples $\{Y_i, i = 1, ..., n\}$ from an unknown data distribution $P_{\mathrm{data}}(Y)$. The maximum likelihood learning maximizes the log-likelihood function

$$L_p(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p(Y_i;\theta). \tag{5}$$

If the sample size n is large, the maximum likelihood estimator minimizes $\mathrm{KL}(P_{\mathrm{data}}|p_\theta)$, the Kullback-Leibler divergence from the data distribution P_{data} to the model distribution p_θ. The log-likelihood is analytically intractable because of the intractable integral $Z(\theta)$.

The gradient of the log-likelihood $L_p(\theta)$ is

$$L'_p(\theta) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial\theta} f(Y_i;\theta) - \mathrm{E}_\theta\left[\frac{\partial}{\partial\theta} f(Y;\theta)\right], \tag{6}$$

where E_θ denotes the expectation with respect to $p(Y;\theta)$. Equation (6) follows from $\frac{\partial}{\partial\theta}\log Z(\theta) = \mathrm{E}_\theta[\frac{\partial}{\partial\theta} f(Y;\theta)]$.

The expectation in equation (6) is analytically intractable and has to be approximated by MCMC, such as Langevin revision dynamics, which iterates the following steps:

$$\begin{aligned} Y_{\tau+1} &= Y_\tau - \frac{\delta^2}{2}\frac{\partial}{\partial Y}\mathcal{E}(Y_\tau;\theta) + \delta U_\tau \\ &= Y_\tau - \frac{\delta^2}{2}\left[\frac{Y_\tau}{s^2} - \frac{\partial}{\partial Y} f(Y_\tau;\theta)\right] + \delta U_\tau, \end{aligned} \tag{7}$$

where τ indexes the time steps of Langevin dynamics, δ is the step size, and $U_\tau \sim \mathrm{N}(0, I_D)$ is the Gaussian white noise term. The Langevin dynamics is a process of stochastic relaxation. The gradient term seeks to reduce the energy function $\mathcal{E}(Y;\theta)$ while the noise term provides the randomness to increase the entropy. A Metropolis-Hastings step can be added to correct for the finite step size.

We can run $\tilde{n}$ parallel chains of Langevin dynamics according to (7) to obtain the synthesized examples $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$. The Monte Carlo approximation to $L'_p(\theta)$ is

$$\begin{aligned} L'_p(\theta) &\approx \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial\theta} f(Y_i;\theta) - \frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}} \frac{\partial}{\partial\theta} f(\tilde{Y}_i;\theta) \\ &= \frac{\partial}{\partial\theta}\left[\frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}} \mathcal{E}(\tilde{Y}_i;\theta) - \frac{1}{n}\sum_{i=1}^{n} \mathcal{E}(Y_i;\theta)\right], \end{aligned} \tag{8}$$

which is the difference between the observed examples and the synthesized examples. See [14] for details.

Algorithm 1 [15] describes the training algorithm. See Figure 1 for an illustration. Step D1 (Langevin revision) tends to settle the synthesized examples at the low energy (high density) regions. Step D2 (density shifting) tends to shift low energy (high density) regions from the synthesized examples to the observed examples.

Algorithm 1 Algorithm D

Input:

(1) training examples $\{Y_i, i = 1, ..., n\}$

(2) number of Langevin steps l_p

(3) number of learning iterations T

Output:

(1) estimated parameters θ

(2) synthesized examples $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$

1: Let $t \leftarrow 0$, initialize θ.
2: Initialize $\tilde{Y}_i$, $i = 1, ..., \tilde{n}$.
3: **repeat**
4: **Step D1 Langevin revision**: For each i, run l_p steps of Langevin dynamics to update $\tilde{Y}_i$, i.e., starting from the current $\tilde{Y}_i$, each step follows equation (7).
5: **Step D2 density shifting**: Update $\theta^{(t+1)} = \theta^{(t)} + \gamma_t L'_p(\theta^{(t)})$, with learning rate γ_t, where $L'_p(\theta^{(t)})$ is computed according to (8).
6: Let $t \leftarrow t + 1$
7: **until** $t = T$

Step D1 needs to compute $\frac{\partial}{\partial Y} f(Y; \theta)$. Step D2 needs to compute $\frac{\partial}{\partial \theta} f(Y; \theta)$. The computations of both derivatives can be powered by back-propagation. Because of the ConvNet structure of $f(Y; \theta)$, the computations of the two derivatives share most of their steps in the chain rule computations.

Because the parameter θ keeps changing in the learning process, the energy landscape and the local energy minima also keep changing. This may help the Langevin revision dynamics avoid being trapped by the local energy minima.

Algorithm D is a stochastic approximation algorithm [42], except that the synthesized examples are obtained by a finite number of Langevin steps in each learning iteration. The convergence of an algorithm of this type to the maximum likelihood estimate has been studied by [43].

Contrastive divergence. If we initialize the synthesized examples $\{\tilde{Y}_i\}$ from the observed examples $\{Y_i\}$, the learning algorithm becomes persistent contrastive divergence [44].

Zero temperature limit. We can write the model $p(Y; \theta) = \exp(-\mathcal{E}(Y; \theta)/T)/Z_T$ where T is the temperature term ($T = 1$ in our model). At the zero-temperature limit $T \to 0$, we can disable the noise term in the Langevin dynamics, so that Step D1 (Langevin revision) becomes gradient descent towards the local minima of the energy function. Define the value function

$$V(\theta, \tilde{Y}_i, i = 1, ..., \tilde{n}) = \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \mathcal{E}(\tilde{Y}_i; \theta) - \frac{1}{n} \sum_{i=1}^{n} \mathcal{E}(Y_i; \theta), \quad (9)$$

then the learning algorithm solves the following minimax game

$$\max_{\theta} \min_{\{\tilde{Y}_i\}} V(\theta, \tilde{Y}_i, i = 1, ..., \tilde{n}), \quad (10)$$

where Step D1 (Langevin revision) seeks to decrease V, while Step D2 (density shifting) seeks to increase V.

The energy-based model can be used for inverse optimal control [45], [46], where the Langevin dynamics or the gradient descent algorithm in Step D1 can be considered an optimal control algorithm, and the energy function can be considered the cost function or critic, which is updated in Step D2.

The energy-based model is also related to the Hopfield model for content addressable memory [47]. The Langevin dynamics in Step D1 can be considered an attractor dynamics towards the local modes for memory recall [48], while Step D2 shifts the local modes towards the observed examples.

2.2 Latent variable model and maximum likelihood learning

As a latent variable model, the generator network seeks to explain the image Y of dimension D by a vector of latent factors X of dimension d, and usually $d \ll D$. The model is of the following form:

$$\begin{aligned} &X \sim \mathrm{N}(0, I_d), \\ &Y = g(X; \alpha) + \epsilon, \ \epsilon \sim \mathrm{N}(0, \sigma^2 I_D). \end{aligned} \quad (11)$$

$g(X; \alpha)$ is a top-down ConvNet defined by the parameters α. The ConvNet g maps the latent factors X to the image Y. See the diagram in (1). Model (11) is a directed graphical model, where Y can be readily generated by ancestral sampling: first sampling X from its known prior distribution $\mathrm{N}(0, I_d)$ and then transforming X to Y via g.

The joint density is $q(X, Y; \alpha) = q(X)q(Y|X; \alpha)$, and

$$\begin{aligned} \log q(X, Y; \alpha) &= -\frac{1}{2\sigma^2} \|Y - g(X; \alpha)\|^2 \\ &- \frac{1}{2} \|X\|^2 + \text{constant}, \end{aligned} \quad (12)$$

where the constant term is independent of X, Y and α. The marginal density is obtained by integrating out the latent factors X, i.e., $q(Y; \alpha) = \int q(X, Y; \alpha) dX$. This integral is analytically intractable. The inference of X given Y is based on the posterior density $q(X|Y; \alpha) = q(X, Y; \alpha)/q(Y; \alpha) \propto q(X, Y; \alpha)$ as a function of X.

For the training data $\{Y_i, i = 1, ..., n\}$, the generator network can be trained by maximizing the log-likelihood

$$L_q(\alpha) = \frac{1}{n} \sum_{i=1}^{n} \log q(Y_i; \alpha). \quad (13)$$

The log-likelihood is intractable because the marginal distribution $q(Y; \alpha)$ is an intractable integral. If the sample size n is large, the maximum likelihood estimator minimizes the Kullback-Leibler divergence $\mathrm{KL}(P_{\text{data}}|q_\alpha)$ from the data distribution P_{data} to the model distribution q_α.

The gradient of $L_q(\alpha)$ is obtained according to the following identity

$$\begin{aligned} \frac{\partial}{\partial \alpha} \log q(Y; \alpha) &= \frac{1}{q(Y; \alpha)} \frac{\partial}{\partial \alpha} \int q(Y, X; \alpha) dX \\ &= \int \left[\frac{\partial}{\partial \alpha} \log q(Y, X; \alpha) \right] \frac{q(Y, X; \alpha)}{q(Y; \alpha)} dX \\ &= \mathrm{E}_{q(X|Y;\alpha)} \left[\frac{\partial}{\partial \alpha} \log q(X, Y; \alpha) \right]. \end{aligned} \quad (14)$$

The above identity underlies the EM algorithm, where $\mathrm{E}_{q(X|Y;\alpha)}$ is the expectation with respect to the posterior distribution of the latent factors $q(X|Y; \alpha)$, and is computed in the E-step. The usefulness of identify (14) lies in the fact that the derivative of the complete-data log-likelihood $\log q(X, Y; \alpha)$ on the right hand side can be obtained in closed form.

In general, the expectation in (14) is analytically intractable even though the term inside the expectation can be easily computed, and the expectation has to be approximated by MCMC

that samples from the posterior $q(X|Y;\alpha)$, such as the Langevin inference dynamics, which iterates

$$X_{\tau+1} = X_\tau + \frac{\delta^2}{2}\frac{\partial}{\partial X}\log q(X_\tau, Y;\alpha) + \delta U_\tau, \tag{15}$$

where τ indexes the time step, δ is the step size, and for notational simplicity, we continue to use U_τ to denote the noise term, but here $U_\tau \sim \mathrm{N}(0, I_d)$. We take the derivative of $\log q(X, Y;\alpha)$ in (15) because this derivative is the same as the derivative of the log-posterior $\log q(X|Y;\alpha)$, since $q(X|Y;\alpha)$ is proportional to $q(X, Y;\alpha)$ as a function of X. The Langevin inference solves a ℓ_2 penalized non-linear least squares problem so that X_i can reconstruct Y_i given the current α. The Langevin inference process performs explaining-away reasoning, where the latent factors in X compete with each other to explain the current residual $Y - g(X;\alpha)$.

With X_i sampled from $q(X_i \mid Y_i;\alpha)$ for each observation Y_i by the Langevin inference process, the Monte Carlo approximation to $L_q'(\alpha)$ is

$$\begin{aligned} L_q'(\alpha) &\approx \frac{1}{n}\sum_{i=1}^n \frac{\partial}{\partial\alpha}\log q(X_i, Y_i;\alpha) \\ &= \frac{1}{n}\sum_{i=1}^n \frac{1}{\sigma^2}(Y_i - g(X_i;\alpha))\frac{\partial}{\partial\alpha}g(X_i;\alpha). \end{aligned} \tag{16}$$

The updating of α solves a non-linear regression problem, so that the learned α enables better reconstruction of Y_i by the inferred X_i. Given the inferred X_i, the learning of α is a supervised learning problem [18].

Algorithm 2 Algorithm G

Input:

(1) training examples $\{Y_i, i = 1, ..., n\}$
(2) number of Langevin steps l_q
(3) number of learning iterations T

Output:

(1) estimated parameters α
(2) inferred latent factors $\{X_i, i = 1, ..., n\}$

1: Let $t \leftarrow 0$, initialize α.
2: Initialize X_i, $i = 1, ..., n$.
3: **repeat**
4: **Step G1 Langevin inference**: For each i, run l_q steps of Langevin dynamics to update X_i, i.e., starting from the current X_i, each step follows equation (15).
5: **Step G2 reconstruction**: Update $\alpha^{(t+1)} = \alpha^{(t)} + \gamma_t {L_q}'(\alpha^{(t)})$, with learning rate γ_t, where ${L_q}'(\alpha^{(t)})$ is computed according to equation (16).
6: Let $t \leftarrow t + 1$
7: **until** $t = T$

Algorithm 2 [25] describes the training algorithm. See Figure 2 for an illustration. Step G1 (Langevin inference) needs to compute $\frac{\partial}{\partial X}g(X;\alpha)$. Step G2 (reconstruction) needs to compute $\frac{\partial}{\partial\alpha}g(X;\alpha)$. The computations of both derivatives can be powered by back-propagation, and the computations of the two derivatives share most of their steps in the chain rule computations.

Algorithm G is a stochastic approximation or stochastic gradient algorithm that converges to the maximum likelihood estimate [43].

The generator network can be considered a generalization of the factor analysis model. In factor analysis, the mapping from the latent factors to the signal is linear. The mapping becomes non-linear in generator network.

As shown in [25], the generator network can be learned directly from the incomplete data. It can also be used for pattern completion in the testing data, which may be considered as an alternative approach to content addressable memory [47].

3 Cooperative Training

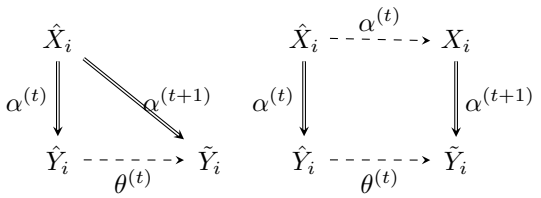
3.1 Cooperation of two algorithms

In Algorithm D and Algorithm G, both steps D1 (Langevin revision) and G1 (Langevin inference) are Langevin dynamics. They may be slow to converge and may become bottlenecks in their respective algorithms. An interesting observation is that the two algorithms can cooperate with each other by jumpstarting each other's Langevin sampling.

Specifically, in Step D1 (Langevin revision), we can initialize the synthesized examples by generating examples from the generator network, which does not require MCMC, because the generator network is a directed graphical model. More specifically, we first generate $\hat{X}_i \sim \mathrm{N}(0, I_d)$, and then generate $\hat{Y}_i = g(\hat{X}_i;\alpha) + \epsilon_i$, for $i = 1, ..., \tilde{n}$. If the current generator q is close to the current descriptor p, then the generated $\{\hat{Y}_i\}$ should be a good initialization for sampling from the descriptor network, i.e., starting from the $\{\hat{Y}_i, i = 1, ..., \tilde{n}\}$ supplied by the generator network, we run Langevin dynamics in Step D1 (Langevin revision) for l_p 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 network. We can then update θ according to Step D2 (density shifting) of Algorithm D. This is modified contrastive divergence.

In order to update α of the generator network, we treat the $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$ produced by the above D1 (Langevin revision) step as the training data for the generator. Since these $\{\tilde{Y}_i\}$ are obtained by the Langevin revision dynamics initialized from the $\{\hat{Y}_i, i = 1, ..., \tilde{n}\}$ produced by the generator network with known latent factors $\{\hat{X}_i, i = 1, ..., \tilde{n}\}$, we can update α by learning from $\{(\tilde{Y}_i, \hat{X}_i), i = 1, ..., \tilde{n}\}$, which is a supervised learning problem, or more specifically, a non-linear regression of $\tilde{Y}_i$ on $\hat{X}_i$. At $\alpha^{(t)}$, the latent factors $\hat{X}_i$ generates and thus reconstructs the initial example $\hat{Y}_i$. After updating α, we want $\hat{X}_i$ to reconstruct the revised example $\tilde{Y}_i$. That is, we revise α to absorb the revision from $\hat{Y}_i$ to $\tilde{Y}_i$, so that the generator shifts its density from $\{\hat{Y}_i\}$ to $\{\tilde{Y}_i\}$. This is MCMC teaching. The reconstruction error can tell us whether the generator has caught up with the descriptor by fully absorbing the revision.

The diagrams in (17) illustrate the basic idea of MCMC teaching:



(17)

In the two diagrams in (17), the double line arrows indicate generation and reconstruction in the generator network, while the dashed line arrows indicate Langevin dynamics for revision and inference in the two nets. The diagram on the right in (17) illustrates a

more rigorous method, where we initialize the Langevin inference of $\{X_i, i = 1, ..., \tilde{n}\}$ in Step G1 (Langevin inference) from $\{\hat{X}_i\}$, and then update α in Step G2 (reconstruction) 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 MCMC.

Algorithm 3 CoopNets Algorithm

Input:

(1) training examples $\{Y_i, i = 1, ..., n\}$

(2) numbers of Langevin steps l_p and l_q

(3) number of learning iterations T

Output:

(1) estimated parameters θ and α

(2) synthesized examples $\{\hat{Y}_i, \tilde{Y}_i, i = 1, ..., \tilde{n}\}$

1: Let $t \leftarrow 0$, initialize θ and α.

2: **repeat**

3: **Step G0 Initial generation**: For $i = 1, ..., \tilde{n}$, generate $\hat{X}_i \sim \mathrm{N}(0, I_d)$, and generate $\hat{Y}_i = g(\hat{X}_i; \alpha^{(t)}) + \epsilon_i$.

4: **Step D1 Langevin revision**: For $i = 1, ..., \tilde{n}$, starting from $\hat{Y}_i$, run l_p steps of Langevin revision dynamics to obtain $\tilde{Y}_i$, each step following equation (7).

5: **Step G1 Langevin inference**: Treat the current $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$ as the training data, for each i, infer $X_i = \hat{X}_i$. Or more rigorously, starting from $X_i = \hat{X}_i$, run l_q steps of Langevin inference dynamics to update X_i, each step following equation (15).

6: **Step D2 Density shifting**: Update $\theta^{(t+1)} = \theta^{(t)} + \gamma_t L_p'(\theta^{(t)})$, where $L_p'(\theta^{(t)})$ is computed according to (8).

7: **Step G2 Reconstruction**: Update $\alpha^{(t+1)} = \alpha^{(t)} + \gamma_t {L_q}'(\alpha^{(t)})$, where ${L_q}'(\alpha^{(t)})$ is computed according to equation (16), except that Y_i is replaced by $\tilde{Y}_i$, and n by $\tilde{n}$. We can run multiple iterations of Step G2 to learn from and reconstruct $\{\tilde{Y}_i\}$, and to allow the generator to catch up with the descriptor.

8: Let $t \leftarrow t + 1$

9: **until** $t = T$

Algorithm 3 describes the cooperative training that interweaves Algorithm D and Algorithm G. For ease of reference, we call the algorithm the CoopNets algorithm. See Figure 3 for the flow chart of the CoopNets algorithm.

The following are the special cases of the above algorithm.

Special case (1): In Step G1, let $l_q = 0$, i.e., disable the Langevin inference process, i.e., the left diagram of (17). We use this algorithm in our experiments and it works well.

Special case (2): In addition to (1), in Step D1, let $l_p = 1$, i.e., we only run one step of Langevin revision. This algorithm is very efficient, and works well according to our experience.

The learning of both the descriptor and the generator follows the "analysis by synthesis" principle [49]. There are three sets of synthesized examples (S stands for synthesis). Data (S1): Initial synthesized examples $\{\hat{Y}_i\}$ generated by Step G0 (initial generation). Data (S2): Revised synthesized examples $\{\tilde{Y}_i\}$ produced by Step D1 (Langevin revision). Data (S3): Reconstructed synthesized examples $\{g(X_i; \alpha^{(t+1)})\}$ produced by Step G2 (reconstruction). The descriptor shifts its density from (S2) towards the observed data, while the generator shifts its density from (S1) towards (S2).

The evolution from (S1) to (S2) is the work of the descriptor. It is a process of stochastic relaxation that settles the synthesized examples in the low energy regions of the descriptor. The descriptor works as an associative memory, with (S1) being the cue, and (S2) being the recalled memory. It serves as a feedback to the generator. The reconstruction of (S2) by (S3) is the work of the generator that seeks to absorb the feedback conveyed by the evolution from (S1) to (S2). The descriptor can test whether the generator learns well by checking whether (S3) is close to (S2). The two nets collaborate and communicate with each other via synthesized data.

The general idea of the interaction between MCMC and the generator can be illustrated by the following diagram,

$$\begin{array}{lccc} \text{MCMC}: & P^{(t)} & \xrightarrow{\text{Markov transition}} & P^{(t+1)} \\ & \Updownarrow & & \Updownarrow \\ \text{Generator}: & \alpha^{(t)} & \xrightarrow{\text{Parameter updating}} & \alpha^{(t+1)} \end{array} \tag{18}$$

where $P^{(t)}$ is the marginal distribution of MCMC and $\alpha^{(t)}$ is the parameter of the generator which traces the evolution of the marginal distribution in MCMC by absorbing the cumulative effect of all the past Markov transitions.

In traditional MCMC, we only have access to Monte Carlo samples, instead of their marginal distributions, which exist only theoretically but are analytically intractable. However, with the generator net, we can actually implement MCMC at the level of the whole distributions, instead of a number of Monte Carlo samples, in the sense that after learning $\alpha^{(t)}$ from the existing samples of $P^{(t)}$, we can replace the existing samples by fresh new samples to rejuvenate the Markov chains, by sampling from the generator defined by the learned $\alpha^{(t)}$, which is illustrated by the two-way arrow between $P^{(t)}$ and $\alpha^{(t)}$. Effectively, the generator powers MCMC by implicitly running an infinite number of parallel chains. Conversely, the MCMC does not only drive the evolution of the samples, it also drives the evolution of a generator model.

3.2 Theoretical understanding

In the CoopNets algorithm, Steps G0 (initial generation), D1 (Langevin revision), and D2 (density shifting) are modified contrastive divergence, and Steps G0 (initial generation), G1 (Langevin inference), and G2 (reconstruction) are MCMC teaching.

(1) Modified contrastive divergence for descriptor (energy-based model). In the traditional contrastive divergence [4], $\hat{Y}_i$ in Step D1 (Langevin revision) is taken to be the observed Y_i. In cooperative learning, $\hat{Y}_i$ is generated by $q(Y; \alpha^{(t)})$. Let $\mathcal{M}_\theta$ be the Markov transition kernel of l_p steps of Langevin dynamics that samples p_θ. For any distribution p and any Markov transition kernel $\mathcal{M}$, let $\mathcal{M}p$ be the marginal distribution obtained by running the Markov transition $\mathcal{M}$ from p. Then similar to the traditional contrastive divergence, the learning gradient of the descriptor θ at iteration t is the gradient of

$$\mathrm{KL}(P_{\text{data}}|p_\theta) - \mathrm{KL}(\mathcal{M}_{\theta^{(t)}} q_{\alpha^{(t)}}|p_\theta) \tag{19}$$

with respect to θ. In the traditional contrastive divergence, P_{data} takes the place of $q_{\alpha^{(t)}}$ in the second KL-divergence.

(2) MCMC teaching of the generator model (latent variable model). The learning gradient of the generator α in the right diagram of (17) is the gradient of

$$\mathrm{KL}(\mathcal{M}_{\theta^{(t)}} q_{\alpha^{(t)}}|q_\alpha) \tag{20}$$

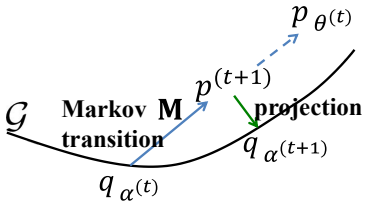


Fig. 4. The MCMC teaching of the generator alternates between Markov transition and projection. The family of the generator models $\mathcal{G}$ is illustrated by the black curve. Each distribution is illustrated by a point.

with respect to α. Here $p^{(t+1)} = \mathcal{M}_{\theta^{(t)}} q_{\alpha^{(t)}}$ takes the place of P_{data} as the data to train the generator model. It is much easier to minimize $\text{KL}(\mathcal{M}_{\theta^{(t)}} q_{\alpha^{(t)}} | q_\alpha)$ than minimizing $\text{KL}(P_{\text{data}}|q_\alpha)$ because the latent variables are essentially known in the former, so that the learning is supervised. The MCMC teaching alternates between Markov transition from $q_{\alpha^{(t)}}$ to $p^{(t+1)}$, and projection from $p^{(t+1)}$ to $q_{\alpha^{(t+1)}}$, as illustrated by Figure 4.

Assume the learning algorithm converges to a fixed point $(\hat{\theta}, \hat{\alpha})$, then

$$\hat{\theta} = \arg\min_\theta \left[\text{KL}(P_{\text{data}}|p_\theta) - \text{KL}(\mathcal{M}_{\hat{\theta}} q_{\hat{\alpha}} | p_\theta)\right], \quad (21)$$

$$\hat{\alpha} = \arg\min_\alpha \text{KL}(\mathcal{M}_{\hat{\theta}} q_{\hat{\alpha}} | q_\alpha). \quad (22)$$

(assuming $\hat{\theta}$ and $\hat{\alpha}$ are local minima). Equation (22) tells us that $q_{\hat{\alpha}}$ seeks to be the stationary distribution of $\mathcal{M}_{\hat{\theta}}$, and the stationary distribution is nothing but $p_{\hat{\theta}}$. In the idealized scenario where the generator q_α has infinite capacity, so that $\min_\alpha \text{KL}(\mathcal{M}_{\hat{\theta}} q_{\hat{\alpha}} | q_\alpha) = 0$, then $q_{\hat{\alpha}} = \mathcal{M}_{\hat{\theta}} q_{\hat{\alpha}}$, so that $q_{\hat{\alpha}}$ is the stationary distribution of $\mathcal{M}_{\hat{\theta}}$, which is $p_{\hat{\theta}}$, thus $q_{\hat{\alpha}} = p_{\hat{\theta}}$. As a consequence, the second divergence in (21) vanishes, i.e., $\text{KL}(\mathcal{M}_{\hat{\theta}} q_{\hat{\alpha}} | p_{\hat{\theta}}) = 0$, so that $\hat{\theta}$ becomes maximum likelihood estimate that minimizes the first KL-divergence $\text{KL}(P_{\text{data}}|p_\theta)$.

To further understand the dynamics of the MCMC teaching in this idealized scenario, suppose the descriptor p_θ is fixed, and it teaches the generator q_α by the MCMC teaching such that $\alpha^{(t+1)} = \arg\min_\alpha \text{KL}(\mathcal{M}_\theta q_{\alpha^{(t)}} | q_\alpha)$, then $q_{\alpha^{(t+1)}} = \mathcal{M}_\theta q_{\alpha^{(t)}}$, so that $q_{\alpha^{(t)}} = \mathcal{M}_\theta^t q_{\alpha^{(0)}} \to p_\theta$, i.e., q_α accumulates the MCMC transitions and convergences to the stationary distribution p_θ.

As is the case with the traditional contrastive divergence, the analysis of the finite capacity situation can be rather involved. We leave it to future investigation, while relying on empirical evaluations in this paper.

[29] learned the generator model by gradient descent on $\text{KL}(q_\alpha|p_{\theta^{(t)}})$ over α. In fact their learning objective is

$$\min_\theta \max_\alpha [\text{KL}(P_{\text{data}}|p_\theta) - \text{KL}(q_\alpha|p_\theta)]. \quad (23)$$

The objective function for α is

$$\text{KL}(q_\alpha|p_{\theta^{(t)}}) = \text{E}_{q_\alpha}[\log q(Y;\alpha)] - \text{E}_{q_\alpha}[\log p(Y;\theta^{(t)})], \quad (24)$$

where the first term is the negative entropy that is intractable, and the second term is the expected energy that is tractable. Our MCMC teaching of the generator is consistent with the learning objective $\text{KL}(q_\alpha|p_{\theta^{(t)}})$, because

$$\text{KL}(p^{(t+1)}|p_{\theta^{(t)}}) \le \text{KL}(q_{\alpha^{(t)}}|p_{\theta^{(t)}}). \quad (25)$$

In fact, $\text{KL}(p^{(t+1)}|p_{\theta^{(t)}}) \to 0$ monotonically as $l_p \to \infty$ due to the second law of thermodynamics [50]. The MCMC teaching of the generator in the cooperative learning algorithm alternates between Markov transition from $q_{\alpha^{(t)}}$ to $p^{(t+1)}$, and projection from $p^{(t+1)}$ to $q_{\alpha^{(t+1)}}$, as illustrated by Figure 4. The reduction of the Kullback-Leibler divergence in (25) and the projection in the MCMC teaching are consistent with the learning objective of reducing $\text{KL}(q_\alpha|p_{\theta^{(t)}})$ in [29]. But the Monte Carlo implementation of $\mathcal{M}$ in our work avoids the need to approximate the intractable entropy term. As to the updating of the descriptor model, [29] uses the initial synthesized examples generated by the generator model, while our method uses the revised synthesized examples obtained by finite-step MCMC towards the descriptor model, which are closer to the fair samples from the descriptor model.

Special case (2) as noise-injected back-propagation. The special case (2) with one step Langevin revision in the previous subsection amounts to a noise-injected back-propagation for minimizing $\text{KL}(q_\alpha|p_\theta)$ with respect to α. Specifically, consider gradient ascent on $\text{E}[f(g(X;\alpha);\theta)]$ with respect to α, where the expectation is with respect to $X \sim \text{N}(0, I_d)$. The chain rule computation involves $\partial f/\partial Y \times \partial Y/\partial \alpha$, with $Y = g(X;\alpha)$. Special case (2) amounts to adding noise to $\partial f/\partial Y$ according to the Langevin dynamics, and then back-propagate to α. The added noise increases the entropy of q_α.

Special case (1) as variational learning of MCMC teaching. For MCMC teaching, the right diagram in (17) leads to the update $\alpha^{(t+1)} = \arg\min_\alpha \text{KL}(\mathcal{M}_\theta q_{\alpha^{(t)}}|q_\alpha)$, where $\{\tilde{Y}_i\} \sim \mathcal{M}_\theta q_{\alpha^{(t)}}$ serve as the training data, and the latent vector X_i is inferred by Langevin dynamics initialized from $\hat{X}_i$. The left diagram in (17), i.e., the special case (1) in the previous subsection, can be viewed as a simplified approximation to the right diagram by fixing the latent factors at $\hat{X}_i$. It minimizes a variational upper bound

$$\text{KL}(\mathcal{M}_\theta q_{\alpha^{(t)}}|q_\alpha) + \text{KL}(q(\hat{X}_i|\tilde{Y}_i, \alpha^{(t)})|q(X_i|\tilde{Y}_i, \alpha)). \quad (26)$$

Our experiments suggest that the variational learning step in the left diagram works as well as the maximum likelihood learning step in the right diagram.

4 Experiments

We scale the training data to the range of the tanh activation function $[-1, 1]$. All learning parameters are initialized from a zero-centered Normal distribution with standard deviation 0.001. For the descriptor network, we adopt the structure of [15], where the bottom-up network consists of multiple layers of convolution by linear filtering, ReLU non-linearity, and down-sampling. We adopt the structure of the generator network of [18], [21], where the top-down network consists of multiple layers of deconvolution by linear superposition with up-sampling, batch normalization [51], and ReLU non-linearity, with tanh non-linearity at the bottom-layer [21] to make the signals fall within $[-1, 1]$. In our experiments, we set $l_q = 0$ and infer $X_i = \hat{X}_i$, i.e., we follow the left diagram in (17). We have also experimented with $l_q > 0$, i.e., the right diagram, but did not observe significant improvement.

4.1 Experiment 1: Generating texture patterns

We conduct experiments on generating texture patterns. We learn a separate model from each texture image. The training images are collected from the Internet, and then resized to 224 × 224 pixels. The synthesized images are of the same size as the training images. We use a 3-layer descriptor network, where the first layer has 100 15 × 15 filters with sub-sampling rate of 3 pixels, the second layer has 70 9 × 9 filters with sub-sampling of 1, and the third layer

Fig. 5. Generating texture patterns. Each row displays one texture experiment, where the first image is the training image, and the rest are 3 of the images generated by the CoopNets algorithm. The observed and synthesized images are of size 224 $\times$ 224 pixels.

has 30 7 $\times$ 7 filters with sub-sampling of 1. We set the standard deviation of the reference distribution of the descriptor network to be $s = 0.012$. We use $l_p = 20$ or 30 steps of Langevin revision dynamics within each learning iteration, and the Langevin step size is set at 0.003. The learning rate is 0.01. Starting from 7 $\times$ 7 latent factors, the generator network has 5 layers of deconvolution with 5 $\times$ 5 kernels (basis functions), with an up-sampling factor of 2 at each layer (i.e., the basis functions are 2 pixels apart). The standard deviation of the noise vector is $\sigma = 0.3$. The learning rate is 10^{-6}. We run 10^4 cooperative learning iterations to train the models.

Figure 5 displays the results of generating texture patterns. The synthesis results of the CoopNets algorithm shown in this paper

Fig. 6. Generating object patterns. Each row displays one object experiment, where the first 3 images are 3 of the training images, and the rest are 6 of the synthesized images generated by the CoopNets algorithm. The observed and synthesized images are of size 64×64 pixels.

are those generated by the descriptor (i.e. $\tilde{Y}_i$) unless otherwise specified. For each category, the first image is the training image, and the rest are 3 of the images generated by the learning algorithm. We run $\tilde{n} = 6$ parallel chains for the first example, where images from 3 of them are presented. We run a single chain for the rest of the examples, where the synthesized images are generated at different iterations. Even though we run a single chain, it is as if we run an infinite number of chains, because in each iteration, we run Langevin revision dynamics from a new image sampled from the generator.

4.2 Experiment 2: Generating object patterns

We study generating object patterns via the CoopNets algorithm. We use object categories selected from Imagenet-1k dataset [52]. Each category contains roughly 1,200+ training images, each of which is resized to 64×64 pixels.

We adopt a 4-layer descriptor network, where the first layer has 64 5 $\times$ 5 filters with sub-sampling of 2 pixels, the second layer has 128 3 $\times$ 3 filters with sub-sampling of 2, the third layer has 256 3 $\times$ 3 filters with sub-sampling of 1, and the final layer is a fully connected layer with 100 channels as output. We set the number of Langevin dynamics steps in each learning iteration to $l_p = 10$ and the step size to 0.002. The standard deviation for reference distribution is $s = 0.016$. The learning rate is 0.007.

Starting from a 100-dimensional latent factor, the generator network has one fully connected layer with 4×4 kernels under the latent factors, which is followed by 4 layers of deconvolution with kernels of size 5×5, and up-sampling factor of 2. The numbers of channels from top layer to bottom layer are 512, 256, 128, 64, and 3 respectively. The output size is 64×64. The learning rate for updating the parameters of generator is 0.0001. We set $\sigma = 0.3$ for the standard deviation of the noise vector ϵ.

The CoopNets algorithm is trained by Adam optimizer [53] with a mini-batch size of 100. We run $\tilde{n} = 144$ parallel chains for synthesis. The number of cooperative learning iterations is $1,000$. After learning the models, we synthesize images using the learned models. As in the CoopNets algorithm, we sample

from the learned descriptor network by running 10 to 50 steps of Langevin dynamics initialized from the examples generated by the learned generator network.

For qualitative experiment, we learn a separate model for each of 7 selected object categories (i.e., sunflower, rose, mushroom, strawberry, sea star, egret, and school bus). Figure 6 shows the synthesis results. Each row displays an experiment, where the first 3 images are 3 typical examples of the training images, and the rest are 6 of the synthesized images generated by the CoopNets algorithm.

For quantitative experiment, we use five object categories (i,e., lemon, lifeboat, strawberry, school bus and zebra), and train models on different numbers of randomly sampled training images for each category. The synthesis quality is quantitatively evaluated using three criteria: (1) average softmax class probability that Inception network [54] assigns to the synthesized images for the underlying category. (2) top-5 classification error by Inception network, i.e., the probability that the underlying category does not belong to the categories with the top 5 softmax probabilities. (3) Average pairwise structural similarity [55] between two randomly selected synthesized images. While (1) and (2) examine the realism of the synthesized images, (3) examines the variabilities of the synthesized images. The lack of variabilities may be caused by mode collapsing.

Figure 7 displays the average results over the 5 categories versus the number of training examples. It can be seen that CoopNets generates images with higher softmax class probabilities, lower classification errors, and higher variabilities than DCGAN [21], VAE [30] and separate training by Algorithm G. The advantage can be due to the fact that both models in CoopNets are learned generatively by maximum likelihood. Our experiments suggest that the CoopNets learning method is stable and does not encounter mode collapsing issue.

4.3 Experiment 3: Generating scene patterns

We conduct experiments on synthesizing scene patterns. We learn a separate model for each of the 4 scene categories (i.e., volcano, desert, rock, and apartment building) selected from MIT place205 dataset [56]. Each category has 10,000+ training images. The images are resized to 64×64 pixels. We adopt the same architecture of CoopNets as the one for object patterns in Section 4.2. Figure 8 displays 36 synthesized scene images generated by the learned model, along with 18 randomly sampled training images for each experiment.

We then learn from mixed images that are randomly sampled from 10 different scene categories (i.e., alp, cliff drop, cliff dwelling, geyser, lakeside, promontory, sandbar, seashore, valley, and volcano) selected from Imagenet-1k dataset. We conduct 7 runs. The numbers of images sampled from each category are 50, 100, 300, 500, 700, 900, and 1,100 respectively in these 7 runs. Figure 9 displays the observed examples randomly sampled from the training set, and the synthesized examples generated by the CoopNets, where the number of training images from each category is 1,100. The synthesized examples are randomly sampled from the learned models without cheery picking. We evaluate the synthesis quality by the Inception score [57]. Table 1 displays the Inception scores of the CoopNets, DCGAN, EBGAN [58], Wasserstein GAN [59], InfoGAN [60], VAE, the method of [29], and separate training by Algorithm G and Algorithm D. For Algorithm D, we initialize the synthesized examples from the observed examples, so it is persistent contrastive divergence [44].

Figure 10 shows 5 examples of interpolation between two latent vectors of X. For each row, the images at the two ends are generated from X vectors randomly sampled from $\mathrm{N}(0, I_d)$. Each image in the middle is obtained by first interpolating the X vectors of the two end images, and then generating the image using the generator, followed by 10 steps of Langevin dynamics. This experiment shows that we learn smooth generator model that traces the manifold of the data distribution.

4.4 Experiment 4: Generating handwritten digits

We learn CoopNets from MNIST dataset [1] of handwritten digits. The training images are grey-scale with size of 28×28 pixels. We adopt a 4-layer descriptor network, where the numbers of filters at different layers are 64, 128, 256, and 100 from bottom to top. The filter size of each layer is 4×4, and the sub-sampling rate is 2. The final layer is a fully connected layer. Taking a 100-dimensional latent vector as input, the generator network consists of one fully connected layer and 3 deconvolutional layers with kernels of size 4×4 and up-sampling factor of 2. The numbers of channels from top layer to bottom layer are 512, 256, 128, and 1 respectively. The output size is 28×28 pixels. Except network architectures, we follow the same hyper-parameter setting as used in Section 4.2. Figure 11 displays some synthesized examples generated by the learned models after training.

To quantitatively evaluate the learned model, we first synthesize 10,000 samples from the CoopNets learned on the training set of 55,000 examples. We then fit a Gaussian Parzen window to the synthesized samples, and estimate the log-likelihood of the testing set using the Parzen window distribution. The standard deviation of the Gaussian is obtained by cross validation on the validation set. This evaluation method was proposed by [61] and has been used by [19], [62], [63] for evaluating generative models with non-tractable likelihoods. Figure 12 displays the Parzen window-based log-likelihood estimates of the MNIST testing set for both the descriptor network and the generator network trained by the CoopNets algorithm with different numbers of training epochs. It can be seen that in the proposed cooperative training scheme, the descriptor network improves the generator network, and the generator network eventually gets very close to the descriptor network. We also compare our model against other baselines, e.g., DBN [63], Stacked CAE [63], Deep GSN [62], and GAN [19] in Table 2, where both descriptor and generator networks outperform other baseline models.

4.5 Experiment 5: Evaluation on large-scale benchmark datasets

To demonstrate how the CoopNets scales with more complex data, we evaluate the model on three challenging large-scale benchmark datasets, i.e., LSUN bedrooms [64], CelebA human faces [65], and Cifar-10 objects [66] datasets.

We first test the CoopNets on the LSUN bedrooms dataset containing 3,033k training images of 256×256 pixels. The network architectures are as follows. The descriptor network consists of 5 convolutional layers with numbers of channels $\{64, 128, 256, 512, 512\}$, filter sizes $\{5, 5, 5, 5, 3\}$, and sub-sampling factors $\{2, 2, 2, 2, 2\}$ at different layers (from bottom to top), and one fully connected layer with 10 filers. The generator network takes as input a 100-dimensional latent factor, and consists of 1 fully connected and 4 deconvolutional layers with numbers of channels $\{512, 256, 128, 64, 3\}$, kernels sizes

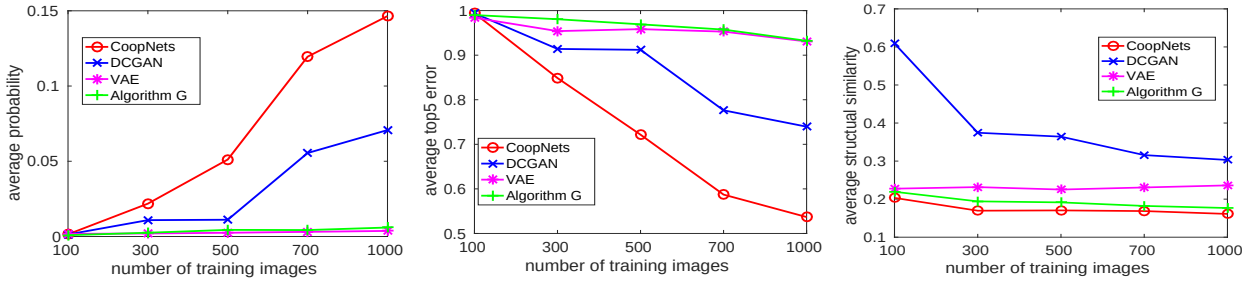


Fig. 7. Left: Average softmax class probability on single Imagenet-1k category versus the number of training images. Middle: Top 5 classification error. Right: Average pairwise structural similarity.

(a) volcano (b) desert (c) rock (d) apartment building

Fig. 8. Generating scene patterns. Both observed and synthesized scene images are shown for each category. The image size is 64×64 pixels. The categories are from MIT places205 dataset. (a) volcano. (b) desert. (c) rock. (d) apartment building.

TABLE 1
Inception scores of different methods on learning from 10 Imagenet-1k scene categories. n is the number of training images randomly sampled from each category.

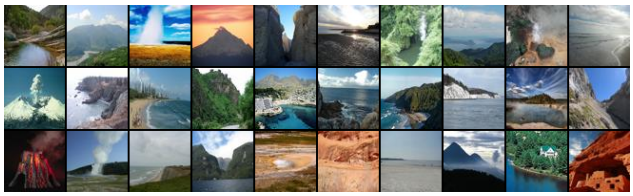
	n = 50	n = 100	n = 300	n = 500	n = 700	n = 900	n = 1100
DCGAN [21]	2.26±.16	2.50±.15	3.16±.15	3.05±.12	3.13±.09	3.34±.05	3.47±.06
EBGAN [58]	2.23±.17	2.40±.14	2.62±.08	2.46±.09	2.65±.04	2.64±.04	2.75±.08
W-GAN [59]	1.80±.09	2.19±.12	2.34±.06	2.62±.08	2.86±.10	2.88±.07	3.14±.06
VAE [30]	1.62±.09	1.63±.06	1.65±.05	1.73±.04	1.67±.03	1.72±.02	1.73±.02
InfoGAN [60]	2.21±.04	1.73±.01	2.15±.03	2.42±.05	2.47±.05	2.29±.03	2.08±.04
Method of [29]	2.44±.27	2.38±.13	2.42±.09	2.94±.11	3.02±.06	3.08±.08	3.15±.06
Algorithm G [25]	1.72±.07	1.94±.09	2.32±.09	2.40±.06	2.45±.05	2.54±.05	2.61±.06
Persistent CD [44]	1.30±.08	1.94±.03	1.80±.02	1.53±.02	1.45±.04	1.35±.02	1.51±.02
CoopNets (ours)	**2.66±.13**	**3.04±.13**	**3.41±.13**	**3.48±.08**	**3.59±.11**	**3.65±.07**	**3.79±.15**

TABLE 2
A comparison of Parzen window-based log-likelihood estimates for MNIST dataset. The mean log-likelihood of testing samples, with the standard error of the mean computed across examples, are reported.

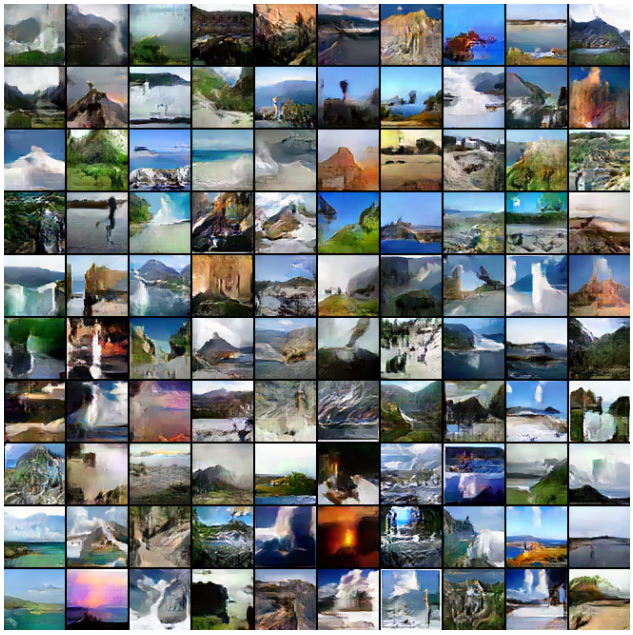
Model	Log-likelihood
DBN [63]	138 ± 2.0
Stacked CAE [63]	121 ± 1.6
Deep GSN [62]	214 ± 1.1
GAN [19]	225 ± 2.0
Generator in CoopNets (ours)	226 ± 2.1
Descriptor in CoopNets (ours)	**228 ± 2.1**

$\{16, 5, 5, 5, 5\}$, and up-sampling factors $\{1, 2, 2, 2, 2\}$ at different layers (from top to bottom). Figure 13 displays the synthesized images.

We then learn a model from the CelebA human faces dataset with 200k training images of 128×128 pixels. For this dataset, we adopt a descriptor network that has 3 convolutional layers with numbers of channels $\{64, 128, 256\}$, filter sizes $\{5, 3, 3\}$ and sub-sampling factors $\{2, 2, 1\}$ at different layers (from bottom to top), and one fully connected layer with 100 filers. Besides, we adopt a generator network that takes a 100-dimensional input and consists of one fully connected and 4 deconvolutional layers with numbers of channels $\{512, 256, 128, 64, 3\}$, kernels sizes $\{4, 5, 5, 5, 5\}$



(a) observed images



(b) synthesized images

Fig. 9. Generating scene patterns. (a) observed images randomly selected from 10 Imagenet-1k scene categories. (b) synthesized images generated by CoopNets learned from 10 Imagenet-1k scene categories. The training set consists of 1,100 images randomly sampled from each category. The observed and synthesized images are of size 64 $\times$ 64 pixels.

Fig. 10. Interpolation between latent vectors of the scene images on the two ends.

and up-sampling factors $\{1, 2, 4, 2, 2\}$ at different layers respectively (from top to bottom). Figure 14 shows synthesis results generated by the learned model.

Figure 15 displays synthesis results generated by the model learned from Cifar-10 objects dataset with 60k training images of 32 $\times$ 32 pixels. The descriptor network has 3 convolutional layers with numbers of channels $\{64, 128, 256\}$, filter sizes $\{5, 3, 3\}$ and

Fig. 11. Generating handwritten digits. The synthesized images are generated by the CoopNets algorithm that learns from MNIST dataset with 55,000 training images. The observed and synthesized images are of size 28 $\times$ 28 pixels.

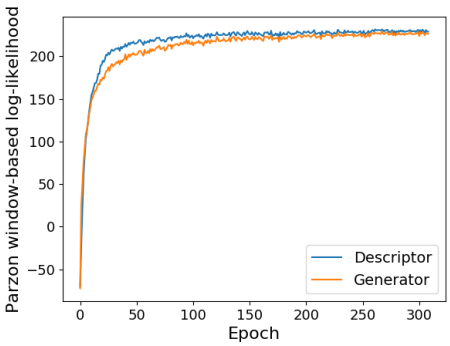
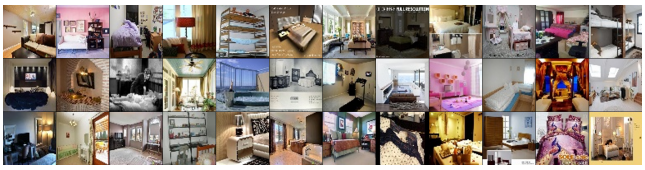


Fig. 12. Parzen window-based log-likelihood estimates of the descriptor network and the generator network in the CoopNets algorithm.

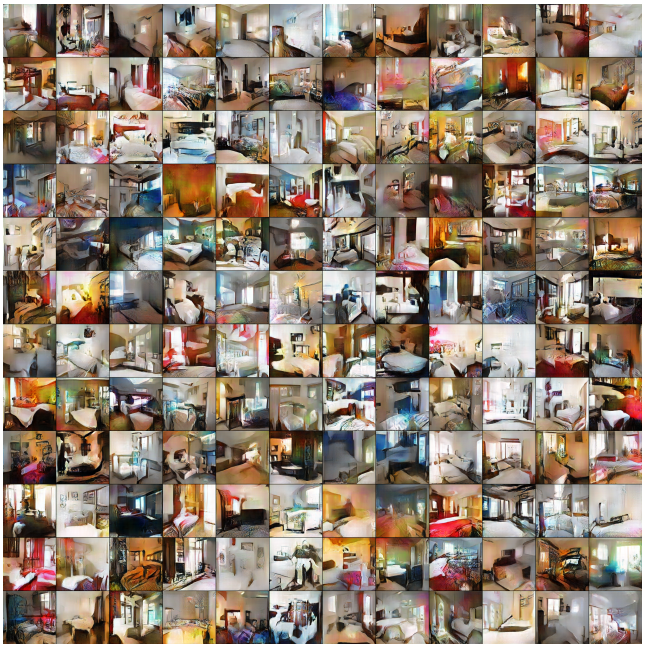
sub-sampling factors $\{2, 2, 2\}$, and one fully connected layer of 100 filters (from bottom to top). Starting from a 100-dimensional latent vector, the generator network has 1 fully connected and 3 deconvolutional layers with numbers of channels $\{256, 128, 64, 3\}$, kernels sizes $\{4, 5, 5, 5\}$ and up-sampling factors $\{1, 2, 2, 2\}$ (from top to bottom). The number of Langevin dynamics steps in each learning iteration is 10. The size of mini-batch is 300. The number of parallel chains is $\tilde{n} = 400$. We disable the noise term in the Langevin revision dynamics in the second half of the learning algorithm for faster convergence.

We evaluate the synthesis results by the Fréchet Inception Distance (FID) [67], which measures the dissimilarity between generated images and real ones. Table 3 shows the performance of CoopNets, DCGAN, W-GAN, and VAE on LSUN bedrooms, CelebA and Cifar-10 datasets with respect to FID. The experiments show that the learned generator is good enough as a standalone model, though the learned descriptor produces even better results with Langevin revision. Both of them outperform the other baseline methods in terms of FID on these three benchmark datasets.

We further conduct a human perceptual study to compare the perceived realism of the synthesized examples generated by different generative models. More specifically, we randomly select 80 human subjects and ask them to rank the models, which includes CoopNets, DCGAN, W-GAN and VAE, according to

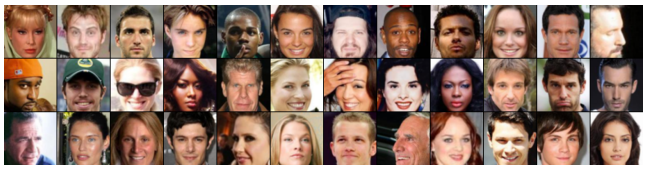


(a) training images

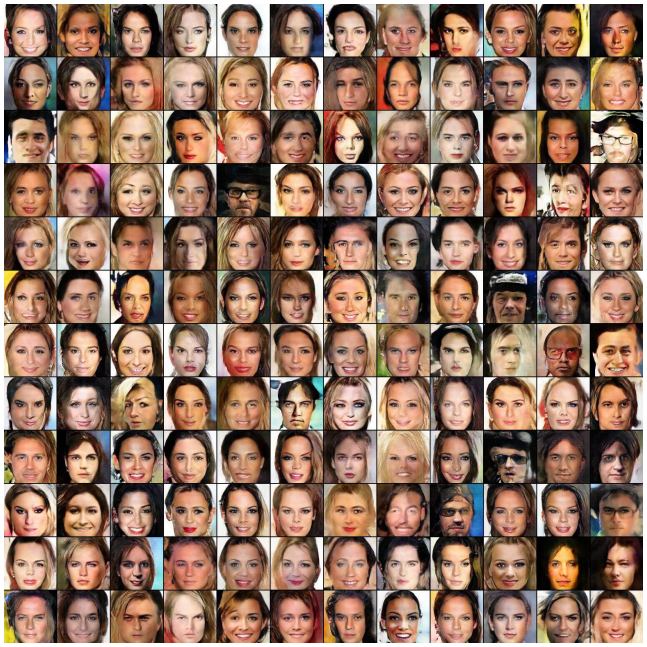


(b) synthesized images

Fig. 13. Generating bedroom images (256 × 256 pixels). The synthesized images are generated by the CoopNets algorithm that learns from LSUN dataset with 3033k training images.



(a) training images



(b) synthesized images

Fig. 14. Generating human face images (128 × 128 pixels). The synthesized images are generated by the CoopNets algorithm that learns from CelebA dataset with 200k training images.

TABLE 3
The performance of CoopNets, DCGAN, W-GAN, and VAE on LSUN bedrooms, CelebA and Cifar-10 datasets with respect to the Fréchet Inception Distance (FID).

	LSUN	CelebA	Cifar-10
W-GAN [59]	67.72	52.54	48.40
DCGAN [21]	70.40	21.40	37.70
VAE [30]	243.47	50.53	126.32
Generator in CoopNets (ours)	64.30	16.98	35.25
Descriptor in CoopNets (ours)	**35.42**	**16.65**	**33.61**

TABLE 4
Human perceptual study for comparing synthesis qualities of different generative models. The numbers are the average rankings.

	LSUN	CelebA	Cifar-10
W-GAN [59]	2.400	3.800	2.171
DCGAN [21]	1.913	2.463	2.395
VAE [30]	3.988	2.300	3.987
CoopNets (ours)	**1.700**	**1.438**	**1.447**

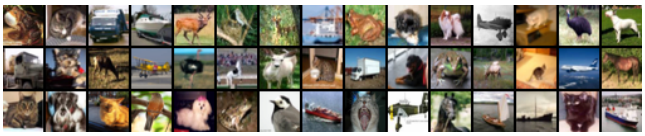
the quality of their synthesized images. For each human subject, we present 16 synthesized images per model without telling the subject which model generates the images. The model order is randomized to ensure fair comparisons. Table 4 summarizes the average ranking results over human subjects on LSUN bedrooms, CelebA and Cifar-10 datasets. It can be seen that our model can generate more realistic images than the other baseline methods in this experiment.

4.6 Experiment 6: Pattern completion

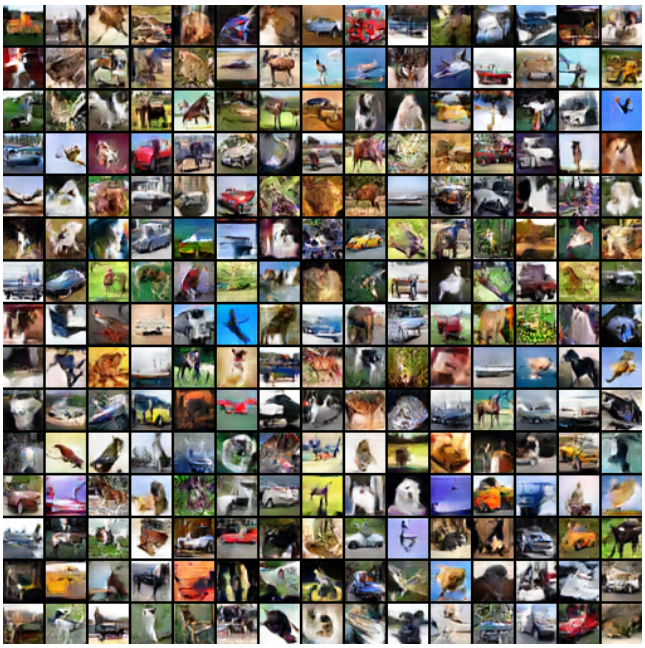
We conduct an experiment on learning from training images of human faces, and then testing the learned models on completing the occluded testing images. This can be considered an inpainting task. It can also be viewed as a form of content addressable memory or associative memory. The purpose of this experiment is to check whether the learned generator model can generalize to the testing data, i.e., whether the learned model overfits or underfits the training data. Underfitting can happen due to mode collapsing. We believe this is a powerful test of the generalizability of the learned generator model.

We adopt a 4-layer descriptor network, where the numbers of filters from bottom layer to top layer are 96, 128, 256, and 50. The filter size of each layer is 5×5, and the sub-sampling rate is 2. The final layer is a fully connected layer. The structure of the generator network is the same as in Section 4.2. The training data are $10,000$ human faces randomly selected from CelebA dataset, and resized to 64×64 pixels.

To quantitatively test whether we have learned a good generator network $g(X;\alpha)$ even though it has never seen the training images directly in the training stage, we apply it to the task of recovering the occluded pixels of testing images. For each occluded testing image Y, we use Step G1 of Algorithm G to infer the latent factors X. The only change is with respect to

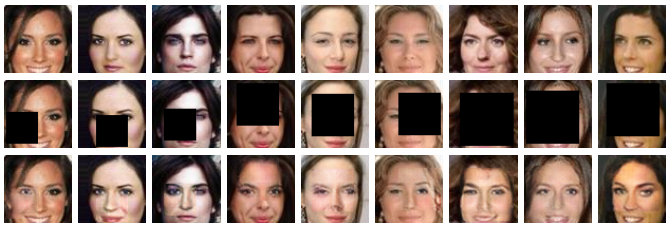


(a) training images

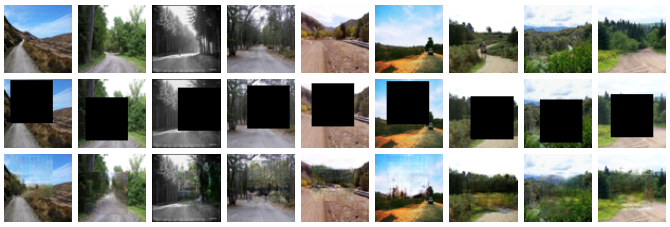


(b) synthesized images

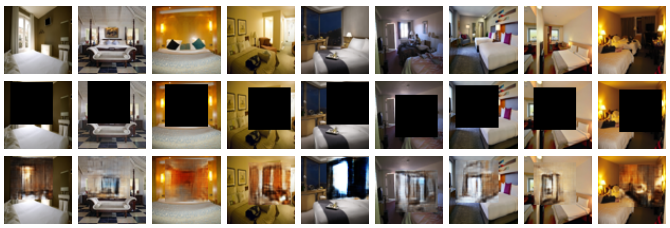
Fig. 15. Generating object images (32 × 32 pixels). The synthesized images are generated by the CoopNets algorithm that learns from Cifar-10 dataset with 60,000 training images.



(a) face



(b) forest road



(c) hotel room

Fig. 16. Pattern completion. First row: original images. Second row: occluded images. Third row: recovered images by the generator network learned via the CoopNets algorithm. (a) face. (b) forest road. (c) hotel room.

the term $\|Y - g(X;\alpha)\|^2$, where the sum of squares is over all the observed pixels of Y in back-propagation computation. We run 1,000 Langevin steps, initializing X from $\mathrm{N}(0, I_d)$. After inferring X, the completed image $g(X;\alpha)$ is automatically obtained. We design 3 experiments, where we randomly place a 30×30, 40×40, or 50×50 mask on each 64×64 testing image. These 3 experiments are denoted by M30, M40, and M50 respectively (M for mask).

We report the recovery errors and compare our method with 7 different image inpainting methods as well as the DCGAN of [21]. For DCGAN, we use the parameter setting in [21] and the number of learning iterations is 600. We use the same $10,000$ training images to learn DCGAN. After the model is learned, we keep the generator and use the same method as ours to infer the latent factors X, and recover the unobserved pixels. In the 7 inpainting methods, Methods MRF-ℓ_1 and MRF-ℓ_2 are based on Markov random field priors where the nearest neighbor potential terms are ℓ_1 and ℓ_2 differences respectively. Methods inter-1 to 5 are interpolation methods. Please refer to [68] for details.

Table 5 displays the recovery errors of the 3 experiments, where the error is measured by per pixel difference (relative to the range of pixel values) between the original image and the recovered image on the occluded region, averaged over 1,000 testing images. We also measure the error by the peak signal-to-noise ratio (PSNR). Figure 16 (a) displays some recovery results by our method. The first row shows the original images as the ground truth. The second row displays the testing images with occluded pixels. The third row displays the recovered images by the learned generator network.

We also apply the same approach to the forest road category and the hotel room category in MIT place205 dataset. See Figure 16 (b) and (c) for some qualitative results. The learned generator is quite imaginative in completing the occluded pixels.

4.7 Experiment 7: Generating dynamic textures

We can learn to generate video sequences by cooperative training of a spatial-temporal descriptor network [41] and a spatial-temporal generator network. The spatial-temporal descriptor network consists of multiple layers of spatial-temporal filters that capture spatial-temporal features at various scales of the video sequences, while the spatial-temporal generator network maps the latent variables to the video sequences by multiple layers of spatial-temporal kernels (basis functions). We call the resulting model the spatial-temporal CoopNets (ST-CoopNets).

We conduct experiments on generating dynamic textures [69] with temporal stationarity. We learn a spatial-temporal CoopNets for each category from one training video. We collect the training video clips from DynTex++ dataset of [70] and the Internet. Each training video clip is of size 128 pixels × 128 pixels × 64 frames.

Since dynamic textures may have structured background that are not stationary in the spatial domain (e.g., burning fire heating a pot shown in Figure 17 (a)), we adopt spatially fully connected and temporally convolutional layer in the bottom-up ConvNet structure of the descriptor. Specifically, we use a 3-layer descriptor network. The first layer has 120 5 pixels × 5 pixels × 5 frames filters with sub-sampling size of 2 pixels and frames. The second layer is

TABLE 5
A comparison of recovery performances of different methods in 3 experiments

	task	CoopNets	DCGAN	MRF-ℓ_1	MRF-ℓ_2	inter-1	inter-2	inter-3	inter-4	inter-5
error	M30	**0.115**	0.211	0.132	0.134	0.120	0.120	0.265	0.120	0.120
	M40	**0.124**	0.212	0.148	0.149	0.135	0.135	0.314	0.135	0.135
	M50	**0.136**	0.214	0.178	0.179	0.170	0.166	0.353	0.164	0.164
PSNR	M30	**16.893**	12.116	15.739	15.692	16.203	16.635	9.524	16.665	16.648
	M40	**16.098**	11.984	14.834	14.785	15.065	15.644	8.178	15.698	15.688
	M50	**15.105**	11.890	13.313	13.309	13.220	14.009	7.327	14.164	14.161

a spatially fully connected layer, which contains 30 filters that are fully connected in the spatial domain but convolutional in the temporal domain. The temporal size of the filters is 5 with sub-sampling size of 2 and padding size of 2 in the temporal dimension. Due to the spatial full connectivity at the second layer, the spatial domain of the feature maps at the third layer is reduced to 1×1. The third layer has 10 $1 \times 1 \times 5$ filters with sub-sampling size of 2 and padding size of 2 in the temporal dimension. We run $l_p = 10$ Langevin revision steps in each iteration with the step size of 0.002. We set the standard deviation for reference distribution to $s = 0.016$. The learning rate is 0.01.

The generator network maps a 5-channel $1 \times 1 \times 10$ latent factors to the video sequence by a 4-layer top-down ConvNet. Due to spatial non-stationarity and temporal stationarity of dynamics textures, we use spatially fully connected and temporally deconvolutional layer in the upper layers of the generator network. Specifically, the first layer uses $1 \times 1 \times 5$ kernels. The second layer is a spatially fully connected layer with kernels of size 5 in the temporal dimension. The third layer uses $5 \times 5 \times 5$ kernels with an up-sampling factor 2. The fourth layer uses $5 \times 5 \times 2$ kernels with an up-sampling size $2 \times 2 \times 1$. The numbers of channels at different layers are 256, 128, 64, and 3 from top to bottom. Batch normalization and ReLU layers are used between consecutive layers and tanh is added at the bottom-layer. The standard deviation of the noise ϵ is set to $\sigma = 1$. The learning rate is 0.0002.

We use Adam optimizer to train the models. We run $\tilde{n} = 2$ parallel chains. Figure 17 displays the results. For each category, the first row shows 6 frames of the observed sequence, and the second and third rows show the corresponding frames of 2 synthesized sequences generated by the learning algorithm. We use the same set of parameters for all the categories without tuning. Our experiments show that the cooperative training of spatial-temporal descriptor and generator can synthesize realistic dynamic textures.

To evaluate the quality of the synthesized examples, we compare our model with some baseline models for dynamic textures (e.g., LDS [69], FFT-LDS [71], MKGPDM [72], and HOSVD [73]) in terms of PSNR and structural similarity measures (SSIM) on 6 dynamic texture videos. Each model learns from 64 image frames of each observed video, and then generates a 64-frame dynamic texture video for evaluation. The PSNR and SSIM are computed between the generated example and the observed example. Table 6 shows the average performances of the models over the 6 videos. Our models are comparable or better than other baseline methods for dynamic textures.

TABLE 6
A comparison of models for synthesizing dynamic textures

Model	PSNR	SSIM
LDS [69]	19.148	0.5939
FFT-LDS [71]	12.463	0.2898
MKGPDM [72]	14.288	0.3577
HOSVD [73]	18.392	0.4573
CoopNets (ours)	**19.407**	**0.5988**

5 Conclusion

This paper studies the fundamental problem of learning two important classes of deep generative models of images. By allowing the two models to cooperate with each other in a cooperative learning algorithm, our method can learn highly realistic generative models. We quantitatively evaluate our method on synthesis qualities of the generated images as well as the recovery errors of the reconstructed images.

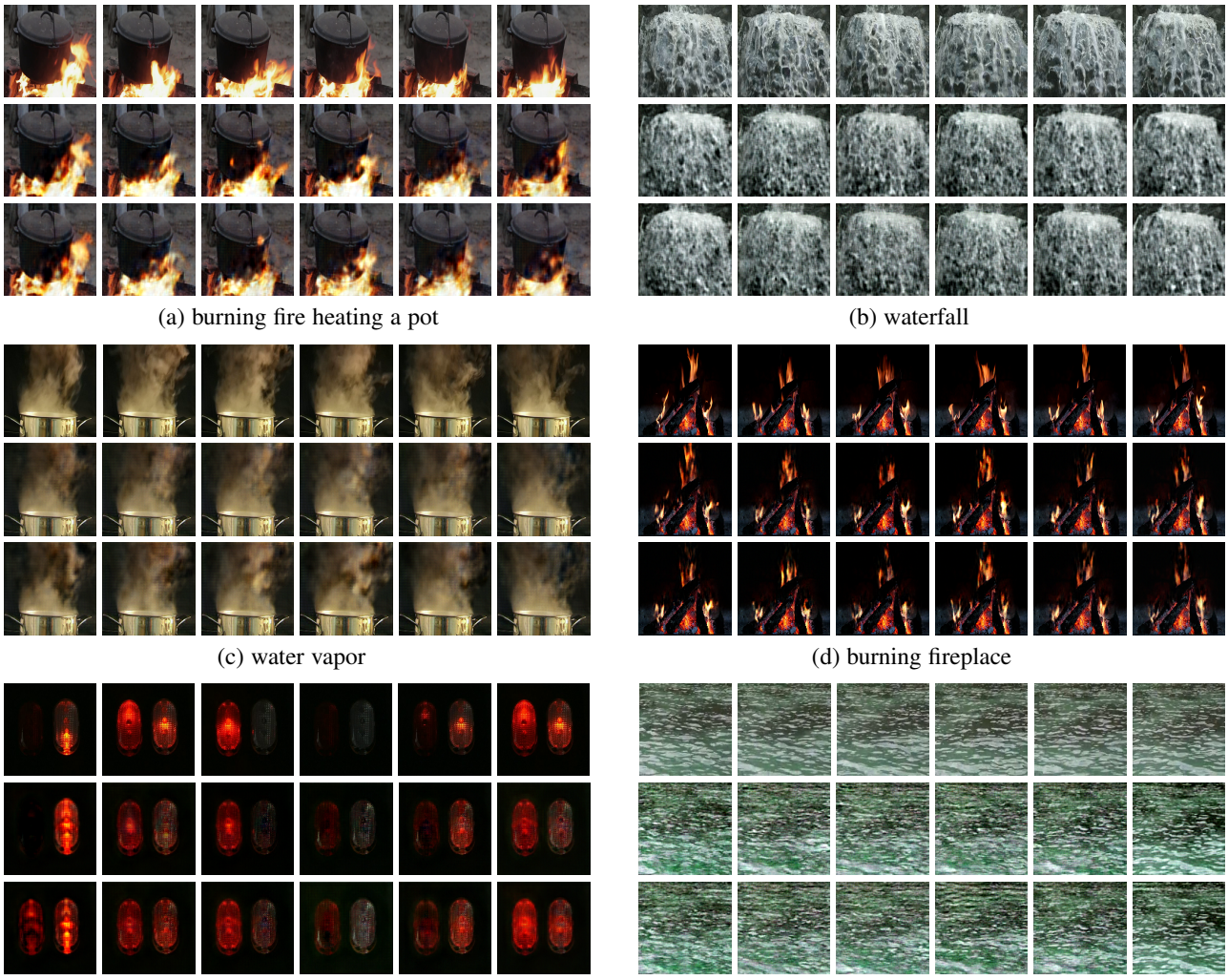
The most unique feature of our work is that the descriptor and the generator networks feed each other the synthesized data in the learning process. The generator feeds the descriptor the initial version of the synthesized data. The descriptor feedbacks the generator the revised version of the synthesized data. The generator then produces the reconstructed version of the synthesized data. While the descriptor learns from finite amount of observed data, the generator learns from virtually infinite amount of synthesized data.

Another unique feature of our work is that the learning process interweaves the existing maximum likelihood learning algorithms for the two networks, so that the two algorithms jumpstart each other's MCMC sampling.

A third unique feature of our work is that the generator accumulates the MCMC transitions of the descriptor via MCMC teaching, and reproduces the MCMC transitions by direct ancestral sampling. In other words, the descriptor distills its MCMC algorithm into the generator. Powering the MCMC sampling of the descriptor model in [14], [15] is a main motivation of this paper, with the bonus of turning the unsupervised learning of the generator [25] into supervised learning.

Our cooperative training method can be generalized to a conditional version for learning the conditional distribution of a high-dimensional output given an input, which may also be high-dimensional. Examples include image generation given class label, text description, or abstract sketch, as well as recovering depth map from image. In the conditional cooperative learning method, the conditional descriptor learns the objective function or the conditional random field, and the conditional generator learns to initialize the sampling or optimization algorithm for the objective function.

Compared to reinforcement learning [74], the descriptor network learns the cost function, the MCMC algorithm is like an optimal control algorithm, and the generator network is like a learned policy. Our method distills the optimal control algorithm into a policy. The optimal control is slow thinking while the policy



(a) burning fire heating a pot (b) waterfall (c) water vapor (d) burning fireplace (e) flashing lights (f) lake

Fig. 17. Generating dynamic textures by the spatial-temporal CoopNets algorithm. For each category, the first row displays the frames of the observed sequence, and the second and third rows display the corresponding frames of two synthesized sequences generated by the learning algorithm. The observed and synthesized videos are of size 128 pixels $\times$ 128 pixels $\times$ 64 frames. (a) burning fire heating a pot. (b) waterfall. (c) water vapor. (d) burning fireplace. (e) flashing lights. (f) lake.

is fast thinking. We shall investigate this connection in our future work.

PROJECT PAGE

http://www.stat.ucla.edu/~jxie/CoopNets/CoopNets.html

ACKNOWLEDGMENTS

We thank Hansheng Jiang, Zilong Zheng, Erik Nijkamp, Tengyu Liu, Yaxuan Zhu, Zhaozhuo Xu and Xiaolin Fang for their assistance with coding and experiments.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

The work is supported by Hikvision gift fund, NSF DMS 1310391, DARPA SIMPLEX N66001-15-C-4035, ONR MURI N00014-16-1-2007, and DARPA ARO W911NF-16-1-0579.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 1, 10

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105. 1

[3] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*. MIT Press, 2006. 1

[4] G. E. Hinton, "Training products of experts by minimizing contrastive divergence." *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002. 1, 4, 7

[5] S.-C. Zhu, Y. N. Wu, and D. Mumford, "Minimax entropy principle and its application to texture modeling," *Neural Computation*, vol. 9, no. 8, pp. 1627–1660, 1997. 1

[6] S. Roth and M. J. Black, "Fields of experts: A framework for learning image priors," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 860–867. 1

[7] D. B. Rubin and D. T. Thayer, "EM algorithms for ML factor analysis," *Psychometrika*, vol. 47, no. 1, pp. 69–76, 1982. 1

[8] S.-C. Zhu, "Statistical modeling and conceptualization of visual patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 6, pp. 691–712, 2003. 1

[9] C.-E. Guo, S.-C. Zhu, and Y. N. Wu, "Modeling visual patterns by integrating descriptive and generative methods," *International Journal of Computer Vision*, vol. 53, no. 1, pp. 5–29, 2003. 1

[10] Y.-W. Teh, M. Welling, S. Osindero, and G. E. Hinton, "Energy-based models for sparse overcomplete representations," *Journal of Machine Learning Research*, vol. 4, pp. 1235–1260, 2003. 1

[11] Y. N. Wu, S.-C. Zhu, and C.-E. Guo, "From information scaling of natural images to regimes of statistical models," *Quarterly of Applied Mathematics*, vol. 66, pp. 81–122, 2008. 1

[12] J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng, "Learning deep energy models," in *International Conference on Machine Learning*, 2011, pp. 1105–1112. 1

[13] J. Dai, Y. Lu, and Y. N. Wu, "Generative modeling of convolutional neural networks," in *International Conference on Learning Representations*, 2015. 1

[14] Y. Lu, S.-C. Zhu, and Y. N. Wu, "Learning FRAME models using CNN filters," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. 1, 4, 15

[15] J. Xie, Y. Lu, S.-C. Zhu, and Y. N. Wu, "A theory of generative ConvNet," in *International Conference on Machine Learning*, 2016. 1, 2, 4, 8, 15

[16] J. Lazarow, L. Jin, and Z. Tu, "Introspective generative modeling: Decide discriminatively," *arXiv preprint arXiv:1704.07820*, 2017. 1

[17] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional neural networks," in *European Conference on Computer Vision*, 2014, pp. 818–833. 1

[18] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1538–1546. 1, 6, 8

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680. 1, 2, 10, 11

[20] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1486–1494. 2

[21] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015. 2, 8, 10, 11, 13, 14

[22] R. M. Neal, "MCMC using hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, 2011. 2

[23] M. Girolami and B. Calderhead, "Riemann manifold langevin and hamiltonian monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 2, pp. 123–214, 2011. 2

[24] S. C. Zhu and D. Mumford, "Grade: Gibbs reaction and diffusion equations," in *International Conference on Computer Vision*, 1998, pp. 847–854. 2

[25] T. Han, Y. Lu, S.-C. Zhu, and Y. N. Wu, "Alternating back-propagation for generator network," in *31st AAAI Conference on Artificial Intelligence*, 2017. 2, 6, 11, 15

[26] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977. 2

[27] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *AISTATS*, 2009. 4

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006. 4

[29] T. Kim and Y. Bengio, "Deep directed generative models with energy-based probability estimation," *arXiv preprint arXiv:1606.03439*, 2016. 4, 8, 10, 11

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations*, 2014. 4, 10, 11, 13

[31] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International Conference on Machine Learning*, 2014, pp. 1278–1286. 4

[32] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," in *International Conference on Machine Learning*, 2014. 4

[33] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015. 4

[34] Z. Tu, "Learning generative models via discriminative approaches," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8. 4

[35] J. Lazarow, L. Jin, and Z. Tu, "Introspective neural networks for generative modeling," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2774–2783. 4

[36] L. Jin, J. Lazarow, and Z. Tu, "Introspective classification with convolutional nets," in *Advances in Neural Information Processing Systems*, 2017, pp. 823–833. 4

[37] K. Lee, W. Xu, F. Fan, and Z. Tu, "Wasserstein introspective neural networks," *arXiv preprint arXiv:1711.08875*, 2017. 4

[38] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in Neural Information Processing Systems*, 2016, pp. 4790–4798. 4

[39] J. Xie, Y. Lu, R. Gao, and Y. N. Wu, "Cooperative learning of energy-based model and latent variable model via MCMC teaching," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2018. 4

[40] J. Xie, Z. Zheng, R. Gao, W. Wang, S.-C. Zhu, and Y. N. Wu, "Learning descriptor networks for 3D shape synthesis and analysis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8629–8638. 4

[41] J. Xie, S.-C. Zhu, and Y. N. Wu, "Synthesizing dynamic patterns by spatial-temporal generative convnet," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7093–7101. 4, 14

[42] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, pp. 400–407, 1951. 5

[43] L. Younes, "On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates," *Stochastics: An International Journal of Probability and Stochastic Processes*, vol. 65, no. 3-4, pp. 177–228, 1999. 5, 6

[44] T. Tieleman, "Training restricted boltzmann machines using approximations to the likelihood gradient," in *International Conference on Machine Learning*, 2008, pp. 1064–1071. 5, 10, 11

[45] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Twenty-Third AAAI Conference on Artificial Intelligence*, vol. 8, 2008, pp. 1433–1438. 5

[46] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1. 5

[47] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. 5, 6

[48] H. S. Seung, "Learning continuous attractors in recurrent networks," in *Advances in Neural Information Processing Systems*, 1998, pp. 654–660. 5

[49] U. Grenander and M. I. Miller, *Pattern theory: from representation to inference*. Oxford University Press, 2007. 7

[50] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012. 8

[51] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 8

[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 9

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 9

[54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826. 10

[55] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. 10

[56] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems*, 2014, pp. 487–495. 10

[57] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2226–2234. 10

[58] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *International Conference for Learning Representations*, 2017. 10, 11

[59] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," *International Conference on Machine Learning*, 2017. 10, 11, 13

[60] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2016, pp. 2172–2180. 10, 11

[61] O. Breuleux, Y. Bengio, and P. Vincent, "Quickly generating representative samples from an rbm-derived process," *Neural Computation*, vol. 23, no. 8, pp. 2058–2073, 2011. 10

[62] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," in *International Conference on Machine Learning*, 2014, pp. 226–234. 10, 11
[63] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. 10, 11
[64] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015. 10
[65] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *IEEE International Conference on Computer Vision*, 2015, pp. 3730–3738. 10
[66] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. 10
[67] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637. 12
[68] J. D'Errico. (2004) Interpolation inpainting. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/4551-inpaint-nans 14
[69] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *International Journal of Computer Vision*, vol. 51, no. 2, pp. 91–109, 2003. 14, 15
[70] B. Ghanem and N. Ahuja, "Maximum margin distance learning for dynamic texture recognition," in *European Conference on Computer Vision*, 2010, pp. 223–236. 14
[71] B. Abraham, O. I. Camps, and M. Sznaier, "Dynamic texture with fourier descriptors," in *Proceedings of the 4th International Workshop on Texture Analysis and Synthesis*, 2005, pp. 53–58. 15
[72] Z. Zhu, X. You, S. Yu, J. Zou, and H. Zhao, "Dynamic texture modeling and synthesis using multi-kernel gaussian process dynamic model," *Signal Processing*, vol. 124, pp. 63–71, 2016. 15
[73] R. Costantini, L. Sbaiz, and S. Susstrunk, "Higher order svd analysis for dynamic texture synthesis," *IEEE Transactions on Image Processing*, vol. 17, no. 1, pp. 42–52, 2008. 15
[74] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT press Cambridge, 1998, vol. 135. 15