Title
Coupling and Learning Hierarchical Generative and Descriptive Models for Image Synthesis and Analysis

Permalink
https://escholarship.org/uc/item/45n9n45p

Author
Lu, Yang

Publication Date
2017-01-01

Peer reviewed|Thesis/dissertation
Coupling and Learning Hierarchical Generative and Descriptive Models for Image Synthesis and Analysis

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Statistics

by

Yang Lu

2017
ABSTRACT OF THE DISSERTATION

Coupling and Learning Hierarchical Generative and Descriptive Models for Image Synthesis and Analysis

by

Yang Lu
Doctor of Philosophy in Statistics
University of California, Los Angeles, 2017
Professor Song-Chun Zhu, Chair

Learning a generative model with compositional structure is a fundamental problem in statistics. My thesis generalizes two major classical statistical models by introducing the Convolutional Neural Networks (ConvNets): (1) Exponential family model, which is generalized to a descriptor model by a bottom-up ConvNet. (2) Latent factor model, which is generalized to a generator model by a top-down ConvNet.

The probability distribution of descriptor is in the form of exponentially tiling of a reference distribution. The descriptor can be derived directly from the discriminative ConvNet. Assuming rectified linear units and Gaussian white noise reference distribution, the descriptor contains a representational structure with multiple layers of binary activation variables, which reconstruct the mean of the Gaussian piece. The model is learned by Maximum Likelihood Estimation (MLE). The Langevin dynamics for data synthesis is driven by reconstruction error, and the corresponding gradient descent dynamics converges to a local energy minimum that is auto-encoding.

The probability distribution of generator is in the form of Multivariate Gaussian, where the mean is computed by a non-linear ConvNet mapping function on latent factors. The model is learned by an alternating back-propagation algorithm, which underlies the famous Expectation-maximization (EM) algorithm. The alternating back-propagation iterates the following two steps. (a) Inferential back-propagation, which infers the latent factors by
Langevin dynamics or gradient descent. (b) *Learning back-propagation*, which updates the parameters given the inferred latent factors by gradient descent. The gradient computations in both steps are powered by back-propagation, and they share most of their code in common.

The learning algorithms of two models can be interwoven into a cooperative training algorithm, where the generator model generates synthesized examples to jump-start the Markov Chain Monte Carlo (MCMC) sampling of the descriptor model to fuel the learning of the descriptor model. The generator model can then learn from how the descriptors MCMC revises the synthesized examples generated by the generator, and the learning is supervised because the latent factors are known.

The experiment results show that the two models can generate realistic images, audios and dynamic patterns. Moreover, the generator can also be used to learn from incomplete or indirect training data. Generator and cooperative training algorithm can outperform generative adversarial network (GAN) and variational auto-encoder (VAE) in data recovery and incompletion tasks.
The dissertation of Yang Lu is approved.

Demetri Terzopoulos

Frederic Paik Schoenberg

Ying Nian Wu

Song-Chun Zhu, Committee Chair

University of California, Los Angeles

2017
# TABLE OF CONTENTS

1 Introduction .......................................................... 1
   1.1 Discriminative vs Generative .................................... 1
   1.2 Two directions of statistical models ........................... 3
      1.2.1 Descriptor as generalized exponential family model .... 3
      1.2.2 Generator as generalized latent factor models .......... 5
      1.2.3 Langevin dynamics sampling .............................. 7
      1.2.4 Cooperative learning .................................... 8
   1.3 Background ....................................................... 11
      1.3.1 ConvNet as GLM ........................................... 11
      1.3.2 Exponential family model ................................ 13
      1.3.3 Latent factor models ................................... 14
   1.4 Related work ..................................................... 15
      1.4.1 Hierarchical generative models .......................... 15
      1.4.2 Intractable inference ..................................... 16
      1.4.3 Connections between two classes of models ............ 16
   1.5 Contributions .................................................... 18

2 Descriptor ............................................................. 20
   2.1 Discriminative ConvNet as unfolded GLM ..................... 20
   2.2 Descriptor as a generalized exponential family model ....... 23
   2.3 Descriptor as a Markov random field and hierarchical FRAME model .... 25
   2.4 A prototype model ............................................. 27
   2.5 Representational structure of descriptor ...................... 31

v
3 Generator

3.1 Generator as generalized factor analysis model

3.2 ReLU and piecewise factor analysis

3.3 Alternating back-propagation

3.4 EM, density mapping, and density shifting

4 Cooperative learning of two models

4.1 CoopNets algorithm

4.2 Generator of infinite capacity

4.3 Generator of finite capacity

4.4 Generator and MCMC

5 Experiment

5.1 Qualitative experiment

5.1.1 Modeling texture patterns

5.1.2 Modeling object patterns

5.1.3 Synthesis and reconstruction by descriptor

5.1.4 Modeling sound patterns with 1D generator

5.1.5 Modeling dynamic patterns with 3D generator

5.2 Quantitative experiment

5.2.1 Learning from incomplete data

5.2.2 Learning generator from indirect data

5.2.3 Pattern reconstruction and completion
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Generating texture patterns. The first image is the training image, and the rest are 2 of the images generated by the learning algorithm.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Generating object patterns. The first row displays 4 of the training images, and the second row displays 4 of the images generated by the learning algorithm.</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>The flow chart of Algorithm D for training the descriptor net. 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.</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>The flow chart of Algorithm G for training the generator net. 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.</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>The flow chart of the CoopNets algorithm. The part of the flow chart for training the descriptor is similar to Algorithm D in Fig. 1.3 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 Fig. 1.4 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).</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>Convolutional neural networks consist of multiple layers of filtering and sub-sampling operations for bottom-up feature extraction, resulting in multiple layers of feature maps and their sub-sampled versions. The top layer features are used for classification via multinomial logistic regression. The discriminative direction is from image to category, whereas the generative direction is from category to image.</td>
<td>21</td>
</tr>
</tbody>
</table>
2.2 (a): Applying a filter of the size $3 \times 3 \times 3$ on an image of the size $6 \times 6 \times 3$ to get a filtered image or feature map of $6 \times 6$ (with proper boundary handling). Each pixel of the filtered image is computed by the weighted sum of the $3 \times 3 \times 3$ pixels of the input image centered at this pixel. There are 3 color channels (R, G, B), so both the input image and the filter are three-dimensional. (b) Rectified Linear Unit (ReLU) non-linear activation function.

2.3 Multi-grid learning. The learning and sampling consist of multiple stages. In each stage, a descriptor is used to model the conditional distribution given the previous stage samples.

2.4 Generated images from multi-grid learning. From left to right are generated $4 \times 4$, $16 \times 16$ and $64 \times 64$ images respectively.

4.1 The learning of the generator alternates between Markov transition and projection. The family of the generator models $G$ is illustrated by the black curve. Each distribution is illustrated by a point.

5.1 Generating texture patterns with descriptor. For each category, the first image is the training image, and the rest are 2 of the images generated by the descriptor.

5.2 Modeling texture patterns with generator. For each example, Left: the $224 \times 224$ observed image. Right: the $448 \times 448$ generated image by the generator.

5.3 Generating object patterns with descriptor. For each category, the first row displays 4 of the training images, and the second row displays 4 of the images generated by the descriptor.

5.4 Modeling object patterns with generator. Left: the synthesized images generated by our method. They are generated by $g(X; W_G)$ with the learned $W$, where $X = (x_1, x_2) \in [-2, 2]^2$, and $X$ is discretized into $9 \times 9$ values. Right: the synthesized images generated using Deep Convolutional Generative Adversarial Net (DCGAN). $X$ is discretized into $9 \times 9$ values within $[-1, 1]^2$. 
5.5 Modeling object patterns with generator. *Left:* each image generated by our method is obtained by first sampling \( X \sim N(0, I_{100}) \) and then generating the image by \( g(X; W_G) \) with the learned \( W \). *Middle:* interpolation. The images at the four corners are reconstructed from the inferred \( X \) vectors of four images randomly selected from the training set. Each image in the middle is obtained by first interpolating the \( X \) vectors of the four corner images, and then generating the image by \( g(X; W_G) \). *Right:* the synthesized images generated by DCGAN, where \( X \) is a 100 dimension vector sampled from uniform distribution.

5.6 Images generated by CoopNets learned from 10 Imagenet scene categories. The training set consists of 700 images randomly sampled from each category.

5.7 Images generated by CoopNets (top) and GAN (bottom). \( M \) is the number of training images randomly sampled from each of the 10 Imagenet scene categories.

5.8 Images generated by CoopNets trained on all the images from a single Imagenet category: school bus.

5.9 Reconstruction by the one-step contrastive divergence. The first row displays 4 of the training images, and the second row displays the corresponding reconstructed images.

5.10 Modeling sound patterns. *Row 1:* the waveform of the training sound (the range is 0-5 seconds). *Row 2:* the waveform of the synthesized sound (the range is 0-11 seconds).

5.11 Synthesized dynamic sequences using three versions of the generator model. For each category, the first row displays frames of the training sequence, and the second row displays the frames of the generated sequence. The first category shows the synthesized sequence generated by model-1 with the original length. Category 2 to category 6 display the synthesized sequences generated by model-2 with doubled length. The last category displays the generated sequence by model-3.
5.12 The top four rows show the comparison results for flame sequence. The bottom four rows show the results for moving grid sequence. For each category: First row: 8 frames of the training sequence. Second row: 8 frames of the generated sequence using our method. Third row: 8 frames of the generated sequence using 2D generator with an auto-regressive model for latent factors. Fourth row: 8 frames of generated sequence using [DCW03].

5.13 First row: bark water training sequence. Second row: frames from the generated sequences using our method. Third row: frames from the generated sequences using 3D GAN [VPT16].

5.14 Learning from incomplete data. The 10 columns belong to experiments P.5, P.7, P.9, P.9, P.9, P.9, P.9, M20, M30, M30 respectively. Row 1: original images, not observed in learning. Row 2: training images. Row 3: recovered images during learning.

5.15 Learning from indirect data. Row 1: the original 64 × 64 × 3 images, which are projected onto 1,000 white noise images. Row 2: the recovered images during learning.

5.16 Comparison between our method and PCA. Row 1: original testing images. Row 2: reconstructions by PCA eigenvectors learned from training images. Row 3: reconstructions by the generator learned from training images. \( d = 20 \) for both methods.

5.17 Interpolation results. The first row displays the occluded training sequence. The second row displays the interpolation results by our method. The third row shows the interpolation results by using Laplacian heat diffusion.
# LIST OF TABLES

5.1 Inception scores of different methods on learning from 10 Imagenet scene categories. \( M \) is the number of training images randomly sampled from each category.

5.2 Inception scores with different numbers of Langevin steps in learning.

5.3 Recovery errors in 5 experiments of learning from occluded images.

5.4 Recover errors for various incomplete training videos with different occlusion masks.

5.5 Reconstruction errors on testing images, after learning from training images using our method (ABP) and PCA.

5.6 Recovery errors in 3 experiments of learning from compressively sensed images.

5.7 Interpolation errors for various training videos with different occlusion masks.

5.8 Recovery errors for various incomplete testing videos with different occlusion masks.
ACKNOWLEDGMENTS

Firstly I would like to express my deepest appreciation to my advisor, Professor Song-Chun Zhu, who provides me the cherish opportunity to conduct the fascinating research. I met Professor Zhu at Lotus Hill Research Institute in China, where I start my research life in computer vision. The Lotus Hill Research Institute is an amazing place for young students to study the most cutting-edge problems in China with patient guidance from Professor Zhu. He continually and convincingly conveyed a spirit of adventure in regard to research and scholarship. Without his encouragement and support, this dissertation would not have been possible.

I feel very fortunate to work with Professor Ying Nian Wu, who enlighten me the beauty of research. I learned from him to conduct research rigorously and think principally. I also learn from you to be humble and kind to others, and indifferent to fame and gains while doing research. My gratitude also goes to my other thesis committee members, Professors Frederic Paik Schoenberg and Demetri Terzopoulos for their encouragement and insightful comments.

I feel very grateful to work with my excellent group members at VCLA. Many friends have helped me stay sane through these difficult years. Their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me. Particularly, I would like to thank to Tianfu Wu for beining wonderful collaborators with me.

I am also indebted to Hang Qi, Dan Xie, and Yuanlu Xu who work for DARPA MSEE project together. I can say I was very fortunate to have the help from them to solve the difficult problem together and it was very fun to work together.

Most importantly, none of this would have been possible without the love and patience of my family. My immediate family to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to thank my parents for giving me a lifelong place to call home and my beloved wife for sharing every moment with me.
VITA

2009 B.S. (Computer Science), Beijing Institute of Technology.

2012 M.S. (Computer Science), Beijing Institute of Technology.

2016-2017 Teaching Assistant, Statistics Department, UCLA.

2012–2017 Research Assistant, Statistics Department, UCLA.

PUBLICATIONS


CHAPTER 1

Introduction

1.1 Discriminative vs Generative

Fueled by the big datasets such as ImageNet [DDS09] and improved computer power brought by the graphical processing units (GPUs), the ConvNets [LBB98] [KSH12] have recently become the most successful discriminative or predictive learning machine.

The turning event for the resurgence of the ConvNet was its resounding victory in a competition on the ImageNet dataset [DDS09] in 2012. The ImageNet dataset was first released in 2009. Starting from 2010, there has been an annual competition on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [RDS14]. One of the tasks is image classification, which is to assign each image to an object category. There are roughly 1.2 million training images, 50,000 validation images, and 100,000 testing images, from 1,000 object categories. In ILSVRC 2012, the ConvNet [KSH12] became the runaway winner of the image classification competition. The winning network has 60 million parameters and 650,000 hidden nodes. It consists of 5 convolutional layers (some of them are followed by sub-sampling and max-pooling layers) and 3 fully-connected layers. Since then, the ConvNet and related deep learning methods have been adopted for many tasks in artificial intelligence, such as those in computer vision, speech recognition, natural language processing, etc., and have achieved state of the art performances, sometimes super-human performances, on these tasks.

The ConvNet is a discriminative or predictive learning machine, where the input to the ConvNet is an image. The output is an object category. The discriminative direction is from image to object category. Such a direction is often called the bottom-up direction.
The ConvNet tells us *how* to discriminate between, say, a bird and a cat. It does not tell us *what* a bird looks like or *what* a cat looks like. Such knowledge is most naturally represented by the generative direction, which is from object category to image. This direction is often called the top-down direction. The generative direction can be mathematically defined by a probability distribution on the image space, or a random field model. We can learn such a statistical model from training images. If we sample from the learned model, we can generate or synthesize new images. We may intuitively consider the sampling process as a matter of imagination, dreaming, or fantasizing, which is a gift that is obviously possessed by a human brain.

Developing generative models and representations is not only important for making the model understandable, it is also of fundamental importance for unsupervised learning, where we are only given unlabeled images without knowing their object categories, because the labeled images may be scarce and expensive to obtain. The generative models enable us to learn the parameters by explaining the image data instead of predicting the object categories.

Figure 1.1: Generating texture patterns. The first image is the training image, and the rest are 2 of the images generated by the learning algorithm.

Despite the successes of the discriminative learning machines such as ConvNet, the progress on developing generative models is still lagging behind. In my thesis, I study how to generalize the two major classes of statistical models to generative models by utilizing the ConvNets. Both of the two generative models can indeed generate surprisingly realistic image patterns. Figs. 1.1 and 1.2 display two examples. In Fig. 1.1 the first image is a single observed training image. The rest are 2 of the images sampled from the learned descriptor.
Figure 1.2: Generating object patterns. The first row displays 4 of the training images, and the second row displays 4 of the images generated by the learning algorithm. In Fig. 1.2, the first row displays 4 of the 7 training images. The second row displays 4 of the images sampled from the learned model. Intuitively, the learned models tell us what ivy leaves look like and what an egret looks like.

1.2 Two directions of statistical models

In statistics, there are major classes of models. (1) Exponential family model. The probability distribution is defined on some statistical features extracted from data. Such a model is a directed model, since the probability distribution is directly defined on data or its transformation. (2) Latent factor models. Such models are also called undirected models, since the data is generated by some latent factors, and the probability distribution is jointly defined on the data and latent factors. The two classes of models can be generalized by two types of ConvNets.

1.2.1 Descriptor as generalized exponential family model

The exponential family model can be generalized by a bottom-up ConvNet, where the statistical features are computed by multilayer linear convolution operations followed by non-linear activations. Such a model is named descriptor, since the model is built on descriptive statistical features computed by the bottom-up ConvNet. The traditional exponential family
Figure 1.3: The flow chart of Algorithm D for training the descriptor net. 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.

The model can be viewed as a single layer descriptor without non-linear activation functions.

The descriptor has no latent factors and therefore has no explicit inference. However a ConvNet contains non-linear activation functions such as rectified linear units, which gives the descriptor an implicit inference by computing the binary activation patterns. The linear convolution and binary activation patterns also provide a representational structure for descriptor.

The descriptor can be learned by maximizing likelihood estimation (MLE), and the gradient to update the model can be computed by matching statistical features from observed data and its expectation from model. It is intractable to obtain an analytical form of computing expectation in descriptor. Instead the expectation is approximated by Markov Chain Monte Carlo (MCMC) sampling.

The training algorithm for the descriptor alternates between the following two steps [XLZ16]. We call it Algorithm D. See Fig. 1.3 for an illustration.

**Step D1 for Langevin revision:** Sampling synthesized examples from the current
model by Langevin dynamics [Nea11, GC11]. 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 net based on the difference between the observed examples and the synthesized examples obtained in D1. 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.

In Step D1, the descriptor net is dreaming by synthesizing images from the current model. In Step D2, the descriptor updates its parameters to make the dream more realistic.

### 1.2.2 Generator as generalized latent factor models

The latent factor models can be generalized by a top-down ConvNet, in which the convolution filters are replaced by deconvolution filters. The data are generated by non-linear basis functions, and these basis functions are defined by a ConvNet computed on latent factors. Such a model is named generator, since the data is generated from transformations of latent factors. The traditional latent factor models can be viewed as a single layer generator without non-linear activation functions.

The generator network is a fundamental representation of knowledge, and it has the following properties: (1) **Analysis:** The model disentangles the variations in the observed signals into independent variations of latent factors. (2) **Synthesis:** The model can synthesize new signals by sampling the factors from the known prior distribution and transforming the factors into the signal. (3) **Embedding:** The model embeds the high-dimensional non-Euclidean manifold formed by the observed signals into the low-dimensional Euclidean space of the latent factors, so that linear interpolation in the low dimensional factor space results in non-linear interpolation in the data space.

The generator can be learned by alternating back-propagation, which underlie the famous EM algorithm. The expectation of complete log-likelihood are approximated by MCMC.
The learning algorithm iterates the following two steps: (1) Back-propagation for inference to compute latent factors given input data. (2) Back-propagation for updating model parameters as a non-linear multivariate regression problem. The both of two back-propagations share the same computation.

The training algorithm for the generator net alternates between the following two steps [HLZ17]. We call it Algorithm G. See Fig. 1.4 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 variables to explain or reconstruct the observed examples.

**Step G2 for reconstruction**: Updating the parameters of the generator net based on the observed examples and their inferred latent factors obtained in G1, so that the inferred latent variables can better reconstruct the observed examples.

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

The training algorithm for generator net is similar to the EM algorithm [DLR77], where step G1 can be mapped to the E-step, and step G2 can be mapped to the M-step. It is an unsupervised learning algorithm because the latent factors are unobserved.

In Step G1, the generator net 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, the generator net updates its parameters to make the thinking more accurate.

Compared to Step D2, Step G2 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.

1.2.3 Langevin dynamics sampling

Both of the two models leverage sampling to approximate the expectation. The descriptor samples data directly, while the generator samples latent factors from posterior distribution given input data.

The Langevin dynamics [Lin08] can be employed to synthesize images by sampling from the descriptor. Interestingly, the dynamics is driven by the reconstruction error, i.e., the difference between the current image and the reconstruction by the binary activation variables mentioned above. Thus image synthesis and image reconstruction are connected.

The Langevin dynamics can also be applied to infer latent factors of a generator. The resulting inference is named explaining-away inference, since the inference process always explains the residual of reconstruction error between training images and generated images from a generator.

The deterministic gradient descent counterpart of the Langevin dynamics was employed by [ZM97] for exploring the local energy minima of the FRAME model. They called it the Gibbs Reaction And Diffusion Equation (GRADE). It defines a dynamics that converges
to a local energy minimum. In the descriptor, the local energy minima are the means of the Gaussian pieces, and they are auto-encoding via the aforementioned binary activation variables. Thus the descriptor hides an auto-encoder at its energy minima. This observation establishes a connection between the Hopfield network for memory [Hop82] and the auto-encoder. In the generator, the Langevin sampling process is an inferential back-propagation, which solves an inverse problem by an explaining-away process, where the latent factors compete with each other to explain each training example.

1.2.4 Cooperative learning

Both the descriptor and the generator can be learned from the training examples by maximum likelihood and they can benefit from cooperating with each other, where the generator net plays the role of the student, and the descriptor net plays the role of the advisor.

The needs for cooperation stem from the fact that the Langevin sampling in Step D1 and Step G1 can be time-consuming. If the two algorithms cooperate, they can jumpstart each other’s Langevin sampling in D1 and G1. The resulting algorithm, which we call the CoopNets algorithm, naturally and seamlessly interweaves the steps in the two algorithms with minimal modifications.

In more plain language, while the descriptor needs to dream hard in Step D1 for synthesis, the generator needs to think hard in Step G1 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, while the descriptor does not need to think in order to learn.

Specifically, we have the following steps in the CoopNets algorithm:

**Step G0 for generation**: Generate the initial synthesized examples using the generator net. These initial examples can be obtained by direct ancestral sampling.

**Step D1 for Langevin revision**: Starting from the initial synthesized examples produced in Step G0, run Langevin revision dynamics for a 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 to shift the density of the descriptor towards the observed examples.

**Step G1 for Langevin inference:** The generator net can learn from the revised synthesized examples produced by Step D1. For each revised synthesized example, we know the values of the latent factors that generate the corresponding initial synthesized example in Step G0, therefore we may simply infer the latent factors to be their known values given in Step G0, or initialize the Langevin inference dynamics in Step G1 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 to update the generator. The generator in Step G0 generates and thus reconstructs the initial synthesized examples. Step G2 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.

Fig. 1.5 shows the flow chart of the CoopNets algorithm. The generator is like the student. It generates the initial draft of the synthesized examples. The descriptor is like the advisor. 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. The generator learns from how the descriptor revises the initial draft by reconstructing the revised draft. Such is the tale of two nets.

The reason we let the generator 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 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 variables.
Figure 1.5: The flow chart of the CoopNets algorithm. The part of the flow chart for training the descriptor is similar to Algorithm D in Fig. 1.3 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 Fig. 1.4 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.

True to the essence of learning, the two nets still learn from the data. The descriptor learns from the observed data and teaches the generator through the synthesized data. Specifically, they collaborate and communicate with each other through synthesized data in the following three versions (S stands for synthesis). (S1) The generator generates the initial draft. (S2) The descriptor revises the draft. (S3) The generator reconstructs the revised draft.

1.3 Background

In this section, I will briefly introduce terminologies and techniques in this paper.

1.3.1 ConvNet as GLM

A ConvNet defines a non-linear mapping function \( Z = f(Y; W) \), where both \( Z \) and \( Y \) are high dimensional data, \( W \) denotes parameters, and \( f(\cdot) \) is a \( L \) layers of linear transformation followed by a element-wise non-linear activation function.

\[
Y^{(l)} = f_l\left(W_l Y^{(l-1)} + b_l\right),
\]

(1.1)

where \( l = 1, 2 \cdots, L \), \( Y^{(0)} = Y \), \( Y^{(L)} = Z \), and \( W = (W_l, b_l) \). The \( W_l \) is the weight matrix, and \( b_l \) is the bias term, and \( f_l \) is the element-wise nonlinear transformation.
A ConvNet can be considered an unfolded version of the generalized linear model (GLM). One may even say that a feedforward neural network is a GLM on steroid. A GLM, such as logistic regression, is characterized by a composition of a linear combination or weighted sum of the predictor variables and a one-dimensional non-linear link function. A feedforward neural network or multi-layer perceptron is simply a recursion of such a compositional scheme, where each predictor variable itself is defined by a non-linear link function of a linear combination or weight sum of predictor variables at the lower layer. The predictor variables at the bottom layer are the raw input variables. In the terminology of neural networks, each predictor variable at each layer is called a unit, a node, a feature, or a filter. The neural network is said to be able to learn multiple layers of features instead of handcrafting them based on the domain knowledge.

A GLM is characterized by a weighted sum of input variables followed by a one-dimensional non-linear link function. A ConvNet, often applied to image, video or speech data, unfolds the GLM structure along two directions. (1) Convolutional: the weighted sum is computed locally around each pixel of the image, mapping an input image to an output image called the filtered image or the feature map. The operation is called convolution or linear filtering. See Fig. 2.2a for an illustration of linear filtering. (2) Hierarchical: there are multiple layers of linear filtering and element-wise non-linear transformation, as well as sub-sampling that makes the filtered images smaller. After a number of layers, the feature maps are reduced to $1 \times 1$ due to repeated sub-sampling. The final layer of features are then used for classification via a multinomial logistic regression.

The convolutional and hierarchical unfolding of the GLM gives rise to a ConvNet structure that is both simple and rich. It defines a rich class of functions that map an image to an object category. The functions are parametrized by the multiple layers of weight and bias parameters, which correspond to the coefficient and intercept parameters of GLMs. The element-wise non-linear transformation corresponds to the link function of GLM. In the modern ConvNet, the non-linear transformation usually takes the form of Rectified Linear Unit (ReLU).

While the GLM can be considered the statistical grounding of ConvNet, the Markov
random field (MRF) or equivalently the Gibbs distribution \cite{Bes74, GG86} can be considered the statistical grounding of the generative perspective of ConvNet that we shall explore in this paper. An MRF or a Gibbs distribution is a probability distribution defined on image. The log probability density or the energy function of a Gibbs distribution involves sum of functions defined on cliques, which are sets of pixels that are neighbors of each other. To connect to ConvNet, the clique functions can be defined by the ConvNet filters, and they can be learned from the data by maximum likelihood.

1.3.2 Exponential family model

An exponential family model has the form of following distribution,

$$P(Y; W) = \frac{1}{Z(W)} \exp \left\{ \sum_{k=1}^{d} w_k \phi_k(Y) \right\} q(Y), \quad (1.2)$$

where $\phi_k(Y)$ is sufficient statistics or features, $W = (w_k, k = 1)$ are the parameters, $q(Y)$ is the reference distribution, and $Z(Y) = \int \exp \left\{ \sum_{k=1}^{d} w_k \phi_k(Y) \right\} q(Y)$ is the normalizing constant. The exponential family models also include Markov random fields models (or the Gibbs distributions), such as Ising models, Potts model, etc. In the machine learning, they are called energy based models, or undirected models.

In computer vision, FRAME (Filters, Random field, And Maximum Entropy) models \cite{ZWM97, XHZ14} also have the form exponential family distribution. whose log probability density consists of compositions of linear filtering and element-wise non-linear transformation. The model is generative in the sense that images can be generated from the probability distribution defined by the model. The probability distribution is the maximum entropy distribution that reproduces the statistical properties of filter responses in the observed images. Being of the maximum entropy, the distribution is the most random distribution that matches the observed statistical properties of filter responses, so that images sampled from this distribution can be considered typical images that share the statistical properties of the observed images.

There are two versions of FRAME models in the literature. The original version is
a stationary model developed for modeling texture patterns [ZWM97], such as those in Fig. 1.1. The more recent version is a non-stationary extension designed to represent object patterns [XHZ14], such as those in Fig. 1.2. Both versions of the FRAME models can be sparsified by selecting a subset of filters from a given dictionary.

1.3.3 Latent factor models

Let $Y$ be a D-dimensional observed data vector. Let $X$ be the d-dimensional vector of continuous latent factors, $X = (x_k, d = 1, \cdots, d)$. The traditional factor analysis model is

$$Y = WX + \epsilon; X \sim N(I_d); \epsilon \sim N(0, \sigma^2 I_D), \quad (1.3)$$

where $W$ is $D \times d$ matrix, $\epsilon$ is a D-dimensional error vector or the observational noise, and $I_d$ stands for the d-dimensional identity matrix. There are three perspectives to view $W$. (1) Basis vectors. Write $W = (W_1, \ldots, W_d)$, where each $W_k$ is a $D$-dimensional column vector. Then $Y = \sum_{k=1}^{d} x_k W_k + \epsilon$, i.e., $W_k$ are the basis vectors and $x_k$ are the coefficients. (2) Loading matrix. Write $W = (w_1, \ldots, w_D)^\top$, where $w_j^\top$ is the $j$-th row of $W$. Then $y_j = \langle w_j, X \rangle + \epsilon_j$, where $y_j$ and $\epsilon_j$ are the $j$-th components of $Y$ and $\epsilon$ respectively. Each $y_j$ is a loading of the $d$ factors where $w_j$ is a vector of loading weights, indicating which factors are important for determining $y_j$. $W$ is called the loading matrix. (3) Matrix factorization. Suppose we observe $Y = (Y_1, \ldots, Y_n)$, whose factors are $X = (X_1, \ldots, X_n)$, then $Y \approx WX$.

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

1.4.1 Hierarchical generative models

The model in the form of exponential tilting of a reference distribution where the exponential tilting is defined by ConvNet was first proposed by [DLW15]. They learned the model by a non-parametric importance sampling scheme. The hierarchical energy-based models [LCH06] were studied by the pioneering work of [HOW06] and [NCK11]. However, their models do not correspond directly to the modern ConvNet.

A main motivation for the descriptor is to reconcile the FRAME model [ZWM97], where the Gabor wavelets play the role of bottom-up filters, and the Olshausen-Field model [OF97], where the wavelets play the role of top-down basis functions, and unfold these models into a hierarchical sparse compositional model [HSH13, XHZ14]. The descriptor may help solve this problem.

The representational structure in the descriptor is similar to but subtly different from the deconvolution network of [ZTF11]. The top-down process of the descriptor is controlled by multiple layers of binary activation variables computed by the bottom-up process. The descriptor can synthesize new images in addition to reconstructing observed images.

Compared to the hierarchical models with explicit binary latent variables such as those based on the Boltzmann machine [HOT06, SH09, LGR09], the descriptor is directly derived from the discriminative ConvNet.

While one type of model consists of exponential family models such as the FRAME model, the other consists of latent variable models such as the Olshausen-Field model. While the former class usually cannot reconstruct the observed data, the latter class typically needs to negotiate with the intractable inference. The descriptor has both explicit bottom-up pass for computing binary variables and explicit top-down pass for reconstructing the image. It strikes a middle ground between the two classes of models.
1.4.2 Intractable inference

As mentioned above, inference intractability is a big hurdle to learn a model with latent factor models. Existing training methods for the latent factor models avoid explain-away inference of latent factors. Two methods have recently been devised to accomplish this. Both methods involve an assisting network with a separate set of parameters in addition to the original network that generates the signals.

One way to get around the intractable inference problem mentioned above is to use the wake-sleep algorithm [HDF95] or the variational auto-encoder [KW14, RMW14, MG14]. The parameters in the top-down generation model and the bottom-up recognition model of a variational auto-encoder are completely separate from each other. In descriptor, there is a common set of parameters. More importantly, the descriptor actually contains an auto-encoder at the local minima of its energy landscape, and the encoding and decoding of this auto-encoder share the same set of parameters.

Another method is variational auto-encoder (VAE) [KW14, RMW14, MG14], where the assisting network is an inferential or recognition network that seeks to approximate the posterior distribution of the latent factors. The other method is the generative adversarial network (GAN) [GPM14, DCF15, RMC15], where the assisting network is a discriminator network that plays an adversarial role against the generator network.

Compared with the inexact inference method, the generator utilizes the Langevin dynamics to rigorously infer latent factors. The generator has tidy representational structure and exact inference algorithm.

1.4.3 Connections between two classes of models

While the GAN models learn a generator with the assistance of a discriminative model, the early work by [Hu07, WZH02] studies the connection between descriptor and discriminator. In [KB16], they study the connection between a descriptor and a generator. In their work, the generator is learned from the current descriptor by minimizing the Kullback-Leibler divergence from the generator to the current descriptor, which can be decomposed into an
energy term and an entropy term, with the latter approximated based on quantities produced by batch normalization [IS15a]. Their work avoids MCMC sampling, in particular, their work does not involve revising and reconstructing the synthesized examples, which plays a pivotal role in our method. In their work, the descriptor acts on the generator directly, whereas in our work, the descriptor acts on the generator through the synthesized data.

The two classes of models have been contrasted by [Zhu03, GZW03, TWO03, WZG08, NCK11]. [Zhu03, GZW03] called the two classes of models the descriptive models and the generative models respectively. Both classes of models can benefit from the two classes high capacity of the multi-layer ConvNets: (a) a bottom-up ConvNet which maps the signal to the features [NCK11, DLW13, LZW16, XLZ16] and (b) a top-down ConvNet [ZF14, DSB15] which maps the latent factors to the signal.

There is an interesting connection between cooperative learning and [KB16]: while [KB16] seeks to minimize the Kullback-Leibler divergence from the generator to the current descriptor, the Langevin revision dynamics in our work runs a Markov chain from the current generator to the current descriptor, and it automatically and monotonically decrease the Kullback-Leibler divergence from the distribution of the revised synthesized data to the distribution of the current descriptor (to zero in the time limit), which is an important property of the Markov chain that underlies the second law of thermodynamics and the so-called “arrow of time.” [CT12] The revised synthesized data are then fed to the generator for it to reconstruct the effect of the Langevin revisions. Therefore our learning method is consistent with that of [KB16]. In fact, the energy and entropy terms in the learning objective function in [KB16] correspond to the gradient and noise terms of the Langevin dynamics in our work, but the Langevin dynamics relieves us the need to approximate the intractable entropy term.

The CoopNets algorithm is related to the contrastive divergence algorithm [Hin02], generative stochastic network [TAY14], knowledge distilling [HVD15], and co-training [BM98]. While the contrastive divergence initializes the MCMC sampling from the observed examples, the CoopNets algorithm initializes the Langevin sampling from the examples supplied by the generator. While the generative stochastic network, the Coopnets absorbs the cumulative effect of all the past Markov transitions and reproduces it in one shot. While knowledge
distilling distills conditional classification knowledge by a smaller discriminative ConvNet, the CoopNets distills descriptor’s knowledge to the generator. While the co-training method learns two learners in the same direction, the CoopNets are of different directions, and they feed each other the synthesized data, instead of class labels.

1.5 Contributions

My thesis generalizes two classical statistical models with modern ConvNets, explores properties of models, and builds their connections. Specifically, the contributions are three-fold:

Contributions of descriptor. The descriptor as generalized exponential family model has following discoveries: (1) It can be derived from the discriminative ConvNet. (2) It contains an explicit representational structure. (3) It is piecewise Gaussian. (4) It can be sampled by a reconstruction driven algorithm. (5) Its local energy minima are auto-encoding. (6) The contrastive divergence learning of it tends to reconstruct the observed images. (7) It is capable of generating realistic image patterns. Our work on the descriptor may pave the way for unsupervised learning of ConvNet from large unlabeled datasets, which are essentially in unlimited supply without any cost for human annotation.

Compared to the work in literature, the conceptual novelty is, starting from a prototype model and then unfolding it, our work reveals a curious representational structure contained in the model that involves multiple layers of activation variables. Such a representational structure is unexpected for the exponential family models, and was not studied by the literature.

Contributions of generator. The main contribution of generator is to propose the alternating back-propagation algorithm for training the generalized latent factor model. Unlike alternating back-propagation, VAE does not perform explicit explain-away inference, while GAN avoids inferring the latent factors altogether. In comparison, the alternating back-propagation algorithm is simpler and more basic, without resorting to an extra network. While it is difficult to compare these methods directly, we illustrate the strength of alternating back-propagation by learning from incomplete and indirect data, where we only
need to explain whatever data we are given. This may prove difficult or less convenient for VAE and GAN.

Meanwhile, alternating back-propagation is complementary to VAE and GAN training. It may use VAE to initialize the inferential back-propagation, and as a result, may improve the inference in VAE. The inferential back-propagation may help infer the latent factors of the observed examples for GAN, thus providing a method to test if GAN can explain the entire training set.

**Contributions of their connections.** The CoopNets algorithm studies the connection between descriptor and generator.

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.
CHAPTER 2

Descriptor

2.1 Discriminative ConvNet as unfolded GLM

The generalized linear model (GLM) has two main components: (1) A weighted sum of input variables. (2) A non-linear link function. The discriminative ConvNet unfolds the GLM for analyzing image data (or video, speech data etc.)

A ConvNet is a composition of multiple layers of linear filtering and element-wise non-linear transformation as expressed by the following recursive formula:

$$Y^{(l)} = f_l(W_lY^{(l-1)} + b_l), \quad (2.1)$$

where $l \in \{1, 2, \ldots, L\}$ indexes the layer. $W_l = \{w_{l,k}, k = 1, \cdots, N_l\}$ are the multiple filters at layer $l$. The filters are locally supported, so the range is within a local support $S_l$ (such as a $7 \times 7$ image patch). The linear filtering operation is followed by a non-linear transformation $f_l()$. For notational simplicity, we do not make local max pooling explicit in (2.1).

See Fig. 2.1 for an illustration. The input image at the bottom layer has 3 color channels. The linear filtering operation at the bottom layer is illustrated by Fig. 2.2a. At each subsequent layer, the yellow squares illustrate the filtered images or multiple feature maps $Y^{(l)}$, and the blue squares illustrate their sub-sampled versions. There are multiple filtered images or feature maps produced by the multiple filters at each layer $l$. Just as the input image has 3 color channels, the multiple filtered images at each layer are also called multiple channels, where each filter corresponds to a channel at layer $l$. Each filter is illustrated by a set of green images. The filter response is a local weighted sum of pixel values around each
Figure 2.1: Convolutional neural networks consist of multiple layers of filtering and sub-sampling operations for bottom-up feature extraction, resulting in multiple layers of feature maps and their sub-sampled versions. The top layer features are used for classification via multinomial logistic regression. The discriminative direction is from image to category, whereas the generative direction is from category to image.

We take $f_l(r) = \max(r, 0)$, the Rectified Linear Unit (ReLU) that is commonly adopted in the modern ConvNet [KSH12]. See Fig. 2.2b for an illustration. This crisp piecewise linear transformation is the root of the binary activation variables and piecewise Gaussian form of the model. But some results in this paper can be extended to more general non-linearity.

Compared to the GLM, the weight parameters $W_l$ and the bias term $b_l$ corresponds to the coefficient and intercept parameters of the GLM, and the non-linear transformation $f_l(r)$ corresponds to the link function. The weighted sum takes place around each location $x$ and over multiple layers $l$, thus the ConvNet can be viewed as an unfolded GLM.

Let $W_{(L)}$ be the top layer filters. The filtered images are usually $1 \times 1$ due to repeated sub-sampling. Suppose there are $C$ categories. For category $c \in \{1, ..., C\}$, the scoring
Figure 2.2: (a): Applying a filter of the size $3 \times 3 \times 3$ on an image of the size $6 \times 6 \times 3$ to get a filtered image or feature map of $6 \times 6$ (with proper boundary handling). Each pixel of the filtered image is computed by the weighted sum of the $3 \times 3 \times 3$ pixels of the input image centered at this pixel. There are 3 color channels (R, G, B), so both the input image and the filter are three-dimensional. (b) Rectified Linear Unit (ReLU) non-linear activation function.
function for classification is

\[ f_c(Y; W_D) = W_L Y^{(L-1)}. \]  

(2.2)

**Definition 1** Discriminative ConvNet: We define the following conditional distribution as the discriminative ConvNet:

\[ p(c|Y; W_D) = \frac{\exp[f_c(Y; W_D) + b_c]}{\sum_{c=1}^{C} \exp[f_c(Y; W_D) + b_c]}, \]

(2.3)

where \( b_c \) is the bias term, and \( W_D \) collects all the weight and bias parameters at all the layers.

The discriminative ConvNet is a multinomial logistic regression (or soft-max) that is commonly used for classification [LBB98, KSH12].

### 2.2 Descriptor as a generalized exponential family model

We shall first define the descriptor.

**Definition 2** Descriptor (fully connected version): We define the following random field model as the fully connected version of the descriptor:

\[ p(Y|c; W_D) = p_c(Y; W_D) = \frac{1}{Z_c(W_D)} \exp[f_c(Y; W_D)]q(Y), \]

(2.4)

where \( q(Y) \) is a reference distribution or the null model, assumed to be Gaussian white noise in this paper. \( Z(W_D) = E_q\{\exp[f_c(Y; W_D)]\} \) is the normalizing constant.

In (2.4), \( p_c(Y; W_D) \) is obtained by the exponential tilting of \( q \), and is the conditional distribution of image given category, \( p(Y|c, W_D) \). The model is a generalization of exponential family model by replace the linear function with a ConvNet \( f_c(Y; W_D) \). The model was first proposed by [DLW15].

We show that the descriptor can be derived from the discriminative ConvNet.
Proposition 1  Descriptor and discriminative ConvNets can be derived from each other:

(a) Let $\rho_c$ be the prior probability of category $c$, if $p(Y|c; W_D) = p_c(Y; W_D)$ is defined according to model (2.4), then $p(c|Y; W_D)$ is given by model (2.3), with $b_c = \log \rho_c - \log Z_c(W_D) + \text{constant}$.

(b) Suppose a base category $c = 1$ is generated by $q(Y)$, and suppose we fix the scoring function and the bias term of the base category $f_1(Y; W_D) = 0$, and $b_1 = 0$. If $p(c|Y; W_D)$ is given by model (2.3), then $p(Y|c; W_D) = p_c(Y; W_D)$ is of the form of model (2.4), with $b_c = \log \rho_c - \log \rho_1 + \log Z_c(W_D)$.

Proposition 1 can be proved by a simple exercise of the Bayes rule. Result (a) has already been explained in [DLW15]. Result (b) is stronger and is new. First, it is entirely reasonable to include Gaussian white noise images as a base category and demand the discriminative ConvNet (2.3) not to misclassify the Gaussian white noise as an object category. It is also reasonable to fix the scoring function and bias term of this base category at 0 in training for the sake of identifiability. In fact, in the binary (two-category) logistic regression, the scoring function and the bias term for the negative category are always fixed at 0. Then

$$\frac{p(c|Y; W_D)}{p(c = 1|Y; W_D)} = \exp[f_c(Y; W_D) + b_c]. \quad (2.5)$$

Meanwhile

$$\frac{p(c|Y; W_D)}{p(c = 1|Y; W_D)} = \frac{p_c(Y; W_D)}{p_c(Y; W_D)} = \frac{\rho_c p_c(Y; W_D)}{\rho_1 q(Y)}, \quad (2.6)$$

because $p(c, Y; W_D) = p(c|Y; W_D)p(Y; W_D)$, where the marginal distribution $P(Y; W_D) = \sum_{c=1}^C \rho_c p(Y|c; W_D)$ is the mixture of all the categories. Thus $p_c(Y; W_D)$ is of the form of model (2.4).

As to learning, we may use the discriminative log-likelihood based on $\log p(c|Y; W_D)$, or we may use the generative log-likelihood based on $\log p(c, Y|W_D) = \log p_c(Y; W_D) + \log \rho_c$. Because $\log p(c, Y|W_D) = \log p(c|Y; W_D) + \log P(Y; W_D)$, the discriminative log-likelihood $\log p(c|Y; W_D)$ is without the marginal log-likelihood $\log P(Y; W_D)$, resulting in the loss of statistical efficiency.
If we only observe unlabeled data \( \{ Y_m, m = 1, \ldots, M \} \), we may still use the exponential tilting form to model and learn from them. A possible model is to learn filters at a certain convolutional layer \( L \in \{ 1, \ldots, L \} \) of a ConvNet.

**Definition 3** Descriptor (convolutional version or FRAME version): we define the following Markov random field model as the convolutional version or the FRAME version of descriptor:

\[
p(Y; W_\mathcal{D}) = \frac{1}{Z(W_\mathcal{D})} \exp \left[ f(Y; W_\mathcal{D}) \right] q(Y),
\]

(2.7)

where \( W_\mathcal{D} \) consists of all the weight and bias terms that define the filters, and \( q \) is the Gaussian white noise model.

Essentially the above model treats the images \( \{ Y_i \} \) as coming from a single meta-category, which is to be discriminated from the base category \( q \) by the filters \( W_\mathcal{D} \) to be learned from the data. However, it may be too easy to discriminate \( \{ Y_i \} \) from \( q \), so that we cannot learn anything meaningful by the discriminative log-likelihood. In this case, we can learn \( W_\mathcal{D} \) based on the generative log-likelihood \( L(W_\mathcal{D}) = \sum_{i=1}^n \log p(Y_i; W_\mathcal{D})/n \), with \( p(Y; W_\mathcal{D}) \) defined by (2.7). The learning of filters \( W_\mathcal{D} \) by the generative log-likelihood is considered to be unsupervised because the observed images are unlabeled, i.e., their categories are unknown.

### 2.3 Descriptor as a Markov random field and hierarchical FRAME model

Model (2.7) is a Markov random field model [Bes74], where the clique functions are \( W_l Y^{(l-1)} \), for all \( l = 1, \cdots, L \). According to the Hammersley-Clifford theorem [Bes74], a Markov random filed model can be written as

\[
p(Y) = \frac{1}{Z} \exp \left[ \sum_C \lambda_C(Y(C)) \right],
\]

(2.8)

where \( C \subset \mathcal{D} \) are the cliques, and each clique \( C \) consists of pixels that are neighbors of each other according to a pre-defined neighborhood system. \( Y(C) \) are the intensities of pixels in
clique $C$, and $\lambda_C$ is the potential function. The challenge in developing a Markov random field model is to specify the clique functions $\lambda_C$ and estimate them from the data. Model (2.7) solves this problem by assuming $\lambda_C(Y(C)) = W_l Y^{(l-1)}$ using the ConvNet filters. The Gaussian white noise $q(Y)$ contributes to cliques that consist of single pixels.

At the first glance, defining clique functions by the ConvNet filters may appear to be arbitrary and ad hoc, but it is actually based on a rich tradition in generative modeling. The first model in the literature that represents the clique functions by non-linear transformations of linear filter responses is the FRAME (Filters, Random field, And Maximum Entropy) model [ZWM97]. The descriptor (2.7) can be considered a hierarchical FRAME model, with alternating layers of linear filtering and non-linearity.

More importantly, the recursive form of equation (2.1) has an interesting justification by generative modeling. Based on filters $W_{l-1}$ at layer $l-1$, each filter $W_{l(k)}$ at layer $l$ corresponds to a non-stationary FRAME model [XHZ14, LZW16] of an image patch defined on the support of the filter:

$$p_k^{(l)}(Y; W_D) = \frac{1}{Z_k^{(l)}(W_D)} \exp \left[ \sum_{k=1}^{N_{l-1}} W_{l(k)} Y^{(l-1)} \right] q(Y),$$

where $W_{l(k)}$ are the parameters of the above exponential family model. The model is also a descriptor, and it can generate vivid object patterns [LZW16]. The bias term $b_{l,k}$ and the ReLU non-linearity $f_l()$ in equation (2.1) can be justified by a mixture model $P_k^{(l)}(Y; W_D) = \alpha p_k^{(l)}(Y; W_D) + (1 - \alpha)q(Y)$, which is a mixture of presence and absence of the object pattern modeled by the non-stationary FRAME model (2.9). Writing $P_k^{(l)}(Y; W_D) = \exp \left[ f_k^{(l)}(Y; W_D) \right] q(Y)$ gives rise to the soft-max non-linearity $\log(1 + e^r)$ that can be approximated by ReLU $\max(0, r)$, and the bias term $b_{l,k}$ that is determined by $\alpha$ and $Z_k^{(l)}(W_D)$. Finally, taking the product of $P_k^{(l)}(Y; W_D)$ over $k$ gives rise to a Product of Experts (PoE) model [Hin02], which is the descriptor (2.7) using filters at layer $l$. See [LZW16] for details.
2.4 A prototype model

We can explain the key properties of the descriptor by the simplest prototype model, which makes crystal clear most of the key elements of the descriptor model \([2.7]\). A similar model was studied by [XLZ15]. The descriptor can be obtained from the prototype model by unfolding the latter both convolutionally and hierarchically, but with much more involved notation that is in danger of obscuring the key ideas. Hence it is helpful to start from the prototype model.

In the prototype model, we assume that the image domain \(D\) is small (e.g., \(10 \times 10\)). Suppose we want to learn a dictionary of filters or basis functions from a set of observed image patches \(\{Y_i, i = 1, ..., n\}\) defined on \(D\). We denote these filters or basis functions by \((W(k), k = 1, ..., K)\), where each \(W(k)\) itself is an image patch defined on \(D\). Let \(W(k)Y\) be the inner product between image patches \(Y\) and \(W(k)\). It is also the response of \(Y\) to the linear filter \(W_k\).

**Definition 4** Prototype model: We define the following random field model as the prototype model:

\[
p(Y; Q) = \frac{1}{Z(W_D)} \exp \left[ \sum_{k=1}^{K} f_1(W(k)Y + b_k) \right] q(Y), \tag{2.10}
\]

where \(b_k\) is the bias term, \(W_D = (W(k), b_k; k = 1, ..., K)\), and \(f_1(r) = \max(r, 0)\). \(q(Y)\) is the Gaussian white noise model,

\[
q(Y) = \frac{1}{(2\pi\sigma^2|^D|/2)} \exp \left[ -\frac{1}{2\sigma^2} ||Y||^2 \right], \tag{2.11}
\]

where \(|D|\) counts the number of pixels in the domain \(D\).

The following are findings about the prototype model.

1. **Piecewise Gaussian and binary activation variables:** The prototype model \((2.10)\) is a piecewise Gaussian distribution. Without loss of generality, let us assume \(\sigma^2 = 1\) in \(q(Y)\). Define the binary activation variable \(\delta_k(Y; W_D) = 1\) if \(W(k)Y + b_k > 0\) and \(\delta_k(Y; W_D) = 0\) otherwise, i.e.,

\[
\delta_k(Y; W_D) = 1(W(k)Y + b_k > 0), \tag{2.12}
\]
where \( \mathbf{1}(\cdot) \) is the indicator function. Then

\[
f_1(W(k)Y + b_k) = \delta_k(Y;W_D)(W(k)Y + b_k).
\]

(2.13)

The image space is divided into \( 2^K \) pieces by the \( K \) hyper-planes, \( Y, W(k)Y + b_k = 0, k = 1, \ldots, K \), according to the values of the binary activation variables \( (\delta_k(Y;W_D), k = 1, \ldots, K) \). Consider the piece where \( \delta_k(Y;W_D) = \delta_k \) for \( k = 1, \ldots, K \). Here we abuse the notation slightly where \( \delta_k \in \{0, 1\} \) on the right hand side denotes the value of \( \delta_k(Y;W_D) \). Write \( \delta(Y;W_D) = (\delta_k(Y;W_D), k = 1, \ldots, K) \), and \( \delta = (\delta_k, k = 1, \ldots, K) \) as an instantiation of \( \delta(Y;W_D) \). We call \( \delta(Y;W_D) \) the activation pattern of \( Y \). Let \( A(\delta;W_D) = \{ Y : \delta(Y;W_D) = \delta \} \) be the piece of image space that consists of images sharing the same activation pattern \( \delta \), then the probability density on this piece

\[
p(Y;W_D, \delta) \propto \exp \left[ \sum_{k=1}^{K} \delta_kb_k + \sum_{k=1}^{K} \delta_k W(k)Y - \frac{\|Y\|^2}{2} \right]
\]

\[\propto \exp \left[ -\frac{1}{2} \|Y - \sum_{k=1}^{K} \delta_k W(k)\|^2 \right], \tag{2.14}\]

which is \( N(\sum_k \delta_k W(k), I_D) \) restricted to the piece \( A(\delta;W_D) \), where the \( I_D \) is the identity matrix (recall we assume \( \sigma^2 = 1 \)). \( \delta = (\delta_k) \) are the binary activation variables that reconstruct the mean of this Gaussian piece, \( \sum_k \delta_k W(k) \), which can be considered an approximated reconstruction of the images in \( A(\delta;W_D) \).

(2) Synthesis via reconstruction: One can sample from \( p(Y;W_D) \) in (2.10) by the Langevin dynamics:

\[
Y_{\tau+1} = Y_\tau - \frac{\epsilon^2}{2} \left[ Y_\tau - \sum_{k=1}^{K} \delta_k(Y_\tau;W_D)W(k) \right] + \epsilon Z_\tau, \tag{2.15}
\]

where \( \tau \) denotes the time step, \( \epsilon \) denotes the step size, assumed to be sufficiently small throughout this paper, and \( Z_\tau \sim N(0, 1) \). The dynamics is driven by the reconstruction error \( Y - \sum_k \delta_k W(k) \), where the reconstruction is based on the binary activation variables \( (\delta_k) \). This links synthesis to reconstruction.

(3) Auto-encoding local modes: The deterministic part of the dynamics \( Y_{\tau+1} = Y_\tau - \)

28
\[ \frac{1}{2} \left[ Y - \sum_{k=1}^{K} \delta_k(Y; W_D)W(k) \right] \]
will converge to a local energy minimum \( \hat{Y} \), where

\[ \hat{Y} = \sum_{k=1}^{K} \delta_k(\hat{Y}; W_D)W(k). \quad (2.16) \]

That is, \( \hat{Y} \) is auto-encoding. The encoding process is bottom-up and infers \( \delta_k = \delta_k(\hat{Y}; W_D) = 1(W(k)\hat{Y} + b_k > 0) \). The decoding process is top-down and reconstructs \( \hat{Y} = \sum_k \delta_k W(k) \).

In the encoding process, \( W(k) \) plays the role of filter. In the decoding process, \( W(k) \) plays the role of basis function. The local modes are the means of the Gaussian pieces mentioned above, but the converse if not true, since \( \hat{Y} \) may not belong to \( A(\delta; W_D) \).

The learning of \( W_D \) from training images \( \{Y_i, i = 1, ..., n\} \) can be accomplished by maximum likelihood. Define \( L(W_D) = \sum_{m=1}^{M} \log p(Y; W_D)/n \), with \( p(Y; W_D) \) defined in (2.10), then

\[
\frac{\partial L(W_D)}{\partial W(k)} = \frac{1}{n} \sum_{i=1}^{n} \delta_k(Y_i; W_D)Y - E_{W_D}[\delta_k(Y; W_D)Y],
\]

\[
\frac{\partial L(W_D)}{\partial b_k} = \frac{1}{n} \sum_{i=1}^{n} \delta_k(Y_i; W_D) - E_{W_D}[\delta_k(Y; W_D)],
\quad (2.17)
\]

where \( E_{W_D} \) is the expectation with respect to \( p(Y; W_D) \), and can be approximated by Monte Carlo samples produced by the Langevin dynamics. At the maximum likelihood estimate of \( W_D \), the model matches the observed images in terms of (1) the frequency that \( \delta_k \) is on, and (2) the average of images on which \( \delta_k \) is on, for every \( k \).

The reference distribution is usually not emphasized in previous treatments of exponential family models. It plays a crucial role in our work. The Gaussian white noise reference distribution makes the density \( p(Y; W_D) \) in (2.10) integrable, and leads to the reconstruction error interpretation in the Langevin dynamics. It is also crucial for the auto-encoder form of the local modes.

The reason we choose Gaussian white noise model as the reference distribution is that it is the maximum entropy distribution with given marginal mean and variance. Thus it is the most featureless distribution. The exponential tilting seeks to explain the departure from the Gaussian distribution by learning non-Gaussian features.
Another justification for the Gaussian white noise distribution is that it is the limiting distribution if we zoom out a texture image due to the central limit theorem, a phenomenon called information scaling by [WZG08]. The exponential tilting is to recover the non-Gaussian distribution before the central limit theorem takes effect.

A key point is that the ReLU $f_1(r) = \max(0,r)$ can be written as $f_1(r) = \delta r$, where $\delta = 1(r > 0)$. Thus given the binary indicators, the scoring function $f(Y; W_D) = \sum_{k=1}^K f_1(W(k)Y + b_k)$ is piecewise linear in $Y$. Together with the Gaussian white noise term $\|Y\|^2/2$, the probability distribution is piecewise Gaussian. The ReLU non-linearity is a perfect match to the Gaussian white noise $q$. The combination of the two gives us a simple and rich structure.

We would also like to compare the prototype model with two strategies for developing generative models. Strategy 1: Treat $\delta_k$ as latent random variables. We put a prior distribution on $(\delta_k, k = 1, \ldots, K)$, usually assumed to be independent of each other, and then let $(\delta_k)$ generate $Y$, e.g., $Y \sim N(\sum_k \delta_k W(k), \sigma^2 I_D)$. The model has an explicit representation of $Y$ in terms of $\sum_k \delta_k W(k)$, where $(W(k))$ play the role of basis functions for representing $Y$. The problem with such a model is that the posterior distribution of $\delta_k$ given $Y$ is usually not in closed form. In fact, inferring $(\delta_k)$ from $Y$ given the basis functions $(W(k))$ is a variable selection problem, where $Y$ is the response variable, and $(W(k))$ are the predictor variables. The difficulty with the inference of $(\delta_k)$ makes it difficult to learn $W_D$. Strategy 2: Assume an exponential family model based on the sufficient statistics or feature statistics $f_1(W(k)Y)$, where $(W(k))$ play the role of linear filters or projections. The problem with such a model is that in general we do not have an explicit representation or reconstruction of the image $Y$. However, with ReLU $f_1()$ and Gaussian white noise $q$, we can elucidate an explicit representation or reconstruction as explained above. Thus our method strikes a middle ground between the above two strategies. Such a representation is not only conceptually appealing, it can also be important for learning, where we may seek to minimize the reconstruction error instead of maximizing the log-likelihood, which requires MCMC sampling. In Section 2.6, we shall see that a popular variation of the maximum likelihood learning algorithm called contrastive divergence actually seeks to minimize the reconstruction error.

In Strategy 1, it is possible to make the posterior distribution of $(\delta_k)$ given $Y$ explicit
at the expense of an implicit prior distribution on \((\delta_k)\), such as in the restricted Boltzmann machine \[\text{HOT06, SH09, LGR09}\]. However, in the hierarchical generalizations such as the deep Boltzmann machine, the posterior of the binary variables becomes intractable \[\text{SH09}\]. In contrast, the prototype model can be easily generalized to a hierarchical model without such a difficulty as we shall see in the next section.

The binary activation variables \((\delta_k)\) indicate the selection of the basis functions from \((W(k))\) for representing \(Y\). In the prototype model, the selection is based on \(\delta_k = 1(W(k)Y + b_k > 0)\). It is possible to generalize the definition of \(\delta_k\) to incorporate explaining-away competition, e.g., among highly correlated basis functions, only the one with the biggest \(|W(k)Y|\) should be selected. Thus writing the model in terms of the binary activation variables may lead to interesting generalizations, which is another advantage of our model.

### 2.5 Representational structure of descriptor

Just as the discriminative ConvNet can be considered an unfolded version of the GLM, the descriptor can be considered an unfolded version of the prototype model. In order to generalize the prototype model \(2.10\) to the descriptor \(2.7\), we only need to add two elements: (1) Horizontal unfolding: make the filters \((W(k))\) convolutional. (2) Vertical unfolding: make the filters \((W(k))\) multi-layer or hierarchical. The results we have obtained for the prototype model can be unfolded accordingly.

Following the analysis of the prototype model, the next plan is to write the scoring function \(f(Y; W_D) = W_l Y^{(l−1)}\) of the descriptor \(2.7\) as a piecewise linear function of \(Y\). We shall use the vector notation for the ConvNet in order to minimize the use of indices (although there are still plenty remaining). For filters at level \(l\), the \(N_l\) filters are denoted by the compact notation \(W_l = (W_l(k, x), k = 1, ..., N_l, x \in S_l)\). In the vector notation, the recursive formula \(2.1\) of ConvNet filters can be rewritten as,

\[
[Y^{(l)}](k, x) = f_l \left( W_l(k, x)Y^{(l−1)} + b_{l,k} \right), \tag{2.18}
\]
where $W_l(k, x)$ matches the dimension of $Y^{(l-1)}$, which is a 3D image containing all the $N_{l-1}$ filtered images at layer $l-1$.

The 3D basis function $W_l(k, x)$ is locally supported, and $W_l(k, x)$ are spatially translated copies for different positions $x$. $W_l(k, x)$ is the unfolded version of $W(k)$ in the prototype model, where $x$ indexes the position for convolutional unfolding, and $l$ indexes the layer for hierarchical unfolding.

Define the binary activation variable

$$\delta_{k,x}^{(l)}(Y; W_D) = 1 \left( W_l(k, x)Y^{(l-1)} + b_{l,k} > 0 \right).$$

Since $Y^{(l-1)}$ corresponds to a non-stationary FRAME model (2.9), $\delta_{k,x}^{(l)}(Y; W_D)$ is a decision maker based on the likelihood ratio test of $H_1 : p_k^{(l)}(Y; W_D, x)$ vs $H_0 : q(Y)$ for detecting the pattern modeled by $W_l(k, x)$. See [MPC14] for an analysis of the number of linear pieces.

According to (2.1), we have the following bottom-up process:

$$\left[ Y^{(l)} \right] (k, x) = \delta_{k,x}^{(l)}(Y; W_D) \left( W_l(k, x)Y^{(l-1)} + b_{l,k} \right).$$

Let $\delta(Y; W_D) = (\delta_{k,x}^{(l)}(Y; W_D), \forall k, x, l)$ be the activation pattern at all the layers. The activation pattern $\delta(Y; W_D)$ can be computed by the bottom-up process (2.19) and (2.20) of the ConvNet.

According to equations (2.20), we have the following top-down process:

$$B^{(l-1)} = B^{(l)}\delta^{(l)}(Y; W_D)W_l,$$

Equation (2.21) is a top-down deconvolution process, where $B^{(l)}\delta^{(l)}(Y; W_D)$ serves as the coefficient of the basis function $W_l$. The top-down deconvolution process (2.21) is similar to but subtly different from that in [ZF14], because equation (2.21) is controlled by the multiple layers of activation variables $\delta^{(l)}$ computed in the bottom-up process of the ConvNet. Specifically, $\delta^{(l)}$ turns on or off the basis function $W_l$, while $\delta^{(l)}$ is determined by $W_l$.

In the bottom-up convolution process (2.20), $W_l$ serve as filters. In the top-down deconvolution process (2.21), $W_l$ serve as basis functions.
Theorem 1  Descriptor is piecewise Gaussian: With ReLU $f_i(r) = \max(0, r)$ and Gaussian white noise $q(Y)$, $p(Y; W_D)$ of model (2.7) is piecewise Gaussian. On each piece $A(\delta; W_D)$, the density is $N(B_{W_D,\delta}, 1)$ truncated to $A(\delta; W_D)$, i.e., $B_{W_D,\delta}$ is an approximated reconstruction of images in $A(\delta; W_D)$.

Theorem 1 follows from the fact that on $A(\delta; W_D)$,

$$p(Y; W_D, \delta) \propto \exp \left[ \alpha_{W_D,\delta} + B_{W_D,\delta} Y - \frac{\|Y\|^2}{2} \right]$$
$$\propto \exp \left[ -\frac{1}{2} \|Y - B_{W_D,\delta}\|^2 \right],$$

which is $N(B_{W_D,\delta}, 1)$ restricted to $A(\delta; W_D)$.

For each $Y$, the binary activation variables in $\delta = \delta(Y; W_D)$ are computed by the bottom-up detection process (2.19) and (2.20), and $B_{W_D,\delta}$ is computed by the top-down deconvolution process (2.21).

One can sample from $p(Y; W_D)$ of model (2.7) by the Langevin dynamics:

$$Y_{\tau+1} = Y_{\tau} - \frac{\epsilon^2}{2} \left[ Y_{\tau} - B_{W_D,\delta(Y_{\tau}; W_D)} \right] + \epsilon Z_\tau,$$

where $Z_\tau \sim N(0, 1)$. Again, the dynamics is driven by the reconstruction error $Y - B_{W_D,\delta(Y_{\tau}; W_D)}$.

The deterministic part of the Langevin equation (2.23) was employed by [ZM97] for exploring the local modes of the FRAME model. They called it the Gibbs Reaction and Diffusion Equation (GRADE). The GRADE attractor dynamics $Y_{\tau+1} = Y_{\tau} - \frac{\epsilon^2}{2} \left[ Y_{\tau} - B_{W_D,\delta(Y_{\tau}; W_D)} \right]$ converges to a local energy minimum that is auto-encoding.

Proposition 2  The local modes are auto-encoding: Let $\hat{Y}$ be a local maximum of $p(Y; W_D)$ of model (2.7), then $\hat{Y}$ is auto-encoding with the following bottom-up and top-down passes:

Bottom-up encoding: $\delta = \delta(\hat{Y}; W_D);$  

Top-down decoding: $\hat{Y} = B_{W_D,\delta}.$

The local energy minima are the means of the Gaussian pieces in Theorem 1 but the reverse is not necessarily true because $B_{W_D,\delta}$ does not necessarily belong to $A(\delta; W_D)$. But if $B_{W_D,\delta} \in A(\delta; W_D)$, then $B_{W_D,\delta}$ must be a local mode.
Proposition 2 can be generalized to general non-linear $f_l()$, whereas Theorem 1 is true only for piecewise linear $f_l()$ such as ReLU.

Proposition 2 is interestingly related to the Hopfield network [Hop82] and attractor network [Seu98]. The main idea of the Hopfield network is to memorize the observations by the local energy minima. Such a memory is content addressable in the sense that if we are given part of an observed image, we may still be able to recall the whole image by running a gradient descent algorithm towards the local mode. Such a gradient descent algorithm is called an attractor dynamics. Proposition 2 shows that the Hopfield minima can be represented by a hierarchical auto-encoder.

2.6 Learning descriptor

The learning of $W_D$ from training images $\{Y_i, i = 1, ..., n\}$ can be accomplished by maximum likelihood. Let

$$L(W_D) = \frac{1}{n} \sum_{i=1}^{n} \log p(Y; W_D), \quad (2.25)$$

with $p(Y; W_D)$ defined in (2.7), then

$$\frac{\partial L(W_D)}{\partial W_D} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial W_D} f(Y_i; W_D) - E_{W_D} \left[ \frac{\partial}{\partial W_D} f(Y; W_D) \right]. \quad (2.26)$$

The expectation can be approximated by Monte Carlo samples [You99] from the Langevin dynamics (2.23). See Algorithm 1 for a description of the learning and sampling algorithm.

The above equations are unfolded versions of (2.17). At the maximum likelihood estimate of $W_D$, the model matches the observed images in terms of (1) the average frequency that the binary activation is on, and (2) the average of patches of the filtered image.

$$\frac{\partial L(W_D)}{\partial W_D} = \frac{1}{n} \sum_{i=1}^{n} \left[ \frac{\partial f(Y_i; W_D)}{\partial W_D} \right] - \frac{1}{n} \left[ \frac{\partial f(\tilde{Y}_i; W_D)}{\partial W_D} \right], \quad (2.27)$$

If we want to learn from big data, we may use the contrastive divergence method [Hin02] by starting the Langevin dynamics from the observed images, and run one or a small number of iterations. Then we use the sampled images to approximate the expectation in equation
Algorithm 1 Algorithm D

Require:

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

(2) number of Langevin steps \( l_D \)

(3) number of learning iterations \( T \)

Ensure:

(1) estimated parameters \( W_D \)

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

1: Let \( t \leftarrow 0 \), initialize \( W_D \).
2: Initialize \( \tilde{Y}_i, i = 1, ..., \tilde{M} \).
3: repeat
4: \textbf{Step D1 Langevin revision}: For each \( i \), run \( l_D \) steps of Langevin dynamics to update \( \tilde{Y}_i \), i.e., starting from the current \( \tilde{Y}_i \), each step follows equation [2.23].
5: \textbf{Step D2 density shifting}: Update \( W_D^{(t+1)} = W_D^{(t)} + \gamma_t L_D'(W_D^{(t)}) \), with learning rate \( \gamma_t \), where \( L_D'(W_D^{(t)}) \) is computed according to [2.27].
6: Let \( t \leftarrow t + 1 \)
7: until \( t = T \)
for updating the parameters. The contrastive divergence algorithm has become very popular for learning generative models and often lead to reasonable performances, although its theoretical properties are still not well understood. We shall show that the behavior of the contrastive divergence learning of the descriptor is interestingly connected to reconstruction.

Suppose we start from an observed image \( Y^{\text{obs}} \), and run a small number of iterations of Langevin dynamics (2.23) to get a synthesized image \( Y^{\text{syn}} \). If both \( Y^{\text{obs}} \) and \( Y^{\text{syn}} \) share the same activation pattern \( \delta(Y^{\text{obs}}; W_D) = \delta(Y^{\text{syn}}; W_D) = \delta \), then \( f(Y; W_D) = a_{W_D,\delta} + B_{W_D,\delta}Y \) for both \( Y^{\text{obs}} \) and \( Y^{\text{syn}} \), where \( a_{W_D,\delta} \) is the intercept term. Then the contribution of \( Y^{\text{obs}} \) to the learning gradient is

\[
\frac{\partial}{\partial w} f(Y^{\text{obs}}; W_D) - \frac{\partial}{\partial W_D} f(Y^{\text{syn}}; W_D) = \frac{\partial}{\partial W_D} B_{W_D,\delta} (Y^{\text{obs}} - Y^{\text{syn}}).
\]  

(2.28)

If \( Y^{\text{syn}} \) is close to the mean \( B_{W_D,\delta} \) and if \( B_{W_D,\delta} \) is a local mode, then the contrastive divergence tends to reconstruct \( Y^{\text{obs}} \) by the local mode \( B_{W_D,\delta} \), because the gradient

\[
\frac{\partial}{\partial W_D} \|Y^{\text{obs}} - B_{W_D,\delta}\|^2/2 = -\frac{\partial}{\partial W_D} B_{W_D,\delta} (Y^{\text{obs}} - B_{W_D,\delta} \).
\]

(2.29)

Hence the contrastive divergence learns the Hopfield network which memorizes the observations by the local modes.

We can establish a precise connection for one-step contrastive divergence.

**Proposition 3** Contrastive divergence learns to reconstruct: If the one-step Langevin dynamics does not change the activation pattern, i.e., \( \delta(Y^{\text{obs}}; W_D) = \delta(Y^{\text{syn}}; W_D) = \delta \), then the one-step contrastive divergence has an expected gradient that is proportional to the reconstruction gradient:

\[
E \left[ \frac{\partial}{\partial W_D} f(Y^{\text{obs}}; W_D) - \frac{\partial}{\partial W_D} f(Y^{\text{syn}}; W_D) \right] \propto \frac{\partial}{\partial W_D} \|Y^{\text{obs}} - Y_{W_D,\delta}\|^2.
\]

(2.30)

This is because

\[
Y^{\text{syn}} = Y^{\text{obs}} - \frac{\epsilon^2}{2} [Y^{\text{obs}} - B_{W_D,\delta}] + \epsilon Z,
\]

(2.31)

hence

\[
E_Z [Y^{\text{obs}} - Y^{\text{syn}}] \propto Y^{\text{obs}} - B_{W_D,\delta},
\]

(2.32)
and Proposition 3 follows from (2.28) and (2.29).

Proposition 3 is related to score matching estimator of [Hyv05], whose connection with the contrastive divergence based on one-step Langevin dynamics was studied by [Hyv07]. The relationship between score matching and auto-encoder was discovered by [Vin11] and [SRB11]. Our work can be considered a sharpened specialization of the above mentioned connection and relationship, where the piecewise linear structure of the ConvNet greatly simplifies the matter by getting rid of the complicated second derivative terms, so that the contrastive divergence gradient becomes exactly proportional to the gradient of the reconstruction error, which is not the case in general score matching estimator. Also, our work gives a novel hierarchical realization of the relationship between probability model and auto-encoder, as well as an explicit hierarchical realization of auto-encoder based sampling of [ABL4]. The connection with the Hopfied network also appears new.

Proposition 3 suggests that we may learn the weight parameters by directly minimizing the reconstruction error, without having to deal with Monte Carlo fluctuations. As to the bias parameters, which threshold the likelihood ratio tests for pattern detection, we can simply set their values to constrain the sparsity [OF97] of activations. Such an unsupervised learning method is as fast as training a discriminative ConvNet in supervised learning.

In general, it is possible for multi-step Langevin dynamics to move out of the Gaussian piece of $Y^{\text{obs}}$ and into another piece of a different activation pattern. But it is unlikely for the activation patterns of $Y^{\text{obs}}$ and $Y^{\text{syn}}$ to be very different. It may be particularly difficult to flip the activation variables at higher layers. In this case, the contrastive divergence may be maximizing the pseudo-likelihood [Bes74] conditioning on the higher layer binary variables. In any case, equation (2.27) makes it clear that we do not have much information to update the bias terms associated with these variables, but we can set their values by sparsity constraints as mentioned above.
2.7 Multi-grid Learning

While the algorithm D learns a descriptor and generates very vivid images, it is still difficult to sample high-dimensional images from the learned descriptor. Image pixels are highly correlated, which prevents the Langevin dynamics jumps out of the local modes. The sampling problem is also a hurdle to learn accurate parameters of a descriptor. To address the sampling problem, we propose a multi-grid learning algorithm as illustrated in Figure 2.3.

Formally, let \( Y = (Y_0, Y_1, \cdots, Y_k) \) denote same images with different of resolutions. Without loss of generality, suppose \( k = 3 \), \( Y_3, Y_2, Y_1 \) and \( Y_0 \) represent \( 64 \times 64 \), \( 16 \times 16 \), \( 4 \times 4 \) and \( 1 \times 1 \) images respectively. We can learn multiple distribution \( p(Y_k) \), which is modeled by a descriptor.

\[
p(Y_k) = \frac{1}{Z(W_D)} \exp \left[ f_k(Y_k; W_D) \right] q(Y_k) \tag{2.33}
\]

The learning algorithm is similar to algorithm D, which updates parameters by equation 2.27 except that the Langevin sampling starts from the up-sampling version of \( Y_{k-1} \).

Compared with the descriptor, the distribution of multi-grid learning \( p(Y_k) \) depends on the down-sampling version of images, which gives partial information as hint for images sampling. The multi-grid learning is connected recovery likelihood, the down-sampling version of
of images can be viewed as a special pattern of occlusion. For instance, from $Y_2$ to $Y_3$, in each $2 \times 2$ cell of image pixels, only one pixel is observable and the others are three copies.

In a more general perspective, the multi-grid learning can be viewed as orthogonal projection, $X = Q^T Y$. $X = (X_0, X_1)$, where $X_0$ is the down-sampling version of $Y$, and $X_1$ is its remainder space. In the special case above, $Q = (q_1, \ldots, q_n)$ forms an orthogonal basis, since the down-sampling is the average of four pixel in each $2 \times 2$ cell, and such average computation has no overlap. Similarly, the up-sampling is the copy of low resolution image to high resolution on each $2 \times 2$ cell without overlap. We can further divide the projection matrix into $Q = (Q_0, Q_1)$, where $Q_0 = (q_1, \ldots, q_d)$ represents the down-sampling operation and $Q_1 = (q_{d+1}, \ldots, q_D)$ maps $Y$ to its remainder space. Accordingly, $X = (X_0 = Q_0^T Y, X_1 = Q_1^T Y)$. The new distribution of $p(X)$ can be obtained by linear transformation,

$$p(X) = \frac{1}{|\text{det}(Q)|}p(QX) = p(Q_0 X_0 + Q_1 X_1) \quad (2.34)$$

The conditional distribution $p(X_1|X_0) = p(X_0, X_1)p(X_0) \propto p(X_0, X_1)$. Then we can sample $p(X_1|X_0)$ by Langevin,

$$X_{1}^{(t+1)} = X_{1}^{(t)} + \frac{\delta^2}{2} \frac{\partial}{\partial X_1} \log p(X_0, X_1^{(t)}) + \delta U^{(t)} \quad (2.35)$$

The above Langevin can be obtained from the Langevin for sampling $p(Y) = p(X_0, X_1)$,

$$X^{(t+1)} = X^{(t)} + \frac{\delta^2}{2} \frac{\partial}{\partial X} \log p(x) + \delta V^{(t)} \quad (2.36)$$

Equation 2.35 can be obtained from equation 2.36 in each iteration, we reset $X_0^{(t)} = X_0$

The Langevin 2.36 can be obtained from the Langevin that sample from $p(Y)$ by mapping $Y^{(t)}$ to $X^{(t)} = Q^T Y^{(t)}$, and $X^{(t+1)} = Q^T Y^{(t+1)}$.

$$Y^{(t+1)} = Y^{(t)} + \frac{\delta^2}{2} \frac{\partial}{\partial Y} \log p(Y^{(t)}) + \delta S^{(t)} \quad (2.37)$$
In order to get equation 2.35, we want to fix $X^{(t)}_0 = Q^T_0 Y^{(t)} = X_0$ for all the iterations $t$. Equivalently, we can fix $Q_0 X^{(t)}_0 = Q_0 Q^T_0 Y^{(t)} = Q_0 X_0$. $Y_0 = Q_0 Q^T_0 Y$ is the projection of $Y$ into the subspace spanned by the basis vectors in $Q_0$. $Y_1 = Y - Y_0 = Q_1 Q^T_1 Y$ is the projection of $Y$ into the subspace spanned by the basis vectors in $Q_1$. So we need to enforce $Y^{(t)}_0 = Y_0 = Q_0 X_0$ for all the iterations $t$. That is, after we get $Y^{(t+1)}$ from equation 2.37, we get $Y^{(t+1)} = Y^{(t+1)}_0 + Y^{(t+1)}_1$, and then we reset $Y^{(t+1)}_0 = Y_0$, and change $Y^{(t+1)} = Y_0 + Y^{(t+1)}_1$.

Figure 2.4 displays the generated images.
CHAPTER 3

Generator

Still let $Y$ be a $D$-dimensional observed data vector, such as an image. Let $X$ be the $d$-dimensional vector of continuous latent factors, $X = (x_k, k = 1, ..., d)$. The traditional factor analysis model is $Y = WX + \epsilon$, where $W$ is $D \times d$ matrix, and $\epsilon$ is a $D$-dimensional error vector or the observational noise. We assume that $X \sim N(0, I_d)$, where $I_d$ stands for the $d$-dimensional identity matrix. We also assume that $\epsilon \sim N(0, \sigma^2 I_D)$, i.e., the observational errors are Gaussian white noises. There are three perspectives to view $W$. (1) Basis vectors. Write $W = (W_1, ..., W_d)$, where each $W_k$ is a $D$-dimensional column vector. Then $Y = \sum_{k=1}^d x_k W_k + \epsilon$, i.e., $W_k$ are the basis vectors and $x_k$ are the coefficients. (2) Loading matrix. Write $W = (w_1, ..., w_D)^\top$, where $w_j^\top$ is the $j$-th row of $W$. Then $y_j = \langle w_j, X \rangle + \epsilon_j$, where $y_j$ and $\epsilon_j$ are the $j$-th components of $Y$ and $\epsilon$ respectively. Each $y_j$ is a loading of the $d$ factors where $w_j$ is a vector of loading weights, indicating which factors are important for determining $y_j$. $W$ is called the loading matrix. (3) Matrix factorization. Suppose we observe $Y = (Y_1, ..., Y_n)$, whose factors are $X = (X_1, ..., X_n)$, then $Y \approx WX$.

The factor analysis model can be learned by the Rubin-Thayer EM algorithm, which involves alternating regressions of $X$ on $Y$ in the E-step and of $Y$ on $X$ in the M-step, with both steps powered by the sweep operator [RT82, LRW98].

3.1 Generator as generalized factor analysis model

In addition to generalizing the prior model of the latent factors $X$ as mentioned in the literature, we can also generalize the mapping from $X$ to $Y$. We consider the generator network model [GPM14] that retains the assumptions that $d < D$, $X \sim N(0, I_d)$, and
\[ \epsilon \sim N(0, \sigma^2 I_D) \] as in traditional factor analysis, but generalizes the linear mapping \( WX \) to a non-linear mapping \( g(X; W_G) \), where \( f \) is a ConvNet, and \( W_G \) collects all the connection weights and bias terms of the ConvNet. Then the model becomes

\[
Y = g(X; W_G) + \epsilon,
\]

\[
X \sim N(0, I_d), \quad \epsilon \sim N(0, \sigma^2 I_D), \quad d < D.
\] (3.1)

The reconstruction error is \( ||Y - g(X; W_G)||^2 \). We may assume more sophisticated models for \( \epsilon \), such as colored noise or non-Gaussian texture. If \( Y \) is binary, we can emit \( Y \) by a probability map \( P = 1/[1 + \exp(-g(X; W_G))] \), where the sigmoid transformation and Bernoulli sampling are carried out pixel-wise. If \( Y \) is multi-level, we may assume multinomial logistic emission model or some ordinal emission model.

Although \( g(X; W_G) \) can be any non-linear mapping, the ConvNet parameterization of \( g(X; W_G) \) makes it particularly close to the original factor analysis. Specifically, we can write the top-down ConvNet as follows:

\[
X^{(l-1)} = g_l(W_l X^{(l)} + b_l),
\] (3.2)

where \( g_l \) is element-wise non-linearity at layer \( l \), \( W_l \) is the matrix of connection weights, \( b_l \) is the vector of bias terms at layer \( l \), and \( W_G = (W_l, b_l, l = 1, ..., L) \). \( X^{(0)} = g(X; W_G) \), and \( X^{(L)} = X \). The top-down ConvNet (3.2) can be considered a recursion of the original factor analysis model, where the factors at the layer \( l - 1 \) are obtained by the linear superposition of the basis vectors or basis functions that are column vectors of \( W_l \), with the factors at the layer \( l \) serving as the coefficients of the linear superposition. In the case of ConvNet, the basis functions are shift-invariant versions of one another, like wavelets.

### 3.2 ReLU and piecewise factor analysis

The generator network is \( Y = g(X; W_G) + \epsilon \), \( X^{(l-1)} = g_l(W_l X^{(l)} + b_l), \) \( l = 1, ..., L \), with \( X^{(0)} = g(X; W_G) \), and \( X^{(L)} = X \). The element-wise non-linearity \( g_l \) in modern ConvNet
is usually the two-piece linearity, such as rectified linear unit (ReLU) \cite{KSH12} or the leaky ReLU \cite{MHN13,XWC15}. Each ReLU unit corresponds to a binary switch. For the case of non-leaky ReLU, following the analysis of \cite{PMB13}, we can write $X^{(l-1)} = \delta_l(W_lX^{(l)} + b_l)$, where $\delta_l = \text{diag}(1(W_lX^{(l)} + b_l > 0))$ is a diagonal matrix, $1()$ is an element-wise indicator function. For the case of leaky ReLU, the 0 values on the diagonal are replaced by a leaking factor (e.g., .2).

$\delta = (\delta_l, l = 1, ..., L)$ forms a classification of $X$ according to the network $W_G$. Specifically, the factor space of $X$ is divided into a large number of pieces by the hyperplanes $W_lX^{(l)} + b_l = 0$, and each piece is indexed by an instantiation of $\delta$. We can write $\delta = \delta(X; W_G)$ to make explicit its dependence on $X$ and $W_G$. On the piece indexed by $\delta$, $g(X; W_G) = W_\delta X + b_\delta$.

Assuming $b_l = 0, \forall l$, for simplicity, we have $W_\delta = \delta_1W_1...\delta_LW_L$. Thus each piece defined by $\delta = \delta(X; W_G)$ corresponds to a linear factor analysis $Y = W_\delta X + \epsilon$, whose basis $W_\delta$ is a multiplicative recomposition of the basis functions at multiple layers ($W_l, l = 1, ..., L$), and the recomposition is controlled by the binary switches at multiple layers $\delta = (\delta_l, l = 1, ..., L)$.

Hence the top-down ConvNet amounts to a reconfigurable basis $W_\delta$ for representing $Y$, and the model is a piecewise linear factor analysis. If we retain the bias term, we will have $Y = W_\delta X + b_\delta + \epsilon$, for an overall bias term that depends on $\delta$. So the distribution of $Y$ is essentially piecewise Gaussian.

The generator model can be considered an explicit implementation of the local linear embedding \cite{RS00}, where $X$ is the embedding of $Y$. In local linear embedding, the mapping between $X$ and $Y$ is implicit. In the generator model, the mapping from $X$ to $Y$ is explicit. With ReLU ConvNet, the mapping is piecewise linear, which is consistent with local linear embedding, except that the partition of the linear pieces by $\delta(X; W_G)$ in the generator model is learned automatically.

The inferential back-propagation is a Langevin dynamics on the energy function $\|Y - g(X; W_G)\|^2/(2\sigma^2) + \|X\|^2/2$. With $g(X; W_G) = W_\delta X$, $\partial g(X; W_G)/\partial X = W_\delta$. If $X$ belongs to the piece defined by $\delta$, then the inferential back-propagation seeks to approximate $Y$ by the basis $W_\delta$ via a ridge regression. Because $X$ keeps changing during the Langevin dynamics, $\delta(X; W_G)$ may also be changing, and the algorithm searches for the optimal reconfigurable
basis $W_δ$ to approximate $Y$. We may solve $X$ by second-order methods such as iterated ridge regression, which can be computationally more expensive than the simple gradient descent.

### 3.3 Alternating back-propagation

Inspired by Rubin-Thayer EM algorithm [RT82, DLR77], we propose an alternating back-propagation algorithm for learning the generator network that iterates the following two-steps:

1. **Inferential back-propagation**: For each training example, infer the continuous latent factors by Langevin dynamics or gradient descent.

2. **Learning back-propagation**: Update the parameters given the inferred latent factors by gradient descent.

The Langevin dynamics [Nea11] is a stochastic sampling counterpart of gradient descent. The gradient computations in both steps are powered by back-propagation. Because of the ConvNet structure, the gradient computation in step (1) is actually a by-product of the gradient computation in step (2) in terms of coding.

Given the factors, the learning of the ConvNet is a supervised learning problem [DSB15] that can be accomplished by the learning back-propagation. With factors unknown, the learning becomes an unsupervised problem, which can be solved by adding the inferential back-propagation as an inner loop of the learning process. We shall show that the alternating back-propagation algorithm can learn realistic generator models of natural images, video sequences, and sounds.

The alternating back-propagation algorithm follows the tradition of alternating operations in unsupervised learning, such as alternating linear regression in the EM algorithm for factor analysis, alternating least squares algorithm for matrix factorization [KBV09, KP08], and alternating gradient descent algorithm for sparse coding [OF97]. All these unsupervised learning algorithms alternate an inference step and a learning step, as is the case with alternating back-propagation.
The inferential back-propagation solves an inverse problem by an explaining-away process, where the latent factors compete with each other to explain each training example. The following are the advantages of the explaining-away inference of the latent factors:

(1) The latent factors may follow sophisticated prior models. For instance, in textured motions \[WZ03\] or dynamic textures \[DCW03a\], the latent factors may follow a dynamic model such as vector auto-regression. By inferring the latent factors that explain the observed examples, we can learn the prior model.

(2) The observed data may be incomplete or indirect. For instance, the training images may contain occluded objects. In this case, the latent factors can still be obtained by explaining the incomplete or indirect observations, and the model can still be learned as before.

Formally, if we observe a training set of data vectors \( \{Y_i, i = 1, \ldots, n\} \), then each \( Y_i \) has a corresponding \( X_i \), but all the \( X_i \) share the same ConvNet \( W_G \). Intuitively, we should infer \( \{X_i\} \) and learn \( W_G \) to minimize the reconstruction error \( \sum_{i=1}^{M} ||Y_i - g(X_i; W_G)||^2 \) plus a regularization term that corresponds to the prior on \( X \).

More formally, the model can be written as \( X \sim p(X) \) and \( [Y|X, W_G] \sim p(Y|X, W_G) \). Adopting the language of the EM algorithm \[DLR77\], the complete-data model is given by

\[
\log p(Y, X; W_G) = \log [p(X)p(Y|X, W_G)] \\
= -\frac{1}{2\sigma^2}||Y - g(X; W_G)||^2 - \frac{1}{2}||X||^2 + \text{const.} \quad (3.3)
\]

The observed-data model is obtained by integrating out the latent factors \( X \): \( p(Y; W_G) = \int p(X)p(Y|X, W_G)dX \). The posterior distribution of \( X \) is given by \( p(X|Y, W_G) = p(Y, X; W_G)/p(Y; W_G) \propto p(X)p(Y|X, W_G) \) as a function of \( X \).

For the training data \( \{Y_i\} \), the complete-data log-likelihood is \( L(W_G, \{X_i\}) = \sum_{i=1}^{n} \log p(Y_i, X_i; W_G) \), where we assume \( \sigma^2 \) is given. Learning and inference can be accomplished by maximizing the complete-data log-likelihood, which can be obtained by the alternating gradient descent algorithm that iterates the following two steps: (1) Inference step: update \( X_i \) by running l
steps of gradient descent. (2) Learning step: update $W_G$ by one step of gradient descent.

A more rigorous method is to maximize the observed-data log-likelihood, which is $L(W_G) = \sum_{i=1}^n \log p(Y_i; W_G) = \sum_{i=1}^M \log \int p(Y_i, X_i; W_G) dX_i$. The observed-data log-likelihood takes into account the uncertainties in inferring $X_i$.

The gradient of $L(W_G)$ can be calculated according to the following well-known fact that underlies the EM algorithm:

$$
\frac{\partial}{\partial W_G} \log p(Y; W_G) = \frac{1}{P(Y; W_G)} \frac{\partial}{\partial W_G} \int p(Y, X; W_G) dX
= E_{p(X|Y, W)} \left[ \frac{\partial}{\partial W_G} \log p(Y, X; W_G) \right].
$$

The expectation with respect to $p(X|Y, W_G)$ can be approximated by drawing samples from $p(X|Y, W_G)$ and then computing the Monte Carlo average.

The Langevin dynamics for sampling $X \sim p(X|Y, W_G)$ iterates

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

where $\tau$ denotes the time step for the Langevin sampling, $s$ is the step size, and $U_{\tau}$ denotes a random vector that follows $N(0, I_d)$. The Langevin dynamics (3.5) is an explain-away process, where the latent factors in $X$ compete to explain away the current residual $Y - g(X_{\tau}; W_G)$.

To explain Langevin dynamics, its continuous time version for sampling $\pi(x) \propto \exp[-E(x)]$ is $x_{t+\Delta t} = x_t - \Delta t E'(x_t)/2 + \sqrt{\Delta t} U_t$. The dynamics has $\pi$ as its stationary distribution, because it can be shown that for any well-behaved testing function $h$, if $x_t \sim \pi$, then $E[h(x_{t+\Delta t})] - E[h(x_t)] \to 0$, as $\Delta t \to 0$, so that $x_{t+\Delta t} \sim \pi$. Alternatively, given $x_t = x$, suppose $x_{t+\Delta t} \sim K(x, y)$, then $[\pi(y)K(y, x)]/[\pi(x)K(x, y)] \to 1$ as $\Delta t \to 0$.

The stochastic gradient algorithm of [You99] can be used for learning, where in each iteration, for each $X_i$, only a single copy of $X_i$ is sampled from $p(X_i|Y_i, W_G)$ by running a finite number of steps of Langevin dynamics starting from the current value of $X_i$, i.e., the warm start. With $\{X_i\}$ sampled in this manner, we can update the parameter $W_G$ based on the gradient $L'(W_G)$, whose Monte Carlo approximation is:
\[ L'(W_G) \approx \sum_{i=1}^{M} \frac{\partial}{\partial W_G} \log p(Y_i, X_i; W_G) \]
\[ = -\sum_{i=1}^{M} \frac{\partial}{\partial W_G} \frac{1}{2\sigma^2} \| Y_i - g(X_i; W_G) \|^2 \]
\[ = \sum_{i=1}^{M} \frac{1}{\sigma^2} (Y_i - g(X_i; W_G)) \frac{\partial}{\partial W_G} g(X_i; W_G). \quad (3.6) \]

Algorithm 2 describes the details of the learning and sampling algorithm.

**Algorithm 2 Algorithm G: Alternating back-propagation**

**Require:**

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

**Ensure:**

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

1: Let \( t \leftarrow 0 \), initialize \( W_G \).
2: Initialize \( X_i, i = 1, ..., n \).
3: repeat
4: \textbf{Step G1 Langevin inference}: For each \( i \), run \( l_G \) steps of Langevin dynamics to update \( X_i \), i.e., starting from the current \( X_i \), each step follows equation (3.5).
5: \textbf{Step G2 reconstruction}: Update \( W_G^{(t+1)} = W_G^{(t)} + \gamma_t L_G'(W_G^{(t)}) \), with learning rate \( \gamma_t \), where \( L_G'(W_G^{(t)}) \) is computed according to equation (3.6).
6: Let \( t \leftarrow t + 1 \)
7: until \( t = T \)

If the Gaussian noise \( U_\tau \) in the Langevin dynamics (3.5) is removed, then the above algorithm becomes the alternating gradient descent algorithm. It is possible to update both \( W_G \) and \( \{X_i\} \) simultaneously by joint gradient descent.
Both the inferential back-propagation and the learning back-propagation are guided by the residual $Y_i - g(X_i; W_g)$. The inferential back-propagation is based on $\partial g(X; W_g)/\partial X$, whereas the learning back-propagation is based on $\partial g(X; W_g)/\partial W_g$. Both gradients can be efficiently computed by back-propagation. The computations of the two gradients share most of their steps. Specifically, for the top-down ConvNet defined by $\partial g(X; W_g)/\partial X$ and $\partial g(X; W_g)/\partial W_g$ share the same code for the chain rule computation of $\partial X^{(l-1)}/\partial X^{(l)}$ for $l = 1, ..., L$. Thus, the code for $\partial g(X; W_g)/\partial X$ is part of the code for $\partial g(X; W_g)/\partial W_g$.

In Algorithm 2, the Langevin dynamics samples from a gradually changing posterior distribution $p(X_i|Y_i, W_g)$ because $W_g$ keeps changing. The updating of both $X_i$ and $W_g$ collaborate to reduce the reconstruction error $||Y_i - g(X_i; W_g)||^2$. The parameter $\sigma^2$ plays the role of annealing or tempering in Langevin sampling. If $\sigma^2$ is very large, then the posterior is close to the prior $N(0, I_d)$. If $\sigma^2$ is very small, then the posterior may be multi-modal, but the evolving energy landscape of $p(X_i|Y_i, W_g)$ may help alleviate the trapping of the local modes. In practice, we tune the value of $\sigma^2$ instead of estimating it. The Langevin dynamics can be extended to Hamiltonian Monte Carlo [Nea11] or more sophisticated versions [GCT11].

### 3.4 EM, density mapping, and density shifting

Suppose the training data $\{Y_i, i = 1, ..., n\}$ come from a data distribution $P_{\text{data}}(Y)$. To understand how the alternating back-propagation algorithm or its EM idealization maps the prior distribution of the latent factors $p(X)$ to the data distribution $P_{\text{data}}(Y)$ by the learned $g(X; W_g)$, we define

$$P_{\text{data}}(X, Y; W_g) = P_{\text{data}}(Y)p(X|Y, W_g)$$
$$= P_{\text{data}}(X; W_g)P_{\text{data}}(Y|X, W_g), \quad (3.7)$$

where $P_{\text{data}}(X; W_g) = \int p(X|Y, W_g)P_{\text{data}}(Y)dy$ is obtained by averaging the posteriors $p(X|Y; W_g)$ over the observed data $Y \sim P_{\text{data}}$. That is, $P_{\text{data}}(X; W_g)$ can be considered the data prior. The data prior $P_{\text{data}}(X; W_g)$ is close to the true prior $p(X)$ in the sense that...
The right hand side of (3.8) is minimized at the maximum likelihood estimate $\hat{W}_G$, hence the data prior $P_{\text{data}}(X; \hat{W}_G)$ at $\hat{W}_G$ should be especially close to the true prior $p(X)$. In other words, at $\hat{W}_G$, the posteriors $p(X|Y, \hat{W}_G)$ of all the data points $Y \sim P_{\text{data}}$ tend to pave the true prior $p(X)$.

From Rubin’s multiple imputation point of view [Rub04] of the EM algorithm, the E-step of EM infers $X_i^{(n)} \sim p(X_i|Y_i, W_t)$ for $m = 1, ..., M$, where $M$ is the number of multiple imputations or multiple guesses of $X_i$. The multiple guesses account for the uncertainty in inferring $X_i$ from $Y_i$. The M-step of EM maximizes $Q(W_t) = \sum_{i=1}^{n} \sum_{m=1}^{M} \log p(Y_i, X_i^{(m)}; W_t)$ to obtain $W_{t+1}$. For each data point $Y_i$, $W_{t+1}$ seeks to reconstruct $Y_i$ by $g(X; W_t)$ from the inferred latent factors $\{X_i^{(m)}, m = 1, ..., M\}$. In other words, the M-step seeks to map $\{X_i^{(m)}\}$ to $Y_i$. Pooling over all $i = 1, ..., n$, $\{X_i^{(m)}, \forall i, m\} \sim P_{\text{data}}(X; W_t)$, hence the M-step seeks to map $P_{\text{data}}(X; W_t)$ to the data distribution $P_{\text{data}}(Y)$. Of course the mapping from $\{X_i^{(m)}\}$ to $Y_i$ cannot be exact. In fact, $g(X; W_t)$ maps $\{X_i^{(m)}\}$ to a $d$-dimensional patch around the $D$-dimensional $Y_i$. The local patches for all $\{Y_i, \forall i\}$ patch up the $d$-dimensional manifold form by the $D$-dimensional observed examples and their interpolations. The EM algorithm is a process of density shifting, so that $P_{\text{data}}(X; W_t)$ shifts towards $p(X)$, thus $g(X; W_t)$ maps $p(X)$ to $P_{\text{data}}(Y)$. 

49
CHAPTER 4

Cooperative learning of two models

4.1 CoopNets algorithm

In Algorithm D and Algorithm G, both steps D1 and G1 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, we can initialize the synthesized examples by generating examples from the generator net, which does not require MCMC, because the generator net is a directed graphical model. More specifically, we first generate \( \hat{X}_i \sim N(0, I_d) \), and then generate \( \hat{Y}_i = g(\hat{X}_i; W_G) + \epsilon_i \), for \( i = 1, ..., \tilde{M} \). If the current generator \( P_G \) is close to the current descriptor \( P_D \), then the generated \( \{ \hat{Y}_i \} \) should be a good initialization for sampling from the descriptor net, i.e., starting from the \( \{ \hat{Y}_i, i = 1, ..., \tilde{M} \} \) supplied by the generator net, we run Langevin dynamics in Step D1 for \( l_D \) steps to get \( \{ \hat{Y}_i, i = 1, ..., \tilde{M} \} \), which are revised versions of \( \{ \hat{Y}_i \} \). These \( \{ \hat{Y}_i \} \) can be used as the synthesized examples from the descriptor net. We can then update \( W_D \) according to Step D2 of Algorithm D.

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

The left diagram in (4.1) illustrates the basic idea.

In the two diagrams in (4.1), the double line arrows indicate generation and reconstruction in the generator net, while the dashed line arrows indicate Langevin dynamics for revision and inference in the two nets. The diagram on the right in (4.1) illustrates a more rigorous method, where we initialize the Langevin inference of $\{X_i, i = 1, \ldots, \tilde{M}\}$ in Step G1 from $\{\hat{X}_i\}$, and then update $W_g$ in Step G2 based on $\{(\tilde{Y}_i, X_i), i = 1, \ldots, \tilde{M}\}$.

Algorithm 3 describes the cooperative training that interweaves Algorithm D and Algorithm G. See Fig. 1.5 for the flow chart of the CoopNets algorithm. In our experiments, we set $l_G = 0$ and infer $X_i = \hat{X}_i$.

The learning of both the descriptor and the generator follows the “analysis by synthesis” principle [GM07]. There are three sets of synthesized examples (S stands for synthesis). (S1) Initial synthesized examples $\{\hat{Y}_i\}$ generated by Step G0. (S2) Revised synthesized examples $\{\tilde{Y}_i\}$ produced by Step D1. (S3) Reconstructed synthesized examples $\{g(X_i; W_g^{(t+1)})\}$ produced by Step G2. The descriptor shifts its density from (S2) towards the observed data, while the generator shifts its density from (S1) towards (S2) (by arriving at (S3)).

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
Algorithm 3 CoopNets Algorithm

Require:

(1) training examples \( \{Y_i, i = 1, \ldots, n\} \)
(2) numbers of Langevin steps \( l_D \) and \( l_G \)
(3) number of learning iterations \( T \)

Ensure:

(1) estimated parameters \( W_D \) and \( W_G \)
(2) synthesized examples \( \{\hat{Y}_i, \tilde{Y}_i, i = 1, \ldots, \tilde{M}\} \)

1: Let \( t \leftarrow 0 \), initialize \( W_D \) and \( W_G \).
2: repeat
3: \hspace{1em} \textbf{Step G0}: For \( i = 1, \ldots, \tilde{M} \), generate \( \hat{X}_i \sim N(0, I_d) \), and generate \( \hat{Y}_i = g(\hat{X}_i; W_G^{(t)}) + \epsilon_i \).
4: \hspace{1em} \textbf{Step D1}: For \( i = 1, \ldots, \tilde{M} \), starting from \( \tilde{Y}_i \), Run \( l_D \) steps of Langevin revision dynamics to obtain \( \tilde{Y}_i \), each step following equation (2.23).
5: \hspace{1em} \textbf{Step G1}: Treat the current \( \{\tilde{Y}_i, i = 1, \ldots, \tilde{M}\} \) as the training data, for each \( i \), infer \( X_i = \hat{X}_i \). Or more rigorously, starting from \( X_i = \hat{X}_i \), run \( l_G \) steps of Langevin inference dynamics to update \( X_i \), each step following equation (3.5).
6: \hspace{1em} \textbf{Step D2}: Update \( W_D^{(t+1)} = W_D^{(t)} + \gamma_t L_D'(W_D^{(t)}) \), where \( L_D'(W_D^{(t)}) \) is computed according to (2.27).
7: \hspace{1em} \textbf{Step G2}: Update \( W_G^{(t+1)} = W_G^{(t)} + \gamma_t L_G'(W_G^{(t)}) \), where \( L_G'(W_G) \) is computed according to equation (3.6), except that \( Y_i \) is replaced by \( \tilde{Y}_i \), and \( n \) by \( \tilde{M} \). We can run multiple iterations of Step G2 to learn from and reconstruct \( \{\hat{Y}_i\} \), and to allow the generator to catch up with the descriptor.
8: Let \( t \leftarrow t + 1 \)
9: until \( t = T \)
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.

In the zero temperature limit, the Langevin revision dynamics becomes gradient descent on the energy function. The CoopNets algorithm approximately solves the following minimax problem:

$$\min_{W_G} \max_{W_D} \left( E_{W_G} [\mathcal{E}(Y; W_D)] - E_{P_{\text{data}}} [\mathcal{E}(Y; W_D)] \right).$$

That is, we change $W_D$ to shift the low energy regions or local energy minima from the synthesized data to the observed data, and we change $W_G$ to map the latent factors to low energy regions or local modes. The temperature should be lowered gradually so that the local energy minima have the correct probabilities.

### 4.2 Generator of infinite capacity

In the CoopNets algorithm, the descriptor learns from the observed examples, while the generator learns from the descriptor through the synthesized examples. Therefore, the descriptor is the driving force in terms of learning, although the generator is the driving force in terms of synthesis. In order to understand the convergence of learning, we can start from Algorithm D for learning the descriptor.

Algorithm D is a stochastic approximation algorithm [RM51], except that the samples are generated by finite step MCMC transitions. According to [You99], Algorithm D converges to the maximum likelihood estimate under suitable regularity conditions on the mixing of the transition kernel of the MCMC and the schedule of the learning rate $\gamma_t$, even if the number of Langevin steps $l_D$ is finite or small (e.g., $l_D = 1$), and even if the number of parallel chains $\tilde{M}$ is finite or small (e.g., $\tilde{M} = 1$). The reason is that the random fluctuations caused by the finite number of chains, $\tilde{M}$, and the limited mixing caused by the finite steps of MCMC, $l_D$, are mitigated if the learning rate $\gamma_t$ is sufficiently small. At learning iteration $t$, let $W_D^{(t)}$ be the estimated parameter of the descriptor. Let $P_{D}^{(t+1)}$ be the marginal distribution of $\{\tilde{Y}_i\}$,
even though \( P_D^{(t+1)} \neq P_D(Y;W_D^{(t)}) \) because \( l_D \) is finite \( (P_D^{(t+1)} = P_D(Y;W_D^{(t)}) \) if \( l_D \to \infty \)), we still have \( W_D^{(t)} \to \hat{W}_D \) in probability according to [You99], where \( \hat{W}_D \) is the maximum likelihood estimate of \( W_D \).

The efficiency of Algorithm D increases if the number of parallel chains \( \tilde{M} \) is large because it leads to more accurate estimation of the expectation in the gradient \( L_D'(W_D) \) of equation (2.27), so that we can afford to use larger learning rate \( \gamma_t \) for faster convergence.

Now let us come back to the CoopNets algorithm. In order to understand how the descriptor net helps the training of the generator net, let us consider the idealized scenario where the number of parallel chains \( \tilde{M} \to \infty \), and the generator has infinite capacity, and in each iteration it estimates \( W_G \) by maximum likelihood using the synthesized data from \( P_D^{(t+1)} \). In this idealized scenario, the learned generator \( P_G(Y;W_G^{(t+1)}) \) will reproduce \( P_D^{(t+1)} \) by minimizing \( \text{KL}(P_D^{(t+1)}(Y)|P_G(Y;W_G)) \), with \( P_D^{(t+1)} \) serving as its data distribution. Then eventually the learned generator \( P_G(Y,\hat{W}_G) \) will reproduce \( P_D(Y;\hat{W}_D) \). Thus the cooperative training helps the learning of the generator. Note that the learned generator \( P_G(Y,\hat{W}_G) \) will not reproduce the distribution of the observed data \( P_{\text{data}} \), unless the descriptor is of infinite capacity too.

Conversely, the generator net also helps the learning of the descriptor net in the CoopNets algorithm. In Algorithm D, it is impractical to make the number of parallel chains \( \tilde{M} \) too large. On the other hand, it would be difficult for a small number of chains \( \{\tilde{Y}_i, i = 1, ..., \tilde{M}\} \) to explore the state space. In the CoopNets algorithm, because \( P_G(Y;W_G^{(t)}) \) reproduces \( P_D^{(t)} \), we can generate a completely new batch of independent samples \( \{\hat{Y}_i\} \) from \( P_G(Y;W_G^{(t)}) \), and revise \( \{\hat{Y}_i\} \) to \( \{\tilde{Y}_i\} \) by Langevin dynamics, instead of running Langevin dynamics from the same old batch of \( \{\tilde{Y}_i\} \) as in the original Algorithm D. This is like implementing an infinite number of parallel chains, because each iteration evolves a fresh batch of examples, as if each iteration evolves a new set of chains. By updating the generator \( W_G \), it is like we are updating the infinite number of parallel chains, because \( W_G \) memorizes the whole distribution. Even if \( \tilde{M} \) in the CoopNets algorithm is small, e.g., \( \tilde{M} = 1 \), viewed from the perspective of Algorithm D, it is as if \( \tilde{M} \to \infty \). Thus the above idealization \( \tilde{M} \to \infty \) is
4.3 Generator of finite capacity

From an information geometry [AN07] point of view, let $\mathcal{D} = \{P_{\mathcal{D}}(Y; W_{\mathcal{D}}), \forall W_{\mathcal{D}}\}$ be the manifold of the descriptor models, where each distribution $P_{\mathcal{D}}(Y; W_{\mathcal{D}})$ is a point on this manifold. Then the maximum likelihood estimate of $W_{\mathcal{D}}$ is a projection of the data distribution $P_{\text{data}}$ onto the manifold $\mathcal{D}$. Let $\mathcal{G} = \{P_{\mathcal{G}}(Y; W_{\mathcal{G}}), \forall W_{\mathcal{G}}\}$ be the manifold of the generator models, where each distribution $P_{\mathcal{G}}(Y; W_{\mathcal{G}})$ is a point on this manifold. Then the maximum likelihood estimate of $W_{\mathcal{G}}$ is a projection of the data distribution $P_{\text{data}}$ onto the manifold $\mathcal{G}$.

From now on, for notational simplicity and with a slight abuse of notation, we use $W_{\mathcal{D}}$ to denote the descriptor distribution $P_{\mathcal{D}}(Y; W_{\mathcal{D}})$, and use $W_{\mathcal{G}}$ to denote the generator distribution $P_{\mathcal{G}}(Y; W_{\mathcal{G}})$.

We assume both the observed data size $n$ and the synthesized data size $\tilde{M}$ are large enough so that we shall work on distributions or populations instead of finite samples. As explained above, assuming $\tilde{M} \to \infty$ is sound because the generator net can supply unlimited number of examples.

The Langevin revision dynamics runs a Markov chain from $W_{\mathcal{G}}^{(t)}$ towards $W_{\mathcal{D}}^{(t)}$. Let $\mathcal{L}_{W_{\mathcal{D}}}$ be the Markov transition kernel of $l_{\mathcal{D}}$ steps of Langevin revisions towards $W_{\mathcal{D}}$. The distribution of the revised synthesized data is

$$P_{\mathcal{D}}^{(t+1)} = \mathcal{L}_{W_{\mathcal{D}}^{(t)}} \cdot W_{\mathcal{G}}^{(t)},$$

where the notation $\mathcal{L} \cdot P$ denotes the marginal distribution obtained by running the Markov transition $\mathcal{L}$ from $P$. The distribution $P_{\mathcal{D}}^{(t+1)}$ is in the middle between the two nets $W_{\mathcal{G}}^{(t)}$ and $W_{\mathcal{D}}^{(t)}$, and it serves as the data distribution to train the generator, i.e., we project this distribution to the manifold $\mathcal{G} = \{P_{\mathcal{G}}(Y; W_{\mathcal{G}}), \forall W_{\mathcal{G}}\} = \{W_{\mathcal{G}}\}$ (recall we use $W_{\mathcal{G}}$ to denote the distribution $P_{\mathcal{G}}(Y; W_{\mathcal{G}})$) in the information geometry picture, so that

$$W_{\mathcal{G}}^{(t+1)} = \arg \min_{\mathcal{G}} \text{KL}(P_{\mathcal{D}}^{(t+1)}|W_{\mathcal{G}}).$$

(4.4)
The learning process alternates between Markov transition in (4.3) and projection in (4.4), as illustrated by Fig. 4.1. Compared to the three sets of synthesized data mentioned at the end of Section 4.1, \( W_g(t) \) corresponds to (S1), \( P_D^{(t+1)} \) corresponds to (S2), and \( W_g^{(t+1)} \) corresponds to (S3).

Figure 4.1: The learning of the generator alternates between Markov transition and projection. The family of the generator models \( G \) is illustrated by the black curve. Each distribution is illustrated by a point.

In the case of \( l_D \to \infty \),

\[
W_D^{(t)} \to \hat{W}_D = \arg \min_D KL(P_{data}|W_D), \quad (4.5)
\]

\[
W_g^{(t)} \to \hat{W}_g = \arg \min_G KL(\hat{W}_D|W_g). \quad (4.6)
\]

That is, we first project \( P_{data} \) onto \( D \), and from there continue to project onto \( G \). Therefore, \( W_D \) converges to the maximum likelihood estimate with \( P_{data} \) being the data distribution, and \( W_g \) converges to the maximum likelihood estimate with \( \hat{W}_D \) serving as the data distribution.

For finite \( l_D \), the algorithm may converge to the following fixed points. The fixed point for the generator satisfies

\[
\hat{W}_g = \arg \min_G KL(L_{\hat{W}_D} \cdot \hat{W}_g|W_g). \quad (4.7)
\]

The fixed point for the descriptor satisfies

\[
\hat{W}_D = \arg \min_D [KL(P_{data}|W_D)
- KL(L_{\hat{W}_D} \cdot \hat{W}_g|W_D)], \quad (4.8)
\]
which is similar to contrastive divergence \cite{Hin02}, except that \(\hat{W}_G\) takes the place of \(P_{\text{data}}\) in the second Kullback-Leibler divergence. Because \(\hat{W}_G\) is supposed to be close to \(\hat{W}_D\), the second Kullback-Leibler divergence is supposed to be small, hence our algorithm is closer to maximum likelihood learning than contrastive divergence.

\cite{KB16} learned the generator by gradient descent on \(\text{KL}(W_G|W_D^{(t)})\) over \(G\). The objective function is \(\text{KL}(W_G|W_D^{(t)}) = E_{W_G}[\log P_G(Y; W_G)] - E_{W_G}[\log P_D(Y; W_D^{(t)})]\), where the first term is the negative entropy that is intractable, and the second term is the expected energy that is tractable. Our learning method for the generator is consistent with the learning objective \(\text{KL}(W_G|W_D^{(t)})\), because

\[
\text{KL}(P_D^{(t)}|P_D^{(t)}) \leq \text{KL}(W_G^{(t)}|W_D^{(t)}). \tag{4.9}
\]

In fact, \(\text{KL}(P_D^{(t+1)}|W_D^{(t)}) \rightarrow 0\) monotonically as \(l_D \rightarrow \infty\) due to the second law of thermodynamics \cite{CT12}. The reduction of the Kullback-Leibler divergence in (4.9) and the projection in (4.4) in our learning of the generator are consistent with the learning objective of reducing \(\text{KL}(W_G|W_D^{(t)})\) in \cite{KB16}. But the Monte Carlo implementation of \(L\) in our work avoids the need to approximate the intractable entropy term.

### 4.4 Generator and MCMC

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

\[
\begin{align*}
\text{MCMC:} & \quad P^{(t)} & \xrightarrow{\text{Markov transition}} & P^{(t+1)} \\
& \downarrow & & \downarrow \\
\text{Generator:} & \quad W^{(t)} & \xrightarrow{\text{Parameter updating}} & W^{(t+1)}
\end{align*}
\]

(4.10)

where \(P^{(t)}\) is the marginal distribution of a MCMC and \(W^{(t)}\) 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 (we drop the subscripts \(D\) and \(G\) here for simplicity and generality).
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 \( W^{(t)} \) from the existing samples of \( P^{(t)} \), we can replace the existing samples by fresh new samples by sampling from the generator defined by the learned \( W^{(t)} \), which is illustrated by the two-way arrow between \( P^{(t)} \) and \( W^{(t)} \). 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.

If the descriptor is fixed, the generator will converge to the descriptor if the generator has infinite capacity. In the CoopNets algorithm, the descriptor is an evolving one, because we keep updating the parameter \( W_D \) of the descriptor by learning from the observed examples. Therefore the Markov chain in (4.10) is a non-stationary one, because the target distribution of the MCMC, i.e., the current descriptor, keeps evolving, so that the Markov transition kernel also keeps changing. Initially, the distribution of the descriptor is of high temperature, and the noise term dominates the Langevin dynamics. By following the descriptor, the generator will gain high entropy and high variance. As the learning proceeds, the parameters of the descriptor become bigger in magnitude, and the distribution becomes colder and multi-modal, so that the Langevin dynamics will be dominated by the gradient term. At the low temperature, the patterns begin to form. By chasing the descriptor, the generator begins to map the latent factors to the local modes to create the patterns. In the end, it is possible that the generator maps each small neighborhood of the factor space into a local mode of the energy landscape of the descriptor. As observed by [XLZ16], the local modes of the descriptor satisfy an auto-encoder \( Y/s^2 = \partial f(Y; W_D)/\partial Y \). Hence the examples generated by the generator \( Y = g(X; W_G) \) satisfy the auto-encoder defined by the descriptor, so that the stochastic relaxation by the descriptor cannot make much revision of the examples generated by the generator. From an embedding perspective, the generator embeds the high-dimensional local modes of the descriptor into its low-dimensional factor space, thus providing a map of the energy landscape of the descriptor. From the MCMC perspective,
the learning process is like a simulated annealing or tempering process \cite{KGV83, MP92}, and the generator traces out this process in distribution as illustrated by diagram (4.10). As is the case with simulated annealing, it is necessary to begin with a high temperature and high entropy distribution to cast a sufficiently wide generator net, and use a slow tempering schedule, or learning schedule in our case, to capture most of the major modes.

MCMC sampling of an evolving descriptor can be easier than sampling from a fixed descriptor, because the energy landscape of an evolving descriptor keeps changing or even destroying its local modes defined by the auto-encoder, thus releasing the Markov chains from the trapping of the local modes.
CHAPTER 5

Experiment

5.1 Qualitative experiment

We show that the both of two models and CoopNets algorithm are capable of learning and generating realistic natural image patterns. Such an empirical proof of concept validates the generative capacity of the models. We also show that the generator can learn sound and video data. We further show that contrastive divergence learning can indeed reconstruct the observed images, thus empirically validating Proposition 3 of descriptor. To learn a descriptor, training images are scaled by subtracting their mean. To learn a generator and CoopNets, images, sounds and videos are scaled so that the intensities are within the range $[-1, 1]$. Throughout the paper, the generator adopts the structure of the generator network of [RMC15, DSB15], where the top-down network consists of multiple layers of deconvolution by linear superposition, ReLU non-linearity, and up-sampling, with tanh non-linearity at the bottom-layer [RMC15] to make the signals fall within $[-1, 1]$. We also adopt batch normalization [IS15b].

The experiments in this section are qualitative and illustrative in nature. They are not intended for quantitative performance study. The code in the experiments is based on the MatConvNet package of [VL15] and [CKF11].

5.1.1 Modeling texture patterns

Experiment 1. Modeling texture patterns with descriptor. We learn a 3-layer descriptor. The first layer has 100 $15 \times 15$ filters with sub-sampling size of 3. The second layer has 64 $5 \times 5$ filters with sub-sampling size of 1. The third layer has 30 $3 \times 3$ filters with sub-sampling
Figure 5.1: Generating texture patterns with descriptor. For each category, the first image is the training image, and the rest are 2 of the images generated by the descriptor.
Figure 5.2: Modeling texture patterns with generator. For each example, *Left:* the $224 \times 224$ observed image. *Right:* the $448 \times 448$ generated image by the generator.

We learn a descriptor for each category from a single training image. Fig. 5.1 displays the results. For each category, the first image is the training image, and the rest are 2 of the images generated by the learning algorithm. We use $\tilde{M} = 16$ parallel chains for Langevin sampling. The number of Langevin iterations between every two consecutive updates of parameters is $L = 10$. The number of learning iterations is $T = 700$.

**Experiment 2. Modeling texture patterns with generator.** We learn a separate model from each texture image. The images are collected from the Internet, and then resized to $224 \times 224$. The synthesized images are $448 \times 448$. Fig. 5.2 shows four examples.

The factors $X$ at the top layer form a $\sqrt{d} \times \sqrt{d}$ image, with each pixel following $N(0, 1)$ independently. The $\sqrt{d} \times \sqrt{d}$ image $X$ is then transformed to $Y$ by the top-down ConvNet. We use $d = 7^2$ in the learning stage for all the texture experiments. In order to obtain the synthesized image, we randomly sample a $14 \times 14$ $X$ from $N(0, I)$, and then expand the
learned network $W$ to generate the $448 \times 448$ synthesized image $g(X; W_g)$.

The training network is as follows. Starting from $7 \times 7$ image $X$, the network has 5 layers of deconvolution with $5 \times 5$ kernels (i.e., linear superposition of $5 \times 5$ basis functions), with an up-sampling factor of 2 at each layer (i.e., the basis functions are 2 pixels apart). The number of channels in the first layer is 512 (i.e., 512 translation invariant basis functions), and is decreased by a factor 2 at each layer. The Langevin steps $l = 10$ with step size $s = .1$.

5.1.2 Modeling object patterns

**Experiment 1. Modeling object patterns with descriptor.** Experiment 5.1.1 shows clearly that the descriptor can learn from images without alignment. We can also specialize it to learning aligned object patterns by using a single top-layer filter that covers the whole image. It is actually a non-stationary FRAME model of the form (2.9), i.e., a convolutional filter at a fixed position before ReLU non-linearity.

We learn a 4-layer descriptor from images of aligned objects. The first layer has 100 $7 \times 7$ filters with sub-sampling size of 2. The second layer has 64 $5 \times 5$ filters with sub-sampling size of 1. The third layer has 20 $3 \times 3$ filters with sub-sampling size of 1. The fourth layer is a fully connected layer with a single filter that covers the whole image. When growing the layers, we always keep the top-layer single filter, and train it together with the existing layers. We learn a descriptor for each category, where the number of training images for each category is around 10, and they are collected from the Internet. Fig. 5.3 shows the results. For each category, the first row displays 4 of the training images, and the second row shows 4 of the images generated by the learning algorithm.

**Experiment 2. Modeling object patterns with generator.** We model object patterns using the network structure that is essentially the same as the network for the texture model with generator, except that we include a fully connected layer under the latent factors $X$, now a $d$-dimensional vector. The images are $64 \times 64$. We use ReLU with a leaking factor .2 [MHN13, XWC15]. The Langevin steps $l = 30$ with step size $s = .3$.

In the first experiment, we learn a model where $X$ has two components, i.e., $X = (x_1, x_2)$,
Figure 5.3: Generating object patterns with descriptor. For each category, the first row displays 4 of the training images, and the second row displays 4 of the images generated by the descriptor.
Figure 5.4: Modeling object patterns with generator. *Left:* the synthesized images generated by our method. They are generated by \( g(X; W_g) \) with the learned \( W \), where \( X = (x_1, x_2) \in [-2, 2]^2 \), and \( X \) is discretized into \( 9 \times 9 \) values. *Right:* the synthesized images generated using Deep Convolutional Generative Adversarial Net (DCGAN). \( X \) is discretized into \( 9 \times 9 \) values within \([−1, 1]^2\).
Figure 5.5: Modeling object patterns with generator. **Left:** each image generated by our method is obtained by first sampling $X \sim N(0, I_{100})$ and then generating the image by $g(X; W_G)$ with the learned $W$. **Middle:** interpolation. The images at the four corners are reconstructed from the inferred $X$ vectors of four images randomly selected from the training set. Each image in the middle is obtained by first interpolating the $X$ vectors of the four corner images, and then generating the image by $g(X; W_G)$. **Right:** the synthesized images generated by DCGAN, where $X$ is a 100 dimension vector sampled from uniform distribution.
and $d = 2$. The training data are 11 images of 6 tigers and 5 lions. After training the model, we generate images using the learned top-down ConvNet for $(x_1, x_2) \in [-2, 2]^2$, where we discretize both $x_1$ and $x_2$ into 9 equally spaced values. The left panel of Fig. 5.4 displays the synthesized images on the $9 \times 9$ panel.

In the second experiment, we learn a model with $d = 100$ from 1000 face images randomly selected from the CelebA dataset [LLW15]. The left panel of Fig. 5.5 displays the images generated by the learned model. The middle panel displays the interpolation results. The images at the four corners are generated by the $X$ vectors of four images randomly selected from the training set. The images in the middle are obtained by first interpolating the $X$’s of the four corner images using the sphere interpolation [DSB16] and then generating the images by the learned ConvNet.

We also provide qualitative comparison with Deep Convolutional Generative Adversarial Net (DCGAN) [GPM14, RMC15]. The right panel of Fig. 5.4 shows the generated results for the lion-tiger dataset using 2-dimensional $X$. The right panel of Fig. 5.5 displays the generated results trained on 1000 aligned faces from celebA dataset, with $d = 100$. We use the code from https://github.com/carpedm20/DCGAN-tensorflow with the tuning parameters as in [RMC15]. We run $T = 600$ iterations as in our method.

Experiment 3. Modeling unaligned object patterns with CoopNets. We conduct experiments on synthesizing images of categories from Imagenet ILSVRC2012 dataset [DDS09]. We adopt a 4-layer descriptor net. The first layer has 64 $5 \times 5$ filters with sub-sampling of 2, the second layers 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 10 and the step size to 0.002. The learning rate is 0.07. The number of learning iterations is about 1,000. After learning the models, we synthesize images using the learned models. We use the learned generator to generate the initial synthesized examples, which are then revised by the learned descriptor by running 10 to 50 steps of Langevin dynamics.

In the first experiment, we learn from images that are randomly sampled from 10 Im-
Figure 5.6: Images generated by CoopNets learned from 10 Imagenet scene categories. The training set consists of 700 images randomly sampled from each category.
Figure 5.7: Images generated by CoopNets (top) and GAN (bottom). \( M \) is the number of training images randomly sampled from each of the 10 Imagenet scene categories.
Table 5.1: Inception scores of different methods on learning from 10 Imagenet scene categories. $M$ is the number of training images randomly sampled from each category.

<table>
<thead>
<tr>
<th>Method</th>
<th>$M = 50$</th>
<th>$M = 100$</th>
<th>$M = 300$</th>
<th>$M = 500$</th>
<th>$M = 700$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoopNets</td>
<td>2.6578 ± .13</td>
<td>3.0444 ± .13</td>
<td>3.2551 ± .12</td>
<td>3.2751 ± .09</td>
<td>3.4932 ± .10</td>
</tr>
<tr>
<td>GAN</td>
<td>2.2630 ± .16</td>
<td>2.5042 ± .15</td>
<td>3.1607 ± .15</td>
<td>3.0548 ± .12</td>
<td>3.1251 ± .09</td>
</tr>
<tr>
<td>VAE</td>
<td>1.6153 ± .09</td>
<td>1.6324 ± .06</td>
<td>1.6467 ± .05</td>
<td>1.7296 ± .04</td>
<td>1.6715 ± .03</td>
</tr>
<tr>
<td>Algorithm G</td>
<td>1.7231 ± .07</td>
<td>1.9418 ± .09</td>
<td>2.3232 ± .09</td>
<td>2.4019 ± .06</td>
<td>2.4506 ± .05</td>
</tr>
</tbody>
</table>

Table 5.2: Inception scores with different numbers of Langevin steps in learning.

<table>
<thead>
<tr>
<th>$l_D$</th>
<th>$l_D = 10$</th>
<th>$l_D = 5$</th>
<th>$l_D = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.27 ± .085</td>
<td>2.95 ± .057</td>
<td>2.80 ± .11</td>
</tr>
</tbody>
</table>

agenet scene categories. We conduct 5 runs. The numbers of images sampled from each category are 50, 100, 300, 500, and 700 respectively in these 5 runs. Fig. 5.6 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 700. The synthesized examples are randomly sampled from the learned models without cheery picking. Fig. 5.7 displays images generated by CoopNets and GAN for various training sizes.

We evaluate the synthesis quality by the Inception score [SGZ16]. Table 5.1 display the Inception scores of the CoopNets, GAN, VAE, and separate training by Algorithm G. We also tried separate training of the descriptor net by Algorithm D, but could not get meaningful synthesis. This suggests that the cooperation of the two algorithms is indeed beneficial.

As a matter of diagnosis, we try different numbers of Langevin steps $l_D$ in learning the CoopNets, where the number of images from each category is 500. Table 5.2 displays the Inception scores.

In our second experiment, we learn from images randomly sampled from a single Imagenet category. Fig. 5.8 shows synthesized examples from one of the 5 categories.
Figure 5.8: Images generated by CoopNets trained on all the images from a single Imagenet category: school bus.
5.1.3 Synthesis and reconstruction by descriptor

We evaluate the one-step contrastive divergence learning on a small training set of 10 images collected from the Internet. The ConvNet structure is the same as in experiment 1. For computational efficiency, we learn all the layers of filters simultaneously. The number of learning iterations is $T = 1200$. Starting from the observed images, the number of Langevin iterations is $L = 1$. Fig. 5.9 shows the results. The first row displays 4 of the training images, and the second row displays the corresponding auto-encoding reconstructions with the learned parameters.

5.1.4 Modeling sound patterns with 1D generator

A sound signal can be treated as a one-dimensional texture image [MS11]. The sound data are collected from the Internet. Each training signal is a 5 second clip with the sampling rate of 11025 Hertz and is represented as a $1 \times 60000$ vector. We learn a separate model from each sound signal.

The latent factors $X$ form a sequence that follows $N(0, I_d)$, with $d = 6$. The top-down network consists of 4 layers of deconvolution with kernels of size $1 \times 25$, and up-sampling factor of 10. The number of channels in the first layer is 256, and decreases by a factor of 2
at each layer. For synthesis, we start from a longer Gaussian white noise sequence $X$ with $d = 12$ and generate the synthesized sound by expanding the learned network. Fig. 5.10 shows the waveforms of the observed sound signal in the first row and the synthesized sound signal in the second row.

5.1.5 Modeling dynamic patterns with 3D generator

We consider the following three versions of generator model with 3D filters for dynamic patterns $Y(x, y, t)$, where $(x, y) \in S$ indexes the pixel in the spatial domain, and $t \in T$ indexes the frame in the temporal domain:

1. The dynamic pattern in $Y(x, y, t)$ is stationary in both the spatial and temporal domains, such as water waves, rain, snow etc. In that case, we assume $X = X(x, y, t, m)$, where $(x, y) \in S^{(0)}$, $t \in T^{(0)}$, and $m = 1, ..., M$, where $S^{(0)}$ and $T^{(0)}$ are sub-sampled spatial and temporal domains, i.e., the index $k$ in $X = (X_k, k = 1, ..., d)$ takes the form $k = (x, y, t, m)$, so that at the top-layer $X$ consists of $M$ Gaussian white noise image sequences. We assume that the kernel functions, i.e., column vectors in $W_l$ for $l = 1, ..., L$, are all
For each category, the first row displays frames of the training sequence, and the second row displays the frames of the generated sequence. The first category shows the synthesized sequence generated by model-1 with the original length. Category 2 to category 6 display the synthesized sequences generated by model-2 with doubled length. The last category displays the generated sequence by model-3.
convolutional, so that \( g(\mathbf{X}; \mathbf{W}_G) \) is stationary in both spatial and temporal domains after \( L \) layers of deconvolution and up-sampling. The up-sampling expands \((S^{(0)}, T^{(0)})\) at the top layer to \((S, T)\) at the bottom layer.

(2) The dynamic pattern in \( \mathbf{Y}(x, y, t) \) is stationary in the temporal domain, but non-stationary in the spatial domain, such as dynamic textures that exhibit stochastic repetitiveness in the temporal domain. In that case, we assume \( \mathbf{X} = \mathbf{X}(t, m) \), where \( t \in T^{(0)} \), and \( m = 1, ..., M \). This model corresponds to the model in (1) where \( S^{(0)} \) is \( 1 \times 1 \). The kernel functions at the top layer are convolutional in the temporal domain, but are fully connected in the spatial domain, so that \( g(\mathbf{X}; \mathbf{W}_G) \) is stationary in the temporal domain but non-stationary in the spatial domain.

(3) The dynamic pattern in \( \mathbf{Y}(x, y, t) \) is non-stationary in both spatial and temporal domains, such as actions and movements. In that case, we assume \( \mathbf{X} = \mathbf{X}(m) \), \( m = 1, ..., M \). The model corresponds to the model in (1) where \( S^{(0)} \) is a single pixel, and \( T^{(0)} \) is a single frame. The kernel functions at the top layer are fully connected in both the temporal and spatial domains.

Models in (1) and (2) can be learned from a single training sequence. Models in (1) can generate image sequences with both the spatial and temporal ranges expanded. Models in (2) can generate image sequences with expanded temporal range. Models in (3) need to be learned from multiple sequences because of the lack of stochastic repetitiveness in either the spatial or temporal domain.

We train the spatial-temporal generator on a wide range of dynamic patterns randomly selected from the DynTex \([PFH]\) database, action database \([GBS07]\), and from the Internet. Each video sequence from the selected category is partitioned into segments of 32 frames, and such segments are used for training and testing. The pixel values are scaled into the range of \([-1, 1]\). For the generator network, we adopt the architecture that is similar to the generator network of \([VPT16]\) with different top-layer structures for different models. All the deconvolution layers are followed by the batch normalization \([IS15a]\) and the ReLU non-linearity except the last layer where tanh is used. The detailed descriptions of our model
Figure 5.12: The top four rows show the comparison results for flame sequence. The bottom four rows show the results for moving grid sequence. For each category: First row: 8 frames of the training sequence. Second row: 8 frames of the generated sequence using our method. Third row: 8 frames of the generated sequence using 2D generator with an auto-regressive model for latent factors. Fourth row: 8 frames of generated sequence using [DCW03b].
Figure 5.13: First row: bark water training sequence. Second row: frames from the generated sequences using our method. Third row: frames from the generated sequences using 3D GAN [VPT16].

structures will be given later.

We fix the standard deviation of noise vector $\sigma = 1$ for qualitative experiments and $\sigma = 0.5$ for quantitative experiments. We use $l = 20$ steps of Langevin dynamics in each learning iteration, and the Langevin step size is set to 0.1. We use the Adam [KB14] optimizer with a fixed learning rate of 0.0002 and momentum term of 0.5. The algorithm is terminated after 500 iterations.

We perform two sets of qualitative experiments to visualize the learned spatial-temporal generator networks and compare with other popular video models in the literature.

**Experiment 1. Dynamic pattern synthesis.** Most of the dynamic patterns exhibit some regularities in either the spatial or temporal domain. By utilizing various convolutional structures, we can synthesize realistic patterns even with a single training video. Specifically, we experiment with three model structures corresponding to the three versions of the generator models described in section 5.1.5.

1. Model-1 corresponds to the model that is stationary in both the spatial and temporal domains. We learn a 4-layer ConvNet, where there are 256, 128, 64 and 3 filters with the size of $4 \times 4 \times 4$ and stride of 2 in 4 layers respectively. In the last layer, the deconvolution filters are followed by a tanh layer, and each of the other layers of filters are followed by a
batch-normalization layer and a ReLU layer. We use $8 \times 8 \times 4 \times 2$ latent factors to generate dynamic patterns.

(2) Model-2 corresponds to the model that is stationary in the temporal domain, but non-stationary in the spatial domain. We learn a 6-layer ConvNet, where the first layer has 512 filters with the size of $4 \times 4 \times 1$ and stride of 1. The second layer has 64 filters with the size of $7 \times 7 \times 7$ and stride of 1. There are 256, 128, 64 and 3 filters with the size of $4 \times 4 \times 4$ and stride of 2 from layer 3 to layer 6. We use $1 \times 1 \times 8 \times 20$ latent factors to generate the dynamic patterns of the double length.

(3) Model-3 corresponds to the model that is non-stationary in both the spatial and temporal domains. We learn a 6-layer ConvNet, where the first layer has 512 filters with the size of $4 \times 4 \times 1$ and stride of 1. The second layer has 384 filters with the size of $7 \times 7 \times 7$ and stride of 1. There are 256, 128, 64 and 3 filters with the size of $4 \times 4 \times 4$ and stride of 2 from layer 3 to layer 6. We use $1 \times 1 \times 1 \times 20$ latent factors to generate dynamic patterns.

We use 64 frames for training in model-1 and model-2, and show various synthesis results in Fig. 5.11.

**Experiment 2. Qualitative comparison.** We compare our method with state space auto-regressive models [HLZ17, DCW03b] for dynamic textures. Since their methods are based on frame-wise modeling, we set the latent dimension to 20 for each frame as suggested in their papers. Fig. 5.12 shows several comparison results. It can be seen that frame-wise modeling can accumulate noises, rendering less sharp synthesis results as time evolves.

We also compare our method with 3D generative adversarial model [VPT16], i.e., 3D GAN. We use the code provided in their webpage. Fig. 5.13 shows one comparison result.

### 5.2 Quantitative experiment

In this section, numerical experimental results are provided for performance comparison.
### Table 5.3: Recovery errors in 5 experiments of learning from occluded images.

<table>
<thead>
<tr>
<th>experiment</th>
<th>P.5</th>
<th>P.7</th>
<th>P.9</th>
<th>M20</th>
<th>M30</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>.0571</td>
<td>.0662</td>
<td>.0771</td>
<td>.0773</td>
<td>.1035</td>
</tr>
</tbody>
</table>

### 5.2.1 Learning from incomplete data

**Experiment 1. Learning 2D generator from incomplete image.** The generator can learn from images with occluded pixels. This task is inspired by the fact that most of the images contain occluded objects. It can be considered a non-linear generalization of matrix completion in recommender system.

The generator can be adapted to this task with minimal modification. The only modification involves the computation of $\|Y - g(X; W_G)\|^2$. For a fully observed image, it is computed by summing over all the pixels. For a partially observed image, we compute it by summing over only the observed pixels. Then we can continue to use the alternating back-propagation algorithm to infer $X$ and learn $W_G$. With inferred $X$ and learned $W_G$, the image can be automatically recovered by $g(X; W_G)$. In the end, we will be able to accomplish the following tasks: (T1) Recover the occluded pixels of training images. (T2) Synthesize new images from the learned model. (T3) Recover the occluded pixels of testing images using the learned model.
Table 5.4: Recover errors for various incomplete training videos with different occlusion masks.

It’s worthy noting that all the training images are partially occluded. Such experiments are different from (1) de-noising auto-encoder [VLB08], where the training images are fully observed, and noises are added as a matter of regularization, (2) in-painting or de-noising, where the prior model or regularization has already been learned or given. (2) is about task (T3) mentioned above, but not about tasks (T1) and (T2).

Learning from incomplete data can be difficult for GAN and VAE, because the occluded pixels are different for different training images.

We evaluate our method on 10,000 images randomly selected from CelebA dataset. We design 5 experiments, with two types of occlusions: (1) 3 experiments are about salt and pepper occlusion, where we randomly place 3 × 3 masks on the 64 × 64 image domain to cover roughly 50%, 70% and 90% of pixels respectively. These 3 experiments are denoted P5, P7, and P9 respectively (P for pepper). (2) 2 experiments are about single region mask occlusion, where we randomly place a 20 × 20 or 30 × 30 mask on the 64 × 64 image domain. These 2 experiments are denoted M20 and M30 respectively (M for mask). We set \( d = 100 \).

Table 5.3 displays the recovery errors of the 5 experiments, where the error is defined as per pixel difference (relative to the range of the pixel values) between the original image and the recovered image on the occluded pixels. We emphasize that the recovery errors are not training errors, because the intensities of the occluded pixels are not observed in training.

<table>
<thead>
<tr>
<th>Methods</th>
<th>P.5</th>
<th>M20</th>
<th>B25x25x15</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D VAE</td>
<td>ours</td>
<td>3D VAE</td>
<td>ours</td>
</tr>
<tr>
<td>windmill</td>
<td>0.0442</td>
<td>0.0464</td>
<td>0.0430</td>
</tr>
<tr>
<td>flag1</td>
<td>0.0425</td>
<td>0.0807</td>
<td>0.0592</td>
</tr>
<tr>
<td>candle</td>
<td>0.0506</td>
<td>0.0454</td>
<td>0.0431</td>
</tr>
<tr>
<td>flamingo</td>
<td>0.0480</td>
<td>0.0464</td>
<td>0.0428</td>
</tr>
<tr>
<td>flag</td>
<td>0.0296</td>
<td>0.0375</td>
<td>0.0363</td>
</tr>
</tbody>
</table>
Fig. 5.14 displays recovery results. In experiment P.9, 90% of pixels are occluded, but we can still learn the model and recover the original images.

**Experiment 2. Learning 3D generator from incomplete videos.** To further evaluate the generator quantitatively, we perform experiment to learn from incomplete videos on 3D generator. We compare our model with the current state of the art generative adversarial model [VPT16]. We also provide a strong baseline model, i.e., 3D variational auto-encoder (3D VAE), based on the original variational auto-encoder [KW14, RMW14]. The 3D variational auto-encoder shares the same generator network structure as our model and 3D GAN [VPT16]. For the inference network for 3D VAE, it has the “mirror” structure as its generator network, with convolution and LeakyReLU (with ratio 0.2) to replace the deconvolution and typical ReLU used in the generator. For all three models, we set the latent factor dimension to 100. Similar to learning 2D generator from incomplete image, the 3D generator and the baseline method can be easily adapted to this task with simple modification in which we only consider the observed pixels for our objective function. To the best of our knowledge, there is no prior work on this task using video generative models, so we only compare our method with the implemented 3D VAE as the baseline.

We experiment on three types of occlusions: (1) salt and pepper mask (P) which covers roughly 50% (P.5) of the pixels per-frame. (2) 2D patch occlusion (P) where we randomly place a 20×20 (M20) mask on each frame. (3) 3D block occlusion (B) where we randomly place a 25×25×15 block in the whole video. Table 5.4 displays the recovery errors on various dynamic patterns for 3 different occlusion types. It can be seen that the inference-based methods are well suited for this task, and our method outperforms variational auto-encoder in terms of recovery errors.

### 5.2.2 Learning generator from indirect data

**Experiment 1. Learning generator from indirect data.** We can learn the model from the compressively sensed data [CRT06]. We generate a set of white noise images as random projections. We then project the training images on these white noise images. We can learn
<table>
<thead>
<tr>
<th>experiment</th>
<th>$d = 20$</th>
<th>$d = 60$</th>
<th>$d = 100$</th>
<th>$d = 200$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP</td>
<td>.0810</td>
<td>.0617</td>
<td>.0549</td>
<td>.0523</td>
</tr>
<tr>
<td>PCA</td>
<td>.1038</td>
<td>.0820</td>
<td>.0722</td>
<td>.0621</td>
</tr>
</tbody>
</table>

Table 5.5: Reconstruction errors on testing images, after learning from training images using our method (ABP) and PCA.

<table>
<thead>
<tr>
<th>experiment</th>
<th>$d = 20$</th>
<th>$d = 60$</th>
<th>$d = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>.0795</td>
<td>.0617</td>
<td>.0625</td>
</tr>
</tbody>
</table>

Table 5.6: Recovery errors in 3 experiments of learning from compressively sensed images.

the model from the random projections instead of the original images. We only need to replace $\|Y - g(X; W_G)\|^2$ by $\|SY - g(X; W_G)\|^2$, where $S$ is the given white noise sensing matrix, and $SY$ is the observation. We can treat $S$ as a fully connected layer of known filters below $g(X; W_G)$, so that we can continue to use alternating back-propagation to infer $X$ and learn $W_G$, thus recovering the image by $g(X; W_G)$. In the end, we will be able to (T1) Recover the original images from their projections during learning. (T2) Synthesize new images from the learned model. (T3) Recover testing images from their projections based on the learned model. Our experiments are different from traditional compressed sensing, which is task (T3), but not tasks (T1) and (T2). Moreover, the image recovery in our work is based on non-linear dimension reduction instead of linear sparsity.

Figure 5.15: Learning from indirect data. Row 1: the original $64 \times 64 \times 3$ images, which are projected onto 1,000 white noise images. Row 2: the recovered images during learning.
We evaluate our method on 1000 face images randomly selected from CelebA dataset. These images are projected onto $K = 1000$ white noise images with each pixel randomly sampled from $N(0, 0.5^2)$. After this random projection, each image of size $64 \times 64 \times 3$ becomes a $K$-dimensional vector. We show the recovery errors for different latent dimensions $d$ in Table 5.6 where the recovery error is defined as the per pixel difference (relative to the range of the pixel values) between the original image and the recovered image. Fig. 5.15 shows some recovery results.

5.2.3 Pattern reconstruction and completion

Experiment 1. Generator reconstruction error on testing images. After learning the generator from the training images (now assumed to be fully observed), we can evaluate the model by the reconstruction error on the testing images. We randomly select 1000 face images for training and 300 images for testing from CelebA dataset. After learning, we infer the latent factors $X$ for each testing image using inferential back-propagation, and then reconstruct the testing image by $g(X; W_G)$ using the inferred $X$ and the learned $W_G$. In the inferential back-propagation for inferring $X$, we initialize $X \sim N(0, I_d)$, and run 300 Langevin steps with step size .05. Table 5.5 shows the reconstruction errors of generator as compared to PCA learning for different latent dimensions $d$. Fig. 5.16 shows some reconstructed testing images. For PCA, we learn the $d$ eigenvectors from the training images, and then project
<table>
<thead>
<tr>
<th>Methods</th>
<th>Consecutive Block</th>
<th>Random Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LHD</td>
<td>ours</td>
</tr>
<tr>
<td>flag</td>
<td>0.1359</td>
<td>0.0392</td>
</tr>
<tr>
<td>waterfall</td>
<td>0.2720</td>
<td>0.2558</td>
</tr>
<tr>
<td>waterstone</td>
<td>0.1899</td>
<td>0.1321</td>
</tr>
<tr>
<td>light</td>
<td>0.0728</td>
<td>0.0725</td>
</tr>
<tr>
<td>elevator</td>
<td>0.1043</td>
<td>0.0768</td>
</tr>
<tr>
<td>candle</td>
<td>0.0598</td>
<td>0.0728</td>
</tr>
</tbody>
</table>

Table 5.7: Interpolation errors for various training videos with different occlusion masks.

Figure 5.17: Interpolation results. The first row displays the occluded training sequence. The second row displays the interpolation results by our method. The third row shows the interpolation results by using Laplacian heat diffusion.

the testing images on the learned eigenvectors for reconstruction.

**Experiment 2.** Video interpolation of time-stationary patterns by 3D generator. For video sequences with repetitive patterns in the temporal domain, our method can efficiently learn such patterns by assuming $X = X(m, t)$ even on a single training sequence. We evaluate our method by interpolation. We consider two types of missing patterns. The first type is consecutive block, in which we entirely block 8 consecutive frames in the middle of the training sequence. The second type is random block, in which we entirely block 3 consecutive frames in 3 randomly chosen positions in the training sequence. Each interpolation experiment is performed on one video training sequence. Note that the current 3D GAN model [VPT16]
cannot easily do this, because the spatial-temporal non-stationary structure is used in their paper and the it lacks of the inference mechanism. So we compare our approach with the widely used Laplacian Heat Diffusion method (LHD) which we extend the 2D Laplacian kernel to 3D version, and we use 500 sweeps for the video interpolation with learning rate $0.5$. Table 5.7 shows the recovery errors for various kinds of dynamic patterns and different missing patterns. The recovery error is defined as per pixel difference between the ground truth video and the recovered video on the occluded pixels. Fig. 5.17 shows the interpolation results on the consecutive block pattern. It can be seen that 3D generator can get sharper and more realistic results.
Experiment 3. **Video recovery by 3D generator.** For this task, we split the training/testing video sequences by about 4:1 in proportion, and randomly block the testing videos using different occlusion patterns. We first learn the spatial-temporal generator network using the clean training videos. Based on the learned model, we do inference only on unblocked pixels of testing videos to infer the latent factors, and then feed the inferred latent factors into the learned generator network to complete the videos. For adversarial trained network [VPTI16], we apply the inferential mechanism powered by Langevin dynamic for direct comparison with our method. For 3D VAE, we use its own learned inference network and iteratively impute the occluded pixels during inference stage [RMW14].

We again experiment on three types of occlusions: (1) salt and pepper mask (M) which covers roughly 50% (P.5) or 90% (P.9) of the pixels per-frame. (2) 2D patch occlusion (P) where we randomly place a 20×20 (M20) or 40×40 (M40) mask on each frame. (3) 3D block occlusion (B) where we randomly place a 35×35×20 block in the whole video. We learn the model using the same Adam optimizer and the same number of iterations (i.e., 500). For the inference stage on occluded testing videos, we run the inference step for 200 iterations with step size 0.05 for Langevin dynamic. Table 5.8 shows recovery errors for various kinds of dynamic patterns and different occluding masks.

Experiment 4. **Images recovery by CoopNets.** We conduct an experiment on learning from training images of human faces, and then testing the learned model on completing the occluded testing images. The structure of the generator network is the same as in [RMC15, DSB15]. We adopt a 4-layer descriptor net. The first layer has 96 5×5 filters with sub-sampling of 2, the second layers has 128 5 ×5 filters with sub-sampling of 2, the third layer has 256 5 ×5 filters with sub-sampling of 2, and the final layer is a fully connected layer with 50 channels as output. We use \( l_D = 10 \) steps of Langevin revision dynamics within each learning iteration, and the Langevin step size is set at 0.002. The learning rate is 0.07. The training data are 10,000 human faces randomly selected from CelebA dataset [LLW15]. We run 600 cooperative learning iterations.

To quantitatively test whether we have learned a good generator net \( g(X; W_G) \) 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 the term $\|Y - g(X; W_G)\|^2$, where the sum of squares is over all the observed pixels of $Y$ in back-propagation computation. We run 1000 Langevin steps, initializing $X$ from $N(0, I_d)$. After inferring $X$, the completed image $g(X; W_G)$ is automatically obtained. We design 3 experiments, where we randomly place a $30 \times 30$, $40 \times 40$, or $50 \times 50$ mask on each $64 \times 64$ testing image. These 3 experiments are denoted by M30, M40, and M50 respectively.

We report the recovery errors and compare our method with 8 different image inpainting methods as well as the DCGAN of \cite{RMC15}. For DCGAN, we use the parameter setting in \cite{RMC15} except changing the number of learning iterations to 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 latent factors $X$, and recover the unobserved pixels. In the 8 in-painting methods, Methods MRF-$\ell_1$ and MRF-$\ell_2$ are based on Markov random field prior where the nearest neighbor potential terms are $\ell_1$ and $\ell_2$ differences respectively. Methods inter-1 to 5 are interpolation methods. Please refer to \cite{DE04} for more details.
CHAPTER 6

Conclusion

My thesis generalizes two major classes of statistical models by leveraging ConvNets and develop a cooperative learning algorithm to connect two models. The descriptor, as a generalized exponential family model, defines statistical feature by a bottom-up ConvNet. The generator, as a generalized latent factor model, defines a hierarchical non-linear basis function by a top-down convNet. The CoopNets algorithm builds connection between the descriptor and generator showing that the generator serves as an accumulated MCMC process, and can provide unlimited number of training samples to the descriptor.

The development of descriptor is very natural, almost axiomatic, with minimal extra assumptions. Assuming the commonly used ReLU non-linearity and a Gaussian white noise base category, the descriptor is naturally derived from the discriminative ConvNet, and the model has a representational structure based on multiple layers of binary activation variables. It is also empirically shown that the model can reconstruct the observed images and synthesize new images. It is possible to generalize the definition of the binary activation variables to take into account explaining-away competitions for learning sharp dictionaries of filters and basis functions at multiple layers.

Some of the results from the descriptor can be mapped to back-propagation in the discriminative ConvNet, but reinterpretation of them in terms of representation is novel and is richly expansive. The descriptor unifies, reconciles or connects the following antagonizing or disparate pairs: (1) discriminative ConvNet and generative perspective CongNet, (2) supervised learning and unsupervised learning, (3) exponential family models and latent variable models, (4) bottom-up filters (operation) and top-down basis functions (representation), (5) synthesis (dream) and reconstruction (memory), (6) hierarchal probability model and hi-
erarchical auto-encoder, (7) Hopfield attractor network and auto-encoder, (8) contrastive divergence (learning) and reconstruction (memory).

The generator proposes an alternating back-propagation algorithm for training the generator network. The generator network can be recognized as a non-linear generalization of the factor analysis model, and the alternating back-propagation algorithm as the non-linear generalization of the alternating regression scheme of the Rubin-Thayer EM algorithm for fitting the factor analysis model. The alternating back-propagation algorithm iterates the inferential back-propagation for inferring the latent factors and the learning back-propagation for updating the parameters. Both back-propagation steps share most of their computing steps in the chain rule calculations.

The algorithm of generator is perhaps the most canonical algorithm for training the generator network. It is based on maximum likelihood, which is theoretically the most accurate estimator. The maximum likelihood learning seeks to explain and charge the whole dataset uniformly, so that there is little concern of under-fitting or biased fitting.

As an unsupervised learning algorithm, the alternating back-propagation algorithm is a natural generalization of the original back-propagation algorithm for supervised learning. It adds an inferential back-propagation step to the learning back-propagation step, with minimal overhead in coding and affordable overhead in computing. The inferential back-propagation seeks to perform accurate explaining-away inference of the latent factors. It can be worthwhile for tasks such as learning from incomplete or indirect data, or learning models where the latent factors themselves follow sophisticated prior models with unknown parameters. The inferential back-propagation may also be used to evaluate the generators learned by other methods on tasks such as reconstructing or completing testing data.

The most unique feature of the CoopNets algorithm is that the two 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 the CoopNets algorithm is that the learning process interweaves
the existing maximum likelihood learning algorithms for the two networks, so that the two
algorithms jump-start each other’s MCMC sampling.

A third unique feature of our work is that the MCMC for the descriptor keeps rejuvenating
the chains by refreshing the samples by independent replacements supplied by the generator,
so that a single chain effectively amounts to an infinite number of chains or the evolution of
the whole marginal distribution. Powering the MCMC sampling of the descriptive models
in descriptor is the main motivation of the CoopNets algorithm, with the bonus of turning
the unsupervised learning of the generator into supervised learning.

The descriptor has the potential to learn from unlabeled data. In the future work, I
shall scale up the unsupervised learning from big unlabeled data using reconstruction based
methods. The generator its variants can be applied to non-linear matrix factorization and
completion. It can also be applied to problems where some components or aspects of the
factors are supervised.
REFERENCES


93


