Title
Learning convolutional latent space energy-based prior model

Author
Xu, Dehong

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Learning convolutional latent space energy-based prior model

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

by

Dehong Xu

2021

ABSTRACT OF THE THESIS

Learning convolutional latent space
energy-based prior model

by

Dehong Xu
Master of Science in Statistics
University of California, Los Angeles, 2021
Professor Ying Nian Wu, Chair

Learning good representations without supervision remains a key challenge in machine learning. We proposed to learn an energy-based model (EBM) in the latent space that stands on the deep generative model. Different from the original latent vector space, we formulate a convolutional feature map EBM in the prior. Using short-run MCMC sampling from the prior and posterior distributions of the latent vector, both the prior EBM and the generator model are learned jointly. Using the convolutional EBM method allows the model to exhibit strong performances in terms of image generation and capture more explainable representations.

The thesis of Dehong Xu is approved.

Song-chun Zhu

Tao Gao

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2021

To my mother and father

and all the others who have encouraged and

supported me ...

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

One potential way to increase data efficiency and generalization of machine learning methods is to train generative models [GPM14] [KW13]. In recent years, deep generative models have achieved impressive results and applications in image, audio, videos, and text generation. Also, Many types of generative models have flourished in recent years, including likelihood-based generative models, which include autoregressive models [UML13], variational autoencoders (VAEs) [KW13] [RMW14], and invertible flows [DKB14] [DSB16]. These models can generalize great representations in an unsupervised way without any labeled data. At the same time, the learned representations can be used in many challenging downstream tasks, such as few-shot learning or reinforcement learning.

The generative model assumes that observed data is generated from a low-dimensional latent space, which often follows a non-informative prior distribution. Although we can learn a powerful top-down network to map the latent space to the image space, another way is to make the prior more expressive. Instead of generating data from Gaussian noise, we can assume the prior distribution follows an energy-based model (EBM), which we call it a latent space EBM [PHN20] [PNC20]. To gain more explainable representations in the latent space, we change the latent vector into a feature map, which may give some texture/texton features as in Filters, random fields, and maximum entropy (FRAME) [ZWM98] [ZM98].

Moreover, in order to yield more explainable representations, we hope to make more effective use of the latent space. Unlike the original dense latent vector in the variational autoencoder (VAE) [KW13] framework, we formulated the latent space as a feature map. In this way, each patch in the feature map would contain more explainable semantic representations.

Both the latent space EBM and the generator network can be learned jointly by maximum likelihood estimate (MLE) or its approximate or variational variants. Each learning iteration involves Markov chain Monte Carlo (MCMC) sampling of the latent vector from both the prior and posterior distributions. Specifically, both the prior and posterior can be sampled by a short-run MCMC [NHZ19] [NPH20], which only runs a fixed number of MCMC iterations from a fixed initial distribution.

The organization of the thesis is as follows. Chapter 2 provides some related works to introduce the background of our work briefly. Then, in Chapter 3, we move into our model construction, how we train the model, and some details. And for Chapter 4, some results are shown in this part, and there also includes some empirical analysis to show how well our model works. Finally, we will draw conclusions based on our methods, experiments, and results in Chapter 5.

CHAPTER 2

Background and Related Work

In this chapter I will describe several previous works that my project is built upon, to provide a context for my methodology.

2.1 Generative model

Generative modeling is currently one of the most popular unsupervised learning methods, which is widely used in computer vision, natural language processing (NLP) and audio generation, etc. In unsupervised learning, the data is provided without any label or feature. Unlike supervised learning tasks, such as classification and regression problems, which try to formulate a $f(x)$ to predict an output label y, in the generative model, we only have raw data x. A general goal for generative modeling or representation learning is to represent the input x by some hidden representations, which can capture some key information, structure, or patterns of x. And hopefully, the representation can be simple, explainable, and easy to understand. This means we need to figure out how the raw data is generated by only given data x itself. In a generative model, we only deal with the real data distribution $P(x)$.
Usually, the model is defined on a high-dimensional data space, such as image space. For natural images that we observe in real life, there are very complex dependencies between pixels. For example, in the same image, the pixels that are near to each other are highly likely to have a similar color. Also, in natural image data, if we want to generate hand-written digits from 0 to 9, in the pixel level, the value of each pixel should be partially decided by the value from other pixels. Say if given half of the written number image, it can help us

determine which digit we will generate so that we can get at least get some clues about the specific value of the rest of the pixels. In other words, in the natural image domain, all the possible images are not uniformly distributed; otherwise, if we drew samples from the image space, almost all of the samples would be just random noise, which we never observe in the real world. In this case, our model needs to capture the hidden information that governs these underlying dependencies so that the model can give images that are more likely to be seen in the real world higher probability.

As a result, for natural images, there should be some basic rules, principles, or properties that the data distribution follows. Considering how human brains process natural image data and how humans understand images, our brain should not process visual signals on the pixel level. Instead, when people try to memorize a single image, our brains could not store all the pixel values, but instead, they may generalize some key features or representations. With these important elements, we can even reconstruct the image in our brain. Following this idea, we assume that besides the data space, there is still a latent space that contains some hidden information about the real data. Similarly, to determine the raw data distribution, the generative model assumes there is a vector of hidden variables z that is used to generate x. Since the high-dimensional data space contains many complex dependencies, we assume the latent space to be a low-dimensional space. And for the latent variables, it is not only seen as the distributed representation of the data but also considered the embedding of the data. The vector can be interpreted as a code, which generalizes the information from the high-dimensional data space.

Further, the latent or hidden space should also have some basic properties. As an embedding of the data, we wish this representation can capture the important properties of the data and help generate new images that are not included in the training set. Because we assume the representation can general some underlying explainable information, by changing the value of the hidden vector, we wish the generated image can also change in a controllable direction. For example, if some dimensions in the latent vector govern the image's light,

then by changing the value of these dimensions, we could get images of the same object but under different lighting conditions. For a generative model, one of the most important goals is the ability to generalize and generate new data that are similar but not identical to the original data by manipulating the hidden vector. In this way, even though we only have a limited number of samples in the training set, we can generate unlimited images as long as we learn a strong and reasonable representation. In other words, we need to learn a strong mapping between the latent space and the data space, so that as long as we walk along with the latent space, the generated data also walks along with the data distribution in some ways.

2.2 Variational autoencoder

Variational Autoencoders (VAEs) [KW13] [Doe16] is one of the most popular generative models nowadays. To learn the projection between latent space and image space, the generative model is a decoder. And we can also pair it with a bottom-up encoder. In this model, we can learn both the decoder and the encoder jointly. The main structure is shown in Figure 2.1 below.

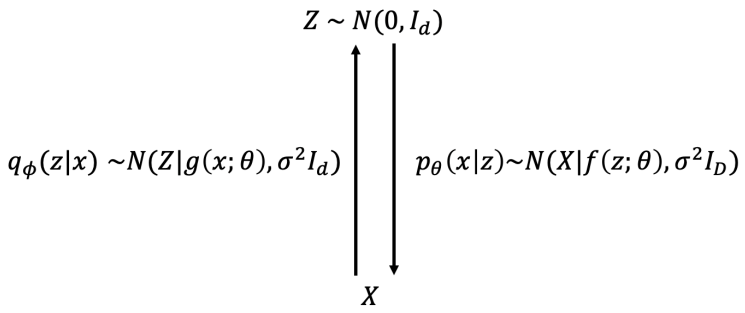

Figure 2.1: Variational autoencoder model.

As we can see in Figure 2.1, the variational autoencoder consists of a latent space and a

data space. Based on the assumption, the data is generated from a low-dimensional vector space. We can build a top-down model, which is often parameterized by a neural network, to accept a sample from the latent space, and then it can output a high-dimensional data point. However, to learn the model, a top-down model may not be enough. We also need to do inference that is to bottom-up mapping from the data space to the latent space. In VAE, we are going to learn both the generative model and the inference model jointly.

Suppose that there is no simple interpretation of the dimensions of z, and instead, samples of z can be drawn from a simple distribution that is standard Gaussian $N(0, I_d)$, which we denote as the prior distribution $p_\theta(z)$. Then we have a deterministic function $f(z;\theta)$, where θ represents all the parameters, and $f : Z \times \Theta \to X$. We hope that by sampling from the simple prior distribution, we can optimize θ such that we can get reasonable X using the generative function $f(z;\theta)$. In other words, we wish to approximate the data distribution by making $f(z;\theta)$ similar to X in the dataset. Formally, the approximation of the data distribution can be written as:

$$P_\theta(X) = \int P_\theta(X|z)P(z)dz$$

In VAE, for simplicity, we assume the conditional distribution to be Gaussian that is $P_\theta(X|z) \sim N(X|f(z;\theta), \sigma^2 I_D)$. In this way, we can use gradient descend to maximize the likelihood by letting $f(z;\theta)$ approach the original X. In the meantime, we also need a variational inference model $q_\phi(z|x)$ to approximate the posterior. In this way, the objective is to minimize the distance between the true posterior and the approximate posterior, which also means we wish the inference model to be close to the true posterior distribution. Therefore, VAE minimizes the Kullback-Leibler divergence (KL divergence) between $q_\phi(z|x)$ and $p_\theta(z|x)$ and apply Bayes rules:

$$
\begin{aligned}
D_{KL}[q_\phi(z|x)||p_\theta(z|x)] &= E_{q_\phi(z|x)}[log\frac{q_\phi(z|x)}{p_\theta(z|x)}] \\
&= logp(x) - [E_{q_\phi(z|x)}logp_\theta(x,z) - E_{q_\phi(z|x)}logq_\phi(z|x)] \\
&= logp(x) - [E_{q_\phi(z|x)}logp_\theta(x|z) - D_{KL}[q_\phi(z|x)||p_\theta(z)]]
\end{aligned}
$$

As we can see above, the first term $logp(x)$ is the data likelihood that we hope to maximize. Minimizing the KL divergence $D_{KL}[q_\phi(z|x)||p_\theta(z|x)]$ is equivalent to maximize the second term above that is $E_{q_\phi(z|x)}logp_\theta(x|z) - D_{KL}[q_\phi(z|x)||p_\theta(z)]$. Because KL divergence is always greater or equal to 0, the data likelihood is larger to equal to the second term, which means it is a lower bound. Since the direct likelihood is hard to maximize, maximizing its lower bound can also be a useful tool for getting a rough idea of how well the model captures a particular datapoint X. And we usually called it the evidence lower bound (ELBO).

Above all, to learn the mapping between these two variables, VAE has a top-down generator $p_\theta(x|z)$, a prior $p_\theta(z)$ and a bottom-up approximate posterior $q_\phi(z|x)$. Both the encoder and decoder are neural networks, which are parameterized by θ and ϕ respectively. Also, all the parameters in the neural network can be trained through back-propagation and reparameterization trick in order to maximize the evidence lower bound (ELBO):

$$
logp_\theta(x) \geq E_{q_\phi(z|x)}[p_\theta(x|z)] - KL[q_\phi(z|x)||p_\theta(z)]
$$

In variational autoencoders, we assume the prior $p_\theta(z)$, the posterior $q_\phi(z|x)$ and the $p_\theta(x|z)$ all follow Gaussian distributions. In this way, we can easily draw samples from them and also make the KL term in the loss function easy to compute.

2.3 Energy-based model

As we have discussed previously, the main purpose of representation learning is to encode dependencies between variables. By capturing those dependencies, a model can be used to answer questions about the values of unknown variables given the values of known variables. Energy-Based Models (EBM) [Ng11] [LCH06] capture dependencies by associating scalar energy (a measure of compatibility) to each configuration of the variables.

In variational autoencoder, due to the computational simplicity, the model uses fixed prior, which is a standard Gaussian distribution to formulate the latent vector. This assumption indeed can make the model have fast sampling speed and easy computation when calculating the KL divergence term. However, for the reason that all the samples are drawn from the same unchangeable prior distribution, the model lacks flexibility. Even though VAE can learn a relatively good mapping between the latent space and the data space, the reconstructed and generated images are often blurred. Thus, dealing with this drawback of VAE, we are thinking about how to build a more flexible and learnable prior distribution for the latent space. And in the meantime, the sampling process and computation are manageable, which is one of the reasons why we plan to use EBM.

The architecture of the EBM is the internal structure of the parameterized energy function $E(\cdot)$, where the energy function could be as simple as a linear combination of basis functions (as in the case of kernel methods), or a set of neural net architectures and weight values. However, because the prior model needs to be a probability distribution, the only consistent way involves normalizing the collection of energies for all possible outputs. The simplest and most common method for turning a collection of arbitrary energies into a collection of numbers between 0 and 1 whose sum (or integral) is 1 is through the Gibbs distribution [Zhu03]:

$$p(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}}$$

where β is an arbitrary positive constant akin to an inverse temperature, and the denominator is called the partition function (by analogy with similar concepts in statistical physics). The definition shown above is a general case involving both x and y, and it can be used in different forms. Basically, EBM aims to learn an energy function $E(\cdot)$ that gives low energy to the input, which is highly likely to appear in the data set, while giving high energy to the input that does not follow the real input data distribution. Then, the exponential term $e^{-E(\cdot)}$ can give a high probability to the data that contains in the training set but a low probability to the rest of the inputs. In representation learning, we only want the distribution defined by $E(\cdot)$ to model the latent variables. If we absorb the parameter β into the energy function, the formulation will look like below:

$$p(z) = \frac{e^{-E(z)}}{\int_{z \in \mathcal{Z}} e^{-E(z)}}$$

Because the latent space in our model remains a continuous space, which is the same as VAE, we use integral to compute the normalizing constant in the denominator. We often use $Z(\cdot)$ to represent the normalizing term. Since the normalizing constant is the integral w.r.t. z, its value would only depend on the parameters. This means that if we fix the value of the parameters, the normalizing term is a constant, and it will not influence the distribution anymore. Then we may input the value of z to get its probability.

CHAPTER 3

Method

In this chapter, I will talk about our model and how we train the model. First, I will start with some basic assumptions and include the main structure of the model. Next, I will derive the loss function from the maximum likelihood perspective. Also, I will talk about some details about how to train the model in a manageable way.

3.1 Model

Let $x \in R^D$ be the observed data examples, such as images, and $z \in R^d$ be latent variables. Suppose we have the following distribution,

$$z \sim p_\alpha(z),\ x \sim p_\beta(x|z)$$

where $p_\alpha(z)$ is the prior model and $p_\beta(x|z)$ is the top-down decoder model. α and β are learnt parameters in these two models. In the classic VAE model, the prior model $p(z)$ just follows a standard Gaussian distribution, which is not informative. To build a more expressive model, we need to have a stronger prior. Thus, in our model, the prior $p_\alpha(z)$ is formulated as an energy-based model or a Gibbs distribution,

$$p_\alpha(z) = \frac{1}{Z(\alpha)} exp(f_\alpha(z)) p_0(z)$$

where $f_\alpha(z)$ is a multi-layer perceptron (MLP) parameterized by α and $p_0(z)$ is a reference distribution, assumed to be isotropic Gaussian as in VAE. $Z(\alpha)$ is a normalizing constant,

which can be calculate by

$$Z(\alpha) = \int exp(f_\alpha(z))p_0(z)dz = E_{p_0(z)}[exp(f_\alpha(z))]$$

We can regard the energy term $exp(f_\alpha(x))$ as a exponential tilting of the original standard normal prior distribution $p_0(z)$. In this way, the energy-based model can approximate many kinds of complicated distributions by changing the values of the neural network parameter α. With this more informative prior, the model can generalize different representations based on various training data. Also, in this model, we made the latent space to be a convolutional feature map. The latent vector z is defined on a 2D plain and on each position of the 2D plain, there is a vector. The advantage of this method is that we also give the latent space some spatial and structural properties. In this way, the model may capture some sketch, texture, or texton patterns. For example, if a vector from a single position in latent 2D plain controls a particular part of the data space, say the bottom-right corner of the image. Then if we change the vector, the bottom-right corner of the image may also change respectively. Furthermore, different dimensions in the vector may represent some properties of the specific part of the image patch, such as light, size of the object, or color. In this case, as long as we change the specific dimension of the latent vector, we can manipulate the image patch in the way that we want.

For the generator model, it is identical to the decoder in the VAE,

$$x \sim N(g_\beta(z), \sigma^2 I_D)$$

The generator is often called a decoder, which is defined as a conditional distribution. As is shown above, we assume the top-down conditional distribution as a Gaussian distribution with mean $g_\beta(z)$ and variance σ^2. The generator is a de-convolutional neural network parameterized by β. The decoder receives a sample from the latent space and gives the mean of the generated data. After sampling from the Gaussian distribution, we can get the generated image.

Thus, as is shown in Figure 3.1, the latent variable Z follows an energy-based prior distri-

bution $p_\alpha(z)$. The probability of each specific latent vector z is calculated by going through a simple multi-layer perceptron (MLP) $f_\alpha(z)$. Since the distribution is formed by using a neural network, we can never know either what exactly the distribution looks like or what is the exact probability of each sample, for the reason that the normalizing constant is intractable. Instead, we can only get the energy for each z, if we input the sample into the neural network. In other words, we cannot directly draw samples from the distribution.

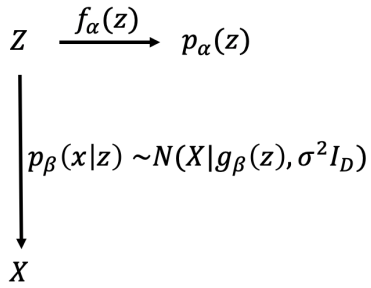

Figure 3.1: EBM prior and top-down generative model.

To sample from x, we can use reparameterization tricks $x = g_\beta(z) + \epsilon$, where $\epsilon \sim N(0, \sigma^2 I_D)$. However, a sample from z may not be easy because we can't directly calculate the integral in the normalizing constant $Z(\alpha)$. So we may need to run a MCMC to approximate the prior distribution, which will mention in the following section.

The marginal distribution is

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\alpha(z)p_\beta(x|z)dz$$

And the posterior distribution is

$$p_\theta(z|x) = \frac{p_\theta(x, z)}{p_\theta(x)} = \frac{p_\alpha(z)p_\beta(x|z)}{\int p_\alpha(z)p_\beta(x|z)dz}$$

3.2 Maximum likelihood

Imagine we have training data $x_1, x_2, ..., x_n$. The log-likelihood is going to be:

$$L(\theta) = \sum_{i=1}^{n} logp_\theta(x_i)$$

where $\theta = (\alpha, \beta)$ represents all the trainable parameters in both the prior and the top-down generator model.

Next, because we need to use gradient ascent to optimize our log-likelihood shown above, we have to get the gradient w.r.t. each parameter included. But first, we need to go through several simple identities, which will help compute the log-likelihood gradient.

For any random variable $x \sim p_\theta(x)$, an equation $E_\theta[\bigtriangledown_\theta logp_\theta(x)] = 0$ always holds. The proof is shown below:

$$E_\theta[\bigtriangledown_\theta logp_\theta(x)] = \int p_\theta(x) \bigtriangledown_\theta logp_\theta(x)dx = \int \bigtriangledown_\theta p_\theta(x)dx = \bigtriangledown_\theta \int p_\theta(x)dx = 0$$

where the reason why integral and the gradient can change positions is that the integral is w.r.t. the ransom variable x, while the derivative is dealing with the parameter θ. And the whole thing equals to 0 is because $\int p_\theta(x)dx = 1$ and the gradient of a constant number should be 0.

Based on the equation above, we can also prove that for two random variables $x, z \sim p_\theta(x, z)$, $E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(x, z)] = \bigtriangledown_\theta logp_\theta(x)$. The detailed proof is shown below:

$$\begin{aligned} E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(x, z)] &= E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(z|x)] + E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(x)] \\ &= 0 + E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(x)] \\ &= \bigtriangledown_\theta logp_\theta(x) \end{aligned}$$

Thus, the gradient can be computed as follows:

$$\bigtriangledown_\theta logp_\theta(x) = E_{p_\theta(z|x)}[\bigtriangledown_\theta logp_\theta(x, z)] = E_{p_\theta(z|x)}[\bigtriangledown_\theta(logp_\alpha(z) + logp_\beta(x|z))]$$

Then, we can derive the gradient for α and β separately. For α from the prior model, if we just consider the gradient w.r.t. α, the second term does not contain anything that related to α, so we only need to care about the first term:

$$\bigtriangledown_\alpha log p_\theta(x) = E_{p_\theta(z|x)}[\bigtriangledown_\alpha log p_\alpha(z)]$$

In this case, the expectation under $p_\theta(z|x)$ can be sampled from the encoder network, so we only need to think about how to get the derivative term inside the expectation.

$$\begin{aligned}\bigtriangledown_\alpha log p_\alpha(z) &= \bigtriangledown_\alpha [f_\alpha(z) + log p_0(z) - log Z(\alpha)] \\ &= \bigtriangledown_\alpha [f_\alpha(z) - log Z(\alpha)]\end{aligned}$$

For the normalizing term $logZ(\alpha)$, applying the identity that we got previously,

$$\begin{aligned}E_{p_\alpha(z)}[log \bigtriangledown_\alpha p_\alpha(z)] &= E_{p_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z) - \bigtriangledown_\alpha log Z(\alpha)] \\ &= E_{p_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z)] - \bigtriangledown_\alpha log Z(\alpha)\end{aligned}$$

Also, we have proved that $E_{p_\alpha(z)}[log \bigtriangledown_\alpha p_\alpha(z)] = 0$. Thus,

$$E_{p_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z)] = \bigtriangledown_\alpha log Z(\alpha)$$

So if we substitute $\bigtriangledown_\alpha log Z(\alpha)$ by $E_{p_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z)]$, the gradient for α is

$$\begin{aligned}\delta_\alpha(x) = \bigtriangledown_\alpha log p_\theta(x) &= E_{p_\theta(z|x)}[\bigtriangledown_\alpha f_\alpha(z) - \bigtriangledown_\alpha log Z(\alpha)] \\ &= E_{p_\theta(z|x)}[\bigtriangledown_\alpha f_\alpha(z)] - E_{p_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z)]\end{aligned}$$

where we need to sample z from both the posterior $p_\theta(z|x)$ and the prior $p_\alpha(z)$ to get the updated α. Because $f_\alpha(z)$ is a feed-forward neural network, where α is the parameter, the gradient $\bigtriangledown_\alpha f_\alpha(z)$ can be directly computed through back-propagation.
For the generative model, to update β

$$\delta_\beta(x) = \bigtriangledown_\beta log p_\theta(x) = E_{p_\theta(z|x)}[\bigtriangledown_\beta log p_\beta(x|z)]$$

where $p_\beta(x|z)$ follows a Gaussian distribution for image data, so that $log p_\beta(x|z) = ||x - g_\beta(z)||^2/(2\sigma^2) + constant$. Both the gradient for α and β can be computed in close form or

derived by back-propagation. This means as long as we can get samples from the posterior and the prior distribution, we can use gradient descent to optimize the loss function.
In the expectation terms, both the prior model $p_\alpha(z)$ and the posterior distribution $p_\theta(z|x)$ require MCMC sampling. And here we use Langevin dynamics,

$$z_{k+1} = z_k + s \bigtriangledown_z log\pi(z_k) + \sqrt{2s}\epsilon_k$$

where $\pi(z)$ is the target distribution that we hope to sample from and k is the index for iteration. s is the step size for Langevin dynamics and $\epsilon_k \sim N(0, I_d)$, which is just a standard Gaussian noise. Suppose we need to draw samples from $p_\alpha(z)$ and $p_\theta(z|x)$. What we are going to do is just let them be the target distribution and run a iteration to approximate. What also need to mention is that $\bigtriangledown_z log\pi(z_k)$ are all computed through back-propagation in each case.

3.3 Short-run MCMC

EBM learned in data space such as image space [GLZ18] can be highly multi-modal, and MCMC sampling can be difficult. Theoretically, it would take infinite steps to get the target distribution using the iterative updating rule, which is impractical. However, we propose to use a short-run MCMC not only to approximate the target distribution but also to make the computation affordable.
Suppose we always start from a fixed simple distribution $p_0(z)$, then we update it using Langevin dynamics and we only do it for K steps, e.g. $K = 20$,

$$z_0 \sim p_0(z),\ z_{k+1} = z_k + s \bigtriangledown_z log\pi(z_k) + \sqrt{2s}\epsilon_k,\ k = 1, 2, ..., K$$

Denote the distribution of z_K to be $\tilde{\pi}(z)$. We put the ˜ sign on top of the symbols to denote distributions or quantities produced by short-run MCMC, and for simplicity, we omit the dependence on K and s in notation.

So that the learning gradients change to

$$\delta_\alpha(x) = E_{\tilde{p}_\theta(z|x)}[\bigtriangledown_\alpha f_\alpha(z)] - E_{\tilde{p}_\alpha(z)}[\bigtriangledown_\alpha f_\alpha(z)]$$

$$\delta_\beta(x) = E_{\tilde{p}_\theta(z|x)}[\bigtriangledown_\beta log p_\beta(x|z)]$$

Above are the update rules for α and β, and the expectations are approximate by the short-run MCMC.

Next, to get the approximation of the posterior distribution $\tilde{p}_\theta(z|x)$ and the prior distribution $\tilde{p}_\alpha(z)$, we should use Langevin dynamics that shown above. Langevin dynamics is an iterative process. We first draw a sample from the fixed standard Gaussian distribution $p_0(z)$ to be our initial z_0. And then we can update the value of z by $z_{k+1} = z_k + s \bigtriangledown_z log\pi(z_k) + \sqrt{2s}\epsilon_k$, where ϵ_k is just a Gaussian random noise that follows $N(0, I_d)$, s is the step size which is also a hyper-parameter, and k is the index of iterations.

Thus, for the reason that the value of z_k is given from the previous iteration, the only term that we are going to compute is the gradient term $\bigtriangledown_z log\pi(z_k)$. Since we are going to sample from both the prior and the posterior, we will let the target distribution $\pi(z_k)$ be $p_\alpha(z)$ and $p_\theta(z|x)$ respectively. So for the prior, we get:

$$\begin{aligned}\bigtriangledown_z log p_\alpha(z) &= \bigtriangledown_z [f_\alpha(z) + log p_0(z) - log Z(\alpha)] \\ &= \bigtriangledown_z f_\alpha(z) + \frac{z}{\sigma^2}\end{aligned}$$

And for posterior,

$$\begin{aligned}\bigtriangledown_z log p_\theta(z|x) &= \bigtriangledown_z log \frac{p_\beta(x|z) \cdot p_\alpha(z)}{p_\theta(x)} \\ &= \bigtriangledown_z [-\frac{||x - g_\beta(z)||^2}{2\sigma^2} + f_\alpha(z) + log p_0(z)]\end{aligned}$$

As we can see above, $\bigtriangledown_z f_\alpha(z)$ can be computed by back-propagation. We can get both $\bigtriangledown_z log p_\alpha(z)$ and $\bigtriangledown_z log p_\theta(z|x)$ easily. So as long as we get the derivatives, we can use them to approximate the target distribution.

CHAPTER 4

Experiments and results

In this chapter, I will explain present a set of experiments that highlight the effectiveness of my model and show some results. In this paper, we focus on image space and we include SVHN [NWC11], CIFAR-10 [KNH], and CelebA [LLW15] as my datasets.

4.1 Image reconstruction

Images contain much redundant information as most of the pixels are correlated and noisy. Therefore we use a latent space to generalize the key information while discarding the redundant information.

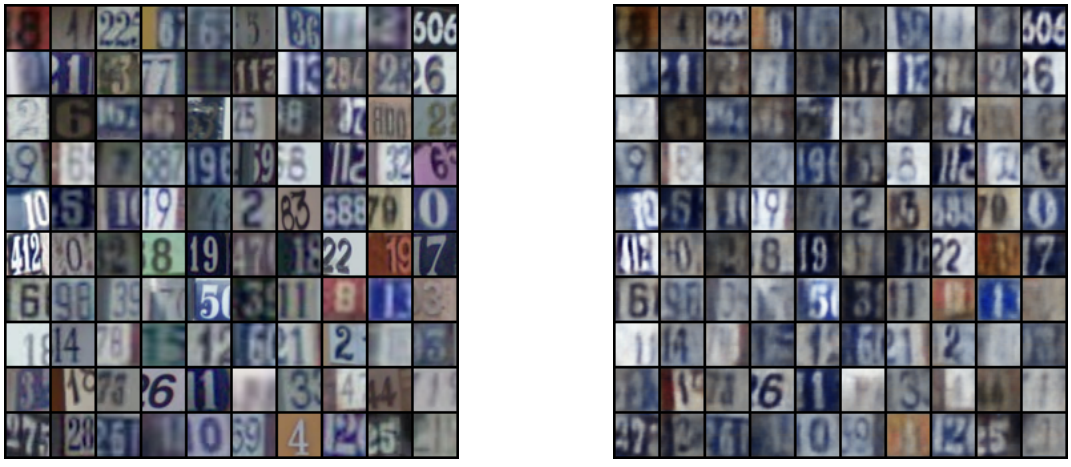

Figure 4.1: Left: SVHN 32x32x3 images, right: reconstructions from our model with a 4x4x100 latent space

To illustrate that the latent vector can extract valuable information out, we can do a reconstruction by purely using the extracted information. Also, to evaluate the accuracy of the posterior inference, we test it by looking at how the image reconstruction task works. If we have a test data example from the data distribution, our well-formed posterior Langevin process should give us a cogent approximate of the posterior and return a representation of that data example in the latent space. In this way, besides helping to learn the latent space EBM model, Langevin dynamics also helps to learn the true posterior $p_\theta(z|x)$ of the top-down generative model. We can then compare the reconstructions of test images with the original data to see how the inference and the generator works.

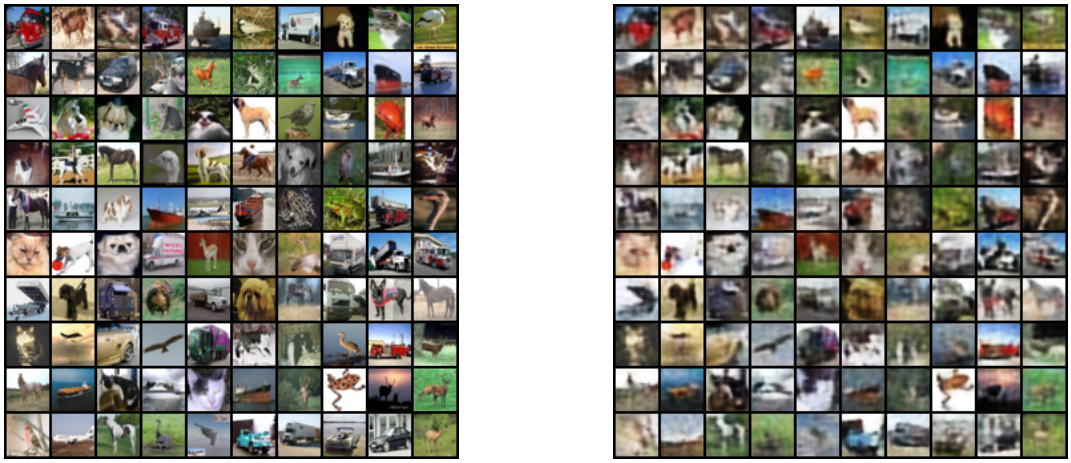

Figure 4.2: Left: CIFAR-10 32x32x3 images, right: reconstructions from our model with a 4x4x100 latent space

In this experiment, we try various datasets and show that we can model x = $32 \times 32 \times 3$ (SVHN and CIFAR-10) or $64 \times 64 \times 3$ (CelebA) images by compressing them to a z = $4 \times 4 \times 100$ feature map latent space. Reconstructions from the latent space are shown below. As we can see in the figure below, compared to the real natural images, the reconstructions look only slightly blurrier than the originals.

Figure 4.3: Left: CelebA 64x64x3 images, right: reconstructions from our model with a 4x4x100 latent space

For implementation details, in the training process, we run 200 epochs. In the short run MCMC, we use 60 Langevin steps for the prior $\tilde{p}_\alpha(z)$ and 30 steps for the posterior $\tilde{p}_\theta(z|x)$. For the learning rate in SGD, we use 1e-5 to update parameters in the latent EBM model while use 1e-4 to update parameters in the generative model.

4.2 Image generation

In this section, we will show some generated results. As we discussed in the previous chapters, an excellent generative model should have the ability to reconstruct the same image from the training set and generate reasonable new images that are different from training data. Only in this way can we conclude that our model can extract useful information from the data because the model is not just mimicking the raw data while learning its distribution. Since we use an EBM prior model in the latent space, we generate images by first randomly draw samples from the latent EBM. And in this process, we may need a short-run MCMC, which is the same as we did in the training procedure.

Figure 4.4: Generated samples for SVHN images.

As we can see in the figures shown here, the generator network p_θ in our framework is well-learned to generate realistic samples and share visual similarities as the training data. Sampling from the latent distribution and getting the corresponding x on image space, generative models should have the ability to generate different samples that are not shown in the training data.

Figure 4.5: Generated samples for CIFAR-10 images.

This could prove in some ways that our latent EBM model generalizes good representations

of the images, and if we move in the latent space, the output of the generator can also move along the image space and still get reasonable natural images.

Figure 4.6: Generated samples for CelebA images.

4.3 Analysis

In this part, we will include some results to show our training process. The construction loss stands for the mean-square error (MSE) between the original data example x and the reconstructed sample $\hat{x}$. It reflects the difference between the real data and the synthesized one on the pixel level. As is shown in Figure 4.7, the construction loss decreases as the iteration goes. In the first 25 iterations, the loss drops quickly. Because we use stochastic gradient descent (SGD) to optimize the loss function, the optimization process may not be a very smooth curve. But we can still see that in the following figures, the training goes well and the reconstruction loss keeps decreasing. Finally, after 150 epochs, the loss becomes stable at a relatively low value.

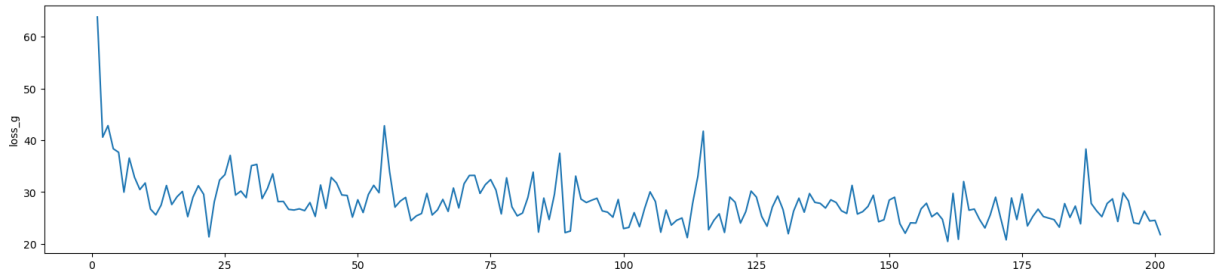

Figure 4.7: Reconstruction loss

In Figure 4.9, we are showing how the positive energy term $E_{\tilde{p}_\theta(z|x)}[f_\alpha(z)]$ and the negative energy term $E_{\tilde{p}_\alpha(z)}[f_\alpha(z)]$ changes along the training procedure. The positive energy is sampled from the approximate posterior, while the negative energy is sampled from the prior EBM model. Both samples are drawn using a short-run MCMC. And Figure 4.8 is the overall energy term.

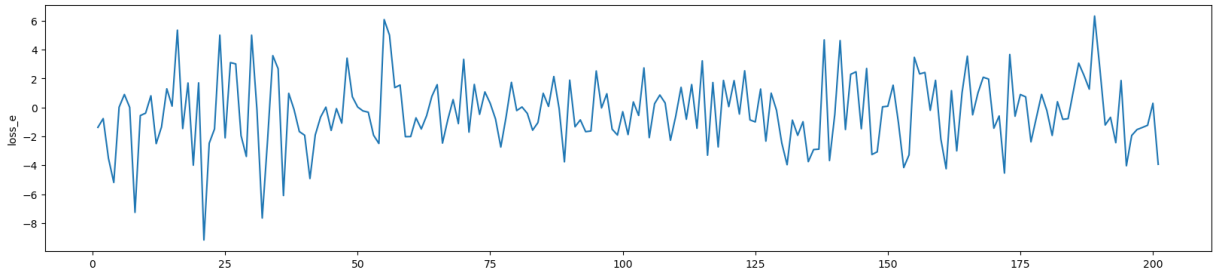

Figure 4.8: Energy term

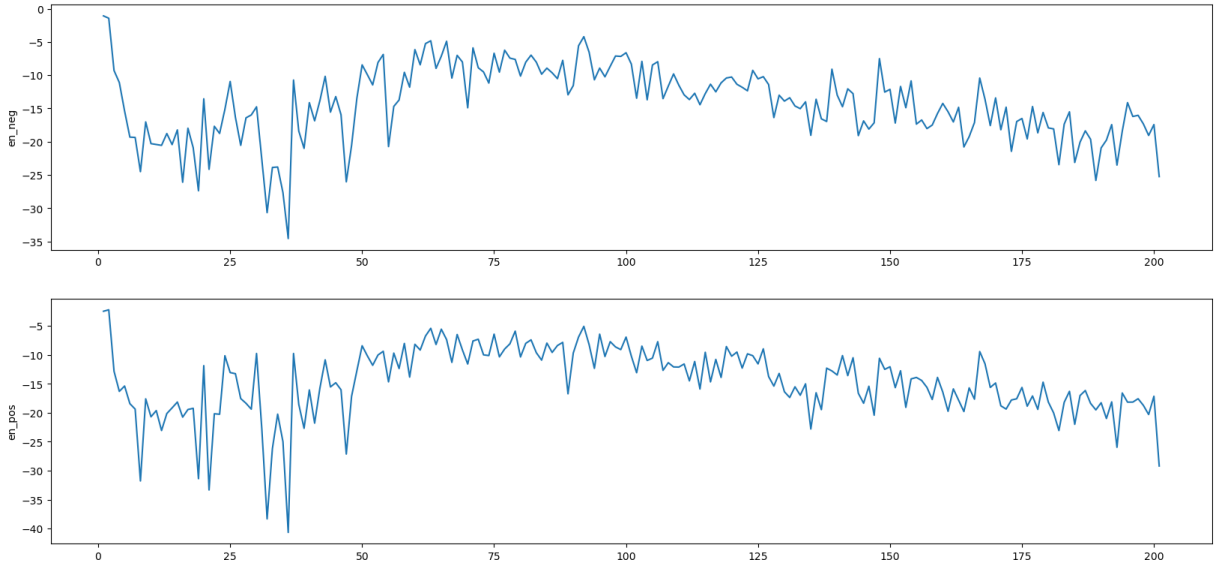

Figure 4.9: Top: negative energy; Bottom: positive energy

CHAPTER 5

Discussion and conclusion

We can regard a top-down model or a directed acyclic graphical model as a simple factorized form that is capable of ancestral sampling. The prototype of such a model is factor analysis, which has been generalized to independent component analysis, sparse coding, etc. An EBM actually defines an unnormalized density or a Gibbs distribution. The prototypes of such a model are exponential family distribution, the Boltzmann machine, and the FRAME (Filters, Random field, And Maximum Entropy) model.

The energy-based model can translate into various forms, such as a loss function or an objective function. It is easy to specify, although optimizing or sampling the energy function requires iterative computation such as MCMC.

Although the top-down model usually assumes independent nodes at the top layer and conditional independent nodes at lower layers, we can modify the top layers by introducing energy terms to correct the simple independence assumptions. Our work is a simple example of this strategy where we correct the prior distribution. Maybe in the later work, we can apply correction to data space.

EBM is a really powerful model that can formulate many complex distributions, but the bottleneck for this work is the MCMC sampling process. However, doing MCMC sampling in the latent space really helps our model to be computational manageable. So building a latent EBM and letting it stands on a top-down neural network can surely get some reasonable representations. And we hope that we can apply EBM to other more applications in future works.

REFERENCES

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation." *arXiv preprint arXiv:1410.8516*, 2014.

[Doe16] Carl Doersch. "Tutorial on variational autoencoders." *arXiv preprint arXiv:1606.05908*, 2016.

[DSB16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp." *arXiv preprint arXiv:1605.08803*, 2016.

[GLZ18] Ruiqi Gao, Yang Lu, Junpei Zhou, Song-Chun Zhu, and Ying Nian Wu. "Learning generative convnets via multi-grid modeling and sampling." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9155–9164, 2018.

[GPM14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks." *arXiv preprint arXiv:1406.2661*, 2014.

[KNH] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research).".

[KW13] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114*, 2013.

[LCH06] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. "A tutorial on energy-based learning." *Predicting structured data*, **1**(0), 2006.

[LLW15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild." In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[Ng11] Andrew Y Ng. "Learning deep energy models." In *ICML*, 2011.

[NHZ19] Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. "Learning non-convergent non-persistent short-run MCMC toward energy-based model." *arXiv preprint arXiv:1904.09770*, 2019.

[NPH20] Erik Nijkamp, Bo Pang, Tian Han, Linqi Zhou, Song-Chun Zhu, and Ying Nian Wu. "Learning multi-layer latent variable model via variational optimization of short run mcmc for approximate inference." In *European Conference on Computer Vision*, pp. 361–378. Springer, 2020.

[NWC11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning." 2011.

[PHN20] Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. "Learning latent space energy-based prior model." *arXiv preprint arXiv:2006.08205*, 2020.

[PNC20] Bo Pang, Erik Nijkamp, Jiali Cui, Tian Han, and Ying Nian Wu. "Semi-supervised learning by latent space energy-based model of symbol-vector coupling." *arXiv preprint arXiv:2010.09359*, 2020.

[RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models." In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

[UML13] Benigno Uria, Iain Murray, and Hugo Larochelle. "RNADE: The real-valued neural autoregressive density-estimator." *arXiv preprint arXiv:1306.0186*, 2013.

[Zhu03] Song-Chun Zhu. "Statistical modeling and conceptualization of visual patterns." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(6):691–712, 2003.

[ZM98] Song Chun Zhu and David Mumford. "Grade: Gibbs reaction and diffusion equations." In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pp. 847–854. IEEE, 1998.

[ZWM98] Song Chun Zhu, Yingnian Wu, and David Mumford. "Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling." *International Journal of Computer Vision*, **27**(2):107–126, 1998.