Entropy in Unsupervised Machine Learning

by

Zengyi Li

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Physics

in the

Graduate Division

of the

University of California, Berkeley

Spring 2022

Entropy in Unsupervised Machine Learning

Abstract

Entropy in Unsupervised Machine Learning

by

Zengyi Li

Doctor of Philosophy in Physics

University of California, Berkeley

Adjunct Professor Friedrich T. Sommer, Co-chair

Associate Professor Michael R. DeWeese, Co-chair

Entropy is a central concept in physics and has deep connections with Information theory, which is one of the foundations of modern machine learning. Specifically, energy-based models are unsupervised machine learning models that adopt a simple yet general formulation based on the principle of maximum entropy. Three Chapters in my thesis are related to energy-based models, and one Chapter uses a Gaussian Coding rate function, which is also related to entropy.

The Boltzmann machine is an energy-based model with strong connections to spin systems in Physics. Boltzmann machines were conceived with bipolar real-valued spin states (up and down) and later generalized to complex valued spin states with unit length. Building on the previous work on complex Boltzmann machines, here we study a generalization where the complex spin states can vary in both, phase and amplitude. Complex Boltzmann machines are closely related to networks of coupled stochastic oscillators and thus can be efficiently implemented in coupled oscillator and also neuromorphic hardwares.

Neural Network Energy-based model (EBM) provide a unified framework for a diverse set of functions, such as sample synthesis, denoising, outlier detection, and Bayesian reasoning. However, the downside of EBMs is that their standard training method based on maximum-likelihood requires expensive sampling and is therefore extremely slow. Denoising score matching is an attractive alternative. Inspired by [143], we study a new method of training EBM in high-dimensional space using multiple scales

with denoising score matching. The resulting model exhibits strong performance on data generation and inpainting.

Another approach that could make training of EBMs efficient is developing an efficient MCMC (Markov Chain Monte Carlo) sampler. The entropy of the proposal distribution of a sampler is an effective measure of the efficiency of the sampler without reference to any detail of the target distributions, thus has the potential to be applied to neural networks energy functions. We developed a new neural network augmented MCMC sampler that can be trained to exactly maximize its proposal entropy. The resulting sampler can adapt to very difficult target distribution geometry, and is shown to improve the training of an EBM.

Self-supervised learning, another important type of unsupervised learning, learns a useful data representation for downstream tasks (for example classification), instead of learning a generative model that fully recreates the dataset. For this, an objective function based on the Gaussian coding rate function called MCR2[178] shows promise. Using this objective, we build a framework that unifies neural networks based non-linear subspace clustering and data-augmentation based self-supervised learning. The resulting algorithm shows strong performance in various subspace clustering tasks.

To everyone in and outside this world.

Contents

List of Figures

List of Tables

Acknowledgments

There are a lot of people without whom I would not have achieved what I have. First I would like to thank my advisor Fritz Sommer, who kindly admitted me, then mostly an experimentalist, to his theory focused lab. His broad interest and first principled thinking provided valuable guidance to my research. He always supported and encouraged me to explore the questions that I find the most interesting and important. This almost laissez-faire approach made my research experience very enjoyable. I'm truly grateful to have such an excellent advisor!

Secondly I would like to thank Yubei Chen, a fellow Redwoodian with whom I had quite a few fruitful collaborations. His deep knowledge in machine learning and solid mathematical foundation proved essential to a lot of our research works. Further, he helped me greatly when I was still a novice in academic writing. I feel lucky to have such an amazing colleague!

I'm also lucky to have spent most of my PhD in Redwood Center for Theoretical Neuroscience, a vibrant research community with friendly students and very supportive faculties. The open-mindedness and the intense and thorough question asking process during every presentation is the most memorable.

I also want to thank all members of my qualifying exam committee and dissertation committee for your support along the way. Especially Mike DeWeese, who welcomes me to his interesting and inspiring lab meetings.

My fascination of entropy stemmed from important mentors in my life even before grad school. My father introduced me to the world of science by answering my endless questions about how everything works when I was little. He once told me that the derivation of Boltzmann distribution was the most elegant theory he could remember from his training as a Chemical Engineer. My undergrad advisor, David Kleinfield, also told me that the most important class in the physics curriculum is Statistical Physics, which I later found out to be an invaluable advice.

Lastly, I want to thank all my family, friends and roommates, especially my wife Sixuan Chen, for being my safe harbor of warmth and safety, especially during this very special time. You are the true reason I can strive!

Chapter 1

Introduction

1.1 Organization of this Thesis

The introduction Chapter provides relevant backgrounds for understanding later chapters, which will be organized by topic. Machine Learning is a field moving in incredibly rapid pace, a lot of interesting development can happen after work in this thesis is done. To provide reader a better context, at the end of some chapters, Epilogue sections are included to provide up to date review of the impact of the work in that chapter, as well as recent developments related to topics in that Chapter.

1.2 Unsupervised Learning and Energy-based models

Modern machine learning is achieving stunning success across more and more task domains that previously were thought to be only solvable by humans. Look no further than famous examples like GPT-3 language models [14] and DALL-E image models[127]. The central driving force that made this possible is the combination of using larger datasets and spending more compute to train the model. To get a sense of the scale, GPT-3 uses 570G of text data and DALL-E used 250 million images. Labeling such datasets by hand is completely infeasible. Thus, unsupervised learning, a learning technique that consumes unlabeled data directly, has been essential to learn from datasets of this scale.

Unsupervised learning uses many different objectives, in this thesis we focus on one of the most basic, where the objective is to recreate a dataset as faithfully as possible.

Regardless of the modality, a dataset can be regarded as a set of points x sampled from a data distribution $p(x)$. The goal of learning would then be to approximate the true data distribution $p(x)$ by a model distribution $q_\theta(x)$ where θ are parameters of the model. Naturally, the learning target consist of a measure of closeness between the two distributions $\mathbb{D}[q_\theta(x)|p(x)]$, whose specific form we will explore later. This class of models are often referred to as *generative models* [157]

Examples of generative models include GANs[49], autoregressive models[117], flow-based models[31][32], variational auto-encoders (VAE)[83], etc. The focus of this thesis is Energy-based models (EBM) [5][94][173], in which the model parameters θ assign an energy value to a sample x (or combination of samples and internal state) $E_\theta(x)$. The model distribution is naturally defined by the Boltzmann distribution with the maximum entropy principle:

$$q_\theta(x) = e^{-E_\theta(x)}/Z_\theta \tag{1.1}$$

Where $Z = \int dx e^{-E_\theta(x)}$ is the partition function. To train an EBM, the most basic method is to minimize the Kullback–Leibler divergence between the model and data distribution:

$$\mathbb{D}_{KL}[p(x)||q_\theta(x)] =< \mathsf{log}[p(x)] - \mathsf{log}[q_\theta(x)] >_{p(x)} \tag{1.2}$$

Where the bracket $<>_{p(x)}$ denotes the average over $p(x)$. The first term in the parenthesis does not depend on the model parameters, thus can be ignored when optimizing the model. The objective thus reduces to simply maximizing the average log likelihood of the data points in the model: $< \mathsf{log}[q_\theta(x)] >_{p(x)}$. *Maximum Likelihood* learning is among the most popular method in statistics to fit models to data.

In the case of EBMs, we use Equation 1.1 to obtain expression for the log likelihood of EBM:

$$\mathsf{log}[q_\theta(x)] = -E_\theta(x) - \mathsf{log}(Z_\theta) \tag{1.3}$$

To optimize θ, one needs to take the derivative of the quantity above with respect to θ. The $E_\theta(x)$ part is usually straight forward, however, for the partition function, we have: $\nabla_\theta \mathsf{log}(Z_\theta) = \frac{1}{Z_\theta}\int dx[-\nabla_\theta E_\theta(x)]e^{-E_\theta(x)} =< -\nabla_\theta E_\theta(x) >_{q_\theta(x)}$. Putting both terms together and including the average over data distribution, we have the following learning rule:

$$\nabla_\theta < \mathsf{log}[q_\theta(x)] >_{p(x)} =< \nabla_\theta E_\theta(x) >_{p(x)} - < \nabla_\theta E_\theta(x) >_{q_\theta(x)} \tag{1.4}$$

As one of the expectations is taken over the model distribution, samples from the EBM are required to train the EBM using maximum likelihood. Unfortunately, this can be computationally very expensive, since obtaining unbiased samples from a complicated distribution defined by an energy function generally requires Markov Chain Monte Carlo (MCMC) methods, which is known to be very slow [115]. Much of this thesis revolves around ideas to remedy this problem. In particular, Chapter 3 describes a new method to train EBM without sampling, and Chapter 4 proposes a neural network MCMC sampler for improving the sampling efficiency.

1.3 Markov Chain Monte Carlo (MCMC) Sampling and its efficiency

In broad strokes, the MCMC sampling process works as follows. First, sample points are initalized according to some initial distribution $p_i(x)$. Second the sample points are modified or "evolved" according to some rule (A Markov Chain) that leaves the target distribution invariant. Therefore, gradually, the samples relax to the target distribution $p(x)$. A Markov Chain is defined by the transition probability $\mathcal{T}(x'|x)$ that describes how the samples are evolved between sampling steps. Often the direct construction of the transition probability is difficult. In such cases the most common way is to evolve with Metropolis-Hastings (MH) updates. Each step update the sample with an arbitrary proposal distribution $q(x'|x)$ combined with a decision process of whether the modification is accepted or rejected. The accept probability $A(x', x) = \mathsf{min}[1, \frac{p(x')q(x|x')}{p(x)q(x'|x)}]$ of the decision process is designed so that the target distribution is preserved.

The efficiency of a MCMC technique is quantified by how many steps it takes to evolve from one sample to another sample which is uncorrelated. Two applications we will investigate, Boltzmann machine and deep EBMs, both use MCMC sampling. For Boltzmann machines, the invariant transition probability $\mathcal{T}(x'|x)$ can easily be derived, and sampling is relatively efficient. Deep EBMs typically use Langevine dynamics, and the corresponding sampling efficiency is very low.

In a M-H style sampler, the core difficulty is the design of efficient proposal distributions. In each step the proposed sample should move far from the previous one (being uncorrelated) and at the same time concentrate on high probability regions of the target distribution to keep the rejection rate low. For example, Langevine dynamics is inefficient because it makes very local moves. In Chapter 4, we propose a new model that learns a proposal distribution to significantly improve sampling efficiency.

1.4 Boltzmann machines

Boltzmann Machines and Restricted Boltzmann Machine

Boltzmann Machines (BM) are a particular type of EBM with strong connections to Ising models in physics. Ising models are physical systems of interacting particles whose local states are described by a binary spin, i.e., up or down. Thus the system state is represented by a binary vector $\mathbf{x} \in [-1, 1]^d$, and energy function $E(x) = -1/2\mathbf{x}^T\mathbf{W}\mathbf{x} - \mathbf{x}^T\mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{d\times d}$ and $\mathbf{b} \in \mathbb{R}^d$ are interactions weights and bias. While the interaction weights of Ising models are given in physics, the weights of BM are learned from data. Like Ising models, BMs have a Boltzmann distribution over $\mathbf{x}$ given by Equation 1.1. In machine learning, this fully visible BM is not a very expressive model. Since for $2^d - 1$ degrees of freedom in the general distribution over $\mathbf{x}$, the system only has $d^2 + d$ parameters. Further, sampling in fully visible BM is very slow, as sampling is a recurrent process in which spins have to be updated individually until convergence. Both the capacity and sampling inefficiency issue can be remedied by a simple alternative, called Restricted Boltzmann Machine (RBM) [138]. In RBMs, units are divided into visible $\mathbf{v}$ and hidden units $\mathbf{h}$, only visible units are fit to data, and there's interaction weights between hidden and visible units, but not amount themselves. The resulting energy function is:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{v}^T\mathbf{b}_v - \mathbf{h}^T\mathbf{b}_h \tag{1.5}$$

We then have the probability for a certain configuration $p_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_\theta}e^{-E_\theta(\mathbf{v},\mathbf{h})}$, where $\theta = [\mathbf{W}, \mathbf{b}_v, \mathbf{b}_h]$ are collection of all parameters in the RBM.

Sampling and Learning in RBM

One useful property of RBMs is that the visible (hidden) units are independent of each other given the hidden (visible) units. This give rise to a natural sampling algorithm called block Gibbs that alternate between sampling all visible and hidden units in parallel. The update rule for each unit is simply the marginal distribution of that unit given all other units: $p(v|h) = \sigma(\mathbf{W}\mathbf{h} + \mathbf{b}_v)$, $p(h|v) = \sigma(\mathbf{v}^T\mathbf{W} + \mathbf{b}_h)$. Where σ denotes sigmoid function.

The learning rule for RBMs is slightly different from standard EBM due to the presence of hidden units. Consider the probability of some configuration $p_\theta(\mathbf{v}) = \sum_\mathbf{h} p_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_\theta}\sum_\mathbf{h} e^{-E_\theta(\mathbf{v},\mathbf{h})}$.

$$\nabla_\theta \mathsf{log}[p_\theta(\mathbf{v})] = \frac{\sum_{\mathbf{v}',\mathbf{h}'} \nabla_\theta E_\theta(\mathbf{v}',\mathbf{h}')e^{-E_\theta(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v}',\mathbf{h}'} e^{-E_\theta(\mathbf{v}',\mathbf{h}')}} - \frac{\sum_{\mathbf{h}'} \nabla_\theta E_\theta(\mathbf{v},\mathbf{h}')e^{-E_\theta(\mathbf{v},\mathbf{h}')}}{\sum_{\mathbf{v}',\mathbf{h}'} e^{-E_\theta(\mathbf{v},\mathbf{h}')}} \tag{1.6}$$

$$= -\left\langle \nabla_\theta E_\theta(\mathbf{v}',\mathbf{h}')\right\rangle_{p_\theta(\mathbf{v}',\mathbf{h}')} + \left\langle \nabla_\theta E_\theta(\mathbf{v},\mathbf{h}')\right\rangle_{p_\theta(\mathbf{h}'|\mathbf{v})} \tag{1.7}$$

Compared to Eq. 1.4 the average over the data distribution is nested by an average over hidden state given a particular sample from the data distribution, and the average over model distribution includes hidden state as well.

As the average over model distribution is still expensive to compute, some techniques are developed to reduce the sampling burden. In particular, Contrastive Divergence (CD) [66][169] and Persistent Contrastive Divergence (PCD) [155] are proven to be very effective for training RBMs. Instead of randomly initializing $\mathbf{v}$ and $\mathbf{h}$, consider initializing $\mathbf{v}$ at a sample $\mathbf{v}_0$ from the data distribution. Block Gibbs sampling can be used to draw a fair sample $\mathbf{h}_0$ from $p(\mathbf{h}|\mathbf{v}_0)$. This sample can be used as a sample for the second average in 1.6. We can then sample $\mathbf{v}_1$ from $p(\mathbf{v}|\mathbf{h}_1)$ using Gibbs sampling, and then $\mathbf{h}_2$ from $p(\mathbf{h}|\mathbf{v}_1)$, etc. For Contrastive Divergence, one use $\mathbf{v}_n$ and $\mathbf{h}_n$ as samples in the first average in 1.6, the resulting algorithm is refereed to as CD_n.

Contrastive divergence is a heuristic method that can quickly learn meaningful features using RBMs, although the resulting update is not a gradient of any function, and it may not converge to a fix point [179]. Note that when $n \to \infty$, $\mathbf{v}_n$ and $\mathbf{h}_n$ will be fair samples from the model distribution, and the resulting gradient update will be exact.

For PCD, a pool of samples from the model are kept between gradient updates of the model. The samples are updated by a few steps of sampling by the current model before used as samples for gradient update. The reasoning behind this is that the model only changes slightly between gradient updates, and a few steps are sufficient for the samples to "catch up" to the current model. PCD does not have the limitation of initializing the chain from data samples, thus may explore the sample space better than CD.

1.5 Training Neural Network Energy-based Models

The more straight-forward and modern approach to EBM is to use a deep neural network to model $E(x)$ directly [173]. When x is continuous, such as images, continuous-space sampling method is required, most of the time, unadjusted Langevine dynamics [35][115] is used:

$$x_{t+1} = x_t - \frac{\epsilon^2}{2}\nabla_x E(x_t) + \epsilon * \mathcal{N}(0, I_d) \tag{1.8}$$

Where $\mathcal{N}(0, I_d)$ denote normal distribution with 0 mean and diagonal covariance, and ϵ is the discretization step size. Physically, this corresponds to a discrete simulation of the diffusion process. This update can also be considered as a proposal distribution, but the M-H accept-reject step is omitted, as it is difficult to achieve reasonable acceptance rate for practical step sizes. Diffusion process is notoriously slow to explore the state space, but still, some practical EBM has been trained with Langevine dynamic.

Analogous to RBM, samples from CD and PCD can be used in place of the full model distribution in Eq 1.4. Although the use of CD is challenging in this case and requires some care[36]. Notably, even though PCD alleviate the burden of sampling by quite a bit, training neural network EBMs can still be extremely expensive [115]. To understand the challenge, one need to realize that computing one step of Langevine dynamics requires 2 passes through the deep neural network, and PCD in general requires $\sim$ 40 steps of sampling per model gradient update [115], that means the training process is $\sim$ 40 times more expensive than training the same network for classification tasks. Besides high computational cost, Langevine dynamics is also ineffective in exploring the sample space, limiting the quality of the resulting EBM.

Score matching

Score matching [72] is an alternative to the maximum likelihood objective. Instead of fitting an energy-based model by matching the gradient of the log probability between the model and data distribution using the following loss:

$$L_{SM}(\theta) = \frac{1}{2}\mathbb{E}_{p(x)}||\nabla_x \mathsf{log}[p(x)] + \nabla_x E_\theta(x)||^2 \tag{1.9}$$

Importantly, the generally intractable partition function does not appear since it doesn't depend on x. It's not difficult to understand that when the gradient matches exactly, so will the density itself. Practically, this objective function is rarely used directly because one usually don't have access to the gradient of the data distribution. An alternative form of score matching exists, but has its own problems which we won't elaborate here, but see[146].

Denoising Score Matching (DSM) [164] is an interesting alternative to the original score matching objective. In DSM, instead of estimating the original $p(x)$, one changes the learning target to a Parzen density estimator of the original distribution: $p_\sigma(\tilde{x}) = \int q_\sigma(\tilde{x}|x)p(x)dx$. Remarkably, the following equation holds:

$$\mathbb{E}_{p_\sigma(\tilde{x})}||\nabla_{\tilde{x}}\mathsf{log}[p_\sigma(\tilde{x})] - \nabla_{\tilde{x}}E_\theta(\tilde{x})||^2 = \mathbb{E}_{p_\sigma(\tilde{x}|x)p(x)}||\nabla_{\tilde{x}}\mathsf{log}[p_\sigma(\tilde{x}|x)] - \nabla_{\tilde{x}}E_\theta(\tilde{x})||^2 \quad (1.10)$$

When $q_\sigma(\tilde{x}|x)$ is Gaussian, $\nabla_{\tilde{x}}\mathsf{log}[p_\sigma(\tilde{x}|x)] = (x - \tilde{x})/2\sigma^2$, which is a vector pointing from the clean sample to the noisy sample, hence the name denoising.

Denoising score matching is a very convenient objective to train an EBM for a data distribution of a certain noise scale [135][134]. However, to generate high quality samples, multiple noise scales are required [143]. Training EBM with denoising score matching of multiple noise scales is studied in Chapter 3.

1.6 Self-supervised feature learning and manifold clustering

Given enough capacity, generative models introduced above will capture all and every details of the data distribution. While this is useful, for some applications, it is sufficient to learn some discriminate features that capture the "useful' information about the data while discarding other information. Examples of this includes learning features for classifying images [21][160] or for speech recognition [6], neither of which requires full reconstruction of the original data. This type of unsupervised learning is often referred to as self-supervised learning, although the boundary of this term is not very clear sometimes.

Here we focus on learning discriminative features from images. One type popular and successful method [21] [61] utilizes the following general procedure: Images from the learning dataset are processed by two random transformations called data-augmentations. Typically the transformations perturb aspects of the images that do not change the underlying object class as perceived by humans, for example, color, zoom, aspect ratio, position, etc. The data-augmentation used is usually found empirically. The neural network is required to output features that are similar for the two augmented images. At the same time, the neural network should not collapse the features of all images to the same point. After training with this procedure, the feature output of the network can be used to classify images with relatively simple classifier, i.e. linear or k-Nearest-Neighbor (kNN), achieving accuracy close to fully supervised methods, where the neural network is also trained with labels. Although the precise reason for the success of such method is still being studied [57][89][187], it can be conceptually understood by assuming that features invariant to those transformations are closely correlated with class information.

In Chapter 5, we study the relationship between augmentation based self-supervised learning and Manifold Clustering, a seemingly unrelated type of classical machine learning problem. To understand Manifold Clustering, consider its linear version, called Subspace Clustering [163], which aims to identify individual linear subspaces from a dataset consist of mixture of them. Additionally, coordinates on those subspaces should be learned. Manifold Clustering goes one step beyond and aims to achieve this for mixture of non-linear manifolds [3].

Being an unsupervised learning problem, Manifold clustering is ill-posed without some assumptions on the underlying data. The most common assumption is that the manifolds are continuous, and there's sufficient data points to outline the underlying structure. Augmentation-based self-supervised learning has the underlying assumption that important features are invariant to augmentations. In Chapter 5, we demonstrate that Self-supervised learning and Manifold Clustering are closely related by tackling them with a unified algorithm that works strongly in both problems.

Chapter 2

Complex Phase-Amplitude Boltzmann Machines

2.1 Introduction

Boltzmann machines are recurrent stochastic neural networks that can be used for learning data distributions. Originally proposed with binary stochastic neurons [5], a complex-valued Boltzmann machine was first introduced under the name DUBM (Directional Unit Boltzmann Machine) [181]. In this model, the neurons represent complex numbers of modulus 1 with arbitrary phase angles. DUBM can learn relative phase distributions. The practical impact of DUBM has been somewhat limited because complex data representing real-world problems often have not only phase but also amplitude variations. From a neuroscience perspective, DUBMs also have the undesirable property that all neurons are active all the time. Here we propose a complex Boltzmann machine whose neurons can represent complex numbers with arbitrary phase angles and amplitudes of 1 or 0. As we demonstrate in simulation experiments, this model enables unsupervised learning of complex-valued data with variable amplitudes. Further, it permits the introduction of regularization of the network activity, such as a sparsity constraint. We also show the necessity of an amplitude-amplitude coupling term that is potentially useful for other types of complex-valued neural networks [54, 158].

2.2 Model Setup

The DUBM model [181] is an energy based model, $p(\boldsymbol{z}) = e^{-E(\boldsymbol{z})}/Z$, for a data distribution of phasor variables, i.e., a vector of complex-valued components z_j with modulus 1. Z is the partition sum. The energy function of the DUBM is given by:

$$E(\boldsymbol{z}) = -\frac{1}{2}\boldsymbol{z}^{\dagger}\boldsymbol{W}\boldsymbol{z} \tag{2.1}$$

where the superscript † denotes the conjugate transpose. The matrix $\boldsymbol{W} \in \mathbf{C}^N$ is a complex coupling matrix. For (2.1) to be real-valued, the matrix is required to be Hermitian, i.e., $\boldsymbol{W}^{\dagger} = \boldsymbol{W}$.

If we allow a state z_j to take two modulus values, 1 and 0, corresponding to an active or inactive neuron, (2.1) induces an amplitude and relative phase distribution. To control the fraction of active units, we add into (2.1) a penalty term of the form: $\boldsymbol{\epsilon}^T|\boldsymbol{z}|$, where $\boldsymbol{\epsilon} \in \mathbf{R}^N$ is a bias vector. Further, we introduce an amplitude-amplitude coupling term: $-\frac{1}{2}|\boldsymbol{z}|^T\boldsymbol{J}|\boldsymbol{z}|$, with $\boldsymbol{J} \in \mathbf{R}^N$ a symmetric real-valued matrix. Putting it all together, the energy function of Complex Amplitude-Phase Boltzmann Machine (CAP-BM) is:

$$E(\boldsymbol{z}) = -\frac{1}{2}\boldsymbol{z}^{\dagger}\boldsymbol{W}\boldsymbol{z} - \frac{1}{2}|\boldsymbol{z}|^T\boldsymbol{J}|\boldsymbol{z}| + \boldsymbol{\epsilon}^T|\boldsymbol{z}| \tag{2.2}$$

Like in the DUBM model, the CAP-BM model is symmetric with respect to global phase shifts in all units. The benefit of the amplitude-amplitude coupling in the CAP-BM might not be obvious here, but we will explore its effect experimentally and argue later why this term is essential.

Sampling in complex Boltzmann machines can be achieved by a Gibbs sampling procedure similar to that in real-valued Boltzmann machines. One difference is that we sample amplitude and phase separately. To achieve this, two marginal probabilities induced by the Boltzmann distribution are required: $P(|z_j| = 1|\,\boldsymbol{z}_{!j})$ and $p(\theta_j|\,|z_j| = 1, \boldsymbol{z}_{!j})$. They represent the marginal probability for a unit to take amplitude 1 and the probability density of its phase, given that it takes amplitude 1. They can be obtained in the same manner as in [5], for derivations, see Appendix A:

$$\begin{aligned} P(|z_j| = 1|\,\boldsymbol{z}_{!j}) &= \frac{1}{1 + (e^{\mu_j - \epsilon_j}\mathrm{I}_0(a_j))^{-1}} & (2.3) \\ p(\theta_j|\,|z_j| = 1, \boldsymbol{z}_{!j}) &= \frac{1}{2\pi\mathrm{I}_0(a_j)}e^{a_j\cos(\alpha_j - \theta_j)} & (2.4) \end{aligned}$$

In the above equations the variables a_j, α_j, μ_j represent the complex and real-valued input sums to neuron j: $u_j = a_j e^{i\alpha_j} = \sum_{k\neq j} W_{jk} z_k$ and $\mu_j = \sum_{k\neq j} J_{jk}|z_k|$. $\mathrm{I}_0(x)$ denotes the zeroth order Bessel function of the first kind, which becomes similar to an exponential function for large arguments. Therefore, $P(|z_j| = 1|\,\boldsymbol{z}_{!j})$ is sigmoid shaped as a function of a_j and μ_j. Similar as that in the DUBM model, the phase distribution $p(\theta_j|\,|z_j| = 1, \boldsymbol{z}_{!j})$ is a von Mises distribution, the circular analog of Gaussian. For a graphic depicting of the behavior of $P(|z_j| = 1|\,\boldsymbol{z}_{!j})$, see Appendix, Figure A.1.

Note here the amplitude depends on phase through a_j, and phase depends on amplitude as units with amplitude 0 do not contribute to the input sum u_j. Therefore, the CAP-BM model is not equivalent to the combination of DUBM and a real-valued Boltzmann machine, in which amplitudes and phases would be modeled separately.

2.3 Learning rules for the Complex Boltzmann machine

Like for the real-valued Boltzmann machine[5], the learning rules for model parameters of the CAP-BM model can be derived for the Maximum Likelihood objective G (derivations, see Appendix A):

$$\frac{\partial G}{\partial b_{jk}} = \langle |z_j||z_k|\cos(\theta_{jk} + \theta_k - \theta_j)\rangle_{\text{sample}} - \langle |z_j||z_k|\cos(\theta_{jk} + \theta_k - \theta_j)\rangle_{\text{model}} \tag{2.5}$$

$$\frac{\partial G}{\partial \theta_{jk}} = -\langle |z_j||z_k|b_{jk}\sin(\theta_{jk} + \theta_k - \theta_j)\rangle_{\text{sample}} + \langle |z_j||z_k|b_{jk}\sin(\theta_{jk} + \theta_k - \theta_j)\rangle_{\text{model}} \tag{2.6}$$

$$\frac{\partial G}{\partial J_{jk}} = \langle |z_j||z_k|\rangle_{\text{sample}} - \langle |z_j||z_k|\rangle_{\text{model}} \tag{2.7}$$

$$\frac{\partial G}{\partial \epsilon_j} = -\langle |z_j|\rangle_{\text{sample}} + \langle |z_j|\rangle_{\text{model}} \tag{2.8}$$

Here b_{jk} and θ_{jk} denote amplitude and phase of complex weight $W_{jk} = b_{jk}e^{i\theta_{jk}}$, J_{jk} the real-valued weight for amplitude-amplitude coupling, and ϵ_j the bias.

The learning rules (2.7) and (2.8) are the same as that of real-valued BM while rules (2.5) and (2.6) are similar to that of DUBM with extra amplitude dependencies. Another similarity to real-valued BMs is that training in our model requires sampling from the model distribution. To speed up the training in real-valued BMs, learning schemes such as Contrastive Divergence (CD) and Persistent Contrastive Divergence (PCD) [66, 155] have been proposed that do not require full model distribution.

Another proposal for higher sampling efficiency is to choose a network architecture, now called the restricted Boltzmann machine [138], in which sampling from model is more parallelizable because recurrent weights within the sets of hidden or visible units are absent. All these techniques for speeding up the training can equally be applied to the CAP-BM model.

2.4 Experiments with a Complex Phase-Amplitude RBM

Here we demonstrate a restricted version of the CAP-BM, referred to as CAP-RBM, on synthetic data and on the MNIST dataset pre-processed with a complex wavelet transform (CWT).

For synthetic dataset, the CAP-RBM was trained using 1-step contrastive divergence (CD-1) [66] on a synthetic dataset of complex-valued images of bars with a noisy sine-wave phase pattern. We compare the performance of models with and without the amplitude-amplitude coupling term J. As can be seen in Fig. 2.1 a), the model without the J term does not form a stable representation of test data.

The necessity of J term can be explained as follows. In equations 2.4 and 2.3 one can see that the amplitude of the complex input sum to a unit, a_j, plays a dual role of controlling the activation of a unit and the variance of phase distribution. Sometimes the data may have sharp amplitude distribution while having large variance on its phase, this distribution cannot be learned without J since this would require a_j to be large and small at the same time.

We then train CAP-RBM on complex wavelet transformed MNIST dataset, where only middle two frequency bands are used and the complex coefficients are thresholded and normalized. Training used PCD [155] algorithm after initializing with CD-1. As can be seen from Fig. 2.1 b) and c), the model captures data distribution well.

Further experimental details are provided in Appendix A.

2.5 Conclusion

In this Chapter we proposed and demonstrated a model of Boltzmann machine with both amplitude and phase variation. Our work differs from previous formulations of

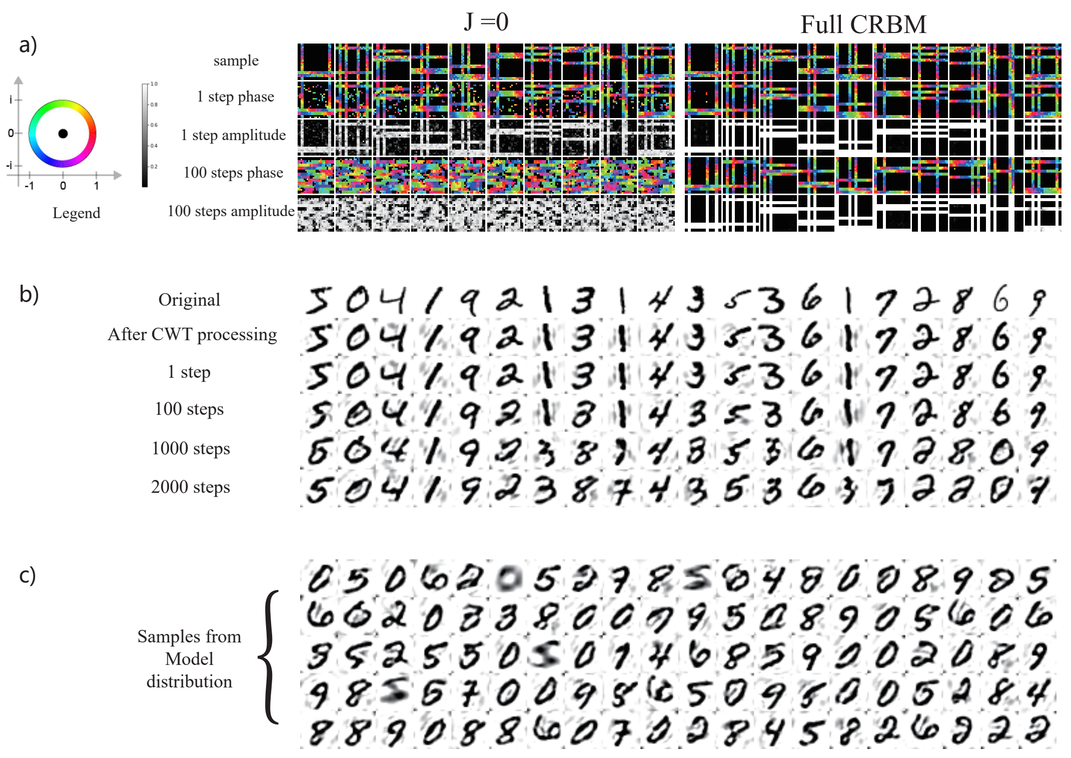


Figure 2.1: Demonstration of the Complex restricted Boltzmann machine (CAP-RBM) on a synthetic bar dataset and on CWT-transformed MNIST dataset. a) Training CAP-RBM on complex images of bars with noisy phase (best viewed in color). The two blocks of images show results of the CAP-RBM without J matrix and the full CAP-RBM. The first row in each block shows samples from dataset. The four lower rows in each block show the expectation of visible unit activity after variable numbers of Gibbs sampling steps initialized at sample. The model without J matrix does not form stable representation of the sample. b) original and reconstructed MNIST digits after various numbers of sampling steps, initialized at samples. c) samples generated from random initialization, global phase for each sample has to be set by hand.

complex BM by being a natural extension of DUBM. In contrast, [125] use variables with binary real and imaginary parts, and [102] use complex-Gaussian visible unit. In particular we showed the importance of an amplitude-amplitude coupling term not seen in previous works on complex-valued neural networks. In addition, this model is potentially directly applicable since new hardware implementation of Boltzmann sampling in complex domain is becoming available. Examples include electronic [167] and optical[150] implementations. Furthermore, there is recent proposal of mapping recurrent network of oscillating spiking neurons to complex networks [43], which could also benefit from a probabilistic interpretation.

Chapter 3

Learning EBMs in high-dimensional spaces with MDSM

3.1 Introduction and Motivation

Treating data as stochastic samples from a probability distribution and developing models that can learn such distributions is at the core for solving a large variety of application problems, such as error correction/denoising [165], outlier/novelty detection [182, 24], sample generation [115, 35], invariant pattern recognition, Bayesian reasoning [170] which relies on good data priors, and many others.

Energy-Based Models (EBMs) [94, 113] assign an energy $E(\boldsymbol{x})$ to each data point $\boldsymbol{x}$ which implicitly defines a probability by the Boltzmann distribution $p_m(\boldsymbol{x}) = e^{-E(\boldsymbol{x})}/Z$. Sampling from this distribution can be used as a generative process that yield plausible samples of $\boldsymbol{x}$.

Compared to other generative models, like GANs [49], flow-based models [31, 84], or auto-regressive models [117, 118], energy-based models have significant advantages. First, they provide explicit (unnormalized) density information, compositionality [65, 55], better mode coverage [90] and flexibility [35]. Further, they do not require special model architecture, unlike auto-regressive and flow-based models. Recently, Energy-based models has been successfully trained with maximum likelihood [115, 35], but training can be very computationally demanding due to the need of sampling model distribution. Variants with a truncated sampling procedure have been proposed, such as contrastive divergence [66]. Such models learn much faster with the draw back of not exploring the state space thoroughly [155].

Score Matching, Denoising Score Matching and Deep Energy Estimators

Score matching (SM) [72] circumvents the requirement of sampling the model distribution. In score matching, the score function is defined to be the gradient of log-density or the negative energy function. The expected $L2$ norm of difference between the model score function and the data score function are minimized.

One convenient way of using score matching is learning the energy function corresponding to a Gaussian kernel Parzen density estimator [121] of the data: $p_{\sigma_0}(\tilde{\boldsymbol{x}}) = \int q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$. Though hard to evaluate, the data score is well defined: $s_d(\tilde{\boldsymbol{x}}) = \nabla_{\tilde{\boldsymbol{x}}}\log(p_{\sigma_0}(\tilde{\boldsymbol{x}}))$, and the corresponding objective $L_{SM}(\theta) =$:

$$\mathbb{E}_{p_{\sigma 0}(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}}\log(p_{\sigma_0}(\tilde{\boldsymbol{x}})) + \nabla_{\tilde{\boldsymbol{x}}}E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.1}$$

L_{SM} is also known as the Fisher divergence or the Fisher information distance [76, 28], $L_{SM} = D_{FD}(p_{\sigma 0}||p_m)$, where $p_m(\boldsymbol{x}) = e^{-E(\boldsymbol{x})}/Z$ is the normalized distribution from the model energy function. While the KL divergence requires the ratio between two density functions, this metric doesn't depend on the normalizing constant Z, which for energy-based model needs global integration through sampling and is rarely accurately available.

[164] studied the connection between denoising auto-encoder and score matching, and proved the remarkable result that the following objective, named *Denoising Score Matching* (DSM), is equivalent to the objective above, and $L_{DSM}(\theta) =$:

$$\mathbb{E}_{p_{\sigma_0}(\tilde{\boldsymbol{x}},\boldsymbol{x})} \parallel \nabla_{\tilde{\boldsymbol{x}}}\log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}}E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.2}$$

Note that in (3.2) the Parzen density score is replaced by the derivative of log density of the single noise kernel $\nabla_{\tilde{\boldsymbol{x}}}\log(q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}))$, which is much easier to evaluate. In the particular case of Gaussian noise, $\log(q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) = -\frac{(\tilde{\boldsymbol{x}}-\boldsymbol{x})^2}{2\sigma_0^2} + C$, and therefore $L_{DSM}(\theta) =$:

$$\mathbb{E}_{p_{\sigma 0}(\tilde{\boldsymbol{x}},\boldsymbol{x})} \parallel \boldsymbol{x} - \tilde{\boldsymbol{x}} + {\sigma_0}^2\nabla_{\tilde{\boldsymbol{x}}}E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.3}$$

The intuition behind the objective (3.3) is simple, it forces the energy gradient to align with the vector pointing from the noisy sample to the clean data sample.

To optimize an objective involving the derivative of a function defined by a neural network, [82] proposed the use of double backpropagation [34]. *Deep energy estimator networks* [135] first applied this technique to learn an energy function defined by a deep neural network. In this work and similarly in [134], an energy-based model was trained to match a Parzen density estimator of data with a certain noise magnitude. The previous models were able to perform denoising task, but they were unable to generate high-quality data samples from a random input initialization. Recently, [143] trained an excellent generative model by fitting a series of score estimators coupled together in a single neural network, each matching the score of a Parzen estimator with a different noise magnitude.

The questions we address here is why learning energy-based models with single noise level does not permit high-quality sample generation and what can be done to improve such energy based models. Our work builds on key ideas from [135, 134, 143].

Section 3.2 provides a geometric view of the learning problem in denoising score matching and provides a theoretical explanation why training with one noise level is insufficient if the data dimension is high.

Section 3.3 presents a novel method for training energy based model, *Multiscale Denoising Score Matching* (MDSM). Section 3.4 describes empirical results of the MDSM model and comparisons with other models.

3.2 A Geometric View of Denoising Score Matching

[143] used denoising score matching with a range of noise levels, achieving great empirical results. The authors explained that large noise perturbation are required to enable the learning of the score in low-data density regions. But it is still unclear why a series of different noise levels are necessary, rather than one single noise level that is large enough. Following [134], we analyze the learning process in denoising score matching based on measure concentration properties of high-dimensional random vectors.

We adopt the common assumption that the data distribution to be learned is high-dimensional, but only has support around a relatively low-dimensional manifold [153, 128, 93]. If the assumption holds, it causes a problem for score matching: The density, or the gradient of the density is then undefined outside the manifold, making it

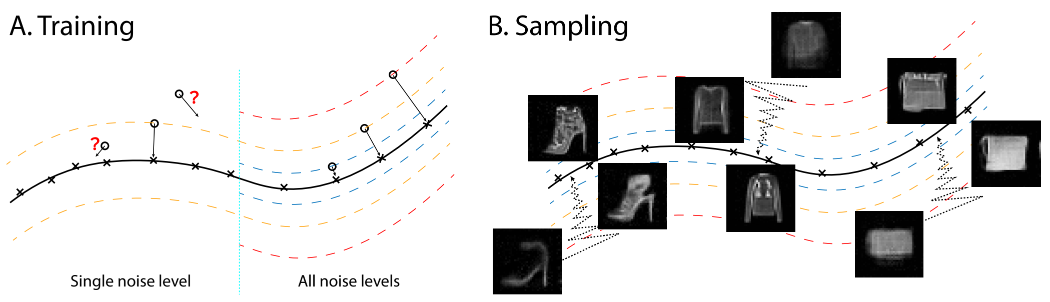


Figure 3.1: Illustration of multiscale denoising score matching. A. During training, derivative of log-likelihood is forced to point toward data manifold, establishing energy difference between points within manifold and points outside. Note that energy is negative log-likelihood therefore energy is higher for point further away from data manifold. B. During annealed Langevin sampling, sample travel from outside data manifold to data manifold. Shown are singled step denoised sample during sampling of an energy function trained with MDSM on Fashion-MNIST (see text for details).

difficult to train a valid density model for the data distribution defined on the entire space. [134] and [143] discussed this problem and proposed to smooth the data distribution with a Gaussian kernel to alleviate the issue.

To further understand the learning in denoising score matching when the data lie on a manifold $\mathcal{X}$ and the data dimension is high, two elementary properties of random Gaussian vectors in high-dimensional spaces are helpful: First, the length distribution of random vectors becomes concentrated at $\sqrt{d}\sigma$ [162], where σ^2 is the variance of a single dimension. Second, a random vector is always close to orthogonal to a fixed vector [151]. With these premises one can visualize the configuration of noisy and noiseless data points that enter the learning process: A data point $\boldsymbol{x}$ sampled from $\mathcal{X}$ and its noisy version $\tilde{\boldsymbol{x}}$ always lie on a line which is almost perpendicular to the tangent space $T_{\boldsymbol{x}}\mathcal{X}$ and intersects $\mathcal{X}$ at $\boldsymbol{x}$. Further, the distance vectors between $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ pairs all have similar length $\sqrt{d}\sigma$. As a consequence, the set of noisy data points concentrate on a set $\tilde{\mathcal{X}}_{\sqrt{d}\sigma,\epsilon}$ that has a distance with $(\sqrt{d}\sigma - \epsilon, \sqrt{d}\sigma + \epsilon)$ from the data manifold $\mathcal{X}$, where $\epsilon \ll \sqrt{d}\sigma$.

Therefore, performing denoising score matching learning with $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ pairs generated with a fixed noise level σ, which is the approach taken previously except in [143], will match the score in the set $\tilde{\mathcal{X}}_{\sqrt{d}\sigma,\epsilon}$ and enable denoising of noisy points in the same set. However, the learning provides little information about the density outside this set, farther or closer to the data manifold, as noisy samples outside $\tilde{\mathcal{X}}_{\sqrt{d}\sigma,\epsilon}$ rarely appear in the training process. An illustration is presented in Figure 3.1A.

Let $\tilde{\mathcal{X}}^C_{\sqrt{d}\sigma,\epsilon}$ denote the complement of the set $\tilde{\mathcal{X}}_{\sqrt{d}\sigma,\epsilon}$. Even if $p_{\sigma_0}(\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^C_{\sqrt{d}\sigma,\epsilon})$ is very small in high-dimensional space, the score in $\tilde{\mathcal{X}}^C_{\sqrt{d}\sigma,\epsilon}$ still plays a critical role in sampling from random initialization. This analysis may explain why models based on denoising score matching, trained with a single noise level encounter difficulties in generating data samples when initialized at random. For an empirical support of this explanation, see our experiments with models trained with single noise magnitudes (Appendix B.2). To remedy this problem, one has to apply a learning procedure of the sort proposed in [143], in which samples with different noise levels are used. Depending on the dimension of the data, the different noise levels have to be spaced narrowly enough to avoid empty regions in the data space. In the following, we will use Gaussian noise and employ a Gaussian scale mixture to produce the noisy data samples for the training (for details, See Section 3.3 and Appendix B.1).

Another interesting property of denoising score matching was suggested in the denoising auto-encoder literature [165, 80]. With increasing noise level, the learned features tend to have larger spatial scale. In our experiment we observe similar phenomenon when training model with denoising score matching with single noise scale. If one compare samples in Figure B.1, Appendix B.2, it is evident that noise level of 0.3 produced a model that learned short range correlation that spans only a few pixels, noise level of 0.6 learns longer stroke structure without coherent overall structure, and noise level of 1 learns more coherent long range structure without details such as stroke width variations. This suggests that training with single noise level in denoising score matching is not sufficient for learning a model capable of high-quality sample synthesis, as such a model have to capture data structure of all scales.

3.3 Learning Energy-Based Model with Multi-Scale Denoising Score Matching

Multiscale Denoising Score Matching

Motivated by the analysis in section 3.2, we strive to develop an EBM based on denoising score matching that can be trained with noisy samples in which the noise level is not fixed but drawn from a distribution. The model should approximate the Parzen density estimator of the data $p_{\sigma_0}(\tilde{\boldsymbol{x}}) = \int q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})dx$. Specifically, the learning should minimize the difference between the derivative of the energy and the score of p_{σ_0} under the expectation $\mathbb{E}_{p_M(\tilde{\boldsymbol{x}})}$ rather than $\mathbb{E}_{p_{\sigma_0}(\tilde{\boldsymbol{x}})}$, the expectation taken in standard denoising score matching. Here $p_M(\tilde{\boldsymbol{x}}) = \int q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})dx$ is chosen to cover the signal space more evenly to avoid the measure concentration issue described above. The resulting *Multiscale Score Matching* (MSM) objective $L_{MSM}(\theta) =:$

$$\mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.4}$$

Compared to the objective of denoising score matching (3.1), the only change in the new objective (3.4) is the expectation. Both objectives are consistent, if $p_M(\tilde{\boldsymbol{x}})$ and $p_{\sigma_0}(\tilde{\boldsymbol{x}})$ have the same support, as shown formally in Proposition 1 of Appendix B.1. In Proposition 2, we prove that Equation 3.4 is equivalent to the following denoising score matching objective $L_{MDSM^*} =:$

$$\mathbb{E}_{p_M(\tilde{\boldsymbol{x}})q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.5}$$

The above results hold for any noise kernel $q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, but Equation 3.5 contains the reversed expectation, which is difficult to evaluate in general. To proceed, we choose $q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ to be Gaussian, and also choose $q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ to be a Gaussian scale mixture: $q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \int q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\sigma)d\sigma$ and $q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}, \sigma^2 I_d)$. After algebraic manipulation and one approximation (see the derivation following Proposition 2 in Appendix B.1), we can transform Equation 3.5 into a more convenient form, which we call *Multiscale Denoising Score Matching* (MDSM), $L_{MDSM} =:$

$$\mathbb{E}_{p(\sigma)q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \tag{3.6}$$

The square loss term evaluated at noisy points $\tilde{\boldsymbol{x}}$ at larger distances from the true data points $\boldsymbol{x}$ will have much larger magnitude. Therefore, in practice it is necessary to add a monotonically decreasing term $l(\sigma)$ for balancing the loss in different noise scales, e.g. $l(\sigma) = \frac{1}{\sigma^2}$. Ideally, we want our model to learn the correct gradient everywhere, so we would need to add noise of all levels. However, learning denoising score matching at very large or very small noise levels is useless. At very large noise levels the information of the original sample is completely lost. Conversely, in the limit of small noise, the noisy sample is virtually indistinguishable from real data. In neither case one can learn a gradient which is informative about the data structure. Thus, the noise range needs only to be broad enough to encourage learning of data features over all scales. Particularly, we do not sample σ but instead choose a series of fixed σ values $\sigma_1 \cdots \sigma_K$. Further, substituting $\log(q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) = -\frac{(\tilde{\boldsymbol{x}}-\boldsymbol{x})^2}{2\sigma_0^2} + C$ into

Equation 3.4, we arrive at the final objective $L(\theta) =$:

$$\sum_{\sigma \in \{\sigma_1 \cdots \sigma_K\}} \mathbb{E}_{q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})} l(\sigma) \parallel \boldsymbol{x} - \tilde{\boldsymbol{x}} + \sigma_0^2 \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}}; \theta) \parallel^2 \tag{3.7}$$

It may seem that σ_0 is an important hyperparameter to our model, but after our approximation σ_0 become just a scaling factor in front of the energy function, and can be simply set to one as long as the temperature range during sampling is scaled accordingly (See Section 3.3). Therefore the only hyper-parameter is the rang of noise levels used during training.

On the surface, objective (3.7) looks similar to the one in [143]. The important difference is that Equation 3.7 approximates a *single* distribution, namely $p_{\sigma_0}(\tilde{\boldsymbol{x}})$, the data smoothed with one fixed kernel $q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. In contrast, [143] approximate the score of *multiple* distributions, the family of distributions $\{p_{\sigma_i}(\tilde{\boldsymbol{x}}) : i = 1, ..., n\}$, resulting from the data smoothed by kernels of different widths σ_i. Because our model learns only a single target distribution, it does not require noise magnitude as input.

Sampling by Annealed Langevin Dynamics

Langevin dynamics has been used to sample from neural network energy functions [35, 115]. However, those studies described difficulties with mode exploration unless very large number of sampling steps is used. To improve mode exploration, we propose incorporating simulated annealing in the Langevin dynamics. Simulated annealing [85, 104] improves mode exploration by sampling first at high temperature and then cooling down gradually. This has been successfully applied to challenging computational problems, such as combinatorial optimization.

To apply simulated annealing to Langevin dynamics. Note that in a model of Brownian motion of a physical particle, the temperature in the Langevin equation enters as a factor $\sqrt{T}$ in front of the noise term, some literature uses $\sqrt{\beta^{-1}}$ where $\beta = 1/T$ [78]. Adopting the $\sqrt{T}$ convention, the Langevin sampling process [11] is given by:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\epsilon^2}{2} \nabla_{\boldsymbol{x}} E(\boldsymbol{x}_t; \theta) + \epsilon \sqrt{T_t} \mathcal{N}(0, I_d) \tag{3.8}$$

where T_t follows some annealing schedule, and ϵ denotes step length, which is fixed. During sampling, samples behave very much like physical particles under Brownian motion in a potential field. Because the particles have average energies close to the their current thermic energy, they explore the state space at different distances from data manifold depending on temperature. Eventually, they settle somewhere on the data manifold. The behavior of the particle's energy value during a typical annealing process is depicted in Appendix Figure B.7B.

If the obtained sample is still slightly noisy, we can apply a single step gradient denoising jump [134] to improve sample quality:

$$\boldsymbol{x}_{clean} = \boldsymbol{x}_{noisy} - \sigma_0^2 \nabla_{\boldsymbol{x}} E(\boldsymbol{x}_{noisy}; \theta) \tag{3.9}$$

This denoising procedure can be applied to noisy sample with any level of Gaussian noise because in our model the gradient automatically has the right magnitude to denoise the sample. This process is justified by the Empirical Bayes interpretation of this denoising process, as studied in [134].

[143] also call their sample generation process annealed Langevin dynamics. It should be noted that their sampling process does not coincide with Equation 3.8. Their sampling procedure is best understood as sequentially sampling a series of distributions corresponding to data distribution corrupted by different levels of noise.

3.4 Image Modeling Results

Training and Sampling Details. The proposed energy-based model is trained on standard image datasets, specifically MNIST, Fashion MNIST, CelebA [97] and CIFAR-10 [87]. During training we set $\sigma_0 = 0.1$ and train over a noise range of $\sigma \in [0.05, 1.2]$, with the different noise uniformly spaced on the batch dimension. For MNIST and Fashion MNIST we used geometrically distributed noise in the range $[0.1, 3]$. The weighting factor $l(\sigma)$ is always set to $1/\sigma^2$ to make the square term roughly independent of σ. We use batch size of 128 and use the Adam optimizer with a learning rate of 5×10^{-5}. For MNIST and Fashion MNIST, we use a 12-Layer ResNet with 64 filters, for the CelebA and CIFAR-10 datasets we used a 18-Layer ResNet with 128 filters [59, 60]. No normalization layer was used in any of the networks. We designed the output layer of all networks to take a generalized quadratic form [41]. Because the energy function is anticipated to be approximately quadratic with respect to the noise level, this modification was able to boost the performance significantly. For more detail on training and model architecture, see Appendix B.4. One notable result is that since our training method does not involve sampling, we achieved a speed up of roughly an order of magnitude compared to the maximum-likelihood training using Langevin dynamics [1]. Our method thus enables the training of energy-based models even when limited computational resources prohibit maximum likelihood methods.

We found that the choice of the maximum noise level has little effect on learning as long as it is large enough to encourage learning of the longest range features in the data. However, as expected, learning with too small or too large noise levels is not beneficial and can even destabilize the training process. Further, our method appeared to be relatively insensitive to how the noise levels are distributed over a chosen range. Geometrically spaced noise as in [143] and linearly spaced noise both work, although in our case learning with linearly spaced noise was somewhat more robust.

For sampling the learned energy function we used annealed Langevin dynamics with an annealing schedule where the temperature varies continuously, see Figure B.7B for the particular shape of annealing schedule we used. In contrast, annealing schedules with theoretical guaranteed convergence property takes extremely long [46]. The range of temperatures to use in the sampling process depends on the choice of σ_0, as the equilibrium distribution is roughly images with Gaussian noise of magnitude $\sqrt{T}\sigma_0$ added on top. To ease traveling between modes far apart and ensure even sampling, the initial temperature needs to be high enough to inject noise of sufficient magnitude. A choice of $T = 100$, which corresponds to added noise of magnitude $\sqrt{100} * 0.1 = 1$, seems to be sufficient starting point. For step length ϵ we generally used 0.02, and $[0.015, 0.05]$ seemed to be a reasonable range for this parameter. After the annealing process we performed a single step denoising to slightly enhance sample quality.

Unconditional Image Generation. We demonstrate the generative ability of our model by displaying samples obtained by annealed Langevin sampling and single step denoising jump. We evaluated 50k sampled images after training on CIFAR-10 with two performance scores, Inception [133] and FID [62]. We achieved Inception Score of 8.31 and FID of 31.7, comparable to modern GAN approaches. Scores for CelebA dataset are not reported here as they are not commonly reported and may depend on the specific pre-processing used. More samples and training images are provided in Appendix for visual inspection. We believe that visual assessment is still essential because of the possible issues with the Inception score [8]. Indeed, we also

[1]For example, on a single GPU, training MNIST with a 12-layer Resnet takes 0.3s per batch with our method, while maximum likelihood training with a modest 30 Langevin steps per weight update takes 3s per batch. Both methods need similar number of weight updates to train.

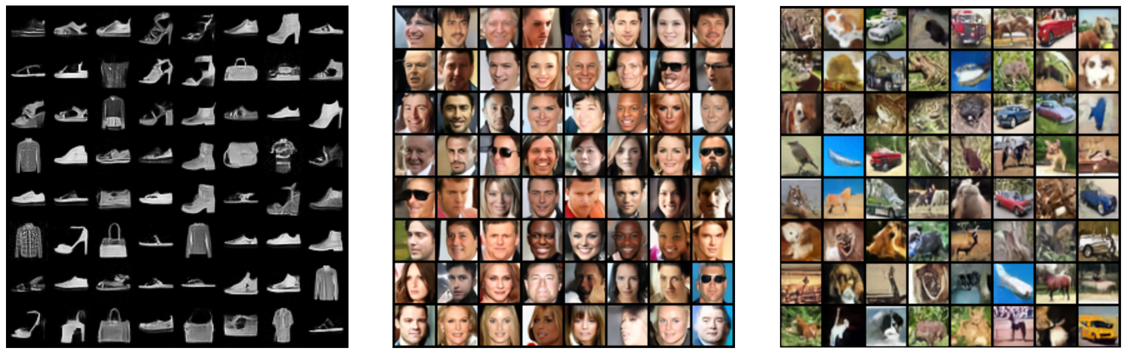


Figure 3.2: Unconditional samples from our model trained on Fashion MNIST, CelebA and CIFAR-10. See Figure B.5 and Figure B.6 in Appendix for more samples and comparison with training data.

found that the visually impressive samples were not necessarily the one achieving the highest Inception Score.

Although overfitting is not a common concern for generative models, we still tested our model for overfitting. We found no indication for overfitting by comparing model samples with their nearest neighbors in the dataset, see Figure B.2 in Appendix.

Table 3.1: Unconditional inception score, FID scores and likelihoods for CIFAR-10

Model	**IS ↑**	**FID ↓**	**Likelihood**	**NLL (bits/dim) ↓**
iResNet [9]	-	65.01	Yes	3.45
PixelCNN [117]	4.60	65.93	Yes	**3.14**
PixelIQN [118]	5.29	49.46	Yes	-
Residual Flow [20]	-	46.37	Yes	3.28
GLOW [84]	-	46.90	Yes	3.35
EBM (ensemble) [35]	6.78	38.2	Yes	- [2]
MDSM (Ours)	8.31	31.7	Yes	7.04 [3]
SNGAN [101]	8.22	**21.7**	No	-
NCSN [143]	**8.91**	25.32	No	-

Mode Coverage. We repeated with our model the 3 channel MNIST mode coverage experiment similar to the one in [90]. An energy-based model was trained on 3-channel data where each channel is a random MNIST digit. Then 8000 samples were taken from the model and each channel was classified using a small MNIST classifier network. We obtained results of the 966 modes, comparable to GAN approaches. Training was successful and our model assigned low energy to all the learned modes, but some modes were not accessed during sampling, likely due to the Langevin Dynamics failing to explore these modes. A better sampling technique such as HMC [105] or a Maximum Entropy Generator [90] could improve this result.

Image Inpainting. Image impainting can be achieved with our model by clamping a part of the image to ground truth and performing the same annealed Langevin and Jump sampling procedure on the missing part of the image. Noise appropriate to the sampling temperature need to be added to the clamped inputs. The quality of inpainting results of our model trained on CelebA and CIFAR-10 can be assessed in Figure 3.3. For CIFAR-10 inpainting results we used the test set.

Log Likelihood Estimation. For energy-based models, the log density can be obtained after estimating the partition function with Annealed Importance Sampling (AIS) [131] or Reverse AIS [15]. In our experiment on CIFAR-10 model, similar to reports in [35], there is still a substantial gap between AIS and Reverse AIS estimation, even after very substantial computational effort. In Table 3.1, we report result from Reverse AIS, as it tends to over-estimate the partition function thus underestimate the density. Our reported density value on the CIFAR dataset underperforms other models likely due to two reasons: the model is approximating a Gaussian kernel density estimator of the data distribution, which is not a very good model on its own. Also, the lower bound obtained by reverse AIS may not be tight due to the difficulty in sampling.

We also report a density of 6.79 bits/dim on MNIST dataset, again not comparable to other density models. Density reported here follows the convention of measuring density of pixel values between $[0, 255]$. More details on this experiment is provided in the Appendix.

Outlier Detection. [24] and [103] have reported intriguing behavior of high dimensional density models on out of distribution samples. Specifically, they showed that a lot of models assign higher likelihood to out of distribution samples than real data

[3]Author reported difficulties evaluating Likelihood
[3]Upper Bound obtained by Reverse AIS

samples. We investigated whether our model behaves similarly.

Our energy function is only trained outside the data manifold where samples are noisy, so the energy value at clean data points may not always be well behaved. Therefore, we added noise with magnitude σ_0 before measuring the energy value. We find that our network behaves similarly to previous likelihood models, it assigns lower energy, thus higher density, to some OOD samples. We show one example of this phenomenon in Appendix Figure B.7A.

We also attempted to use the denoising performance, or the objective function to perform outlier detection. Intriguingly, the results are similar as using the energy value. Denoising performance seems to correlate more with the variance of the original image than the content of the image.

3.5 Discussion

The central goal of our work is to investigate how to build EMBs in high-dimensional spaces, with an objective function similar to denoising score matching, or the "Fisher divergence". We first provided analyses and empirical results for understanding the limitations of learning the structure of high-dimensional data with denoising score matching. We found that the objective function L_{SM} confines learning to a small set due to the measure concentration phenomenon in high-dimensional random vectors. Therefore, sampling the learned distribution starting from the outside of the set, where the gradient is learned more accurately, does not produce good result. In our opinion, the expectation w.r.t. the target distribution $\mathbb{E}_{p_{\sigma 0}(\tilde{\boldsymbol{x}})}$ is not of critical importance and it actually only enforces the score matching in the confined high-probability region. Since sampling from random initial location requires the model score matches the target distribution score everywhere, we propose that $\mathbb{E}_{p_{\sigma 0}(\tilde{\boldsymbol{x}})}$ should be replaced by $\mathbb{E}_{p_M(\tilde{\boldsymbol{x}})}$, where $p_M(\tilde{\boldsymbol{x}})$ covers a much larger range in the signal space than $p_{\sigma 0}(\tilde{\boldsymbol{x}})$. This leads to the multiscale denoising score matching, which can be viewed as "multiscale Fisher Divergence". The resulting *Multiscale Denoising Score Matching* (MDSM) EBM model is capable of denoising, producing high-quality samples from random noise, performing image inpainting, etc. While also providing density information, our model learns an order of magnitude faster than the models based on maximum likelihood and sampling.

Previous efforts to learn energy-based models with score matching [82, 146] are unable to produce high-quality samples, and sometimes are computationally intensive. [135] and [134] trained energy-based models with the denoising score matching objective, their method is compuatationally efficient but the resulting models cannot perform sample synthesis from random noise initialization. The recently proposed NCSN [143], though does not belong to EBMs, is capable of high-quality sample synthesis. This model learns a sequence of score functions, each approximates the data smoothed by a different sized Gaussian. Sample generation in NCSN is achieved by sequentially sampling from this set of distributions. Our MDSM method instead learns an energy-based model corresponding to $p_{\sigma_0}(\tilde{\boldsymbol{x}})$ for a fixed σ_0. This method improves score matching in high-dimensional space by matching the gradient of an energy function to the score of $p_{\sigma_0}(\tilde{\boldsymbol{x}})$ in the whole space and avoids measure concentration issue.

All told, we offer a novel EBM model that achieves high-quality sample synthesis, which among EBM approaches provides a new state-of-the art. Compared to the NCSN model, our model is more parsimonious and can support single step denoising without prior knowledge of the noise magnitude. Our model performs sightly worse than the NCSN model in sample quality, which could have several reasons. First, the derivation of Equation 3.6 requires an approximation to keep the training procedure tractable, which could be inaccurate. Second, the NCSN's output is a vector that, at least during optimization, does not always have to be the derivative of a scalar function. In contrast, in our model the network output is a scalar. Thus it is possible that the NCSN model performs better because it explores a larger set of functions during optimization.

3.6 Epilogue

Although [143] and MDSM achieved strong generative performance, the formulation of learning data distribution perturbed by different level of noises is still unsatisfactory. However, a later work [68] showed that the Denoising Score Matching objective can be used to train a diffusion probabilistic model [140] very efficiently. Notably, the new diffusion model achieves extremely strong generative performance, reaching as low as 3.17 on CIFAR-10 FID score, for example. Later it was shown that diffusion model outperforms GAN in larger scale image synthesis [30] without speed limitations [171]. Therefore, it appears that diffusion is the better formulation compared to multi-scale sampling in [143] and Annealed Langevine Dynamics used in this Chapter.

Further study showed that diffusion model has strong connection to stochastic differential equations [145], and it can be cast to a Maximum Likelihood formulation, where the exact density of samples can be evaluated [144]. This also provided a principled way to set the weight of different noise scales. Ironically, denoising score matching, which was originally considered an alternative to maximum likelihood, turned out to be compatible with maximum likelihood, but still having the advantage of not needing to perform sampling.

Besides score matching, various other techniques has been developed to train EBMs without sampling, or with greatly reduced sampling burden. Examples include Diffusion Recovery Likelihood [44], Flow Contrastive Estimation [45], Improved Contrastive Divergence [36], Learned Stein Discrepancy[50] ,None-Newtoniam HMC [161], etc. Another class of methods use EBM as a correction to another generative model with tractable density, for example, Neural Transport MCMC[114] and VAEBM[172]. It is also possible to train a generator network in place of MCMC to train EBM[91][38][2]. It is fair to say research into EBMs has been pretty thorough.

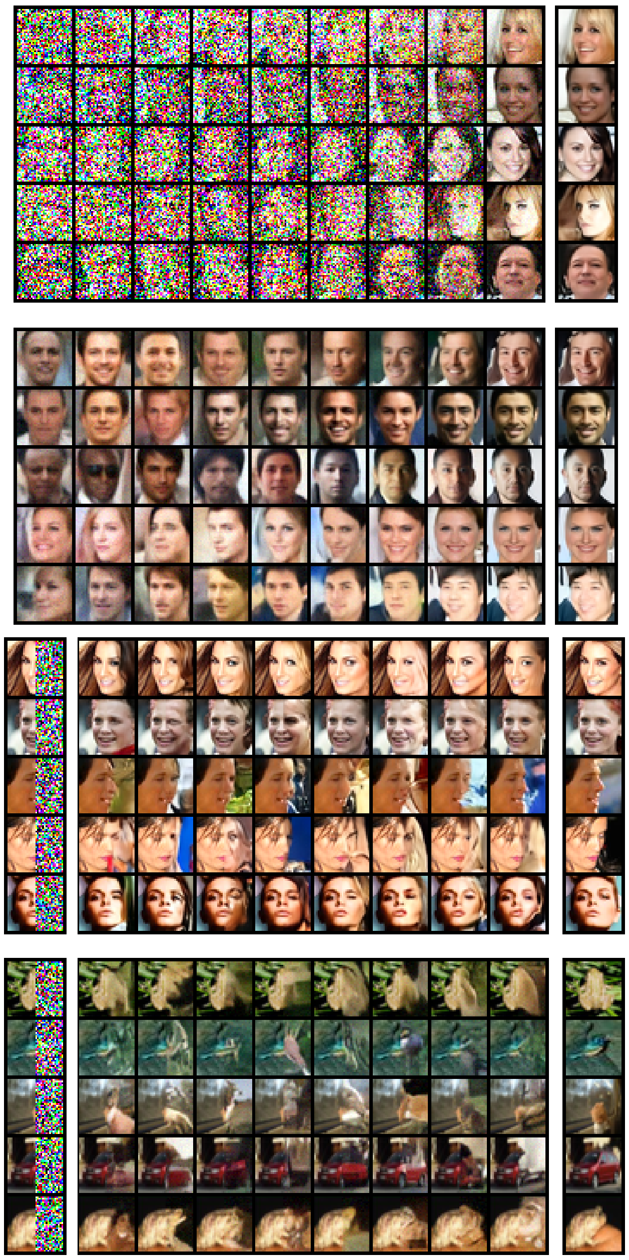


Figure 3.3: Demonstration of the sampling process (top two), and image inpainting (bottom two). The sampling process is shown with Gaussian noise (first), and denoised by single step gradient jump (second). The column next to sampling process shows samples after the last denoising jump at the end of sampling. Inpainting results are shown next to initial image (left column) and the ground truth image (right column).

Chapter 4

A Neural Network MCMC Sampler That Maximizes Proposal Entropy

4.1 Introduction

Sampling from unnormalized distributions is important for many applications, including statistics, simulations of physical systems, and machine learning. However, the inefficiency of state-of-the-art sampling methods remains a main bottleneck for many challenging applications, such as protein folding [116], energy-based model training [115], etc.

A prominent strategy for sampling is the Markov Chain Monte Carlo (MCMC) method [107]. In MCMC, one chooses a transition kernel that leaves the target distribution invariant and constructs a Markov Chain by applying the kernel repeatedly. The MCMC method relies only on the ergodicity assumption. Other than that it is general, if enough computation is performed, the Markov Chain generates correct samples from any target distribution, no matter how complex the distribution is. However, the performance of MCMC depends critically on how well the chosen transition kernel explores the state space of the problem. If exploration is ineffective, samples will be highly correlated and of very limited use for downstream applications. Thus, despite the theoretical guarantee that MCMC algorithms are exact, practically they may still suffer from inefficiencies.

Take, for example, Hamiltonian Monte Carlo (HMC) sampling [106], a type of

MCMC technique. HMC is regarded state-of-the-art for sampling in continuous spaces [126]. It uses a set of auxiliary momentum variables and generates new samples by simulating a Hamiltonian dynamics starting from the previous sample. This allows the sample to travel in state space much further than possible with other techniques, most of whom have more pronounced random walk behavior. Theoretical analysis shows that the cost of traversing an d-dimensional state space and generating an uncorrelated proposal is $O(d^{\frac{1}{4}})$ for HMC, which is lower than $O(d^{\frac{1}{3}})$ for Langevin Monte Carlo, and $O(d)$ for random walk. However, unfavorable geometry of a target distribution may still render HMC ineffective because the Hamiltonian dynamics has to be simulated numerically. Numerical errors in the simulation are corrected by a Metropolis-Hastings (MH) accept-reject step of a proposed sample. If the the target distribution has unfavorable geometric properties, for example, large differences in variance along different directions, the numerical integrator in HMC will produce high errors, leading to a very low accept probability [13]. For simple distributions this inefficiency can be mitigated by an adaptive re-scaling matrix [106]. For analytically tractable distributions, one can also use the Riemann manifold HMC method [47]. But in most other cases, the Hessian required in Riemann manifold HMC algorithm is often intractable or expensive to compute, preventing its application.

Recently, approaches have been proposed that inherit the exact sampling property from the MCMC method, while potentially mitigating the described issues of unfavorable geometry. One approach is MCMC samplers augmented with neural networks [142, 95, 53], the other approach is neural transport MCMC techniques [69, 114]. A disadvantage of these recent techniques is that their objectives optimize the quality of proposed samples, but do not explicitly encourage exploration speed of the sampler. One notable exception is L2HMC [95], a method whose objective includes the size of the expected L2 jump, thereby encouraging exploration. But the L2 expected jump objective is not very general, it only works for simple distributions (see Figure 4.1, and below).

In another recent work [156], exploration speed is encouraged by a quite general objective: the entropy of the proposal distribution. In continuous space, the entropy of a distribution is essentially the logarithm of its volume in state space. Thus, the entropy objective naturally encourages the proposal distribution to “fill up” the target state space as well as possible independent of the geometry of the target distribution. The authors demonstrated the effectiveness of this objective on samplers with simple linear adaptive parameters.

Here we employ the entropy-based objective in a neural network MCMC sampler for optimizing exploration speed. To build the model, we design a flexible proposal distribution where the optimization of the entropy objective is tractable. Inspired by the HMC and the L2HMC[95] algorithm, the proposed sampler uses a special architecture that utilizes the gradient of the target distribution to aid sampling. The proposed method is demonstrated for a 2D distribution in Figure 4.1. The sampler, trained with the entropy-based objective, generates samples that explore the target distribution quite well. In contrast, sampling by constructing proposals with higher L2 expected jump leads to a less desirable result (right panel).

The reminder of the paper is organized as follows. Section 2 briefly introduces MCMC methods. In Section 3 we formulate the model. Section 4 discusses relationships and differences of the new model and HMC-based models from the literature. In Section 5 we will show with experiments that the newly proposed method achieves significant improvement in sampling efficiency over previous techniques. Further we demonstrate how our model can be applied in training an energy-based model of images. The paper concludes with a discussion in Section 6.

4.2 Preliminaries: MCMC methods, From vanilla to Learned

Consider the problem of sampling from a target distribution $p(x) = e^{-U(x)}/Z$ defined by the energy function $U(x)$ in a continuous state space. MCMC methods solve this problem by constructing and running a Markov Chain with a transition probability $p(x'|x)$ that leaves $p(x)$ invariant. The most general invariance condition is: $p(x') = \int p(x'|x)p(x)dx$ for all x', which is typically enforced by the simpler but more restrictive condition of *detailed balance*: $p(x)p(x'|x) = p(x')p(x|x')$.

For a general distribution $p(x)$ it is difficult to directly construct a $p(x'|x)$ that satisfies detailed balance. However, one can easily[1] make any transition probability satisfy detailed balance by adding a Metropolis-Hastings (M-H) accept-reject step [58]. When we sample x' at step t from an arbitrary proposal distribution $q(x'|x^t)$, the M-H accept-reject process accepts the new sample $x^{t+1} = x'$ with probability $A(x', x) = \min\left(1, \frac{p(x')q(x^t|x')}{p(x^t)q(x'|x^t)}\right)$. If x' is rejected, the new sample is set to the previous state $x^{t+1} = x^t$. This transition kernel $p(x'|x)$ constructed from $q(x'|x)$ and $A(x', x)$ leaves any target distribution $p(x)$ invariant.

[1]Up to ergodic and aperiodic assumptions.

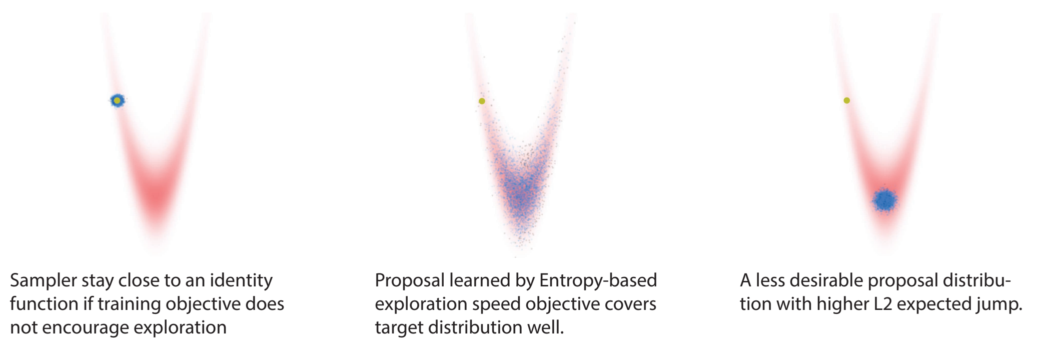


Figure 4.1: Illustration of the benefit of learning for exploring a state space. The three panels show samples from different proposal distributions, with the target distribution depicted in red. In each panel, the large yellow dot on the top left is the initial point x, blue and black dots are accepted and rejected samples from the proposal distribution $q(x'|x)$, respectively. Left panel: Langevin sampler without learning – the samples are highly biased towards the (arbitrary) initial point, exploration is poor and samples will be highly correlated. Middle panel: Samples obtained from our method that optimize the entropy objective – the samples travels far within target distribution $p(x)$ and has little correlation to the initial point. Right panel: Samples that have a higher value in the L2 jump objective than the samples in the middle panel – exploration is still worse despite higher L2 jump.

Most popular MCMC techniques use the described M-H accept-reject step to enforce detailed balance, for example, Random Walk Metropolis (RWM), Metropolis-Adjusted Langevin Algorithm (MALA) and Hamiltonian Monte Carlo (HMC). For brevity, we will focus on MCMC methods that use the M-H step, although some alternatives do exist [139]. These methods share the requirement that the accept probability in the M-H step must be tractable to compute. For two of the mentioned MCMC methods this is indeed the case. In the Gaussian random-walk sampler, the proposal distribution is a Gaussian around the current position: $x' = x + \epsilon * \mathcal{N}(0, \mathbf{I})$, which has the form $x' = x + z$. Thus, forward and reverse proposal probabilities are given by $q(x'|x) = p_{\mathcal{N}}\left[(x' - x)/\epsilon\right]$ and $q(x|x') = p_{\mathcal{N}}\left[-(x' - x)/\epsilon\right]$, where $p_{\mathcal{N}}$ denote the density function of Gaussian with 0 mean and unit diagonal variance. The probability ratio $\frac{q(x^t|x')}{q(x'|x^t)}$ used in the M-H step is therefore equal to 1. In MALA the proposal distribution is a single step of Langevin dynamics with step size ϵ: $x' = x + z$ with $z = -\frac{\epsilon^2}{2}\partial_x U(x) + \epsilon N(0, \mathbf{I})$. We then have $q(x'|x) = p_{\mathcal{N}}\left[(x' - x)/\epsilon + \frac{\epsilon}{2}\partial_x U(x)\right]$ and $q(x|x') = p_{\mathcal{N}}\left[-(x' - x)/\epsilon + \frac{\epsilon}{2}\partial_{x'} U(x')\right]$. Both, the forward and reverse proposal probability are tractable since they are the density of Gaussians evaluated at a known location.

Next we introduce the HMC sampler and show how it can be formulated as a M-H sampler. Basic HMC involves a Gaussian auxiliary variable v of the same dimension as x, which plays the role of the momentum in physics. HMC sampling consists of two steps: 1. The momentum is sampled from a normal distribution $\mathcal{N}(v; 0, \mathbf{I})$. 2. The Hamiltonian dynamics is simulated for a certain duration with initial condition x and v, typically by running a few steps of the leapfrog integrator[106]. Then, a M-H accept-reject process with accept probability $A(x', v', x, v) = \min\left(1, \frac{p(x',v')q(x,v|x',v')}{p(x,v)q(x',v'|x,v)}\right) = \min\left(1, \frac{p(x')p_{\mathcal{N}}(v')}{p(x)p_{\mathcal{N}}(v)}\right)$ is performed to correct for error in the integration process. We have $\frac{q(x,v|x',v')}{q(x',v'|x,v)} = 1$ since the Hamiltonian transition is *volume-preserving* over (x, v). Both HMC steps leave the joint distribution $p(x, v)$ invariant, therefore HMC samples from the correct distribution $p(x)$ after marginalizing over v. To express basic HMC in the standard M-H scheme, step 1 and 2 can be aggregated into a single proposal distribution on x with the proposal probability: $q(x'|x) = p_{\mathcal{N}}(v)$ and $q(x|x') = p_{\mathcal{N}}(v')$. Note, although the probability $q(x'|x)$ can be calculated after the Hamiltonian dynamics is simulated, this term is intractable for general x and x'. The reason is that it is difficult to solve for the v at x to make the transition to x' using the Hamiltonian dynamics. This issue is absent in RWM and MALA, where $q(x'|x)$ is tractable for any x and x'.

Previous work on augmenting MCMC sampler with neural networks also relied on the M-H procedure to ensure asymptotic correctness of the sampling process, for example [142] and [95]. They used HMC style accept-reject probabilities that lead to intractable $q(x'|x)$. Here, we strive for a flexible sampler for which $q(x'|x)$ is tractable. This maintains the tractable M-H step while allowing us to train this sampler to explore the state space by directly optimizing the proposal entropy objective, which is a function of $q(x'|x)$.

4.3 Gradient based sampler with tractable proposal probability

We “abuse” the power of neural networks to design a sampler that is flexible and has tractable proposal probability $q(x'|x)$ between any two points. However, without any extra help, the sampler would be modeling a conditional distribution $q(x'|x)$ with brute force, which might be possible but requires a large model capacity. Thus, our method uses the gradient of the target distribution to guide proposal distribtion. Specifically, we use an architecture similar to L2HMC [95], which itself was inspired by the HMC algorithm and RealNVP [32]. To quantify the benefit of using the target distribution gradient, we provide ablation studies of our model in the Appendix C.1.

Proposal model and how to use gradient information

We restrict our sampler to the simple general form $x' = x + z$. As discussed in Section 4.2, the sampler will have tractable proposal probability if one can calculate the probability of any given z. To fulfill this requirement, we model vector z by a flow model[2]: $z = f(z_0; x, U)$, with inverse $z_0 = f^{-1}(z; x, U)$. Here z_0 is sampled from a fixed Gaussian base distribution. The flow model f is a flexible and trainable invertible function of z conditioned on x, U, and it has tractable Jacobian determinant w.r.t. z. The flow model f can be viewed as a change of variable from the Gaussian base distribution $z0$ to z. The proposed sampler then has tractable forward and reverse proposal probability: $q(x'|x) = p_Z(x' - x; x)$, $q(x|x') = p_Z(x - x'; x')$, where $p_Z(z; x) = p_{\mathcal{N}}(z_0)|\frac{\partial z}{\partial z_0}|^{-1}$ is the density defined by the flow model f. Note, this sampler is ergodic and aperiodic, since $q(x'|x) \neq 0$ for and x and x', which follows from the invertibility of f. Thus, combined with the M-H step, the sampling process will be asymptotically correct. The sampling process first consists of drawing from $p_{\mathcal{N}}(z_0)$ and then evaluating $z = f(z_0; x, U)$ and $q(x'|x)$. Next, the reverse $z_0' = f^{-1}(-z; x + z, U)$ is evaluated at $x' = x + z$ to obtain the reverse proposal probability $q(x|x')$. Finally, the sample is accepted with the standard M-H rule.

For the flow model f, we use an architecture similar to a non-volume preserving coupling-based flow RealNVP [32]. In the coupling-based flow, half of the components of the state vector are kept fixed and used to update the other half through an affine transform parameterized by a neural network. The gradient of the target distribution enters our model in those affine transformations. To motivate this particular model choice, we take a closer look at the standard HMC algorithm. Basic HMC starts with drawing a random initial momentum v^0, followed by several steps of leapfrog integration. In the nth leapfrog step, the integrator first updates v with a half step of the gradient: $v^{n\prime} = v^{n-1} - \frac{\epsilon}{2}\partial_x U(x^{n-1})$, followed by a full step of x update: $x^n = x^{n-1} + \epsilon v^{n\prime}$, and another half step of v update: $v^n = v^{n\prime} - \frac{\epsilon}{2}\partial_x U(x^n)$. After several steps, the overall update of x can be written as: $x^n = x^0 + \sum_{i=0}^{n} v^{i\prime}$, which has the form $x' = x + z$ with $z = \sum_i^n v^{i\prime} = -nv^0 - \frac{n\epsilon}{2}\left[\partial_x U(x^0)\right] - \epsilon\left[\sum_{i=1}^{n}(n-i)\partial_x U(x^i)\right]$. This equation suggest that when generating z through affine transformations, gradi-

[2]For more details on flow models, see [86, 119].

ent should enter through the shift term with a negative sign.

Model formulation

To formulate our model (Equation 4.1, 4.2), we use a binary mask vector m and its complement $\overline{m}$ to select half of z's dimensions for update at a time. As discussed above, we include the gradient term with a negative sign in the shift term. We also use an element-wise scaling on the gradient term as in [95]. However, two issues remain. First, as required by the coupling-based architecture, the gradient term can only depend on the masked version of vector z. Second, it is unclear where the gradient should be evaluated to sample effectively. As discussed above, the sampler should evaluate the gradient at points far away from x, similar as in HMC, to travel long distances in the state space. To handle these issues, we use another neural network R which receives x and the masked z as input, and evaluates the gradient at $x + R$. During training, R learns where the gradient should be evaluated based on the masked z.

We denote the input to network R by $\zeta_m^n = (x, m \odot z^n)$ and the input to the other networks by $\xi_m^n = (x, m \odot z^n, \partial U(x + R(\zeta_m^n)))$, where $\odot$ is the Hadamard product (element wise multiply). Further, we denote the neural network outputs that parameterize the affine transform by $S(\xi_m^n)$, $Q(\xi_m^n)$ and $T(\xi_m^n)$. For notational clarity we omit dependencies of the mask m and all neural network terms on the step number n.

Additionally, we introduce a scale parameter ϵ, which modifies the x update to $x' = x + \epsilon z$. We also define $\epsilon' = \epsilon/(2N)$, with N the total number of z update steps. This parameterization makes our sampler equivalent to the MALA algorithm with step size ϵ at initialization, where the neural network outputs are zero. The resulting update rule is:

$$z^{n\prime} = m \odot z^{n-1} + \overline{m} \odot \left(z^{n-1} \odot \exp[S(\xi_m^{n-1})] - \epsilon' \{\partial U[x + R(\zeta_m^{n-1})] \odot \exp[Q(\xi_m^{n-1})] + T(\xi_m^{n-1})\}\right) \tag{4.1}$$

$$z^n = \overline{m} \odot z^{n\prime} + m \odot \left(z^{n\prime} \odot \exp[S(\xi_{\overline{m}}^{n\prime})] - \epsilon' \{\partial U \left[x + R(\zeta_{\overline{m}}^{n\prime})\right] \odot \exp[Q(\xi_{\overline{m}}^{n\prime})] + T(\xi_{\overline{m}}^{n\prime})\}\right) \tag{4.2}$$

The log determinant of N steps of transformation is:

$$\log \left| \frac{\partial z^N}{\partial z^0} \right| = \epsilon \, \mathbf{1} * \mathbf{1} + \sum_{n=1}^{N} \mathbf{1} * \left[\overline{m} \odot S(\xi_m^{n-1})\right] + \mathbf{1} * \left[m \odot S(\xi_{\overline{m}}^{n'})\right] \tag{4.3}$$

where $\mathbf{1}$ is the vector of 1-entries with the same dimension as z, $*$ denotes the dot product.

Optimizing the proposal entropy objective

The proposal entropy can be expressed as:

$$H\left(X'|X=x\right)=-\int dx' q\left(x'|x\right)\log\left[q\left(x'|x\right)\right]=-\int dz^0 p_{\mathcal{N}}\left(z^0\right)\left[\log\left(p_{\mathcal{N}}\left(z^0\right)\right)-\log\left|\frac{\partial z^N}{\partial z^0}\right|\right] \tag{4.4}$$

For each x, we aim to optimize $S(x) = \exp\left[\beta H(X'|X=x)\right] \times a(x)$, where $a(x) = \int A(x',x)q(x'|x)dx'$ is the average accept probability of the proposal distribution at x. Following [156], we transform this objective into log space and use Jensen's inequality to obtain a lower bound:

$$\begin{aligned}\log S(x) &= \log\int A(x',x)q(x'|x)dx' + \beta H(X'|X=x)\\ &\geq \int \log\left[A(x'x)\right]q(x'|x)dx' + \beta H(X'|X=x) = L(x)\end{aligned}$$

The distribution $q(x'|x)$ is reparameterizable, therefore the expectation over $q(x'|x)$ can be expressed as expectation over $p_{\mathcal{N}}(z_0)$. Expanding the lower bound $L(x)$ and omitting the (constant) entropy of the base distribution $p_{\mathcal{N}}(z_0)$, we arrive at:

$$L(x) = \int dz^0 p_{\mathcal{N}}(z^0)\left[\min\left(0, \log\frac{p(x')}{p(x)} + \log\frac{q(x|x')}{q(x'|x)}\right) - \beta\log\left|\frac{\partial z^N}{\partial z^0}\right|\right] \tag{4.5}$$

During training we maximize $L(x)$ with x sampled from the target distribution $p(x)$ if it is available, or with x obtained from the bootstrapping process [142] which maintains a buffer of samples and updates them continuously. Typically, only one sample of z^0 is used for each x.

A curious feature of our model is that during training one has to back-propagate over the gradient of the target distribution multiple times to optimize R. In [156] the authors avoid multiple back-propagation by stopping the derivative calculation at the density gradient term. In our experiment we do not use this trick and perform full back-propagation without encountering any issue. We found that stopping the derivative computation instead harms performance.

The entropy-based exploration objective contains a parameter β that controls the balance between acceptance rate and proposal entropy. As in [156], We use a simple adaptive scheme to adjust β to maintain a constant accept rate close to a target accept rate. The target accept rate is chosen empirically. As expected, we find that the target accept rate needs to be lower for more complicated distributions.

4.4 Related work: other Neural Network samplers inspired by HMC

Here we discuss other neural network MCMC samplers and how they differ from our method. Methods we compare ours to in the Results are marked with **bold font**.

A-NICE-MC [142], which was generalized in [148], used the same accept probability as HMC, but replaced the Hamiltonian dynamics by a flexible volume-preserving flow [31]. A-NICE-MC matches samples from $q(x'|x)$ directly to samples from $p(x)$, using adversarial loss. This permits training the sampler on empirical distributions, i.e., in cases where only samples but not the density function is available. The problem with this method is that samples from the resulting sampler can be highly correlated because the adversarial objective only optimizes for the quality of the proposed sample. If the sampler produces a high quality sample x, the learning objective does not encourage the next sample x' to be substantially different from x. The authors used a pairwise discriminator that empirically mitigated this issue but the benefit in exploration speed is limited.

Another related sampling approach is **Neural Transport MCMC** [100, 69, 114] , which fits a distribution defined by a flow model $p_g(x)$ to the target distribution using $\mathbf{KL}[p_g(x)||p(x)]$. Sampling is then performed with HMC in the latent space of the flow model. Due to the invariance of the KL-divergence with respect to a change of variables, the "transported distribution" in z space $p_{g^{-1}}(z)$ will be fitted to resemble the Gaussian prior $p_{\mathcal{N}}(z)$. Samples of x can then be obtained by passing z through the transport map. Neural transport MCMC improves sampling efficiency compared to sampling in the original space because a distribution closer to a Gaussian is easier to sample. However, the sampling cost is not a monotonic function of the KL-divergence used to optimize the transport map [92].

Another line of work connects the MCMC method to Variational Inference [132, 185]. Simply put, they improve the variational approximation by running several steps of MCMC transitions initialized from a variational distribution. The MCMC steps are optimized by minimizing the KL-divergence between the resulting distribution and the true posterior. This amounts to optimizing a "burn in" process in MCMC. In our setup however, the exact sampling is guaranteed by the M-H process, thus the KL divergence loss is no longer applicable. Like in variational inference, the **Normalizing flow Langevin MC** (NFLMC) [53] also used a KL divergence loss. Strictly speaking, this model is a normalizing flow but not a MCMC method. We compare our method to it, because the model architecture, like ours, uses the gradient of the target distribution.

Another related technique is [109], where the authors trained an independent M-H sampler by minimizing $\mathbf{KL}\left[p(x)q(x'|x)||p(x')q(x|x')\right]$. This objective can be viewed as a lower bound of the M-H accept rate. However, as discussed in [156], this type of objective is not applicable for samplers that condition on the previous state.

All the mentioned techniques have in common that their objective does not encourage exploration speed. In contrast, **L2HMC** [95, 154] does encourage fast exploration of the state space by employing a variant of the expected square jump objective [122]: $L(x) = \int dx' q(x'|x)A(x', x)||x' - x||^2$. This objective provides a learning signal even when x is drawn from the exact target distribution $p(x)$. L2HMC generalized the Hamiltonian dynamics with a flexible non-volume-preserving transformation [32]. The architecture of L2HMC is very flexible and uses gradient of target distribution. However, the L2 expected jump objective in L2HMC improves exploration speed only in well-structured distributions (see Figure 4.1).

The shortcomings of the discussed methods led us to consider the use of an entropy-based objective. However, L2HMC does not have tractable proposal probability $p(x'|x)$, preventing the direct application of the entropy-based objective. In principle, the proposal entropy objective could be optimized for the L2HMC sampler with variational inference [124, 141], but our preliminary experiments using this idea were not promising. Therefore, we designed our sampler to possess tractable proposal probability and investigated tractable optimization of the proposal entropy objective.

4.5 Experimental result

synthetic dataset and bayesian logistic regression

First we demonstrate that our technique accelerates sampling of the funnel distribution, a particularly challenging example from [108]. We then compare our model with A-NICE-MC [142], L2HMC [95], Normalizing flow Langevin MC (NFLMC) [53] as well as NeuTra [69] on several other synthetic datasets and a Bayesian logistic regression task. We additionally compare to gradMALA [156] to show the benefit of using neural network over linear adaptive sampler. For all experiments, we report Effective Sample Size [71] per M-H step (ESS/MH) and/or ESS per target density gradient evaluation (ESS/grad). All results are given in *minimum* ESS over all dimensions unless otherwise noted. In terms of these performance measures, larger numbers are better.

Here a brief description of the datasets used in our experiments:

Ill Conditioned Gaussian: 50 dimensional ill-conditioned Gaussian task described in [95], a Gaussian with diagonal covariance matrix with log-linearly distributed entries between $[10^{-2}, 10^{2}]$.

Strongly correlated Gaussian: 2 dimensional Gaussian with variance $[10^{2}, 10^{-1}]$ rotated by $\frac{\pi}{4}$, same as in [95].

Funnel distribution: The density function is $p_{funnel}(x) = \mathcal{N}(x_0; 0, \sigma^2)\mathcal{N}(x_{1:n}; 0, \mathbf{I}\exp(-2x_0))$. This is a challenging distribution because the spatial scale of $x_{1:n}$ varies drastically depending on the value of x_0. This geometry causes problems to adaptation algorithms that rely on a spatial scale. An important detail is that earlier work, such as [12] used $\sigma = 3$, while some recent works used $\sigma = 1$. We run experiments with $\sigma = 1$ for comparison with recent techniques and also demonstrate our method on a 20 dimensional funnel distribution with $\sigma = 3$. We denote the two variants by Funnel-1 versus Funnel-3.

Bayesian Logistic regression: We follow the setup in [71] and used the German, Heart and Australian datasets from the UCI data registry.

In Figure 4.2, we compare our method with HMC on the 20d Funnel-3 distribution. As discussed in [12], the stepsize of HMC needs to be manually tuned down to allow traveling into the neck of the funnel, otherwise the sampling process will be biased. We thus tune the stepsize of HMC to be the largest that still allows traveling into the neck. Each HMC proposal is set to use the same number of gradient steps as each proposal of the trained sampler. As can be seen, the samples proposed by our method travel significantly further than the HMC samples. Our method achieves 0.256 (ESS/MH), compared to 0.0079 (ESS/MH) with HMC.

As a demonstration we provide a visualization of the resulting chain of samples in Figure 4.2 and the learned proposal distributions in Appendix C.2. The energy value for the neck of the funnel can be very different than for the base, which makes it hard for methods such as HMC to mix between them [12]. In contrast, our model can produce very asymmetric $q(x'|x)$ and $q(x|x')$, making mixing between different energy levels possible.

Performances on other synthetic datasets and the Bayesian Logistic Regression are

[1]Author reported difficulties evaluating Likelihood

[2]Upper Bound obtained by Reverse AIS

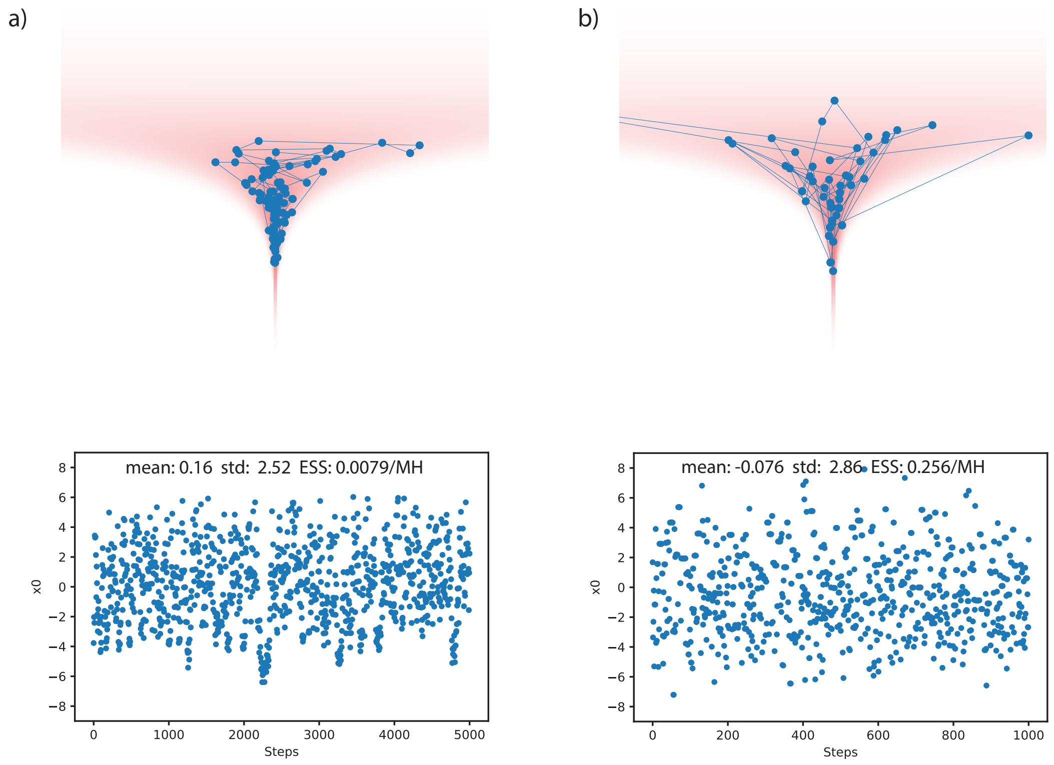


Figure 4.2: Comparison of our method with HMC on the 20d Funnel-3 distribution. a) Chain and samples of x_0 (from neck to base direction) for HMC. b) Same as a) but for our learned sampler. Note, samples in a) look significantly more correlated than those in b), although they are plotted over a longer time scale.

shown in Table 4.1. In all these datasets our method outperformed previous neural network based MCMC approaches by significant margin. Our model also outperform gradMALA [156], which use the same objective but only use linear adaptive parameters. The experiments used various parameter settings, as detailed in Appendix C.2. Results of other models are adopted or converted from numbers reported in the original papers. The Appendix provides further experimental results, ablation studies, visualizations and details on the implementation of the model.

Dataset (measure)	**L2HMC**	**Ours**
50d ICG (ESS/MH)	0.783	**0.86**
2d SCG (ESS/MH)	0.497	**0.89**
50d ICG (ESS/grad)	7.83×10^{-2}	$\mathbf{2.15 \times 10^{-1}}$
2d SCG (ESS/grad)	2.32×10^{-2}	$\mathbf{2.2 \times 10^{-1}}$

Dataset (measure)	**Neutra**	**Ours**
Funnel-1 x_0 (ESS/grad)	8.9×10^{-3}	$\mathbf{3.7 \times 10^{-2}}$
Funnel-1 $x_{1 \cdots 99}$(ESS/grad)	4.9×10^{-2}	$\mathbf{7.2 \times 10^{-2}}$

Dataset (measure)	**A-NICE-MC**	**NFLMC**	**Ours**
German (ESS/5k)	926.49	1176.8	**3150**
Australian (ESS/5k)	1015.75	1586.4	**2950**
Heart (ESS/5k)	1251.16	2000	**3600**

Table 4.1: Performance Comparisons. SCG: strongly correlated Gaussian, ICG: Ill-conditioned Gaussian. German, Autralian, Heart: Datasets for Bayesian Logistic regression. ESS: Effective Sample Size (a correlation measure)

Training a convergent deep energy-based model

A very challenging application of the MCMC method is training a deep energy-based model (EBM) of images [173, 115, 35]. We demonstrate stable training of a convergent EBM, and that the learned sampler achieves better proposal entropy early during training, as well as better sample quality at convergence, compared to the MALA algorithm. An added benefit is that, like in adaptive MALA, tuning the Langevin dynamics step size is no longer needed, instead, one only needs to specify a target accept rate. This contrasts earlier work using unadjusted Langevin dynamics, where step size needs to be carefully tuned [115].

Following [115], we use the Oxford flowers dataset of 8189 28∗28 colored images. We dequantize the images to 5 bits by adding uniform noise and use logit transform [32]. Sampling is performed in the logit space with variant 2 of our sampler (without the R network, see Appendix C.1). During training, we use Persistent Contrastive Divergence (PCD) [155] with a replay buffer size of 10000. We alternate between training the sampler and updating samples for the EBM training. Each EBM training step uses 40 sampling steps, with a target accept rate of 0.6.

Figure 4.3 depicts samples from the trained EBM replay buffer, as well as samples from a 100k step sampling process –for demonstrating that in general the sampling sequence converges to a fixed low energy state. We also show that early during training, the proposal entropy of the learned sampler is higher than that of an adaptive MALA algorithm with the same accept rate target. Later during training, the proposal entropy is not significantly different (See Figure C.3 a)). This is likely because the explorable volume around samples becomes too small for the learned sampler to make a difference. Additionally, we show that the model trained with the learned sampler achieves better sample quality by evaluating the Fréchet Inception Distance (FID) [63] between the replay buffer and ground truth data. A model trained with the learned sampler achieves 38.1 FID, while a model trained with MALA achieves 43.0 FID (evaluated at a late checkpoint, lower is better). We provide a plot that tracks the FID during training in Appendix Figure C.3.

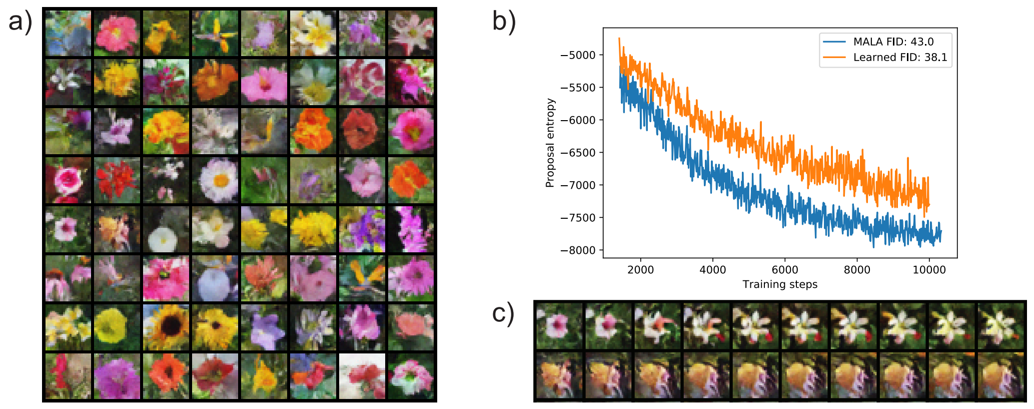


Figure 4.3: Training of convergent EBM with pixel space sampling. a) Samples from replay buffer after training. b) Proposal entropy of trained sampler vs. MALA early during training –note that entropy of learned sampler is significantly higher. c) Samples from 100k sampling steps by the learned sampler, initialized at samples from replay buffer. Large transitions like the one in the first row are rare, this atypical example was selected for display.

4.6 Discussion

In this paper we propose a gradient based neural network MCMC sampler with tractable proposal probability. The training is based on the entropy-based exploration speed objective. Thanks to an objective that explicitly encourages exploration, our method achieves better performance than previous neural network based MCMC samplers on a variety of tasks. Compared to the manifold HMC [12] methods, our model provides a more scalable alternative for mitigating unfavorable geometry in the target distribution.

There are many potential applications of our method beyond what is demonstrated in this paper. For example, training latent-variable models [70], latent sampling in GANs [19], and others applications outside machine learning, such as molecular dynamics simulation [116]. For future research, it would be interesting to investigate other architectures, such as an auto-regressive architecture, or alternative masking strategies to improve the expressiveness of the proposed model. Another promising direction could be to combine our technique with neural transport MCMC.

Our proposed sampler provides more efficient exploration of the target distribution, as measured by the ESS results and proposal entropy. However, the exploration provided is *local*. In EBM training for example, as reported previously [115], the learned energy landscape is highly multi-modal with high energy barriers in between the minimas. Sample proposal cannot cross those high barriers since it will result in high rejection probability. Our sampler achieves a small level of mixing as is visible in some sampling examples. But our sampler, being a local algorithm, cannot explore different modes efficiently – it exhibits the same shortcoming in mixing as Langevin dynamics sampler.

4.7 Epilogue

Improving the sampling efficiency of MCMC techniques continued to be an active area of research [16]. Although the proposal entropy of HMC is not exactly tractable, [67] developed an approximation that can be used to optimize the mass matrix. Adaptive MCMC using neural networks also found application in sampling the Lattice Gauge Theory in Physics [42].

One interesting development in HMC is the use of non-Newtonian momentum instead of the quadratic momentum from physics [161]. Dynamics with a quadratic momentum tends to spend longer time at high energy areas, where the momentum and speed is low, but plausible samples are mostly at low energy areas. By reversing this effect with a hand-crafted momentum term, HMC sampling efficiency can be greatly improved. The sampler is shown to work much better than Langevin dynamics on training convergent EBM on images.

Chapter 5

Neural Manifold Clustering and Embedding

5.1 Introduction

Here we investigate unsupervised representation learning, which aims to learn structures (features) from data without the use of any label. If the data lie in a linear subspace, the linear subspace can be extracted by Principal Component Analysis (PCA) [77], one of the most basic forms of unsupervised learning. When the data occupy a union of several linear subspaces, subspace clustering (SC) [163] is needed to cluster each data point to a subspace as well as estimating the parameters of each subspace. Here we are concerned with even more challenging scenarios, when data points come from a union of several non-linear low-dimensional manifolds. In such scenarios, the clustering problem can be formulated as follows [39]:

Task 1 *Manifold Clustering and Embedding: Given that the data points come from a union of low-dimensional manifolds, we shall segment the data points based on their corresponding manifolds, and obtain a low-dimensional embedding for each manifold.*

Various methods have been developed to solve this problem [3], but it is still an open question how to use neural networks effectively in manifold clustering problems [56]. In this paper, we propose *neural manifold clustering and embedding* (NMCE) that follows three principles: 1) The clustering and representation should respect a domain-specific constraint, e.g. local neighborhoods, local linear interpolation or data augmentation invariances. 2) The embedding of a particular manifold shall not collapse. 3) The embedding of identified manifolds shall be linearized and separated, i.e. they occupy different linear subspaces. We achieve 1) using data augmentations, and achieve 2) and 3) with the subspace feature learning algorithm maximal coding rate reduction (MCR^2) objective [178]. This work make the following specific contributions:

1. We combine data augmentation with MCR^2 to yield a novel algorithm for general purpose manifold clustering and embedding (NMCE). We also discuss connections between the algorithm and self-supervised contrastive learning.
2. We demonstrate that NMCE achieves strong performance on standard subspace clustering benchmarks, and can outperform the best clustering algorithms on more challenging high-dimensional image datasets like CIFAR-10 and CIFAR-20. Further, empirical evaluation suggests that our algorithm also learns a meaningful feature space.

5.2 Related Work

Manifold Learning. In classical manifold learning, the goal is to map the manifold-structured data points to a low-dimensional representation space such that the manifold structure is preserved. There are two key ingredients: 1) Choosing a geometric property from the original data space to be preserved. For example, the local euclidean neighborhood [10], or linear interpolation by neighboring data points [129]. 2) The embedding should not collapse to a trivial solution. To avoid the trivial solution, the variance of the embedding space is typically constrained in spectral-based manifold learning methods.

Manifold Clustering and Embedding. When the data should be modeled as a union of several manifolds, manifold clustering is needed in addition to manifold learning. When these manifolds are linear, subspace clustering algorithms [98, 40, 163] can be used. When they are non-linear, manifold clustering and embedding methods were proposed. They generally divide into 3 categories [3]: 1. Locality preserving. 2. Kernel based. 3. Neural Network based. Locality preserving techniques implicitly make the assumption that the manifolds are smooth, and are sampled densely [147, 39, 23]. Additionally, smoothness assumption can be employed [48]. Our method generalizes those techniques by realizing them with geometrical constraints. The success of kernel based techniques depends strongly on the suitability of the underlying kernel, and generally requires a representation of the data in a space with higher dimension than the data space [123]. Deep subspace clustering methods, such as [75, 183, 186] jointly perform linear subspace clustering and representation learning, and has the potential to handle high dimensional non-linear data effectively. However, it has been shown that most performance gains obtained by those methods should be attributed to an ad-hoc post-processing step applied to the self-expression matrix. Using neural networks only provide a very marginal gain compared to clustering the raw data directly using linear SC [56]. Our work differs from those techniques mainly in two aspects: i) While most of the previous methods were generative (autoencoders, GANs), our loss function is defined in the latent embedding space and is best understood as a contrastive method. ii) While previous methods use self-expression based SC to guide feature learning, ours uses MCR^2 to learn the subspace features. Recently, some deep SC techniques also applied data augmentation [149, 1]. However, in those works, data augmentation played a complementary role of improving the performance. In our method, data augmentation plays a central role for enabling the identification of the clusters.

Self-Supervised Representation Learning. Recently, self-supervised representation learning achieved tremendous success with deep neural networks. Similar to manifold clustering and embedding, there are also two essential ingredients: 1) Data augmentations are used to define the domain-specific invariance. 2) The latent representation should not collapse. The second requirement can be achieved either by contrastive learning [21], momentum encoder [61, 52] or siamese network structure [22]. More directly related to our work is [152], which proposed feature orthogonalization and decorrelation, alongside contrastive learning. Recently, variance regularization along was also successfully used to achieve principle 2) [180, 7], attaining state-of-the-art SSL representation performance. Part of our method, the total coding rate (TCR) objective achieves a similar effect, see discussion in Appendix D.1. However, beyond self-supervised features, our algorithm additionally show strong clustering performance, and directly learns a meaningful latent space. The simultaneous manifold clustering and embedding in NMCE is also related to online deep clustering [17, 18] method. Also notable is [29], where the concept of population consistency is closely related to the constraint functional we discussed.

Clustering with Data Augmentation Our method use data augmentation to ensure correct clustering of the training data. Although not explicitly pointed out, this is also the case for other clustering techniques [137, 159, 96]. Our understanding of data augmentation is also consistent to works that specifically study the success of data augmentations [57, 89].

5.3 Neural Manifold clustering and Embedding

Problem Setup

Let's assume data points $x_i \in \mathbb{R}^d$ are sampled from a union $\bigcup_{j=1}^{n} \mathsf{X}_j$ of manifolds $\mathsf{X}_1, \mathsf{X}_2, ..., \mathsf{X}_n$.[1] As stated in Task 1, the goal of manifold clustering and embedding is to assign each data point to the corresponding manifold (clustering), as well as learning a coordinate for each manifold (manifold learning). To achieve this goal, we use neural network f, which learns to map a data point x to the feature embedding $z \in \mathbb{R}^{d_{emb}}$ and the cluster assignment $c \in [1, n]$, i.e. $z, c = f(x)$. The cluster assignment shall be equal to the ground-truth manifold assignment,[2] z should parameterize the coordinates of the corresponding manifold X_c.

To make the feature space easier to work with, one can enforce the additional separability requirement that for any $\mathsf{X}_j, \mathsf{X}_k$ and $j \neq k$, the feature vectors are perpendicular, $\mathsf{Z}_j \perp \mathsf{Z}_k$. Here Z_j denotes the embedding feature vectors of data points in X_j, and we define perpendicular between two sets in the following fashion: If $\tilde{\mathsf{Z}}_j \subseteq \mathsf{Z}_j, \tilde{\mathsf{Z}}_k \subseteq \mathsf{Z}_k$ such that $\forall z_j \in \tilde{\mathsf{Z}}_j, z_k \in \tilde{\mathsf{Z}}_k$, we have $z_j \cdot z_k \neq 0$, then either $\tilde{\mathsf{Z}}_j$ or $\tilde{\mathsf{Z}}_k$ has zero measure.

In the following, we first argue that to make clustering possible with a neural network, one should define the additional geometric constraint that makes the manifold clusters identifiable. Second, we discuss how to implement the required geometric constraints and combine them with a recently proposed joint clustering and subspace learning algorithm MCR^2 [178] to achieve neural manifold clustering and embeddng (NMCE).

Clustering always involves implicit assumptions

Even the simplest clustering algorithms rely on implicit assumptions. For example, in k-means clustering, the implicit assumption is that the clusters in the original data space are continuous in terms of L2 distance. For Linear SC, the assumption is that data points are co-linear within each cluster. If a neural network is trained on example data to learn a cluster assignment function $c = f(x)$, the resulting clustering will be arbitrary and not resemble the solution of k-means or linear SC clustering. This is because neural networks are flexible and no constraint is imposed on the

[1]We do not consider topological issues here and assume that all of them are homeomorphic to $\mathbb{R}^{d_i}$ for some d_i

[2]Up to a permutation since the training is unsupervised.

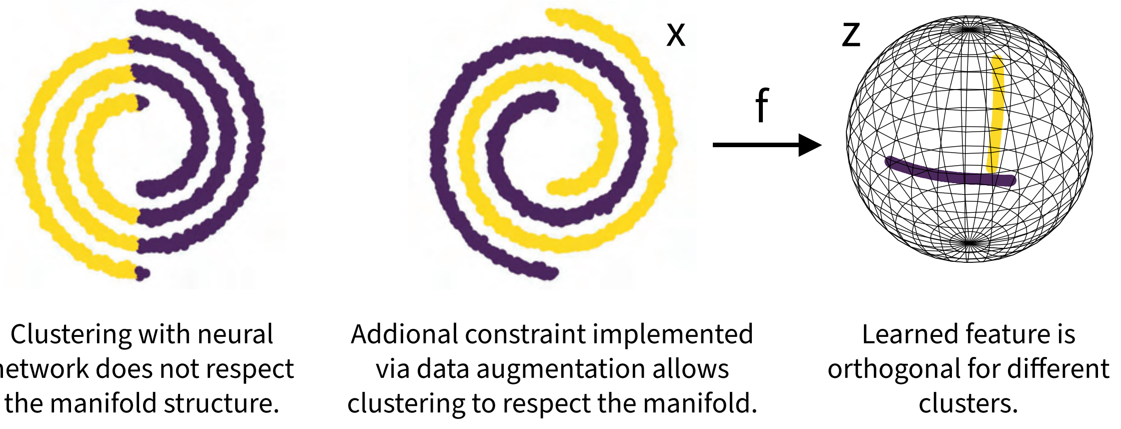

x
f
z
Clustering with neural network does not respect the manifold structure.
Addional constraint implemented via data augmentation allows clustering to respect the manifold.
Learned feature is orthogonal for different clusters.

Figure 5.1: Locality constraint enforced by adding Gaussian noise as data augmentation allows neural network to find the desired non-linear manifolds.

clustering function to force the result to respect the geometry of the original data. One example of this is shown in Figure 5.1 left panel, where a deep clustering method outputs a rather arbitrary clustering for the double spiral data.

To make the clusters learnable for a neural network, one needs to introduce constraints explicitly. In the example in Figure 5.1, one can easily reason that, to be able to separate the two spirals from each other, the clustering function needs to ensure that all close neighbors of a point from the data distribution are assigned to the same cluster, essentially the same assumption implicitly made in locality-based manifold clustering algorithms [3]. We formalize the notion of constraints to a constraint functional $D(f)$ (for a specific data distribution) that has the following property: All cluster assignment functions f that makes D attain its minimum value (assumed to be 0) and also cluster data points to the ground truth number of clusters will correctly cluster all data points. For example, we can construct a functional D that takes value $D = 0$ for all clustering functions that satisfy the locality constraint on the dataset, and $D > 0$ otherwise. This notion of constraint function is completely general. For example, linear subspace clustering is recovered if the constraint function take value 0 if and only if the found clusters are linear subspace.

In practice, one usually cannot optimize the neural network clustering function f subject to $D = 0$, since the correct solution would have to be found at initialization. A more practical way to use the constraint function is to use the relaxed objective with weighting λ:

$$L(f) = L_{clst}(f) + \lambda * D(f) \tag{5.1}$$

where L_{clst} is some objective function that will force f to cluster the dataset. With a suitable λ, optimizing this objective leads to learning of the correct clusters, by the assumption above. To achieve manifold clustering, one just needs to find the appropriate constraint functional.

Subspace feature learning with Maximum Coding Rate Reduction

Having introduced explicit constraints for learning the manifold clustering with neural networks, we still need a concrete algorithm for learning a linear subspace-structured representation given the clustering (manifold learning). Fortunately, the recently proposed principle of Maximum Coding Rate Reduction (MCR^2) [178] provides a principled learning objective for this purpose. We denote the dataset (union of all manifolds) by X, and random variable that represent distribution on each manifold X_i by $\mathcal{X}_i$. For a certain encoder $\mathcal{Z} = f(\mathcal{X})$ (without clustering output), $z \in \mathbb{R}^{d_{emb}}$, the Gaussian coding rate function is defined to be:

$$R(\mathcal{Z}, \epsilon) = \frac{1}{2}\mathsf{logdet}(\mathbf{I} + \frac{d_{emb}}{\epsilon^2}\mathsf{cov}(\mathcal{Z})) \tag{5.2}$$

where cov denotes the covariance matrix function for a vector random variable: $\mathsf{cov}(\mathcal{Z}) = \mathbf{E}_{p(z)}[zz^T]$. This function is approximately the Shannon coding rate of a multivariate Gaussian distribution given average distortion ϵ [27], and can be motivated by a ball-packing argument. However, we focus on its geometric implication and do not discuss the information-theoretic aspect further. Readers interested to the full introduction of this objective can refer to the original papers [178, 98].

Suppose for now that we are also given the cluster assignment function $c(x)$ that outputs the manifold index $c(x) = i$ for $x \in \mathsf{X}_i$. We can then calculate the average coding rate for a particular cluster i as $R(\mathcal{Z}_i, \epsilon)$, where $\mathcal{Z}_i = f(\mathcal{X}_i)$. The MCR^2 principle states that, to learn a subspace-structured representation, one needs to optimize f by maximizing the difference between the coding rate of all of $\mathcal{Z}$ and the sum of coding rate for each cluster $\mathcal{Z}_i$:

$$\Delta R(\mathcal{Z}, c, \epsilon) = R(\mathcal{Z}, \epsilon) - \sum_{i=1}^{n} R(\mathcal{Z}_i, \epsilon) \tag{5.3}$$

Intuitively, this objective minimizes the sum of volume of each cluster to push each of them into a lower-dimensional representation, while a the same time maximize the total volume of the union of all clusters to avoid collapse. It has been shown that MCR^2 guarantees the perpendicularity requirement in the problem setup above. Theorem A.6 in [178] states that under the assumption that $\mathsf{rank}(\mathsf{Z}_i)$ is bounded, and the coding error ϵ is sufficiently low, maximizing the MCR^2 objective guarantees that $\mathsf{Z}_i \perp \mathsf{Z}_j$ for any $i \neq j$, and the rank of each Z_i is as large as possible.

Neural Manifold Clustering and Embedding

The MCR^2 principle can be extended to the case where the labeling function $c(x)$ is not given, but is instead learned jointly by optimizing the MCR^2 objective. In this case, we fuse the clustering function into f: $z, c = f(x)$, note that now $\mathcal{Z}_i$ depends on f implicitly. However, as discussed earlier, one will not be able to find the correct clusters by using MCR^2 alone, as it is unconstrained in the data space. An additional constraint term $D(f)$ has to be included to make the clusters identifiable. This leads to the Neural Manifold Clustering and Embedding (NMCE) objective:

$$L_{NMCE}(f, \epsilon) = \sum_{i=1}^{n} R(\mathcal{Z}_i, \epsilon) - R(\mathcal{Z}, \epsilon) + \lambda D(f) \tag{5.4}$$

It is possible to replace the MCR^2 objective in (5.4) by any other objective for subspace feature learning, however, this generalization is left for future work. Given a suitable constraint functional D, the NMCE objective will guide the neural network to cluster the manifold correctly and also learn the subspace-structured feature, thus achieving manifold clustering and embedding. We explain in the next section how D can be practically implemented.

Implementing constraints via data augmentation

Here we propose a simple method to implement various very useful constraints. The proposed method uses data augmentations, a very common technique in machine learning, and enforce two conditions on the clustering and embedding function f: 1. Augmented data points generated from the same point should belong to the same cluster. 2. The learnt feature embedding of augmented data points should be similar if they are generated from the same point. We use $\mathcal{T}(x)$ to represent a random augmentation of x: $c, z = f(\mathcal{T}(x))$, $c', z' = f(\mathcal{T}'(x))$, then $D(f) = E_{p(x)}\mathsf{sim}(z, z')$. Here, sim is a function that measures similarity between latent representations, for example cosine similarity, or L2 distance. The requirements $c = c'$ can be enforced by using average of c and c' to assign z and z' to clusters during training.

For the double spiral example, the augmentation is simply to add a small amount of Gaussian noise to data points. This will have the effect of forcing neighboring points in the data manifold to also be neighbors in the feature space and be assigned to the same cluster. The result of using this constraint in the NMCE objective can be seen in Figure 5.1 middle and right panel. For more difficult datasets like images, one needs to add augmentations to constrain the clustering in the desired way. The quality of clustering is typically measured by how well it correspond to the underlying content information (object class). Therefore, we need augmentations that perturbs style information (color, textures and background) but preserves content information, so that the clustering will be based on content but not style. Fortunately, extensive research in self-supervised contrastive learning has empirically found augmentations that achieves that very effectively [21, 89].

Multistage training and relationship to self-supervised contrastive learning

In practice, we find that for all but the simplest toy experiments it is difficult to optimize the full NMCE objective (Eq. 5.4) from scratch. This is because the clusters are incorrectly assigned at the start of training, and the sum in Eq 5.4 effectively cancels out the total coding rate term $R(\mathcal{Z}, \epsilon)$, making it hard to initialize the training. We observe that to achieve high coding rate difference, it is critical to achieve high total coding rate first, so we resort to a multistage training procedure, with the first stage always being optimizing the following objective:

$$L_{TCR}(f, \epsilon) = -R(\mathcal{Z}, \epsilon) + \lambda D(f) \tag{5.5}$$

We call this the Total Coding Rate (TCR) objective, which is a novel self-supervised learning objective by itself. Through simple arguments (Appendix D.1) one can see that this objective encourages the covariance matrix of $\mathcal{Z}$ to be diagonal. Along with a similarity constraint between augmented samples, this objective turns out to asymptotically achieve the same goal as VICReg [7] and BarlowTwins [180]. We discuss this further in Appendix D.1.

After training with the TCR objective, we found that usually the feature $\mathcal{Z}$ already possess approximate subspace structure. For simple tasks, the features can be directly clustered with standard linear SC techniques such as EnSC [174]. For more difficult tasks, we found that the full NMCE objective performs much better.

5.4 Results

We provide code for our experiment here[3],

For the synthetic experiments, a MLP backbone is used as backbone and two linear last layers are used to produce feature and cluster logits output, ELU is used as the activation function. For all image datasets, standard ResNet with ReLU activation and various sizes is used as the backbone. After the global average pooling layer, one linear layer with 4096 output units, batch normalization and ReLU activation is used. After this layer, two linear heads are used to produce feature and cluster logits outputs. The number of clusters used is always equal to the ground truth. For all experiments, two augmented samples, or ”views”, are used for each data sample in a batch. Gumbel-Softmax [73] is used to learn the cluster assignments. Similar to [178], feature vectors are always normalized to the unit sphere. The constraint term D is always cosine similarity between two augmented samples. Cluster assignment probabilities and feature vectors are averaged between augmentations before sending into the NMCE loss, which enforce consistency between the augmentations. The regularization strength λ is always determined empirically. Hyper-parameters and further details are available in the Appendix D.2.

Table 5.1: Clustering performance comparison on COIL20 and COIL100. Listed are error rates, NMCE is our method, see text for references for other listed methods .

Dataset	**SSC**	**KSSC**	**AE+EDSC**	**DSC**	**S2ConvSCN**	**MLRDSC-DA**	**AK-SC**	**NMCE (Ours)**
COIL20	14.83	24.65	14.79	5.42	2.14	1.79	**0.0**	**0.0**
COIL100	44.90	47.18	38.88	30.96	26.67	20.67	-	**11.53**

Synthetic and image datasets

For the toy example in Figure 5.1, data augmentation is simply adding a small amount of noise, and full NMCE objective is directly used, the clustering output is jointly learned.

To verify the NMCE objective, we perform synthetic experiment by clustering a mixture of manifold-structured data generated by passing Gaussian noise through two randomly initialized MLPs. Small Gaussian noise data augmentation is used to

[3]https://github.com/zengyi-li/NMCE-release

enforce the locality constraint. A two stage training process is used, the first stage uses TCR objective while the second stage uses full NMCE objective. Much like the toy experiment, the result is consistent with our interpretation, see Appendix D.1 for results and details.

To compare NMCE with other subspace clustering methods, we apply it to COIL20 [110] and COIL100 [111], both are standard datasets commonly used in SC literature. They consist of images of objects taken on a rotating stage in 5 degree intervals. COIL20 has 20 objects with total of 1440 images, and COIL100 has 100 objects with total of 7200 images. Images are gray-scaled and down-sampled by 2x. For COIL20, a 18-layer ResNet with 32 filters are used as the backbone, the feature dimension is 40. For COIL100, and 10-layer ResNet with 32 filters is used, and feature dimension is 200. We determined the best augmentation policy for this experiment by manual experimentation on COIL20, and the same policy is applied to COIL100. Details on the searched policy can be found in Appendix D.2. For this experiment, we use EnSC [174] to cluster features learned with TCR objective. We found this procedure already performs strongly, and full NMCE objective is not necessary.

We compare with classical baseline such as SSC [40],KSSC[123] and AE+EDSC[74], as well as deep SC techniques including , DSC[75], S2ConvSCN[183] and MLRDSC-DA[1]. We also include AK-SC[4], where data-augmentation is combined with classical SC. As can be seen in Table 5.1, our method achieves 0.0 error rate for COIL20, and error rate of 11.53 for COIL100, on par with previous best result of 0.0 for COIL-20, and outperforming previous best result of 20.67 on COIL-100. This suggests that NMCE can better leverage the non-linear processing capability of deep networks, unlike previous deep SC methods, which only marginally outperform linear SC on the pixel space [56].

Self-supervised learning and clustering of natural images

Recently, unsupervised clustering has been extended to more challenging image datasets such as CIFAR-10, CIFAR-20 [88] and STL-10 [25]. The original MCR^2 paper [178] also performed those experiments, but they used augmentation in a very different way (See Appendix D.1 for a discussion). The CIFAR-10 experiment used 50000 images from the training set, CIFAR-20 used 50000 images from training set of CIFAR-100 with 20 coarse labels. The STL-10 experiment used 13000 labeled images from original train and test set. ImageNet-10 and ImagetNet-Dogs used 13000 and 19500 images subset from ImageNet, respectively [96]. We use ResNet-18 as the backbone to compare with MCR^2, and use ResNet-34 as the backbone to compare with other clustering methods. In both cases, the feature dimension is 128. Standard image augmentations for self-supervised learning is used [21].

Table 5.2: Supervised evaluation performance for different feature types and evaluation algorithms. See text for details.

Model	**Proj**	**Pre-feature**	**Pool**	**Proj (16 avg)**	**Pre-feature (16 avg)**	**Pool (16 avg)**
SVM	0.911	0.889	0.895	0.922	0.905	**0.929**
kNN	0.904	0.851	0.105	**0.910**	0.800	0.103
NearSub	0.898	0.902	0.903	0.903	0.909	**0.911**

Training strategy, supervised evaluation

Here we use a three stage training procedure: 1. Train with TCR objective. 2. Reinitialize the last linear projection layer and add a cluster assignment layer. Then, freeze parameters in the backbone network and train the two new layers with full NMCE objective. 3. Fine tune the entire network with NMCE objective.

As discussed earlier, the first stage is very similar to self-supervised contrastive learning. Therefore, we study features learned at this stage with supervised evaluation, the result with ResNet-34 is listed in Table 5.2. To evaluate the learned features, we use SVM, kNN and a Nearest Subspace classifier (NearSub). We use SVM instead of linear evaluation training because we find that it is more stable and insensitive to particular parameter settings. Here we compare three types of features, Proj: 128d output of the projector head. Pre-feature: 4096d output of the linear layer after backbone. Pool: the 2048d feature from the last average pooling layer. We find that averaging features obtained from images processed by training augmentations improves the result. Therefore we also compare performance in this setting and denote it by 16avg, as 16 augmentations are used per image. Images from both training and test set are augmented. As can be seen from the table, without augmentation, SVM using the projector head feature gives the best accuracy. This is surprising, as most self-supervised learning techniques effectively use Pool features because the performance obtained with Proj features is poor. When averaging is used, a SVM with Pool feature becomes the best performer, however, SVM and kNN with Proj feature also achieve very high accuracy. In Appendix, we further compared Proj and Pool features using SVM in cases where limited labeled data are available. In such cases, Proj feature significantly outperform Pool feature, showing that the TCR objective encourages learning of meaningful feature in the latent space – unlike other algorithms, where the gap between performance of Pool and Proj features can be very large [21].

Table 5.3: Clustering performance comparisons. Clustering: clustering specific methods. SC: subspace clustering methods. See text for details.

Models	**CIFAR-10**			**CIFAR-20**			**STL-10**			**ImageNet-10**			**ImageNet-Dogs**		
	ACC	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**
Clustering															
k-means	0.525	0.589	0.276	0.130	0.084	0.028	0.192	0.125	0.061	0.241	0.119	0.057	0.105	0.055	0.020
Spectral	0.455	0.574	0.256	0.136	0.090	0.022	0.159	0.098	0.048	0.271	0.151	0.076	0.111	0.038	0.013
CC	0.790	0.705	0.637	0.429	0.431	0.266	**0.850**	**0.764**	**0.726**	0.893	0.859	0.822	0.429	0.455	0.474
CRLC	0.799	0.679	0.634	0.425	0.416	0.263	0.818	0.729	0.682						
MoCo Baseline	0.776	0.669	0.608	0.397	0.390	0.242	0.728	0.615	0.524	-	-	-	0.338	0.347	0.197
MiCE	0.835	0.737	0.698	0.440	0.436	0.280	0.752	0.635	0.575	-	-	-	0.439	0.428	0.286
TCC	**0.906**	0.790	0.733	0.491	0.479	0.312	0.814	0.732	0.689	0.897	0.848	0.825	0.546	0.512	0.409
ConCURL	0.846	0.762	0.715	0.479	0.468	0.303	0.749	0.636	0.566	**0.958**	**0.907**	**0.909**	**0.695**	**0.630**	**0.531**
IDFD	0.815	0.711	0.663	0.425	0.426	0.264	0.756	0.643	0.575	0.954	0.898	0.901	0.591	0.546	0.413
SC															
MCR^2-EnSC	0.684	0.630	0.508	0.347	0.362	0.167	0.491	0.446	0.290	-	-	-	-	-	-
MCR^2-ESC	0.653	0.629	0.438	-	-	-	-	-	-	-	-	-	-	-	-
MCR^2-SENet	0.765	0.655	0.573	-	-	-	-	-	-	-	-	-	-	-	-
Ours															
NMCE-Res18	0.830	0.761	0.710	0.437	0.488	0.322	0.725	0.614	0.552	0.906	0.819	0.808	0.398	0.393	0.227
NMCE-Res34	0.891	**0.812**	**0.795**	**0.531**	**0.524**	**0.375**	0.711	0.600	0.530	-	-	-	-	-	-

Clustering Performance

We compare clustering performance of NMCE after all 3 stages of training to other techniques. Due to space limits, we only compare with important baseline methods as well as highest performing deep learning based techniques, rather than being fully comprehensive. For a more complete list of methods, see for example [137]. We follow the convention in the field and use clustering Accuracy (ACC), Normalized Mutual Information (NMI), and Adjusted Rand Score (ARI) as metrics. For definitions, see [178], for example. We list results in Table 5.3, K-means[99] and Spectral clustering[112] results were adopted from [184] and [137], we always selected the best result. For deep learning based techniques, we compare our ResNet-34 results with CC[96], CRLC[33], MiCE[159], TCC[137], ConCURL[29], and IDFD[152]. We did not include [120] or [81] since the former is a post processing step, and the later uses a special network architecture. We also separately compare ResNet-18 result with features learned by MCR^2 (MCR^2-CTRL to be precise, see Appendix D.1) and clustered by EnSC[174], ESC[175] or SENet[184]. They are denoted by MCR^2-EnSC,

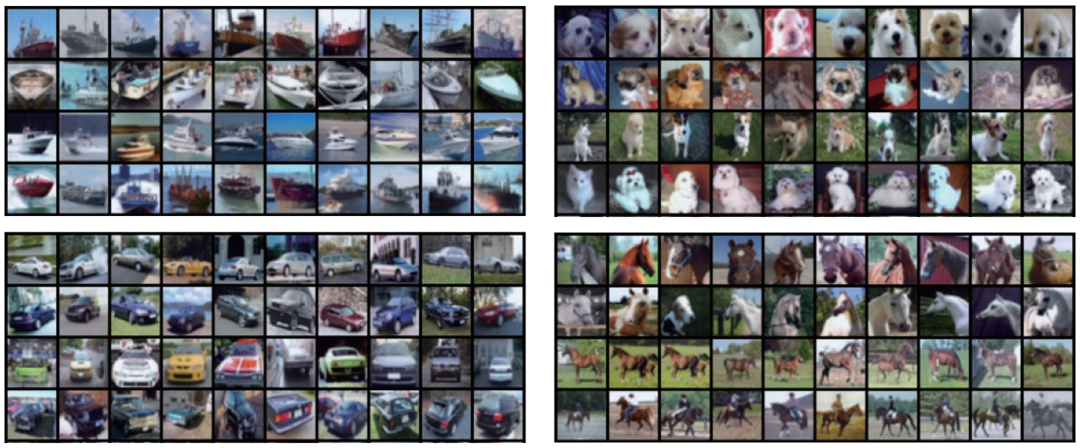


Figure 5.2: Visualization of principal components from subspace representation learned from CIFAR-10. Rows in each panel are training images that has the largest cosine similarities with different principle components of that subspace. Similarity of images within each component is apparent. For a more complete visualization, see Figure D.1.

MCR^2-ESC and MCR^2-SENet, respectively. For MCR^2-EnSC, we adopt the result from [178], since the result is higher and include all 3 datasets, MCR^2-ESC and MCR^2-SENet results are from [184].

Compared with the result from MCR^2, our method achieved a substantial gain in all metrics considered. For example, the accuracy is improved by more than 6% on CIFAR-10, and by 9% on CIFAR-20. This shows that we used data augmentation in a much more effective way than original MCR^2. Our method can also outperform previous techniques which were specifically optimized for clustering, this happens on most CIFAR-10 and CIFAR-20 metrics, for example.

In the experiments above, stage 1 learns pre-features that are approximately subspace-structured, and stage 2 can be seen as performing subspace clustering on the pre-feature output, as MCR2 was originally a subspace clustering objective [98]. For CIFAR and STL-10 experiments, fine tuning the entire network with the NMCE objective (stage 3) improves performance by a small but significant amount, see Appendix D.1). Features learned from stage 1 of the training can also be directly clustered by methods like EnSC, but the results were rather poor for the image dataset. It seems that NMCE is a more robust way to cluster the learned features for the image experiments.

After all 3 training stages, one can visualize the structure of each subspace by performing PCA and displaying the training examples that have the highest cosine similarity to each of the principle components. In Figure 5.2 we show this for 4 out of 10 clusters of CIFAR-10. As can be seen, principle components in each cluster correspond to semantically meaningful sub-clusters. For example, the first row in ”dog” cluster are mostly close-ups of white dogs. This shows that after training with the full NMCE objective, samples are clustered based on content similarity in the latent space, instead of being distributed uniformly on the hyper-sphere, as training with just the TCR objective or other self-supervised learning methods would encourage [168, 37, 180, 7, 187].

5.5 Discussion

In this paper we proposed a general method for manifold clustering and embedding with neural networks. It consists of adding geometric constraint to make the clusters identifiable, then performing clustering and subspace feature learning. In the special case where the constraint is implemented by data-augmentation, we discussed its relationship to self-supervised learning and demonstrated its competitive performance on several important benchmarks.

In the future, it is possible to extend this method beyond data augmentation to other type of constraint functions, or improves its performance using a stronger subspace feature learning algorithm.

Bibliography

[1] Mahdi Abavisani et al. “Deep Subspace Clustering with Data Augmentation.” In: *NeurIPS*. 2020.

[2] M Ehsan Abbasnejad et al. “A generative adversarial density estimator”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10782–10791.

[3] Maryam Abdolali and Nicolas Gillis. “Beyond linear subspace clustering: A comparative study of nonlinear manifold clustering algorithms”. In: *arXiv preprint arXiv:2103.10656* (2021).

[4] Maryam Abdolali and Nicolas Gillis. “Subspace Clustering Using Unsupervised Data Augmentation”. In: *ResearchGate https://www.researchgate.net/publication/355095903* (Oct. 2021).

[5] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. “A learning algorithm for Boltzmann machines”. In: *Cognitive science* 9.1 (1985), pp. 147–169.

[6] Alexei Baevski et al. “wav2vec 2.0: A framework for self-supervised learning of speech representations”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12449–12460.

[7] Adrien Bardes, Jean Ponce, and Yann LeCun. “Vicreg: Variance-invariance-covariance regularization for self-supervised learning”. In: *arXiv preprint arXiv:2105.04906* (2021).

[8] Shane Barratt and Rishi Sharma. “A note on the inception score”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Workshop on Theoretical Foundations and Applications of Deep Generative Models. 2018.

[9] Jens Behrmann et al. “Invertible Residual Networks”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019.

[10] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396.

[11] Guillaume Bellec et al. “Deep Rewiring: Training very sparse deep networks”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.

[12] Michael Betancourt. “A general metric for Riemannian manifold Hamiltonian Monte Carlo”. In: *International Conference on Geometric Science of Information*. Springer. 2013, pp. 327–334.

[13] Michael Betancourt et al. “The geometric foundations of hamiltonian monte carlo”. In: *Bernoulli* 23.4A (2017), pp. 2257–2298.

[14] Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[15] Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. “Accurate and conservative estimates of MRF log-likelihood using reverse annealing”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2015.

[16] Chris Cannella and Vahid Tarokh. “Semi-Empirical Objective Functions for Neural MCMC Proposal Optimization”. In: *arXiv preprint arXiv:2106.02104* (2021).

[17] Mathilde Caron et al. “Deep Clustering for Unsupervised Learning of Visual Features”. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*. Vol. 11218. Lecture Notes in Computer Science. Springer, 2018, pp. 139–156.

[18] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[19] Tong Che et al. “Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling”. In: *arXiv preprint arXiv:2003.06060* (2020).

[20] Tian Qi Chen et al. “Residual Flows for Invertible Generative Modeling”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[21] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[22] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[23] Yubei Chen, Dylan M. Paiton, and Bruno A. Olshausen. "The Sparse Manifold Transform". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 2018, pp. 10534–10545.

[24] Hyunsun Choi, Eric Jang, and Alexander A Alemi. "Waic, but why? generative ensembles for robust anomaly detection". In: *arXiv preprint arXiv:1810.01392* (2018).

[25] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[26] Victor G Turrisi da Costa et al. "Solo-learn: A Library of Self-supervised Methods for Visual Representation Learning". In: *arXiv preprint arXiv:2108.01775* (2021).

[27] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[28] Anirban DasGupta. *Asymptotic theory of statistics and probability*. Springer Science & Business Media, 2008.

[29] Aniket Anand Deshmukh et al. "Representation Learning for Clustering via Building Consensus". In: *arXiv preprint arXiv:2105.01289* (2021).

[30] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in Neural Information Processing Systems* 34 (2021).

[31] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015.

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

[33] Kien Do, Truyen Tran, and Svetha Venkatesh. "Clustering by Maximizing Mutual Information Across Views". In: *arXiv preprint arXiv:2107.11635* (2021).

[34] Harris Drucker and Yann Le Cun. “Double backpropagation increasing generalization performance”. In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN).* 1991.

[35] Yilun Du and Igor Mordatch. “Implicit generation and generalization in energy-based models”. In: *Advances in Neural Information Processing Systems (NeurIPS).* 2019.

[36] Yilun Du et al. “Improved Contrastive Divergence Training of Energy Based Models”. In: *Proceedings of the 38th International Conference on Machine Learning (ICML-21).* 2021.

[37] Aiden Durrant and Georgios Leontidis. “Hyperspherically Regularized Networks for BYOL Improves Feature Uniformity and Separability”. In: *arXiv preprint arXiv:2105.00925* (2021).

[38] David Duvenaud et al. “No MCMC for Me: Amortized Samplers for Fast and Stable Training of Energy-Based Models”. In: 2021. URL: `https://arxiv.org/abs/2010.04230`.

[39] Ehsan Elhamifar and René Vidal. “Sparse manifold clustering and embedding”. In: *Advances in neural information processing systems* 24 (2011), pp. 55–63.

[40] Ehsan Elhamifar and René Vidal. “Sparse subspace clustering: Algorithm, theory, and applications”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.11 (2013), pp. 2765–2781.

[41] Fenglei Fan, Wenxiang Cong, and Ge Wang. “A new type of neurons for machine learning”. In: *International Journal for Numerical Methods in Biomedical Engineering (JNMBE)* (2018).

[42] Sam Foreman, Xiao-Yong Jin, and James C Osborn. “Deep Learning Hamiltonian Monte Carlo”. In: *arXiv preprint arXiv:2105.03418* (2021).

[43] E Paxon Frady and Friedrich T Sommer. “Robust computation with rhythmic spike patterns”. In: *Proceedings of the National Academy of Sciences* 116.36 (2019), pp. 18050–18059.

[44] R Gao et al. “Learning energy-based models by diffusion recovery likelihood”. In: *International Conference on Learning Representations (ICLR 2021).* 2021.

[45] Ruiqi Gao et al. “Flow contrastive estimation of energy-based models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 7518–7528.

[46] Stuart Geman and Donald Geman. “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (1984).

[47] Mark Girolami and Ben Calderhead. “Riemann manifold langevin and hamiltonian monte carlo methods”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2 (2011), pp. 123–214.

[48] Dian Gong, Xuemei Zhao, and Gérard Medioni. “Robust multiple manifolds structure learning”. In: *International Conference on Machine Learning.* PMLR. 2012.

[49] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems (NIPS).* 2014.

[50] Will Grathwohl et al. “Learning the stein discrepancy for training and evaluating energy-based models without sampling”. In: *International Conference on Machine Learning.* PMLR. 2020, pp. 3732–3747.

[51] Will Grathwohl et al. “Your classifier is secretly an energy based model and you should treat it like one”. In: *International Conference on Learning Representations.* 2019.

[52] Jean-Bastien Grill et al. “Bootstrap your own latent: A new approach to self-supervised learning”. In: *arXiv preprint arXiv:2006.07733* (2020).

[53] Minghao Gu, Shiliang Sun, and Yan Liu. “Dynamical Sampling with Langevin Normalization Flows”. In: *Entropy* 21.11 (2019), p. 1096.

[54] Nitzan Guberman. “On complex valued convolutional neural networks”. In: *arXiv preprint arXiv:1602.09046* (2016).

[55] Tuomas Haarnoja et al. “Reinforcement learning with deep energy-based policies”. In: *Proceedings of the International Conference on Machine Learning (ICML).* 2017.

[56] Benjamin D Haeffele, Chong You, and René Vidal. “A critique of self-expressive deep subspace clustering”. In: *arXiv preprint arXiv:2010.03697* (2020).

[57] Jeff Z HaoChen et al. “Provable Guarantees for Self-Supervised Deep Learning with Spectral Contrastive Loss”. In: *arXiv preprint arXiv:2106.04156* (2021).

[58] W.K. Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (1970), pp. 97–109.

[59] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[60] Kaiming He et al. “Identity mappings in deep residual networks”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2016.

[61] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[62] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[63] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.

[64] Geoffrey E Hinton. “A practical guide to training restricted Boltzmann machines”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.

[65] Geoffrey E Hinton. “Products of experts”. In: *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. 1999.

[66] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[67] Marcel Hirt, Michalis Titsias, and Petros Dellaportas. “Entropy-based adaptive Hamiltonian Monte Carlo”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[68] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[69] Matthew Hoffman et al. “Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport”. In: *arXiv preprint arXiv:1903.03704* (2019).

[70] Matthew D Hoffman. “Learning deep latent Gaussian models with Markov chain Monte Carlo”. In: *International conference on machine learning*. 2017, pp. 1510–1519.

[71] Matthew D Hoffman and Andrew Gelman. “The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo.” In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1593–1623.

[72] Aapo Hyvärinen. “Estimation of non-normalized statistical models by score matching”. In: *Journal of Machine Learning Research (JMLR)* (2005).

[73] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[74] Pan Ji, Mathieu Salzmann, and Hongdong Li. “Efficient dense subspace clustering”. In: *IEEE Winter Conference on Applications of Computer Vision.* IEEE. 2014, pp. 461–468.

[75] Pan Ji et al. “Deep subspace clustering networks”. In: *arXiv preprint arXiv:1709.02508* (2017).

[76] Oliver Thomas Johnson. *Information theory and the central limit theorem.* World Scientific, 2004.

[77] Ian T. Jolliffe. *Principal Component Analysis.* Springer, 1986.

[78] Richard Jordan, David Kinderlehrer, and Felix Otto. “The variational formulation of the Fokker–Planck equation”. In: *SIAM Journal on Mathematical Analysis* (1998).

[79] Zhao Kang et al. “Logdet rank minimization with application to subspace clustering”. In: *Computational intelligence and neuroscience* 2015 (2015).

[80] Yan Karklin and Eero P Simoncelli. “Efficient coding of natural images with a population of noisy linear-nonlinear neurons”. In: *Advances in Neural Information Processing Systems (NIPS).* 2011.

[81] Vitaliy Kinakh, Slava Voloshynovskiy, and Olga Taran. “ScatSimCLR: self-supervised contrastive learning with pretext task regularization for small-scale datasets”. In: *2nd Visual Inductive Priors for Data-Efficient Deep Learning Workshop.* 2021.

[82] Diederik P. Kingma and Yann LeCun. “Regularized estimation of image statistics by Score Matching”. In: *Advances in Neural Information Processing Systems (NIPS).* 2010.

[83] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *CoRR* abs/1312.6114 (2014).

[84] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *Advances in Neural Information Processing Systems (NeurIPS).* 2018.

[85] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. “Optimization by simulated annealing”. In: *Science* (1983).

[86] Ivan Kobyzev, Simon Prince, and Marcus A Brubaker. “Normalizing flows: Introduction and ideas”. In: *arXiv preprint arXiv:1908.09257* (2019).

[87] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, Canada, 2009.

[88] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[89] Julius von Kügelgen et al. “Self-Supervised Learning with Data Augmentations Provably Isolates Content from Style”. In: *arXiv preprint arXiv:2106.04619* (2021).

[90] Rithesh Kumar et al. “Maximum Entropy Generators for Energy-Based Models”. In: *arXiv preprint arXiv:1901.08508* (2019).

[91] Rithesh Kumar et al. “Maximum entropy generators for energy-based models”. In: *arXiv preprint arXiv:1901.08508* (2019).

[92] Ian Langmore et al. “A Condition Number for Hamiltonian Monte Carlo”. In: *arXiv preprint arXiv:1905.09813* (2019).

[93] Neil Lawrence. “Probabilistic non-linear principal component analysis with Gaussian process latent variable models”. In: *Journal of Machine Learning Research (JMLR)* (2005).

[94] Yann LeCun et al. “A tutorial on energy-based learning”. In: *Bakir et al. (eds) “Predicting Structured Data”*. MIT Press, 2006.

[95] Daniel Levy, Matt D Hoffman, and Jascha Sohl-Dickstein. “Generalizing Hamiltonian Monte Carlo with Neural Networks”. In: *International Conference on Learning Representations*. 2018.

[96] Yunfan Li et al. “Contrastive clustering”. In: *2021 AAAI Conference on Artificial Intelligence (AAAI)*. 2021.

[97] Ziwei Liu et al. “Deep learning face attributes in the wild”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015.

[98] Yi Ma et al. “Segmentation of multivariate mixed data via lossy data coding and compression”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.9 (2007), pp. 1546–1562.

[99] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1.14. Oakland, CA, USA. 1967, pp. 281–297.

[100] Youssef Marzouk et al. "An introduction to sampling via measure transport". In: *arXiv preprint arXiv:1602.05023* (2016).

[101] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.

[102] Toru Nakashika, Shinji Takaki, and Junichi Yamagishi. "Complex-Valued Restricted Boltzmann Machine for Direct Learning of Frequency Spectra." In: *INTERSPEECH*. 2017, pp. 4021–4025.

[103] Eric T. Nalisnick et al. "Do Deep Generative Models Know What They Don't Know?" In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2019.

[104] Radford M Neal. "Annealed importance sampling". In: *Statistics and Computing* (2001).

[105] Radford M Neal. "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* (2011).

[106] Radford M Neal et al. "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[107] Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, Ontario, Canada, 1993.

[108] Radford M Neal. "Slice sampling". In: *Annals of statistics* (2003), pp. 705–741.

[109] Kirill Neklyudov et al. "Metropolis-hastings view on variational inference and adversarial training". In: *arXiv preprint arXiv:1810.07151* (2018).

[110] SA Nene, SK Nayar, and H Murase. "Columbia object image library (coil-20)". In: *Technical Report* 005-96 (1996).

[111] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. "Columbia object image library (coil-100)". In: *Technical Report* 006-96 (1996).

[112] Andrew Y Ng, Michael I Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems*. 2002, pp. 849–856.

[113] Jiquan Ngiam et al. "Learning deep energy models". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2011.

[114] Erik Nijkamp et al. "Learning Energy-based Model with Flow-based Backbone by Neural Transport MCMC". In: *arXiv preprint arXiv:2006.06897* (2020).

[115] Erik Nijkamp et al. “On the Anatomy of MCMC-based Maximum Likelihood Learning of Energy-Based Models”. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*. 2019.

[116] Frank Noé et al. “Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning”. In: *Science* 365.6457 (2019), eaaw1147.

[117] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel Recurrent Neural Networks”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2016.

[118] Georg Ostrovski, Will Dabney, and Rémi Munos. “Autoregressive Quantile Networks for Generative Modeling”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2018.

[119] George Papamakarios et al. “Normalizing flows for probabilistic modeling and inference”. In: *arXiv preprint arXiv:1912.02762* (2019).

[120] Sungwon Park et al. “Improving Unsupervised Image Clustering With Robust Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12278–12287.

[121] Emanuel Parzen. “On Estimation of a Probability Density Function and Mode”. In: *The Annals of Mathematical Statistics* (1962).

[122] Cristian Pasarica and Andrew Gelman. “Adaptively scaling the Metropolis algorithm using expected squared jumped distance”. In: *Statistica Sinica* (2010), pp. 343–364.

[123] Vishal M Patel and René Vidal. “Kernel sparse subspace clustering”. In: *2014 ieee international conference on image processing (icip)*. IEEE. 2014, pp. 2849–2853.

[124] Ben Poole et al. “On variational bounds of mutual information”. In: *arXiv preprint arXiv:1905.06922* (2019).

[125] Călin-Adrian Popa. “Complex-Valued Deep Boltzmann Machines”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[126] Tijana Radivojević and Elena Akhmatskaya. “Modified Hamiltonian Monte Carlo for Bayesian Inference”. In: *Statistics and Computing* 30.2 (2020), pp. 377–404.

[127] Aditya Ramesh et al. "Zero-Shot Text-to-Image Generation". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 8821–8831. URL: `https://proceedings.mlr.press/v139/ramesh21a.html`.

[128] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* (2000).

[129] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500 (2000), pp. 2323–2326.

[130] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited, 2016.

[131] Ruslan Salakhutdinov and Iain Murray. "On the quantitative analysis of deep belief networks". In: *Proceedings of the International Conference on Machine learning (ICML)*. 2008.

[132] Tim Salimans, Diederik Kingma, and Max Welling. "Markov chain monte carlo and variational inference: Bridging the gap". In: *International Conference on Machine Learning*. 2015, pp. 1218–1226.

[133] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[134] Saeed Saremi and Aapo Hyvärinen. "Neural Empirical Bayes". In: *Journal of Machine Learning Research (JMLR)* (2019).

[135] Saeed Saremi et al. "Deep energy estimator networks". In: *arXiv preprint arXiv:1805.08306* (2018).

[136] Ivan W Selesnick, Richard G Baraniuk, and Nick C Kingsbury. "The dual-tree complex wavelet transform". In: *IEEE signal processing magazine* 22.6 (2005), pp. 123–151.

[137] Yuming Shen et al. "You Never Cluster Alone". In: *arXiv preprint arXiv:2106.01908* (2021).

[138] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory.* Tech. rep. Colorado Univ at Boulder Dept of Computer Science, 1986.

[139] Jascha Sohl-Dickstein, Mayur Mudigonda, and Michael R DeWeese. "Hamiltonian Monte Carlo without detailed balance". In: *arXiv preprint arXiv:1409.5191* (2014).

[140] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2256–2265. URL: `https://proceedings.mlr.press/v37/sohl-dickstein15.html`.

[141] Jiaming Song and Stefano Ermon. “Understanding the limitations of variational mutual information estimators”. In: *arXiv preprint arXiv:1910.06222* (2019).

[142] Jiaming Song, Shengjia Zhao, and Stefano Ermon. “A-nice-mc: Adversarial training for mcmc”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5140–5150.

[143] Yang Song and Stefano Ermon. “Generative Modeling by Estimating Gradients of the Data Distribution”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[144] Yang Song et al. “Maximum likelihood training of score-based diffusion models”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[145] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*. 2020.

[146] Yang Song et al. “Sliced Score Matching: A Scalable Approach to Density and Score Estimation”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*. 2019.

[147] R. Souvenir and R. Pless. “Manifold Clustering”. In: *International Conference on Computer Vision (ICCV)*. 2005, pp. 648–653.

[148] Span Spanbauer, Cameron Freer, and Vikash Mansinghka. “Deep Involutive Generative Models for Neural MCMC”. In: *arXiv preprint arXiv:2006.15167* (2020).

[149] Xiukun Sun et al. “Self-supervised deep multi-view subspace clustering”. In: *Asian Conference on Machine Learning*. PMLR. 2019, pp. 1001–1016.

[150] Yutaka Takeda et al. “Boltzmann sampling for an XY model using a non-degenerate optical parametric oscillator network”. In: *Quantum Science and Technology* 3.1 (2017), p. 014004.

[151] Terence Tao. *Topics in random matrix theory*. Vol. 132. American Mathematical Soc., 2012.

[152] Yaling Tao, Kentaro Takagi, and Kouta Nakata. “Clustering-friendly Representation Learning via Instance Discrimination and Feature Decorrelation”. In: *International Conference on Learning Representations*. 2021.

[153] Joshua B Tenenbaum, Vin De Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* (2000).

[154] Achille Thin et al. “Metropolized Flow: from Invertible Flow to MCMC”. In: *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models* (2020).

[155] Tijmen Tieleman. “Training restricted Boltzmann machines using approximations to the likelihood gradient”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1064–1071.

[156] Michalis Titsias and Petros Dellaportas. “Gradient-based Adaptive Markov Chain Monte Carlo”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 15730–15739.

[157] Jakub M. Tomczak. *Deep Generative Modeling*. Vol. 1. Springer, Cham, 2022.

[158] Chiheb Trabelsi et al. “Deep Complex Networks”. In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=H1T2hmZAb`.

[159] Tsung Wei Tsai, Chongxuan Li, and Jun Zhu. “MiCE: Mixture of Contrastive Experts for Unsupervised Image Clustering”. In: *International Conference on Learning Representations*. 2020.

[160] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv e-prints* (2018), arXiv–1807.

[161] Greg Ver Steeg and Aram Galstyan. “Hamiltonian Dynamics with Non-Newtonian Momentum for Rapid Sampling”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[162] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*. Vol. 47. Cambridge University Press, 2018.

[163] René Vidal, Yi Ma, and S Shankar Sastry. *Generalized principal component analysis*. Vol. 5. Springer, 2016.

[164] Pascal Vincent. “A connection between score matching and denoising autoencoders”. In: *Neural computation* (2011).

[165] Pascal Vincent et al. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. In: *Journal of Machine Learning Research (JMLR)* (2010).

[166] Martin J Wainwright and Eero P Simoncelli. "Scale mixtures of Gaussians and the statistics of natural images". In: *Advances in Neural Information Processing Systems (NIPS)*. 2000.

[167] Tianshi Wang, Leon Wu, and Jaijeet Roychowdhury. "New computational results and hardware prototypes for oscillator-based Ising machines". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–2.

[168] Tongzhou Wang and Phillip Isola. "Understanding contrastive representation learning through alignment and uniformity on the hypersphere". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9929–9939.

[169] Max Welling and Geoffrey E Hinton. "A new learning algorithm for mean field Boltzmann machines". In: *International Conference on Artificial Neural Networks*. Springer. 2002, pp. 351–357.

[170] Max Welling and Yee W Teh. "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2011.

[171] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. "Tackling the Generative Learning Trilemma with Denoising Diffusion GANs". In: *arXiv preprint arXiv:2112.07804* (2021).

[172] Zhisheng Xiao et al. "VAEBM: A Symbiosis between Variational Autoencoders and Energy-based Models". In: *International Conference on Learning Representations*. 2020.

[173] Jianwen Xie et al. "A theory of generative convnet". In: *International Conference on Machine Learning*. 2016, pp. 2635–2644.

[174] Chong You et al. "Oracle based active set algorithm for scalable elastic net subspace clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3928–3937.

[175] Chong You et al. "Scalable exemplar-based subspace clustering on class-imbalanced data". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 67–83.

[176] Yang You, Igor Gitman, and Boris Ginsburg. "Large batch training of convolutional networks". In: *arXiv preprint arXiv:1708.03888* (2017).

[177] Lantao Yu et al. "Training Deep Energy-Based Models with f-Divergence Minimization". In: *International Conference on Machine Learning*. 2020.

[178] Yaodong Yu et al. *Learning Diverse and Discriminative Representations via the Principle of Maximal Coding Rate Reduction*. 2020. arXiv: `2006.08558 [cs.LG]`.

[179] Alan L Yuille. "The convergence of contrastive divergences". In: *Advances in neural information processing systems* 17 (2004).

[180] Jure Zbontar et al. "Barlow twins: Self-supervised learning via redundancy reduction". In: *arXiv preprint arXiv:2103.03230* (2021).

[181] Richard S Zemel, Christopher KI Williams, and Michael C Mozer. "Lending direction to neural networks". In: *Neural Networks* 8.4 (1995), pp. 503–512.

[182] Shuangfei Zhai et al. "Deep Structured Energy Based Models for Anomaly Detection". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2016.

[183] Junjian Zhang et al. "Self-supervised convolutional subspace clustering network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5473–5482.

[184] Shangzhi Zhang et al. "Learning a Self-Expressive Network for Subspace Clustering". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12393–12403.

[185] Yichuan Zhang, José Miguel Hernández-Lobato, and Zoubin Ghahramani. "Ergodic measure preserving flows". In: *arXiv preprint arXiv:1805.10377* 3.4 (2018).

[186] Pan Zhou, Yunqing Hou, and Jiashi Feng. "Deep adversarial subspace clustering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1596–1604.

[187] Roland S Zimmermann et al. "Contrastive Learning Inverts the Data Generating Process". In: *arXiv preprint arXiv:2102.08850* (2021).

Appendix A

Supplementary material for CAP-BM

A.1 Sampling rules derivation

The amplitude probability distribution $P(|z_j| = 1|\, \boldsymbol{z}_{!j}) =: P_j$ is computed as marginals of the Boltzmann distribution induced by energy function (2.2):

$$P_j = \int_{2\pi} d\theta\, p(|z_j| = 1, \theta_j = \theta|\, \boldsymbol{z}_{!j}) = \frac{\int_{2\pi} d\theta\, e^{-E_j(|z_j|=1,\theta_j=\theta)-E_{!j}}}{\boldsymbol{Z}} \tag{A.1}$$

Here E_j and $E_{!j}$ denote parts of the energy function that depend and do not depend on z_j, respectively. To avoid dealing with the intractable partition function $\boldsymbol{Z}$, one can calculate $P_j/1 - P_j$ then solve for P_j:

$$1 - P_j = \frac{\int_{2\pi} d\theta\, e^{-E_j(|z_j|=0,\theta_j=\theta)-E_{!j}}}{\boldsymbol{Z}} \tag{A.2}$$

Divide (A.1) and (A.2) and insert the following expression easily derived from (2.2): $E_j = -Re(z_j^* u_j) - |z_j|\mu_j + \epsilon_j$ where $u_j = \sum_{k\neq j} w_{jk} z_k = a_j e^{i\alpha_j}$ and $\mu_j = \sum_{k\neq j} J_{jk}|z_k|$ is the complex and real-valued postsynaptic sums, respectively. Putting it all together yields:

$$\frac{P_j}{1-P_j} = \frac{\int_{2\pi} d\theta\, e^{-E_j(|z_j|=1,\theta_j=\theta)-E_{!j}}}{\int_{2\pi} d\theta\, e^{-E_{!j}}} = \frac{1}{2\pi}\int_{2\pi} d\theta\, e^{a_j\cos(\alpha_j-\theta)+\mu_j-\epsilon_j} = e^{\mu_j-\epsilon_j}\mathrm{I}_0(a_j)$$

where I_0 denotes the zeroth order modified Bessel function of the first kind. Solving for P_j yields:

$$P(|z_j| = 1|\, \boldsymbol{z}_{!j}) = \frac{1}{1 + (e^{\mu_j-\epsilon_j}\mathrm{I}_0(a_j))^{-1}} \tag{A.3}$$

We note that result (A.3) can also serve as the natural amplitude activation function for a continuous-valued complex neural network that also has amplitude-amplitude coupling. See Figure A.1 for a plot illustrating some of its properties.

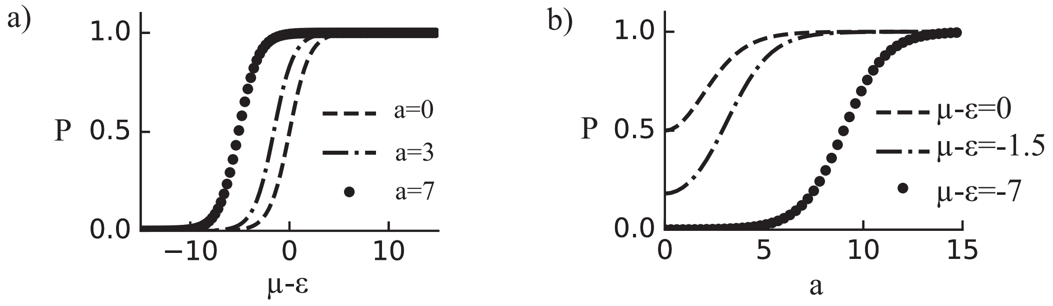


Figure A.1: The amplitude activation function P_j. a) P_j is a sigmoid function with respect to the real-valued postsynaptic sum with the saturation levels at $P_j = 0$ and $P_j = 1$. A horizontal offset is determined by a. b) TP_j as a function of the modulus of the complex postsynaptic sum is also approximately sigmoid shaped except near $a = 0$ where it always have slop 0. For large negative values of $\mu - \epsilon$, saturation levels are at $P_j = 0$ and $P_j = 1$. However, when $\mu - \epsilon = 0$, the value of P_j near $a = 0$ rises to a value of 0.5.

Finally, for obtaining the phase distribution $p(\theta_j|\,|z_j| = 1, \boldsymbol{z}_{!j})$, we use Bayes rule:

$$\begin{aligned} p(\theta_j|\,|z_j| = 1, \boldsymbol{z}_{!j}) = \frac{p(|z_j| = 1, \theta_j\,|\boldsymbol{z}_{!j})}{P(|z_j| = 1\,|\boldsymbol{z}_{!j})} &= \frac{e^{-E_j(|z_j|=1,\theta_j)-E_{!j}}}{\int_{2\pi} d\theta\, e^{-E_j(|z_j|=1,\theta_j=\theta)-E_{!j}}} \\ &= \frac{1}{2\pi \mathrm{I}_0(a_j)} e^{a_j \cos(\alpha_j - \theta_j)} \end{aligned} \tag{A.4}$$

which is a von Mises distribution, the circular analog of Gaussian with its mode at $\theta_j = \alpha_j$.

A.2 Learning rules derivation

We start with KL-divergence between model and data distribution, which is equivalent to maximum likelihood:

$$G = p(\boldsymbol{v}) \| p_m^\infty(\boldsymbol{v}) = \sum_{2^N} \int_{(2\pi)^N} d\theta^N\, p(\boldsymbol{v}) \log(\frac{p(\boldsymbol{v})}{p_m^\infty(\boldsymbol{v})}) = -H(p(\boldsymbol{v})) - \langle \log(p_m^\infty(\boldsymbol{v})) \rangle_{p(\boldsymbol{v})}$$

where $p(\boldsymbol{v})$ and $p_m^\infty(\boldsymbol{v})$ denote data and model distribution of visible units. The sum over 2^N denotes all 2^N possibilities of the modulus of visible units and the integral over $(2\pi)^N$ is the integration over the phase angles of all visible units.

Writing the complex weights in polar coordinates $w_{jk} = b_{jk}e^{i\theta_{jk}}$, we compute the derivative of G w.r.t. b_{jk}:

$$\frac{\partial G}{\partial b_{jk}} = -\left\langle \frac{\partial \log(p_m^\infty(\boldsymbol{v}))}{b_{jk}} \right\rangle_{p(\boldsymbol{v})}$$
$$= -\left\langle \frac{\sum_{2^M}\int d\theta^M\, e^{-E(\boldsymbol{v},\boldsymbol{h})}(-\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial b_{jk}})}{\sum_{2^M}\int d\theta^M\, e^{-E(\boldsymbol{v},\boldsymbol{h})}} - \frac{\sum_{2^{N+M}}\int d\theta^{N+M}\, e^{-E(\boldsymbol{v},\boldsymbol{h})}(-\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial b_{jk}})}{\sum_{2^{N+M}}\int d\theta^{N+M}\, e^{-E(\boldsymbol{v},\boldsymbol{h})}} \right\rangle_{p(\boldsymbol{v})}$$

The sum and integral over M variables denote the average over hidden unit states, each term inside becomes an average, either over the marginal distribution of the hidden variables given the visibles, or an average over the free model distribution:

$$\frac{\partial G}{\partial b_{jk}} = \left\langle \left\langle \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial b_{jk}} \right\rangle_{p(\boldsymbol{h}|\boldsymbol{v})} - \left\langle \frac{\partial E(\boldsymbol{v}',\boldsymbol{h}')}{\partial b_{jk}} \right\rangle_{p(\boldsymbol{v}',\boldsymbol{h}')} \right\rangle_{p(\boldsymbol{v})}$$
$$= \langle |z_j||z_k|\cos(\theta_{jk}+\theta_k-\theta_j)\rangle_{\text{sample}} - \langle |z_j||z_k|\cos(\theta_{jk}+\theta_k-\theta_j)\rangle_{\text{model}}$$

Similarly, the gradients w.r.t θ_{jk}, J_{jk} and ϵ_j are:

$$\frac{\partial G}{\partial \theta_{jk}} = -\langle |z_j||z_k|b_{jk}\sin(\theta_{jk}+\theta_k-\theta_j)\rangle_{\text{sample}} + \langle |z_j||z_k|b_{jk}\sin(\theta_{jk}+\theta_k-\theta_j)\rangle_{\text{model}}$$

$$\frac{\partial G}{\partial J_{jk}} = \langle |z_j||z_k|\rangle_{\text{sample}} - \langle |z_j||z_k|\rangle_{\text{model}} \qquad \frac{\partial G}{\partial \epsilon_j} = -\langle |z_j|\rangle_{\text{sample}} + \langle |z_j|\rangle_{\text{model}}$$

A.3 Experimental Details

General Setup and Toy experiment

Energy function of CAP-RBM can be written in the obvious way. In the following, $\boldsymbol{v}$ and $\boldsymbol{h}$ denote the complex visible and hidden units, and $\boldsymbol{a}$ and $\boldsymbol{b}$ the bias vectors for visible and hidden units. The energy function is then:

$$E = -\boldsymbol{v}^\dagger \boldsymbol{W}\boldsymbol{h} - |\boldsymbol{v}|^T \boldsymbol{J}|\boldsymbol{h}| + \boldsymbol{a}^T|\boldsymbol{v}| + \boldsymbol{b}^T|\boldsymbol{h}| \qquad \text{(A.5)}$$

Naively written in this way, this function is not necessarily real, but various simple arguments can show that we can just take the real part without causing any issue.

As in real-valued RBM, alternating parallel Gibbs sampling can be applied to hidden and visible units to sample from the model distribution. Rate, instead of sample, is used to generate an output from the CAP-RBM, for example, to compute weight updates during training, or to display visible unit activity. Rate is defined as the expected complex activity or expected modulus given fixed input to that unit. Do note that, in general, the expected modulus of a complex unit is not equal to the modulus of the expected complex activity, a slightly subtle point.

We trained our CAP-BM using 1 step Contrastive divergence (CD-1), or that followed by Persistent Contrastive divergence (PCD). The procedure of applying those methods are exactly the same as in real-valued RBMs. We observe that those methods behave as expected: CD-1 only explores the state space in vicinity to data, and forms relatively stable representations quickly. PCD learning is slower but it produces a higher quality model [64]. In our case it is able to produce a generative model for MNIST digits in CWT representation.

We first investigate learning in the CAP-RBM on an artificial dataset of random bars with noisy phase structure. Each data sample is a 24X24 complex image, each has random numbers (2-4 each direction) of horizontal and vertical 2 pixels wide bars consist of complex numbers of modulus 1. A sinusoidal phase pattern with random overall phase offset is assigned to each bar. Additional phase perturbation is added to each pixel in a bar, sampled uniformly from $(0, 0.6)$. We compare a full CAP-RBM and a CAP-RBM without $\boldsymbol{J}$ couplings by their ability to simultaneously learn the bar-shaped amplitude pattern and the sinusoidal phase pattern. We trained both models on 40000 training examples using 10 epochs of (CD-1). The phase on bars from full CAP-RBM model appears smooth because rate, instead of samples, are shown.

MNIST experiment

A complex wavelet transform (CWT) was used to produce a complex representation of the original MNIST image. The CWT employs localized and oriented band pass filters with 6 orientation angles. Roughly speaking, the modulus of the resulting complex coefficients represents local power of a particular spatial frequency at different orientations, the phase represents its spatial phase value [136]. We used slightly modified version of CWT (dtcwt library 0.12.0, circularly symmetric filters). The CWT were modified so that phases of filters progress mostly in the same direction when image is gradually translated. This is achieved by using complex conjugate of two of the directional filter coefficient outputs from the software package. Then average is taken over all maximum magnitudes for each frequency band across all images and the result is used to set the amplitude for each frequency band during reconstruction.

The described modified CWT was used to transform all 60000 MNIST images. Complex coefficients were normalized by the maximum modulus in its frequency band and thresholded at a cut-off modulus of 0.15 before normalizing to modulus 1. Such thresholding is not uncommon in the CWT literature [136], and it is necessary here because coefficients with small modulus contribute little to the reconstruction but add noise to the input of other units, which deteriorates learning considerably.

Despite the thresholding of CWT coefficients, the reconstruction quality remains excellent because most of the information about digit shape is stored in phase relations between complex coefficients (Fig. 2.1 b) second row).

MNIST digit images have a resolution of $28 \times 28 = 784$ pixels. Each band pass filter in the CWT downsamples the image by a factor of 2 so there are total of 5 frequency bands. Each band has 6 directional filters. Thus, after a full DWT transform, each image is represented by a total of $(14\times14+7\times7+4\times4+2\times2+1)\times6 = 1596$ complex numbers. In our experiments, only two bands are used for learning, resulting in 7×7 and $4 \times 4 = 390$ coefficients. The highest frequency band is suppressed to limit the number of input parameters into our model, and because the high-frequency structure is relatively unimportant for expressing the relevant features of MNIST digits. Those coefficients are set to 0 during reconstruction. The low frequency bands are also suppressed in the learning because they only represent an amplitude envelope over all digits and contain little digit-specific information – they are set to their global average during reconstruction.

Since CAP-RBM only learns relative phase structure, visible unit activities sometimes do not have the correct global phase to yield a reasonable image reconstruction. This usually happens after large numbers of free Gibbs sampling. In cases this happened, a global phase offset was manually added based on the similarity of resulting reconstruction to hand written character.

To produce a generative model of MNIST in CWT domain, we use a CAP-RBM with 400 hidden units and perform 10 epochs of CD1 training as initialization. Subsequently, 100 epochs of PCD training were performed without weight decay, followed by another 100 epochs of PCD training with weight decay. All experiments were implemented in numpy.

Appendix B

Supplementary material for MDSM

B.1 MDSM Objective

In this section, we provide a formal discussion of the MDSM objective and suggest it as an improved score matching formulation in high-dimensional space.

[164] illustrated the connection between the model score $-\nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}}; \theta)$ with the score of Parzen window density estimator $\nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}}))$. Specifically, the objective is Equation 3.1 which we restate here:

$$L_{SM}(\theta) = \mathbb{E}_{p_{\sigma 0}(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}}; \theta) \parallel^2 \tag{B.1}$$

Our key observation is: in high-dimensional space, due the concentration of measure, the expectation w.r.t. $p_{\sigma 0}(\tilde{\boldsymbol{x}})$ over weighs a thin shell at roughly distance $\sqrt{d}\sigma$ to the empirical distribution $p(x)$. Though in theory this is not a problem, in practice this leads to results that the score are only well matched on this shell. Based on this observation, we suggest to replace the expectation w.r.t. $p_{\sigma 0}(\tilde{\boldsymbol{x}})$ with a distribution $p_{\sigma'}(\tilde{\boldsymbol{x}})$ that has the same support as $p_{\sigma 0}(\tilde{\boldsymbol{x}})$ but can avoid the measure concentration problem. We call this *multiscale score matching* and the objective is the following:

$$L_{MSM}(\theta) = \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}}; \theta) \parallel^2 \tag{B.2}$$

Proposition 1 $L_{MSM}(\theta) = 0 \iff L_{SM}(\theta) = 0 \iff \theta = \theta^*$.

Proof 1 *Given that $p_M(\tilde{\boldsymbol{x}})$ and $p_{\sigma 0}(\tilde{\boldsymbol{x}})$ has the same support, it's clear that $L_{MSM} = 0$ would be equivalent to $L_{SM} = 0$. Due to the proof of the Theorem 2 in [72], we have $L_{SM}(\theta) \iff \theta = \theta^*$. Thus, $L_{MSM}(\theta) = 0 \iff \theta = \theta^*$.*

Proposition 2 *$L_{MSM}(\theta) \smile L_{MDSM^*} = \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})q_{\sigma 0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2$.*

Proof 2 *We follow the same procedure as in [164] to prove this result.*

$$\begin{aligned}
J_{MSM}(\theta) &= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})) + \nabla_{\tilde{x}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 \\
&= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{x}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 + 2S(\theta) + C
\end{aligned}$$

$$\begin{aligned}
S(\theta) &= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \langle \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \\
&= \int_{\tilde{\boldsymbol{x}}} p_M(\tilde{\boldsymbol{x}}) \langle \nabla_{\tilde{\boldsymbol{x}}} \log(p_{\sigma_0}(\tilde{\boldsymbol{x}})), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} p_M(\tilde{\boldsymbol{x}}) \langle \frac{\nabla_{\tilde{\boldsymbol{x}}} p_{\sigma_0}(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})}, \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} \langle \nabla_{\tilde{\boldsymbol{x}}} p_{\sigma_0}(\tilde{\boldsymbol{x}}), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} \langle \nabla_{\tilde{\boldsymbol{x}}} \int_{\boldsymbol{x}} p(\boldsymbol{x}) q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) d\boldsymbol{x}, \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} \langle \int_{\boldsymbol{x}} p(\boldsymbol{x}) \nabla_{\tilde{\boldsymbol{x}}} q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) d\boldsymbol{x}, \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} \langle \int_{\boldsymbol{x}} p(\boldsymbol{x}) q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \nabla_{\tilde{\boldsymbol{x}}} \log q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) d\boldsymbol{x}, \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} \\
&= \int_{\tilde{\boldsymbol{x}}} \int_{\boldsymbol{x}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} p(\boldsymbol{x}) q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \langle \nabla_{\tilde{\boldsymbol{x}}} \log q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} d\boldsymbol{x} \\
&= \int_{\tilde{\boldsymbol{x}}} \int_{\boldsymbol{x}} \frac{p_M(\tilde{\boldsymbol{x}})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})} p_{\sigma_0}(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \langle \nabla_{\tilde{\boldsymbol{x}}} \log q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} d\boldsymbol{x} \\
&= \int_{\tilde{\boldsymbol{x}}} \int_{\boldsymbol{x}} p_M(\tilde{\boldsymbol{x}}) q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) \langle \nabla_{\tilde{\boldsymbol{x}}} \log q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle \ d\tilde{\boldsymbol{x}} d\boldsymbol{x}
\end{aligned}$$

Thus we have:

$$\begin{aligned}
L_{MSM}(\theta) &= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 + 2S(\theta) + C \\
&= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}}) q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 + 2\mathbb{E}_{p_M(\tilde{\boldsymbol{x}}) q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})} \langle \nabla_{\tilde{\boldsymbol{x}}} \log q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x}), \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta)\rangle + C \\
&= \mathbb{E}_{p_M(\tilde{\boldsymbol{x}}) q_{\sigma 0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})} \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 + C'
\end{aligned}$$

So $L_{MSM}(\theta) \smile L_{MDSM^*}$.

The above analysis applies to any noise distribution, not limited to Gaussian. but L_{MDSM^*} has a reversed expectation form that is not easy to work with. To proceed further we study the case where $q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ is Gaussian and choose $q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ as a Gaussian scale mixture [166] and $p_M(\tilde{\boldsymbol{x}}) = \int q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})dx$. By Proposition 1 and Proposition 2, we have the following form to optimize:

$$\begin{aligned}
L_{MDSM^*}(\theta) &= \int_{\tilde{\boldsymbol{x}}}\int_{\boldsymbol{x}} p_M(\tilde{\boldsymbol{x}})q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 d\tilde{\boldsymbol{x}}d\boldsymbol{x} \\
&= \int_{\tilde{\boldsymbol{x}}}\int_{\boldsymbol{x}} \frac{q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})}{q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}})} p_M(\tilde{\boldsymbol{x}})q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}}) \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 d\tilde{\boldsymbol{x}}d\boldsymbol{x} \\
&= \int_{\tilde{\boldsymbol{x}}}\int_{\boldsymbol{x}} \frac{q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})}{q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}})} p_M(\boldsymbol{x},\tilde{\boldsymbol{x}}) \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 d\tilde{\boldsymbol{x}}d\boldsymbol{x} \\
&= \int_{\tilde{\boldsymbol{x}}}\int_{\boldsymbol{x}} \frac{q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})}{q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}})} q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x}) \parallel \nabla_{\tilde{\boldsymbol{x}}} \log(q_{\sigma 0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})) + \nabla_{\tilde{\boldsymbol{x}}} E(\tilde{\boldsymbol{x}};\theta) \parallel^2 d\tilde{\boldsymbol{x}}d\boldsymbol{x} \qquad (*) \\
&\approx L_{MDSM}(\theta)
\end{aligned}$$

To minimize Equation (*), we can use the following importance sampling procedure [130]: we can sample from the empirical distribution $p(\boldsymbol{x})$, then sample the Gaussian scale mixture $q_M(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ and finally weight the sample by $\frac{q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})}{q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}})}$. We expect the ratio to be close to 1 for the following reasons: Using Bayes rule, $q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = \frac{p(\boldsymbol{x})q_{\sigma_0}(\tilde{\boldsymbol{x}}|\boldsymbol{x})}{p_{\sigma_0}(\tilde{\boldsymbol{x}})}$ we can see that $q_{\sigma_0}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$ only has support on discret data points $\boldsymbol{x}$, same thing holds for $q_M(\boldsymbol{x}|\tilde{\boldsymbol{x}})$. because in $\tilde{\boldsymbol{x}}$ is generated by adding Gaussian noise to real data sample, both estimators should give results highly concentrated on the original sample point $\boldsymbol{x}$. Therefore, in practice, we ignore the weighting factor and use Equation 3.6. Improving upon this approximation is left for future work.

B.2 Problem with Single Noise Denoising Score Matching

To compare with previous method, we trained energy-based model with denoising score matching using one noise level on MNIST, initialized the sampling with Gaussian noise of the same level, and sampled with Langevin dynamics at $T = 1$ for

1000 steps and perform one denoise jump to recover the model's best estimate of the clean sample, see Figure B.1. We used the same 12-layer ResNet as other MNIST experiments. Models were trained for 100000 steps before sampling.

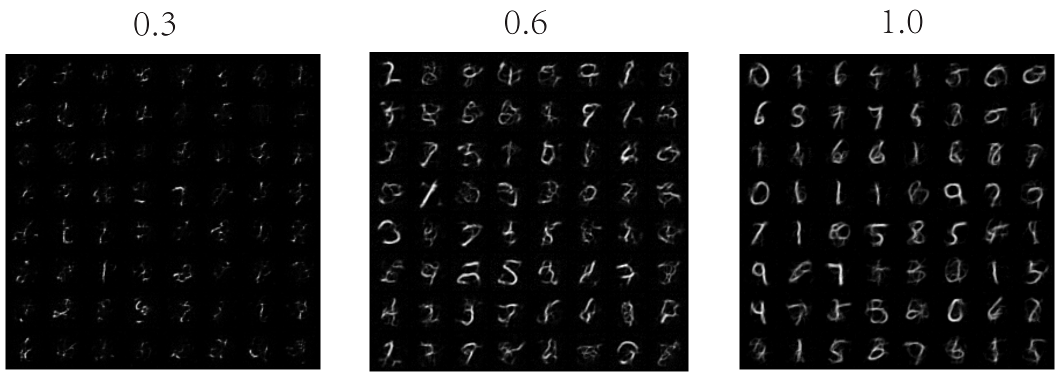


Figure B.1: Denoised samples from energy-based model trained with denoising score matching with single magnitude Gaussian noise on MNIST. Noise magnitude used in training is shown above samples.

B.3 Overfitting Test

We demonstrate that the model does not simply memorize training examples by comparing model samples with their nearest neighbors in the training set. We use Fashion MNIST for this demonstration because overfitting can occur there easier than on more complicated datasets, see Figure B.2.

B.4 Details on Training and Sampling

We used a custom designed ResNet architecture for all experiments. For MNIST and Fashion-MNIST we used a 12-layer ResNet with 64 filters on first layer, while for CelebA and CIFAR dataset we used a 18-layer ResNet with 128 filters on the first layer. All network used the ELU activation function. We did not use any normalization in the ResBlocks and the filer number is doubled at each downsampling block. Details about the structure of our networks used can be found in our code release. All mentioned models can be trained on 2 GPUs within 2 days.

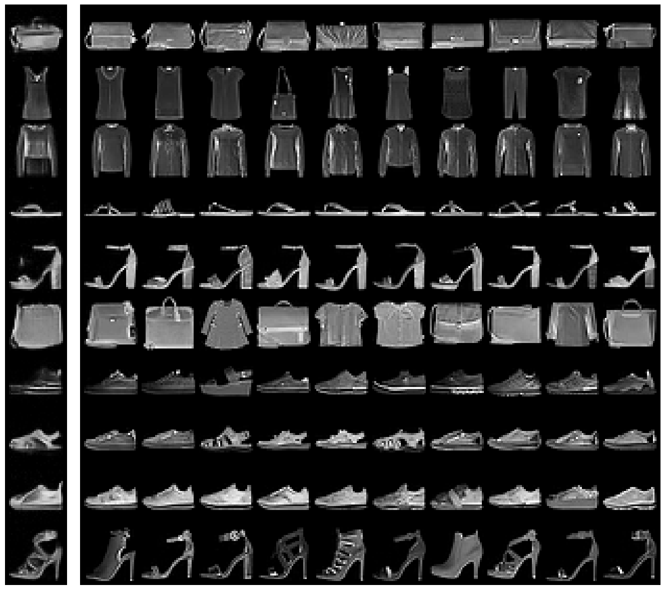


Figure B.2: Samples from energy-based model trained on Fashion MNIST (Left column) next to 10 (L2) nearest neighbors in the training set.

Since the gradient of our energy model scales linearly with the noise, we expected our energy function to scale quadratically with noise magnitude. Therefore, we modified the standard energy-based network output layer to take a flexible quadratic form [41]:

$$E_{out} = (\sum_i a_i h_i + b_1)(\sum_i c_i h_i + b_2) + \sum_i d_i h_i^2 + b_3 \tag{B.3}$$

where a_i, c_i, d_i and b_1, b_2, b_3 are learnable parameters, and h_i is the (flattened) output of last residual block. We found this modification to significantly improve performance compared to using a simple linear last layer.

For CIFAR and CelebA results we trained for 300k weight updates, saving a checkpoint every 5000 updates. We then took 1000 samples from each saved networks and used the network with the lowest FID score. For MNIST and Fashion MNIST we simply trained for 100k updates and used the last checkpoint. During training we pad MNIST and Fashion MNIST to 32*32 for convenience and randomly flipped CelebA images. No other modification was performed. We only constrained the gradient of the energy function, the energy value itself could in principle be unbounded. However, we observed that they naturally stabilize so we did not explicitly regularize them. The annealing sampling schedule is optimized to improve sample quality for CIFAR-10 dataset, and consist of a total of 2700 steps. For other datasets the shape has less effect on sample quality, see Figure B.7 B for the shape of annealing schedule used.

For the Log likelihood estimation we initialized reverse chain on test images, then sample 10000 intermediate distribution using 10 steps HMC updates each. Temperature schedule is roughly exponential shaped and the reference distribution is an isotropic Gaussian. The variance of estimation was generally less than 10% on the log scale. Due to the high variance of results, and to avoid getting dominated by a single outlier, we report average of the log density instead of log of average density.

B.5 Extended Samples and Inpainting Results

We provide more inpainting examples and further demonstrate the mixing during sampling process in Figure B.3. We also provide more samples for readers to visually judge the quality of our sample generation in Figure B.4, B.5 and B.6. All samples are randomly selected.

B.6 Sampling Process and Energy Value Comparison

Here we show how the average energy of samples behaves vs the sampling temperature. We also show an example of our model making out of distribution error that is common in most other likelihood based models [103] Figure B.7.

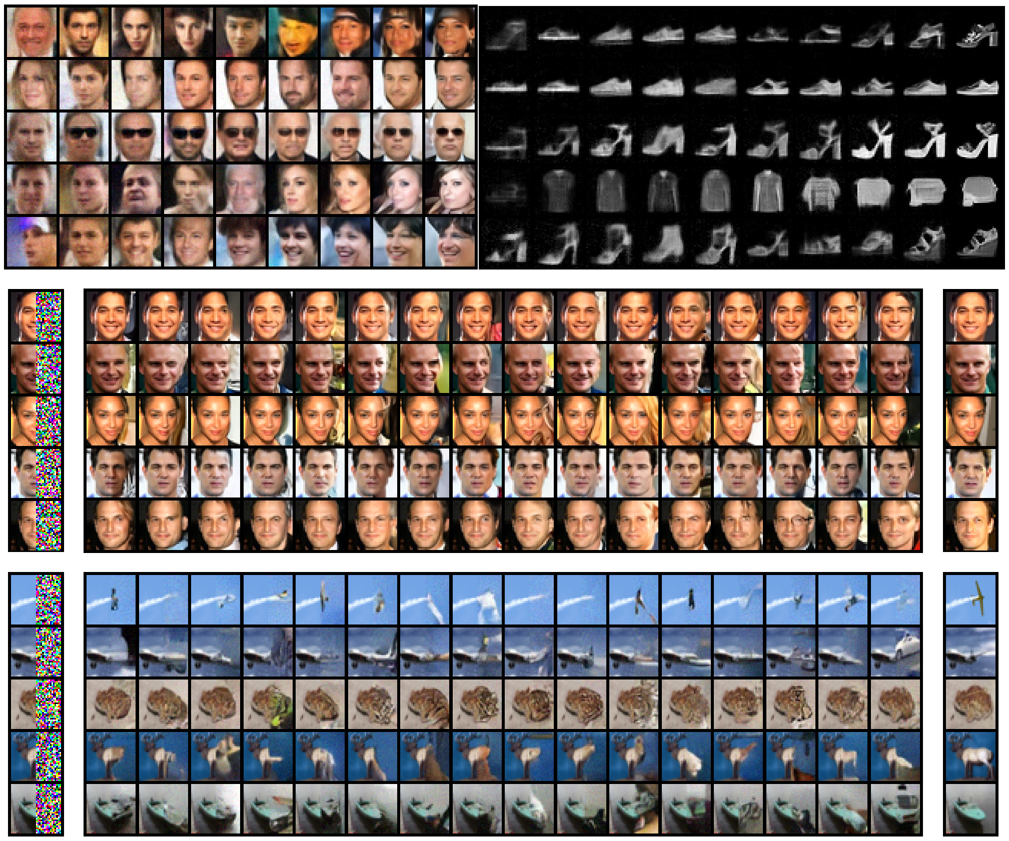


Figure B.3: Denoised sampling process and inpainting results. Sampling process is from left to right.

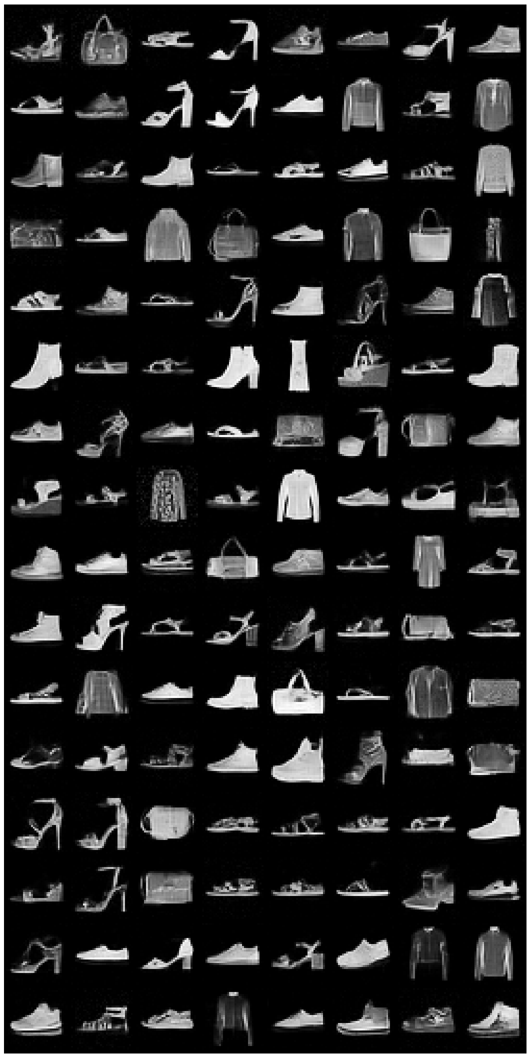
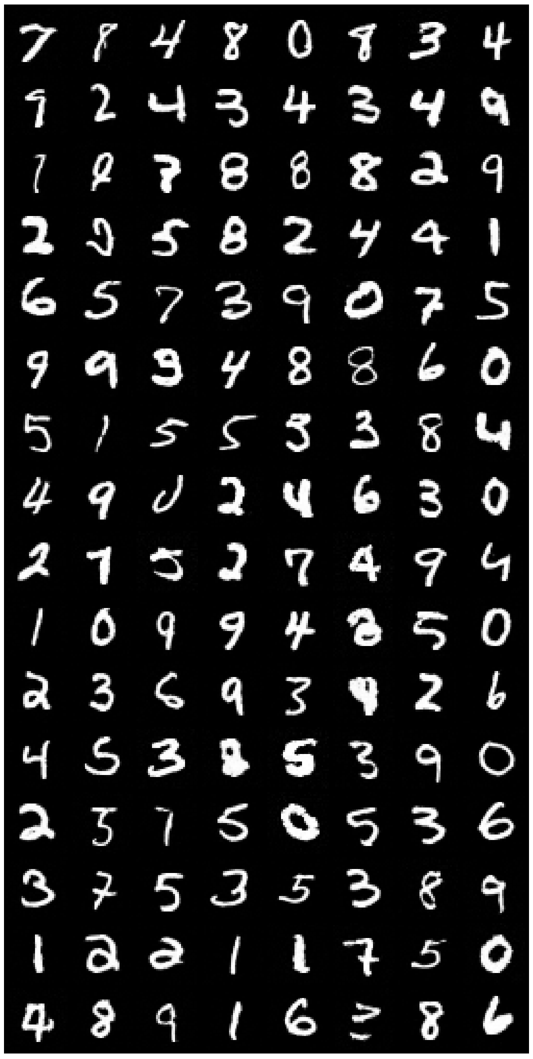



Figure B.4: Extended Fashion MNIST and MNIST samples

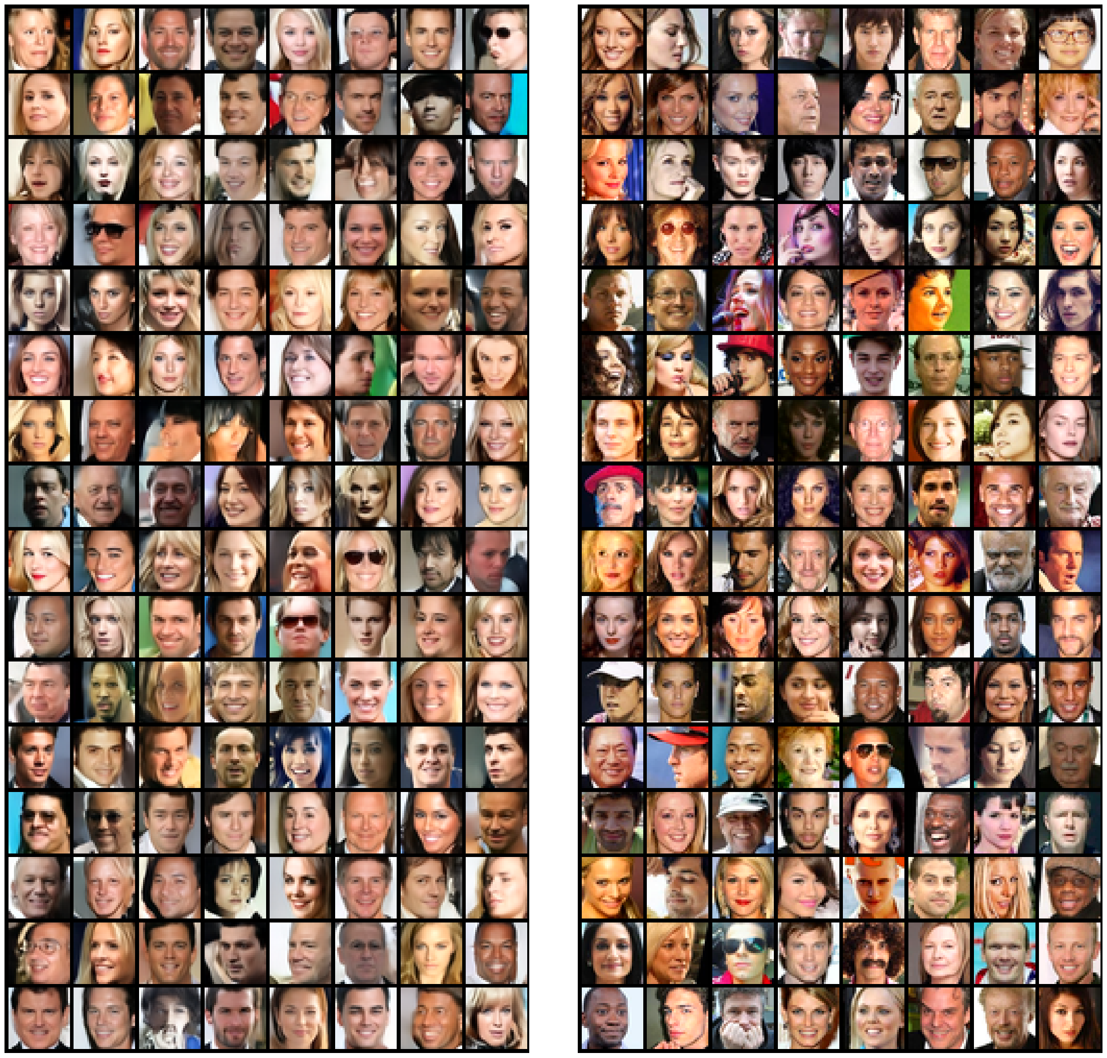


Figure B.5: Samples (left panel) from network trained on CelebA, and training examples from the dataset (right panel).

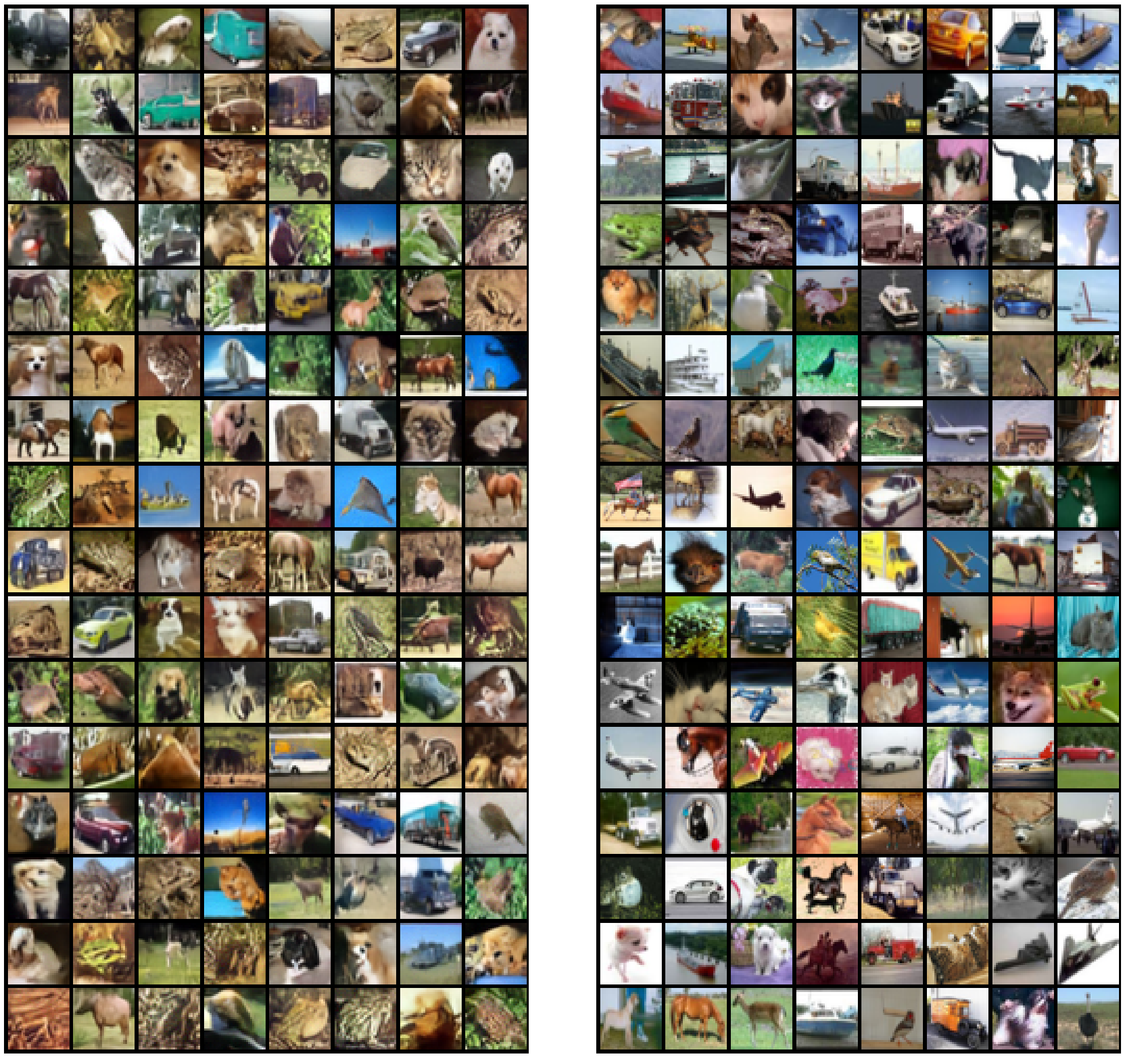


Figure B.6: Samples (left panel) from energy-based model trained on CIFAR-10 next to training examples (right panel).

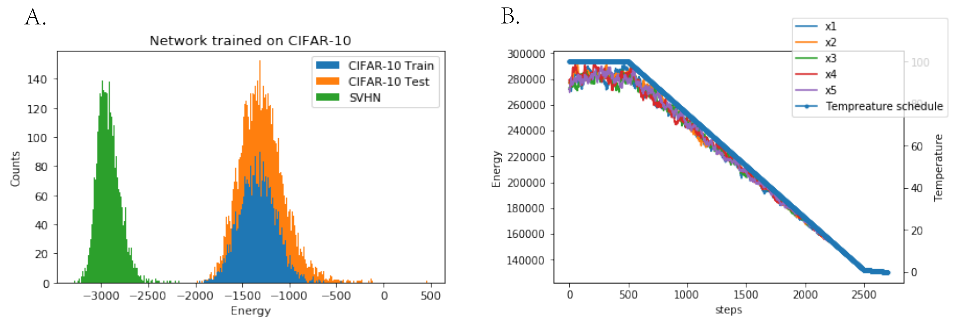


Figure B.7: A. Energy values for CIFAR-10 train, CIFAR-10 test and SVHN datasets for a network trained on CIFAR-10 images. Note that the network does not over fit to the training set, but just like most deep likelihood model, it assigns lower energy to SVHN images than its own training data. B. Annealing schedule and a typical energy trace for a sample during Annealed Langevin Sampling. The energy of the sample is proportional to the temperature, indicating sampling is close to a quasi-static process.

Appendix C

Supplementary material for NEM-MC

C.1 Ablation study: effect of gradient information

In the main text we propose to use gradient information in sampling, inspired by the success of gradient-based sampling techniques. Here we present an ablation study that shows the impact of gradient information and of some architectural simplifications. We compare the full model to two ablated model variants: 1. The original model with data distribution gradient set to 0, this is equivalent to generating proposals with a Real-NVP flow model conditioned on x. 2. A Langevin dynamics with noise generated by model variant 1. Specifically, variant 2 consists of the following proposal process $x' = x + \epsilon z - \frac{\epsilon^2}{2}\partial_x U(x) \odot \exp\left[S(x)\right] + T(x)$, where $z = f(z_0; x)$ is modeled by the original architecture with the gradient turned off, and S, T are neural networks. Thus, variant 2 implements Langevin dynamics with flexible noise and an element-wise affine transformed gradient. It is not flexible enough to express the full covariance Langevin dynamics [156], but we found it to be sufficient for energy-based model training. The computation time is significantly smaller than for our full model because it only uses per step only 2 gradient evaluations instead of $4N$ evaluations.

For training on the 100d Funnel-1 distribution, the learning rate of each model is tuned to the maximum stable value, with both models using the same learning rate schedule. As can be seen in Figure A1, both ablated models learn more slowly and have difficulty with mixing between energy levels, resulting in proposal distributions with poor coverage.

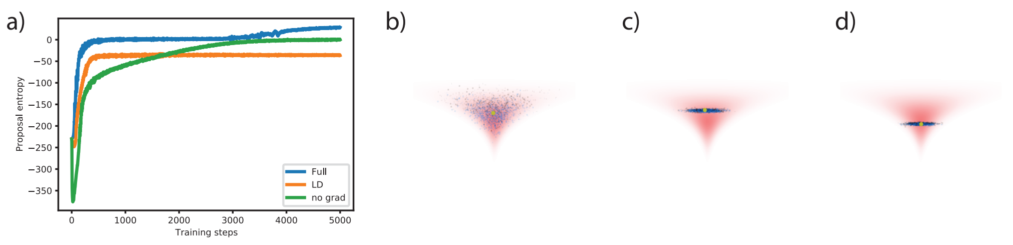


Figure C.1: Illustrating the role of gradient information. a) Comparison of proposal entropy (up to a common constant) during training. Full: full model with gradient, LD: model variant 2, no grad: model variant 1. b), c), d): Example proposal distribution of Full, LD and no grad model.

C.2 Experimental details

Architecture and training details We use a single network for S, Q and T. The network is a 5-layer MLP with ELU activation function and constant width that depends on the dataset (See Table C.1). The weights of both input and output layer are indexed with step number as a means to condition on the step number. The two substeps of the z update need not to be indexed as they use disjoint sets of input and output units indicated by the masks. The weights of all other layers are shared across different steps. We use a separate network with the same architecture and size for R and we condition on the step number in the same way. Weights of the output layers of all networks are initialized at 0. We use random masks that are drawn independently for each z update step.

When training the sampler, we use gradient clipping with L2 norm of 10. Additionally, we use a cosine annealing learning rate schedule. For training the Bayesian logistic regression task, we use a sample buffer of size equal to the batch size. For ESS evaluation we sample for 2000 steps and calculate the correlation function for samples from the later 1000 steps. One exception is the HMC 20d Funnel-3 result, where we sampled for 50000 steps. For the ESS/5k result used in Bayesian logistic regression task, we simply multiply the obtained ESS/MH value by 5000.

Other hyperparameters used in each experiments are listed in Table C.1. All models are trained with batch size of 8192, for 5000 steps, except for 20d Funnel-3, where the batch size is 1024, and the number of training steps is 20000. The momentum parameters in the Adam optimizer are always set to $(0.9, 0.999)$. The scale parameter ϵ is chosen to be 0.01 for EBM training and 0.1 for all other experiments.

For deep EBM training we use architecture variant 2, described in section C.1 because it uses less gradient computation. We use a small 4-layer convnet with 64 filters and ELU activation for S, Q and T. In the invertible network we use a fixed checkerboard mask. For the energy function in the EBM we used a 6 layer convnet with 64 filters and ELU activation function. In the EBM experiment we use 4 steps of z updates. We use a replay buffer with 10000 samples, in each training step we first draw 64 samples randomly from the replay buffer and train the sampler for 5 consecutive steps. Updated samples are put back into the replay buffer. If the average accept rate of the batches is higher than target accept rate -0.1, another 128 samples are draw from the replay buffer and are updated 40 times using the sampler. These samples are then used as negative samples in the Maximum Likelihood training of the EBM. The EBM uses the Adam optimizer with a learning rate of 10^{-4} and Adam momentum parameters of $(0.5, 0.9)$. The sampler uses the same learning parameter, and has an accept rate target of 0.6. The EBM was trained for a total of 100000 steps, where the learning rate is decreased by a factor of 0.2 at steps 80000 and 90000. All results with the MALA sampler are generated by loading a checkpoint at 1000 steps with the trained sampler. For reasons unclear to us, MALA is unable to stably initialize the training, although the later training process with MALA is stable. The FID is calculated for a model trained by learned sampler and trained by MALA at 82k steps.

Other datasets Recent studies of neural network based samplers used datasets other than those reported in the Results. We experimented with applying our model on some of those datasets. In particular, we attempted applying our method to the rotated 100d ill-conditioned Gaussian task in [69], without getting satisfactory result. Our model is able to learn the 2d SCG tasks which shows that it is able to learn a non-diagonal covariance, but in this task it learns very slowly and does not achieve good performance. We believe this is because coupling-based architectures do not have the right inductive bias to efficiently learn the strongly non-diagonal covariance structure, perhaps the autoregressive architecture used in [69] would be more appropriate.

[95] also presented the rough well distribution. We tried implementing it as described in the paper, but found that already a well-tuned HMC can easily sample from this distribution. Therefore we did not proceed to train our sampler on it. We should

	Width	Steps	AR	LR	Min LR
50d ICG	256	1	0.9	10^{-3}	10^{-5}
2d SCG	32	1	0.9	10^{-3}	10^{-5}
100d Funnel-1	512	3	0.7	10^{-3}	10^{-5}
20d Funnel-3	1024	4	0.6	5×10^{-4}	10^{-7}
German	128	1	0.7	10^{-3}	10^{-5}
Australian	128	1	0.8	10^{-3}	10^{-5}
Heart	128	1	0.9	10^{-3}	10^{-5}

Table C.1: Table for hyperparamters used in synthetic datasets and Bayesian Logistic regression. Width: width of MLP networks. Steps: Steps of updates in invertible model f. AR: target acceptance rate. LR: learning rate, Min LR: terminal learning rate of cosine annealing schedule.

also add a comment regarding the 2d SCG task in [95]: In the paper the authors stated that the covariance is $[10^2, 10^{-2}]$, but in the provided code it is $[10^2, 10^{-1}]$, so we use the latter in our experiment.

Some neural network MCMC studies considered the mixture of Gaussian distribution. Our model optimizes a local exploration speed objective starting from small initialization. It is therefore inherently local and not able to explore modes far away and separated by high energy barriers. The temperature annealing trick in the L2HMC paper [95] does result in a sampler that can partially mix between the modes in a 2d mixture of Gaussian distribution. However, this approach cannot be expected to scale up to higher dimensions and more complicated datasets, therefore we did not pursue it. We believe multi-modal target distributions are a separate problem that might be solvable with techniques such as the independent M-H sampler [109], however, not with our current model.

C.3 Additional experimental results

Comparing computation time with HMC Here we compare the efficiency of our method to HMC in terms of actual execution speed (ESS/s) on Bayesian Logistic regression tasks.

We follow [142] in using HMC with 40 Leapfrog steps and the optimal step size reported in this paper. The learned sampler and HMC are both executed on the

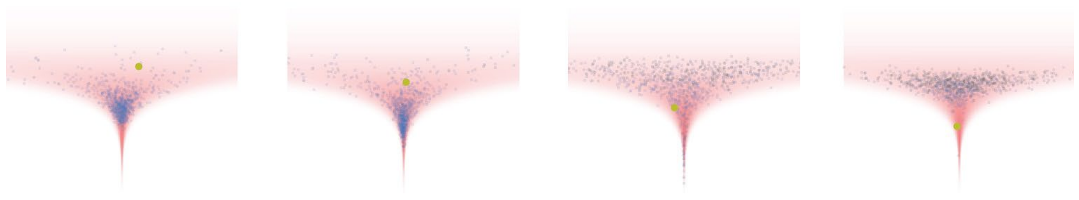


Figure C.2: Visualizations of proposal distributions learned on the Funnel-3 distribution. Yellow dot: x, Blue dots: accepted x', Black dots: rejected x'. The sampler has an accept rate of 0.6. Although not perfectly covering the target distribution, the proposed samples travel far from the previous sample and in a manner that complies to the geometry of the target distribution

same 2080Ti GPU with batch size 64. Results of this experiment are listed in Table C.2. As can be seen, the learned model outperforms HMC significantly.

We did not compare run time with other neural network based MCMC methods due to the difficulty of reproducing the previous models. In our experiment, the run time does not significantly depend on the batch size for batches as large as 10000, indicating that the execution speed is likely limited by *other* factors than the FLOPs (floating point operation per second) of the GPU. This makes execution speed a poor indicator of the true computation budget of the algorithm, but we still provide some execution speed results to comply with the community standard.

Visualization of proposal distributions on Funnel-3 distribution The proposal distributions learned on the 20d Funnel-3 distribution ars visualized in Figure C.2. This demonstrates the ability of the sampler to adapt to the geometry of the target distribution. Further, it shows that the sampler can generate proposal points across regions of very different probability density, since the neck and base of the funnel have very different densities but proposals can travel between them easily.

Our sampler can achieve a significant speed up in terms of ESS/MH compared to HMC sampler, the improvement on the 20d funnel distribution is comparable to the one obtained by the Riemann Manifold HMC. However, the 100d funnel used in the manifold HMC paper could not be handled by our method.

Further EBM results Figure C.3 displays further results from the deep EBM training. a) shows that the learned sampler achieves better proposal entropy early

Dataset (measure)	**HMC**	**Ours**
German (ESS/s)	772.7	**3651**
Australian (ESS/s)	127.2	**3433**
Heart (ESS/s)	997.1	**4235**

Table C.2: Comparing ESS/s between learned sampler and HMC on Bayesian Logistic regression task. Learned sampler is significantly more efficient.

during training. b) shows that the learned sampler converges faster than MALA, as indicated by the lower FID. c) shows the EBM remains stable with a mixture of positive and negative energy differences between batch of real samples and a batch of samples from replay buffer. d) Compares the L2 expected jump of MALA and the learned sampler, plotted in log scale. It has almost the exact same shape as the proposal entropy plot in the main text. f) and g) provide a sanity check showing that the learned sampler does not use a trivial solution. In g) the pixel-wise standard deviation of variable $z = f(z_0; x)$ (note we use variant 2 in Appendix C.1 which does not employ the gradient) is displayed after normalizing it into the image range. One can clearly see image-like structures similar to the sample of which the proposal is generated from. A MALA sampler would produce uniform images in this test, as z is just a Gaussian in MALA. This shows that the learned sampler is utilizing the structures of the samples to generate better proposals.

As shown in Figure C.3 a), MALA achieves similar proposal entropy if not slightly higher later during EBM training, while the learned sampler helps training initialization and early training. This suggests for future research that the optimal strategy could be to use the learned sampler initially and then switch to MALA once it produces similar proposal entropy.

Since we do not use noise initialization during EBM training, our model does not provide a meaningful gradient to re-sample from noise after the model has converged. This is quite different from what was reported in for example [177, 51]. The combination of non-mixing as discussed earlier, and inability of sampling from noise brings the problem of not being able to obtain new samples of the model distribution after the EBM is trained, replay buffer is all we have to work with. Resolving this difficulty is left for future study.

Checks of correctness for the proposed sampler and the EBM training process We run some simple experiments to check the correctness of samples generated from the proposed sampler and show that the EBM training process is not biased by the learned sampler, see Figure C.4.

To check if the learned sampler generates correct samples for the Bayesian Logistic regression task, we compare dimension-wise mean, standard deviation and 4th moment of samples from the learned sampler and samples from the HMC sampler. We average over large amount of samples to obtain the moments, and the result matches very closely, indicating that the learned sampler samples from the correct target distribution, see Figure C.4 a).

Second, we sample an EBM energy function trained by the learned sampler with the MALA sampler, samples are initialized with samples from the replay buffer. As shown in Figure C.4 b), MALA generate plausible samples from the model and does not cause issues such as saturated image [115], indicating that the learned EBM is not biased by the learned sampler and is indeed a valid energy function of the data distribution.

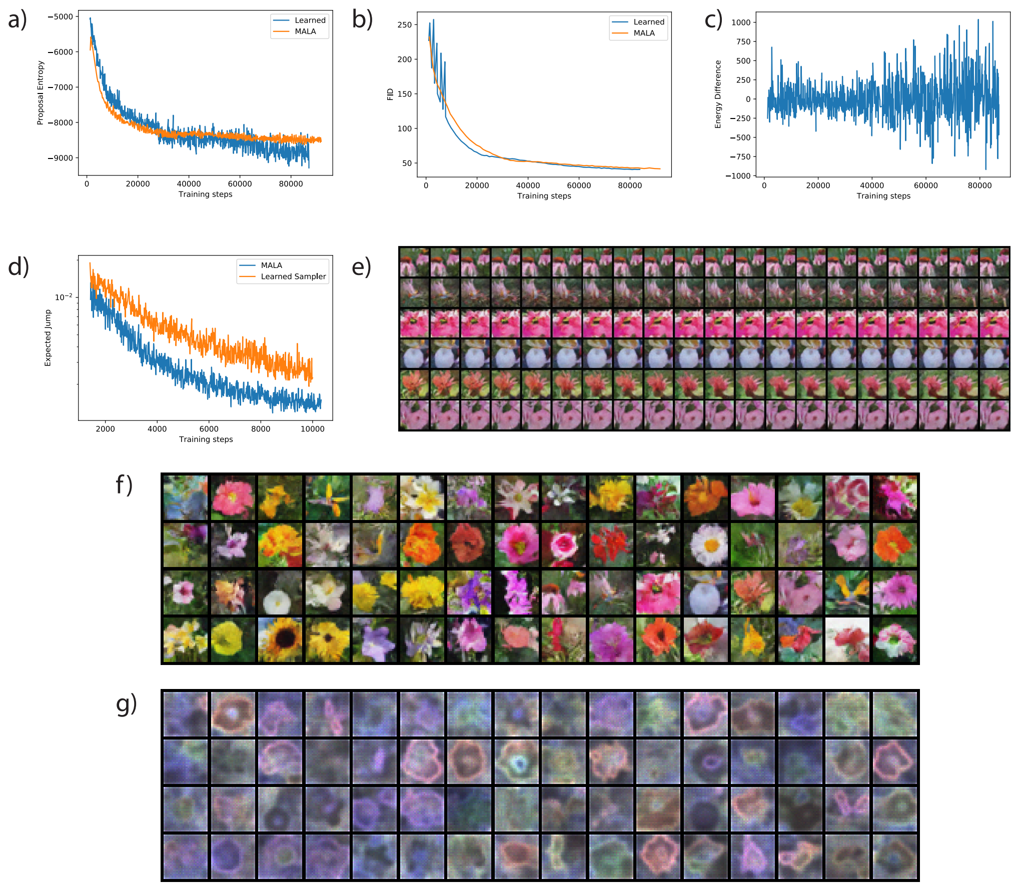
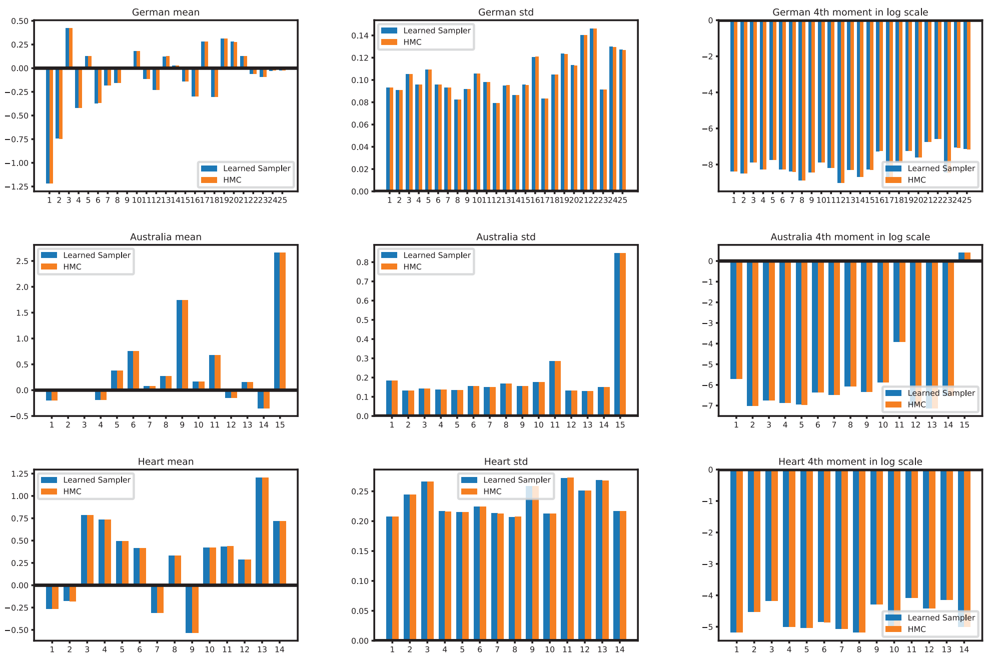


Figure C.3: Further results for Deep EBM. a), b) and c): Proposal Entropy, FID of replay buffer and energy difference during training. Result for MALA is also included in a) and b). a) shows that the learned sampler achieves better proposal entropy early during training. b) shows that the learned sampler converges faster than MALA. c) shows the EBM remain stable with a mixture of positive and negative energy difference. d) Compares L2 expected jump of MALA and learned sampler, plotted in log scale. It has almost the exact same shape as the proposal entropy plot in the main text. e) More samples from sampling process of 100k steps with the learned sampler. f) g) Samples from the replay buffer and the corresponding visualization of the pixel-wise variance of displacement vector z evaluated at the samples. Images in f) and g) are arranged in the same order. Image-like structures that depends on the sample of origin are clearly visible in g). A MALA sampler would give uniform variance.

a)



b)

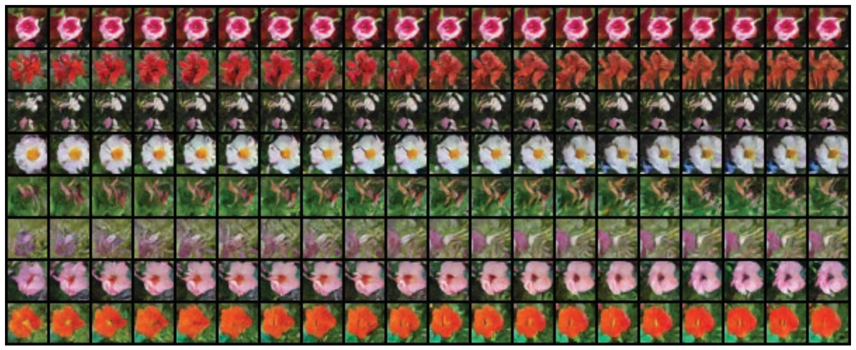


Figure C.4: Checking correctness of samples and EBM training process. a) Comparing dimension-wise mean, standard deviation and 4th moment of samples obtained from HMC and learned sampler for the Bayesian Logistic Regression datasets. Moments are matching very closely, indicating the learned sampler generates samples from the correct target distribution. b) 100k sampling steps by MALA sampler on an EBM energy function trained by the adaptive sampler, samples are initialized from the replay buffer. Samples look plausible throughout the sampling process. This indicates that stable attractor basins are formed are not specific to the learned sampler, and that EBM training is not biased by the adaptive sampler.

Appendix D

Supplementary material for NMCE

D.1 Additional Results and Discussions

Synthetic experiment

We verify the proposed manifold clustering and embedding algorithm by a simple synthetic experiment. Ground truth data from a union of two manifolds with dimension 3 and 6 is generated by passing 3d and 6d iid. Gaussian noise through two randomly initialized neural networks with Leaky-ReLU activation function (negative part slop 0.2). Data augmentation is Gaussian noise with manitude 0.1. Training used two stages, the first stage used only TCR objective, the second stage with full NMCE objective.

Two situations are examined. One where the random neural network also has randomly initialized biases, which will cause the two manifold to be far apart, making locality constraint implemented by noise augmentation sufficient for identifying the manifolds. Another situation is when bias is not used, the two manifolds then intersect at 0, where the density is rather high. This makes the manifolds not identifiable with only locality constraint.

Results are listed in Table D.1. As can be seen, when the clusters are identifiable, NMCE is able to correctly cluster the data points, as well as learn latent features that is perpendicular between different clusters. At the same time, the feature is not collapsed within each cluster, since if so the average cosine similarity within cluster would be 1. When the clusters are not identifiable, they cannot be learned correctly.

Table D.1: Result from synthetic experiment. Accuracy is in % (chance level is 50%), z-sim is the average absolute value of cosine similarity between feature vectors z for different pairs of z. True Cluster: pairs of z are from different ground truth clusters. Found Clusters: pairs of z are from two different found clusters. Within Cluster: pairs of z are randomly picked from the same found cluster, averaged between two found clusters.

Dataset	**Accuracy**	**z-sim: True Cluster**	**z-sim: Found Clusters**	**z-sim: Within Cluster**
Identifiable	100.0	0.017	0.017	0.503
Not-identifiable	69.8	0.717	0.287	0.770

Relationship to self-supervised learning with MCR^2

The original paper [178] performed self-supervised learning using MCR^2 objective without any additional term. Their method treats different augmentations of the same image as a self-labeled subspace. They used large number of augmentations (50) of each image, with only 20 images in each batch. The performance of this method is rather poor, which is expected based on our understanding. In this case, augmented images will form a subspace with certain dimension in the feature space, thus large amount of information about augmentations will be preserved in the latent space. Clustering can then utilize style information and not respect class information.

To improve performance, a variant called MCR^2-CTRL is developed, where the total coding rate term is down-scaled. This variant performs significantly better, and is also used in our comparison. This result is also expected, since decreasing the subspace term effectively contract different augmentations of the same image in the feature space, making the feature better respect the constraint needed for correct clustering. However, since the total coding rate is not high in this case, the feature is not diverse enough to achieve good performance.

Fine-tuning backbone with NMCE objective

For CIFAR-10, CIFAR-20 and STL-10 experiments, the first training stage already learns very strong self-supervised features, which is then clustered into subspaces in the second stage with backbone network frozen. The clustering performance is already quite good after this stage. In the third stage, the backbone is fine-tuned, which further improves clustering performance. In Table D.2, we show the effect of fine-tuning backbone network on CIFAR-10 and CIFAR-20 experiment with ResNet-18. As can be seen, fine-tuning produces a small but noticeable gain in clustering performance for all metrics and both datasets. This indicates that using the full NMCE objective can improve performance. If the optimization issue can be resolved, and the entire network is trained with the NMCE objective from scratch, the performance may be further improved.

Table D.2: Fine tuning backbone with NMCE objective. Results shown are from ResNet-18. Fine tuning backbone improves result slightly but notably.

Model	ACC	NMI	ARI
CIFAR-10 before	0.819	0.743	0.690
CIFAR-10 after	**0.830**	**0.761**	**0.710**
CIFAR-20 before	0.422	0.471	0.300
CIFAR-20 after	**0.437**	**0.488**	**0.322**

Compare pool and proj feature on low data classification

Here we compare SVM accuracy of Pool and Proj features from CIFAR-10 ResNet-18 experiment. The feature averaged over 16 augmentations. The accuracy is plotted as a function of the number of labeled training samples used, see Figure D.1.

As can be seen, Proj feature clearly outperforms Pool feature when few labeled examples are available. This makes Proj feature much more useful than Pool feature, since labeled examples are often scarce real applications.

Note that what shown here is obviously not the optimal way to use labeled example, if one further leverage the clustering information, accuracy should reach 90 with only 10 labeled examples.

Effect of Lambda parameter

Here we study the effect of λ parameter that balances the constraint and subspace feature learning term. CIFAR-10 self-supervised accuracy with SVM and kNN evaluation on Proj feature is listed in Table D.3. As can be seen, the accuracy is reasonable

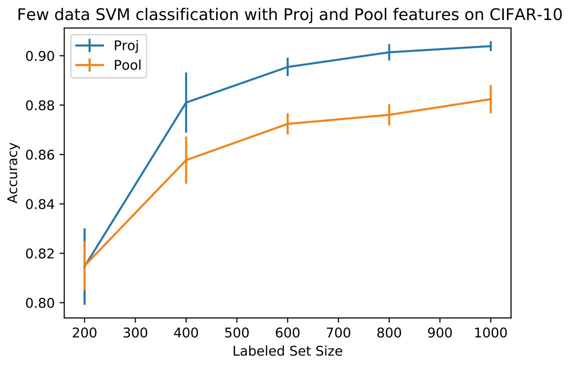


Figure D.1: CIFAR-10 SVM test accuracy plotted against number of labeled examples used for Pool and Proj feature from ResNet-18 experiment. Features averaged over 16 augmentations is used. Error bar is std. over 10 random sampling of training examples.

in a range of λ spanning more than 3x low to high, indicating that the quality of the learned feature is not very sensitive to this parameter.

Table D.3: Effect of parameter λ.

λ	**20**	**30**	**40**	**50**	**60**	**70**
Proj SVM acc	0.899	0.902	0.903	0.902	0.903	0.903
Proj kNN acc	0.890	0.895	0.894	0.896	0.895	0.897

Understanding the feature space

Here we show that the TCR objective used in the first stage in our training procedure for CIFAR and STL-10 datasets theoretically achieve the same result as the recently proposed VICReg [7] and BarlowTwins[180], both are self-supervised learning algorithms. The optimization target of the two techniques are both making the covariance matrix of latent vector Z approach the diagonal matrix. The first stage training using TCR essentially achieves the same result. To see this, we note the following property of the coding rate function [79]: For any $Z \in \mathbb{R}^{m\times d}$:

$$\mathsf{logdet}(\mathbf{I} + Z^T Z) = \sum_{i=1}^{\mathsf{min}(m,d)} \mathsf{log}(1+\sigma_i^2) \tag{D.1}$$

Where σ_i is the ith singular value of Z. Additionally, we have: $\sum_{i=1}^{\mathsf{min}(m,d)} \sigma_i^2 = ||Z||_{\mathbf{F}}^2$, which follows easily from $||Z||_{\mathbf{F}}^2 = \mathsf{tr}(Z^T Z)$. Since the function $\mathsf{log}(1+x)$ is concave, the optimization problem $\mathsf{max}_{x_1,x_2,\ldots,x_n} \sum_{i=1}^{n} \mathsf{log}(1+x_i)$ given $\sum_{i=1}^{n} x_i = C$ reaches maxima when all x are equal to each other. Since we normalize the row of Z, $||Z||_{\mathbf{F}}^2 = m$, optimization of Equation D.1 result in solution with uniform singular value, which is equivalent to diagonal covariance.

We could not successfully reproduce VICReg in our setup due to the large amount of hyper-parameters that needs to be tuned. Therefore we resort to the open-source library solo-learn [26], which provided VICReg implementation. Running the provided script for VICReg produced accuracy of 91.61% on CIFAR-10. We also implemented TCR objective in solo-learn library. Running TCR obtained accuracy of 92.1%. All hyper-parameters and augmentations are the same as VICReg, except batch size is 1024 instead of 256, projection dimension is 64 instead of 2048. Larger batch size or smaller projection dimension didn't work for VICReg, so we stayed with the original parameters.

The covariance matrices of learned feature for SimCLR, VICReg and TCR computed over entire training set are visualized in Figure D.2. For VICReg, first 128 dimension is visualized out of 2048. As can be seen, the diagonal structure is visible in SimCLR feature space, TCR feature space is the closest to diagonal matrix. VICReg feature space is also quite close to diagonal, but the off-diagonal terms seems noisier. Additionally, we plot normalized singular values for VICReg projection space in Figure D.4. This can be compared to TCR and SimCLR singular values in Figure D.3 a). As can be seen, TCR achieves flatter singular value distributions than VICReg, neither SimCLR or VICReg are close to full rank in projection space.

We demonstrate subspace structure of feature space after clustering with full NMCE objective by plotting singular values of each learned subspace in Figure D.3 b). Each subspace found are around rank 10. In all other panels of Figure D.3, we display samples whose feature vector has the highest cosine similarity to the top 10 principle components of each subspace. One can see that most principle components represent a interpretable sub-cluster within each class (even if the sub-cluster is of a different class than the cluster). Same as in Figure 5.2, feature vectors calculated by averaging 16 augmented images are used.

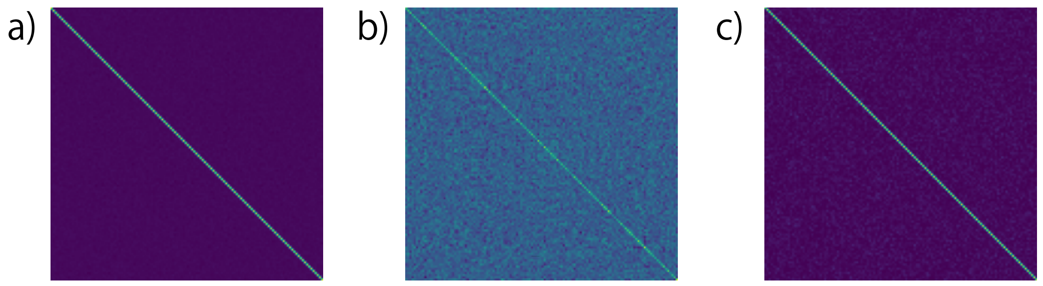


Figure D.2: a), b), c) Covariance matrix of feature vector computed over training set for TCR, SimCLR and VICReg. VICReg result is slightly noisier on the diagonal than TCR. VICReg result is the first 128 dimensions out of 2048, see text for details.

D.2 Experimental Details

We list hyper-parameters for each experiment in Table D.4.

Toy and synthetic experiments

The toy dataset consists of double spiral with radius approximately equal to 15 and Gaussian noise magnitude of 0.05, samples are generated online for each batch. Data augmentation is Gaussian noise with magnitude 0.05.

COIL-20 and COIL-100

The augmentation policy we found with manual search on COIL20 is (all are from torchvision transforms): 1. Random Horizontal Flip with $p = 0.5$. 2. RandomPerspective with magnitude 0.3 and $p = 0.6$ 3. ColorJitter with manitude (0.8, 0.8, 0.8, 0.2), always applied. The entire dataset is passed as a single batch.

CIFAR-10, CIFAR-20, STL-10, ImageNet-10 and ImageNet-Dogs

For CIFAR-10 and CIFAR-20, we use standard ResNet-18 and ResNet-34 with 64 input filters. The first layer uses 3x3 kernel, and and no max pooling is used. For other experiments, we use standard ResNet-18 and ResNet-34 with 5x5 first layer kernel and max pooling. For COIL-20 and COIL-100 experiment, 32 input filters is used, and the ResNet-10 is obtained by reducing number of blocks in each stage in ResNet-18 to 1. For full details, see our code release.

For CIFAR-10, CIFAR-20, STL-10, ImageNet-10 and ImageNet-Dogs experiments, we used LARS optimizer [176], with base batch size 256, for other experiments we used Adam. We note that stage 2 and 3 in the 3 stage training process is quite sensitive to weight decay, a careful search of this parameter is usually required for good performance.

Computational Cost

All experiments involving ResNet-34 requires 8 GPUs, others can be done in 1 GPU.

Our objective doesn't add significant computational burden compared to neural-networks involved. The covariance matrix is computed within a batch. For a batch of latent feature vectors with shape $[B, d]$, where B is batch size and d is latent dimension. We first compute the covariance matrix with shape $[d, d]$, with $\mathcal{O}(B^2 d)$ cost. Then the log determinant of this matrix is calculated, which we assume has $\mathcal{O}(d^3)$ cost. The $\mathcal{O}(B^2)$ scaling with respect to batch size is the same as most contrastive learning method such as SimCLR, which is known to scale to very large batch size.

Table D.4: Hyper-parameters for all experiments. lr: learning rate. wd: weight decay. ϵ: coding error. d_z: dimension of feature output. λ: regularization constant. bs: batch size. epochs (steps): total epochs trained, or total steps trained if the entire dataset is passed at once. S1, S2, S3 denote 3 training stages.

Model	**lr**	**wd**	ϵ	d_z	λ	**bs**	**epochs (steps)**
Double Spiral	1e-3	1e-6	0.01	6	4000	4096	30000
Synthetic	1e-3	1e-6	0.01	12	100	4096	3000
COIL-20	1e-3	1e-6	0.01	40	20	1440	2000
COIL-100	1e-3	1e-6	0.001	200	20	7200	10000
CIFAR-10 ResNet-18 S1	1	1e-6	0.2	128	50	1024	600
CIFAR-10 ResNet-18 S2	0.5	0.005	0.2	128	50	1024	100
CIFAR-10 ResNet-18 S3	0.003	0.005	0.2	128	50	1024	100
CIFAR-10 ResNet-34 S1	1	1e-6	0.2	128	50	1024	1000
CIFAR-10 ResNet-34 S2	0.5	0.001	0.2	128	0	1024	100
CIFAR-10 ResNet-34 S3	0.003	0.001	0.2	128	0	1024	100
CIFAR-20 ResNet-18 S1	1	1e-6	0.2	128	50	1024	600
CIFAR-20 ResNet-18 S2	0.5	0.001	0.2	128	0	1024	100
CIFAR-20 ResNet-18 S3	0.003	0.001	0.2	128	0	1024	100
CIFAR-20 ResNet-34 S1	1	1e-6	0.2	128	50	1024	1000
CIFAR-20 ResNet-34 S2	0.5	0.002	0.2	128	0	1024	100
CIFAR-20 ResNet-34 S3	0.003	0.002	0.2	128	0	1024	100
STL-10 ResNet-18 S1	1	1e-6	0.2	128	30	1024	1000
STL-10 ResNet-18 S2	0.5	0.002	0.2	128	0	1024	400
STL-10 ResNet-18 S3	0.0005	0.002	0.2	128	30	1024	400
STL-10 ResNet-34 S1	1	1e-6	0.2	128	30	1024	2000
STL-10 ResNet-34 S2	0.5	0.002	0.2	128	0	1024	400
STL-10 ResNet-34 S3	0.0005	0.002	0.2	128	30	1024	400

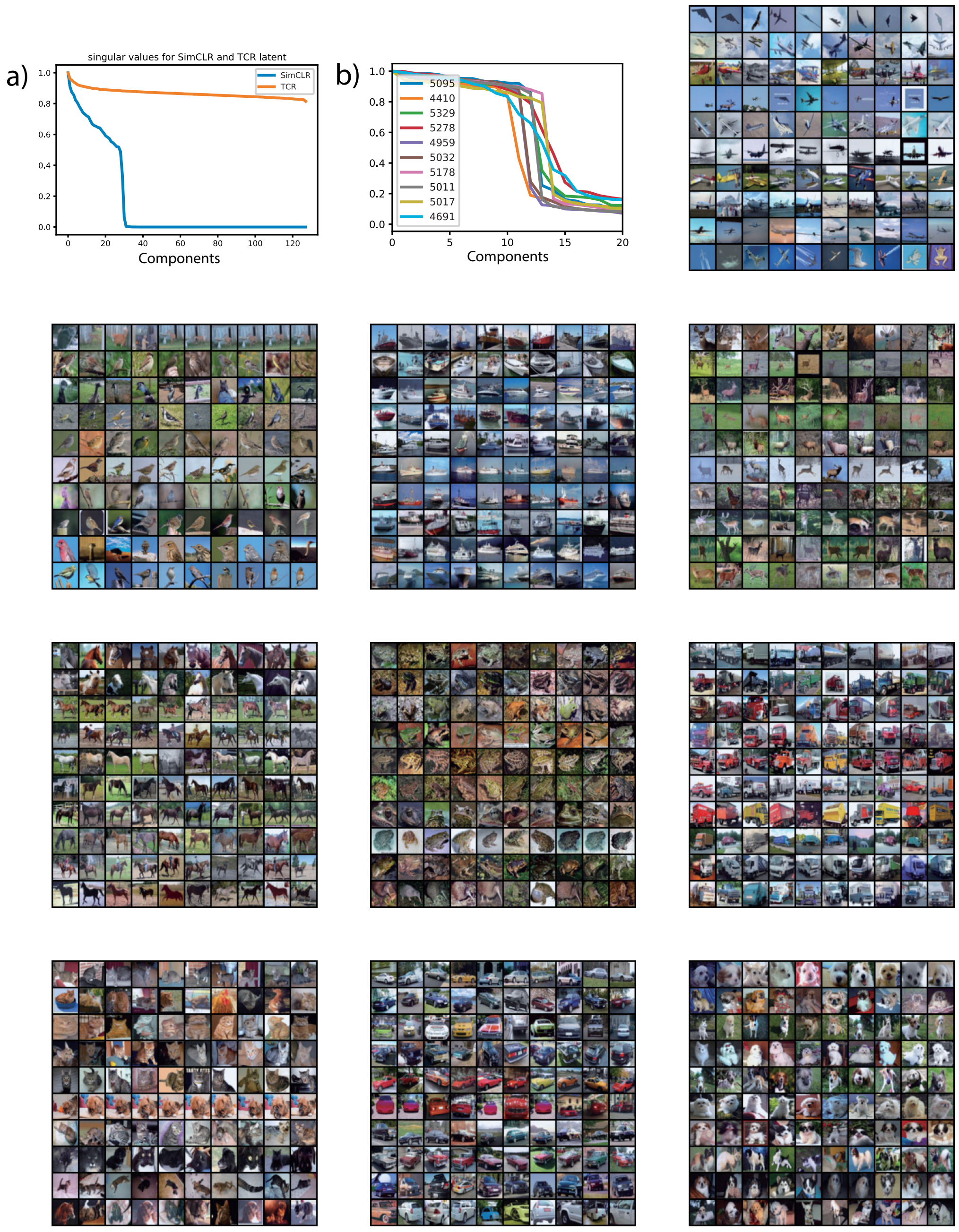


Figure D.3: a) Singular values of feature vector distribution for SimCLR and TCR. b) Singular values for subspaces learned after all 3 training stages. The rest of panels: visualizing 10 training examples most similar to principal components of each clustered subspaces.

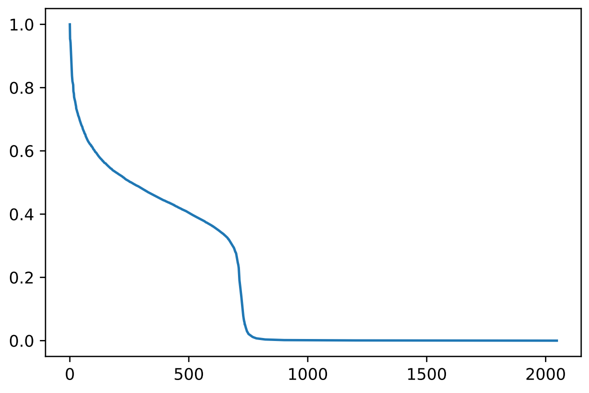


Figure D.4: Singular values of feature vector distribution for VICReg using ResNet-18. VICReg uses 2048d feature space.