NO CONDITIONAL MODELS FOR ME: TRAINING JOINT EBMS ON MIXED CONTINUOUS AND DISCRETE DATA

Jacob Kelly
University of Toronto & Vector Institute
jkelly@cs.toronto.edu

Will Grathwohl
University of Toronto & Vector Institute
wgrathwohl@cs.toronto.edu

ABSTRACT

We propose energy-based models of the joint distribution of data and supervision. While challenging to work with, this approach gives flexibility when designing energy functions and easy parameterization for structured supervision. Further, these models naturally allow training on partially observed data and predictions conditioned on any subset of the modeled variables. We identify and address the main difficulty in working with these models: sampling from the joint distribution of data and supervision. We build upon recent developments in discrete MCMC sampling and apply them alongside continuous MCMC techniques developed for energy-based models. We present experimental results demonstrating that our proposed approach can successfully train joint energy-based models on high-dimensional data with structured supervision capable of both accurate prediction and conditional sampling.

1 INTRODUCTION

The flexibility of Energy-Based Models (EBMs) make them an appealing approach for a number of generative modelling tasks. The ease with which structure may be incorporated and their relationship to physical systems has long made them popular in the broader scientific community (Ingraham et al., 2019; Du et al., 2020c; Noé et al., 2019). After a period of reduced interest from the machine learning community, EBMs have regained popularity as an approach for generative modeling (Nijkamp et al., 2020a; Du & Mordatch, 2019) and have found utility in many applications.

One such application is Joint Energy-based Models (JEM) (Grathwohl et al., 2019). These models re-purpose existing state-of-the-art classifier architectures to define an energy-based model of the joint distribution $p(x, y)$ between data and class labels, instead of the conditional distribution of labels given data $p(y|x)$ as is typically used for classification. Moving from conditional to joint models gives improved calibration, out-of-distribution detection, and adversarial robustness while retaining high discriminative performance. This approach has been extended to domains beyond images (He et al., 2021; Hataya et al., 2021) and to new applications such as semi-supervised learning (Grathwohl et al., 2020; Zhao et al., 2020).

Grathwohl et al. (2019) focus on models for image classification where x is a continuous images and y is a discrete, 1-dimensional class label. Naively training this model would require MCMC sampling from the joint distribution of x and y which is challenging because y is discrete. Thankfully, the structure of the energy-function allows us to marginalize out the label y to compute an EBM for $\log p(x)$ and a normalized $p(y|x)$ model. We can then train to maximize the factorized likelihood $\log p(x, y) = \log p(x) + \log p(y|x)$. Techniques for training EBMs on continuous data are used to maximize the first term and the second term is optimized to minimize cross-entropy.

In many situations we are interested in supervision which is more complicated than a 1-dimensional class label. We may, for example, wish to predict a segmentation mask, or a text caption given an image. Here the supervisory information is rich, discrete, high dimensional, and its variables have correlations which are important to model. In settings such as these we may not be able to marginalize out the y or compute a normalized $p(y|x)$. Thus, we will not be able to train in the same way as Grathwohl et al. (2019).

An alternative approach to train this model would be to maximize $\log p(x, y)$ directly. This would require sampling from the joint directly which can be challenging given the mixed continuous-discrete nature of the sampling problem. Notably though, solving this sampling task would enable the training of models simultaneously capable of discriminative modeling, conditional sampling, and semi-supervised learning.

In this work we take a first step to address these issues and demonstrate that joint EBMs can be trained on high-dimensional mixed continuous and and discrete data. These models generate convincing conditional samples and make accurate predictions. To accomplish this, we build upon recent work on MCMC sampling in discrete spaces (Grathwohl et al., 2021) and MCMC-based EBM training (Nijkamp et al., 2020a; Du & Mordatch, 2019).

In the next sections we provide some background on EBMs and MCMC sampling. We then present our approach to train and sample from joint continuous-discrete EBMs. Finally, we present results training these models on some small but exemplary datasets.

2 Background

In the following sections we refer to our input data $x \in \mathcal{X}$ and supervision $y \in \mathcal{Y}$. We assume that x is continuous and y is discrete (either binary or categorical).

2.1 Energy-Based Models

An energy-based model is any model which parameterizes a probability distribution as

$$p_\theta(x) = \frac{e^{f_\theta(x)}}{Z(\theta)} \tag{1}$$

where $f_\theta : \mathbb{R}^D \to \mathbb{R}$ fully specifies the model and $Z(\theta) = \int e^{f_\theta(x)} \mathrm{d}x$ is the normalizing constant.

While the flexibility of EBMs makes them appealing, this flexibility comes with the cost of making sampling and likelihood evaluation much more challenging than it is for more restricted model classes. In fact, likelihoods cannot be exactly computed, or even lower-bounded. EBMs are typically trained with gradient-based optimization using the following estimator for the gradient of the maximum likelihood objective:

$$\nabla_\theta \log p_\theta(x) = \nabla_\theta f_\theta(x) - \mathbb{E}_{p_\theta(x')}[\nabla_\theta f_\theta(x')]. \tag{2}$$

Use of this estimator requires generating samples from $p_\theta(x)$. Since exact sampling is intractable, we resort to using approximate samples generated with MCMC (Tieleman, 2008). Fortunately, a host of techniques and tricks have been developed to make training with MCMC efficient when using deep neural networks to define the energy-function (Du & Mordatch, 2019; Du et al., 2020b; Nijkamp et al., 2019; 2020a; Xie et al., 2016).

The preferred sampling approach for continuous data is Langevin Dynamics (Welling & Teh, 2011) which updates samples with

$$x_{t+1} = x_t + \frac{\epsilon^2}{2\lambda} \nabla_x f_\theta(x) + \epsilon\alpha, \qquad \alpha \sim N(0, I) \tag{3}$$

where the step-size ϵ, and the temperature λ are hyperparameters. Common values for image data are $\epsilon = 0.01$, $\lambda = \frac{1}{20{,}000}$. The low temperature is necessary to generate samples quickly enough for efficient training.

2.2 Gibbs-With-Gradients

Gradient-based MCMC methods such as Hamiltonian Monte-Carlo (Neal et al., 2011) or Langevin Dynamics (Equation 3) have enabled the training of deep EBMs on continuous data. Similar progress has not been made on discrete data because a lack of scalable MCMC methods for discrete distributions.

In the following we discuss sampling from $p(y) = \frac{e^{f(y)}}{Z}$ where $y \in \mathcal{Y}$ is discrete. Recent work (Grathwohl et al., 2021) presents an MCMC sampler for discrete distributions which has enabled the training of deep EBMs on discrete data. The sampler, Gibbs-With-Gradients, is based on

the Metropolis-Hastings algorithm and defines a proposal distribution, $q(y'|y)$, which makes local updates $y' \in H(y)$ (where $H(y)$ is the Hamming-ball of size 1 around x).

Given the current state of a sampling chain, y_t, the sampler proposes an update by sampling $y' \sim q(y'|y_t)$. This proposed update is accepted with probability $\min\{\exp(f(y') - f(y))\frac{q(y|y')}{q(y'|y)}, 1\}$. If the sample is accepted we set $y_{t+1} = y'$, otherwise we set $y_{t+1} = y_t$.

Recent theoretical work on MCMC (Zanella, 2020) shows that the following proposal:

$$q(y'|y_t) = \frac{\exp\left(\frac{f(y')-f(y_t)}{2}\right)}{\sum_{y'' \in H(y_t)} \exp\left(\frac{f(y'')-f(y_t)}{2}\right)}. \tag{4}$$

is near optimal for MCMC in these settings. Unfortunately, sampling from this proposal and computing likelihoods requires the computation of $f(y') - f(y)$ for each $y' \in H(y)$ which requires $O(D)$ function evaluations for D-dimensional data.

Grathwohl et al. (2021) noticed that discrete distributions are often implemented as continuous functions of real-valued inputs. In these settings the discrete structure is created by restricting the domain to a discrete subset of the possible inputs. A prime example is the Ising model $\log p(y) = y^T W y + b^T y - \log Z$. The log-probability function can accept inputs in $\mathbb{R}^D$ but the distribution is defined on $y \in \{0, 1\}^D$. In these settings we can utilize a Taylor-series approximation

$$f(y') - f(y) \approx (y' - y)^T \nabla_y f(y) \tag{5}$$

Using this approximation, we can approximate the proposal of Equation 4 using only $O(1)$ function evaluations. Despite these approximations, Grathwohl et al. (2021) demonstrate that this approach to sampling (when it can be applied) greatly outperforms other generic discrete MCMC samplers such as Gibbs sampling or the Hamming-Ball Sampler (Titsias & Yau, 2017).

3 Joint Energy-Based Models

We wish to train an EBM for the joint distribution of data $x \in \mathcal{X}$ and supervision $y \in \mathcal{Y}$ where the supervision may be high dimensional and structured. Our models take the form

$$\log p_\theta(x, y) = f_\theta(x, y) - \log Z(\theta) \tag{6}$$

where $f_\theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a flexible parametric function. Examples could include:

- $f_\theta(x, y) = \texttt{neural\_net}(\texttt{concat}(x, y); \theta)$
- $f_\theta(x, y) = g_\theta(x)^T h_\theta(y)$ where $g_\theta : \mathcal{X} \to \mathbb{R}^D$ and $h_\theta : \mathcal{Y} \to \mathbb{R}^D$

Importantly, models of this form can represent correlations between the variables of y and can define distributions over highly-structured y. For example, if y is permutation invariant or graph-structured, a Deep Set (Zaheer et al., 2017) or Graph Convolutional network (Zhang et al., 2019), respectively can be used to parameterize the energy function defining a model which respects the invariance structure in the data.

Working with models of this form can be challenging because both conditionals $p(x|y), p(y|x)$ and both marginals $p(x), p(y)$ may be unnormalized. Intriguingly though, if we are able to sample from the joint then we will easily be able to maximize likelihood of any marginal and make predictions of any chosen variables given any subset of the remaining variables. Thus in this setting, many distinct problems that have been addressed in notably different ways are reduced to solving a single, albeit challenging, problem.

To make predictions of y given x we can use MCMC to draw samples from $p(y|x)$. This distribution has the form

$$\log p(y|x) = \log p(x, y) - \log p(x) = f_\theta(x, y) - C(x) \tag{7}$$

where $C(x)$ is a constant that does not depend on y. Thus, $p(y|x)$ is an EBM with unnormalized log probability given by evaluating $f_\theta(x, y)$ with x fixed to the conditioning value. Symmetrically, we

can apply the same approach to sample from $p(x|y)$. Given our joint model, we can sample from any conditional by fixing the observed variables and evaluating $f_\theta(x, y)$ (and its gradient).

A similar argument allows us to train on partially-observed data. Assume we have observed a sample x' with no corresponding y. We can estimate the marginal likelihood's gradient with

$$\nabla_\theta \log p_\theta(x') = \mathbb{E}_{p_\theta(y|x')}[\nabla_\theta f_\theta(x', y)] - \mathbb{E}_{p_\theta(x,y)}[\nabla_\theta f_\theta(x, y)]. \tag{8}$$

Thus, dealing with unnormalized joint models, making predictions, training on fully-observed data, training on partially-observed data, and drawing joint samples are roughly equivalent problems. Of course, joint sampling from high-dimensional, unnormalized distributions is an incredibly difficult task but recent advances in gradient-based sampling and EBMs (Welling & Teh, 2011; Grathwohl et al., 2021; Du et al., 2020c; Grathwohl et al., 2019) have demonstrated that accurate samples can be generated from large-scale EBMs.

4 Training

In this section we outline our simple approach to sample from $p_\theta(x, y)$ where x is continuous and y is discrete. Defining efficient MCMC samplers for such distributons is still an open problem. When y is held fixed, we find that $p_\theta(x|y)$ is simply an EBM defined on continuous x. In this setting Langevin Dynamics has been successfully applied to generate samples. With x held fixed, we find $p_\theta(y|x)$ is an EBM defined on discrete variables and Gibbs-With-Gradients (Grathwohl et al., 2021) may be applied to generate samples.

To sample from $p_\theta(x, y)$ we can apply block Metropolis-Within-Gibbs (MWG) using x and y each as their own block. MWG works similarly to Gibbs sampling where we iteratively sample $y_{t+1} \sim p_\theta(y|x_t)$ and then $x_{t+1} \sim p_\theta(x|y_{t+1})$. Instead of sampling exactly from the conditional (which we cannot do since the conditional is unnormalized), we update our current sample y_t using a Markov transition kernel applied to $p(y|x_t)$ and then update x_t likewise. In practice, we hold x_t fixed, then update y_t using one step of Gibbs-With-Gradients to obtain y_{t+1}. We then hold y_{t+1} fixed and apply one step of Langevin dynamics to sample x_{t+1}.

It is likely that more involved approaches (Zhou, 2019) could lead to improved performance or more efficient sampling but we found this simple approach to work well for our applications. In the following we will refer to the Markov transition kernel for 1 step of Langevin Dyanmics applied to $p_\theta(x|y_t)$ as $\mathcal{T}_c(x_{t+1}|x_t, y_t)$ and we refer to the Markov transition kernel for 1 step of Gibbs-With-Gradients applied to $p_\theta(y|x_t)$ as $\mathcal{T}_d(y_{t+1}|y_t, x_t)$. Pseudo-code for our proposed joint sampler can be seen in Algorithm 1.

Algorithm 1 Joint Sampling

Input: EBM $p_\theta(x, y) \propto e^{f_\theta(x,y)}$, initial distributions $p_0(x), q_0(y)$, number of steps T
Output: Approximate samples $x_T, y_T \sim p_\theta(x, y)$
Initialize samples $x_0 \sim p_0(x), y_0 \sim q_0(y)$
$t = 0$
while $t < T$ **do**
  Sample $y_{t+1} \sim \mathcal{T}_d(y_{t+1}|y_t, x_t)$ {Gibbs-With-Gradients}
  Sample $x_{t+1} \sim \mathcal{T}_c(x_{t+1}|x_t, y_{t+1})$ {Langevin Dynamics}
  $t = t + 1$
end while
return x_T, y_T

5 Evaluation

Prediction In classification tasks, predictions are typically made by returning the configuration of y which maximizes $p_\theta(y|x)$. When y is high-dimensional and its dimensions are correlated, performing this maximization is not straightforward. In these settings, we propose to use MCMC to instead sample from $p_\theta(y|x)$ by repeatedly applying the transition kernel $\mathcal{T}_d(y_{t+1}|y_t, x_t)$. We refer to this method of prediction as "sample-based".

As a useful comparison, in some of our experiments y is sufficiently low-dimensional that it is tractable to marginalize over all configurations and obtain the exact $y' = \arg\max_y p_\theta(y|x)$. We refer to this method of prediction as "energy-based".

Conditional Sampling Conditional sampling can be achieved through repeated application of the Markov transition kernel $\mathcal{T}_c(x_{t+1}|x_t, y_t)$. Sometimes, we may condition on only a subset of the supervision. In this case, we apply MWG as in Algorithm 1 with a modified Markov transition kernel $\mathcal{T}_d(y_{t+1}|y_t, x_t)$. This kernel updates the subset of supervision which has not been conditioned on and holds the subset of supervision that is being conditioned on constant.

6 Experiments

We demonstrate that our approach is able to train joint models which can accurately predict the labels and generate conditional samples. Throughout our experiments, we use models of the form

$$f_\theta(x, y) = \texttt{neural\_net}(\texttt{concat}(h_\phi(x), y); \theta)$$

where y is a concatenation of one-hot vectors for each of the attributes, and h_ϕ encodes data features.

The appendix includes details of the dataset preprocessing, architectures, additional samples, and density visualizations (where applicable) for all experiments.

6.1 A Toy Model

First, we demonstrate our approach on a toy dataset generated with NumPy (Harris et al., 2020) consisting of eight gaussian modes on a circle. The supervision consists of three binary attributes, each of which defines an even partition of the eight modes. The label y has six dimensions as it is the concatenation of three 2-dimensional one-hot vectors.

In Figure 1, we plot samples conditioned on either configuration of one of these attributes. This model obtains $> 99\%$ predictive accuracy of the three attributes both with sample-based and energy-based prediction.

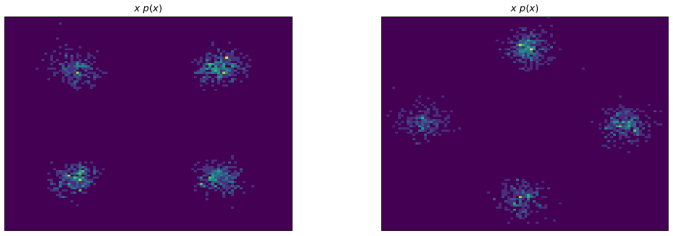

Figure 1: Conditional samples on toy data.

6.2 Conditional Generative Modeling on MNIST

We scale our approach to jointly model the distribution of 28×28 MNIST handwritten digits and their 10 possible labels. Figure 2 shows our model obtains both high-quality unconditional samples and high-quality class-conditional samples. Additionally, our model achieves 93% sampling-based accuracy and 98% energy-based accuracy on the test set.

6.3 High-dimensional Supervision on dSprites

Last, we apply our method to the dSprites dataset (Matthey et al., 2017). This dataset consists of 64×64 (downsampled to 32×32) binary images of several shapes at different sizes, orientations, and positions. We restrict this dataset to the ellipse shape and orientations that are multiples of 90 degrees. The attributes are the horizontal and vertical coordinates of the shape within the image,

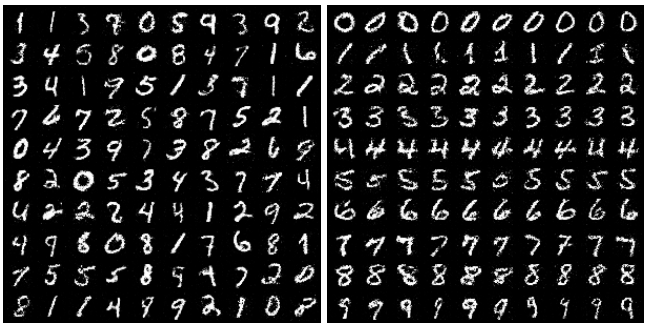

Figure 2: Unconditional samples (left) and conditional samples (right) on MNIST.

each of which has 16 possibilities. This results in a 32 dimensional one-hot attribute vector y with 256 possible configurations.

In Figure 3 we demonstrate our model can conditionally sample the horizontal and vertical coordinates. In the left figure, each row conditions on a different horizontal coordinate. As we move from the top row to the bottom row, samples are conditioned to be further to the right. In the right figure, each column conditions on a different vertical coordinate. As we move from the leftmost column to the rightmost column, samples are conditioned to be further to the bottom. Our model achieves 89% sampling-based accuracy.

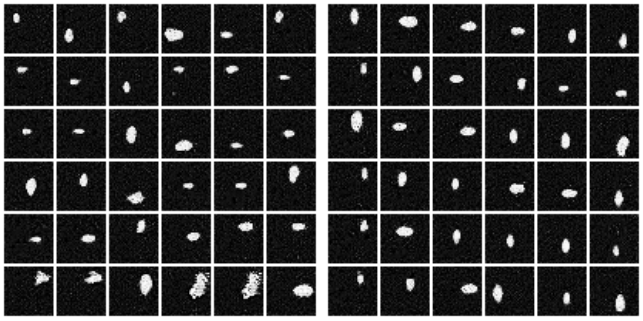

Figure 3: Horizontal (left, row-fixed) and vertical (right, column-fixed) conditioning on dSprites.

7 Related Work

Structured prediction problems have been a key application of EBMs. In this setting we wish to make predictions of highly structured y given inputs x. To capture complex correlations, this is often phrased in an energy-minimization framework (Gygli et al., 2017; Belanger et al., 2017) which has many similarities to the MCMC sampling we use. We believe our approach could be applied to these applications to add many of the benefits reported in Grathwohl et al. (2019).

Next are works that explore the unique capabilities of EBMs for challenging tasks such as continual learning (Li et al., 2020) and compositional generation (Du et al., 2020a). We believe the techniques and architectures presented in this work could extend the range of problems these ideas can be applied to.

8 Conclusions and Next Steps

In this work we developed an approach to model the joint distribution of data and supervision using EBMs. This approach allows us to train on data with limited or partial supervision, and allows us to obtain any of the conditional distributions. Our work is enabled by recent advances in discrete and continuous sampling techniques for EBMs. We demonstrated our approach simultaneously achieves accurate prediction and conditional sampling. This work is a first step, and in the future we hope to scale our approach to larger datasets and incorporate more complex structured supervision.

REFERENCES

David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pp. 429–439. PMLR, 2017.

Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019.

Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation and inference with energy based models. In *NeurIPS 2020*, 2020a.

Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Improved contrastive divergence training of energy based models. *arXiv preprint arXiv:2012.01316*, 2020b.

Yilun Du, Joshua Meier, Jerry Ma, Rob Fergus, and Alexander Rives. Energy-based models for atomic-resolution protein conformations. *arXiv preprint arXiv:2004.13167*, 2020c.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Will Grathwohl, Jacob Kelly, Milad Hashemi, Mohammad Norouzi, Kevin Swersky, and David Duvenaud. No mcmc for me: Amortized sampling for fast and stable training of energy-based models. *arXiv preprint arXiv:2010.04230*, 2020.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris J. Maddison. Oops i took a gradient: Scalable sampling for discrete distributions, 2021.

Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep value networks learn to evaluate and iteratively refine structured outputs. In *International Conference on Machine Learning*, pp. 1341–1351. PMLR, 2017.

Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Ryuichiro Hataya, Hideki Nakayama, and Kazuki Yoshizoe. Graph energy-based model for substructure preserving molecular design. *arXiv preprint arXiv:2102.04600*, 2021.

Tianxing He, Bryan McCann, Caiming Xiong, and Ehsan Hosseini-Asl. Joint energy-based model training for better calibrated natural language understanding models. *arXiv preprint arXiv:2101.06829*, 2021.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

John Ingraham, Adam J Riesselman, Chris Sander, and Debora S Marks. Learning protein structure with a differentiable simulator. In *ICLR*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Shuang Li, Yilun Du, Gido M van de Ven, Antonio Torralba, and Igor Mordatch. Energy-based models for continual learning. *arXiv preprint arXiv:2011.12216*, 2020.

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

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

Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5272–5280, 2020a.

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

Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7:1, 2017.

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning*, pp. 1064–1071, 2008.

Michalis K Titsias and Christopher Yau. The hamming ball sampler. *Journal of the American Statistical Association*, 112(520):1598–1611, 2017.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688. Citeseer, 2011.

Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative convnet. In *International Conference on Machine Learning*, pp. 2635–2644. PMLR, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

Giacomo Zanella. Informed proposals for local mcmc in discrete spaces. *Journal of the American Statistical Association*, 115(530):852–865, 2020.

Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

Stephen Zhao, Jörn-Henrik Jacobsen, and Will Grathwohl. Joint energy-based models for semi-supervised classification. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2020.

Guangyao Zhou. Mixed hamiltonian monte carlo for mixed discrete and continuous variables. *arXiv preprint arXiv:1909.04852*, 2019.

A JOINT ENERGY-BASED MODELS

A.1 MAXIMUM LIKELIHOOD GRADIENT

We give a short proof of Equation 8 adapted from Nijkamp et al. (2020b). Assuming we are only given x', then we can see

$$\begin{aligned}
\nabla_\theta \log p_\theta(x') &= \frac{\nabla_\theta p_\theta(x')}{p_\theta(x')} \\
&= \frac{1}{p_\theta(x')} \nabla_\theta \int_{\mathcal{Y}} p_\theta(x', y) dy && \text{(marginalize)} \\
&= \frac{1}{p_\theta(x')} \int_{\mathcal{Y}} \nabla_\theta p_\theta(x', y) dy && \text{(differentiate under the integral)} \\
&= \frac{1}{p_\theta(x')} \int_{\mathcal{Y}} p_\theta(x', y) \nabla_\theta \log p_\theta(x', y) dy && \text{(log-derivative trick)} \\
&= \int_{\mathcal{Y}} \frac{p_\theta(x', y)}{p_\theta(x)} \nabla_\theta \log p_\theta(x', y) dy \\
&= \int_{\mathcal{Y}} p_\theta(y|x') \nabla_\theta \log p_\theta(x', y) dy \\
&= \mathbb{E}_{p_\theta(y|x')} \left[\nabla_\theta \log p_\theta(x', y) \right] \\
&= \mathbb{E}_{p_\theta(y|x')} \left[\nabla_\theta f_\theta(x', y) \right] - \mathbb{E}_{p_\theta(x,y)} \left[\nabla_\theta f_\theta(x, y) \right] && (9)
\end{aligned}$$

where the last line is obtained by applying Equation 2 to the gradient of the joint log density $\nabla_\theta \log p_\theta(x', y)$.

A.2 JEM

We give a brief background on JEM (Grathwohl et al., 2019) for the uninitiated reader as an informative contrast to our approach.

Typically, a K-class classification task is solved by fitting a model of $p_\theta(y|x)$. This is usually done with a function $f_\theta : \mathcal{X} \to \mathbb{R}^K$ where

$$p_\theta(y|x) = \frac{e^{f_\theta(x)[y]}}{\sum_{y'=1}^{K} e^{f_\theta(x)[y']}} \tag{10}$$

and $y \in \{1, \dots, K\}$ is the index representing the class-label. Grathwohl et al. (2019) notices that the same function f_θ can be used as well to define an energy-based model of the joint as

$$p_\theta(x, y) = \frac{e^{f_\theta(x)[y]}}{Z(\theta)}, \qquad Z(\theta) = \sum_y \int e^{f_\theta(x)[y]} dx. \tag{11}$$

With this model, the label can be marginalized out to reveal an EBM for the unconditional data distribution

$$p_\theta(x) = \frac{\sum_y e^{f_\theta(x)[y]}}{Z(\theta)} \tag{12}$$

and, when computing the conditional we find

$$p_\theta(y|x) = \frac{p_\theta(x, y)}{p_\theta(x)} = \frac{e^{f_\theta(x)[y]}}{\sum_{y'=1}^{K} e^{f_\theta(x)[y']}} \tag{13}$$

which is identical to the conditional distribution parameterized by standard classifiers. JEM models are trained to maximize $\log p_\theta(x) + \log p_\theta(y|x)$ as defined in Equations 12 and 13. The first term is optimized using the gradient estimator of Equation 2.

B EXPERIMENTAL DETAILS

We use the Adam optimizer with default parameters $\beta_1 = 0.9, \beta_2 = 0.999$ (Kingma & Ba, 2014) throughout our experiments. In Table 1, Table 2, and Table 3 we provide hyperparameters for each of our experiments. The main hyperparameters we tuned were the step size ϵ and temperature λ for Langevin dynamics.

Sampling During training, we use Persistent Contrastive Divergence (PCD) and a replay buffer with a standard size of 10000 and replacement rate of 0.05 throughout all of our experiments (Du & Mordatch, 2019). At test time, we generate samples from random noise and do not use the replay buffer.

Sampling Noise The initial distribution for Langevin dynamics is uniform with bounds equal to the per-dimension minimum and maximum values of one batch of data. The initial distribution for Gibbs-With-Gradients is set to be uniform over all possible configurations on dSprites, and with probabilities proportional to the number of occurrences in one batch of data for the toy data and MNIST experiments.

Exponential Moving Average We keep an exponential moving average (EMA), as in Du et al. (2020b), of the training parameters θ being optimized by applying the following update after each training iteration:

$$\hat{\theta} = \mu \cdot \hat{\theta} + (1 - \mu) \cdot \theta.$$

We initialize $\hat{\theta}_0 = \theta_0$ and set $\mu = 0.99$ in our experiments. At test time we use the parameters $\hat{\theta}$.

Type	Hyperparameter	Value
Optimization	learning rate	10^{-4}
	warmup	10^3
	batch size	1000
	iterations	5500
Sampling	ϵ (Langevin)	10^{-1}
	λ (Langevin)	0.5
Sampling (Train)	steps	20
	buffer size	10000
	replacement rate	0.05
Sampling (Test)	steps	200
	sampled-based prediction samples	10
	μ (EMA)	0.99

Table 1: Hyperparameters for toy data.

Type	Hyperparameter	Value
Optimization	learning rate	10^{-4}
	warmup	10^4
	batch size	100
	iterations	110000
Sampling	ϵ	3×10^{-3}
	λ	1.5×10^{-6}
Sampling (Train)	steps	40
	buffer size	10000
	replacement rate	0.05
Sampling (Test)	steps	200
	sampled-based prediction samples	10
	μ EMA	0.99

Table 2: Hyperparameters for MNIST.

Type	Hyperparameter	Value
Optimization	learning rate	10^{-5}
	warmup	10^4
	batch size	100
	iterations	180000
Sampling	ϵ	3×10^{-3}
	λ	4.5×10^{-7}
Sampling (Train)	steps	40
	buffer size	10000
	replacement rate	0.05
Sampling (Test)	steps	200
	sampled-based prediction samples	10
	μ EMA	0.99

Table 3: Hyperparameters for dSprites.

B.1 Toy Data

B.1.1 Architecture

The following architecture is implemented in PyTorch (Paszke et al., 2019).

```
Sequential(
  (0): Linear(in_features=8, out_features=256, bias=True)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=256, out_features=256, bias=True)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=256, out_features=256, bias=True)
  (5): ELU(alpha=1.0)
  (6): Linear(in_features=256, out_features=1, bias=True)
)
```

B.2 MNIST

B.2.1 Data Processing

We applied standard dequantization

$$\tilde{x} = \frac{x * 255 + u}{256}, u \sim \mathcal{U}[0, 1]$$

followed by the standard logit transformation

$$\hat{x} = \tilde{x} * (1 - 2 \times 10^{-6}) + 10^{-6}$$
$$\overline{x} = \log\left(\frac{\hat{x}}{1 - \hat{x}}\right)$$

B.2.2 ARCHITECTURE

The following architecture is implemented in PyTorch (Paszke et al., 2019).

```
Sequential(
  (0): Linear(in_features=794, out_features=1000, bias=True)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=1000, out_features=500, bias=True)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=500, out_features=500, bias=True)
  (5): ELU(alpha=1.0)
  (6): Linear(in_features=500, out_features=250, bias=True)
  (7): ELU(alpha=1.0)
  (8): Linear(in_features=250, out_features=250, bias=True)
  (9): ELU(alpha=1.0)
  (10): Linear(in_features=250, out_features=250, bias=True)
  (11): ELU(alpha=1.0)
  (12): Linear(in_features=250, out_features=1, bias=True)
)
```

B.3 DSPRITES

B.3.1 DATA PROCESSING

We downsampled images from their original 64×64 size to 32×32. We binned the horizontal and vertical attributes from 32 options each to 16 options each. We applied a significant amount of dequantization to the images with the following function

$$\tilde{x} = \frac{x * 7 + u}{8}, u \sim \mathcal{U}[0, 1]$$

followed by the same logit transformation as was used for MNIST.

The test set was constructed by taking random samples from the dataset. The final processed dataset has 46,652 examples in the training set, and 2,500 examples in the test set.

B.3.2 ARCHITECTURE

We use the following CNN architecture for $h_\phi(x)$, implemented in PyTorch (Paszke et al., 2019).

```
(cnn): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): SiLU(inplace=True)
    (2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): SiLU(inplace=True)
    (4): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (5): SiLU(inplace=True)
```

```
        (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (7): SiLU(inplace=True)
        (8): Conv2d(128, 16, kernel_size=(1, 1), stride=(1, 1))
        (9): Flatten(start_dim=1, end_dim=-1)
    )
```

where `SiLU` is the Sigmoid Linear Unit, also known as the Swish activation function (Hendrycks & Gimpel, 2016; Elfwing et al., 2018; Ramachandran et al., 2017).

The convolutional neural network has 256 flattened output features which are concatenated to the 32 dimensional supervision and fed through the following MLP.

```
    (mlp): Sequential(
        (0): Linear(in_features=288, out_features=256, bias=True)
        (1): SiLU(inplace=True)
        (2): Linear(in_features=256, out_features=256, bias=True)
        (3): SiLU(inplace=True)
        (4): Linear(in_features=256, out_features=1, bias=True)
    )
```

C ADDITIONAL SAMPLES

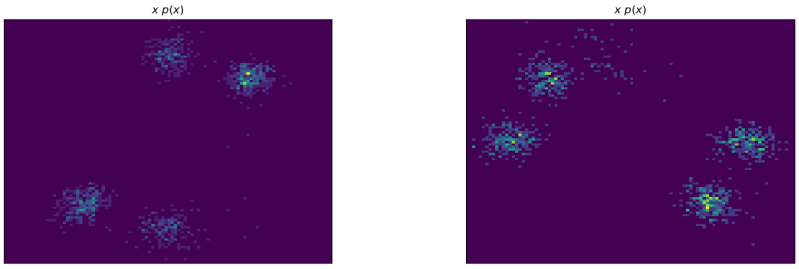

Figure 4: Class-conditional samples on toy data.

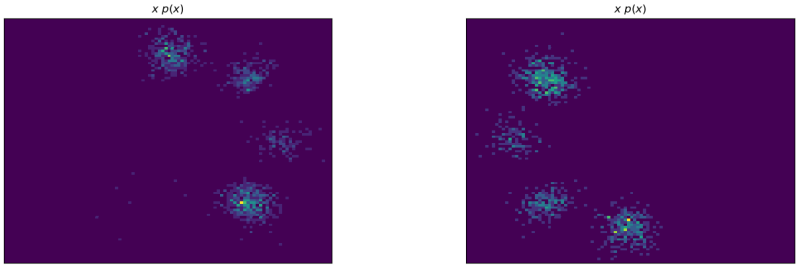

Figure 5: Class-conditional samples on toy data.

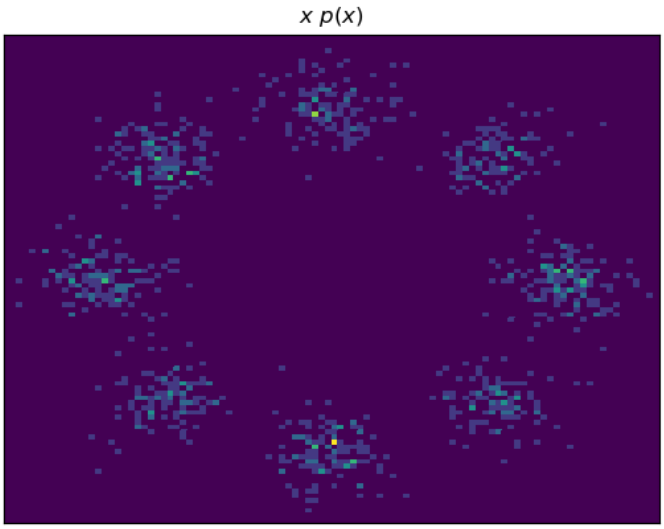

Figure 6: Unconditional samples on toy data.

Figure 7: Class-conditional densities on toy data.

Figure 8: Class-conditional densities on toy data.

Figure 9: Class-conditional densities on toy data.

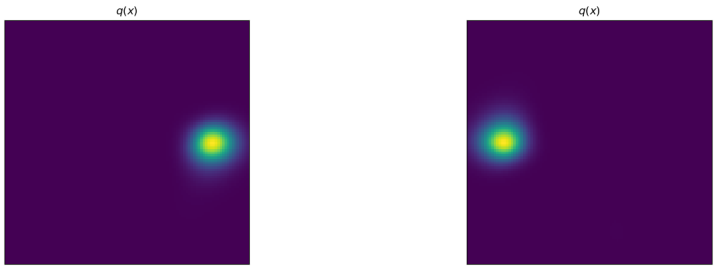

Figure 10: Class-conditional densities on toy data.

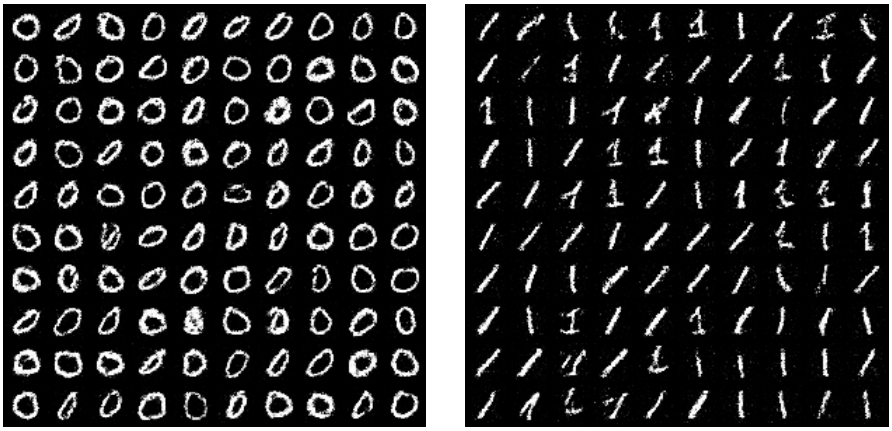

Figure 11: Class-conditional samples on MNIST.

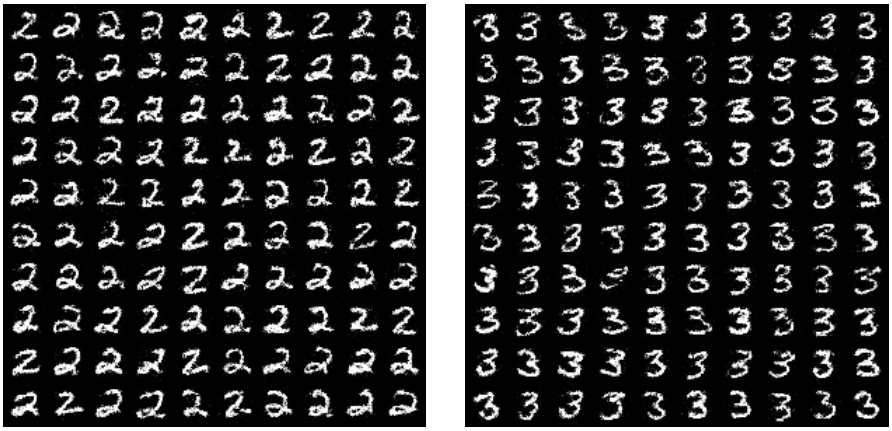

Figure 12: Class-conditional samples on MNIST.

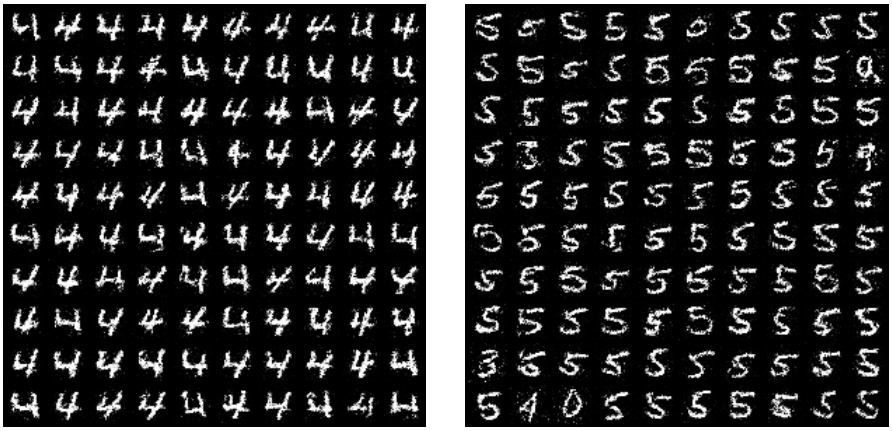

Figure 13: Class-conditional samples on MNIST.

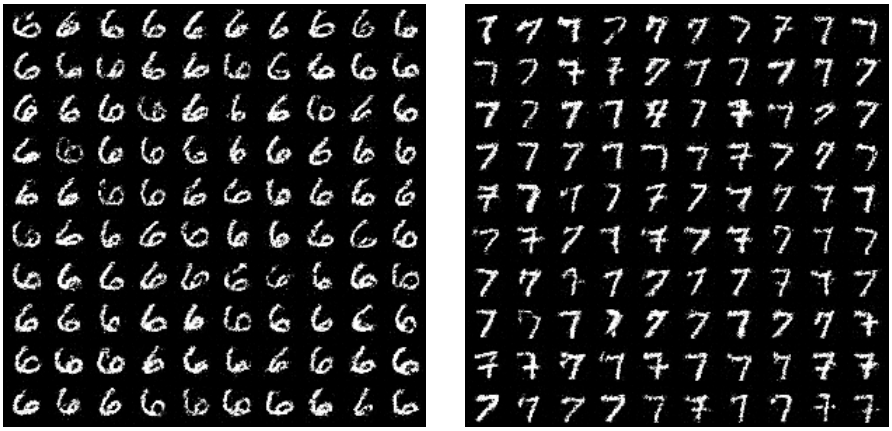

Figure 14: Class-conditional samples on MNIST.

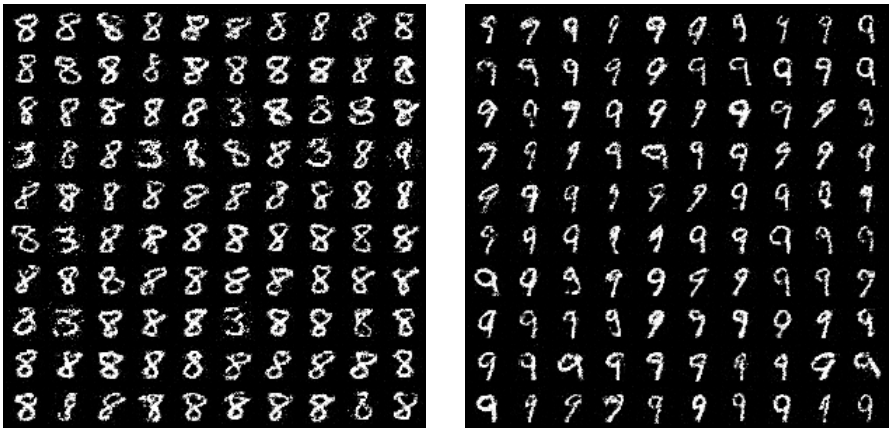

Figure 15: Class-conditional samples on MNIST.

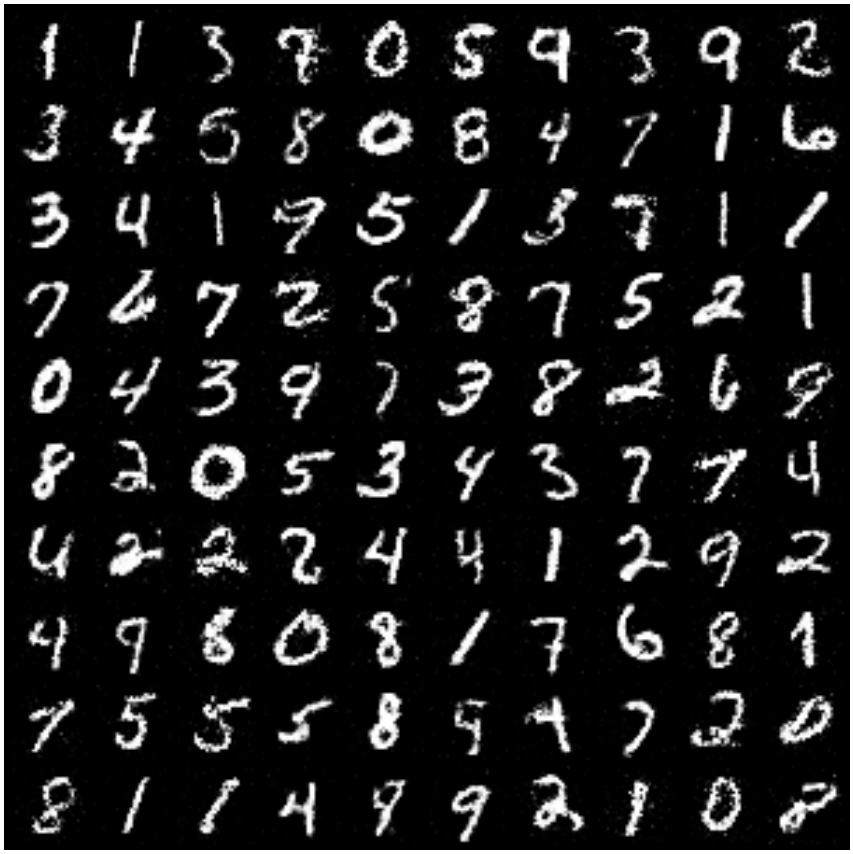

Figure 16: Unconditional samples on MNIST.

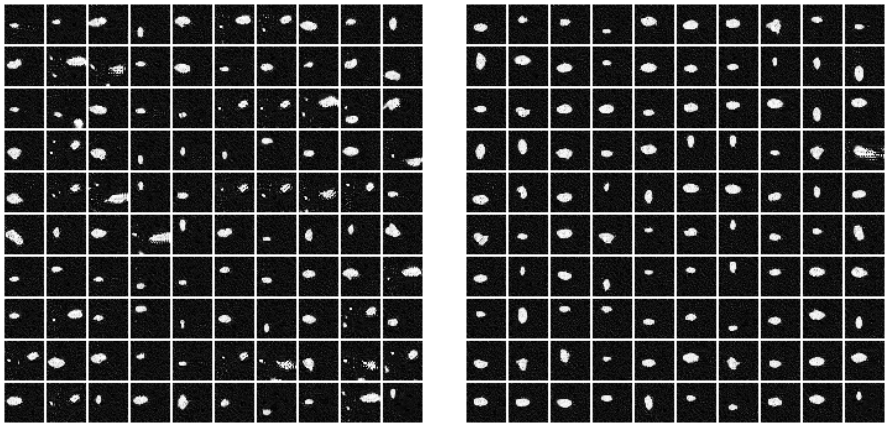

Figure 17: Class-conditional samples on dSprites. Left: Horizontal coordinate 0. Right: Horizontal coordinate 3.

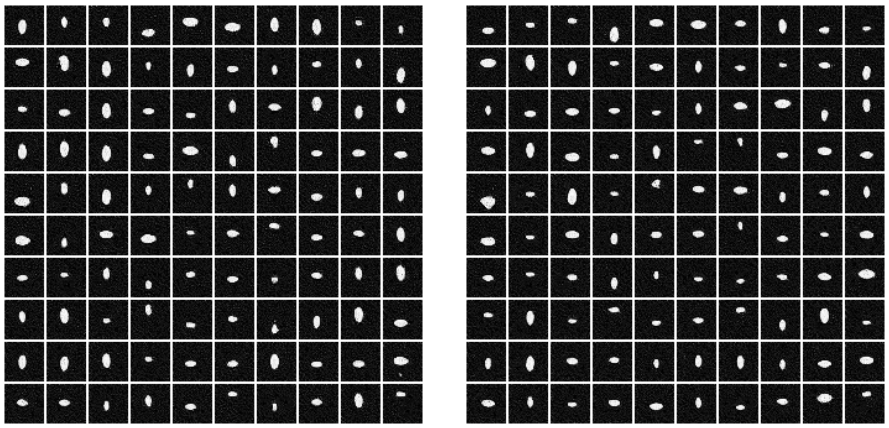

Figure 18: Class-conditional samples on dSprites. Left: Horizontal coordinate 6. Right: Horizontal coordinate 9.

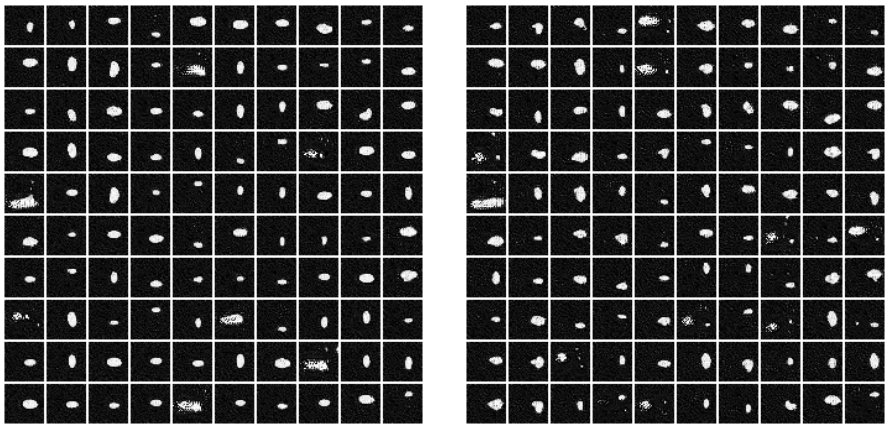

Figure 19: Class-conditional samples on dSprites. Left: Horizontal coordinate 12. Right: Horizontal coordinate 15.

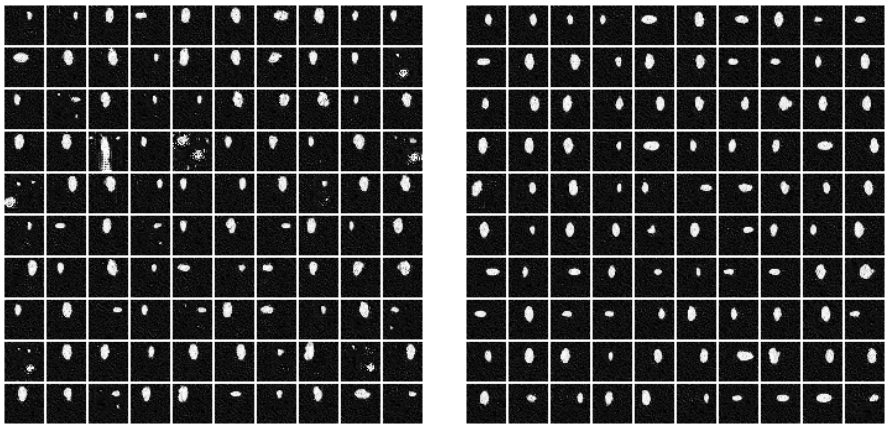

Figure 20: Class-conditional samples on dSprites. Left: Vertical coordinate 0. Right: Vertical coordinate 3.

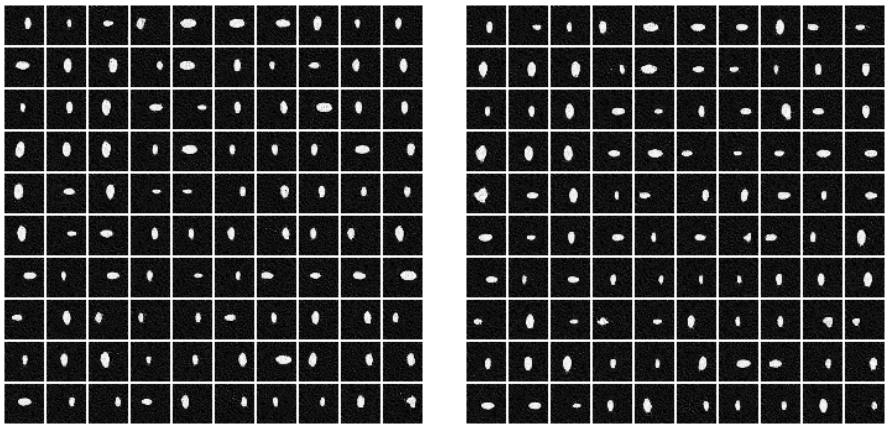

Figure 21: Class-conditional samples on dSprites. Left: Vertical coordinate 6. Right: Vertical coordinate 9.

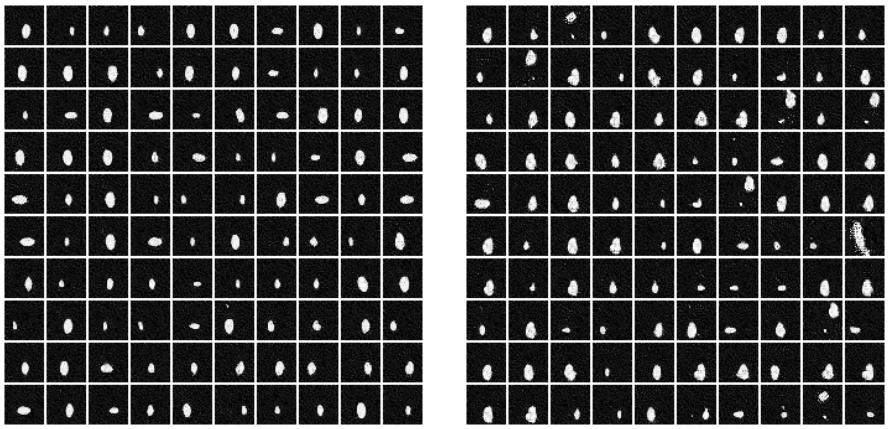

Figure 22: Class-conditional samples on dSprites. Left: Vertical coordinate 12. Right: Vertical coordinate 15.

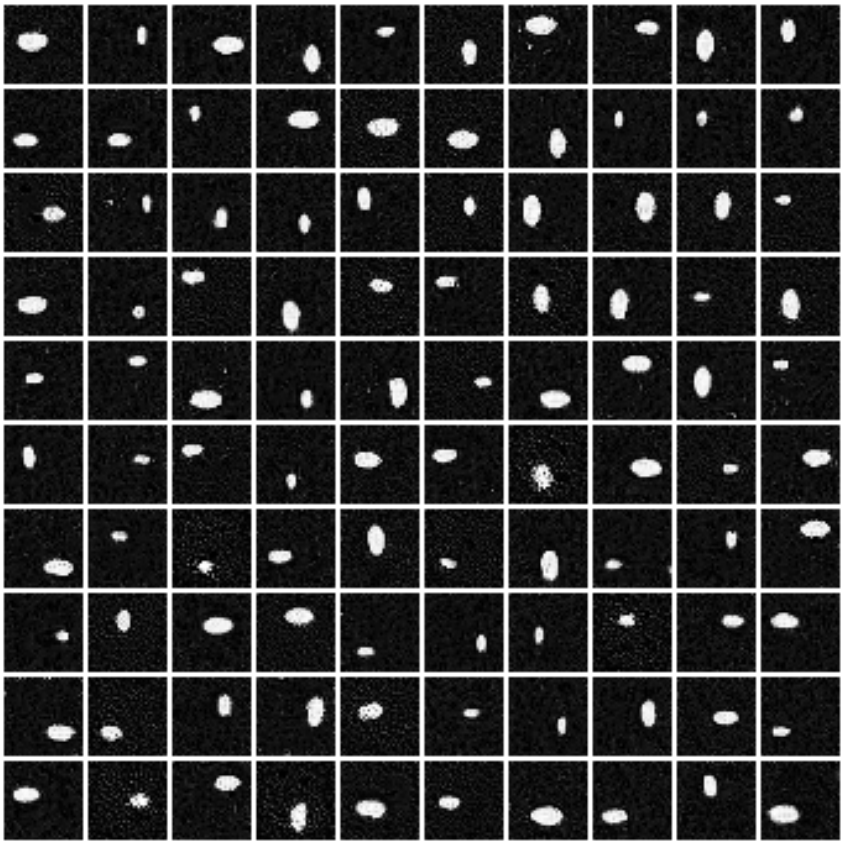

Figure 23: Unconditional samples on dSprites.