Advances in Deep Generative Modeling With Applications to Image Generation and Neuroscience

Gabriel Loaiza Ganem

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2019

ABSTRACT

Advances in Deep Generative Modeling With Applications to Image Generation and Neuroscience

Gabriel Loaiza Ganem

Deep generative modeling is an increasingly popular area of machine learning that takes advantage of recent developments in neural networks in order to estimate the distribution of observed data. In this dissertation we introduce three advances in this area. The first one, Maximum Entropy Flow Networks, allows to do maximum entropy modeling by combining normalizing flows with the augmented Lagrangian optimization method. The second one is the continuous Bernoulli, a new [0,1]-supported distribution which we introduce with the motivation of fixing the pervasive error in variational autoencoders of using a Bernoulli likelihood for non-binary data. The last one, Deep Random Splines, is a novel distribution over functions, where samples are obtained by sampling Gaussian noise and transforming it through a neural network to obtain the parameters of a spline. We apply these to model texture images, natural images and neural population data, respectively; and observe significant improvements over current state of the art alternatives.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

Going through a PhD has been a fulfilling and rewarding experience, during which I have learned an enormous amount. This was in good part due to my advisor, John Cunningham, whom I want to thank for his continuous guidance, support and patience, which were all fundamental to the completion of this dissertation. I have fond memories of our weekly discussions, which were always intellectually stimulating. Thank you, John!

I also want to thank my dissertation committee members, Liam Paninski, David Blei, David Knowles and Nikolaus Kriegeskorte for reading this work and helping to improve it; as well as Columbia University and its Statistics Department, as this work would not have been possible without their financial support.

I was very lucky to interact with many fantastic people at Columbia from whom I learned a lot, among them Yuanjun Gao, Sean Perkins, Karen Schroeder, Mark Churchland, Andrew Miller, Evan Archer, Scott Linderman, Christian Naesseth, Sean Bittner, Taiga Abe, Shreya Saxena, Joshua Glaser, Kenneth Kay, Peter Orbanz, Arian Maleki and Andrew Gelman.

I also want to thank Papá, Mamá, Ignatz, Añu, Abuelo, and Tita as well as Omar, Sylvain, Sebastián and Nathan for their constant support and encouragement.

Finally, I also want to thank my undergraduate professors, specially Alberto Tubilla, Carlos Bosch and Zeferino Parada for giving me the proper foundations to pursue graduate studies, and helping me do so.

To my parents, Gabriel and Claudia.

Chapter 1

Introduction

The goal of generative modeling is, given some data, to estimate the distribution that generated it. In a way this is the most general statistical problem, as any question of interest about a set of random variables can be formulated in terms of their joint distribution. The price to pay for solving such a general problem is that it is hard to do so, for example, kernel density estimation [Rosenblatt, 1956, Parzen, 1962] suffers from very slow convergence rates in high dimensions. Hierarchical and graphical models [Gelman and Hill, 2006, Wainwright and Jordan, 2008] have been very successful approaches, as they allow to model complex structure in the data, which can help alleviate the curse of dimensionality. Significant effort has been devoted to performing inference on these models [Dempster et al., 1977, Jordan et al., 1999, Murphy et al., 1999].

On the other side of the machine learning spectrum, thanks to back-propagation [Rumelhart et al., 1988] neural networks have been remarkably successful in supervised tasks such as classification [LeCun et al., 1998, Krizhevsky et al., 2012, He et al., 2016]. While we do not yet fully understand why these incredibly flexible models do not overfit [Zhang et al., 2016], they have been empirically shown to significantly outperform alternatives. More recently, neural networks have also been used, at the cost of some parameter interpretability but to much empirical success, in generative

modeling. The most popular frameworks for deep generative models are Generative Adversarial Networks (GAN) [Goodfellow et al., 2014], Variational Autoencoders (VAE) [Kingma and Welling, 2013] and to a lesser degree, Normalizing Flows [Dinh et al., 2014, Rezende and Mohamed, 2015]. All these methods take advantage of the huge flexibility of neural networks and use it for generative modeling.

In order to fully take advantage of neural networks, deep generative modeling requires large amounts of data. As such, image modeling is among the most popular applications of deep generative modeling, as large image datasets such as MNIST [LeCun, 1998], CIFAR-10 [Krizhevsky et al., 2009] and ImageNet [Russakovsky et al., 2015] are easily available. Deep generative modeling has also been applied in areas such as image generation [Gregor et al., 2015, Denton et al., 2015], text generation [Hu et al., 2017], text to image synthesis [Zhang et al., 2017], neuroscience [Gao et al., 2016], chemistry [Gómez-Bombarelli et al., 2018] and music generation [Mogren, 2016], among others. Recent developments in microelectrode arrays for neural recordings [Ballini et al., 2014] allow to measure population level spiking activity, resulting in the type of large datasets that are amenable to deep generative modeling. These datasets, after spike sorting [Lee et al., 2017], contain simultaneous spiking times (i.e. times when neurons fire) over large populations of neurons. Analyzing these datasets is a challenging task that can help to further our understanding of the brain, and doing so with neural networks and generative modeling is an exciting research area. In what follows we give a brief description of the main deep generative modeling frameworks.

GAN work by transforming Gaussian noise through a neural network, called the generator. The idea is that the distribution of the transformed variable should match the true data generating distribution. Computing the distribution of the transformed variables is not tractable, so estimating the parameters of the generator through maximum likelihood cannot be done. In order to circumvent this issue, a second neural network, called the discriminator, is introduced. The discriminator is a binary classifier that tries to differentiate between true and generated samples. The generator and dis-

criminator are trained together on a saddle point optimization (often called adversarial training) objective, which the generator tries to minimize and the discriminator to maximize. Goodfellow et al. [2014] show that if the discriminator has infinite capacity, the adversarial objective is equivalent to minimizing the Jensen-Shannon divergence between real and generated data. There are extensions where the minimized divergence is changed [Nowozin et al., 2016, Arjovsky et al., 2017]. However, GAN suffer from several issues: First, the capacity needed for the discriminator for the adversarial objective to be close enough to the divergences that it is supposedly approximating is not achievable in practice, as shown by Arora et al. [2017]. Second, in practice the adversarial training objective is optimized with stochastic gradient methods, which have no guarantees (not even local ones, unlike for minimization problems) for this type of optimization problems. Third, despite some efforts, evaluating the performance of a trained generator remains challenging [Salimans et al., 2016, Arora and Zhang, 2017, Heusel et al., 2017].

VAE proceed in a similar fashion to GAN by transforming noise (usually Gaussian) through a neural network. The difference is that the output of the network is not the data directly, but the parameters of a distribution which we assume generated the data. This allows to interpret the initial noise as local latent variable, and approximate posterior inference can be performed on it. This allows VAE to not only posit a generative model for the data of interest, but to carry out dimensionality reduction at the same time (there are GAN efforts to achieve this [Donahue et al., 2016], but they still suffer from all the GAN issues). While the likelihood still cannot be computed, a lower bound called the ELBO is tractable, which allows to estimate the model parameters and perform approximate posterior inference, thus resulting in a more sound procedure than GAN, both from a probabilistic and from an optimization perspective. Furthermore, there are extensions of VAE that allow to regain some of the interpretability that is lost by using neural networks, see for example Gao et al. [2016], Johnson et al. [2016].

Normalizing flows are like GAN in that they transform Gaussian noise directly into the data through a neural network. However, the neural network is carefully designed in such a way that the density of the transformed variable can be efficiently computed. This is achieved by constructing invertible neural networks and using the change of variable theorem. The result is that adversarial training is no longer necessary and the network can be trained with maximum likelihood. While, unlike in VAE, the likelihood can be computed exactly and not just a lower bound, normalizing flows do not perform dimensionality reduction as the "latent" variables (which can actually be computed exactly) need to have the same dimension as the observed data due to the imposed invertibility of the neural network.

GAN have had impressive empirical success, particularly for image modeling [Denton et al., 2015, Radford et al., 2015], where conventional wisdom says that GAN produces sharper images than the alternatives, particularly VAE, which produces blurry images. However, recent advances in VAE [Gulrajani et al., 2016] and normalizing flows [Kingma and Dhariwal, 2018] have helped breach this gap in empirical performance. Furthermore, VAE and normalizing flows are much more principled models than GAN, as they train on an objective that has a valid and well understood probabilistic meaning, while also avoiding the GAN pitfalls of unstable training and difficulty to evaluate performance.

In this dissertation we make three contributions to the area of deep generative modeling. The first one, Maximum Entropy Flow Networks (MEFN), uses normalizing flows for maximum entropy modeling. Maximum entropy modeling is a modeling framework proposed by Jaynes [1957] that, while related to maximum likelihood, is neither frequentist nor Bayesian and assumes that the data came from the maximally uninformative distribution out of the set of distributions which obey certain prespecified constraints. Using normalizing flows allows density evaluation and thus entropy estimation, which we combine with the augmented Lagrangian method to optimize the resulting optimization problem. We then apply MEFN for generating texture

images, and achieve state of the art results.

The second one, the continuous Bernoulli, is a new distribution supported on the $[0, 1]$ interval which we introduce to correct a common error in the implementation of Bernoulli VAE of modeling real valued (almost binary) data as binary. We fully characterize this distribution and show that using it significantly improves VAE performance.

The third and last one is Deep Random Splines (DRS), a novel distribution over functions which we use to model the firing rate of neural populations with a VAE-type model. DRS work by sampling Gaussian noise and transforming it through a neural network, obtaining the parameters of a spline. Those parameters are then run through the method of alternating projections in order to ensure the splines satisfy any shape constraints of interest, such as the nonnegativity that is needed for modeling firing rates. Our model achieves significantly better dimensionality reduction than competing alternatives.

The rest of this dissertation is organized as follows: on chapter 2 we cover the necessary background material, on chapters 3 to 5 we introduce MEFN, the continuous Bernoulli and DRS, respectively. Note that chapters 3 to 5 use different notation. While common elements follow the same notation across chapters (e.g. θ always denotes generative parameters), some notations are different. We do this as it should always be clear from context what we are referring to, and this avoids running out of common symbols and having to use more obscure and uncommon notation that might confuse a reader who is familiar with the corresponding literature. The subsections of chapter 2 follow the notation of the chapter for which they are background. Finally, we conclude in chapter 6. The code used for this dissertation can be found at `https://github.com/gabloa`.

Chapter 2

Background

In order to be as self-contained as possible, we introduce all the necessary background material in this chapter. We also include some material that is not directly used in this dissertation with the goal of giving a short overview of the areas to which we make contributions in later chapters.

2.1 Maximum Entropy Modeling and the Gibbs Distribution

In chapter 3 we will propose novel methodology to solve the maximum entropy problem, which we define in this section. Consider a continuous random variable $X \in \mathscr{X} \subseteq \mathbb{R}^D$ with density p, where p has differential entropy $H(p) = -\int p(z) \log p(z) dz$ and support $\text{supp}(p)$. The goal of ME modeling is to find, and then be able to easily sample from, the maximum entropy distribution given a set of moment and support constraints, namely the solution to:

$$\begin{aligned} p^* = \underset{p}{\text{maximize}} \;\; & H(p) \\ \text{subject to } & E_{X \sim p}[T(X)] = 0 \\ & \text{supp}(p) = \mathscr{X} \end{aligned} \tag{2.1}$$

where $T(x) = (T_1(x), ..., T_m(x)) : \mathscr{X} \to \mathbb{R}^m$ is the vector of known (assumed sufficient) statistics, and $\mathscr{X}$ is the given support of the distribution. Under standard regularity conditions, the optimization problem can be solved by Lagrange multipliers, yielding an exponential family p^* of the form:

$$p^*(x) \propto e^{\eta^{*\top} T(x)} \mathbb{1}(x \in \mathscr{X}) \tag{2.2}$$

where $\eta^* \in \mathbb{R}^m$ is the choice of natural parameters of p^* such that $E_{p^*}[T(X)] = 0$ (this distribution is often called the Gibbs distribution). To see this informally, consider the Lagrangian functional:

$$J(p, \eta) = -\int p(x) \log p(x) dx + \eta_0 \Big(\int p(x) dx - 1 \Big) + \sum_{j=1}^{m} \eta_j \int p(x) T_j(x) dx \tag{2.3}$$

where the η terms are the Lagrangian multipliers (η_0 corresponds to the implicit constraint that p is a density) and the support term is omitted (as is the nonnegativity constraint, as the solution will be nonnegative regardless). Differentiating with respect to p:

$$\frac{\partial J(p, \eta)}{\partial p}(x) = -\log p(x) - 1 + \eta_0 + \sum_{j=1}^{m} \eta_j T_j(x) \tag{2.4}$$

By setting the derivative equal to 0 and solving for $p(x)$:

$$p^*(x) = e^{\eta_0^* - 1 + \eta^{*\top} T(x)} \tag{2.5}$$

Finally, the second derivative is negative, so p^* is indeed a maximum:

$$\frac{\partial^2 J(p^*, \eta)}{\partial p^2}(x) = -\frac{1}{p^*(x)} < 0 \tag{2.6}$$

Despite this simple form, these distributions are only in rare cases tractable from the standpoint of calculating η^*, calculating the normalizing constant of p^*, and sampling from the resulting distribution. There is extensive literature on finding η^* numerically [Darroch and Ratcliff, 1972, Salakhutdinov et al., 2002, Della Pietra et al., 1997, Dudik et al., 2004, Malouf, 2002, Collins et al., 2002], but doing so requires

computing normalizing constants, which poses a challenge even for problems with modest dimensions. To see why this is the case, consider the exponential family:

$$p_\eta(x) = C^{-1}(\eta) e^{\eta^\top T(x)} \mathbb{1}(x \in \mathscr{X}) \tag{2.7}$$

where $C(\eta) = \int_{\mathscr{X}} e^{\eta^\top T(x)} dx$. By properties of exponential families:

$$-\frac{\partial \log C(\eta)}{\partial \eta} = E_{X \sim p_\eta}[T(X)] \tag{2.8}$$

Since $E_{X \sim p_{\eta^*}}[T(X)] = 0$, this means that finding p^* is equivalent to finding a critical point of $-\log C(\eta)$. Furthermore, by properties of exponential families:

$$-\frac{\partial^2 \log C(\eta)}{\partial \eta^2} = \mathrm{cov}_{X \sim p_\eta}\Big(T(X)\Big) \succeq 0 \tag{2.9}$$

Thus, to find η^* we can minimize $-\log C(\eta)$. In most practical problems, this is not tractable as we do not have access to the normalizing constant. Finally, even if η^* is somehow correctly found, it is still not trivial to sample from p^*. Problem-specific sampling methods, such as MCMC have to be designed [Zhu et al., 2000]. This requires problem-specific considerations and faces the usual MCMC issues: having to wait a burn-in period, the chains might mix very slowly so the target distribution might not be sampled from, and having to throw away samples because they are correlated instead of iid.

2.2 The Augmented Lagrangian Method

In chapter 3, we will use the augmented Lagrangian method [Bertsekas, 2014], which we introduce in this section, to solve the maximum entropy problem. The method allows to solve constrained optimization problems of the form:

$$\begin{aligned} \theta^* = \underset{\theta}{\text{minimize}} \quad & -H(\theta) \\ \text{subject to } & R(\theta) = 0 \end{aligned} \tag{2.10}$$

Algorithm 1 Augmented Lagrangian Method

1: Set $\gamma \in (0, 1)$ and $\beta > 1$.

2: Initialize θ_0, c_0 and λ_0.

3: **for** $k = 1, ..., k_{\max}$ **do**

4: Find θ_{k+1}, the minimizer of $L(\cdot; \lambda_k, c_k)$ using θ_k as initial point.

5: Update $\lambda_{k+1} = \lambda_k + c_k R(\theta_k)$

6: Update $c_{k+1} = \begin{cases} \beta c_k, & \text{if } ||R(\theta_{k+1})|| > \gamma ||R(\theta_k)|| \\ c_k, & \text{otherwise} \end{cases}$

7: **end for**

where both $H : \mathbb{R}^q \to \mathbb{R}$ and $R : \mathbb{R}^q \to \mathbb{R}^m$ are smooth functions. The augmented Lagrangian is defined as:

$$L(\theta; \lambda, c) = -H(\theta) + \lambda^\top R(\theta) + \frac{c}{2}||R(\theta)||^2 \tag{2.11}$$

where $c > 0$ and $\lambda \in \mathbb{R}^m$. The augmented Lagrangian is minimized for a non-decreasing sequence of c and well-chosen λ (see details in algorithm 1). It might at first appear that the middle term is unnecessary, as optimizing $L(\theta; 0, c_k)$ should recover the solution as $c_k \to \infty$. However, doing this is numerically unstable, as c_k has to go to infinity. The idea behind the augmented Lagrangian method is that if λ is close to the Lagrange multiplier associated with equation 2.10, then c does not have to grow to infinity.

A couple of things should be noted about algorithm 1: First, the λ updates are simply updates on the direction of the augmented Lagrangian's gradient with respect to λ, with step-size c_k. Second, the intuition behind the c updates is that c should only be increased if the constraints did not sufficiently decrease. Under some regularity conditions it can be proved that c_k is the optimal step-size for the λ updates, that c_k will not diverge to infinity (thus avoiding the ill-conditioning problem) and that the augmented Lagrangian method will solve the constrained minimization problem of equation 2.10, see Bertsekas [2014].

2.3 Texture Networks

Constructing generative models to generate random images with certain texture is an important task in computer vision which can be naturally formulated as a maximum entropy problem. In this section, we will explain the approach followed by Ulyanov et al. [2016]. While they do not formulate the problem as a maximum entropy problem, we will show in chapter 3 that doing so outperforms their method. Having a texture image $\widetilde{x}$, the goal is to train a neural network f_θ such that after applying the network to a random noise input X_0, an image $f_\theta(X_0)$ of the same texture as $\widetilde{x}$ is generated. To achieve this goal, a complicated loss T introduced by Gatys et al. [2015] is used. The idea is that $T(\theta)$ is small only when the the images produced by the network and the original image correspond to images with the same texture. The network is trained to minimize:

$$\theta^* = \underset{\theta}{\text{minimize}} \;\; R(\theta) \tag{2.12}$$

where $R(\theta) = E[T(f_\theta(X_0))]$. It should be noted that while this produces high-quality synthetic texture images, this network only focuses on generating feature-matching images. Doing so can be deeply problematic: this could result in obtaining a point mass at the original image. In what follows, we give a brief description of the loss used. A convolutional neural network is pre-trained for image recognition [Simonyan and Zisserman, 2014, Chatfield et al., 2014] and the i-th feature of the l-th convolutional layer applied to image x is denoted $F_i^l(x)$. Then, the Gram matrix $G^l(x)$ is defined as:

$$G_{ij}^l(x) = F_i^l(x)^\top F_j^l(x) \tag{2.13}$$

Finally, the loss is defined as:

$$T(x) = \sum ||G^l(x) - G^l(\widetilde{x})||^2 \tag{2.14}$$

where the sum is over a selected set of indices l of convolutional layers.

2.4 Normalizing Flows

Following Rezende and Mohamed [2015], we define a *normalizing flow* as the transformation of a probability density through a sequence of invertible mappings. Normalizing flows provide an elegant way of generating a complicated distribution while maintaining tractable density evaluation, which we will use in chapter 3 to solve the maximum entropy problem. Starting with a simple distribution $X_0 \in \mathbb{R}^D \sim p_0$ (usually taken to be a standard multivariate Gaussian), and by applying k invertible and smooth functions $f_i : \mathbb{R}^D \to \mathbb{R}^D (i = 1, ..., k)$, the resulting variable $X_k = f_k \circ f_{k-1} \circ \cdots \circ f_1(X_0)$ has density:

$$p_k(x_k) = p_0(f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_k^{-1}(x_k)) \prod_{i=1}^{k} \left| \det \left(\frac{\partial f_i^{-1}}{\partial x_i}(x_i) \right) \right| \tag{2.15}$$

$$= p_0(x_0) \prod_{i=1}^{k} \left| \det \left(\frac{\partial f_i}{\partial x_{i-1}}(x_{i-1}) \right) \right|^{-1} \tag{2.16}$$

Equations 2.15 and 2.16 allow to compute the density of the transformed variable under two (sometimes overlapping) scenarios: when we have access to f_i^{-1} and when we have access to f_i, respectively. In applications such as maximum likelihood estimation where we have observations of X_k, we need to compute f_i^{-1}, and thus equation 2.15 is used. The f_i functions have then to be constructed in such a way that the determinant of the Jacobian of their inverses, $|\partial f_i^{-1}/\partial x_i|$, can be computed efficiently. In applications where we have access to the non-transformed variable, such as variational inference, equation 2.16 is used instead and the normalizing flows have to be constructed in such a way that $|\partial f_i/\partial x_{i-1}|$ can be computed efficiently. Finally, note that efficient sampling of observations of X_k requires efficiently computing each of the f_i functions. In the rest of this section we go over popular normalizing flows for both of these scenarios.

2.4.1 Fast Backward Computations

Papamakarios et al. [2017] propose the Masked Autoregressive Flow, which is a type of normalizing flow designed for maximum likelihood estimation, that is, it allows for fast computation of f_i^{-1} for efficient evaluation of the log likelihood, although it suffers from slow f_i evaluations and thus slow sampling. For notational simplicity, we drop the i index and write x_{i-1} as x and x_i as f as each f_i is built in a similar way. In order to obtain f from x, the following recursion is performed:

$$f_d = x_d \exp\left(g_\theta^{(d)}(f_{:d-1})\right) + h_\theta^{(d)}(f_{:d-1}) \text{ for } d = 1, \ldots, D \tag{2.17}$$

where f_d and x_d are the d-th coordinates of f and x, respectively, $f_{:d-1}$ are the first $d-1$ coordinates of f and each $g_\theta^{(d)}$ and $h_\theta^{(d)}$ is a neural network parameterized by θ that depends only on the first $d-1$ coordinates of f. Note that this transformation results in a triangular Jacobian, whose determinant can be computed efficiently. Computing f from x is slow, as each coordinate requires computing all the previous ones beforehand, that is, f_d depends on $f_{:d-1}$. However, computing the inverse, i.e. x from f does not require sequential computations:

$$x_d = \left(f_d - h_\theta^{(d)}(f_{:d-1})\right) \exp\left(-g_\theta^{(d)}(f_{:d-1})\right) \tag{2.18}$$

Equation 2.18 can be computed at the same time for $d = 1, \ldots, D$, thus resulting in fast evaluation of the inverse normalizing flow. Further speedups are achieved by using binary masks on f in order to only have one neural network g_θ and one h_θ instead of D of each.

2.4.2 Fast Forward Computations

Rezende and Mohamed [2015] proposed two specific families of transformations for variational inference, namely planar flows and radial flows, respectively:

$$f = x + uh(w^T x + b) \qquad \text{and} \qquad f = x + \beta h(\alpha, r)(x - x'), \tag{2.19}$$

where $b \in \mathbb{R}$, $u, w \in \mathbb{R}^D$ and h is an activation function in the planar case, and where $\beta \in \mathbb{R}$, $\alpha > 0$, $x' \in \mathbb{R}^D$, $h(\alpha, r) = 1/(\alpha + r)$ and $r = ||x - x'||$ in the radial. For a certain domain of the parameters, the transformations are invertible and the determinant of the Jacobian can be easily computed with the matrix determinant lemma. However, x cannot be analytically recovered from f.

Kingma et al. [2016] propose the Inverse Autoregressive Flow, which is very similar to the inverse of the Masked Autoregressive Flow (section 2.4.1). Obtaining x from f is slow, so that while this normalizing flow is well suited for variational inference, it is not so for maximum likelihood estimation. Computing f from x is done as follows:

$$f_d = \frac{x_d - h_\phi^{(d)}(x_{:d-1})}{g_\phi^{(d)}(x_{:d-1})} \tag{2.20}$$

where again, instead of implementing each of the $2D$ neural networks $h_\phi^{(d)}$ and $g_\phi^{(d)}$, they are all implemented at the same time by using binary masks for their inputs. Note that we parameterize these networks with ϕ and not θ to make notation consistent with section 2.5, as these normalizing flows are usually used for variational inference and not for the generative part of the model which we parameterize with θ.

2.4.3 Fast Backward and Forward Computations

There are also normalizing flows which allow, at the price of some flexibility as compared to their autoregressive counterparts, for both efficient backward anf forward computations. Dinh et al. [2016] proposed one such normalizing flow, called real NVP, which uses a convolutional, multiscale structure that is suitable for image modeling. The real NVP is constructed by stacking multiple coupling layers on top of each other. The coupling layer which transforms x into f is defined as follows:

$$\begin{cases} f_A = x_A \\ f_B = x_B \odot \exp\Big(g_\theta(x_A)\Big) + h_\theta(x_A) \end{cases} \tag{2.21}$$

where A and B are a partition of the indices $\{1, \dots, D\}$, h_θ and g_θ are neural networks parameterized by θ mapping from $\mathbb{R}^{|A|}$ to $\mathbb{R}^{|B|}$ and $\odot$ denotes element-wise product.

This is clearly invertible as:

$$\begin{cases} x_A = f_A \\ x_B = \Big(f_B - h_\theta(f_A)\Big) \odot \exp\Big(-g_\theta(f_A)\Big) \end{cases} \tag{2.22}$$

A key feature of coupling layers is that their Jacobian is triangular and can thus be computed efficiently. Furthermore, computing the Jacobian requires only evaluating g_θ. Invertibility of h_θ and g_θ is thus not required, and these functions can indeed be arbitrary. In practice, they are taken as convolutional neural networks. When stacking coupling layers on top of each other, the index partitions are changed to ensure that all the coordinates are transformed. There is further work in making normalizing flows even more flexible, for example Kingma and Dhariwal [2018] use invertible 1×1 convolutions in addition to coupling layers, and some works change the affine relation between f_B and x_B to more general invertible one, such as monotonic cubic splines [Durkan et al., 2019a] or monotonic rational-quadratic transforms [Durkan et al., 2019b].

2.5 Variational Autoencoders

Autoencoding variational Bayes [Kingma and Welling, 2013] is a technique to perform inference in the model:

$$\begin{cases} Z_r \sim \pi(z) \\ X_r | Z_r \sim p_\theta(x|z_r) \ , \ \text{for } r = 1, \dots, R, \end{cases} \tag{2.23}$$

where each $Z_r \in \mathbb{R}^M$ is a local hidden variable, and θ are parameters for the likelihood of observables X_r. We refer to these models as *Variational Autoencoders* (VAE), and we will use them in chapters 4 and 5. We denote the observed data $(x_1, \dots, x_R)$ by $\mathbf{x}$ and the corresponding latent variables $(z_1, \dots, z_R)$ by $\mathbf{z}$. The prior $\pi(z)$ is conventionally a Gaussian $\mathcal{N}(0, I_M)$. When the data is binary, i.e. $x_r \in \{0,1\}^D$, the conditional likelihood $p_\theta(x_r|z_r)$ is chosen to be $\mathscr{B}(\lambda_\theta(z_r))$, where $\lambda_\theta : \mathbb{R}^M \to \mathbb{R}^D$ is a neural

network with parameters θ. $\mathscr{B}(\lambda)$ denotes the product of D independent Bernoulli distributions, with parameters $\lambda \in [0,1]^D$ (in the standard way we overload $\mathscr{B}(\cdot)$ to be the univariate Bernoulli or the product of independent Bernoullis). When the data is real-valued, a Gaussian distribution is commonly used instead of a Bernoulli. Since direct maximum likelihood estimation of θ is intractable, variational autoencoders use VBEM Jordan et al. [1999], first positing a now-standard variational posterior family:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{r=1}^{R} q_\phi(z_r|x_r), \text{with } q_\phi(z_r|x_r) = \mathscr{N}\Big(m_\phi(x_r), \text{diag}\left(s_\phi^2(x_r)\right)\Big) \tag{2.24}$$

where $m_\phi : \mathbb{R}^D \to \mathbb{R}^M$, $s_\phi : \mathbb{R}^D \to \mathbb{R}_+^M$ are neural networks parameterized by ϕ. Note that amortized inference [Gershman and Goodman, 2014, Rezende et al., 2014] is used, that is, instead of having separate variational parameters for each observation, a neural network is used to map from observables to variational parameters. This makes training faster and reduces computations at test time at the price of inference suboptimality [Cremer et al., 2018]. To train the model, the evidence lower bound (ELBO) $\mathscr{E}(\theta, \phi)$ is maximized over both generative and posterior (decoder and encoder) parameters (θ, ϕ):

$$\mathscr{E}(\theta,\phi) = \sum_{r=1}^{R} E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)] - KL(q_\phi(z_r|x_r)||\pi(z_r)) \leq \log p_\theta(\mathbf{x}) \tag{2.25}$$

The log likelihood is lower bounded by the ELBO, and the bound becomes tight when the approximate posterior matches the true posterior. Optimizing the ELBO is usually done with stochastic gradient methods [Robbins and Monro, 1951], which allow to use data mini-batches and use sample based estimates of the involved expectations. In order to do this, the gradient of the ELBO with respect to both θ and ϕ has to be estimated. Doing this for the KL term is not difficult as it can be written down analytically:

$$KL(q_\phi(z_r|x_r)||\pi(z_r)) = \frac{1}{2}\sum_{j=1}^{M}\Big(1 + 2\log s_{\phi,j}(x_r) - m_{\phi,j}^2(x_r) - s_{\phi,j}^2(x_r)\Big) \tag{2.26}$$

where $m_{\phi,j}$ and $s_{\phi,j}$ are the j-th coordinates of m_ϕ and s_ϕ, respectively. The first term (called reconstruction term) is slightly trickier: its gradient with respect to θ can

be easily estimated since the gradient can be brought down inside the expectation, allowing to use a sample based estimator. However, this cannot be done with ϕ, as the expectation is with respect to a distribution that depends on ϕ. One possible solution is to use the log derivative trick, which results in the score estimator [Kleijnen and Rubinstein, 1996]:

$$\nabla_\phi E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)] = E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)\nabla_\phi q_\phi(z_r|x_r)] \tag{2.27}$$

While equation 2.27 allows to use sample based estimators of the gradient of the ELBO with respect to ϕ, empirically these estimators suffer from high variance. To avoid this high variance issue, the reconstruction term is usually written down using the reparameterization trick:

$$E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)] = E_{\epsilon\sim\mathcal{N}(0,I_M)}\Big[\log p_\theta\Big(x_r|m_\phi(x_r) + s_\phi(x_r)\odot\epsilon\Big)\Big] \tag{2.28}$$

where $\odot$ denotes elementwise product. The reparameterization trick enables writting the reconstruction term as an expectation with respect to a distribution that does not depend on ϕ. When computing the gradient with respect to ϕ, this allows to simply bring the gradient inside the expectation.

Note that making the prior depend on learnable generative parameters θ can be done with the same procedure of maximizing the ELBO. Normalizing flows with tractable forward computations (section 2.4.2) can be used to obtain more flexible posterior approximations than the Gaussian one presented here, as the they still allow the use of the reparameterization trick (tractably, thanks to the fast forward computations) and tractable density evaluation means that the KL term can still be computed.

2.5.1 Importance Weighting

Burda et al. [2015] modify the ELBO to obtain a tighter bound to the log likelihood by using importance weights:

$$\mathscr{L}_k(\theta,\phi) = E_{\mathbf{z}^{(1)},\dots,\mathbf{z}^{(k)}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\frac{1}{k}\sum_{j=1}^{k}\frac{p_\theta(\mathbf{x}|\mathbf{z}^{(j)})\pi(\mathbf{z}^{(j)})}{q_\phi(\mathbf{z}^{(j)}|\mathbf{x})}\right] \leq \log p_\theta(\mathbf{x}) \tag{2.29}$$

Note that when $k = 1$, this modified objective recovers the ELBO. This objective has the advantage that, as k goes to infinity, $\mathscr{L}_k$ converges to the log likelihood under reasonable assumptions. While this objective does provide a better approximation to the log likelihood than the regular ELBO, maximizing $\mathscr{L}_k$ results in worse posterior approximations and is slower. Thus, $\mathscr{L}_k$ can also be used not as an objective, but simply to better approximate the log likelihood after having trained using the ELBO.

2.5.2 β-VAE

With the goal of obtaining disentangled representations, Higgins et al. [2017] modify the ELBO by weighting the KL term differently:

$$\mathscr{E}_\beta(\theta, \phi) = \sum_{r=1}^{R} E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)] - \beta KL(q_\phi(z_r|x_r)||\pi(z_r)) \tag{2.30}$$

when $\beta > 1$, the approximate posterior is more strongly pushed towards matching the standard Gaussian prior, resulting in more independent coordinates of the latent representations. The β-VAE objective can be interpreted as maximizing the reconstruction term of the ELBO subject to the constraint that the KL between the approximate posterior and the prior is less than a certain threshold. This allows to interpret β as the Lagrange multiplier obtained by the KKT conditions of the corresponding constrained optimization problem.

2.5.3 PfLDS

Gao et al. [2016] introduce a VAE-type model for neural population data, called PfLDS, against which we compare our method in chapter 5. Suppose the spiking times of N neurons are simultaneously recorded for R trials over a time interval $[T_1, T_2)$, which is partitioned into B disjoint intervals, which we call time bins. Let $x_{r,n,b}$ be the observed number of spikes on time bin b for neuron n on trial r. To incorporate the temporal structure of the data into the model, there is not just a latent variable Z_r per trial, but a latent variable per trial per time bin, $Z_{r,b}$. The evolution of this latent

trajectory is modeled with a Gaussian linear dynamical system, which is transformed through a neural network to obtain the parameters of the Poisson distribution which is used to model the observed spike counts:

$$\begin{cases} Z_{r,1} \sim \mathcal{N}(\mu_1, \Sigma_1) \text{ for } r = 1, \dots, R \\ Z_{r,b+1} | Z_{r,b} \sim \mathcal{N}(AZ_{r,b}, \Sigma) \text{ for } b = 1, \dots, B-1 \\ \lambda_{r,n,b} = f_{\theta_-}^{(n,b)}(Z_{r,b}) \text{ for } n = 1, \dots, N \\ X_{r,n,b} | \lambda_{r,n,b} \sim \mathcal{P}(\lambda_{r,n,b}) \end{cases} \tag{2.31}$$

where each $f_{\theta_-}^{(n,b)}$ is a neural network (we denote the parameters of all these networks with θ_-), $\mathcal{P}$ denotes the Poisson distribution and the generative parameters of the model are $\theta = (\mu_1, \Sigma_1, A, \Sigma, \theta_-)$. Once again in order to account for the temporal structure of the data, the approximate posterior over $\mathbf{Z}_r = (Z_{r,1}, \dots, Z_{r,B})$ is modified: instead of the usual diagonal covariance structure, a block tridiagonal structure is used. The model is then trained by maximizing the ELBO.

2.5.4 Discrete Latent Variables

While many standard VAE choices, such a Gaussian priors, can be changed while retaining the ability to optimize the ELBO, using discrete latent variables is not so straightforward. This is because the reparameterization trick cannot be used anymore as writing the latents as a transformation of parameter-free noise can only be done through a transformation that takes finitely many values, thus not allowing the use of gradient methods. Maddison et al. [2016] and Jang et al. [2016] simultaneously proposed a method to circumvent this issue. Their method allows to maximize expressions of the form:

$$q^* = \underset{q}{\text{maximize}} \;\; E_{Z \sim q}[f(Z)] \tag{2.32}$$

where q is a discrete distribution over the finite set $\{1, \dots, C\}$ and $f : \{1, \dots, C\} \to \mathbb{R}$. Note that a VAE with discrete latent variables falls exactly into this category due to

the reconstruction term. While score estimators (equation 2.27) can still be used in this setting, they still suffer from the same high variance issue as in the continuous case. The solution is to use a continuous relaxation $\widetilde{q}_a$ of q. By using a one-hot representation of the elements of $\{1, \dots, C\}$, the support of $\widetilde{q}_a$ is chosen to be the $(C-1)$-dimensional simplex $\mathscr{S}^{C-1} = \{(x_1, \dots, x_C) : x_c \geq 0 \text{ and } \sum_{c=1}^{C} x_c = 1\}$. A sample $\widetilde{Z}$ from this relaxation can be obtained by:

$$\widetilde{Z} = \text{softmax}\left(\frac{\log a + \epsilon}{\tau}\right) \tag{2.33}$$

where $a \in (0, \infty)^C$ and $\tau \in (0, \infty)$ are the parameters of the distribution and $\epsilon \in \mathbb{R}^C$ is a random vector with independent Gumbel(0,1) distributed entries. This distribution, called Gumbel-softmax or concrete distribution, has the property that as the temperature τ (which is treated as a hyperparameter) goes to 0, $\widetilde{q}_a$ converges to the distribution proportional to a on the vertices of $\mathscr{S}^{C-1}$ (i.e. one-hot vectors). The concrete distribution also admits a closed form density and allows to use the reparameterization trick by relaxing the objective of equation 2.32 to:

$$a^* = \underset{a}{\text{maximize}} \; E_{Z \sim \widetilde{q}_a}[\widetilde{f}(Z)] \tag{2.34}$$

where $\widetilde{f} : \mathscr{S}^{C-1} \to \mathbb{R}$ is a relaxation of f. The fact that not only q but also f needs to be relaxed is often glossed over or ignored in the literature. Further studying this is an interesting project for future research, as this seemingly unimportant technicality is very similar to the seemingly unimportant technicality that is the basis for chapter 4, namely using a Bernoulli likelihood to model $[0, 1]$-valued data.

2.6 Evaluation Metrics

Constructing metrics to evaluate performance of implicit generative models (i.e. when there is no access to the likelihood or a bound on it) is an active area of research. The goal is having some measure of how good a trained model is when having access to nothing but samples from the model and real samples, which is typical in models

such as GAN [Goodfellow et al., 2014]. Some of these metrics are specifically designed to measure mostly GAN-specific problems such as mode collapse [Arora and Zhang, 2017], but some other metrics attempt to measure sample quality. In this section we present two methods to do so. Although we do not use GAN in chapter 4, we use one of these metrics to show the empirical advantage produced by our model.

2.6.1 Inception Score

The inception score [Salimans et al., 2016] is a popular technique used to evaluate performance of deep generative models. In order to compute the inception score, a labeled dataset $(x_1, y_1), \ldots, (x_R, y_R)$ is needed. As a first step, a classifier is trained: given an input x, the classifier returns a probability distribution $p(y|x)$ over the labels. The inception score for a generative model $p_\theta(x)$ is given by:

$$\exp\left(E_{X\sim p_\theta}[KL(p(y|X)||p(y))]\right) \tag{2.35}$$

where $p(y) = \int p(y|x)p_\theta(x)dx$. Note that the inception score can be easily approximated by sampling from $p_\theta(x)$. To understand why the inception score is a reasonable metric, note that:

$$E_{X\sim p_\theta}[KL(p(y|X)||p(y))] = H(Y) - H(Y|X) \tag{2.36}$$

where $H(Y)$ and $H(Y|X)$ denote the entropy of Y and the conditional entropy of Y given X, respectively. A high inception score thus embodies two desirable properties of a generative model: the entropy of the labels, $H(Y)$ is large, meaning that the model generates samples from all possible labels; and $H(Y|X)$ is small, meaning that conditioning on a sample reduces uncertainty about its label.

2.6.2 Fréchet Inception Distance

The Fréchet inception distance [Heusel et al., 2017] is another metric to evaluate performance of deep generative models. Once again, a classifier is trained to predict

the labels and a particular layer of this classifier is chosen. Denoting $l(X)$ as the chosen layer evaluated at input X, the Fréchet inception distance is given by:

$$||\mu_g - \mu_{true}||_2^2 - \text{tr}(\Sigma_{true} + \Sigma_g - 2(\Sigma_{true}\Sigma_g)^{1/2}) \tag{2.37}$$

where $\mu_g = E[l(X)]$ and $\Sigma_g = cov(l(X))$ under p_θ and μ_{true} and Σ_{true} are defined analogously but with respect to p_{true}, the true data generating distribution. Note that sample based approximations can be used for all these quantities. The Fréchet inception distance corresponds to placing Gaussian observation models on layer l for generated and real data and then computing the Fréchet distance between these two Gaussians, so that lower values of the Fréchet inception distance are better. The Fréchet inception distance has the advantage over the inception score that it compares the generated samples against real ones instead of evaluating them in a vacuum (although the inception score also uses real samples when training the classifier). However, the choice of the layer l can have a big impact on this metric.

2.7 Poisson Processes

A Poisson process in a measurable space $\mathscr{S}$ is a distribution over subsets of $\mathscr{S}$. We will use Poisson processes in chapter 5 to model spiking times of neural populations. A Poisson process is parameterized by a measure G in $\mathscr{S}$, and we say that a random set S follows a Poisson process on $\mathscr{S}$, which we denote by $S \sim \mathscr{PP}_{\mathscr{S}}(G)$, if the following conditions hold:

1. For any disjoint measurable subsets $A_1, \ldots, A_n$ of $\mathscr{S}$, $|A_1 \cap S|, \ldots, |A_n \cap S|$ are independent random variables, where $|\cdot|$ denotes the number of elements in a set.

2. For any measurable subset A of $\mathscr{S}$, $|A \cap S| \sim \mathscr{P}(G(A))$, where $\mathscr{P}$ denotes the Poisson distribution.

Under mild conditions on $\mathscr{S}$ and G, the Poisson process exists [Kingman, 1992] and, if G is finite, can be sampled as follows:

1. Sample $K \sim \mathscr{P}(G(\mathscr{S}))$.
2. Sample $x_1, \ldots, x_K$ iid from a distribution proportional to G, i.e., $G(\cdot)/G(\mathscr{S})$, and take $S = \{x_1, \ldots, x_K\}$.

If G admits a density with respect to some base measure, the density g (called intensity) is commonly used to parameterize the process instead of G. For our purposes, we will consider the case where $\mathscr{S} = [T_1, T_2)$ and the base measure will be the Lebesgue measure, as we will be interested in modeling events (elements of S) in time. If $S = \{x_k\}_{k=1}^K \sim \mathscr{P}\mathscr{P}_{\mathscr{S}}(g)$, then the log likelihood of S is given by:

$$\log p(\{x_k\}_{k=1}^K | g) = \sum_{k=1}^K \log g(x_k) - \int_{\mathscr{S}} g(t) dt \tag{2.38}$$

A useful extension of Poisson processes are the so called marked Poisson processes. These are simply Poisson processes over $\mathscr{S} \times \{1, \ldots, N\}$. This allows to model N different types of events that can happen in the same time interval. Bayesian estimation of the intensity of Poisson processes can be performed by placing a Gaussian process [Rasmussen, 2004] prior on the log intensity function, resulting in log Gaussian Cox processes [Møller et al., 1998]. However, due to the integral in equation 2.38, this procedure results in a difficult, doubly intractable inference procedure, despite some efforts by Cunningham et al. [2008], Adams et al. [2009], Lloyd et al. [2015].

2.8 Splines and Nonnegative Polynomials

Splines are a flexible class of functions which we will use in chapter 5 to model intensity functions of Poisson processes. Consider the interval $[T_1, T_2)$ and $I + 1$ fixed knots $T_1 = t_0 < \cdots < t_I = T_2$. A spline of degree d and smoothness $s < d$ is a continuous, s times differentiable function on $[T_1, T_2)$ which is a polynomial in each interval $[t_{i-1}, t_i)$

for $i = 1, \ldots, I$. Splines have the nice property that they form a vector space whose basis functions can be written down explicitly [Wahba, 1990]. This allows to easily fit splines in regression settings by using linear regression with the basis functions as features. Splines are popular for regression as they are more flexible than linear models, while not being overly complex.

Since we will later use spline to model intensity functions of Poisson processes, we are interested in nonnegative splines. Unfortunately, the space of nonnegative splines does not form a vector space anymore, so the same "trick" that is used in the regression setting cannot be applied anymore. The other "natural" way of parameterizing splines, namely with the $d+1$ polynomial coefficients for each of the I intervals, also does not lend itself to easily characterize nonnegativity [Schmidt and Hess, 1988]. A beautiful but perhaps lesser known spline result (see Lasserre [2010]) gives that a polynomial $p(t)$ of degree d, where $d = 2k+1$ for some $k \in \mathbb{N}$, is nonnegative in the interval $[l, u)$ if and only if it can be written down as follows:

$$p(t) = (u-t)[t]^\top Q_1 [t] + (t-l)[t]^\top Q_2 [t] \tag{2.39}$$

where $[t] = (1, t, t^2, \ldots, t^k)^\top$ and Q_1 and Q_2 are $(k+1) \times (k+1)$ symmetric positive semidefinite matrices. An analogous result holds for polynomials of even degree $d = 2k$, which are nonnegative in $[l, u)$ if and only if:

$$p(t) = [t]^\top Q_1 [t] + (u-t)(t-l)\widetilde{[t]}^\top Q_2 \widetilde{[t]} \tag{2.40}$$

where in this case Q_2 is now a $k \times k$ symmetric positive semidefinite matrix and $\widetilde{[t]} = (1, t, t^2, \ldots, t^{k-1})^\top$. These results will later allow us to parameterize nonnegative splines.

2.9 Method of Alternating Projections

Consider closed, convex sets $\mathscr{C}_0, \ldots, \mathscr{C}_{s+1}$ in a Hilbert space $\mathbb{H}$ with nonempty intersection. The method of alternating projections [von Neumann, 1950, Bauschke

and Borwein, 1996] allows to recover a point in $\cap_{j=0}^{s+1}\mathscr{C}_j$ when projecting onto this intersection is hard but projecting onto each of the sets $\mathscr{C}_j$ is easy. We will use this in chapter 5 to obtain nonnegative splines. As the name suggests, the method proceeds by iteratively projecting onto each of the sets in a cycling fashion. Formally, given a starting point $\psi^{(0)} \in \mathbb{H}$, the k-th step of the method of alternating projections is obtained by:

$$\psi^{(k)} = P_{k \bmod (s+2)}(\psi^{(k-1)}) \tag{2.41}$$

where P_j is the projection onto $\mathscr{C}_j$ for $j = 0, \ldots, s+1$. As $k \to \infty$, $\psi^{(k)}$ converges to a point in $\cap_{j=0}^{s+1}\mathscr{C}_j$.

Chapter 3

Maximum Entropy Flow Networks

3.1 Introduction

Maximum entropy modeling is a flexible and popular framework for formulating statistical models given partial knowledge. In this chapter, rather than the traditional method of optimizing over the continuous density directly, we learn a smooth and invertible transformation that maps a simple distribution to the desired maximum entropy distribution. Doing so is nontrivial in that the objective being maximized (entropy) is a function of the density itself. By exploiting recent developments in normalizing flow networks, we cast the maximum entropy problem into a finite-dimensional constrained optimization, and solve the problem by combining stochastic optimization with the augmented Lagrangian method. Simulation results demonstrate the effectiveness of our method, and applications to finance and computer vision show the flexibility and accuracy of using maximum entropy flow networks.

The maximum entropy (ME) principle [Jaynes, 1957] states that subject to some given prior knowledge, typically some given list of moment constraints, the distribution that makes minimal additional assumptions – and is therefore appropriate for a range of applications from hypothesis testing to price forecasting to texture synthesis – is that which has the largest entropy of any distribution obeying those constraints. First

introduced in statistical mechanics by Jaynes [1957], and considered both celebrated and controversial, ME has been extensively applied in areas including natural language processing [Berger et al., 1996], ecology [Phillips et al., 2006], finance [Buchen and Kelly, 1996], computer vision [Zhu et al., 1998], and many more.

Continuous ME modeling problems typically include certain expectation constraints, and are usually solved by introducing Lagrange multipliers, which under typical assumptions yields an exponential family distribution with natural parameters such that the expectation constraints are obeyed (see section 2.1). Unfortunately, fitting ME distributions in even modest dimensions poses significant challenges. First, optimizing the Lagrangian for a Gibbs distribution (the solution to the ME problem) requires evaluating the normalizing constant, which is in general computationally very costly and error prone. Secondly, in all but the rarest cases, there is no way to draw samples independently and identically from this Gibbs distribution, even if one could derive it. Third, unlike in the discrete case where a number of recent and exciting works have addressed the problem of estimating entropy from discrete-valued data [Jiao et al., 2015, Valiant and Valiant, 2013], estimating differential entropy from data samples remains inefficient and typically biased. These shortcomings are critical and costly, given the common use of ME distributions for generating reference data samples for a null distribution of a test statistic. There is thus ample need for a method that can both solve the ME problem and produce a solution that is easy and fast to sample.

In this chapter we develop Maximum Entropy Flow Networks (MEFN), a stochastic-optimization-based framework and algorithm for fitting continuous maximum entropy models. Two key steps are required. First, conceptually, we replace the idea of maximizing entropy over a density directly with maximizing, over the parameter space of an indexed function family, the entropy of the density induced by mapping a simple distribution (a Gaussian) through that optimized function. Modern neural networks, particularly in variational inference [Kingma and Welling, 2013, Rezende and Mohamed, 2015], have successfully employed this same idea to generate complex

distributions, and we look to similar technologies. Secondly, unlike most other objectives in this network literature, the entropy objective itself requires evaluation of the target density directly, which is unavailable in most traditional architectures. We overcome this potential issue by learning a smooth, invertible transformation that maps a simple distribution to an (approximate) ME distribution. Recent developments in normalizing flows [Rezende and Mohamed, 2015, Dinh et al., 2016] allow us to avoid biased and computationally inefficient estimators of differential entropy (such as the nearest-neighbor class of estimators like that of Kozachenko-Leonenko; see Berrett et al. [2016]). Our approach avoids calculation of normalizing constants by learning a map with an easy-to-compute Jacobian, yielding tractable probability density computation. The resulting transformation also allows us to reliably generate iid samples from the learned ME distribution. We demonstrate MEFN in detail in examples where we can access ground truth, and then we demonstrate further the ability of MEFN networks in equity option prices fitting and texture synthesis.

Primary contributions of this chapter, which was published as Loaiza-Ganem et al. [2017], include: *(i)* addressing the substantial need for methods to sample ME distributions; *(ii)* introducing ME problems, and the value of including entropy in a range of generative modeling problems, to the deep learning community; *(iii)* the novel use of *constrained* optimization for a deep learning application; and *(iv)* the application of MEFN to option pricing and texture synthesis, where in the latter we show significant increase in the diversity of synthesized textures (over current state of the art) by using MEFN.

3.2 Maximum Entropy Flow Network Algorithm

3.2.1 Formulation

Instead of directly finding the Gibbs distribution (solving equation 2.2), we propose solving the ME problem (equation 2.1) directly by optimizing a transformation that

maps a random variable X_0, with simple distribution p_0, to the ME distribution. Given a parametric family of invertible transformations $\mathscr{F} = \{f_\theta, \theta \in \mathbb{R}^q\}$, we denote $p_\theta(x) = p_0(x_0)|\det(J_\theta(x_0))|^{-1}$ as the distribution of the variable $f_\theta(X_0)$, where J_θ is the Jacobian of f_θ. We then rewrite the ME problem as:

$$\begin{aligned} \theta^* = \underset{\theta}{\text{maximize}} \;\; & H(p_\theta) \\ \text{subject to } & R(\theta) := E_{X_0 \sim p_0}[T(f_\theta(X_0))] = 0 \\ & \text{supp}(p_\theta) = \mathscr{X}. \end{aligned} \tag{3.1}$$

When p_0 is continuous and $\mathscr{F}$ is suitably general, the program in equation 3.1 recovers the ME distribution p_θ exactly. With a flexible transformation family, the ME distribution can be well approximated. In practice, we take $\mathscr{F}$ to be a family of normalizing flows with fast forward computations (sections 2.4.2 and 2.4.3) followed by some invertible function g which maps to $\mathscr{X}$, the support of interest. In experiments we found that taking p_0 to be a standard multivariate normal distribution achieves good empirical performance. Taking p_0 to be a bounded distribution (e.g. uniform distribution) is problematic for learning transformations near the boundary, and heavy tailed distributions (e.g. Cauchy distribution) caused similar trouble due to large numbers of outliers.

3.2.2 Algorithm

We solved equation 3.1 using the augmented Lagrangian method (see section 2.2). As a technical note, the augmented Lagrangian method is guaranteed to converge under some regularity conditions [Bertsekas, 2014]. As is usual in neural networks, a proof of these conditions is challenging and not yet available, though intuitive arguments suggest that most of them should hold. Due to the non rigorous nature of these arguments, we rely on the empirical results of the algorithm to claim that it is indeed solving the optimization problem.

For a fixed (λ, c) pair, we optimize the augmented Lagrangian L (equation 2.11)

with stochastic gradient descent. Owing to our choice of network and the resulting ability to efficiently calculate the density $p_\theta(x^{(i)})$ for any sample point $x^{(i)}$ (which are easy-to-sample iid draws from the multivariate normal p_0), we compute the unbiased estimator of $H(p_\theta)$ with:

$$H(p_\theta) \approx -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(f_\theta(x^{(i)})) \tag{3.2}$$

$R(\theta)$ can also be estimated without bias by taking a sample average of $x^{(i)}$ draws:

$$R(\theta) \approx \frac{1}{n}\sum_{i=1}^{n} T(f_\theta(x^{(i)})) \tag{3.3}$$

Since $\nabla_\theta ||R(\theta)||^2 = 2(\nabla_\theta R(\theta))R(\theta)$, the gradient of the third term of the augmented Lagrangian can also be estimated without bias:

$$\nabla_\theta \frac{c}{2}||R(\theta)||^2 \approx c \cdot \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} \nabla_\theta T(f_\theta(x^{(i)})) \cdot \frac{2}{n}\sum_{i=\frac{n}{2}+1}^{n} T(f_\theta(x^{(i)})) \tag{3.4}$$

The resulting optimization procedure is detailed in Algorithm 2, of which step 9 requires some detail: denoting θ_k as the resulting θ after i_{max} SGD iterations at the augmented Lagrangian iteration k, the usual update rule for c [Bertsekas, 2014] is:

$$c_{k+1} = \begin{cases} \beta c_k, \text{ if } ||R(\theta_{k+1})|| > \gamma ||R(\theta_k)|| \\ c_k, \text{ otherwise} \end{cases} \tag{3.5}$$

where $\gamma \in (0, 1)$ and $\beta > 1$. Monte Carlo estimation of $R(\theta)$ sometimes caused c to be updated too fast, causing numerical issues. Accordingly, we changed the hard update rule for c to a probabilistic update rule: a hypothesis test is carried out with null hypothesis $H_0 : E[||R(\theta_{k+1})||] = E[\gamma||R(\theta_k)||]$ and alternative hypothesis $H_1 : E[||R(\theta_{k+1})||] > E[\gamma||R(\theta_k)||]$. The p-value p is computed, and c_{k+1} is updated to βc_k with probability $1 - p$. We used a two-sample t-test to calculate the p-value. What results is a robust and novel algorithm for estimating maximum entropy distributions, while preserving the critical properties of being both easy to calculate densities of particular points, and being trivially able to produce truly iid samples.

Algorithm 2 Training MEFN

initialize $\theta = \theta_0$, set $c_0 > 0$ and λ_0.

for Augmented Lagrangian iteration $k = 1, ..., k_{\max}$ **do**

  for SGD iteration $i = 1, ..., i_{\max}$ **do**

    Sample $x^{(1)}, ..., x^{(n)} \sim p_0$, get transformed variables $x_\theta^{(i)} = f_\theta(x^{(i)}), i = 1, ..., n$

    Update θ by descending its stochastic gradient (using e.g. ADADELTA [Zeiler, 2012]):

$$\nabla_\theta L(\theta; \lambda_k, c_k) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p_\theta(x_\theta^{(i)}) + \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta T(x_\theta^{(i)})\lambda_k + c_k \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} \nabla_\theta T(x_\theta^{(i)}) \cdot \frac{2}{n}\sum_{i=\frac{n}{2}+1}^{n} T(x_\theta^{(i)})$$

  end for

  Sample $x^{(1)}, ..., x^{(\widetilde{n})} \sim p_0$, get transformed variables $x_\theta^{(i)} = f_\theta(x^{(i)}), i = 1, ..., \widetilde{n}$

  Update $\lambda_{k+1} = \lambda_k + c_k \frac{1}{\widetilde{n}} \sum_{i=1}^{\widetilde{n}} T(x_\theta^{(i)})$

  Update $c_{k+1} \geq c_k$ (see text for detail)

end for

Below we give a more thorough discussion of the regularity conditions which ensure that the augmented Lagrangian method will work and why some should hold in our case. The goal is simply to give some intuition and not to attempt to prove that they indeed hold. The conditions [Bertsekas, 2014] are:

- There exists a strict local minimum θ^* of the optimization problem of equation 3.1:

  If the function class $\mathscr{F}$ is rich enough that it contains a true solver of the maximum entropy problem, then a global optimum exists. Although not rigorous, we would expect that even in the finite expressivity case that a global optimum remains, and indeed, recent theoretical work [Raghu et al., 2016, Poole et al., 2016] has gotten close to proving this.

- θ^* is a regular point of the optimization problem, that is, the rows of $\nabla_\theta R(\theta^*)$

are linearly independent:

Again, this is not formal, but we should not expect this to cause any issues. This clearly depends on the specific form of T, but the condition basically says that there should not be redundant constraints at the optimum, so if T is reasonable this should not happen.

- $H(p_\theta)$ and $R(\theta)$ are twice continuously differentiable on a neighborhood around θ^*:

  This holds by the smoothness of the normalizing flows.

- $y^\top \nabla^2_\theta L(\theta^*; \lambda^*, 0)y > 0$ for every $y \neq 0$ such that $\nabla_\theta R(\theta^*)y = 0$, where λ^* is the true Lagrange multiplier:

  This condition is harder to justify. It would appear it is just asking that the Lagrangian (not the augmented Lagrangian) be strictly convex in feasible directions, but it is actually stronger than this and some simple functions might not satisfy the property. For example, if the function we are optimizing was x^4 and we had no constraints, the Lagrangian's Hessian would be $12x^2$, which is 0 at $x^* = 0$ thus not satisfying the condition. Importantly, these conditions are sufficient but not necessary, so even if this doesn't hold the augmented Lagrangian method might work (it certainly would for x^4). Because of this and the non-rigorous justifications of the first two conditions, we relied instead on the empirical performance to justify that we are indeed recovering the maximum entropy distribution.

If all of these conditions hold, the augmented Lagrangian (for large enough c and λ close enough to λ^*) has a unique optimum in a neighborhood around θ^* that is close to θ^* (as $\lambda \to \lambda^*$ it converges to θ^*) and its Hessian at this optimum is positive-definite. Furthermore, $\lambda_k \to \lambda^*$. This implies that gradient descent (with the usual caveats of being started close enough to the solution and with the right steps) will correctly

recover θ^* using the augmented Lagrangian method. This of course just guarantees convergence to a local optimum, but if there are no additional assumptions such as convexity, it can be very hard to ensure that it is indeed a global optimum. Some recent research has attempted to explain why optimization algorithms perform so well for neural networks [Choromanska et al., 2015, Kawaguchi, 2016], but we leave such attempts for our case for future research.

3.3 Experiments

We first construct an ME problem with a known solution (section 3.3.1), and we analyze the MEFN algorithm with respect to the ground truth and to an approximate Gibbs solution. These examples test the validity of our algorithm and illustrate its performance. Sections 3.3.2 and 3.3.3 then apply MEFN to a financial data application (predicting equity option values) and texture synthesis, respectively, to illustrate the flexibility and practicality of our algorithm. Architectural and training choices are detailed in appendix A.

3.3.1 A Maximum Entropy Problem With Known Solution

Following the setup of the typical ME problem, suppose we are given a specified support $\mathscr{S} = \{x = (x_1, \ldots, x_{D-1}) : x_i \geq 0 \text{ and } \sum_{k=1}^{D-1} z_k \leq 1\}$ and a set of constraints $E[\log X_d] = \kappa_d (d = 1, ..., D)$, where $X_D = 1 - \sum_{d=1}^{D-1} X_d$. We then write the maximum entropy program:

$$\begin{aligned} p^* = \underset{p}{\text{maximize}} \;\; & H(p) \\ \text{subject to } & E_{X\sim p}[\log X_d - \kappa_d] = 0 \quad \forall d = 1, ..., D \\ & \text{supp}(p) = \mathscr{S} \end{aligned} \tag{3.6}$$

This is a general ME problem that can be solved via MEFN. Of course, we have particularly chosen this example because, though it may not obviously appear so,

the solution has a standard and tractable form, namely the Dirichlet. This choice allows us to consider a complicated optimization program that happens to have known global optimum, providing a solid test bed for the MEFN (and for the Gibbs approach against which we will compare). Specifically, given a parameter $\alpha \in \mathbb{R}^D$, the Dirichlet has density:

$$p(x_1, \ldots, x_{D-1}) = \frac{1}{B(\alpha)} \prod_{d=1}^{D} x_d^{\alpha_d - 1} \mathbb{1}\left((x_1, \ldots, x_{D-1}) \in \mathscr{S}\right) \tag{3.7}$$

where $B(\alpha)$ is the multivariate Beta function, and $x_D = 1 - \sum_{d=1}^{D-1} x_d$. Note that this Dirichlet is a distribution on $\mathscr{S}$ and not on the $(D-1)$-dimensional simplex $\mathscr{S}^{D-1} = \{(x_1, \ldots, x_D) : x_d \geq 0 \text{ and } \sum_{d=1}^{D} x_d = 1\}$ (an often ignored and seemingly unimportant technicality that needs to be correct here to ensure the proper transformation of measure). Connecting this familiar distribution to the ME problem above, we simply have to choose α such that $\kappa_d = \psi(\alpha_d) - \psi(\alpha_0)$ for $d = 1, ..., D$, where $\alpha_0 = \sum_{d=1}^{D} \alpha_d$ and ψ is the digamma function. We then can pose the above ME problem to the MEFN and compare performance against ground truth.

Before doing so, we must stipulate the transformation g that maps the Euclidean space of the multivariate normal p_0 to the desired support $\mathscr{S}$. Any sensible choice will work well (another point of flexibility for the MEFN); we use the standard transformation:

$$g(x_1, ..., x_{D-1}) = \left(\frac{e^{x_1}}{\sum_{d=1}^{D-1} e^{x_d} + 1}, ..., \frac{e^{x_{D-1}}}{\sum_{d=1}^{D-1} e^{x_d} + 1} \right)^\top \tag{3.8}$$

Note that the MEFN outputs vectors in $\mathbb{R}^{D-1}$, and not $\mathbb{R}^D$, because the Dirichlet is specified as a distribution on $\mathscr{S}$ (and not on the simplex $\mathscr{S}^{D-1}$). Accordingly, the Jacobian is a square matrix and its determinant can be computed efficiently using the matrix determinant lemma. Here, p_0 is set to the $(D-1)$-dimensional standard normal.

We proceed as follows: We choose α and compute the constraints $\kappa_1, ..., \kappa_D$. We run MEFN pretending we do not know α or the Dirichlet form. We then take a random sample from the fitted distribution and a random sample from the Dirichlet with

parameter α, and compare the two samples using the maximum mean discrepancy (MMD) kernel two sample test [Gretton et al., 2012], which assesses the fit quality. We take the sample size to be 300 for the two sample kernel test. Figure 3.1 shows an example of the transformation from normal (left panel) to MEFN (middle panel), and comparing that to the ground truth Dirichlet (right panel). The MEFN and ground truth Dirichlet densities shown in purple match closely, and the samples drawn (red) indeed appear to be iid draws from the same (maximum entropy) distribution in both cases.

Additionally, the middle panel of figure 3.1 shows an important cautionary tale that foreshadows our texture synthesis results (section 3.3.3). One might suppose that satisfying the moment matching constraints is adequate to produce a distribution which, if not technically the ME distribution, is still interestingly variable. The middle panel shows the failure of this intuition: in dark green, we show a network trained to simply match the moments specified above, and the resulting distribution quite poorly expresses the variability available to a distribution with these constraints, leading to samples that are needlessly similar. Given the substantial interest in using networks to learn implicit generative models (e.g., Mohamed and Lakshminarayanan [2016]), this concern is particularly relevant and highlights the importance of considering entropy.

Figure 3.2 quantitatively analyzes these results. In the left panel, for a specific choice of $\alpha = (1, 2, 3)$, we show our unbiased entropy estimate of the MEFN distribution p_θ as a function of the number of SGD iterations (red), along with the ground truth maximum entropy $H(p^*)$ (green line). Note that the MEFN stabilizes at the correct value (as a stochastic estimator, variance around that value is expected). In the middle panel, we show the distribution of MMD values for the kernel two sample test, as well as the observed statistic for the MEFN (red) and for a randomly chosen Dirichlet distribution (gray; chosen to be close to the true optimum, making a conservative comparison). The MMD test does not reject MEFN as being different from the true ME distribution p^*, but it does reject a Dirichlet whose KL to the true p^* is small

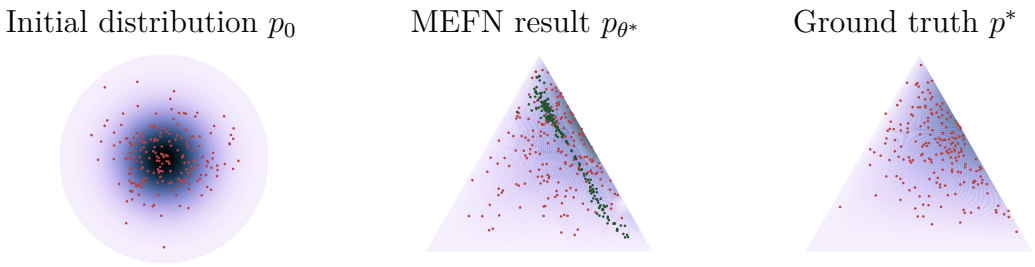


Figure 3.1: Example results from the ME problem with known Dirichlet ground truth. *Left panel*: The normal density p_0 (purple) and iid samples from p_0 (red points). *Middle panel*: The MEFN transforms p_0 to the desired maximum entropy distribution p_{θ^*} on the simplex (calculated density p_{θ^*} in purple). Truly iid samples are easily drawn from p_{θ^*} (red points) by drawing from p_0 and mapping those points through f_{θ^*}. Shown in the middle panel are the same points in the top left panel mapped through f_{θ^*}. Samples corresponding to training the same network as MEFN to simply match the specified moments (ignoring entropy) are also shown (dark green points; see text). *Right panel*: The ground truth (in this example, known to be Dirichlet) distribution in purple, and iid samples from it in red.

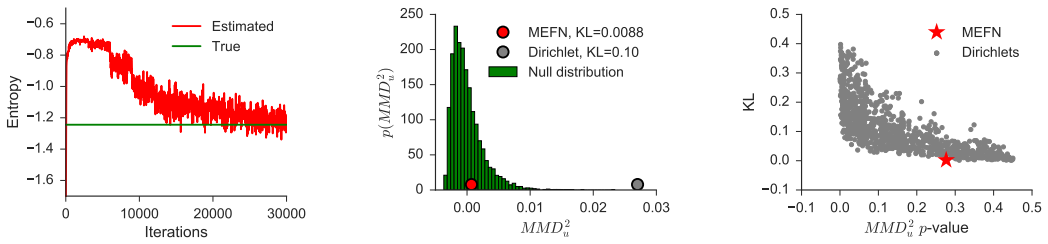


Figure 3.2: Quantitative analysis of simulation results. See text for description.

(see legend). In the right panel, for many different Dirichlets in a small grid around a single true p^*, the kernel two sample test statistic is computed, the MMD p-value is calculated, as is the KL to the true distribution. We plot a scatter of these points in grey, and we plot the particular MEFN solution as a red star. We see that for other Dirichlets with similar KL to the true distribution as the MEFN distribution, the p-values seem uniform, meaning that the KL to the true is indeed very small. Again this is conservative, as the grey points have access to the known Dirichlet form, whereas the MEFN considered the entire space (within its network capacity) of $\mathscr{S}$ supported distributions. Given this fact, the performance of MEFN is impressive.

3.3.2 Risk-Neutral Asset Pricing

We extract the risk-neutral asset price probability distribution based on option prices, an active and interesting area for ME models. We give a brief introduction of the problem and refer interested readers to see Buchen and Kelly [1996] for a more detailed explanation. Denoting S_t as the price of an asset at time t, the buyer of a European call option for the stock that expires at time t_e with strike price K will receive a payoff of $c_K = (S_{t_e} - K)_+ = \max(S_{t_e} - K, 0)$ at time t_e. Under the efficient market assumption, the risk-neutral probability distribution for the stock price at time t_e satisfies:

$$c_K = F(t_e) E_q[(S_{t_e} - K)_+], \tag{3.9}$$

where $F(t_e)$ is the risk-free discount factor and q is the risk-neutral measure. We also have that, under the risk-neutral measure, the current stock price S_0 is the discounted expected value of S_{t_e}:

$$S_0 = F(t_e)E_q(S_{t_e}). \tag{3.10}$$

When given m options that expire at time t_e with strikes $K_1, ..., K_m$ and prices $c_{K_1}, ..., c_{K_m}$, we get m expectation constraints on $q(S_{t_e})$ from equation 3.9, together with equation 3.10, we have $m + 1$ expectation constraints in total. With that partial knowledge we can approximate $q(S_{t_e})$, which is helpful for understanding the market expected volatility and identify mispricing in option markets, etc.

Inferring the risk-neutral density of asset price from a finite number of option prices is an important question in finance and has been studied extensively [Buchen and Kelly, 1996, Borwein et al., 2003, Bondarenko, 2003, Figlewski, 2008]. One popular method proposed by Buchen and Kelly [1996] estimates the probability density as the maximum entropy distribution satisfying the expectation constraints and a positivity support constraint by fitting a Gibbs distribution, which results in a piece-wise linear log density:

$$p(x) \propto \exp\left\{\eta_0 x + \sum_{i=1}^{m} \eta_i (x - K_i)_+\right\} \mathbb{1}\left(x \geq 0\right) \tag{3.11}$$

and optimize the distribution with numerical methods. Here we compare the performance of the MEFN algorithm with the method proposed in Buchen and Kelly [1996]. To enforce the positivity constraint we choose $g(x) = e^{ax+b}$, where a and b are additional parameters.

We collect the closing price of European call options on Nov. 1 2016 for the stock AAPL (Apple inc.) that expires on t_e = Jun. 16 2017. We use $m = 4$ of the options with highest trading volume as training data and the rest as testing data. On the left panel of figure 3.3, we show the fitted risk-neutral density of S_{t_e} by MEFN (red line) with that of the fitted Gibbs distribution result (blue line). We find that while the distributions share similar location and variability, the distribution inferred by MEFN is smoother and arguably more plausible. In the middle panel we show a Q-Q plot

of the quantiles of the MEFN and Gibbs distributions. We can see that the quantile pairs match the identity closely, which should happen if both methods recovered the exact same distribution. This highlights the effectiveness of MEFN. There does exist a small mismatch in the tails: the distribution inferred by MEFN has slightly heavier tails. This mismatch is difficult to interpret: given that both the Gibbs and MEFN distributions are fit with option price data (and given that one can observe at most one value from the distribution, namely the stock price at expiration), it is fundamentally unclear which distribution is superior, in the sense of better capturing the true ME distribution's tails. On the right panel we show the fitted option price for the two fitted distributions (for each strike price, we can recover the fitted option price by equation 3.9). We noted that the fitted option price and strike price lines for both methods are very similar (they are mostly indiscernible on the right panel of figure 3.3). We also compare the fitted performance on the test data by computing the root mean square error for the fitted and test data. We observe that the predictive performances for both methods are comparable.

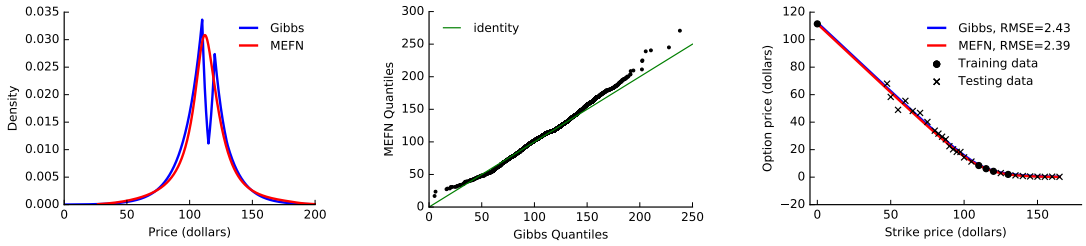


Figure 3.3: Constructing risk-neutral measure from observed option price. *Left panel*: fitted risk-neutral measure by Gibbs and MEFN method. *Middle panel*: Q-Q plot for the quantiles from the distributions on the left panel. *Right panel*: observed and fitted option price for different strikes.

We note that for this specific application, there are practical concerns such as the microstructure noise in the data and inefficiency in the market, etc. Applying a

pre-processing procedure and incorporating prior assumptions can be helpful for getting a more full-fledged method (see e.g. Figlewski [2008]). Here we mainly focus on illustrating the ability of the MEFN method to approximate the ME distribution for non-typical distributions. Future work for this application includes fitting a risk-neutral distribution for multi-dimensional assets by incorporating dependence structure on assets.

3.3.3 Modeling Images of Textures

Constructing generative models to generate random images with certain texture structure is an important task in computer vision. A line of texture synthesis research proceeds by first extracting a set of features that characterizes the target texture and then generate images that match the features. The seminal work of Zhu et al. [1998] proposes constructing texture models under the ME framework, where features (or filters) of the given texture image are adaptively added in the model and a Gibbs distribution whose expected feature matches the target texture is learnt. One major difficulty with the method is that both model learning and image generation involve sampling from a complicated Gibbs distribution. More recent works exploit more complicated features [Portilla and Simoncelli, 2000, Gatys et al., 2015, Ulyanov et al., 2016]. Ulyanov et al. [2016] propose the *texture network* (see section 2.3), which uses a texture loss function by using the Gram matrices of the outputs of some convolutional layers of a pre-trained deep neural network for object recognition.

While the use of these complicated features does provide high-quality synthetic texture images, that work focuses exclusively on generating images that match these feature (moments). Importantly, this network focuses only on generating feature-matching images without using the ME framework to promote the diversity of the samples. Doing so can be deeply problematic: in figure 3.1 (middle panel), we showed the lack of diversity resulting from only moment matching in that Dirichlet setting, and further we note that the extreme pathology would result in a point mass on the training

image – a global optimum for this objective, but obviously a terrible generative model for synthesizing textures. Ideally, the MEFN will match the moments *and* promote sample diversity.

We applied MEFN to texture synthesis with an RGB representation of the 224×224 pixel images , $x \in \mathscr{X} = [0,1]^D$, where $D = 224 \times 224 \times 3$. We follow Ulyanov et al. [2016] to create a texture loss measure $T : [0,1]^D \to \mathbb{R}$, and aim to sample a diverse set of images with small moment violation. For the transformation family $\mathscr{F}$ we use the real NVP network structure proposed in Dinh et al. [2016] (see section 2.4). For fair comparison, we use the same real NVP structure for both[1].

As is shown in top row of figure 3.4, both methods generate visually pleasing images capturing the texture structure well. The bottom row of figure 3.4 shows that texture cost (left panel) is similar for both methods, while MEFN generates figures with much larger entropy than the texture network formulation (middle panel), which is desirable (as previously discussed). The bottom right panel of figure 3.4 compares the marginal distribution of the RGB values sampled from the networks: we found that MEFN generates a more variable distribution of RGB values than the texture net.

We compute in table 3.1 the average pairwise Euclidean distance between randomly sampled images ($d_{L^2} = \text{mean}_{i \neq j} \|x_i - x_j\|_2^2$), and MEFN gives higher d_{L^2}, quantifying diversity across images. We also consider an ANOVA-style analysis to measure the diversity of the images, where we think of the RGB values for the same pixel across multiple images as a group, and compute the within and between group variance. Specifically, denoting x_i^d as the pixel value for a specific pixel $d = 1, ..., D$ for an image $i = 1, ..., n$. We partition the total sum of square $\text{SST} = \sum_{i,d}(x_i^d - \bar{x})^2$ as the within group error $\text{SSW} = \sum_{i,d}(x_i^d - \bar{x}^d)^2$ and between group error $\text{SSB} = \sum_{i,d} n(\bar{x}^d - \bar{x})^2$, where $\bar{x}$ and $\bar{x}^d$ are the mean pixel values across all data and for a specific pixel

[1] Ulyanov et al. [2016] use a quite different generative network structure, which is not invertible and is therefore infeasible for entropy evaluation, so we replace their generative network by the real NVP structure.

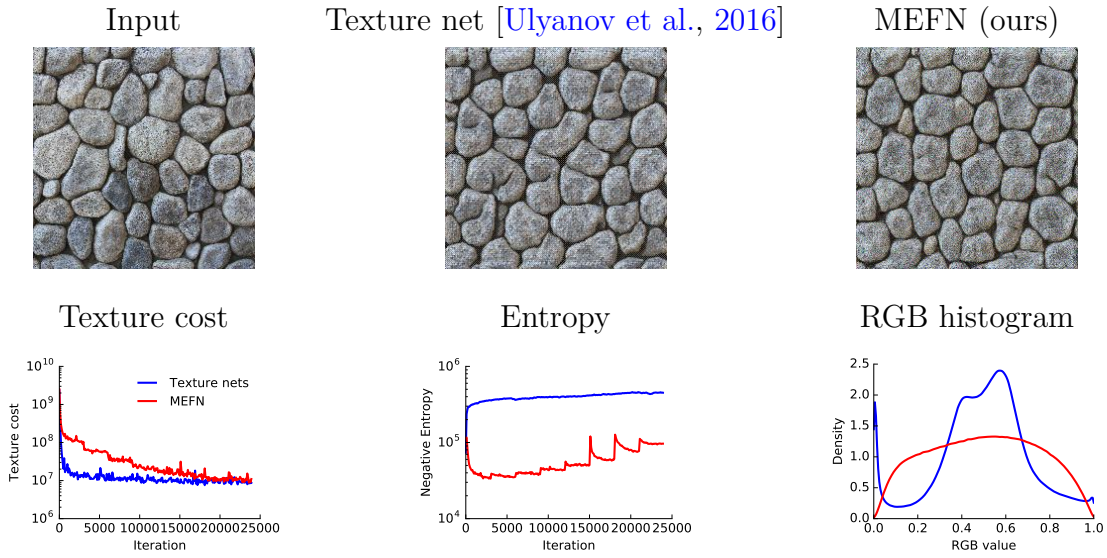


Figure 3.4: Analysis of texture synthesis experiment. See text for description.

d. Ideally we want the samples to exhibit large variability across images (large SSW, within a group/pixel) and no structure in the mean image (small SSB, across groups/pixels). Indeed, MEFN has a larger SSW, implying higher variability around the mean image, a smaller SSB, implying the stationarity of the generated samples, and a larger SST, implying larger total variability also. MEFN produces images that are conclusively more variable without sacrificing the quality of the texture, implicating the broad utility of ME.

Table 3.1: Quantitative measure of image diversity using 20 randomly sampled images.

Method	d_{L^2}	SST	SSW	SSB
Texture net	11534	128680	109577	19103
MEFN	17014	**175604**	**161639**	**13964**

We tried our texture modeling approach with many different textures, and although MEFN samples don't always exhibit more visual diversity than samples obtained from the texture network, they always have more entropy as in figure 3.4. Figure 3.5 shows

two positive examples, i.e. textures in which samples from MEFN do exhibit higher visual diversity than those from the texture network, as well as a negative example, in which MEFN achieves less visual diversity than the texture network, regardless of the fact that MEFN samples do have larger entropy. We hypothesize that this curious behavior is due to the optimization achieving a local optimum in which the brick boundaries and dark brick locations are not diverse but the entropy within each brick is large. It should also be noted that among the experiments that we ran, this was the only negative example that we got, and that slightly modifying the hyperparameters caused the issue to disappear.

3.4 Conclusions

In this chapter we propose a general framework for fitting ME models. This approach is novel and has three key features. First, by learning a transformation of a simple distribution rather than the distribution itself, we are able to avoid explicitly computing an intractable normalizing constant for the ME distribution. Second, by combining stochastic optimization with the augmented Lagrangian method, we can fit the model efficiently, allowing us to evaluate the ME density of any point simply and accurately. Third, critically, this construction allows us to trivially sample iid from a ME distribution, extending the utility and efficiency of the ME framework more generally. Also, accuracy equivalent to the classic Gibbs approach is in itself a contribution (owing to these other features). We illustrate the MEFN in both a simulated case with known ground truth and real data examples.

There are a few recent works encouraging sample diversity in the setting of texture modeling. Ulyanov et al. [2017] extended Ulyanov et al. [2016] by adding a penalty term using the Kozachenko-Leonenko estimator Kozachenko and Leonenko [1987] of entropy. Their generative network is an arbitrary deep neural network rather than a normalizing flow, which is more flexible but cannot give the probability density of each sample

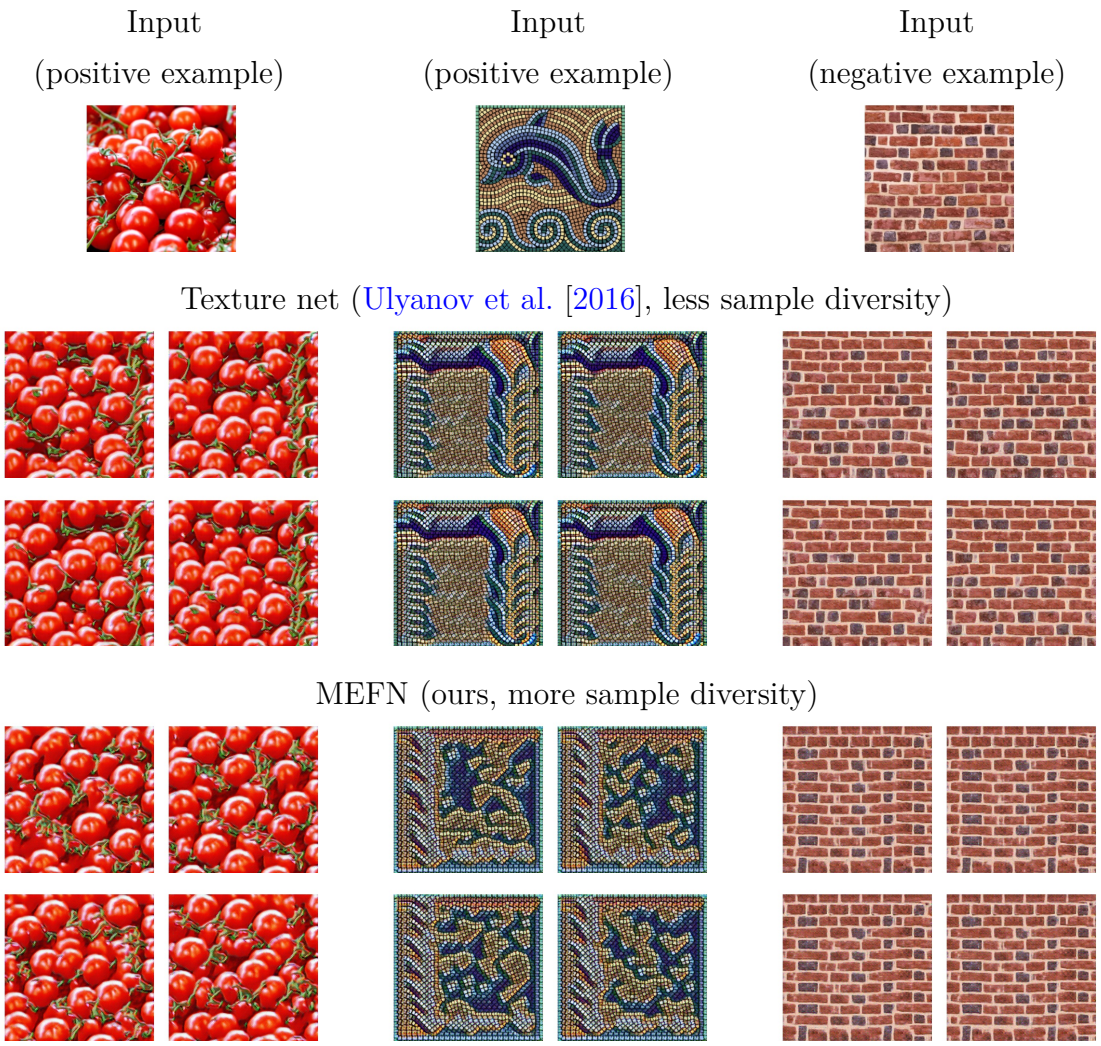


Figure 3.5: MEFN and texture network samples.

easily so as to compute an unbiased estimator of the entropy. Kozachenko-Leonenko is a biased estimator for entropy and requires a fairly large number of samples to get good performance in high-dimensional settings, hindering the scalability and accuracy of the method; indeed, our choice of normalizing flow networks was driven by these practical issues with Kozachenko-Leonenko. Lu et al. [2016] extended Zhu et al. [1998] by using a more flexible set of filters derived from a pre-trained deep neural networks, and using parallel MCMC chains to learn and sample from the Gibbs distribution.

Running parallel MCMC chains results in diverse samples but can be computationally intensive for generating each new sample image. Our MEFN framework enables truly iid sampling with the ease of a feed forward network.

Chapter 4

The Continuous Bernoulli

4.1 Introduction

Variational autoencoders (see section 2.5) have quickly become a central tool in machine learning, applicable to a broad range of data types and latent variable models. By far the most common first step, taken by seminal papers and by core software libraries alike, is to model MNIST data using a deep network parameterizing a Bernoulli likelihood. This practice contains what appears to be and what is often set aside as a minor inconvenience: the pixel data is $[0, 1]$ valued, not $\{0, 1\}$ as supported by the Bernoulli likelihood. In this chapter we show that, far from being a triviality or nuisance that is convenient to ignore, this error has profound importance to VAE, both qualitative and quantitative. We introduce and fully characterize a new $[0, 1]$-supported, single parameter distribution: the *continuous Bernoulli*, which patches this pervasive bug in VAE. This distribution is not nitpicking; it produces meaningful performance improvements across a range of metrics and datasets, including sharper image samples, and suggests a broader class of performant VAE.

Variational autoencoders have become a central tool for probabilistic modeling of complex, high dimensional data, and have been applied across image generation [Gregor et al., 2015], text generation [Hu et al., 2017], neuroscience [Gao et al., 2016],

chemistry [Gómez-Bombarelli et al., 2018], and more. One critical choice in the design of any VAE is the choice of likelihood (decoder) distribution, which stipulates the stochastic relationship between latents and observables. Consider then using a VAE to model the MNIST dataset, by far the most common first step for introducing and implementing VAE. An apparently innocuous practice is to use a Bernoulli likelihood to model this $[0, 1]$-valued data (grayscale pixel values), in disagreement with the $\{0, 1\}$ support of the Bernoulli distribution. Though doing so will not throw an obvious type error, the implied object is no longer a coherent probabilistic model, due to a neglected normalizing constant. This practice is extremely pervasive in the VAE literature, including the seminal work of Kingma and Welling [2013] (who, while aware of it, set it aside as an inconvenience), highly-cited follow up work (for example [Larsen et al., 2015, Sønderby et al., 2016, Jiang et al., 2016, Dilokthanakul et al., 2016] to name but a few), VAE tutorials [Doersch, 2016, tut], including those in hugely popular deep learning frameworks such as PyTorch [Paszke et al., 2017] and Keras [Chollet et al., 2015], and more.

In this chapter, whose content is in Loaiza-Ganem and Cunningham [2019], we introduce and fully characterize the *continuous Bernoulli* distribution (section 4.2), both as a means to study the impact of this widespread modeling error, and to provide a proper VAE for $[0, 1]$-valued data. Before these details, let us ask the central question: *who cares*?

First, theoretically, ignoring normalizing constants is unthinkable throughout most of probabilistic machine learning: these objects serve a central role in restricted Boltzmann machines [Smolensky, 1986, Hinton, 2002], graphical models [Koller et al., 2009, Pearl, 1982, Murphy et al., 1999, Wainwright and Jordan, 2008], maximum entropy modeling [Jaynes, 1957, Malouf, 2002, Loaiza-Ganem et al., 2017], the "Occam's razor" nature of Bayesian models [MacKay, 2003], and much more.

Second, one might suppose this error can be interpreted or fixed via data augmentation, binarizing data, stipulating a different lower bound, or as a nonprobabilistic

model with a "negative binary cross-entropy" objective. Section 4.3 explores these possibilities and finds them wanting. Also, one might be tempted to call the Bernoulli VAE a toy model or a minor point. Let us avoid that trap: MNIST is likely the single most widely used dataset in machine learning, and VAE is quickly becoming one of our most popular probabilistic models.

Third, and most importantly, empiricism; section 4.4 shows three key results: *(i)* as a result of this error, we show that the Bernoulli VAE significantly underperforms the continuous Bernoulli VAE across a range of evaluation metrics, models, and datasets; *(ii)* a further unexpected finding is that this performance loss is significant even when the data is close to binary, a result that becomes clear by consideration of continuous Bernoulli limits; and *(iii)* we further compare the continuous Bernoulli to beta likelihood and Gaussian likelihood VAE, again finding the continuous Bernoulli performant.

All together this work suggests that careful treatment of data type – neither ignoring normalizing constants nor defaulting immediately to a Gaussian likelihood – can produce optimal results when modeling some of the most core datasets in machine learning.

4.2 The Continuous Bernoulli Distribution

When using a Bernoulli VAE to model binary data $x_1, \dots, x_R \in \{0, 1\}^D$, the reconstruction term in the ELBO 2.25 is:

$$E_{q_\phi(z_r|x_r)}[\log p_\theta(x_r|z_r)] = E_{q_\phi(z_r|x_r)}\Big[\sum_{d=1}^{D} x_{r,d} \log \lambda_{\theta,d}(z_r) + (1 - x_{r,d}) \log(1 - \lambda_{\theta,d}(z_r))\Big] \tag{4.1}$$

where $x_{r,d}$ and $\lambda_{\theta,d}(z_r)$ are the d-th coordinates of x_r and $\lambda_\theta(z_r)$, respectively. However, critically, Bernoulli likelihoods and the reconstruction term of equation 4.1 are commonly used for $[0, 1]$-valued data, which loses the interpretation of probabilistic inference. To clarify, hereafter we denote the Bernoulli distribution as $\widetilde{p}(x|\lambda) = \lambda^x(1-\lambda)^{1-x}$ to emphasize the fact that it is an unnormalized distribution (when evaluated over

$[0, 1]$). We will also make this explicit in the ELBO, writing $\mathscr{E}(\widetilde{p}, \theta, \phi)$ to denote that the reconstruction term of equation 4.1 is being used. When analyzing $[0, 1]$-valued data such as MNIST, the Bernoulli VAE has optimal parameter values $\theta^*(\widetilde{p})$ and $\phi^*(\widetilde{p})$; that is:

$$(\theta^*(\widetilde{p}), \phi^*(\widetilde{p})) = \underset{(\theta,\phi)}{\text{maximize}} \ \mathscr{E}(\widetilde{p}, \theta, \phi). \tag{4.2}$$

In order to analyze the implications of this modeling error, we introduce the continuous Bernoulli, a novel distribution on $[0, 1]$, which is parameterized by $\lambda \in (0, 1)$ and defined by:

$$X \sim \mathscr{CB}(\lambda) \iff p(x|\lambda) \propto \widetilde{p}(x|\lambda) = \lambda^x (1 - \lambda)^{1-x}. \tag{4.3}$$

We now fully characterize this distribution. Note that if $X \sim \mathscr{CB}(0.5)$ then X follows a uniform distribution on $[0, 1]$, which makes the proofs of the following propositions trivial for that case. In the following proofs we thus assume that $\lambda \neq 0.5$.

Proposition 1 ($\mathscr{CB}$ density and mean): The continuous Bernoulli distribution is well defined, that is, $0 < \int_0^1 \widetilde{p}(x|\lambda)dx < \infty$ for every $\lambda \in (0, 1)$. Furthermore, if $X \sim \mathscr{CB}(\lambda)$, then the density function of X and its expected value are:

$$p(x|\lambda) = C(\lambda)\lambda^x(1 - \lambda)^{1-x}, \text{where } C(\lambda) = \begin{cases} \dfrac{2\tanh^{-1}(1 - 2\lambda)}{1 - 2\lambda} & \text{if } \lambda \neq 0.5 \\ 2 & \text{otherwise} \end{cases} \tag{4.4}$$

$$\mu(\lambda) := E[X] = \begin{cases} \dfrac{\lambda}{2\lambda - 1} + \dfrac{1}{2\tanh^{-1}(1 - 2\lambda)} & \text{if } \lambda \neq 0.5 \\ 0.5 & \text{otherwise} \end{cases} \tag{4.5}$$

Proof:

We define the function $\widetilde{F}(\cdot|\lambda) : [0, 1] \to [0, 1]$ as:

$$\widetilde{F}(x|\lambda) = -\frac{\lambda^x(1 - \lambda)^{1-x} + \lambda - 1}{2\tanh^{-1}(1 - 2\lambda)} \tag{4.6}$$

It is straightforward to verify that $\widetilde{F}'(x|\lambda) = \widetilde{p}(x|\lambda)$, so that for every $x \in [0, 1]$:

$$\int_0^x \lambda^s(1 - \lambda)^{1-s}ds = \widetilde{F}(x|\lambda) - \widetilde{F}(0|\lambda) \tag{4.7}$$

In particular:

$$0 < C(\lambda) = \int_0^1 \lambda^s (1-\lambda)^{1-s} ds = \widetilde{F}(1|\lambda) - \widetilde{F}(0|\lambda) = \frac{2\tanh^{-1}(1-2\lambda)}{1-2\lambda} < \infty \quad (4.8)$$

And thus $p(x|\lambda) = C(\lambda)\widetilde{p}(x|\lambda)$. Furthermore:

$$\mu(\lambda) = \int_0^1 C(\lambda) x \lambda^x (1-\lambda)^{1-x} dx \quad (4.9)$$

$$= C(\lambda) \frac{(\lambda-1)(1-\lambda)^{-x}\lambda^x(x\log(1-\lambda) - x\log\lambda + 1)}{(\log(1-\lambda) - \log\lambda)^2}\Bigg|_{x=0}^{x=1} \quad (4.10)$$

$$= \frac{\lambda}{2\lambda - 1} + \frac{1}{2\tanh^{-1}(1-2\lambda)} \quad (4.11)$$

□

Figure 4.1 shows $\log C(\lambda)$, $p(x|\lambda)$, and $\mu(\lambda)$. Some notes warrant mention: *(i)* unlike the Bernoulli, the mean of the continuous Bernoulli is not λ; *(ii)* however, like for the Bernoulli, $\mu(\lambda)$ is increasing on λ and goes to 0 or 1 when λ goes to 0 or 1; *(iii)* the continuous Bernoulli is *not* a beta distribution (see appendix B for a detailed analysis of the differences), nor any other $[0, 1]$-supported distribution we are aware of; *(iv)* $C(\lambda)$ and $\mu(\lambda)$ are continuous functions of λ; and *(v)* the log normalizing constant $\log C(\lambda)$ is well characterized but numerically unstable close to $\lambda = 0.5$, so our implementation uses a Taylor approximation near that critical point to calculate $\log C(\lambda)$ to high numerical precision.

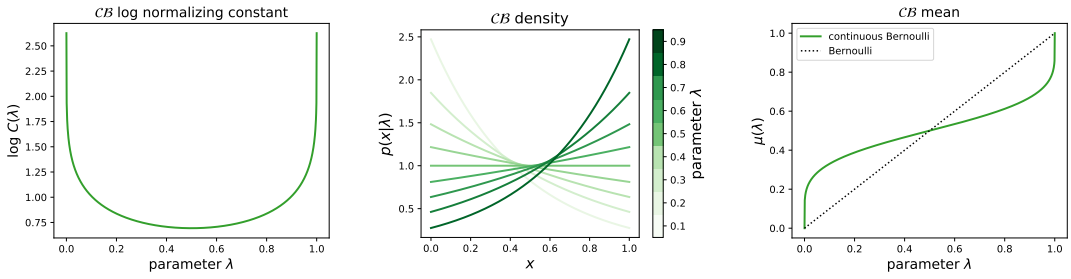


Figure 4.1: Continuous Bernoulli log normalizing constant (*left panel*), pdf (*middle panel*) and mean (*right panel*).

Proposition 2 ($\mathscr{CB}$ additional properties): The continuous Bernoulli forms

an exponential family with sufficient statistic $T(x) = x$ and natural parameter $\log(\lambda/(1-\lambda))$. Furthermore, if $X \sim \mathscr{CB}(\lambda)$, the following hold:

1. The variance of X is:

$$\text{var}(X) = \begin{cases} \dfrac{(\lambda-1)\lambda}{(1-2\lambda)^2} + \dfrac{1}{(2\tanh^{-1}(1-2\lambda))^2} & \text{if } \lambda \neq 0.5 \\ 1/12 & \text{otherwise} \end{cases} \tag{4.12}$$

2. The cumulative distribution function of X is:

$$F(x|\lambda) = \begin{cases} \dfrac{\lambda^x(1-\lambda)^{1-x} + \lambda - 1}{2\lambda - 1} & \text{if } \lambda \neq 0.5 \\ x & \text{otherwise} \end{cases} \tag{4.13}$$

3. The inverse cumulative distribution function of X is:

$$F^{-1}(u|\lambda) = \begin{cases} \dfrac{\log(u(2\lambda-1)+1-\lambda) - \log(1-\lambda)}{\log(\lambda) - \log(1-\lambda)} & \text{if } \lambda \neq 0.5 \\ u & \text{otherwise} \end{cases} \tag{4.14}$$

   and thus we immediately have the reparameterization $X = F^{-1}(U)$ (in distribution), where $U \sim \text{Uniform}(0, 1)$.

4. The characteristic function of X is:

$$\varphi_X(t) = \begin{cases} C(\lambda)\dfrac{i(1-\lambda e^{it} - \lambda)}{t + 2i\tanh^{-1}(1-2\lambda)} & \text{if } \lambda \neq 0.5 \text{ and } t \neq 0 \\ \dfrac{e^{it}-1}{it} & \text{if } \lambda = 0.5 \text{ and } t \neq 0 \\ 1 & \text{if } t = 0 \end{cases} \tag{4.15}$$

5. The entropy of X is:

$$H(X) = \mu(\lambda)\log\frac{1-\lambda}{\lambda} - \log C(\lambda) - \log(1-\lambda) \tag{4.16}$$

6. The KL divergence between two continuous Bernoulli distributions is:

$$KL\Big(p(x|\lambda_1)||p(x|\lambda_2)\Big) = \mu(\lambda_1)\log\frac{\lambda_1(1-\lambda_2)}{\lambda_2(1-\lambda_1)} + \log\frac{C(\lambda_1)(1-\lambda_1)}{C(\lambda_2)(1-\lambda_2)} \tag{4.17}$$

7. $C(\lambda)$ is convex.

Proof:

1. Clearly:

$$p(x|\lambda) = \mathbb{1}(x \in [0,1]) e^{x \log \frac{\lambda}{1-\lambda} + (\log(1-\lambda) + \log C(\lambda))} \tag{4.18}$$

So that the continuous Bernoulli indeed forms an exponential family with sufficient statistic $T(x) = x$ and natural parameter $\log(\lambda/(1-\lambda))$. □

2. Also:

$$\begin{aligned} \text{var}(X) &= \int_0^1 C(\lambda)(x - \mu(\lambda))^2 \lambda^x (1-\lambda)^{1-x} dx & (4.19)\\ &= -C(\lambda) \frac{(1-\lambda)^{1-x}\lambda^x \left(4(\mu(\lambda) - x)\tanh^{-1}(1-2\lambda)\left((\mu(\lambda) - x)\tanh^{-1}(1-2\lambda) - 1\right) + 2\right)}{\left(2\tanh^{-1}(1-2\lambda)\right)^3}\Bigg|_{x=0}^{x=1} & (4.20)\\ &= \frac{(\lambda-1)\lambda}{(1-2\lambda)^2} + \frac{1}{(2\tanh^{-1}(1-2\lambda))^2} & (4.21) \end{aligned}$$

□

3. Furthermore:

$$F(x) = C(\lambda)\widetilde{F}(x|\lambda) = \frac{\lambda^x(1-\lambda)^{1-x} + \lambda - 1}{2\lambda - 1} \tag{4.22}$$

The above equation, equated to u, can easily be inverted algebraically to obtain:

$$F^{-1}(u|\lambda) = \frac{\log(u(2\lambda - 1) + 1 - \lambda) - \log(1-\lambda)}{\log(\lambda) - \log(1-\lambda)} \tag{4.23}$$

□

4. Since $\varphi_X(t) = E[e^{itX}]$, the case where $t = 0$ is trivial. We have:

$$\begin{aligned} E[\cos(tX)] &= \int_0^1 \cos(tx) C(\lambda) \lambda^x (1-\lambda)^{1-x} dx & (4.24)\\ &= C(\lambda) \frac{(\lambda-1)(1-\lambda)^{-x}\lambda^x \left((\log(1-\lambda) - \log\lambda)\cos(tx) - t\sin(tx)\right)}{t^2 + \log^2(1-\lambda) + \log^2\lambda - 2\log(1-\lambda)\log\lambda}\Bigg|_{x=0}^{x=1} & (4.25) \end{aligned}$$

Also:

$$\begin{aligned} E[\sin(tX)] &= \int_0^1 \sin(tx) C(\lambda) \lambda^x (1-\lambda)^{1-x} dx & (4.26)\\ &= C(\lambda) \frac{(\lambda-1)(1-\lambda)^{-x}\lambda^x \left((\log(1-\lambda) - \log\lambda)\sin(tx) - t\cos(tx)\right)}{t^2 + \log^2(1-\lambda) + \log^2\lambda - 2\log(1-\lambda)\log\lambda}\Bigg|_{x=0}^{x=1} & (4.27) \end{aligned}$$

Combining the above two expressions and simplifying, we get:

$$\varphi_X(t) = E[e^{itX}] = E[\cos(tX) + i\sin(tX)] = C(\lambda)\frac{i(1-\lambda e^{it}-\lambda)}{t + 2i\tanh^{-1}(1-2\lambda)} \tag{4.28}$$

□

5. We have:

$$H(X) = -E[\log p(X|\lambda)] = -E[\log C(\lambda) + X\log\lambda + (1-X)\log(1-\lambda)] \tag{4.29}$$

$$= -\log C(\lambda) - \mu(\lambda)\log\lambda - (1-\mu(\lambda))\log(1-\lambda) \tag{4.30}$$

$$= \mu(\lambda)\log\frac{1-\lambda}{\lambda} - \log C(\lambda) - \log(1-\lambda) \tag{4.31}$$

□

6. The KL divergence between $X_1 \sim \mathscr{CB}(\lambda_1)$ and $X_2 \sim \mathscr{CB}(\lambda_2)$ is given by:

$$KL\Big(p(x|\lambda_1)||p(x|\lambda_2)\Big) = E\Big[\log\frac{p(X_1|\lambda_1)}{p(X_1|\lambda_2)}\Big] \tag{4.32}$$

$$= -E[\log C(\lambda_2) + X_1\log\lambda_2 + (1-X_1)\log(1-\lambda_2)] - H(X_1) \tag{4.33}$$

$$= \mu(\lambda_1)\log\frac{\lambda_1(1-\lambda_2)}{\lambda_2(1-\lambda_1)} + \log\frac{C(\lambda_1)(1-\lambda_1)}{C(\lambda_2)(1-\lambda_2)} \tag{4.34}$$

□

7. Finally, to show that $C(\lambda)$ is convex, we first show that the function:

$$g(\nu) = \begin{cases} \dfrac{\tanh^{-1}(\nu)}{\nu}, & \text{if } \nu \neq 0 \\ 1, & \text{otherwise} \end{cases} \tag{4.35}$$

is convex in $(-1, 1)$. The Taylor series expansion of $\tanh^{-1}(\nu)$ is given by:

$$\tanh^{-1}(\nu) = \sum_{k=0}^{\infty}\frac{1}{2k+1}\nu^{2k+1} \tag{4.36}$$

So that $g(\nu)$ admits the expansion:

$$g(\nu) = \sum_{k=0}^{\infty}\frac{1}{2k+1}\nu^{2k} \tag{4.37}$$

Then:

$$g''(\nu) = \sum_{k=1}^{\infty}\frac{2k(2k-1)}{2k+1}\nu^{2k-2} \geq 0 \tag{4.38}$$

So that g is convex. Since $C(\lambda) = 2g(1 - 2\lambda)$, $C(\lambda)$ is convex as well. □

Proposition 3 ($\mathscr{CB}$ Bernoulli limit): $\mathscr{CB}(\lambda) \xrightarrow{\lambda \to 0} \delta(0)$ and $\mathscr{CB}(\lambda) \xrightarrow{\lambda \to 1} \delta(1)$ in distribution; that is, the continuous Bernoulli becomes a Bernoulli in the limit.

Proof: We have that $F(x|\lambda) \xrightarrow{\lambda \to 0} \mathbb{1}(x > 0)$ and $F(x|\lambda) \xrightarrow{\lambda \to 1} \mathbb{1}(x = 1)$, which concludes the proof. □

This proposition might at a first glance suggest that, as long as the estimated parameters are close to 0 or 1 (which should happen when the data is close to binary), then the practice of erroneously applying a Bernoulli VAE should be of little consequence. However, the above reasoning is wrong, as it ignores the shape of $\log C(\lambda)$: as $\lambda \to 0$ or $\lambda \to 1$, $\log C(\lambda) \to \infty$ (figure 4.1, left). Thus, if data is close to binary, the term neglected by the Bernoulli VAE becomes even more important, the exact opposite conclusion than the naive one presented above.

Proposition 4 ($\mathscr{CB}$ normalizing constant bound): $C(\lambda) \geq 2$, with equality if and only if $\lambda = 0.5$. And thus it follows that, for any x, λ, we have $\log p(x|\lambda) > \log \widetilde{p}(x|\lambda)$.

Proof:

It is straightforward to verify, using L'Hôpital's rule, that:

$$\frac{\partial}{\partial \lambda} C(\lambda)\Big|_{\lambda=0.5} = 0 \tag{4.39}$$

Thus, 0.5 is a local optimum of $C(\lambda)$. Since $C(\lambda)$ is convex, $\lambda = 0.5$ is a global minimizer of $C(\lambda)$, and since $C(0.5) = 2$, this finishes the proof. □

This proposition allows us to interpret $\mathscr{E}(\widetilde{p}, \theta, \phi)$ as a *lower* lower bound of the log likelihood (section 4.3).

Proposition 5 ($\mathscr{CB}$ maximum likelihood): For an observed sample $x_1, \dots, x_R \sim_{iid} \mathscr{CB}(\lambda)$, the maximum likelihood estimator $\hat{\lambda}$ of λ is such that:

$$\mu(\hat{\lambda}) = \frac{1}{R} \sum_{r=1}^{R} x_r \tag{4.40}$$

Proof:

The proof follows from the fact that doing maximum likelihood in an exponential family reduces to doing moment matching on the sufficient statistic. □

Beyond characterizing a novel and interesting distribution, these propositions now allow full analysis of the error in applying a Bernoulli VAE to the wrong data type.

4.3 The Continuous Bernoulli VAE

We define the continuous Bernoulli VAE analogously to the Bernoulli VAE:

$$\begin{cases} Z_r \sim \mathcal{N}(0, I_M) \\ X_r | Z_r \sim \mathscr{CB}\left(\lambda_\theta(Z_r)\right) \text{ for } r = 1, \dots, R \end{cases} \tag{4.41}$$

where again $\lambda_\theta : \mathbb{R}^M \to \mathbb{R}^D$ is a neural network with parameters θ, and $\mathscr{CB}(\lambda)$ now denotes the product of D independent *continuous* Bernoulli distributions. Operationally, this results only in a change to the optimized objective; for clarity we compare the ELBO of the continuous Bernoulli VAE (top), $\mathscr{E}(p, \theta, \phi)$, to the Bernoulli VAE (bottom):

$$\mathscr{E}(p, \theta, \phi) = \sum_{r=1}^{R} -KL(q_\phi || \pi) + E_{q_\phi}\left[\sum_{d=1}^{D} x_{r,d} \log \lambda_{\theta,d}(z_r) + (1 - x_{r,d}) \log(1 - \lambda_{\theta,d}(z_r)) + \log C(\lambda_{\theta,d}(z_r))\right]$$

$$\mathscr{E}(\widetilde{p}, \theta, \phi) = \sum_{r=1}^{R} -KL(q_\phi || \pi) + E_{q_\phi}\left[\sum_{d=1}^{D} x_{r,d} \log \lambda_{\theta,d}(z_r) + (1 - x_{r,d}) \log(1 - \lambda_{\theta,d}(z_r))\right],$$

Analogously, we denote $\theta^*(p)$ and $\phi^*(p)$ as the maximizers of the continuous Bernoulli ELBO:

$$(\theta^*(p), \phi^*(p)) = \underset{(\theta,\phi)}{\text{maximize}} \ \mathscr{E}(p, \theta, \phi). \tag{4.42}$$

Immediately, a number of potential interpretations for the Bernoulli VAE come to mind, some of which have appeared in literature. We analyze each in turn.

4.3.1 Binarizing

One obvious workaround is to simply binarize any $[0, 1]$-valued data (MNIST pixel values or otherwise), so that it accords with the Bernoulli likelihood [Larochelle and Murray, 2011], a practice that has been followed in some subsequent works (e.g. [Rezende and Mohamed, 2015, Burda et al., 2015]). First, modifying data to fit a model, particularly an unsupervised model, is fundamentally problematic. Second, many $[0, 1]$-valued datasets are heavily degraded by binarization (see appendix B for CIFAR-10 samples), indicating major practical limitations.

4.3.2 Data Augmentation

Because the expectation of a Bernoulli random variable is precisely its parameter, the Bernoulli VAE might (erroneously) be assumed to be equivalent to a continuous Bernoulli VAE on an infinitely augmented dataset, obtained by sampling binary data whose mean is given by the observed data; indeed this idea is suggested by Kingma and Welling [2013][1]. However, this interpretation does not hold[2]; it would result in a reconstruction term as in the first line in the equation below, while a correct Bernoulli VAE on the augmented dataset would have a reconstruction term given by the second line (not equal, as the order of expectation can not be switched because q_ϕ depends on X_r on the second line):

$$E_{z_n \sim q_\phi(z_n|x_n)}\left[E_{X_n \sim \mathscr{B}(x_n)}\left[\sum_{d=1}^{D} X_{n,d} \log \lambda_{\theta,d}(z_n) + (1 - X_{n,d}) \log \lambda_{\theta,d}(z_n)\right]\right] \tag{4.43}$$

$$\neq E_{X_n \sim \mathscr{B}(x_n)}\left[E_{z_n \sim q_\phi(z_n|X_n)}\left[\sum_{d=1}^{D} X_{n,d} \log \lambda_{\theta,d}(z_n) + (1 - X_{n,d}) \log \lambda_{\theta,d}(z_n)\right]\right].$$

[1] see the comments in https://openreview.net/forum?id=33X9fd2-9FyZd

[2] see http://ruishu.io/2018/03/19/bernoulli-vae/ for a looser lower bound interpretation

4.3.3 Bernoulli VAE as a Different Objective

Knoblauch et al. [2019] study changing the reconstruction and/or the KL term in the ELBO. While their main focus is to obtain more robust inference, they provide a framework in which the Bernoulli VAE corresponds simply to a different (nonprobabilistic) loss. In this perspective, empirical results must determine the adequacy of this objective; section 4.4 shows the Bernoulli VAE to underperform its proper probabilistic counterpart across a range of settings.

4.3.4 Bernoulli VAE as a Lower Lower Bound

Because the continuous Bernoulli ELBO and the Bernoulli ELBO are related by:

$$\mathscr{E}(\widetilde{p}, \theta, \phi) + \sum_{r=1}^{R} \sum_{d=1}^{D} E_{z_r \sim q_\phi(z_r|x_r)}[\log C(\lambda_{\theta,d}(z_r))] = \mathscr{E}(p, \theta, \phi) \tag{4.44}$$

and recalling Proposition 4, since $\log 2 > 0$, we get that $\mathscr{E}(\widetilde{p}, \theta, \phi) < \mathscr{E}(p, \theta, \phi)$. That is, using the Bernoulli VAE results in optimizing an even-lower bound of the log likelihood than using the continuous Bernoulli ELBO. Note however that unlike the ELBO, $\mathscr{E}(\widetilde{p}, \theta, \phi)$ is not tight even if the approximate posterior matches the true posterior.

4.3.5 Mean Parameterization

The conventional maximum likelihood estimator for a Bernoulli, namely:

$$\hat{\lambda}_{\mathscr{B}} = \frac{1}{R} \sum_{r=1}^{R} x_r \tag{4.45}$$

maximizes $\widetilde{p}(x_1, ..., x_R|\lambda)$ regardless of whether data is $\{0, 1\}$ and $[0, 1]$. As a thought experiment, consider $x_1, \ldots, x_R \sim_{iid} \mathscr{CB}(\lambda)$. Proposition 5 tells us that the correct maximum likelihood estimator, $\hat{\lambda}_{\mathscr{CB}}$ is such that:

$$\mu(\hat{\lambda}_{\mathscr{CB}}) = \frac{1}{R} \sum_{r=1}^{R} x_r \tag{4.46}$$

where μ is the $\mathscr{CB}$ mean of equation 4.5. Thus, while using $\hat{\lambda}_{\mathscr{B}}$ is incorrect, one can (surprisingly) still recover the correct maximum likelihood estimator via:

$$\hat{\lambda}_{\mathscr{CB}} = \mu^{-1}(\hat{\lambda}_{\mathscr{B}}) \tag{4.47}$$

One might then (wrongly) think that training a Bernoulli VAE, and then subsequently transforming the decoder parameters with μ^{-1}, would be equivalent to training a continuous Bernoulli VAE; that is: $\lambda_{\theta^*(p)}$ might be equal to $\mu^{-1}(\lambda_{\theta^*(\widetilde{p})})$. This reasoning is incorrect: the KL term in the ELBO implies that:

$$\lambda_{\theta^*(p)}(z_r) \neq \mu^{-1}(x_r) \tag{4.48}$$

and so too:

$$\lambda_{\theta^*(\widetilde{p})}(z_r) \neq x_r \tag{4.49}$$

and as such:

$$\lambda_{\theta^*(p)} \neq \mu^{-1}(\lambda_{\theta^*(\widetilde{p})}) \tag{4.50}$$

In fact, our experiments will show that despite this flawed reasoning, applying this transformation can recover some, but not all, of the performance loss from using a Bernoulli VAE.

4.4 Experiments

We have introduced the continuous Bernoulli distribution to give a proper probabilistic VAE for $[0, 1]$-valued data. The essential question that we now address is how much, if any, improvement we achieve by making this choice.

4.4.1 MNIST

One frequently noted shortcoming of VAE (and Bernoulli VAE on MNIST in particular) is that samples from this model are blurry. As noted, the convexity of $\log C(\lambda)$ can be seen as regularizing sample values from the VAE to be more extremal; that is,

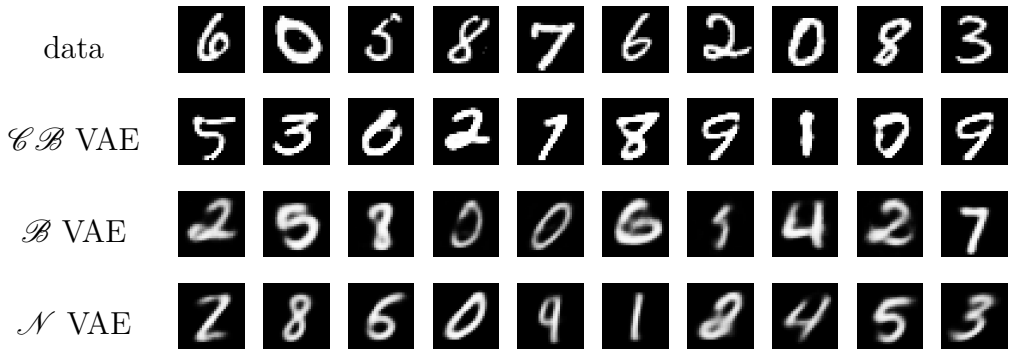


Figure 4.2: Samples from MNIST, continuous Bernoulli VAE, Bernoulli VAE, and Gaussian VAE.

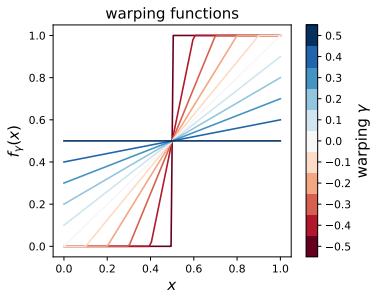
sharper. As such we first compared samples from a trained continuous Bernoulli VAE against samples from the MNIST dataset itself, from a trained Bernoulli VAE, and from a trained Gaussian VAE (namely the usual VAE model with a decoder likelihood $p_\theta(x|z) = \mathscr{N}(\eta_\theta(z), \sigma^2_\theta(z))$, where we use η to avoid confusion with μ of equation 4.5). These samples are shown in figure 4.2. In each case, as is standard, we show the parameter output by the generative/decoder network for a given latent draw: $\lambda_{\theta^*(p)}(z)$ for the $\mathscr{CB}$ VAE, $\lambda_{\theta^*(\widetilde{p})}(z)$ for the $\mathscr{B}$ VAE, and $\eta_{\theta^*}(z)$ for the $\mathscr{N}$ VAE. Qualitatively, the continuous Bernoulli VAE achieves considerably superior samples vs the Bernoulli or Gaussian VAE, owing to the effect of $\log C(\lambda)$ pushing the decoder toward sharper images. Further samples are in appendix B.

4.4.2 Warped MNIST Datasets

The most common justification for the Bernoulli VAE is that MNIST pixel values are "close" to binary. An important study is thus to ask how the performance of continuous Bernoulli VAE vs the Bernoulli VAE changes as a function of this "closeness". We formalize this concept by introducing a warping function $f_\gamma(x)$ that, depending on the warping parameter γ, transforms individual pixel values to produce a dataset that is anywhere from fully binarized (every pixel becomes $\{0, 1\}$) to fully degraded (every pixel becomes 0.5). Figure 4.3 shows f_γ for different values of γ, and the (rather

uninformative) warping equation appears next to figure 4.3.

Importantly, $\gamma = 0$ corresponds to an unwarped dataset, i.e., the original MNIST dataset. Further, note that negative values of γ warp pixel values towards being more binarized, completely binarizing it in the case where $\gamma = -0.5$, whereas positive values of γ push the pixel values towards 0.5, recovering constant data at $\gamma = 0.5$. We then train our competing VAE models on the datasets induced by different values of γ and compare the difference in performance as γ changes. Note importantly that, because γ induces different datasets, performance values should primarily be compared between VAE models at each γ value; the trend as a function of γ is not of particular interest.



$$f_\gamma(x) = \begin{cases} \mathbb{1}(x \geq 0.5), & \text{if } \gamma = -0.5 \\ \min\left(1, \max\left(0, \dfrac{x+\gamma}{1+2\gamma}\right)\right), & \text{if } \gamma \in (-0.5, 0) \\ \gamma + (1-2\gamma)x, & \text{if } \gamma \in [0, 0.5] \end{cases}$$

Figure 4.3: f_γ for different γ values.

Figure 4.4 shows the results of various models applied to these different datasets (all values are an average of 10 separate training runs). The same neural network architectures are used across this figure, with architectural choices that are quite standard (detailed in appendix B, along with training hyperparameters). The left panel shows ELBO values. In dark blue is the continuous Bernoulli VAE ELBO, namely $\mathscr{E}(p, \theta^*(p), \phi^*(p))$. In light blue is the same ELBO when evaluated on a trained Bernoulli VAE: $\mathscr{E}(p, \theta^*(\widetilde{p}), \phi^*(\widetilde{p}))$. Most importantly, note the $\gamma = 0$ values; the continuous Bernoulli VAE achieves a 300 nat improvement over the Bernoulli VAE. This finding supports the previous qualitative finding and the theoretical motivation for this work: significant quantitative gains are achieved via the continuous Bernoulli model. This finding remains true across a range of γ (dark blue above light blue

in figure 4.4), indicating that regardless of how 'close' to binary a dataset is, the continuous Bernoulli is a superior VAE model.

One might then wonder if the continuous Bernoulli is outperforming simply because the Bernoulli needs a mean correction. We thus apply μ^{-1}, namely the map from Bernoulli to continuous Bernoulli maximum likelihood estimators (equation 4.5 and section 4.3.5), and evaluate the same ELBO on $\mu^{-1}(\lambda_{\theta^*(\widetilde{p})})$ as the decoder shown in light red (figure 4.4, left). This result, which is only achieved via the introduction of the continuous Bernoulli, shows two important findings: first, that indeed some improvement over the Bernoulli VAE is achieved by post hoc correction to a continuous Bernoulli parameterization; but second, that this transformation is still inadequate to achieve the full performance of the continuous Bernoulli VAE. We also note that

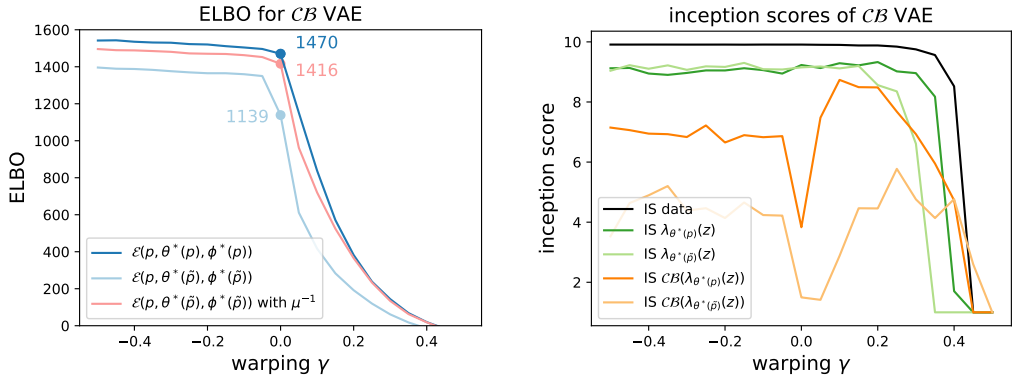


Figure 4.4: Continuous Bernoulli comparisons against Bernoulli VAE. See text for details.

we ran the same experiment with log likelihood instead of ELBO (using importance weighted estimates with $k = 100$ samples; see section 2.5.1), and the same results held (up to small numerical differences; these traces are omitted for figure clarity). We also ran the same experiment with a more complicated approximate posterior distribution using 10 layers of planar flows (see section 2.4.2), results are in the middle panel of figure 4.5. We can see that not only do the results hold: the gap between continuous Bernoulli and Bernoulli increases even further. We ran the same experiment yet again

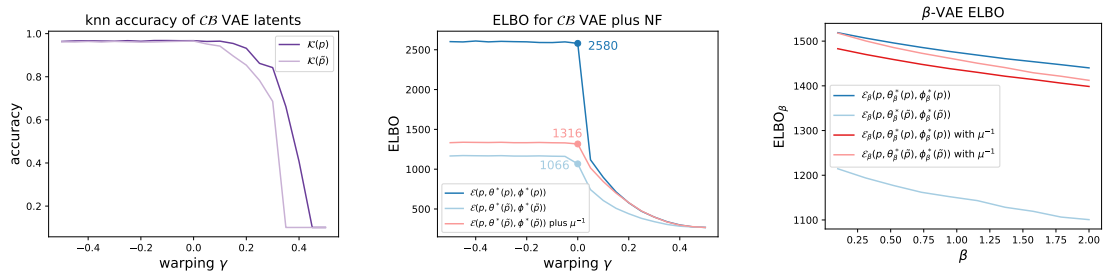


Figure 4.5: More continuous Bernoulli comparisons against Bernoulli VAE. See text for details.

for the β-VAE (see section 2.5.2), sweeping a range of β values, and the same results held, as is shown in the right panel of figure 4.5 (which shows only $\gamma = 0$). It is natural to then wonder if this performance is an artifact of ELBO and log likelihood; thus, we also evaluated the same datasets and models using different evaluation metrics. In the right panel of figure 4.4, we use the inception score (see section 2.6.1) to measure sample quality produced by the different models (higher is better). Once again, we see that including the normalizing constant produces better samples (dark traces / continuous Bernoulli lie above light traces / Bernoulli). We include that comparison on both the decoder parameters λ (dark and light green) and also samples from distributions indexed by those parameters (dark and light orange). One can imagine a variety of other parameter transformations that may be of interest; we include several in appendix B, where again we find that the continuous Bernoulli VAE outperforms its Bernoulli counterpart.

In the left panel of figure 4.5, to measure usefulness of the latent representations of these models, we compute $m_{\phi^*(p)}(x_r)$ and $m_{\phi^*(\widetilde{p})}(x_r)$ (note that m is the variational posterior mean from equation 2.24 and not the continuous Bernoulli mean) for training data and use a 15-nearest neighbor classifier to predict the test labels. The left panel of figure 4.5 shows the accuracy of the classifiers (denoted $\mathscr{K}(\phi)$) as a function of γ. Once again, the continuous Bernoulli VAE outperforms the Bernoulli VAE.

Now that the continuous Bernoulli VAE gives us a proper model on $[0, 1]$, we can

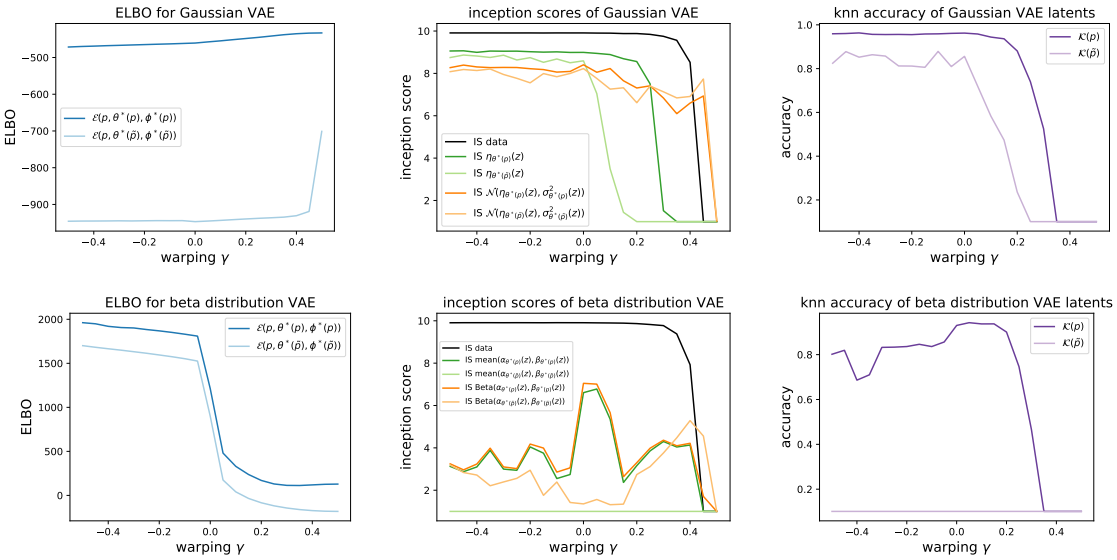


Figure 4.6: Gaussian (*top panels*) and beta (*bottom panels*) distributions comparisons between including and excluding the normalizing constants. Left panels show ELBOs, middle panels inceptions scores, and right panels 15-nearest neighbors accuracy.

also propose other natural models. Here we introduce and compare against the beta distribution VAE (not β-VAE [Higgins et al., 2017]); as the name implies, the generative likelihood is $\text{Beta}(\alpha_\theta(z), \beta_\theta(z))$. We repeated the same warped MNIST experiments using Gaussian VAE and beta distribution VAE, both including and ignoring the normalizing constants of those distributions, as an analogy to the continuous Bernoulli and Bernoulli distributions. First, figure 4.6 shows again that that ignoring the normalizing constant hurts performance in every metric and model (dark above light). Second, interestingly, we find that the continuous Bernoulli VAE outperforms both the beta distribution VAE and the Gaussian VAE in terms of inception scores, and that the beta distribution VAE dominates in terms of ELBO. This figure clarifies that the continuous Bernoulli and beta distribution are to be preferred over the Gaussian for VAE applied to $[0, 1]$ valued data, and that ignoring normalizing constants is indeed unwise.

A few additional notes warrant mention on figure 4.6. Unlike with the continuous

Bernoulli, we should not expect the Gaussian and beta normalizing constants to go to infinity as extrema are reached, so we do not observe the same patterns with respect to γ as we did with the continuous Bernoulli. Note also that the lower lower bound property of ignoring normalizing constants does not hold in general, as it is a direct consequence of the continuous Bernoulli log normalizing constant being nonnegative.

4.4.3 CIFAR-10

We repeat the same experiments as in the previous section on the CIFAR-10 dataset (without common preprocessing that takes the data outside $[0, 1]$ support), a dataset often considered to be a bad fit for Bernoulli VAE. For brevity we evaluated only the non-warped data $\gamma = 0$, leading to the results shown in table 4.1 (note the colored column headers, to connect to the panels in figures 4.4 to 4.6). The top section shows results for the continuous Bernoulli VAE (first row, top), the Bernoulli VAE (third row, top), and the Bernoulli VAE with a continuous Bernoulli inverse map μ^{-1} (second row, top). Across all metrics – ELBO, inception score with parameters λ, inception score with continuous Bernoulli samples given λ, and k nearest neighbors – the Bernoulli VAE is suboptimal. Interestingly, unlike in MNIST, here we see that using the continuous Bernoulli parameter correction μ^{-1} (section 4.3.5) to a Bernoulli VAE is optimal under some metrics. Again we note that this is a result belonging to the continuous Bernoulli, so even these results emphasize the importance of the continuous Bernoulli.

We then repeat the same set of experiments for Gaussian and beta distribution VAE (middle and bottom sections of table 4.1). Again, ignoring normalizing constants produces significant performance loss across all metrics. Comparing metrics across the continuous Bernoulli, Gaussian, and beta sections, we see again that the Gaussian VAE is suboptimal across all metrics, with the optimal distribution being the continuous Bernoulli or beta distribution VAE, depending on the metric.

Table 4.1: Comparisons of training with and without normalizing constants for CIFAR-10. For connection to the panels in figures 4.4 to 4.6, column headers are colored accordingly.

distribution	objective	map	$\mathscr{E}(p, \theta^*, \phi^*)$	IS w/ samples	IS w/ parameters	$\mathscr{K}(\phi^*)$
$\mathscr{CB}/\mathscr{B}$	$\mathscr{E}(p, \theta, \phi)$	$\cdot$	**1007**	1.15	2.31	**0.43**
	$\mathscr{E}(\widetilde{p}, \theta, \phi)$	μ^{-1}	916	**1.49**	**4.55**	0.42
	$\mathscr{E}(\widetilde{p}, \theta, \phi)$	$\cdot$	475	1.41	1.39	0.42
Gaussian	$\mathscr{E}(p, \theta, \phi)$	$\cdot$	**1891**	**1.86**	**3.04**	**0.42**
	$\mathscr{E}(\widetilde{p}, \theta, \phi)$	$\cdot$	-42411	1.24	1.00	0.1
beta	$\mathscr{E}(p, \theta, \phi)$	$\cdot$	**3121**	**2.98**	**4.07**	**0.47**
	$\mathscr{E}(\widetilde{p}, \theta, \phi)$	$\cdot$	-38913	1.39	1.00	0.1

4.4.4 Simulated Data

Finally, one might wonder if the performance improvements of the continuous Bernoulli VAE over the Bernoulli VAE and its corrected version with μ^{-1} are merely an artifact of not having access to the log likelihood and having to optimize the ELBO instead. In this section we show, empirically, that the answer is no. In order to do this, we consider estimating the parameters of a mixture of continuous Bernoulli distributions, of which the VAE can be thought of as a generalization with infinitely many components. We proceed as follows: We randomly set a mixture of continuous Bernoulli distributions, p_{true}, with K components in 50 dimensions (independent of each other) and sample from this mixture 10000 times in order to generate a simulated dataset. We then use the EM algorithm [Dempster et al., 1977] to estimate the mixture coefficients and the corresponding continuous Bernoulli parameters. We do this in two different ways: the first one is done correctly, i.e. using a continuous Bernoulli likelihood; and the second incorrectly, i.e. using a Bernoulli likelihood. We then measure how close the estimated parameters are from the true ones. Since doing this in a Euclidean distance

sense involves solving a hard optimization problem over permutations, we instead measure closeness with the KL divergence between the true distribution p_{true} and the estimated one, p_{est}.

The results of performing the procedure described above 10 times and averaging the KL values (over these 10 repetitions), along with standard errors, are shown in figure 4.7. First, we can see that when using the correct continuous Bernoulli likelihood, the EM algorithm correctly recovers the true distribution. We can also see that, as the number of mixture components K gets larger, ignoring the normalizing constant results in worse performance, even after correcting with μ^{-1}, which helps but does not fully remedy the situation (except at $K = 1$, as noted in section 4.3.5).

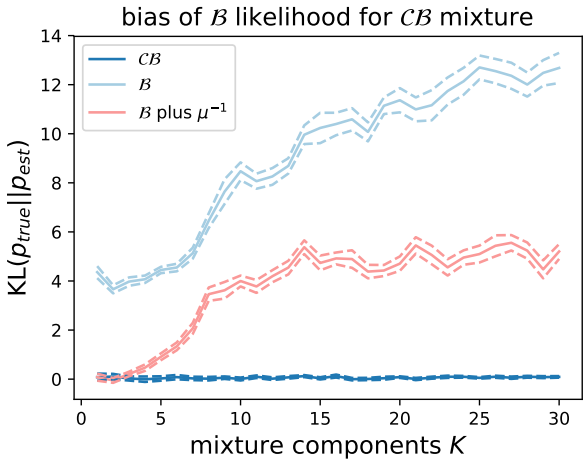


Figure 4.7: Bias of the EM algorithm to estimate continuous Bernoulli parameters when using a continuous Bernoulli likelihood (dark blue), Bernoulli likelihood (light blue) and a Bernoulli likelihood plus a μ^{-1} correction.

4.5 Conclusions

In this chapter we introduce and characterize a novel probability distribution – the continuous Bernoulli – to study the effect of using a Bernoulli VAE on $[0, 1]$-valued

intensity data, a pervasive error in highly cited papers, publicly available implementations, and core software tutorials alike. We show that this practice is equivalent to ignoring the normalizing constant of a continuous Bernoulli, and that doing so results in significant performance decrease in the qualitative appearance of samples from these models, the ELBO (approximately 300 nats), the inception score, and in terms of the latent representation (via k nearest neighbors). Several surprising findings are shown, including: *(i)* that some plausible interpretations of ignoring a normalizing constant are in fact wrong; *(ii)* the (possibly counterintuitive) fact that this normalizing constant is most critical when data is near binary; and *(iii)* that the Gaussian VAE often underperforms VAE models with the appropriate data type (continuous Bernoulli or beta distributions).

Taken together, these findings suggest an important potential role for the continuous Bernoulli distribution going forward. On this point, we note that our characterization of the continuous Bernoulli properties (such as its ease of reparameterization, likelihood evaluation, and sampling) make it compatible with the vast array of VAE improvements that have been proposed in the literature, including flexible posterior approximations [Rezende and Mohamed, 2015, Kingma et al., 2016], disentangling [Higgins et al., 2017], discrete codes [Maddison et al., 2016, Jang et al., 2016], variance control strategies [Miller et al., 2017], and more.

Chapter 5

Deep Random Splines

5.1 Introduction

Gaussian processes (GPs) [Rasmussen, 2004] are the leading class of distributions on random functions, but they suffer from well known issues including difficulty scaling and inflexibility with respect to certain shape constraints (such as nonnegativity). In this chapter we propose Deep Random Splines, a flexible class of random functions obtained by transforming Gaussian noise through a deep neural network whose output are the parameters of a spline. Unlike Gaussian processes, Deep Random Splines allow us to readily enforce shape constraints while inheriting the richness and tractability of deep generative models. We also present an observational model for point process data which uses Deep Random Splines to model the intensity function of each point process and apply it to neural population data to obtain a low-dimensional representation of spiking activity. Inference is performed via a variational autoencoder that uses a novel recurrent encoder architecture that can handle multiple point processes as input. We use a newly collected dataset where a primate completes a pedaling task, and observe better dimensionality reduction with our model than with competing alternatives.

GPs allow control of the smoothness of the function they are modeling by choosing an appropriate kernel but have the disadvantage that, except in special cases (for example

Gilboa et al. [2015], Flaxman et al. [2015]), inference in GP models scales poorly in both memory and runtime. Furthermore, GPs cannot easily handle shape constraints. It can often be of interest to model a function under some shape constraint, for example nonnegativity, monotonicity or convexity/concavity [Møller et al., 1998, Schmidt and Hess, 1988, Ramsay, 1988, Mammen, 1991]. While some shape constraints can be enforced by transforming the GP or by enforcing them at a finite number of points, doing so cannot always be done and usually makes inference harder, see for example Lin and Dunson [2014].

Splines are another popular tool for modeling unknown functions (see section 2.8). When there are no shape constraints, frequentist inference is straightforward and can be performed using linear regression, by writing the spline as a linear combination of basis functions. Under shape constraints, the basis function expansion usually no longer applies, since the space of shape constrained splines is not typically a vector space. However, the problem can usually still be written down as a tractable constrained optimization problem [Schmidt and Hess, 1988]. Furthermore, when using splines to model a random function, a distribution must be placed on the spline's parameters, so the inference problem becomes Bayesian. DiMatteo et al. [2001] proposed a method to perform Bayesian inference in a setting without shape constraints, but the method relies on the basis function expansion and cannot be used in a shape constrained setting. Furthermore, fairly simple distributions have to be placed on the spline parameters for their approximate posterior sampling algorithm to work adequately, which results in the splines having a restrictive and oversimplified distribution.

On the other hand, deep probabilistic models take advantage of the major progress in neural networks to fit rich, complex distributions to data in a tractable way [Rezende et al., 2014, Mohamed and Lakshminarayanan, 2016, Kingma and Welling, 2013, Gao et al., 2016, Johnson et al., 2016]. However, their goal is not usually to model random functions.

In this chapter, whose content [Loaiza-Ganem et al., 2019] was presented at Cosyne

2019 and at an ICLR 2019 workshop, we introduce Deep Random Splines (DRS), an alternative to GPs for modeling random functions. DRS are a deep probabilistic model in which standard Gaussian noise is transformed through a neural network to obtain the parameters of a spline, and the random function is then the corresponding spline. This combines the complexity of deep generative models and the ability to enforce shape constraints of splines.

We use DRS to model the nonnegative intensity functions of Poisson processes (see section 2.7). In order to ensure that the splines are nonnegative, we use a parameterization of nonnegative splines that can be written as an intersection of convex sets, and then use the method of alternating projections (see section 2.9) to obtain a point in that intersection (and differentiate through that during learning). To perform scalable inference, we use a variational autoencoder (see section 2.5) with a novel encoder architecture that takes multiple, truly continuous point processes as input (not discretized in bins, as is common).

Our contributions are: (i) Introducing DRS, (ii) using the method of alternating projections to constrain splines, (iii) proposing a variational autoencoder model whith a novel encoder architecture for point process data which uses DRS, and (iv) showing that our model outperforms commonly used alternatives in both simulated and real data.

5.2 Deep Random Splines

Throughout the chapter we will consider splines on the interval $[T_1, T_2)$ and will select $I+1$ fixed knots $T_1 = t_0 < \cdots < t_I = T_2$. We will denote the set of splines of degree d and smoothness s by $\mathscr{G}_{d,s} = \{g_\psi : \psi \in \Psi_{d,s}\}$, where $\Psi_{d,s}$ is the set of parameters of each polynomial in each interval. That is, every $\psi \in \Psi_{d,s}$ contains the parameters of each of the I polynomial pieces (it does not contain the locations of the knots as we take them to be fixed since we observed overfitting when not doing so). While the most natural

ways to parameterize splines of degree d are a linear combination of basis functions or with the $d+1$ polynomial coefficients of each interval, these parameterizations do not lend themselves to easily enforce constraints such as nonnegativity [Schmidt and Hess, 1988]. We will thus use a parameterization based on equation 2.39. We will denote by $\Psi \subseteq \Psi_{d,s}$ the subset of spline parameters that result in the splines having the shape constraint of interest, for example, nonnegativity.

DRS are a distribution over $\mathscr{G}_{d,s}$. To sample from a DRS, a standard Gaussian random variable $Z \in \mathbb{R}^M$ is transformed through a neural network parameterized by θ, $f_\theta : \mathbb{R}^M \to \Psi$. The DRS is then given by $g_{f_\theta(Z)}$ and inference on θ can be performed through a variational autoencoder [Kingma and Welling, 2013]. Note that f maps to Ψ, thus ensuring that the spline has the relevant shape constraint.

5.2.1 Parameterizing Nonnegative Splines

We now explain how we can parameterize nonnegative splines. We add the nonnegativity constraint to the spline as we will use it for our model in section 5.3, but constraints such as monotonicity and convexity/concavity can be enforced in an analogous way. In order to achieve this, we use a parameterization of nonnegative splines that might seem overly complicated at first. However, it has the critical advantage that it decomposes into the intersection of convex sets that are easily characterized in terms of the parameters, which is not the case for the naive parameterization which only includes the $d+1$ coefficients of every polynomial. We will see how to take advantage of this fact in the next section.

Recall, from equation 2.39, that a polynomial $p(t)$ of degree d, where $d = 2k+1$ for some $k \in \mathbb{N}$, is nonnegative in the interval $[l, u)$ if and only if it can be written down as follows:

$$p(t) = (u-t)[t]^\top Q_1[t] + (t-l)[t]^\top Q_2[t] \tag{5.1}$$

where $[t] = (1, t, t^2, \ldots, t^k)^\top$ and Q_1 and Q_2 are $(k+1) \times (k+1)$ symmetric positive semidefinite matrices. It follows that a piecewise polynomial of degree d with knots

$t_0, \ldots, t_I$ defined as $p^{(i)}(t)$ for $t \in [t_{i-1}, t_i)$ for $i = 1, \ldots, I$ is nonnegative if and only if it can be written as:

$$p^{(i)}(t) = (t_i - t)[t]^\top Q_1^{(i)}[t] + (t - t_{i-1})[t]^\top Q_2^{(i)}[t] \tag{5.2}$$

for $i = 1, \ldots, I$, where each $Q_1^{(i)}$ and $Q_2^{(i)}$ are $(k+1) \times (k+1)$ symmetric positive semidefinite matrices. We can thus parameterize every piecewise nonnegative polynomial on our I intervals with $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$. If no constraints are added on these parameters, the resulting piecewise polynomial might not be smooth, so certain constraints have to be added in order to guarantee that we are parameterizing a nonnegative spline and not just a nonnegative piecewise polynomial. To that end, we define $\mathscr{C}_1$ as the set of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ such that:

$$p^{(i)}(t_i) = p^{(i+1)}(t_i) \text{ for } i = 1, \ldots, I-1 \tag{5.3}$$

That is, $\mathscr{C}_1$ is the set of parameters whose resulting piecewise polynomial as in equation 5.2 is continuous. Analogously, let $\mathscr{C}_j$ for $j = 2, 3, \ldots$ be the set of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ such that:

$$\frac{\partial^{j-1}}{\partial t^{j-1}} p^{(i)}(t_i) = \frac{\partial^{j-1}}{\partial t^{j-1}} p^{(i+1)}(t_i) \text{ for } i = 1, \ldots, I-1 \tag{5.4}$$

So that $\mathscr{C}_j$ is the set of parameters whose corresponding piecewise polynomials have matching left and right $(j-1)$-th derivatives. Let $\mathscr{C}_0$ be the set of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ which are symmetric positive semidefinite. We can then parameterize the set of nonnegative splines on $[T_1, T_2)$ by $\Psi = \cap_{j=0}^{s+1} \mathscr{C}_j$. Note that the case where d is even can be treated analogously (see equation 2.40).

5.2.2 Enforcing Nonnegativity

In order to use a DRS, f_θ has to map to Ψ, that is, we need to have a way for a neural network to map to the parameter set corresponding to nonnegative splines. We achieve this by taking $f_\theta(z) = h(\widetilde{f}_\theta(z))$, where $\widetilde{f}_\theta$ is an arbitrary neural network and h is a surjective function onto Ψ. The most natural choice for h is the projection onto

Ψ. However, while computing the projection onto Ψ (for Ψ as in section 5.2.1) can be done by solving the following convex optimization problem:

$$h\Big((Q_1^{(i)}, Q_2^{(i)})_{i=1}^I\Big) = \underset{(X^{(i)},Y^{(i)})_{i=1}^I}{\text{minimize}} \sum_{i=1}^{I} ||X^{(i)} - Q_1^{(i)}||_F^2 + ||Y^{(i)} - Q_2^{(i)}||_F^2 \qquad (5.5)$$
$$\text{subject to } (X^{(i)}, Y^{(i)})_{i=1}^I \in \cap_{j=0}^{s+1} \mathscr{C}_j$$

where $||\cdot||_F$ denotes the Frobenius norm, it cannot be done analytically. This is an issue because when we train the model, we will need to differentiate f_θ with respect to θ. Note that Amos and Kolter [2017] propose a method to have an optimization problem as a layer in a neural network. One might hope to use their method for our problem, but it cannot be applied due to the semidefinite constraint on our matrices. Thus, we apply the method of alternating projections. In our case, projecting onto $\mathscr{C}_0$ can be done by doing eigenvalue decompositions of $Q_1^{(i)}$ and $Q_2^{(i)}$ and zeroing out negative elements in the diagonal matrices containing the eigenvalues. While this might seem computationally expensive, the matrices are small and this can be done efficiently. For example, for cubic splines ($d = 3$), there are $2I$ matrices each one of size 2×2. Projecting onto $\mathscr{C}_j$ for $j = 1, \dots s+1$ can be done analytically as it can be formulated as a quadratic optimization problem with linear constraints. Furthermore, because of the local nature of the constraints where every interval is only constrained by its neighboring intervals, this quadratic optimization problem can be reduced to solving a tridiagonal system of linear equations of size $I - 1$ which can be solved efficiently in $O(I)$ time with simplified Gaussian elimination. While the derivation of this fact is a straightforward application of the KKT conditions, the algebra is cumbersome, so we omit it here to include it in appendix C.

By letting h be a finite number of iterations of the method of alternating projections, we can ensure that f_θ maps (approximately) to Ψ, while still being able to compute $\nabla_\theta f_\theta(z)$. Note that we could find such an h function using Dykstra's algorithm (not to be confused with Dijkstra's shortest path algorithm), which is a modification of the method of alternating projections that converges to the projection of $\psi^{(0)}$ onto $\cap_{j=0}^{s+1} \mathscr{C}_j$

[Dykstra, 1983, Boyle and Dykstra, 1986, Tibshirani, 2017]), but we found that the method of alternating projections was faster to differentiate when using reverse mode automatic differentiation packages [Abadi et al., 2016].

Another way of finding such an h would be unrolling any iterative optimization method that solves the projection onto Ψ, such as gradient-based methods or Newton methods. We found the alternating projections method more convenient as it does not involve additional hyperparameters such as learning rate that drastically affect performance. Furthermore, the method of alternating projections is known to have a linear convergence rate (as fast as gradient-based methods) that is independent of the starting point [Bauschke and Borwein, 1996]. This last observation is important, as the starting point in our case is determined by the output of $\widetilde{f}_\theta$, so that the convergence rate being independent of the starting point ensures that $\widetilde{f}_\theta$ cannot learn to ignore h, which is not the case for gradient-based and Newton methods (for a fixed number of iterations and learning rate, there might exist an initial point that is too far away to actually reach the projection). Finally, note that if we wanted to enforce, for example, that the spline be monotonic, we could parameterize its derivative and force it to be nonnegative or nonpositive. Convexity or concavity can be enforced analogously.

5.3 Deep Random Splines as Intensity Functions of Point Processes

Splines have the key property that they can be analytically integrated (as the integral of polynomials can be computed in closed form), which allows to exactly evaluate the log likelihood in equation 2.38 when g is a spline. This was one of our main motivations for using splines to model intensity functions of point processes, as the double intractability issue of log-Gaussian Cox processes [Møller et al., 1998] is thus avoided. As a consequence, fitting a DRS to observed events is more tractable than fitting models that use GPs to represent g. Splines also vary smoothly, which

incorporates the reasonable assumption that the expected number of events changes smoothly over time, which is another important property that motivated our use of splines.

5.3.1 Our Model

Suppose we observe N simultaneous point processes in $[T_1, T_2)$ a total of R repetitions (we will call each one of these repetitions/samples a trial). Let $X_{r,n}$ denote the n-th point process of the r-th trial. Looking ahead to an application we study in the results, data of this type is a standard setup for microelectrode array data, where N neurons are measured from time T_1 to time T_2 for R repetitions, and each event in the point processes corresponds to a spike (the time at which the neurons "fired"). Each $X_{r,n}$ is also called a spike train. The model we propose, which we call DRS-VAE, is as follows:

$$\begin{cases} Z_r \sim \mathcal{N}(0, I_M) \text{ for } r = 1, \dots, R \\ \psi_{r,n} = f_\theta^{(n)}(Z_r) \text{ for } n = 1, \dots, N \\ X_{r,n} | \psi_{r,n} \sim \mathscr{P}\mathscr{P}_{[T_1,T_2)}(g_{\psi_{r,n}}) \end{cases} \tag{5.6}$$

where each $f_\theta^{(n)} : \mathbb{R}^M \to \Psi$ is obtained as described in section 5.2.2. The hidden state Z_r for the r-th trial $\mathbf{X}_r := (X_{r,1}, \dots, X_{r,N})$ can be thought as a low-dimensional representation of $\mathbf{X}_r$. Note that while the intensity function of every point process and every trial is a DRS, the latent state Z_r of each trial is shared among the N point processes. Note also that the data we are modeling can be thought of as R marked point processes [Kingman, 1992], where the mark of the event $x_{r,n,k}$ (the k-th event of the n-th point process of the r-th trial) is n. In this setting, $g_{\psi_{r,n}}$ corresponds to the conditional (on Z_r and on the mark being n) intensity of the process for the r-th trial. Once again, one might think that our parameterization of nonnegative splines is unnecessarily complicated and that having $f_\theta^{(n)}$ in equation 5.6 be a simpler parameterization of an arbitrary spline (e.g. basis coefficients) and using $\tau(g_{\psi_{r,n}})$ instead of $g_{\psi_{r,n}}$, where

τ is a nonnegative function, might be a better solution to enforcing nonnegativity constraints. The function τ would have to be chosen in such a way that the integral of equation 2.38 can still be computed analytically, making $\tau(t) = t^2$ a natural choice. While this would avoid having to use the method of alternating projections, we found that squared splines perform very poorly as they oscillate too much.

5.3.2 Inference

In order to perform inference in our model, we use autoencoding variational Bayes. Because of the point process nature of the data, m_ϕ and s_ϕ from equation 2.24 require a recurrent architecture, since their input $\mathbf{x}_r = (x_{r,1}, x_{r,2}, \ldots, x_{r,N})$ consists of N point processes. This is challenging because the input is not just a sequence, but N sequences of different lengths (numbers of events). In order to deal with this, we use N separate LSTMs [Hochreiter and Schmidhuber, 1997], one per point process. Each LSTM takes as input the events of the corresponding point process. The final states of each LSTM are then concatenated and transformed through a dense layer (followed by an exponential activation in the case of s_ϕ to ensure positivity) in order to map to the hidden space $\mathbb{R}^M$. We also tried bidirectional LSTMs [Graves and Schmidhuber, 2005] but found regular LSTMs to be faster while having similar performance. The architecture is depicted in figure 5.1. The ELBO for our model is then given by:

$$\mathscr{E}(\theta, \phi) = \sum_{r=1}^{R} -KL(q_\phi(z_r|x_r)||\pi) + E_{q_\phi(z_r|x_r)}\Big[\sum_{n=1}^{N}\sum_{k=1}^{K_{r,n}} \log g_{\psi_{r,n}}(x_{r,n,k}) - \int_{\mathscr{S}} g_{\psi_{r,n}}(t)dt\Big] \tag{5.7}$$

where $K_{r,n}$ is the number of events in the n-th point process of the r-th trial. Gao et al. [2016] have a similar model, where a hidden Markov model is transformed through a neural network to obtain event counts on time bins. The hidden state for a trial in their model is then an entire hidden Markov chain, which will have significantly higher dimension than our hidden state. Also, their model can be recovered from ours if we change the standard Gaussian distribution of Z_r in equation 5.6 to reflect their Markovian structure and choose $\mathscr{G}$ to be piecewise constant, nonnegative functions. We also emphasize the fact that our model is very easy to extend: for example, it would be straightforward to extend it to multi-dimensional point processes (not neural

data any more) by changing $\mathscr{G}$ and its parameterization. It is also straightforward to use a more complicated point process than the Poisson one by allowing the intensity to depend on previous event history. Furthermore, DRS can be used in settings that require random functions, even if no point process is involved.

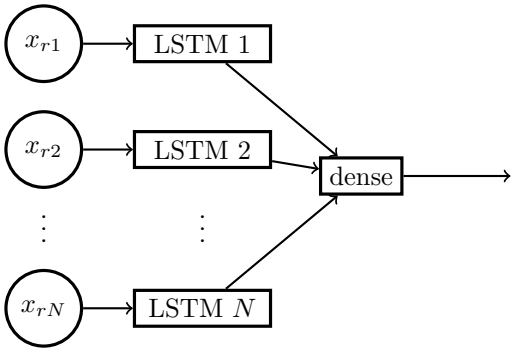


Figure 5.1: Encoder architecture.

5.4 Experiments

5.4.1 Simulated Data

We simulated data with the following procedure: First, we set 2 different types of trials. For each type of trial, we sampled one true intensity function on $[0, 10)$ for each of the $N = 2$ point processes by sampling from a GP and exponentiating the result. We then sampled 600 times from each type of trial, resulting in 1200 trials. We randomly selected 1000 trials for training and set aside the rest for testing. We then fit the model described in section 5.3.1 an compare against other methods that perform intensity estimation while recovering a low-dimensional representation of trial: the PP-GPFA model [Duncker and Sahani, 2018], the PfLDS model [Gao et al., 2016] and the GPFA model [Yu et al., 2009]. The two latter models discretize time into B time bins and have a latent variable per time bin and per trial (as opposed to our model which is only per trial), while the former recovers continuous latent trajectories.

They do this as a way of enforcing temporal smoothness by placing an appropriate prior over their latent trajectories, which we do not have to do as we implicitly enforce temporal smoothness by using splines to model intensity functions.

We used a uniform grid with 11 knots (resulting in $I = 10$ intervals), $d = 3$ and $s = 2$. Since a twice-differentiable cubic spline on I intervals has $I + 3$ degrees of freedom, when discretizing time for PfLDS and GPFA we use $B = I + 3 = 13$ time bins. This way the distribution recovered by PfLDS also has $B = 13$ degrees of freedom, while the distribution recovered by GPFA has even more. We set the latent dimension M in our model to 2 and we also set the latent dimension per time bin in PfLDS and GPFA to 2, meaning that the overall latent dimension for an entire trial was $2B = 26$. These two choices make the comparison conservative as they allow more flexibility for the two competing methods than for ours. For PP-GPFA we set the continuous latent trajectory to have dimension 2. Our architecture and hyperparameter choices are included in appendix C.

The left panel of figure 5.2 shows the posterior means of the hidden variables in our model for each of the 200 test trials. Each posterior mean is colored according to its type of trial. We can see that different types of trials form separate clusters, meaning that our model successfully obtains low-dimensional representations of the trials. Note that the model is trained without having access to the type of each trial; colors are assigned in the figure post hoc. The middle panel shows the events (in black) for a particular point process on a particular trial, along with the true intensity (in green) that generated the events and posterior samples from our model (in purple), PP-GPFA (in orange), PfLDS (in blue), and GPFA (in red) of the corresponding intensities. Note that since PfLDS and GPFA parameterize the number of counts on each time bin, they do not have a corresponding intensity. We plot instead a piecewise constant intensity on each time bin in such a way that the expected number of events in each time bin is equal to the integral of the intensity. We can see that our method recovers a smooth function that is closer to the truth than the ones recovered with competing methods.

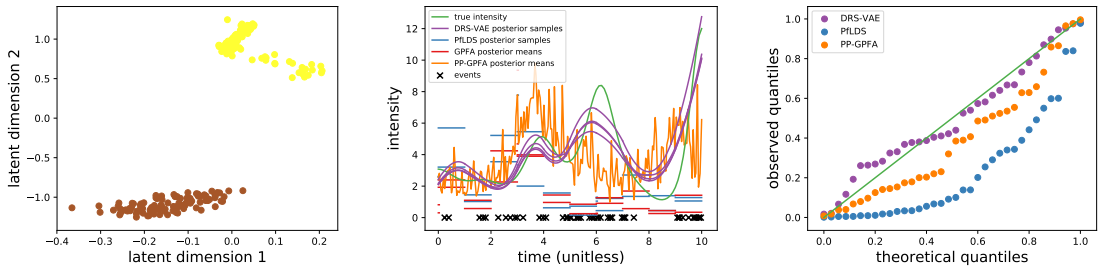


Figure 5.2: Posterior means of the hidden variables of DRS-VAE by type of trial on simulated data (*left panel*), comparison of posterior intensities of our method (DRS-VAE) against competing alternatives on simulated data (*middle panel*) and Q-Q plot of events (*right panel*).

The right panel of the figure shows a Q-Q plot (with time rescaled as in Brown et al. [2002]) and we can see, once again, that our method recovers better intensities. Table 5.1 shows performance from our model compared against PP-GPFA, PfLDS and GPFA. The second column shows the per-trial ELBO on test data, and we can see that our model has a larger ELBO than the alternatives. While having a better ELBO does not imply that our log likelihood is better, it does suggest that it is. Since both PfLDS and GPFA put a distribution on event counts on time bins instead of a distribution on event times as our models does, the log likelihoods are not directly comparable.

Table 5.1: Quantitative comparison of our method (DRS-VAE) against competing alternatives on simulated data.

METHOD	**ELBO**	$\mathbf{L^2}$	**p-VALUE**
DRS-VAE	**57.1**	$\mathbf{0.11 \pm 0.09}$	–
PfLDS	52.3	0.21 ± 0.10	$< 10^{-44}$
GPFA	–	0.21 ± 0.10	$< 10^{-45}$
PP-GPFA	29.0	0.38 ± 0.24	$< 10^{-70}$

However, in the case of PfLDS, we can easily convert from the Poisson likelihood on time bins to the piecewise constant intensity Poisson process likelihood, so that the numbers become comparable. In order to get a quantitative comparison between our model and GPFA, we take advantage of the fact that we know the true intensity that generated the data and compare average L^2 distance, across point processes and trials, between posterior intensity samples and actual intensity function. Once again, we can see that our method outperforms the alternatives. Table 5.1 also includes the standard deviation of these L^2 distances. Since the standard deviations are somewhat large in comparison to the means, for each of the two competing alternatives, we carry out a two sample t-test comparing the L^2 distance means obtained with our method against the alternative. The p-values indicate that our method recovers intensity functions that are closer to the truth in a statistically significant way.

5.4.2 Real Data

5.4.2.1 Reaching Data

We also fit our model to the dataset collected by Churchland et al. [2012]. The dataset, after preprocessing (see appendix C for details), consists of measurements of 20 neurons for 3590 trials on the interval $[-100, 300)$ (in ms) of a primate. In each trial, the primate reaches with its arm to a specific location, which changes from trial to trial (we can think of the 40 locations as types of trials), where time 0 corresponds to the beginning of the movement. We randomly split the data into a training set with 3000 trials and a test set with the rest of the trials.

We used twice-differentiable cubic splines and 18 uniformly spaced knots (that is, 17 intervals). For the comparison against PfLDS, we split time into 20 bins, resulting in time bins of $20ms$ (which is a standard length), once again making sure that the degrees of freedom are comparable. Further architectural details are included in appendix C. Since we do not have access to the ground truth, we do not compare

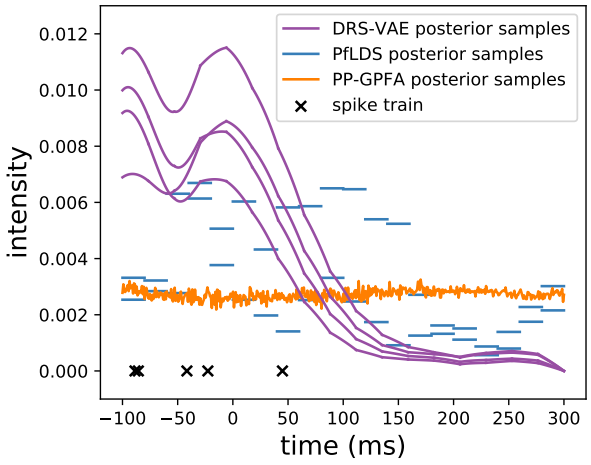


Figure 5.3: Comparison of posterior intensities of our method (DRS-VAE) against competing alternatives on reaching data.

against GPFA as the L^2 metric computed in the previous section cannot be used here. Again, we used a hidden dimension $M = 2$ for our model, resulting in hidden trajectories of dimension 40 for PfLDS, and continuous trajectories of dimension 2 for PP-GPFA. We experimented with larger values of M but did not observe significant improvements in either model.

Figure 5.3 shows the spike train (black) for a particular neuron on a particular trial, along with posterior samples from our model (in purple), PP-GPFA (in orange) and PfLDS (in blue) of the corresponding intensities. We can see that the posterior samples from our method look more plausible and smoother than the other ones.

Table 5.2 shows the per-trial ELBO on test data for our model and for the competing alternatives. Again, our model has a larger ELBO, even when PfLDS has access to 20 times more hidden dimensions: our method is more successful at producing low-dimensional representations of trials than PfLDS. The table also shows the percentage of correctly predicted test trial types when using 15-nearest neighbors on the posterior means of train data (the entire trajectories are used for PfLDS and 20 uniformly spaced points along each dimension of the continuous trajectories of PP-GPFA, resulting in

40 dimensional latent representations). While 23.7% might seem small, it should be noted that it is significantly better than random guessing (which would have 2.5% accuracy) and that the model was not trained to minimize this objective. Regardless, we can see that our method outperforms both PP-GPFA and PfLDS in this metric, even when using a much lower-dimensional representation of each trial. The table also includes the percentage of explained variation when doing ANOVA on the test posterior means (denoted SSG/SST), using trial type as groups. Once again, we can see that our model recovers a more meaningful representation of the trials.

5.4.2.2 Cycling Data

We also fit our model to our newly collected dataset. After preprocessing (see appendix C), it consists of 1300 and 188 train and test trials, respectively. During each trial, 20 neurons were recorded as the primate turns a hand-held pedal to navigate through a virtual environment. There are 8 trial types, based on whether the primate is pedaling forward or backward and over what distance.

We use the same hyperparameter settings as for the reaching data, except we use 26 uniformly spaced knots (25 intervals) and 28 bins for PfLDS, as well as a hidden dimension $M = 10$, resulting in hidden trajectories of dimension 280 for PfLDS (analogously, we set PP-GPFA to have 10 dimensional continuous trajectories, and take 28 uniformly spaced points along each dimension to obtain 280 dimensional latent

Table 5.2: Quantitative comparison of our method (DRS-VAE) against competing alternatives on reaching data.

METHOD	**ELBO**	**15-NN**	**SSG/SST**
DRS-VAE	$\mathbf{-500.8}$	**23.7**%	**73.9**%
PfLDS	-505.7	3.1%	6.2%
PP-GPFA	-523.2	14.1%	30.5%

representations). Results are also summarized in table 5.3. We can see that while our ELBO is higher than for PP-GPFA, it is actually lower than for PfLDS, which we believe is caused by an artifact of preprocessing the data rather than any essential performance loss.

While the ELBO was better for PfLDS, the quality of our latent representations is significantly better, as shown by the accuracy of 15-nearest neighbors to predict test trial types (random guessing would have 12.5% accuracy) and the ANOVA percentage of explained variation of the test posterior means, which are also better than for PP-GPFA. This is particularly impressive as our latent representations have 28 times fewer dimensions. We did experiment with different hyperparameter settings, and found that the ELBO of PfLDS increased slightly when using more time bins (at the cost of even higher-dimensional latent representations), whereas our ELBO remained the same when increasing the number of intervals. However, even in this setting the accuracy of 15-nearest neighbors and the percentage of explained variation did not improve for PfLDS.

5.5 Conclusions

In this chapter we introduced Deep Random Splines, an alternative to Gaussian processes to model random functions. Owing to our key modeling choices and use of

Table 5.3: Quantitative comparison of our method (DRS-VAE) against competing alternatives on cycling data.

METHOD	ELBO	15-NN	SSG/SST
DRS-VAE	6372	**55.9**%	**70.0**%
PfLDS	**6532**	11.7%	3.2%
PP-GPFA	6079	51.1%	14.6%

results from the spline and optimization literatures, fitting DRS is tractable and allows one to enforce shape constraints on the random functions. While we only enforced nonnegativity and smoothness here, it is straightforward to enforce constraints such as monotonicity (or convexity/concavity). We also proposed a variational autoencoder that takes advantage of DRS to accurately model and produce meaningful low-dimensional representations of neural activity.

Future work includes using DRS-VAE for multi-dimensional point processes, for example spatial point processes. While splines would become harder to use in such a setting, they could be replaced by any family of easily-integrable nonnegative functions, such as, for example, conic combinations of Gaussian kernels. Another line of future work involves using a more complicated point process than the Poisson, for example a Hawkes process, by allowing the parameters of the spline in a certain interval to depend on the previous spiking history of previous intervals. Finally, DRS can be applied in more general settings than the one explored in this chapter since they can be used in any setting where a random function is involved, having many potential applications beyond what we analyzed here.

Chapter 6

Conclusions

In this dissertation we introduce three advances to the deep generative modeling area: Maximum Entropy Flow Networks, a method for maximum entropy modeling; a new $[0, 1]$-supported distribution, the continuous Bernoulli, which we develop to fix a pervasive error in variational autoencoders; and Deep Random Splines, a novel distribution over functions which allows to enforce shape constraints in a more tractable way than Gaussian processes. We apply these to model texture images, natural images and neural population data, respectively, and observe significant improvements over previous state of the art methods.

Maximum Entropy Flow Networks work by combining normalizing flows with the augmented Lagrangian optimization method. Normalizing flows enable tractable entropy estimation, and the augmented Lagrangian method allows to efficiently optimize the resulting objective in a more numerically stable way than a naive penalty method. Applying our method to generate texture images results in significantly more diverse images than the previous state of the art, while maintaining comparable image quality. This opens up interesting avenues for future research in areas where maximum entropy modeling can be applied, such as natural language processing or ecology.

We also introduced the continuous Bernoulli distribution to fix a pervasive and often set aside as inconsequential error in Bernoulli variational autoencoders, namely using

a Bernoulli likelihood with non-binary data. Here, we showed that this innocuous looking error can be interpreted as ignoring a normalizing constant during training, which results in a loss of interpretability and a decrease in performance over several important metrics. We completely characterize this new distribution, which can be used to model $[0, 1]$-valued data, particularly when it is close to the boundaries.

Finally, we also introduced Deep Random Splines, a novel distribution over functions. To obtain a sample from this distribution, Gaussian noise is mapped through a neural network to obtain the parameters of a spline. If enforcing shape constraints is needed, a finite number of steps of the method of alternating projections are applied to the output of the network in order to enforce them. Backpropagation can be carried out through this procedure in order to learn the parameters of the neural network. In particular, we used Deep Random Splines in a variational autoencoder to model neural population data. Our model outperforms competing alternatives and recovers significantly lower-dimensional representations of trials. This opens up exciting avenues for future research, such as further improving our neural population model, modeling spatial or spatio-temporal point processes, or using Deep Random Splines for a completely different application, such as shape-constrained regression.

We hope that the contributions presented here will be useful to deep generative modeling, computational neuroscience and the broader machine learning community, and that they will inspire further research.

Bibliography

Pytorch VAE turotial: https://github.com/pytorch/examples/tree/master/vae, Keras VAE turotial: https://blog.keras.io/building-autoencoders-in-keras.html.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Ryan Prescott Adams, Iain Murray, and David JC MacKay. Tractable nonparametric bayesian inference in poisson processes with gaussian process intensities. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 9–16. ACM, 2009.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

Sanjeev Arora and Yi Zhang. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224*, 2017.

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). In *Proceedings of the 34th*

International Conference on Machine Learning-Volume 70, pages 224–232. JMLR. org, 2017.

Marco Ballini, Jan Müller, Paolo Livi, Yihui Chen, Urs Frey, Alexander Stettler, Amir Shadmani, Vijay Viswam, Ian Lloyd Jones, and David Jäckel. A 1024-channel cmos microelectrode array with 26,400 electrodes for recording and stimulation of electrogenic cells in vitro. *IEEE journal of solid-state circuits*, 49(11):2705–2719, 2014.

Heinz H Bauschke and Jonathan M Borwein. On projection algorithms for solving convex feasibility problems. *SIAM review*, 38(3):367–426, 1996.

Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1): 39–71, 1996.

Thomas B Berrett, Richard J Samworth, and Ming Yuan. Efficient multivariate entropy estimation via k-nearest neighbour distances. *arXiv preprint arXiv:1606.00304*, 2016.

Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Oleg Bondarenko. Estimation of risk-neutral densities using positive convolution approximation. *Journal of Econometrics*, 116(1):85–112, 2003.

Jonathan Borwein, Rustum Choksi, and Pierre Maréchal. Probability distributions of assets inferred from option prices via the principle of maximum entropy. *SIAM Journal on Optimization*, 14(2):464–478, 2003.

James P Boyle and Richard L Dykstra. A method for finding projections onto the intersection of convex sets in hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer, 1986.

Emery N Brown, Riccardo Barbieri, Valérie Ventura, Robert E Kass, and Loren M Frank. The time-rescaling theorem and its application to neural spike train data analysis. *Neural computation*, 14(2):325–346, 2002.

Peter W Buchen and Michael Kelly. The maximum entropy distribution of an asset inferred from option prices. *Journal of Financial and Quantitative Analysis*, 31(01): 143–159, 1996.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

François Chollet et al. Keras. https://keras.io, 2015.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

Mark M Churchland, John P Cunningham, Matthew T Kaufman, Justin D Foster, Paul Nuyujukian, Stephen I Ryu, and Krishna V Shenoy. Neural population dynamics during reaching. *Nature*, 487(7405):51, 2012.

Michael Collins, Robert E Schapire, and Yoram Singer. Logistic regression, adaboost and bregman distances. *Machine Learning*, 48(1-3):253–285, 2002.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*, 2018.

John P Cunningham, Krishna V Shenoy, and Maneesh Sahani. Fast gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pages 192–199. ACM, 2008.

John N Darroch and Douglas Ratcliff. Generalized iterative scaling for log-linear models. *The annals of mathematical statistics*, pages 1470–1480, 1972.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE transactions on pattern analysis and machine intelligence*, 19 (4):380–393, 1997.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

Ilaria DiMatteo, Christopher R Genovese, and Robert E Kass. Bayesian curve-fitting with free-knot splines. *Biometrika*, 88(4):1055–1071, 2001.

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

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

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Miroslav Dudik, Steven J Phillips, and Robert E Schapire. Performance guarantees for regularized maximum entropy density estimation. In *International Conference on Computational Learning Theory*, pages 472–486. Springer, 2004.

Lea Duncker and Maneesh Sahani. Temporal alignment and latent gaussian process factor inference in population spike trains. In *Advances in Neural Information Processing Systems*, pages 10445–10455, 2018.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. *arXiv preprint arXiv:1906.02145*, 2019a.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019b.

Richard L Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983.

Stephen Figlewski. Estimating the implied risk neutral density. 2008.

Seth Flaxman, Andrew Wilson, Daniel Neill, Hannes Nickisch, and Alex Smola. Fast kronecker inference in gaussian processes with non-gaussian likelihoods. In *International Conference on Machine Learning*, pages 607–616, 2015.

Yuanjun Gao, Evan W Archer, Liam Paninski, and John P Cunningham. Linear dynamical neural population models through nonlinear embeddings. In *Advances in neural information processing systems*, pages 163–171, 2016.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015.

Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Elad Gilboa, Yunus Saatçi, and John P Cunningham. Scaling multidimensional inference for structured gaussian processes. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):424–436, 2015.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6): 602–610, 2005.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, volume 3, 2017.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106 (4):620, 1957.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

Jiantao Jiao, Kartik Venkat, Yanjun Han, and Tsachy Weissman. Minimax estimation of functionals of discrete distributions. *IEEE Transactions on Information Theory*, 61(5):2835–2885, 2015.

Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2): 183–233, 1999.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances In Neural Information Processing Systems*, pages 586–594, 2016.

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

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

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.

John Frank Charles Kingman. *Poisson processes*, volume 3. Clarendon Press, 1992.

Jack PC Kleijnen and Reuven Y Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. Generalized variational inference. *arXiv preprint arXiv:1904.02063*, 2019.

Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

Jean-Bernard Lasserre. *Moments, positive polynomials and their applications*, volume 1. World Scientific, 2010.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Jin Hyung Lee, David E Carlson, Hooshmand Shokri Razaghi, Weichi Yao, Georges A Goetz, Espen Hagen, Eleanor Batty, EJ Chichilnisky, Gaute T Einevoll, and Liam Paninski. Yass: Yet another spike sorter. In *Advances in neural information processing systems*, pages 4002–4012, 2017.

Lizhen Lin and David B Dunson. Bayesian monotone regression using gaussian process projection. *Biometrika*, 101(2):303–317, 2014.

Chris Lloyd, Tom Gunter, Michael Osborne, and Stephen Roberts. Variational inference for gaussian process modulated poisson processes. In *International Conference on Machine Learning*, pages 1814–1822, 2015.

Gabriel Loaiza-Ganem and John P Cunningham. The continuous bernoulli: fixing a pervasive error in variational autoencoders. *arXiv preprint arXiv:1907.06845*, 2019.

Gabriel Loaiza-Ganem, Yuanjun Gao, and John P Cunningham. Maximum entropy flow networks. *International Conference on Learning Representations*, 2017.

Gabriel Loaiza-Ganem, Sean M Perkins, Karen E Schroeder, Mark M Churchland, and John P Cunningham. Deep random splines for point process intensity estimation of neural population data. *arXiv preprint arXiv:1903.02610*, 2019.

Yang Lu, Song-chun Zhu, and Ying Nian Wu. Learning frame models using cnn filters. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

Enno Mammen. Estimating a smooth monotone regression function. *The Annals of Statistics*, pages 724–740, 1991.

Andrew Miller, Nick Foti, Alexander D'Amour, and Ryan P Adams. Reducing reparameterization gradient variance. In *Advances in Neural Information Processing Systems*, pages 3708–3718, 2017.

Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

Shakir Mohamed and Balaji Lakshminarayanan. Learning in implicit generative models. *arXiv preprint arXiv:1610.03483*, 2016.

Jesper Møller, Anne Randi Syversveen, and Rasmus Plenge Waagepetersen. Log gaussian cox processes. *Scandinavian journal of statistics*, 25(3):451–482, 1998.

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Judea Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

Steven J Phillips, Robert P Anderson, and Robert E Schapire. Maximum entropy modeling of species geographic distributions. *Ecological modelling*, 190(3):231–259, 2006.

Ben Poole, Subhaneil Lahiri, Maithreyi Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances In Neural Information Processing Systems*, pages 3360–3368, 2016.

Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein.

On the expressive power of deep neural networks. *arXiv preprint arXiv:1606.05336*, 2016.

James O Ramsay. Monotone regression splines in action. *Statistical science*, pages 425–441, 1988.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Ruslan Salakhutdinov, Sam Roweis, and Zoubin Ghahramani. On the convergence of bound optimization algorithms. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 509–516. Morgan Kaufmann Publishers Inc., 2002.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

Jochen W Schmidt and Walter Hess. Positivity of cubic polynomials on intervals and positive spline interpolation. *BIT Numerical Mathematics*, 28(2):340–352, 1988.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ryan J Tibshirani. Dykstra's algorithm, admm, and coordinate descent: Connections, insights, and extensions. In *Advances in Neural Information Processing Systems*, pages 517–528, 2017.

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *arXiv preprint arXiv:1603.03417*, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. *arXiv preprint arXiv:1701.02096*, 2017.

Paul Valiant and Gregory Valiant. Estimating the unseen: improved estimators for entropy and other properties. In *Advances in Neural Information Processing Systems*, pages 2157–2165, 2013.

J von Neumann. The geometry of orthogonal spaces, functional operators-vol. ii. *Annals of Math. Studies*, 22, 1950.

Grace Wahba. *Spline models for observational data*, volume 59. Siam, 1990.

Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2): 1–305, 2008.

M Byron Yu, John P Cunningham, Gopal Santhanam, Stephen I Ryu, Krishna V Shenoy, and Maneesh Sahani. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. In *Advances in neural information processing systems*, pages 1881–1888, 2009.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

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

Song Chun Zhu, Xiu Wen Liu, and Ying Nian Wu. Exploring texture ensembles by efficient markov chain monte carlo-toward a" trichromacy" theory of texture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):554–569, 2000.

Appendix A

Maximum Entropy Flow Networks

Architectural and Training Choices

For sections 3.3.1 and 3.3.2, we use 10 layers of planar flow, and use with ADADELTA [Zeiler, 2012]. For section 3.3.3 we use real NVP structure and use ADAM [Kingma and Ba, 2014] with learning rate = 0.001. For all our experiments, we use $i_{max} = 3000$, $\beta = 4$, $\gamma = 0.25$. For section 3.3.1 and section 3.3.2 we use $n = 300$, $\widetilde{n} = 1000$, $k_{\max} = 10$; for section 3.3.3 we use $n = \widetilde{n} = 2$, $k_{\max} = 8$. In section 3.3.3, we use 3 residual blocks with 32 feature maps for each coupling layer and downscale 3 times.

Appendix B

The Continuous Bernoulli

Continuous Bernoulli and Beta Differences

While the beta distribution, having two parameters, is more flexible than the continuous Bernoulli, these distributions have a key difference: concentration of mass at the extrema. For a given continuous Bernoulli parameter λ, beta distribution parameters α_λ and β_λ can be found so that the corresponding distributions are very close (by matching the mean and variance), but this closeness does not happen at the extrema. To see this, we denote a beta density as $p(x|\alpha, \beta)$. We have:

$$\lim_{x\to 0} \log \frac{p(x|\lambda)}{p(x|\alpha,\beta)} = \lim_{x\to 0} \log C(\lambda) + \log B(\alpha,\beta) + x\log\lambda + (1-x)\log(1-\lambda) \tag{B.1}$$

$$- (\alpha-1)\log x - (\beta-1)\log(1-x) \tag{B.2}$$

$$= \begin{cases} -\infty & \text{if } \alpha < 1 \\ \log C(\lambda) + \log(1-\lambda) + \log B(\alpha,\beta) & \text{if } \alpha = 1 \\ \infty & \text{if } \alpha > 1 \end{cases} \tag{B.3}$$

where $B(\cdot, \cdot)$ is the beta function. This implies that:

$$\lim_{x \to 0} \frac{p(x|\lambda)}{p(x|\alpha, \beta)} = \begin{cases} 0 & \text{if } \alpha < 1 \\ C(\lambda)(1-\lambda)B(\alpha, \beta) & \text{if } \alpha = 1 \\ \infty & \text{if } \alpha > 1 \end{cases} \tag{B.4}$$

So that the continuous Bernoulli and the beta can place a comparable amount of mass around 0 only for $\alpha = 1$, otherwise the continuous Bernoulli places more mass if $\alpha > 1$ and less if $\alpha < 1$. In particular, for $\lambda < 0.5$ we have that $\alpha_\lambda < 1$, so that the beta distribution places much more mass around 0 than the continuous Bernoulli to which it is very similar, as can be seen in figure B.1. Note that this does not imply that the continuous Bernoulli is not placing most of its mass around 0, just not as much as the beta. This key insight highlights that even "similar looking" continuous Bernoullis and betas behave considerably differently at the extrema, which is precisely the most important part of the densities when modeling almost binary data (such as MNIST, which while almost binary, does have grayscale pixels). Empirically, we find that the beta distributed VAE produces means that are less extremal (i.e. grayer images, as seen in figure B.6), which implies that each beta was shifted away from 0 and 1 to reduce adding too much mass to the extrema (precisely the effect of figure B.1). Finally, note than an analogous discussion holds around 1, since:

$$\lim_{x \to 1} \frac{p(x|\lambda)}{p(x|\alpha, \beta)} = \begin{cases} 0 & \text{if } \beta < 1 \\ C(\lambda)\lambda B(\alpha, \beta) & \text{if } \beta = 1 \\ \infty & \text{if } \beta > 1 \end{cases} \tag{B.5}$$

Architectural and Training Choices

We preprocess MNIST by following the standard procedure of adding uniform $[0, 1]$ noise to the integer pixel values between 0 and 255 and then dividing by 256, resulting

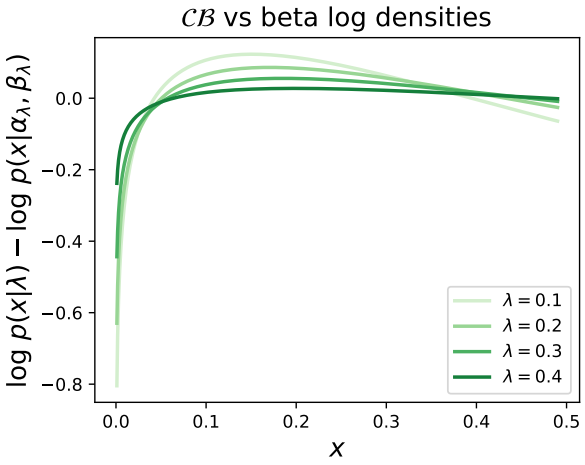


Figure B.1: Behavior of continuous Bernoulli against similar beta around 0.

in values in $[0, 1]$. For all our MNIST experiments, we use a latent dimension of 20, an encoder with two hidden layers with 500 units each, with *ReLU* nonlinearities, followed by a dropout layer (with parameter 0.9) [Srivastava et al., 2014]. The output layer of the encoder has no nonlinearity for the mean and a softplus nonlinearity for the standard deviation. The decoder also has two hidden layers with 500 units, *ReLU* nonlinearities and dropout, as does the classifier we used to compute the inception score (which has a softmax nonlinearity). The decoder has softplus nonlinearities to enforce nonnegativity (Gaussian standard deviation and beta parameters), sigmoid to enforce values in $(0, 1)$ (continuous Bernoulli, Bernoulli and Gaussian mean). We use a learning rate of 0.001 except for the Gaussian VAE, where we use 0.0001, and optimize with ADAM [Kingma and Ba, 2014] for 100 epochs.

We preprocess CIFAR-10 in the same fashion as MNIST. For CIFAR-10 the latent dimension is 50 and the learning rate for continuous Bernoulli VAE and Bernoulli VAE is 0.001, and 0.0001 for the other distributions. The encoder consists of four convolutional layers, followed by two fully connected ones. The convolutions have respectively, 3, 32, 32 and 32 features, kernel size 2, 2, 3 and 3, strides 1, 2, 1, 1 and are followed by *ReLU* nonlinearities. The fully connected hidden layer has 128 units

and a *ReLU* non linearity. The decoder has an analogous "reversed" architecture. The classifier used to compute the inception score has a convolution with 32 features and kernel size 3 followed by a *ReLU* activation, 10 residual blocks [He et al., 2016] at 3 different resolutions and a dense layer with a softmax nonlinearity. Each residual block consists a convolution, *ReLU*, batch normalization, another convolution and adding the result to the input. At the end of each residual block, the resolution is decreased with a convolution with stride two that doubles the number of features, and then dropout (with parameter 0.5) is applied.

Further Experimental Results

MNIST

Inception scores

In this section we show more inception scores obtained by the continuous Bernoulli VAE and Bernoulli VAE on MNIST, by transforming the decoder with μ or μ^{-1} and/or by sampling from $\mathscr{CB}$ or $\mathscr{B}$. The results are in figure B.2. Left panel has data (black), $\mathscr{B}(\lambda_{\theta^*(p)})$ (dark orange), $\mathscr{B}(\lambda_{\theta^*(\widetilde{p})})$ (light orange), $\mathscr{B}(\mu(\lambda_{\theta^*(p)}))$ (dark purple), $\mathscr{B}(\mu(\lambda_{\theta^*(\widetilde{p})}))$ (light purple), $\mu(\lambda_{\theta^*(p)})$ (dark green) and $\mu(\lambda_{\theta^*(\widetilde{p})})$ (light green). Right panel has data (black), $\mathscr{CB}(\mu^{-1}(\lambda_{\theta^*(p)}))$ (dark orange), $\mathscr{CB}(\mu^{-1}(\lambda_{\theta^*(\widetilde{p})}))$ (light orange), $\mathscr{B}(\mu^{-1}(\lambda_{\theta^*(p)}))$ (dark purple), $\mathscr{B}(\mu^{-1}(\lambda_{\theta^*(\widetilde{p})}))$ (light purple), $\mu^{-1}(\lambda_{\theta^*(p)})$ (dark green) and $\mu^{-1}(\lambda_{\theta^*(\widetilde{p})})$ (light green). While these inception scores are not what one would normally sample, we can see that the continuous Bernoulli VAE achieves better inception scores than the Bernoulli VAE, regardless of how the decoder is used to obtain the samples. The only situation in which the Bernoulli VAE manages to perform similarly to the continuous Bernoulli is when the decoder is corrected by applying μ^{-1} after training, similarly to what we observed for the ELBO (although the ELBO is still lower after doing this correction).

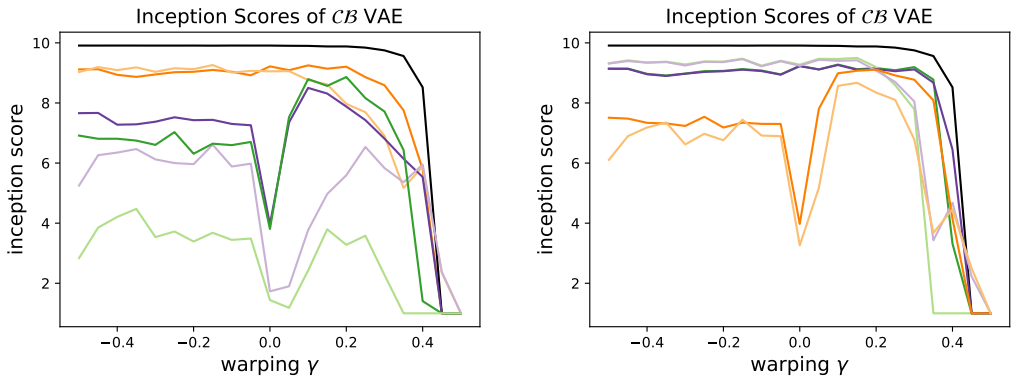


Figure B.2: Inception scores for continuous Bernoulli VAE (dark) and Bernoulli VAE (light). See text for details.

Samples

In this section we show samples used to compute the inception scores for MNIST (for warping $\gamma = 0$, that is, without transforming the data). Samples can be seen in figures B.3 to B.6.

CIFAR-10 samples

In this section we show samples used to compute the inception scores for CIFAR-10 in figures B.7 to B.9.

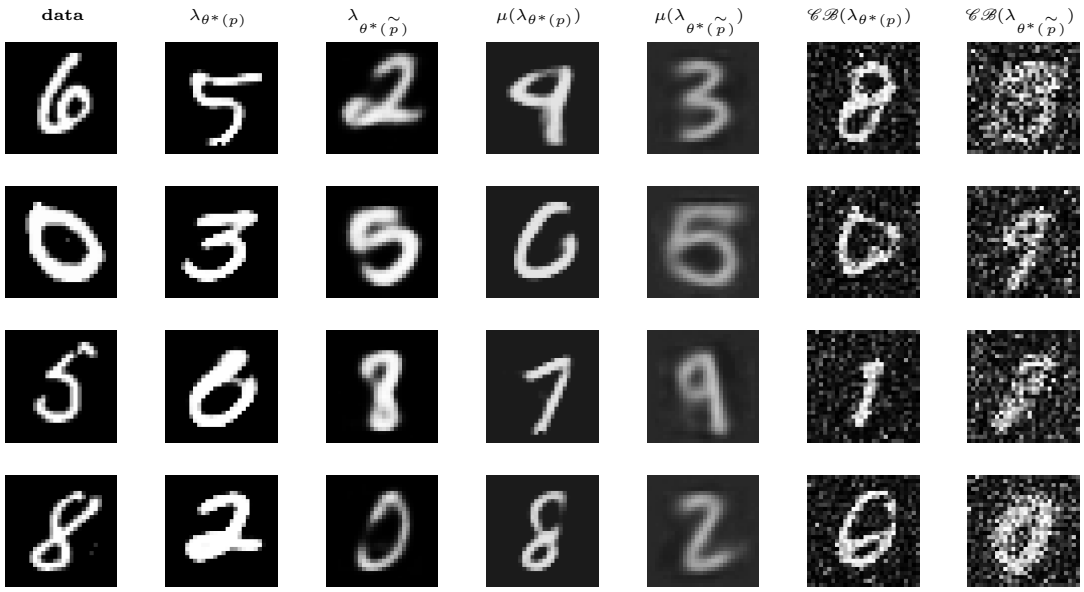


Figure B.3: MNIST continuous Bernoulli VAE and Bernoulli VAE samples 1. First three columns are also shown in the main manuscript.

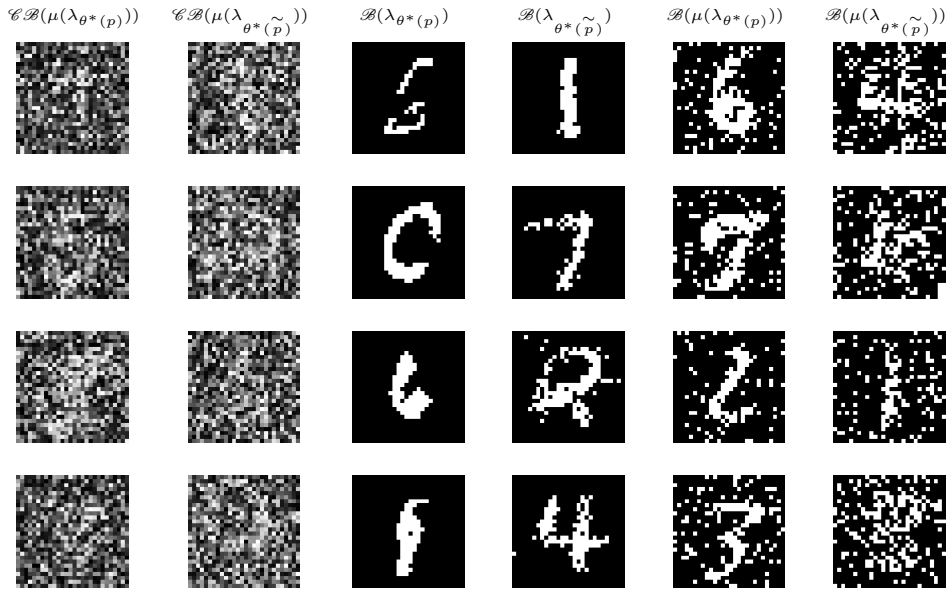


Figure B.4: MNIST continuous Bernoulli VAE and Bernoulli VAE samples 2.

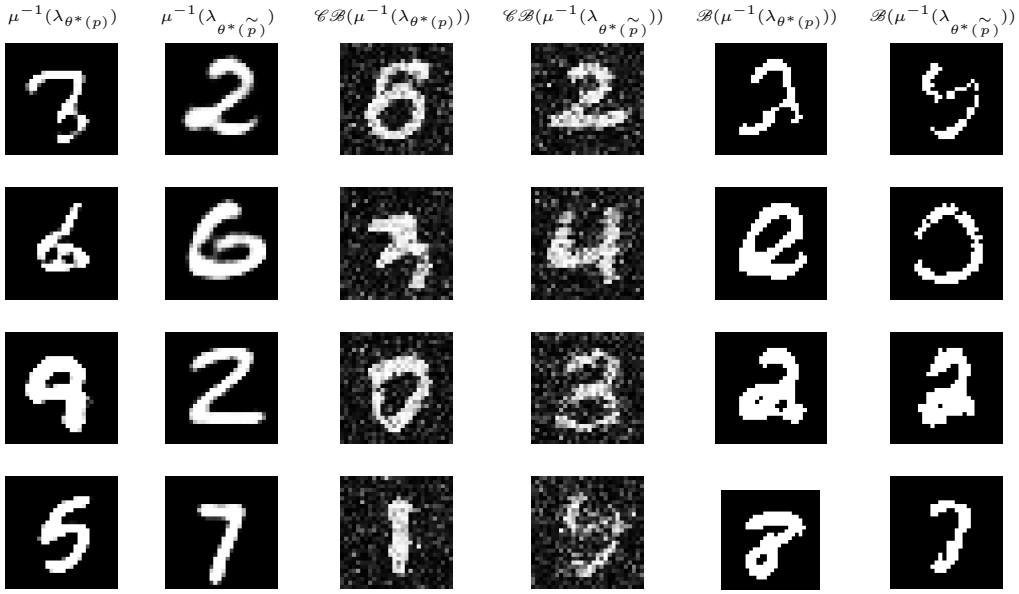


Figure B.5: MNIST continuous Bernoulli VAE and Bernoulli VAE samples 3.

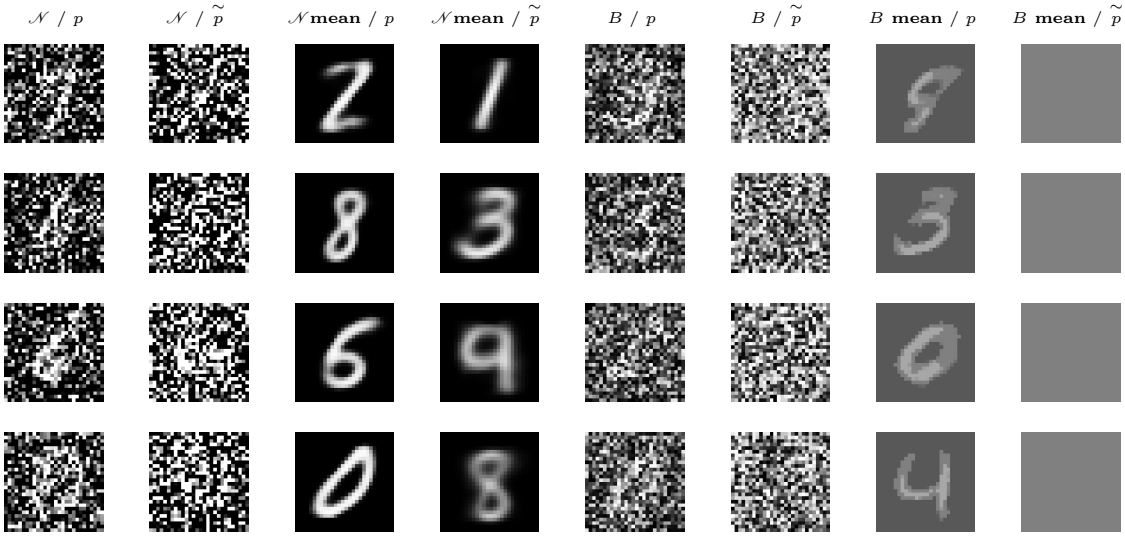


Figure B.6: MNIST Gaussian VAE (denoted $\mathcal{N}$) and beta distribution VAE (denoted B) samples, both including normalizing constants and ignoring them (denoted with tilde). Third columns is also shown in the main manuscript.

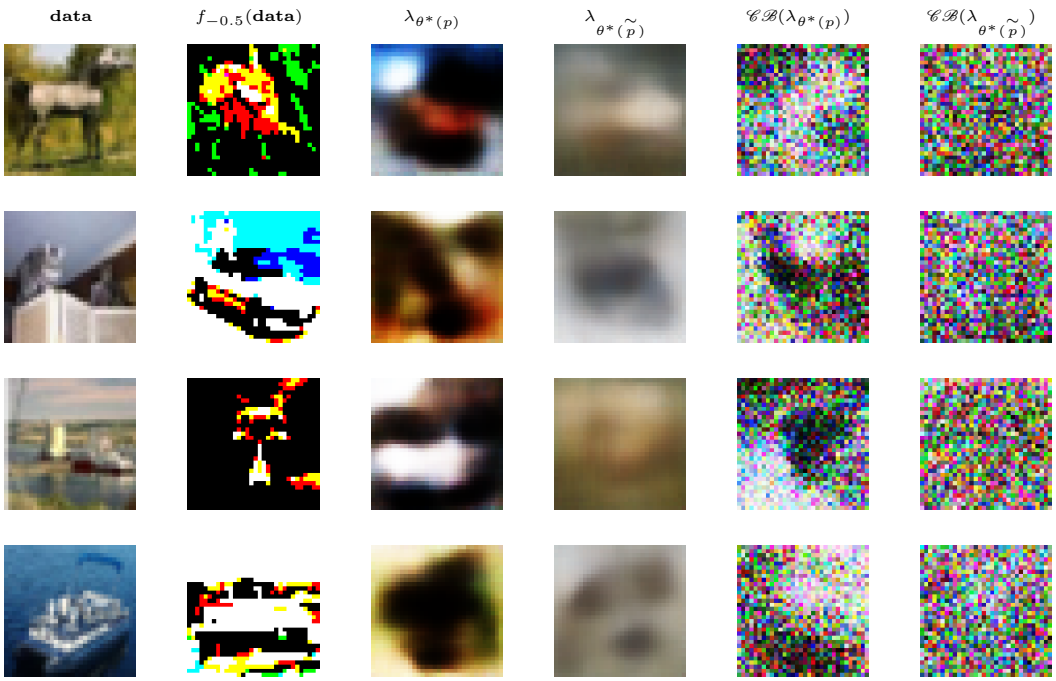


Figure B.7: CIFAR-10 continuous Bernoulli VAE and Bernoulli VAE samples 1.

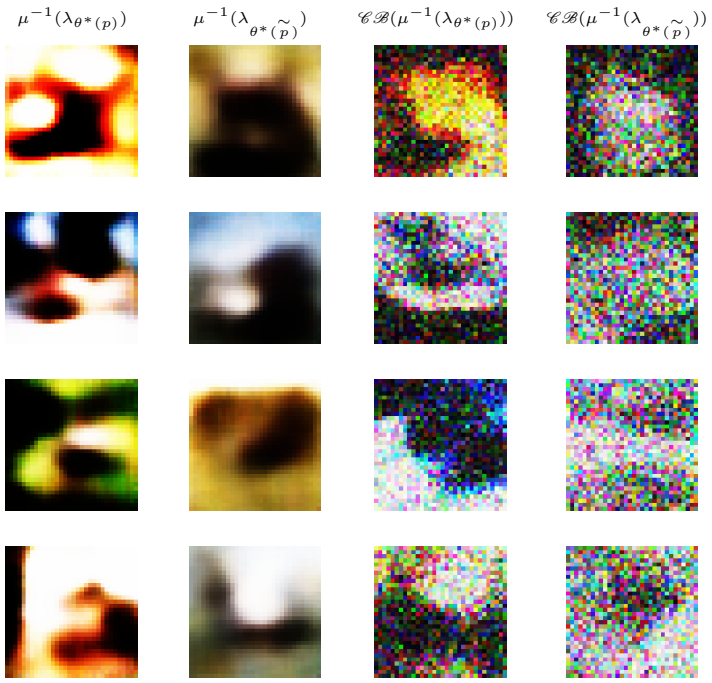


Figure B.8: CIFAR-10 continuous Bernoulli VAE and Bernoulli VAE samples 2.

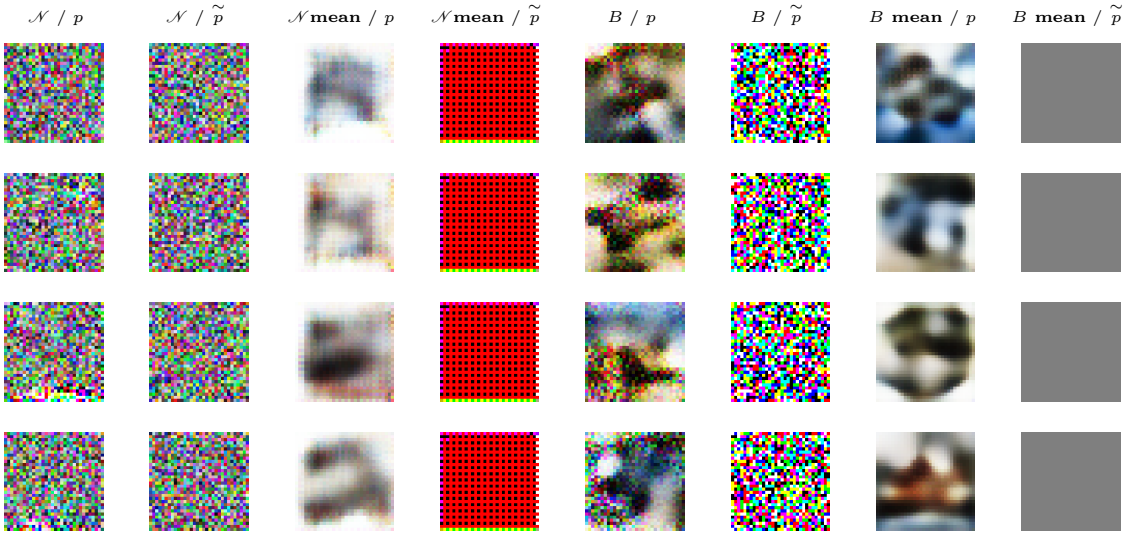


Figure B.9: CIFAR-10 Gaussian VAE (denoted $\mathcal{N}$) and beta distribution VAE (denoted B) samples, both including normalizing constants and ignoring them (denoted with tilde).

Appendix C

Deep Random Splines

Projecting onto the Space of Smooth Splines

As mentioned in section 5.2.2, mapping to $\Psi = \cap_{j=0}^{s+1} \mathscr{C}_j$ can be achieved through the method of alternating projections. As mentioned previously, projecting onto $\mathscr{C}_0$ can be easily done through eigen-decomposition. We now go through the details on how to project onto the other $\mathscr{C}_j$ sets. We will only cover $\mathscr{C}_1$, $\mathscr{C}_2$ and $\mathscr{C}_3$ for odd-degree splines as we used splines of degree 3 and smoothness 2, but projecting onto $\mathscr{C}_j$ for $j \geq 4$ for higher degree splines can be done in an analogous way. Projections for even degree splines can also be derived in an analogous way.

Continuity Projection for Splines of Odd Degree

Suppose we are given $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$, which are $(k+1) \times (k+1)$ matrices (not necessarily in Ψ), defining a piecewise polynomial as in equation 5.2. Computing the projection $(X_*^{(i)}, Y_*^{(i)})_{i=1}^I$ of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ onto $\mathscr{C}_1$ can be done by solving the following

optimization problem:

$$(X_*^{(i)}, Y_*^{(i)})_{i=1}^I = \underset{(X^{(i)},Y^{(i)})_{i=1}^I}{\text{minimize}} \sum_{i=1}^{I} ||X^{(i)} - Q_1^{(i)}||_F^2 + ||Y^{(i)} - Q_2^{(i)}||_F^2 \tag{C.1}$$
$$\text{subject to } (t_i - t_{i-1})[t_i]^\top Y^{(i)}[t_i] = (t_{i+1} - t_i)[t_i]^\top X^{(i+1)}[t_i]$$
$$\text{for } i = 1, \ldots, I-1$$

where $||\cdot||_F$ denotes the Frobenius norm and each constraint is merely forcing the piecewise function to be continuous at knot i for $i = 1, \ldots, I-1$. Note that this is a quadratic optimization problem with linear constraints, and can be solved analytically. The corresponding Lagrangian is:

$$L((X^{(i)}, Y^{(i)})_{i=1}^I, \lambda) = \sum_{i=1}^{I} ||X^{(i)} - Q_1^{(i)}||_F^2 + ||Y^{(i)} - Q_2^{(i)}||_F^2 \tag{C.2}$$
$$+ \sum_{i=1}^{I-1} \lambda_i \Big((t_i - t_{i-1})[t_i]^\top Y^{(i)}[t_i] - (t_{i+1} - t_i)[t_i]^\top X^{(i+1)}[t_i] \Big)$$

where $\lambda = (\lambda_1, \ldots, \lambda_{I-1})^\top \in \mathbb{R}^{I-1}$. By solving the KKT conditions, it can be verified that:

$$\begin{cases} X_*^{(i)} = Q_1^{(i)} + \frac{\lambda_{i-1}^*}{2} A_{i-1} \text{ , for } i = 1, \ldots, I \\ Y_*^{(i)} = Q_2^{(i)} - \frac{c_i \lambda_i^*}{2} A_i \text{ , for } i = 1, \ldots, I \\ \lambda_i^* = \frac{2}{1+c_i^2} \frac{[t_i]^\top (c_i Q_2^{(i)} - Q_1^{(i+1)})[t_i]}{([t_i]^\top [t_i])^2} \text{ , for } i = 1, \ldots, I-1 \end{cases} \tag{C.3}$$

where $c_i = \frac{t_i - t_{i-1}}{t_{i+1} - t_i}$ for $i = 1, \ldots, I-1$, $c_I = 0$, $\lambda_0^* = 0$, $\lambda_I^* = 0$ and $A_i = [t_i][t_i]^\top$ for $i = 0, \ldots, I$.

Differentiability Projection for Splines of Odd Degree

Analogously, computing the projection $(X_*^{(i)}, Y_*^{(i)})_{i=1}^I$ of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ onto $\mathscr{C}_2$ can be done by solving the following optimization problem:

$$\begin{aligned}(X_*^{(i)}, Y_*^{(i)})_{i=1}^I = \underset{(X^{(i)},Y^{(i)})_{i=1}^I}{\text{minimize}} \quad & \sum_{i=1}^{I} ||X^{(i)} - Q_1^{(i)}||_F^2 + ||Y^{(i)} - Q_2^{(i)}||_F^2 \\ \text{subject to} \quad & -[t_i]^\top X^{(i)}[t_i] + [t_i]^\top Y^{(i)}[t_i] \\ & + (t_i - t_{i-1})[t_i']^\top Y^{(i)}[t_i] + (t_i - t_{i-1})[t_i]^\top Y^{(i)}[t_i'] \\ & = -[t_i]^\top X^{(i+1)}[t_i] + (t_{i+1} - t_i)[t_i']^\top X^{(i+1)}[t_i] \\ & + (t_{i+1} - t_i)[t_i]^\top X^{(i+1)}[t_i'] + [t_i]^\top Y^{(i+1)}[t_i] \\ \text{for } & i = 1, \ldots, I-1\end{aligned} \tag{C.4}$$

where $[t'] = (0, 1, 2t, 3t^2, \ldots, kt^{k-1})^\top$ and each constraint is now forcing the values of the left and right derivatives of the piecewise function to match at knot i for $i = 1, \ldots, I-1$. Again, this is a quadratic optimization problem with linear constraints. By writing the Lagrangian and solving the KKT conditions, we get:

$$\begin{cases} X_*^{(i)} = Q_1^{(i)} + \frac{\lambda_i^*}{2} A_i - \frac{\lambda_{i-i}^*}{2}(A_{i-1} - (t_i - t_{i-1})M_{i-1}) \text{ , for } i = 1, \ldots, I \\ Y_*^{(i)} = Q_2^{(i)} - \frac{\lambda_i^*}{2}(A_i + (t_i - t_{i-1})M_i) + \frac{\lambda_{i-1}^*}{2} A_{i-1} \text{ , for } i = 1, \ldots, I \end{cases} \tag{C.5}$$

where $M_i = [t_i][t_i']^\top + [t_i'][t_i]^\top$ for $i = 0, \ldots, I$ and:

$$\begin{aligned} & \lambda_{i-1}^* \Big([t_i]^\top (A_{i-1} - \frac{t_i - t_{i-1}}{2} M_{i-1})[t_i] + (t_i - t_{i-1})[t_i']^\top A_{i-1}[t_i] \Big) \\ & + \lambda_i^* \Big([t_i]^\top (-2A_i - \frac{t_{i+1} - 2t_i + t_{i-1}}{2} M_i)[t_i] \\ & \qquad + (t_{i+1} - 2t_i + t_{i-1} - (t_i - t_{i-1})^2 - (t_{i+1} - t_i)^2)[t_i']^\top M_i[t_i] \Big) \\ & + \lambda_{i+1}^* \Big([t_i]^\top (A_{i+1} + \frac{t_{i+1} - t_i}{2} M_{i+1})[t_i] - (t_{i+1} - t_i)[t_i']^\top A_{i+1}[t_i] \Big) \\ & = [t_i]^\top (Q_1^{(i)} - Q_1^{(i+1)} - Q_2^{(i)} + Q_2^{(i+1)})[t_i] + 2[t_i']^\top ((t_{i+1} - t_i)Q_1^{(i+1)} - (t_i - t_{i-1})Q_2^{(i)})[t_i] \end{aligned} \tag{C.6}$$

for $i = 1, \ldots, I-1$ and again, $\lambda_0^* = 0$ and $\lambda_I^* = 0$. This is a tridiagonal system of $I-1$ linear equations with $I-1$ unknowns and can be solved efficiently in $O(I)$ time with simplified Gaussian elimination.

Second Differentiability Projection for Splines of Odd Degree

Finally, computing the projection $(X_*^{(i)}, Y_*^{(i)})_{i=1}^I$ of $(Q_1^{(i)}, Q_2^{(i)})_{i=1}^I$ onto $\mathscr{C}_2$ can be done by solving the following optimization problem:

$$
\begin{aligned}
(X_*^{(i)}, Y_*^{(i)})_{i=1}^I = \underset{(X^{(i)}, Y^{(i)})_{i=1}^I}{\text{minimize}} \quad & \sum_{i=1}^{I} ||X^{(i)} - Q_1^{(i)}||_F^2 + ||Y^{(i)} - Q_2^{(i)}||_F^2 \\
\text{subject to} \quad & -2[t_i']^\top X^{(i)}[t_i] - 2[t_i]^\top X^{(i)}[t_i'] + 2[t_i']^\top Y^{(i)}[t_i] + 2[t_i]^\top Y^{(i)}[t_i'] \\
& + (t_i - t_{i-1})[t_i'']^\top Y^{(i)}[t_i] + 2(t_i - t_{i-1})[t_i']^\top Y^{(i)}[t_i'] \\
& + (t_i - t_{i-1})[t_i]^\top Y^{(i)}[t_i''] \\
& = -2[t_i']^\top X^{(i+1)}[t_i] - 2[t_i]^\top X^{(i+1)}[t_i'] + (t_{i+1} - t_i)[t_i'']^\top X^{(i+1)}[t_i] \\
& + 2(t_{i+1} - t_i)[t_i']^\top X^{(i+1)}[t_i'] + (t_{i+1} - t_i)[t_i]^\top X^{(i+1)}[t_i''] \\
& + 2[t_i']^\top Y^{(i+1)}[t_i] + 2[t_i]^\top Y^{(i+1)}[t_i'] \\
\text{for } i = 1, \ldots, I-1 &
\end{aligned}
\tag{C.7}
$$

where $[t''] = (0, 0, 2, 6t, \ldots, k(k-1)t^{k-2})^\top$ and each constraint is now forcing the values of the left and right second derivatives of the piecewise function to match at knot i for $i = 1, \ldots, I-1$. Again, this is a quadratic optimization problem with linear constraints. By writing the Lagrangian and solving the KKT conditions, we get:

$$
\begin{cases}
X_*^{(i)} = Q_1^{(i)} + \lambda_i^* M_i - \frac{\lambda_{i-i}^*}{2} B_{i-1} \text{ , for } i = 1, \ldots, I \\
Y_*^{(i)} = Q_2^{(i)} - \frac{\lambda_i^*}{2} E_i + \lambda_{i-1}^* M_{i-1} \text{ , for } i = 1, \ldots, I
\end{cases}
\tag{C.8}
$$

where $B_{i-1} = 2M_{i-1} - (t_i - t_{i-1})([t''_{i-1}][t_{i-1}]^\top + 2[t'_{i-1}][t'_{i-1}]^\top + [t_{i-1}][t''_{i-1}]^\top)$ and $E_i = 2M_i - (t_i - t_{i-1})([t''_i][t_i]^\top + 2[t'_i][t'_i]^\top + [t_i][t''_i]^\top)$ for $i = 1, \ldots, I$ and:

$$
\begin{aligned}
&\lambda^*_{i-1}\Big([t'_i]^\top(2B_{i-1} + 4M_{i-1})[t_i] + 2(t_i - t_{i-1})[t''_i]^\top M_{i-1}[t_i] + 2(t_i - t_{i-1})[t'_i]^\top M_{i-1}[t'_i]\Big) \\
&+ \lambda^*_i\Big([t'_i]^\top(-8M_i - 2E_i - 2B_i)[t_i] + [t''_i]^\top((t_{i+1} - t_i)B_i - (t_i - t_{i-1})E_i)[t_i] \\
&\qquad + [t'_i]^\top((t_{i+1} - t_i)B_i - (t_i - t_{i-1})E_i)[t'_i]\Big) \\
&+ \lambda^*_{i+1}\Big([t'_i]^\top(E_{i+1} + 4M_{i+1})[t_i] - 2(t_{i+1} - t_i)[t''_i]^\top M_{i+1}[t_i] - 2(t_{i+1} - t_i)[t'_i]^\top M_{i+1}[t'_i]\Big) \\
&= 4[t'_i]^\top(Q_1^{(i)} - Q_1^{(i+1)} - Q_2^{(i)} + Q_2^{(i+1)})[t_i] + 2[t''_i]^\top((t_{i+1} - t_i)Q_1^{(i+1)} - (t_i - t_{i-1})Q_2^{(i)})[t_i] \\
&\quad + 2[t'_i]^\top((t_{i+1} - t_i)Q_1^{(i+1)} - (t_i - t_{i-1})Q_2^{(i)})[t'_i] \text{ , for } i = 1, \ldots, I-1
\end{aligned}
\tag{C.9}
$$

where again, $\lambda^*_0 = 0$ and $\lambda^*_I = 0$. Again, this is a tridiagonal system of $I - 1$ linear equations with $I - 1$ unknowns that can be solved efficiently.

Architectural Choices and Training Parameters

For our simulated data experiment, the state of each LSTM has 100 units, and $\widetilde{f}$ is a feed-forward neural network with ReLU activations and with 3 hidden layers, each one with 100 units. We apply 102 iterations of the method of alternating projections. For the feed-forward architecture in PfLDS, we also used 3 hidden layers, each with 100 units. We used a mini-batch of size 2 and the learning rate was 0.001. For the real data experiments we used the same choices, except the state of each LSTM has 25 units and $\widetilde{f}$ is a feed-forward network with ReLU activations and with 3 hidden layers, each one with 10 units (we tried more complicated architectures but saw no improvement).

Data Preprocessing

Reaching Data Preprocessing

We include only successful trials (i.e. when the primate reaches to the correct location) and use only spikes occurring in a window of $-100ms$ and $300ms$ from the time that movement starts. We also reduce the total number of neurons as inference with our method requires one LSTM per neuron and having too many neurons renders training slow. In order to do so, we use the following GLM:

$$y_r \sim \text{Multinomial}\Big(C, \text{softmax}(\widetilde{K}_{r,\cdot}^{\top}\beta)\Big) \tag{C.10}$$

where y_r is the trial type of trial r, $C = 40$ is the number of trial types, $\widetilde{K}_{r,\cdot} \in \mathbb{R}^N$ is a vector containing the (centered and standardized) number of spikes in trial r for each of the $N = 223$ neurons, and $\beta \in \mathbb{R}^{N \times C}$ are the GLM parameters. We train the GLM using group lasso [Yuan and Lin, 2006], where the groups are defined by neurons. That is, the GLM is trained through maximum likelihood with an added penalty:

$$\lambda \sum_{n=1}^{N} ||\beta_{n,\cdot}||_2^2 \tag{C.11}$$

where $\beta_{n,\cdot}$ is the n^{th} row of β. This makes it so that the coefficients in each group hit zero simultaneously. A neuron n is removed if $||\hat{\beta}_{n,\cdot}|| = 0$. We use a regularization parameter λ such that all but 20 neurons are removed. This provides a principled way of reducing the number of neurons while making sure that the kept neurons are useful. As PfLDS does not require the use of LSTMs, it can be run on the data without removing neurons. While doing this did increase performance of PfLDS, it did so very marginally and our model still heavily outperformed PfLDS.

Cycling Data Preprocessing

Once again, we only keep successful trials (i.e. when the primate pedals in the correct direction and speed) and reduce the total number of neurons $N = 256$ to 20 by using

group lasso. Since each trial has a different length, we extend every trial to have the same length as the longest trial. We add no spikes to these extended time periods.