University of Edinburgh

School of Informatics

Doctor of Philosophy

Accurate and reliable probabilistic modeling with high-dimensional data

Artur Bekasov

2022

Abstract

Machine learning studies algorithms for *learning from data.* Probabilistic modeling and reasoning define a principled framework for machine learning, where *probability theory* is used to represent and manipulate knowledge. In this thesis we focus on two fundamental tasks in probabilistic machine learning: *probabilistic prediction* and *density estimation.* We study reliability of probabilistic predictive models, propose flexible models for density estimation, and propose a novel training regime for densities with low-dimensional structure.

Neural networks demonstrate state-of-the-art performance in many different prediction tasks. At the same time, modern neural networks trained by maximum likelihood have poorly calibrated predictive uncertainties and suffer from adversarial examples. We hypothesize that careful probabilistic treatment of neural networks would make them better calibrated and more robust. However, *Bayesian neural networks* have to rely on uninformative priors and crude approximations, which makes it difficult to test this hypothesis. In this thesis we take a step back and study adversarial robustness of a simple, *linear* model, demonstrating that it no longer suffers from calibration errors on adversarial points when the approximate inference method is accurate and the prior is chosen carefully.

Classic density estimation methods do not scale to complex, high-dimensional data like natural images. *Normalizing flows* model the target density as an invertible transformation of a simple base density, and demonstrate good results in high-dimensional density estimation tasks. State-of-the-art normalizing flow architectures rely on parametrizations of univariate invertible functions. Simple additive/affine parametrizations are often used, stacking many layers to express complex transformations. In this thesis we propose novel parametrizations based on *cubic* and *rational-quadratic* splines. The proposed flows demonstrate improved parameter-efficiency and advance state-of-the-art on several density estimation benchmarks.

The *manifold hypothesis* says that the data are likely to lie on a lower-dimensional manifold. This assumption is built into many machine learning models, but using it with density models like normalizing flows is difficult: the standard likelihood-based training objective becomes ill-defined. *Injective* normalizing flows can be implemented, but their training objective is no longer tractable, requiring approximations or heuristic alternatives. In this thesis we propose a novel training objective that uses *nested dropout* to align the latent space of a normalizing flow, allowing us to extract a sequence of manifold densities from the trained model. Our experiments demonstrate that the manifolds fit by the method match the data well.

Lay summary

Algorithms are lists of simple instructions that a computer can follow to perform a useful task. Consider an algorithm that can take an image of a medical scan and *predict* if the patient is likely to have a particular medical condition. Alternatively, consider an algorithm that can *generate* medical scans that we can plausibly observe for a patient with a particular condition. Such algorithms could potentially automate expensive and time-consuming diagnostic assessment, or help train medical professionals. Unfortunately, developing such algorithms by hand is infeasible: the logic that we would need to implement is simply too complex.

In *machine learning* we overcome this complexity by considering algorithms that use historical data (e.g. past medical scans with their associated conditions) to *learn* the required logic. Machine learning has been applied successfully across science and industry to solve tasks previously believed to require "intelligence". This includes identifying and classifying objects in images, understanding and generating speech and language, or matching human skill on games like chess and Go.

Problems tackled by machine learning algorithms are often associated with *uncertainty*. For example, given a single, low-quality medical scan, a doctor should realize the there is not enough information to give a *confident* diagnosis. Similarly, a doctor who has only ever seen a handful of scans would not consider themselves a subject expert, and will be reluctant to give confident diagnoses. The *probabilistic* approach to machine learning allows us to build algorithms that mimic this behavior, and only make confident predictions when justified to do so.

Unfortunately, probabilistic methods can be more difficult to implement and run, which is why machine learning practitioners often choose simpler alternatives. In this thesis, we demonstrate that even in a simple task these alternatives can have undesirable side effects. We then demonstrate how probabilistic methods can alleviate these side effects. Next, we propose a novel method that increases the complexity of tasks that we can solve using the probabilistic approach. Finally, we propose a way of using our prior knowledge about certain tasks to make learning more efficient. We hope our work motivates more practitioners to turn to probabilistic methods, and encourages further research in this area.

Acknowledgments

This thesis would not be possible without the guidance and support of my advisor, Iain. Iain had a significant impact on my growth and development as a researcher, and the clarity of his thinking and his deep understanding of the field were always an example to me. I am especially grateful to Iain for his support during the unusual circumstances surrounding the beginning of my PhD.

I am grateful to Sandra who relentlessly cheered me on and was there for me through thick and thin, and to the rest of my family whose support allowed me to move abroad to pursue higher education.

I thank my collaborators, Conor and George, who made the Chapter 3 possible, and who are the smartest people that I have ever worked with. A separate thank you to Conor whose wit and humor always provided a respite during conference travels and weekends at the office.

I am grateful to Carl, Ivana, Magda, Nestor, Benedek and Maria for all the fun times we had during and after work, and to all the members of the CDT and the inhabitants of the Informatics Forum for the talks and hallway conversations that contributed to my thinking.

I thank the other two members of my supervisory team, Michael and Hakan, who gave me useful advice during annual reviews. I am grateful to my colleagues at Amazon, and especially to Asim for supporting my decision to pursue a PhD, and to Tammo and Dustin for their patience and understanding while I was writing this thesis.

I thank the workshop and conference reviewers, who donated their time to read the work in this thesis and give feedback. In particular, I would like to thank Adam Golinski for suggesting the greedy sorting baseline and the fine-tuning experiment in Chapter 4.

The experiments in this thesis were made possible by the hard work of the contributors to `NumPy`, `Matplotlib`, `SciPy`, `PyTorch`, `Pyro`, and other open-source packages that I used. I am grateful to `nflows` contributors who helped polish and extend the package since its release.

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

Declaration

I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contribution and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

The work presented in Chapter 3 was previously published in the proceedings of the Thirty-third Conference on Neural Information Processing Systems (NeurIPS, 2019) as *Neural Spline Flows* by Conor Durkan*, Artur Bekasov*, Iain Murray, George Papamakarios (* indicates joint-first authorship). I collaborated with Conor on deriving and implementing the proposed and baseline methods (as reflected in joint-first authorship), was responsible for the generative image modeling experiments, and implemented substantial parts of `nflows`, the normalizing flow framework used for this project.

(*Artur Bekasov*)

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this thesis we study computer algorithms that can *learn from data*, i.e. extract *knowledge* from data, present this knowledge in a human-understandable form, or use it for making predictions. We can use such algorithms to solve complex tasks believed to require "intelligence", understand complex natural phenomena and make predictions about them, or design better medical interventions and government policies. Algorithms for learning from data are especially vital in the modern information age, where the rapid development of technology for gathering, storing and transmitting information resulted in an abundance of data across many different domains. As the size and complexity of available datasets grow, so does the importance of *automated* ways of analyzing this data (Hastie et al., 2001b).

Machine learning is a field at the intersection of computer science, statistics and mathematics that studies algorithms for learning from data. This learning can take many forms, as determined by the target application and by the data available. In this thesis we focus on two fundamental forms of learning: *supervised* learning and *unsupervised* learning.

In supervised learning, our aim is to use a dataset of labeled datapoints to *predict* labels for previously unseen datapoints. *Image classification* is a common example of a supervised learning task, where we have access to a dataset of human-labeled images, and use it to train a model to predict labels for new images. For example, we can classify handwritten digits (LeCun and Cortes, 2010), classify objects in natural images (Russakovsky et al., 2015), or detect presence of metastatic cancer tissue in images of histopathologic scans (Veeling et al., 2018).

In unsupervised learning, we aim to learn statistical *patterns* in unlabeled data, and use the discovered patterns to summarize the data, generate new datapoints, or define representations for further tasks. For example, *generative modeling* is an unsupervised learning task where we train a model that can sample novel points from the data distribution.

Besides obvious artistic applications (White et al., 2021), we can use such a model to sample future world states in model-based reinforcement learning (Racanière et al., 2017), learn simulators in likelihood-free inference (Cranmer et al., 2020), or use the latent variables discovered by the model as representations in downstream tasks (Brown et al., 2020).

Probabilistic modeling and reasoning define a principled framework for machine learning, where *probability theory* is used as a universal language for representing and manipulating knowledge (Murphy, 2012). Probabilistic machine learning is appealing as it explicitly accounts for the uncertainties present in all learning tasks: the irreducible uncertainty inherent to the world, and the uncertainty that results from learning with a finite dataset.

From a probabilistic viewpoint, supervised learning is formalized as the problem of estimating the conditional *predictive* distribution $p(y \mid x)$ for the label y and the datapoint x. Given a previously unseen datapoint x, the distribution $p(y \mid x)$ represents our belief that y is the label of x. Unsupervised learning is formalized as the *density estimation* task, where we aim to estimate the unconditional data distribution $p(x)$. We can then use the moments or parameters of $p(x)$ to summarize the data, sample *new* points from it, compute expectations with respect to it, and evaluate the density of new datapoints (Papamakarios, 2019).

1.1 Approximate inference for probabilistic prediction

In the probabilistic treatment, learning starts with defining a probabilistic *model* and a *prior* distribution on its parameters, which express our prior assumptions about the underlying process that generates the data. We then perform *Bayesian inference* to compute the *posterior* distribution over the parameter settings, where high densities should be associated with parameter values that are consistent with the data *and* the prior. In other words, having observed some data produced by the underlying process, we use the Bayesian machinery to *update* our prior belief, and express the new belief via the posterior distribution.

For complex, high-dimensional data the probabilistic model should be *flexible* enough to express the underlying process (Ghahramani, 2015). For example, in image classification the image label (say, the class of the object in the image) is a complex, non-linear function of the raw pixel values. A *neural network* is a parametric model that is defined as a chain of simple transformations, or *layers*. While each individual layer is not expressive, stacking multiple layers allows the neural network to express complex, non-linear functions. In fact, a neural network is a *universal* approximator: a neural network with sufficiently many hidden units can approximate *any* function (Hornik et al., 1989). Neural networks consistently demonstrate state-of-the-art results in supervised learning applications across various domains (Goodfellow et al., 2016).

Unfortunately, neural network models present a set of problems for the probabilistic treatment. Their individual parameters are not interpretable, which makes it difficult to define informative priors. The posteriors of such models are complex, high-dimensional and multi-modal, making probabilistic inference and prediction intractable. *Bayesian neural networks* (BNNs, MacKay, 1992a; Neal, 1994) use approximate inference methods to estimate the posterior over the neural network parameters, but rely on uninformative priors and approximations that make naive assumptions (Graves, 2011; Blundell et al., 2015).

Instead, most practitioners resort to *maximum a posteriori* (MAP) or *maximum likelihood* (MLE) estimation, where we compute a *point-estimate* of the full Bayesian posterior. Both approaches re-define inference as *optimization*, where non-convex optimization methods like gradient descent are used to find a local minimum in the parameter space. These approximations greatly simplify both inference and prediction, and accuracy of their predictions matches (or even beats) BNNs in practice, especially with large datasets.

While accurate, predictions made by modern non-Bayesian neural networks can fail in other aspects. In particular, their predictions are surprisingly *miscalibrated*: their confidences are poorly correlated with how accurate these predictions turn out to be (Guo et al., 2017). A form of miscalibration that has recently attracted a lot of interest is the *adversarial example* phenomenon (Szegedy et al., 2014; Goodfellow et al., 2014b). Adversarial examples are points in the data distribution that an otherwise accurate model makes *confident* errors on. Existence of such points means we can not rely on neural network predictions when making decisions associated with risk.

The fact that neural networks can make *confident* yet *wrong* predictions suggests that they underestimate their predictive uncertainty, i.e. do not know *when they do not know*. Researchers suggest that misspecified priors and the inability of MLE/MAP training to capture the model uncertainty underpin their miscalibration and the adversarial phenomenon (Gal and Smith, 2018). Unfortunately, testing this hypothesis is difficult in BNNs, which have to rely on crude approximations and uninformative priors.

In this thesis we take a step back and study adversarial robustness of a simpler model class, where meaningful priors can be defined, and approximate inference methods can be accurate. We take a *linear* model that suffers from calibration errors on adversarial points when trained by MLE/MAP, and measure the extent to which various approximate Bayesian inference methods can eliminate these errors. We show that *accurate* inference and *informative* priors can completely eliminate the calibration errors in this setup. We hope that our findings will further stimulate the development of Bayesian methods for neural networks.

1.2 Density estimation with normalizing flows

Density estimation is a fundamental problem in probabilistic machine learning. Classic density estimation methods like histograms, *kernel density estimation* (KDE) and mixture models do not scale to high-dimensional, complex densities, such as densities of natural images. *Normalizing flows* have been proposed as flexible parametric density models (Tabak and Turner, 2013; Rippel and Adams, 2013). A normalizing flow models a density as an invertible transformation of a simple base density like a multivariate standard normal. An arbitrarily complex density can be expressed by a flow with a sufficiently flexible invertible transformation. Parametrizing such flexible transformations has been the main focus of normalizing flow research.

Autoregressive (Kingma et al., 2016; Papamakarios et al., 2017) and *coupling* (Dinh et al., 2017) transformations have been proposed, where we use *neural networks* to parametrize invertible *elementwise* transformations of the inputs. Such transformations stay invertible by construction, yet can take advantage of the expressive power of neural networks. Flows based on autoregressive and coupling layers demonstrate excellent results across various density estimation tasks (Kingma and Dhariwal, 2018; Huang et al., 2018).

At the same time, the expressivity of autoregressive/coupling transformations depends on the expressivity of the elementwise function parametrizations. The first autoregressive (Kingma et al., 2016; Papamakarios et al., 2017) and coupling (Dinh et al., 2017) normalizing flows use simple additive/affine functions, and overcome the limited expressivity by stacking many transformations in a chain. Like in neural networks, this combination can produce expressive models, but long chains of transformations increase computational cost, and can result in vanishing/exploding gradients (Koehler et al., 2021). Instead, more complex invertible univariate function parametrizations have been proposed (Müller et al., 2018; Ho et al., 2019), which we review in Section 3.2.

In this thesis we propose novel invertible univariate function parametrizations based on monotonic *cubic polynomial splines* and monotonic *rational-quadratic splines*, which we use to build more parameter-efficient normalizing flows that improve upon state-of-the-art on several density estimation benchmarks.

1.3 Normalizing flows and low-dimensional structure

The *curse of dimensionality* makes high-dimensional density estimation problems particularly challenging. One implication of this phenomenon is that the number of datapoints required for an accurate estimate grows *exponentially* with the number of data dimensions, so we have to make assumptions about the density in order to make density estimation feasible. Common assumptions include the smoothness of the density, presence of symmetries or invariances in it, or its low intrinsic dimensionality (Papamakarios, 2019).

In normalizing flows based on autoregressive or coupling layers, smoothness and symmetries/invariances are typically enforced via architectural choices in the neural networks that parametrize the transformations. The *low intrinsic dimensionality* assumption (the *manifold hypothesis*) is the assumption that the target density lies on a lower-dimensional manifold, and does not span the full space. Building this assumption into normalizing flows is more difficult: the density is no longer well-defined in ambient space, which means that the standard normalizing flow training objective is also ill-defined. Kumar et al. (2020) and Brehmer and Cranmer (2020) use a more general normalizing flow setup that supports *injective* transformations, but the likelihood of an injective flow is no longer tractable, so the authors must use approximations or alternative, heuristic objectives.

In this thesis we take a different approach, and propose a novel training objective for a *standard*, full-dimensional flow. The new objective is based on the *nested dropout* idea by Rippel et al. (2014), and regularizes the latent space of the flow to allow extracting a sequence of densities on manifolds from the trained model. Our experiments show that the flow indeed captures the underlying data manifolds when using the proposed training objective. Our method does not require knowledge of the manifold dimension at training time, and is suitable for modeling data that lie *close* to a lower-dimensional manifold.

1.4 Thesis structure

This thesis consists of an introduction, three content chapters, and a discussion. Each of the content chapters is written to be self-contained: it provides the necessary background, and reviews the related literature. Chapter 2 is devoted to probabilistic prediction and Bayesian inference, where we study adversarial robustness of a Bayesian linear model. In Chapters 3 and 4 we study density estimation, focusing on normalizing flows: in Chapter 3 we propose novel normalizing flow parametrizations based on monotonic splines, while in Chapter 4 we propose a novel training objective for standard flows that allows extracting manifold densities from the trained model. Finally, in Chapter 5 we discuss our contributions and their implications, as well as avenues for future work.

Mathematical notation Following a common mathematical notation, in the rest of the thesis we use the default font for scalar variables (x) and scalar functions (f), bold font for vector variables ($\mathbf{x}$) and vector functions ($\mathbf{f}$), bold capitals for matrix variables ($\mathbf{M}$), and calligraphic capitals for sets ($\mathcal{D}$). We use $\mathbf{x}_{i:j}$ as a shortcut for $[\mathbf{x}_i \;\; \mathbf{x}_{i+1} \;\; \ldots \;\; \mathbf{x}_{j-1} \;\; \mathbf{x}_j]^\top$, only defined for $j \geq i$. The log function is always a natural logarithm, unless stated otherwise.

Chapter 2

Adversarial robustness of approximate Bayesian inference

This chapter is based on *Bayesian Adversarial Spheres: Bayesian Inference and Adversarial Examples in a Noiseless Setting* (Bekasov and Murray, 2018), a paper presented at the Bayesian Deep Learning Workshop, part of the Thirty-second Annual Conference on Neural Information Processing Systems (NeurIPS, 2018). The paper was selected for a spotlight presentation at the workshop.

In this chapter we expand on the paper by providing a more extensive introduction, background and discussion; performing a deeper analysis of the logistic regression posterior approximations in the spheres problem; and including new experiments on Bayesian logistic regression with *learned* neural features for planar classification, the spheres problem, and image classification.

2.1 Introduction

Neural networks are powerful function approximators, making them an important tool in the toolbox of a modern machine learning practitioner. *Learning-as-optimization* is the mainstream approach when it comes to training modern neural networks. However, the alternative — *learning-as-Bayesian-inference* — has attracted a lot of research interest throughout the years. In their pioneering work MacKay (1992a) and Neal (1994) explored the motivations behind Bayesian treatment of neural networks, as well as practical considerations of doing Bayesian inference for complex neural network posteriors. Since then the community has made significant progress in both methodology and applications of Bayesian neural networks (Gal, 2016).

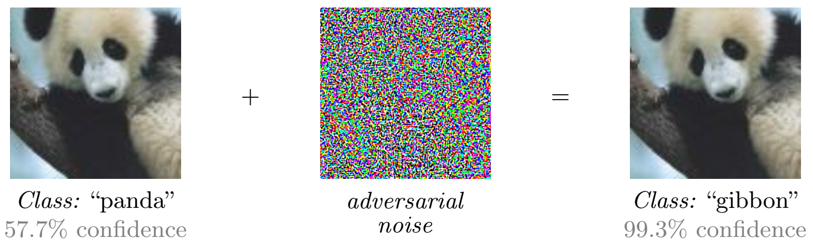


Figure 2.1: Adversarial example phenomenon. By adding adversarially constructed noise to the correctly classified input image we force the classifier to output an incorrect prediction with high confidence, even though the two images look indistinguishable to our eye. Reproduced with permission from (Goodfellow et al., 2014b).

However, Bayesian methods are yet to see wide adoption by deep learning practitioners. One common justification is that (regularized) maximum likelihood training is easy to implement and cheap to run, while its predictive performance as measured by accuracy or mean squared error (MSE) is often comparable to more involved Bayesian inference methods. This is especially true in the large data regime, where there is less uncertainty about the model parameters, and the regularization induced by the Bayesian prior and inference procedure plays a smaller role (Kendall and Gal, 2017).

Predictive performance is not the only aspect that practitioners care about, however. For example, when using model predictions for real-world decision-making it is important to have a *reliable* measure of how confident the model is, and hence how much we can trust its prediction. This is particularly important for risk-sensitive applications in e.g. medicine or autonomous driving, where machine learning models with *calibrated* predictive uncertainty — a strong correlation between predictive confidence and expected prediction error (Gal, 2016) — are especially desirable.

A particular manifestation of miscalibration that has recently attracted a lot of interest in the machine learning community is the *adversarial example* phenomenon (Szegedy et al., 2014; Goodfellow et al., 2014b). Adversarial examples are points found on the data manifold that an otherwise accurate model *confidently* misclassifies. In particular, such points can lie so close to correctly classified points that a human can not distinguish between a correctly classified point and an adversarial point. Fig. 2.1 illustrates the phenomenon in image data.

Rawat et al. (2017) and Gal and Smith (2018) report that *Bayesian neural networks* (BNNs, Blundell et al., 2015; Louizos and Welling, 2017) are better calibrated, and in particular become more *uncertain* on adversarial points when compared to neural networks trained by maximum likelihood. Unfortunately, studying these models is difficult, as the probabilistic treatment of neural networks is notoriously hard: their weights are not interpretable and hence difficult to put an informed prior on, and their posteriors are complex and high-dimensional, making accurate inference challenging. As a result, we typically assume uninformative priors and use approximate inference methods that make strong assumptions with poorly understood effects. This is consistent with the observations made by Rawat et al. and Gal and Smith that adversarial robustness of Bayesian neural networks depends strongly on the inference method used.

In this work, we take a step back and study the interaction between the Bayesian treatment and adversarial robustness using simple, *linear* predictive models and synthetic data. In particular, we consider the "adversarial sphere" problem introduced by Gilmer et al. (2018), but, instead of a multi-layer neural network like in the original work, we use a *Bayesian logistic regression* model with handcrafted features. This allows us to reason about the parameters of the model and perform accurate approximate inference.

We first demonstrate that when using *maximum likelihood* (MLE) or *maximum a posteriori* (MAP) estimation this simple model makes *zero* mistakes on a large test set, yet adversarial optimization can discover *confident* errors on the data manifold. We then experiment with the prior and the approximate inference method to try to make the predictions uncertain on these points.

We find that performing *accurate* inference and using *informative* priors eliminates the calibration errors in this setting. At the same time, we find that *bootstrap*, a popular non-Bayesian uncertainty estimation method, does *not* have the same effect. We use the simplicity of the setup to study the posterior approximations of bootstrap and other methods more closely.

2.1.1 Adversarial robustness of hybrid models

In complex prediction problems models with handcrafted features are outperformed by end-to-end trained neural networks, and hence are of little interest to practitioners. At the same time, as mentioned above, approximate Bayesian inference can be inaccurate when it comes to complex models like neural networks. These two considerations limit the practical impact of our work.

What if we consider a Bayesian linear model, but instead of *designing* features for it we would *learn* a feature space using a non-linear model like a neural network? After all, a neural network classifier itself could be interpreted as a logistic regression with an *adaptive basis* (Murphy, 2012, Chapter 16), i.e. a logistic regression fit in a learned neural feature space. The feature space can be fit via maximum likelihood, which could potentially get us the best of both worlds: a non-linear model trained cheaply using standard optimization tools, but with a Bayesian layer on top of it for better predictive uncertainty.

A *hybrid* approach of this form has been explored before (Snoek et al., 2015; Bauer et al., 2017; Bradshaw et al., 2017), with promising results on tasks designed to measure model calibration. In the remainder of this chapter we study robustness of these models as measured by calibration errors on adversarial points. We run experiments on synthetic data, including the "adversarial sphere" setup with no handcrafted features, as well as high-dimensional natural image datasets.

We observe that a hybrid model does *not* retain the adversarial robustness of a fully Bayesian classifier. Further analysis suggests that such methods are fundamentally flawed, due to the fact that the neural representation model itself remains vulnerable.

2.1.2 Contributions

In summary, we make the following contributions in this chapter:

- We show that an accurate *linear* model with handcrafted features suffers from calibration errors on adversarial points in the concentric spheres problem introduced by Gilmer et al. (2018).
- We use the spheres problem with a linear model to show that approximate Bayesian inference methods take the model from being *confidently wrong* to being *highly uncertain* on adversarial points, and that the same effect is *not* achieved when using crude inference methods like MAP or bootstrap.
- We compare the inference results of MCMC, SVI and Laplace, studying the posterior approximations and the failure modes of these methods. We analyze the transferability of the adversarial points across predictive ensembles, showcasing the inability of bootstrap to explore certain directions in the posterior.
- We evaluate an informed *hierarchical* prior that incorporates our knowledge about the symmetries in the problem, and demonstrate that it completely eliminates the calibration errors.
- We extend the experiments to Bayesian logistic regression with *learned*, *neural* features, running the method on non-linear 2D classification, the high-dimensional spheres problem, and an image classification problem. We demonstrate that the positive results above do *not* transfer to such models, because the feature model itself remains vulnerable.

2.2 Background

Consider an abstract binary[1] classifier $p(y = 1 \mid \mathbf{x}) = f_\theta(\mathbf{x})$ with parameters θ. In a standard supervised learning setup given a set of training examples $\mathcal{D}$ we first choose how to parametrize f_θ (as a linear classifier, a neural network, a decision tree, etc.), and possibly define a prior $p(\theta)$. We then fit $\theta^* = \operatorname{argmin}_\theta \mathcal{L}(\theta; \mathcal{D})$ for some loss $\mathcal{L}$, or perform inference to get the posterior $p(\theta \mid \mathcal{D})$. To guide our choice of the parametrization and the fitting/inference procedure, we need to understand what properties we want a *good* classifier to have.

Naturally, we want the predictions made by a good classifier to be *correct* and to *generalize* to unseen data, i.e. we want it to have high accuracy on a held-out test dataset. While accuracy is undoubtedly of great importance, it is not the *only* desirable property of a good classifier. For example, in certain applications it is crucial for the model's predictions to be *interpretable*, so that a practitioner can debug the model, and a user can understand the reason for a particular prediction. In this work, we focus on another important property of a good classifier: *calibration*. In particular, we use *adversarial optimization* to find points for which the model is most miscalibrated.

2.2.1 Model calibration

In probabilistic prediction, we report a *confidence/uncertainty* alongside the prediction. For a probabilistic *binary* classifier, the output $f_\theta(\mathbf{x})$ defines the confidence, which is equal to $f_\theta(\mathbf{x})$ for $y = 1$, and $1 - f_\theta(\mathbf{x})$ for $y = 0$. We would like this confidence to be *calibrated*, meaning that we want the classifier to *reliably* indicate to what degree it is uncertain about its prediction, and hence how likely it is to be wrong. More precisely, for all $\mathbf{x}$ s.t. $\alpha < f_\theta(\mathbf{x}) < \beta$ we want the classifier's accuracy on these $\mathbf{x}$'s to be between α and β. The mismatch between the expected accuracy and the measured accuracy for a set of (α, β) ranges defines the common calibration metric — Expected Calibration Error (ECE, Naeini et al., 2015).

Guo et al. (2017) report that modern neural networks trained by maximum likelihood are surprisingly miscalibrated as measured by ECE, and in fact become *less* calibrated as their predictive performance improves. Knowing when to trust the model's prediction is crucial if the model is to be used for decision making, especially in safety-critical applications like autonomous driving or medicine.

Non-Bayesian methods for calibrating neural networks have been proposed. For example, we can use *temperature scaling* (Guo et al., 2017) to re-scale the predictive uncertainties

[1]We consider binary classification problems in this chapter to simplify the mathematical notation and the experiments, and to make the results easier to interpret.

after training, choosing the "temperature" that maximizes the likelihood on the validation set. Mukhoti et al. (2020) propose replacing the maximum likelihood loss with the *focal loss* (Lin et al., 2017) that penalizes low-entropy predictive distributions, thus discouraging the network from becoming over-confident. Finally, we can directly optimize a differentiable approximation to ECE by replacing the non-differentiable binning operation with *soft binning* (Karandikar et al., 2021; Bohdal et al., 2021).

We do not expect the non-Bayesian methods above to make the model uncertain on out-of-distribution points: it is only the training/validation points that contribute to the training/calibration objectives. Bayesian methods, on the other hand, explicitly model the *uncertainty in the model parameters*, which is an important factor in the predictive uncertainty. Intuitively, if a prediction varies significantly under different parameter settings *all of which fit the training data well*, we must be uncertain in that prediction. Bayesian inference can be applied to neural networks, which empirically results in better calibrated predictive uncertainties (e.g. Blundell et al., 2015; Louizos and Welling, 2017).

2.2.2 Bayesian neural networks

A *Bayesian neural network* (BNN, MacKay, 1992b; Neal, 1994) is a neural network with a prior distribution on its parameters, where instead of *optimizing* we perform *inference* to compute the posterior distribution over the parameters given the training data. Conceptually, BNNs are an attractive idea, because they promise automatic regularization, principled model selection and comparison, principled transfer learning and active learning, as well as calibrated predictive uncertainties.

Unfortunately, neural networks present a set of challenges for Bayesian inference methods. The individual weights are meaningless, making informative priors difficult to define. The parameter space is high-dimensional (models used in practice can have *millions* of parameters), and there are complex dependencies and symmetries among the parameters, which means that posteriors are high-dimensional, complex and multi-modal. Various approximate inference methods have been explored over the years:

- **Laplace approximation** (MacKay, 1992a). However, even fitting a simple multivariate Gaussian posterior is expensive, and we need to use approximations to make it feasible (Kirkpatrick et al., 2017; Ritter et al., 2018).
- **Stochastic variational inference** (SVI, Graves, 2011; Blundell et al., 2015; Zhang et al., 2017). Arguably the most active area, but methods are tied to the expressivity of the variational family. Simple variational families (e.g. a diagonal Gaussian) are commonly used, more expressive parametrizations have been proposed (Louizos and Welling, 2016, 2017), but it is not clear how faithfully the distributions approximate

the true Bayesian posterior.

- **Markov chain Monte Carlo** (MCMC, Neal, 1994). Asymptotically exact, but expensive, and hence only feasible for small neural networks and datasets. There is work on stochastic sampling methods that allow using larger datasets (Welling and Teh, 2011; Chen et al., 2014).
- **Assumed Density Filtering** (ADF, Jylänki et al., 2014) and **Expectation Propagation** (EP, Soudry et al., 2014; Hernández-Lobato and Adams, 2015). ADF can be inaccurate due to its inability to iteratively refine the approximation, while the memory requirements of EP scale linearly with the number of datapoints, limiting use to small datasets. Li et al. (2015) propose a stochastic version of EP, but memory requirements are lifted at the cost of a reduced approximation quality.

In summary, we do not have a method that could perform *accurate* inference for *large neural networks* on *large datasets*, which keeps this an important research direction.

2.2.3 Hybrid Bayesian models

Prior work has shown that some benefits of Bayesian treatment might be achieved by only performing Bayesian inference for the *last* layer of a neural network, fixing the rest of the parameters to their MAP approximation. The setup might also be interpreted as using the neural network to learn a set of basis functions for Bayesian logistic regression[2]. The approach makes intuitive sense: we treat earlier layers as defining a "feature extractor", and only take uncertainty into account when making decisions based on the extracted features.

Snoek et al. (2015) successfully use a hybrid model of this sort as a surrogate model for Bayesian optimization, replacing the more traditional but less scalable Gaussian Processes model. Bauer et al. (2017) apply a hybrid model to few-shot learning: the posterior of the Bayesian logistic regression layer is used as a prior when adapting the model to novel tasks. Finally, Azizzadenesheli et al. (2018) use the same approach in a reinforcement learning setting to parametrize a deep *Bayesian* Q-network that predicts the reward, enabling principled exploration via Thompson sampling (Russo et al., 2018).

The method is appealing as it allows us to use the well-studied, accurate methods for Bayesian inference in a *linear* model, while minimizing the computational overhead by fitting a point estimate for the vast majority of the parameters.

[2]Other Bayesian models can be used (Bradshaw et al. (2017) use a Gaussian Process), but we restrict ourselves to Bayesian logistic regression in this work.

2.2.4 Adversarial examples

Adversarial examples are points for which a model outputs confident, *wrong* predictions, and which are produced via small, *imperceptible* perturbations of correctly classified points (Szegedy et al., 2014; Goodfellow et al., 2014b). Adversarial examples can reliably be found for a variety of neural networks (Moosavi-Dezfooli et al., 2016), with some points being general enough to transfer *across* models (Moosavi-Dezfooli et al., 2017). The adversarial phenomenon has recently attracted a keen interest of the community, and numerous methods have been developed for "attacking" the models, i.e. generating adversarial examples, and "defending" the models, i.e. making them robust to adversarial inputs (Yuan et al., 2017).

The adversarial example phenomenon is worrisome from the AI safety and security perspective (Kurakin et al., 2017; Eykholt et al., 2018). More generally, the existence of adversarial examples suggests that current models and training methods fail to (a) generalize in an expected way, and (b) capture regions in the input space for which predictions must be uncertain. While (a) raises questions about whether the models we are fitting have the right inductive bias for the tasks of interest, (b) suggests an interpretation of adversarial examples as *calibration errors*, which could be remedied through careful handling of uncertainty.

We focus on aspect (b) in this work: instead of measuring calibration on validation/test data, we run adversarial optimization to find points in the input space that the model is *most* miscalibrated on. In other words, adversarial optimization is used as a particularly harsh critic of model calibration, one that specifically looks for the parts of the input space with underestimated predictive uncertainty.

2.2.5 Bayesian inference and adversarial robustness

Adversarial robustness of Bayesian neural networks and non-Bayesian uncertainty estimation methods has been studied before. Rawat et al. (2017) evaluate state-of-the-art Bayesian neural networks, and observe increased predictive uncertainty on adversarial examples, but note that the quality of said predictive uncertainty varies across inference methods. Li and Gal (2017) use adversarial examples as a benchmark for a novel neural network loss designed to provide cheap predictive uncertainty estimates, and also observe increased uncertainty on adversarial inputs. Bradshaw et al. (2017) make similar observations for a *hybrid* neural network/Gaussian Process model (similar to the one we use in Section 2.2.3). Finally, Gal and Smith (2018) discuss *idealized* neural networks that would have no adversarial examples, arguing that the Bayesian approach is vital for implementing such models.

Note that the motivation for using Bayesian methods in this line of work is to *detect* adversarial examples: we hope that the predictions will become *uncertain*, indicating the lack of data and/or inductive bias in the model to justify making a confident prediction.

We believe adversarial examples can only be *eliminated* by modeling the right symmetries and invariances in the data, which is tangential to uncertainty estimation. Imposing such symmetries through architecture design has been studied before (Weiler et al., 2018). In the Bayesian setting, it is also possible to express the known symmetries and invariances using the *prior*, but it is difficult to do so for the non-interpretable neural network parameters.

When it comes to *linear* models, in their seminal paper Goodfellow et al. (2014b) demonstrate adversarial vulnerability of such models, and argue that "Linear models lack the capacity to resist adversarial perturbation". Moreover, the authors conjecture that adversarial examples in neural networks are caused by the *locally linear* nature of these models. Experiments performed by Tabacof and Valle (2015) show that the issue is more complex, providing examples of problems where linear models are *less* susceptible to adversarial attacks than deep models. Finally, Tanay and Griffin (2016) present a theoretical analysis of adversarial examples in linear models, pointing to *overfitting* as one of the underlying causes of this phenomenon.

In summary, while there is evidence that Bayesian treatment improves adversarial robustness of neural networks, the difficulty associated with the probabilistic treatment of such models prevents us from making strong conclusions. To the best of our knowledge, adversarial robustness of Bayesian *linear* models has not been studied before, which motivates our work.

2.3 Methods

We now introduce the methods used in this chapter, including the model, the inference methods, and the adversarial optimization methods.

2.3.1 Bayesian logistic regression

We use the logistic regression model as a linear classifier:

$$p(y = 1 \mid \mathbf{x}; \theta) = f(\mathbf{x}; \theta) = \sigma\Big(\mathbf{w}^\top \mathbf{x} + b\Big) \tag{2.1}$$

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \tag{2.2}$$

where $\theta = \{\mathbf{w}, b\}$ are the learnable parameters of the model, σ is the logistic sigmoid function, and $y \in \{0, 1\}$ To simplify the notation in the rest of the chapter, we include b in $\mathbf{w}$, as the same effect can be achieved by adding a constant feature to $\mathbf{x}$:

$$\hat{\mathbf{w}}^\top \hat{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} + b, \tag{2.3}$$

where $\hat{\mathbf{w}} = [\mathbf{w} \quad b]^\top$ and $\hat{\mathbf{x}} = [\mathbf{x} \quad 1]^\top$. The log likelihood given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ is then defined as:

$$\begin{aligned} p(\mathcal{D} \mid \mathbf{w}) &= \prod_{i=1}^{N} p(y = y_i \mid \mathbf{x}_i; \mathbf{w}) && (2.4) \\ &= \prod_{i=1}^{N} \Big[y_i f(\mathbf{x}_i; \mathbf{w}) + (1 - y_i)(1 - f(\mathbf{x}_i; \mathbf{w})) \Big]. && (2.5) \end{aligned}$$

Together with a prior $p(\mathbf{w})$ specified by the practitioner, the likelihood defines the posterior, which we know up to a normalizing constant:

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \mathbf{w}) p(\mathbf{w}) \tag{2.6}$$

Given the posterior, we *marginalize* over the parameters to compute the probabilistic prediction:

$$p(y = 1 | \mathbf{x}, \mathcal{D}) = \int p(y = 1 \mid \mathbf{x}; \mathbf{w}) p(\mathbf{w} \mid \mathcal{D}) \, \mathrm{d}\mathbf{w} \tag{2.7}$$

In general, the integral in Eq. (2.7) is intractable. If we can *sample* from the posterior (e.g. using MCMC methods, as discussed in Section 2.3.4), we can compute a Monte Carlo estimate of the predictive distribution:

$$p(y = 1 | \mathbf{x}, \mathcal{D}) \approx \frac{1}{M} \sum_{i=1}^{M} p(y = 1 \mid \mathbf{x}; \mathbf{w}_i), \quad \text{where} \quad \mathbf{w}_i \sim p(\mathbf{w} \mid \mathcal{D}). \tag{2.8}$$

When we approximate $p(\mathbf{w} \mid \mathcal{D})$ with a Gaussian distribution (e.g. when using variational or Laplace approximations from Sections 2.3.2 and 2.3.3), we can use the *probit approximation* (Murphy, 2012, Section 8.4):

$$\begin{aligned} p(y = 1 | \mathbf{x}, \mathcal{D}) &\approx \sigma\big(\kappa(\sigma_a^2)\mu_a\big) && (2.9) \\ \mu_a &= \mathbf{m}^\top \mathbf{x} && (2.10) \\ \sigma_a^2 &= \mathbf{x}^\top \mathbf{V} \mathbf{x} && (2.11) \\ \kappa(\sigma_a^2) &= \Big(1 + \frac{\pi}{8}\sigma_a^2\Big)^{-\frac{1}{2}}, && (2.12) \end{aligned}$$

where $\mathbf{m}$ and $\mathbf{V}$ are the mean and the covariance of the posterior approximation:

$$p(\mathbf{w} \mid \mathcal{D}) \approx \mathcal{N}(\mathbf{w} \mid \mathbf{m}, \mathbf{V}). \tag{2.13}$$

Alternatively, we can find the mode $\mathbf{w}^*$ of the posterior by *optimizing*, i.e. compute the *maximum a posteriori estimate* (MAP), in which case the predictive distribution simplifies to:

$$p(y=1|\mathbf{x},\mathcal{D}) \approx p(y=1\,|\,\mathbf{x};\mathbf{w}^*), \quad \text{where} \quad \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}\,|\,\mathcal{D}). \tag{2.14}$$

When using a uniform, improper prior, we arrive at the *maximum likelihood estimate* (MLE). The logistic regression posterior is known to be *log-concave* when using a Gaussian prior, hence if we minimize the negative log posterior then the optimization surface is *convex*. Second-order gradient optimizers can quickly find the global minimum in this case. We use the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS, Shanno, 1985) optimizer in our experiments.

2.3.2 Variational inference

The *variational* method for inference works by defining a parametric distribution (*a variational family*, e.g. a Gaussian parametrized by its mean and covariance), and optimizing over the parameters to minimize a distance (or, more precisely, a *divergence*) between the parametrized distribution and the posterior. The *Kullback–Leibler (KL)* divergence is a common choice for the distance measure:

$$\begin{aligned} D_{\mathrm{KL}}(q_\theta(\mathbf{w}) \,\|\, p(\mathbf{w}\,|\,\mathcal{D})) &= \mathbb{E}_{q_\theta}\left[\log \frac{q_\theta(\mathbf{w})}{p(\mathbf{w}\,|\,\mathcal{D})}\right] &(2.15)\\ &= \mathbb{E}_{q_\theta}\left[\log \frac{q_\theta(\mathbf{w})}{p(\mathbf{w},\mathcal{D})}\right] + \log p(\mathcal{D}), &(2.16)\end{aligned}$$

where q_θ is the variational distribution with parameters θ. Unfortunately, we can not evaluate this divergence, as the marginal likelihood (or *model evidence*) $\log p(\mathcal{D})$ is intractable. Instead, we rewrite Eq. (2.16) as

$$\log p(\mathcal{D}) = D_{\mathrm{KL}}(q_\theta(\mathbf{w}) \,\|\, p(\mathbf{w}\,|\,\mathcal{D})) + \underbrace{\mathbb{E}_{q_\theta}\left[\log \frac{p(\mathbf{w},\mathcal{D})}{q_\theta(\mathbf{w})}\right]}_{ELBO}, \tag{2.17}$$

where *ELBO* is the *evidence lower bound*, as $ELBO \leq \log p(\mathcal{D})$ due to the KL divergence always being non-negative. As $\log p(\mathcal{D})$ is fixed with respect to the variational parameters θ, instead of minimizing the KL divergence we can maximize the *ELBO*, or equivalently minimize the negative *ELBO*:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, ELBO(\theta) = \underset{\theta}{\operatorname{argmin}} -ELBO(\theta). \tag{2.18}$$

The *ELBO* itself consists of three terms: the negative entropy, the cross-entropy and the log-likelihood expectation:

$$\mathit{ELBO} = \underbrace{E_q\left[\log q(\mathbf{w})\right]}_{\text{neg. entropy}} - \underbrace{E_q\left[\log p(\mathcal{D}|\mathbf{w})\right]}_{\text{log-likelihood exp.}} - \underbrace{E_q\left[\log p(\mathbf{w})\right]}_{\text{cross-entropy}}. \tag{2.19}$$

For a Gaussian variational family and a Gaussian prior, both the negative entropy and the cross-entropy terms have easy-to-evaluate analytical expressions. The log-likelihood expectation does not have an analytical expression, but a Monte Carlo approximation is typically sufficient to obtain useful learning signal. We can further avoid evaluating the log likelihood on the whole dataset by using mini-batch training, resulting in *stochastic* variational inference (SVI, Hoffman et al., 2013; Ranganath et al., 2014). For a given batch $(y^{(i)}, \mathbf{x}^{(i)})_{i=1}^{M}$:

$$E_q\left[\log p(\mathcal{D}|\mathbf{w})\right] \approx \frac{N}{M}\sum_{i=1}^{M}\log p(y^{(i)}|\mathbf{x}^{(i)};\mathbf{w}); \quad \mathbf{w}\sim q(\mathbf{w}). \tag{2.20}$$

To reduce the variance of this estimator, we can sample a different weight setting *per datapoint* in the batch:

$$E_q\left[\log p(\mathcal{D}|\mathbf{w})\right] \approx \frac{N}{M}\sum_{i=1}^{M}\log p(y^{(i)}|\mathbf{x}^{(i)};\mathbf{w}^{(i)}); \quad \mathbf{w}^{(i)}\sim q(\mathbf{w}). \tag{2.21}$$

Eq. (2.21) could be implemented naturally and efficiently using the *local reparametrization trick* (Kingma et al., 2015). The trick involves sampling the activations directly (which for a Gaussian variational family would also be Gaussian), as opposed to sampling the weights and computing the corresponding activations.

2.3.3 Laplace approximation

Laplace approximation (MacKay, 2003, Chapter 27) fits a Gaussian to the posterior by matching the curvature of the posterior at a mode. We can find a posterior mode by optimizing, and then fit the Gaussian covariance by matching the curvature of the log posterior at that mode. In practice, this means computing the MAP estimate $\mathbf{w}^*$, and using the inverse of a Hessian of negative log posterior at $\mathbf{w}^*$ as the covariance:

$$\mathcal{L}(\mathbf{w}) = -\log\big[p(\mathcal{D}\,|\,\mathbf{w})p(\mathbf{w})\big], \quad \mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}}\mathcal{L}(\mathbf{w}), \quad H = \nabla\nabla\mathcal{L}(\mathbf{w}^*), \tag{2.22}$$

$$p(\mathbf{w}\,|\,\mathcal{D}) \approx q(\mathbf{w}\,|\,\mathcal{D}) = \mathcal{N}\big(\mathbf{w}\,|\,\mathbf{w}^*, H^{-1}\big) \tag{2.23}$$

We refer the reader to MacKay (2003, Chapter 27) for further details and derivations.

The method, while conceptually simple, scales poorly when implemented naively: computing and inverting the Hessian (the size of which scales quadratically with the number of parameters) is problematic for high-dimensional parameter spaces. Kirkpatrick et al. (2017) use a *diagonal approximation* to the Hessian in order to improve scalability, making a naive assumption that all the parameters are independent. Ritter et al. (2018) use a more accurate approximation of the Hessian based on *Kronecker factorization*, relaxing the independence assumption by modeling the diagonal *blocks* of the Hessian.

2.3.4 Markov chain Monte Carlo

Instead of fitting a parametric distribution as in variational/Laplace approximations, methods based on *Monte Carlo* sampling allow us to sample from the target distribution *directly*. *Markov Chain* Monte Carlo (MCMC) methods in particular are able to sample from complex, high-dimensional distributions (e.g. MacKay, 2003, Chapter 29). The *Metropolis* algorithm, a special case of the *Metropolis–Hastings* algorithm, is a classic MCMC method that works by iteratively *proposing* transitions $\mathbf{w}_i \to \hat{\mathbf{w}}$ in the parameter space using a symmetric *proposal distribution* (e.g. a Gaussian) conditioned on the current position $\mathbf{w}_i$:

$$\hat{\mathbf{w}} \sim q(\hat{\mathbf{w}} \,|\, \mathbf{w}_i). \tag{2.24}$$

The transition is then stochastically *accepted* (we transition to the proposed location) or *rejected* (we stay at the current location), where the probability of acceptance depends on the posterior density ratio between $\hat{\mathbf{w}}$ and $\mathbf{w}_i$:

$$a = \frac{p(\hat{\mathbf{w}} \,|\, \mathcal{D})}{p(\mathbf{w}_i \,|\, \mathcal{D})}, \qquad u \sim \mathcal{U}(0,1), \qquad \mathbf{w}_{i+1} = \begin{cases} \hat{\mathbf{w}} & \text{if } u < a \\ \mathbf{w}_i & \text{otherwise} \end{cases}. \tag{2.25}$$

Transitions that increase the posterior probability ($p(\hat{\mathbf{w}} \,|\, \mathcal{D}) > p(\mathbf{w}_i \,|\, \mathcal{D})$) are always accepted, while ones that decrease it are *sometimes* accepted, but with low probability if the decrease is significant. The explored locations in the parameter space will *asymptotically* be distributed according to the target (posterior) distribution. In other words, as $M \to \infty$, our approximation in Eq. (2.8) will approach the true Bayesian predictive distribution.

In practice, however, we can only obtain a *finite* set of samples, which means there will be an error in our approximation of the predictive distribution. This error, unsurprisingly, depends on the number of samples: obtaining more samples reduces the error (all the way to zero in the limit of infinitely many samples), but increases the computational/memory cost. The error also depends on how well the chain *mixes*, i.e. on how *correlated* the subsequent samples are, and is reduced when the correlation decreases. For the Metropolis algorithm, the mixing is determined by the proposal distribution $q(\hat{\mathbf{w}} \mid \mathbf{w}_i)$. If it only proposes transitions close to the current location (e.g. the variance of the Gaussian proposal is small), the correlation will be high. If it proposes transitions too far away, we run the risk of rarely accepting transitions and staying at the same location, which also increases the correlation

Alternative Monte Carlo methods have been developed that aim to de-correlate the samples, in order to reduce the estimation error without using many samples. In particular, *Hamiltonian Monte Carlo* (HMC, Duane et al., 1987) uses the gradient of the target density to make *multiple* steps for each transition. However, HMC is itself sensitive to the number of steps L it takes for each transition: if L is too small the subsequent samples have higher correlation, but if L is too large we waste computation taking a lot of steps to obtain a single sample. The No-U-Turn sampler (NUTS, Hoffman and Gelman, 2014) that we use in our experiments tunes the L parameter automatically during sampling, making it possible to run HMC with little to no hyperparameter tuning.

2.3.5 Bootstrap

Bootstrap is a baseline approach for obtaining predictive uncertainty that is easy to implement for an arbitrary predictor class. The input dataset is re-sampled with replacement N times, and N models are trained independently, each on a different re-sampled dataset. In the end, we obtain a set of weight vectors that defines a predictive ensemble, which allows for a direct comparison to an ensemble defined by MCMC samples. As in MCMC, the prediction of the ensemble is computed by averaging the predictions of all the ensemble members, see Eq. (2.8).

In certain circumstances bootstrap can be interpreted as a non-parametric approximation to the Bayesian posterior with a non-informative prior (Hastie et al., 2001a, §8.4). However, it is not clear how general this correspondence is, and what properties this approximation has.

2.3.6 Adaptive basis logistic regression

In its abstract form, a neural network model is defined as a sequence of M transformations L_i:

$$f(\mathbf{x}) = L_M \circ L_{M-1} \circ \cdots \circ L_1(\mathbf{x}) \tag{2.26}$$

Each L_i is typically a linear transformation followed by an elementwise non-linearity.

An alternative view of the same model is to consider the first $M-1$ layers to be learning a neural *basis* for the last layer L_M (Murphy, 2012, Chapter 16). As explored by Snoek et al. (2015) and Bauer et al. (2017), we consider a *hybrid* model, where said last linear layer is replaced with Bayesian logistic regression.

Depending on the interpretation, we can consider the proposed hybrid model to either learn a neural basis for a Bayesian linear model, or a Bayesian neural network where we only marginalize over the last layer, using a MAP approximation for other parameters. The latter interpretation leads to a simple extension: marginalizing over the last K layers, and training the first $M-K$ layers by (regularized) maximum likelihood.

Following Snoek et al. (2015) and Bauer et al. (2017), in our experiments we train the model sequentially. We first fit the representation layers, simultaneously fitting a point-estimate for the Bayesian layers. Following this we *fix* the learned representation layers, and perform inference for the Bayesian layers only, discarding the point-estimate fit in the earlier stage. In some cases, e.g. when we use variational inference for the Bayesian layers, we could train the model end-to-end using the corresponding objective (e.g. variational ELBO, Bradshaw et al., 2017), but we do not explore this.

2.3.7 Adversarial optimization

The goal of adversarial optimization is to find *valid* points that an otherwise accurate model confidently misclassifies. By *valid* we define points that lie on the data manifold (or in the data distribution). More precisely, we define a *constrained* adversarial optimization task for a given predictor f and a true class y:

$$\begin{aligned} \text{minimize} \quad \mathcal{L}_{adv}(\mathbf{x}) &= \mathrm{BCE}(f(\mathbf{x}), 1-y) && (2.27)\\ &= -y\log z - (1-y)\log(1-z) && (2.28)\\ \text{subject to} \quad \mathbf{x} & \textit{ is on manifold} && (2.29)\end{aligned}$$

In other words, we try to find the point $\mathbf{x}$ that maximizes the prediction for the wrong class, constraining $\mathbf{x}$ to stay on the data manifold.

The precise definition of "on manifold" depends on the problem at hand. In this work, we consider two scenarios: a spherical manifold with a given radius r, and a manifold of a particular natural image dataset. For the former, trivially

$$\mathbf{x} \textit{ is on manifold} \iff \|\mathbf{x}\| = r. \tag{2.30}$$

Characterizing the image manifold for the latter is tricky, but a common proxy used is to

Algorithm 2.1 Projected gradient descent for adversarial optimization. Takes a differentiable adversarial loss function $\mathcal{L}_{adv}$, step size α, projection function g, initial point $\mathbf{x}_0$, number of steps N. Outputs x_N, the point at the final step.

```
1: for $t \leftarrow 0, N$ do
2:     $\hat{\mathbf{x}} \leftarrow \mathbf{x}_t - \alpha \nabla_{\mathbf{x}_t} \mathcal{L}_{adv}(\mathbf{x}_t)$        ▷ Gradient descent step
3:     $\mathbf{x}_{t+1} \leftarrow g(\hat{\mathbf{x}})$        ▷ Projection step
```

keep $\mathbf{x}$ a small ℓ_∞ norm (or *max* norm) away from some valid image $\mathbf{x}_0$:

$$\mathbf{x} \textit{ is on manifold} \iff \|\mathbf{x} - \mathbf{x}_0\|_\infty < \epsilon \tag{2.31}$$

$$\iff \max |\mathbf{x} - \mathbf{x}_0| < \epsilon, \tag{2.32}$$

where ϵ is the chosen threshold, and $\mathbf{x}_0$ is typically set to the image we initialize the optimization from. In other words, $\mathbf{x}$ is considered valid if all its pixel values are *at most* ϵ away from the corresponding values in $\mathbf{x}_0$, for some small ϵ.

In line with Gilmer et al. (2018) and Madry et al. (2018), we solve the constrained optimization problem using *Projected Gradient Descent* (PGD). PGD augments any standard gradient-based optimizer with an additional projection step, where the current point is projected to a point that satisfies the constraint.

We use the Algorithm 2.1 to apply PGD to adversarial optimization, where $\mathcal{L}_{adv}$ is the adversarial loss from Eq. (2.28), and g is a function that projects the argument to a point *on* the manifold. In particular, if we want to keep the point on a sphere with radius r, we use the following g:

$$g(\hat{\mathbf{x}}) = r \frac{\hat{\mathbf{x}}}{\|\hat{\mathbf{x}}\|} \tag{2.33}$$

To keep the point within an ℓ_∞ norm of less than ϵ from some point $\mathbf{x}_0$:

$$g(\hat{\mathbf{x}}) = \min(\mathbf{x}_0 + \epsilon, \max(\hat{\mathbf{x}}, \mathbf{x}_0 - \epsilon)), \tag{2.34}$$

where min and max are applied elementwise.

In our experiments we use the standard Stochastic Gradient Descent (SGD) update as given in Algorithm 2.1, in some cases augmenting it with momentum and annealing the step size α. The gradient step is not aware of the projection step, however, so accelerated gradient descent methods have to be used with care. The literature on accelerating PGD is scarce, but in practice we find that momentum works well and speeds up convergence. Adaptive gradient methods like Adam (Kingma and Ba, 2015), on the other hand, performed poorly in our preliminary experiments when used with PGD.

2.3.8 Adversarial loss smoothing

Adversarial optimization requires finding gradients of the BCE loss with respect to the inputs, i.e. $\nabla_{\mathbf{x}_t}\mathcal{L}_{adv}(\mathbf{x}_t)$ in Algorithm 2.1. This, in turn, involves finding the gradient of the prediction with respect to the input, $\nabla_{\mathbf{x}} f_\theta(\mathbf{x})$. When f_θ is a logistic regression model:

$$\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) = \nabla_{\mathbf{x}}\sigma\left(\mathbf{w}^\top\mathbf{x}\right) \tag{2.35}$$

$$= \nabla_{\mathbf{x}} z \frac{\partial\sigma(z)}{\partial z} \quad \text{where} \quad z = \mathbf{w}^\top\mathbf{x}. \tag{2.36}$$

The problem lies with the derivative of the sigmoid in Eq. (2.36). For large magnitudes of z the sigmoid saturates, and its derivative vanishes, i.e. becomes exactly zero due to finite precision. This is why in practice the BCE loss is re-written as follows:

$$\mathrm{BCE}(f(\mathbf{x}), y) = -y\log f_\theta(\mathbf{x}) - (1-y)\log(1 - f_\theta(\mathbf{x})) \tag{2.37}$$

$$= z - yz + \log(1 + \exp(-z)) \tag{2.38}$$

$$= \max(z, 0) - yz + \log(1 + \exp(-|z|)) \tag{2.39}$$

where $z = \mathbf{w}^\top\mathbf{x}$ are the logits, and $\log(1 + \exp(-|z|))$ has an implementation that is numerically stable even for large $|z|$ values (for example, `numpy.log1p`).

In this work, however, f will often be an *ensemble* of logistic regression models, rather than a single model, where the prediction is computed as:

$$f(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M}\sigma\left(\mathbf{w}_i{}^\top\mathbf{x}\right), \tag{2.40}$$

where M is the number of models in the ensemble.

We are unable to use the Eqs. (2.37) to (2.39) trick in this case. Instead, we note that re-scaling the input to the sigmoid will not change the sign of the sigmoid's derivative, but will change its magnitude. Hence, during adversarial optimization, we use an alternative predictor:

$$\hat{f}(\mathbf{x}, \tau) = \frac{1}{M}\sum_{i=1}^{M}\sigma\left(\tau\mathbf{w}_i{}^\top\mathbf{x}\right), \tag{2.41}$$

where τ is often referred to as *temperature*. Fig. 2.2 demonstrates the effect of the temperature on the predictive surface of a random ensemble. We see that at lower temperatures the predictive surface is "smoothed out", and its derivative can guide the optimizer towards the extrema. To make use of this during optimization, we start with $\tau = 0$ and continuously

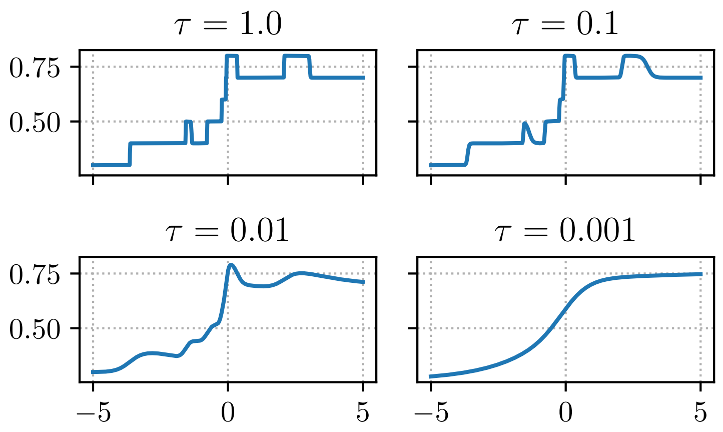


Figure 2.2: Temperature loss smoothing. Plots of $\hat{f}(\mathbf{x}, \tau)$ as defined in Eq. (2.41) for an random ensemble in 1D, varying the value of τ. Ensemble size $M = 10$, a weight and a bias (joined into $\mathbf{w}_i$) are sampled from $\mathcal{N}(\mathbf{0}, 10^6\mathbb{I})$, which simulates the case where ensemble members are making extremely confident predictions. Decreasing the temperature τ "smooths" the function, providing a useful derivative for optimization.

increase it to 1 according to a cosine schedule (Loshchilov and Hutter, 2017):

$$\tau(t) = \tau_{end} + (\tau_{start} - \tau_{end})\frac{1}{2}\left(1 + \cos\left(\frac{\pi t}{N}\right)\right), \tag{2.42}$$

where t is the optimization step, τ_{start} and τ_{end} are the temperatures at the start and at the end of the optimization run ($\tau_{start} = 0$ and $\tau_{end} = 1$ in our case), and N is the total number of optimization steps. In other words, during the course of optimization we are continuously morphing the optimization surface from the "smooth" approximation of the loss surface $\hat{f}$ to the *actual* loss surface f.

Empirically this method reliably finds calibration errors for a wide class of learning methods. The method is an instance of *continuation methods* (Bengio et al., 2009) for minimizing complex, potentially non-convex losses using similar "smoothed"-to-original transformations of the optimization surface.

The vanishing gradient problem is an example of an adversarial defense by *obfuscated gradients* (Athalye et al., 2018), albeit created unintentionally in this case. Learning an approximation to the ensemble that has well-behaving gradients, amongst other methods for attacking obfuscated gradient-based adversarial defenses discussed by Athalye et al., could provide an alternative to the temperature annealing method above, but we leave this for future work.

2.4 Experimental setup

In this chapter, we run experiments to evaluate the uncertainties of the Bayesian logistic regression model using various inference methods. In particular, we are looking for predictions that would be accurate and confident on the validation points, and accurate *or uncertain* on adversarial points. In other words, we would like the model to assign high probability $p(y \mid \mathbf{x})$ to the true labels y for both validation and adversarial $(\mathbf{x}, y)$ pairs. The model is able to maximize $p(y \mid \mathbf{x})$ for $\mathbf{x}$ that it knows it is likely to be wrong about by increasing the entropy of $p(y \mid \mathbf{x})$, i.e. by becoming more uncertain.

2.4.1 Datasets

In their work Gilmer et al. (2018) introduce a synthetic *concentric spheres* dataset that is particularly useful for our purpose. In this dataset the data lie on two concentric spheres (or *hyperspheres*), with one sphere having a smaller radius than the other. The class of a datapoint is determined by which sphere it lies on: $y = 0$ for points on the inner sphere, $y = 1$ for points on the outer sphere. We plot an example of the concentric spheres dataset in 2D in Fig. 2.3.

Such dataset is easy to generate for any number of dimensions D by sampling data points from an D-dimensional spherical Gaussian, and projecting each point onto a sphere with a class-dependent radius:

$$\mathbf{x} = r\frac{\mathbf{z}}{\|\mathbf{z}\|}, \qquad r = \begin{cases} r_1 & \text{if } y = 0 \\ r_2 & \text{if } y = 1 \end{cases}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_D). \tag{2.43}$$

As a result $\mathbf{x}$ will be uniformly distributed on the surface of a sphere with radius r. To see that this is the case, first note that if vectors $\mathbf{z}$ are sampled from a spherical Gaussian distribution then

$$\begin{aligned} \log p(\mathbf{z}) &= \log \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma\mathbb{I}) && (2.44) \\ &= -\frac{1}{2}\mathbf{z}^\top \frac{1}{\sigma}\mathbb{I}\mathbf{z} + \text{const.} && (2.45) \\ &= -\frac{1}{2\sigma}\|\mathbf{z}\|^2 + \text{const.} && (2.46) \end{aligned}$$

We can see that the density of a spherical Gaussian only depends on the *norm* of the vector, and is invariant to the “direction” ($\frac{\mathbf{z}}{\|\mathbf{z}\|}$) of the vector. Intuitively, this means that the distribution over the Gaussian-distributed vectors *after normalization* (i.e. the distribution on a sphere) is indeed uniform. To show this more rigorously, we use the sum rule of probability to find the density[3] of $\mathbf{x}$:

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \, \mathrm{d}\mathbf{z} \tag{2.47}$$

$$= \int \delta\left(\mathbf{x} - r\frac{\mathbf{z}}{\|\mathbf{z}\|}\right) \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma\mathbb{I}) \, \mathrm{d}\mathbf{z} \tag{2.48}$$

$$= \int_{\left\{\mathbf{z} \,:\, \mathbf{x} = r\frac{\mathbf{z}}{\|\mathbf{z}\|}\right\}} \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma\mathbb{I}) \, \mathrm{d}\mathbf{z}. \tag{2.49}$$

In other words, it is the integral over all the vectors $\mathbf{z}$ that produce $\mathbf{x}$ when normalized. We can find the same set of vectors by integrating over the (positive) lengths c for a fixed direction $\frac{\mathbf{z}}{\|\mathbf{z}\|}$:

$$p(\mathbf{x}) = \int_0^\infty \mathcal{N}\left(c\frac{\mathbf{z}}{\|\mathbf{z}\|} \mid \mathbf{0}, \sigma\mathbb{I}\right) \mathrm{d}c \tag{2.50}$$

$$= \int_0^\infty f(c) \, \mathrm{d}c \tag{2.51}$$

$$= \text{const.}, \tag{2.52}$$

where we use Eq. (2.46) and the fact that $\left\|c\frac{\mathbf{z}}{\|\mathbf{z}\|}\right\| = c$ to go from Eq. (2.50) to Eq. (2.51). This completes the proof. □

The sphere dataset is defined for an arbitrary dimension D. Moreover, we always know the set of true decision boundaries: any hypersphere with a radius r^* s.t. $r_1 < r^* < r_2$ perfectly separates the data. The *maximum margin* decision boundary is a hypersphere with a radius $r^* = (r_1 + r_2)/2$.

Gilmer et al. (2018) demonstrate that for D as high as 500 one could learn an extremely accurate (error rate less than 10^{-6}) neural network for this dataset. At the same time, we can use adversarial optimization to find points *on the data manifold* that the same highly accurate model misclassifies with high confidence. Note that the true data manifold is well-defined for this problem: it is a union of the two spheres. The existence of such errors and the simplicity of the set-up make it particularly interesting for studying adversarial robustness.

In addition to the spheres dataset, we use the `moons` dataset from `sklearn.datasets` (Pedregosa et al., 2011) as a simple planar classification benchmark. As the non-synthetic dataset, we use the CIFAR-10 dataset (Krizhevsky, 2009). To simplify the setup, we only consider two classes from CIFAR-10 (`truck` and `car`) to define a binary classification task.

[3]Note this density is only well-defined for points $\mathbf{x}$ on the sphere with radius r ($\|\mathbf{x}\| = r$), and is *not* well-defined in the ambient space.

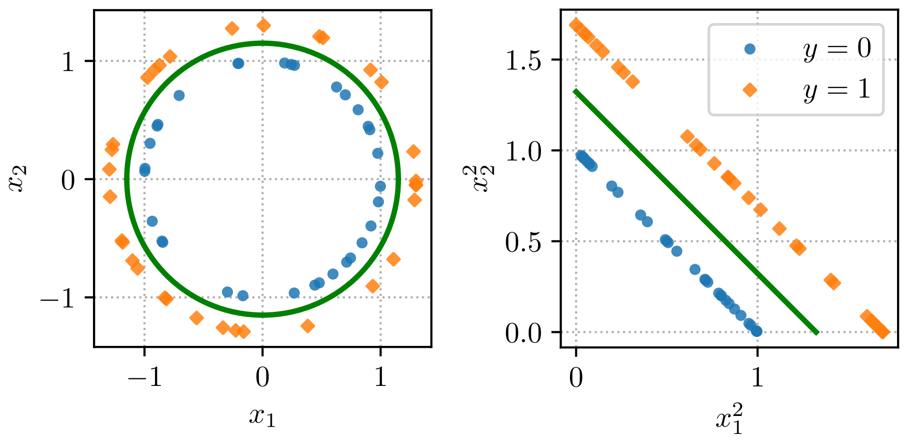


Figure 2.3: The concentric spheres dataset in 2D. **Left:** The dataset. Datapoints colored by true class. Max. margin decision boundary marked by a green line. **Right:** The same dataset with features transformed by the elementwise square function in Eq. (2.54). The data become linearly separable.

2.4.2 Model and prior

We use a logistic regression model with a bias in all of our experiments, only changing the feature space and the inference method. In all cases we standardize the logistic regression inputs to have zero mean and unit variance, using statistics computed on the training set.

For most experiments, apart from the ones that use a hierarchical prior (Section 2.5.6), we use a spherical Gaussian prior for the regression weights (including the bias):

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \,|\, \mathbf{0}, \sigma_w \mathbb{I}), \tag{2.53}$$

with $\sigma_w = 10$. We use the same prior for MAP inference, which is equivalent to using a corresponding L2 regularizer in maximum likelihood training. This wide prior is chosen to represent our lack of prior belief about the weight values, which is common in practice.

2.4.3 Handcrafted features

The concentric spheres problem could be "linearized" by applying a square operation to each data dimension:

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & \dots & x_D^2 \end{bmatrix}^\top. \tag{2.54}$$

A logistic regression in such transformed space defines a decision boundary that is a special case of a *quadric surface* in a D-dimensional space:

$$\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) = \sum_{i=1}^{D} w_i x_i^2 = -b. \tag{2.55}$$

Eq. (2.55) can parameterize a subset of quadric surfaces that includes an *ellipsoid*, obtained by keeping all w_i's positive. The target decision boundary (a sphere) is, in turn, a special case of an ellipsoid with all w_i's being positive *and equal*: $w_1 = w_2 = ... = w_D$. This explains why the elementwise transformation in Eq. (2.54) allows us to match the non-linear target decision boundary with a linear model, as visualized in Fig. 2.3.

2.4.4 Learned features

We use a simple feedforward neural network when learning features for the spheres problem. The neural network has 3 hidden layers of size 1000, and includes batch normalization (Ioffe and Szegedy, 2015). We add a linear layer followed by a sigmoid to train the representation on the spheres classification task. When training we sample new data for each batch, which means the model effectively sees 10^8 points during training (2×10^5 steps with batch size 500). The trained model achieves perfect accuracy on 10^6 validation points. For evaluation, we replace the added linear and sigmoid layers with Bayesian logistic regression, keeping the representation neural network fixed during inference.

For the image experiments, we use a Wide Residual Network by Zagoruyko and Komodakis (2016) with the "deepening" factor $l = 40$ and the "widening" factor $k = 2$. We either load a checkpoint pre-trained on ImageNet as provided by Hendrycks et al. (2019), or train on the binary image classification task with CIFAR-10 images directly, optimizing for 30 epochs with a batch size of 128.

2.5 Experiments: handcrafted features on spheres

We begin by considering the concentric sphere problem. We use the handcrafted features to turn the target decision boundary into a linear one, which allows it to be expressed by a simple logistic regression classifier.

We generate 1000 training and 10^6 validation samples on the concentric spheres with $D = 500$, as described in Section 2.4.1. Note that the training sample size is orders of magnitude smaller than the one used by Gilmer et al. (2018). Due to the linear model being much less flexible than a multi-layer neural network, we have to reduce the dataset size to replicate the same phenomenon, i.e. the model being accurate on validation points, but suffering from calibration errors on adversarial points.

Table 2.1: Results for Bayesian logistic regression with elementwise squared features. Confidence defined as $p(y_{true} \mid \mathbf{x})$. Error defined as $p(1 - y_{true} \mid \mathbf{x})$. Adversarial dataset produced by perturbing validation points as described in Section 2.3.7.

	Validation		Adversarial	
Model	Log likelihood	Confidence	Log likelihood	Max. error
MAP	−0.000	1.000	−1.829	0.964
Bootstrap	−0.000	1.000	−1.343	0.869
SVI	−0.017	0.984	−0.802	0.632
MCMC	−0.024	0.976	−0.737	0.612

Using this setup we compare the adversarial robustness of MAP, bootstrap, MCMC, and SVI. The results are shown in Table 2.1. All methods achieve perfect accuracy on the validation set (error rate $< 10^{-6}$), so the accuracy numbers are omitted. We now compare and discuss the results of the considered methods.

2.5.1 Maximum a posteriori

MAP is a *point* estimate, and as such is a crude approximation to the full Bayesian treatment.[4] While it captures a mode of the posterior, it does not capture any of the *uncertainty* around the mode. For the concentric sphere problem, while MAP is sufficient to demonstrate perfect accuracy on a large validation set, adversarial optimization finds points *on the data manifold* (i.e. on the spheres) that the same accurate model misclassifies with *more than 96% confidence.*

This replicates the results of Goodfellow et al. (2014b), who use a binary image classification task and ℓ_∞ norm-constrained adversarial optimization to demonstrate vulnerability of high-dimensional linear models. Goodfellow et al. (2014b, Section 3) conjecture that this phenomenon would only be observable in sufficiently high-dimensional problems. A nice property of the sphere problem is that it is defined for all dimensions $D \geq 2$. We have used this property and empirically confirmed the conjecture in our preliminary experiments — we only find calibration errors when the feature space is sufficiently high-dimensional. Moreover, in our setting we observe calibration errors only when the problem is sufficiently high-dimensional *and* when the amount of data is insufficient to fully restrict the linear model, which Goodfellow et al. do not discuss.

Our findings highlight that adversarial examples are not a neural network-specific issue, and that adversarial robustness must be a consideration even when deploying simple (e.g. linear) models. What is more important for this thesis, however, is that such simplified

[4]MAP is equivalent to regularized Maximum Likelihood Estimate (MLE), and does not provide a measure of posterior uncertainty generally associated with Bayesian methods (Murphy, 2012, Section 5.2.1).

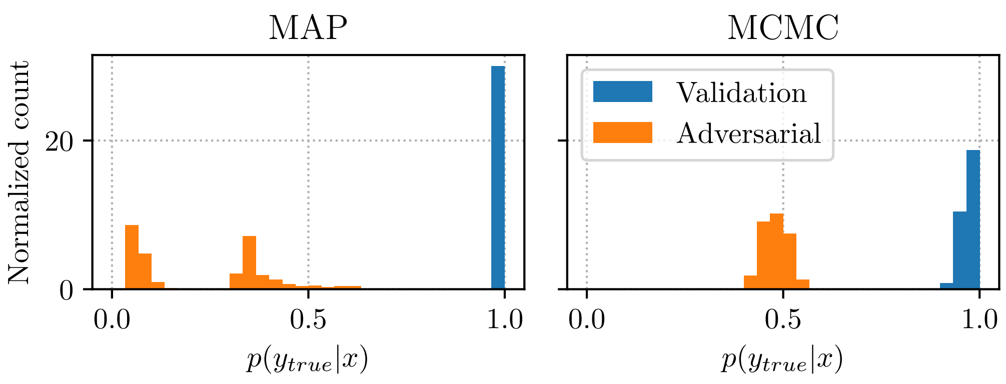


Figure 2.4: Probabilities assigned to the true class by MAP and MCMC Bayesian logistic regression with handcrafted features for validation and adversarial points. We see that while MAP makes confident mistakes ($p(y_{true} \mid x) \to 0$) for some of the adversarial points, MCMC at most becomes uncertain. At the same time, the confidence of MCMC is reduced for validation points.

setting allows us to study the phenomenon more deeply.

Given that the training dataset used is small (1000 points in a 500-dimensional space), one might argue that calibration errors in this setting could be caused by overfitting, in which case they could potentially be alleviated by tuning the regularization hyper-parameters. In practice, tuning the Gaussian prior variance (i.e. the L2 regularization coefficient) has not been fruitful: the picture is the same unless excessive regularization causes the model to underfit the training set. This contradicts the theoretical analysis of Tanay and Griffin (2016), who suggest regularization is one way to solve adversarial examples in a linear model.

Alternative regularization methods could be used. However, choosing and tuning regularization is tricky in this problem: for many sensible regularization settings the validation negative log likelihood is too small to be measured (i.e. exactly zero with finite precision), hence it is not clear what criterion to use.

2.5.2 Markov chain Monte Carlo

An MCMC method (HMC, Duane et al., 1987; Neal, 2011) shows a noticeable improvement on the MAP results. The average log probability for the adversarial points is significantly higher, and adversarial optimization is unable to uncover points which the model *confidently* misclassifies. Fig. 2.4 compares the predictions of MAP and MCMC models for validation and adversarial points. Unlike the MAP model, the MCMC ensemble never makes a confident prediction for an adversarial point.

At the same time, the MCMC ensemble is marginally less confident on the validation data. One could argue that the predictive uncertainty of MCMC is poorly calibrated: after all, the model is making *no* mistakes on 10^6 validation points, so should arguably be close-to-100% confident on these points. We argue that the confidences of MCMC are not unjustified: it so happens that the data in the concentric spheres problem are restricted to lie *exactly* on the spheres, with no additional noise present to push some points off the surface. In other words, the problem is linearly separable in the feature space. If that was *not* the case, and the training data were a sample from an underlying *noisy* process, a completely confident model could obtain an infinite validation NLL by making a *single* mistake on the validation data.

Neither the model nor the wide prior that we use make assumptions about the linear separability / lack of noise. The logistic regression model prescribes a Bernoulli sampling process for the labels, which in practice means that weight settings with *low-confidence* mistakes suffer little in terms of training NLL. We hypothesize that predictions would become more confident if we used our prior knowledge about the linear separability and considered *hard* classifiers (i.e. replaced the sigmoid likelihood with a step function), or used a prior distribution with support restricted to high-norm weights. We leave such experiments for future work.

We believe that the model parameters in the posterior are highly correlated: if we ignore normalization, the true weights are all equal, as previously discussed in Section 2.4.3. MCMC methods often struggle with correlations in the posterior, which results in slow mixing (high auto-correlation in the samples), and higher estimation error. In our experiments we use *thinning* to improve mixing: we run a longer Markov chain, but take every k-th sample, jumping over multiple transitions. This way we reduce auto-correlation, but keep the final number of samples fixed, enabling fair comparison to alternative inference methods. We use $k = 10$ in our experiments.

Note that when there are no additional constraints on the number of samples, thinning is often unnecessary and is always inefficient (Link and Eaton, 2012). There are other approaches for improving MCMC mixing in problems with highly correlated parameters, e.g. estimating a non-diagonal (dense) mass matrix for use in an HMC sampler. Using such techniques could further improve our MCMC results.

2.5.3 Bootstrap

Bootstrap is a crude approximation to the Bayesian method, but one which aims to capture some of the uncertainty in the posterior, as discussed in Section 2.3.5. It is trivial to implement and run for an arbitrary predictor class, and as such is a popular choice among

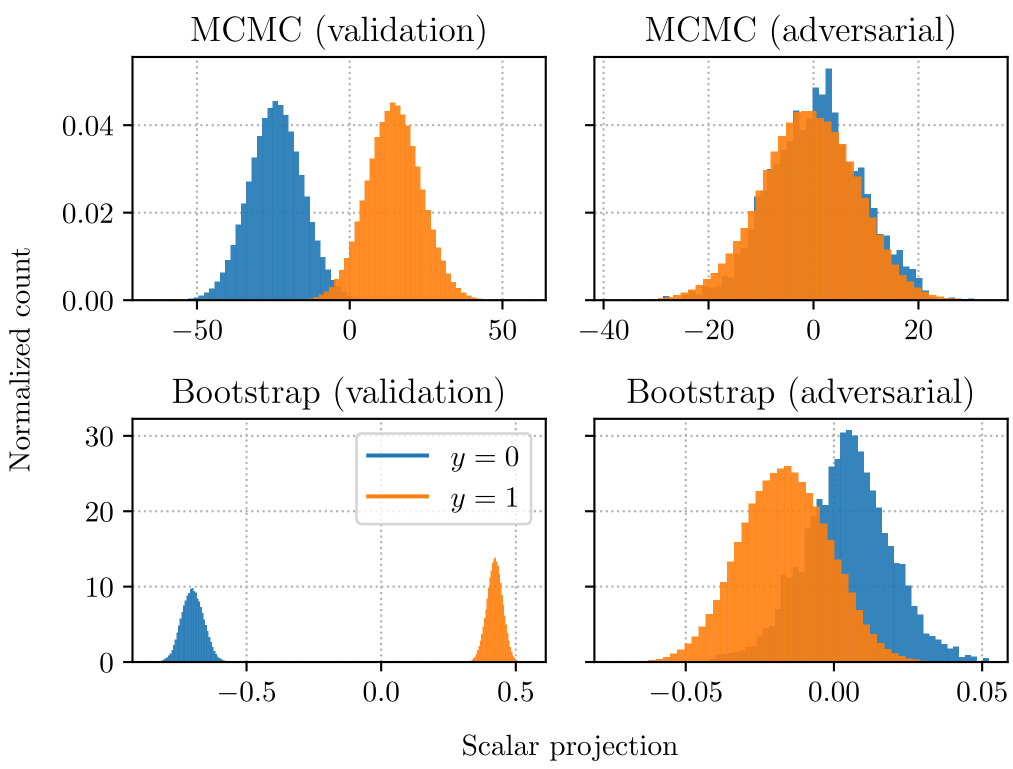


Figure 2.5: Scalar projections of weights onto validation/adversarial directions for MCMC and bootstrap. We plot $\mathbf{w}_i \cdot \hat{\mathbf{x}}_j$ values, which are the (signed) lengths of weight vector ($\mathbf{w}_i$) projections onto *direction* vectors (normalized position vectors, $\hat{\mathbf{x}}_j = \mathbf{x}_j / \|\mathbf{x}_j\|$) of either validation points or adversarial points found for the bootstrap ensemble. The colors denote the true classes of the points that we project onto. The scalar projections are proportional to the prediction logits, and so, as expected, on validation points the values are negative for $y = 0$ and positive for $y = 1$. On adversarial points, however, bootstrap places *most* of the mass on the wrong scalar projections, which explains the mistakes. At the same time, bootstrap places *some* mass on the correct values, which explains the marginally increased uncertainty.

the practitioners. Here bootstrap marginally improves upon the results of MAP, but does not reach the MCMC results: the ensemble is still making confidently wrong predictions.

To understand the nature of these mistakes, we investigate the distributions of ensemble weights along validation and adversarial directions. In particular, we compute the *scalar projections*[5] of the weights onto position vectors of (a) validation points (b) adversarial points found for the bootstrap ensemble. These values are proportional to the prediction

[5]The *scalar projection* of a vector $\mathbf{a}$ onto vector $\mathbf{b}$ is defined as $\mathbf{a}\frac{\mathbf{b}}{\|\mathbf{b}\|}$, i.e. the norm of a vector resulting from projecting $\mathbf{a}$ onto a normalized $\hat{\mathbf{b}} = \frac{\mathbf{b}}{\|\mathbf{b}\|}$, but *with a negative sign* if the projection vector has the opposite direction to $\mathbf{b}$.

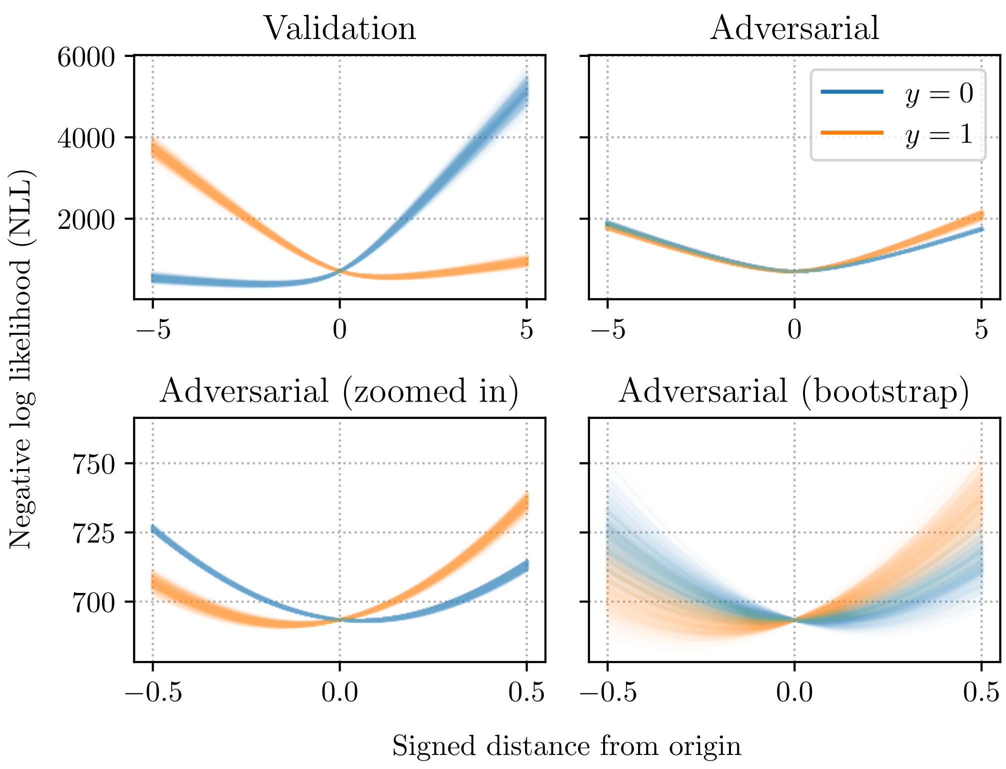


Figure 2.6: Change in training negative log likelihood (NLL) values when moving along validation/adversarial directions in weight space. We move along the position vectors of either validation points or adversarial points found for the bootstrap ensemble, and measure the training NLL (lower is better) when using the point *as the weight vector*. The colors denote the true classes of the directions that we move along, hence we expect low NLL for *negative* norms when $y = 0$, and for *positive* norms when $y = 1$. This is indeed the case for validation directions, but is *not* the case for adversarial directions: the NLL values see little change when moving along these directions, and the smallest NLL values are achieved for the *wrong* norm sign. Moreover, the norms with the smallest NLL *do not change* significantly when re-sampling the training points with replacement, as done when training with bootstrap.

logits[6], and hence we seek negative values for $y = 0$, and positive values for $y = 1$. We plot the scalar projection histograms in Fig. 2.5.

For both methods the scalar projections onto the validation points are as expected, although there is *some* overlap between the class histograms for MCMC, which explains the reduced confidence in predictions on validation data. For the adversarial points, on the other hand, MCMC places almost equal mass on both positive and negative projection values (positive

[6]More precisely, these values correspond to logits downscaled by the norms of the feature vectors. Note that in this setting even after normalization the feature vectors have high norms. As a result, large logit magnitudes (i.e. confident predictions) are achieved even by small norm weight vectors.

and negative logits, equivalently), which supports the uncertain predictions. The picture is different for bootstrap: while there is *some* mass placed on the correct projection sign, *most* of the mass is misplaced for both classes. The misplaced mass explains the mistakes made by bootstrap for these points. At the same time, the presence of *some* mass on both scalar projection signs explains the marginally increased uncertainty for these predictions.

Why exactly does bootstrap put most of the mass on the wrong weight values? To understand this, we plot the negative log likelihood (NLL) values as they change when moving along the validation/adversarial directions. This way we hope to understand what weights the training data favors in these directions. Note that while we use the data/feature space to pick directions, we move along these directions *in the weight space.* The results are shown in Fig. 2.6.

As we can see, in comparison to validation directions, the magnitude of the NLL change is small as we move along the adversarial directions. At the same time, the *wrong* class has the smallest NLL, which does not change as we re-sample the training data. In other words, it so happens that this particular finite data sample prefers the wrong class in adversarial directions, and the spread of the data is insufficient for bootstrap sampling to have a significant impact on this. The ability of MCMC to explore does *not* depend on the spread of the data, hence it can realize that neither class is *strongly* favored by the data in these directions.

2.5.4 Stochastic Variational Inference

Results of SVI with the full-covariance Gaussian variational family are comparable to MCMC results. In Fig. 2.7 we compare the samples from the SVI posterior to MCMC samples. We see that MCMC explores a bigger volume of the posterior, especially higher norm weights. A full-covariance Gaussian can be a good approximation to the posterior of logistic regression (Murphy, 2012, Section 8.4), but it can not represent such posterior *exactly*, so it is expected that some probability mass will be missed.

At the same time, the *directions* (normalized weights) are seemingly explored well by both SVI and MCMC, which likely explains why the downstream performance of the two methods is comparable. Rezende and Mohamed (2015) provide examples of challenging inference problems where *normalizing flows* (see Chapter 3) show clear improvement over simple variational families. Our own experiments with variational distributions parametrized by normalizing flows did *not* yield a noticeable improvement in the spheres problem, further suggesting that the multivariate Gaussian distribution is able to capture the most important factors of variation in the posterior.

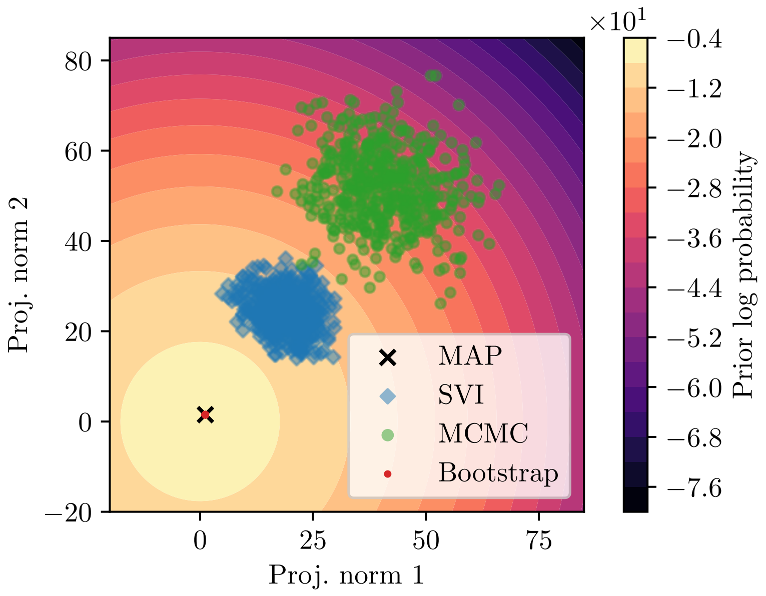


Figure 2.7: Visualization of the posterior samples of the inference methods considered. We consider a 2D plane spanned by two arbitrary weight vectors, and compute an orthogonal projection of the weight samples onto that plane. We also plot the contour lines of the prior log probability. We see that while the results of SVI and MCMC differ marginally, MCMC explores a larger volume of the posterior. Bootstrap samples hardly leave the immediate vicinity of the posterior mode.

2.5.5 Laplace approximation

When we use the Laplace approximation, we observe that the predictions become uncertain for all input points, even though the model still has perfect accuracy on the validation set. When using the probit prediction in Eq. (2.9) and a spherical posterior $\mathbf{V} = \sigma_w^2 \mathbb{I}$, as σ_w^2 increases, the κ value approaches 0 (noting that $0 \leq \kappa \leq 1$) and so does the activation $\kappa(\sigma_a^2)\mu_a$. As a result, the sigmoid output approaches 0.5, meaning that *all* the predictions become uncertain. Note that κ is non-negative and has no impact on the *sign* of the activation, so the decision boundary and hence the accuracy of the model are unchanged.

We suspect that the Laplace approximation overestimates the model uncertainty because it is only matching the posterior *at the mode*, which does not work well in the spheres problem which is linearly separable. We visualize this behavior using a toy 1D classification task, plotting the posteriors in Fig. 2.8. When the problem is linearly separable (the plot on the right) and the prior is sufficiently wide, the posterior resembles the prior at the mode. As we move away from the mode, however, the prior is sharply "cut" by a logistic likelihood for parameter settings that result in mistakes on the training points. When only matching curvature at the mode, such "cuts" are not captured.

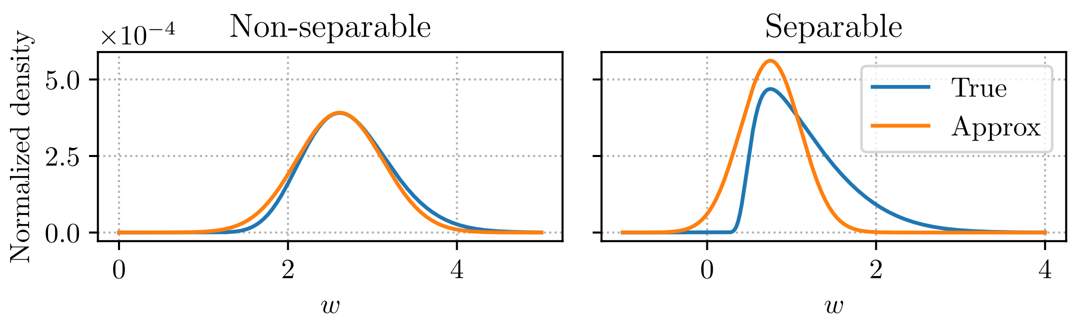


Figure 2.8: Laplace posterior fits for a toy 1D classification problem. In the *non-separable* problem classes are defined by two univariate Gaussians that heavily overlap. In the *separable* problem the two Gaussians are moved further apart. w has a Gaussian prior: $p(w) = \mathcal{N}(0, 1)$

Kuss and Rasmussen (2005) study the Laplace approximation in the context of inference for Gaussian Process classification, and also note the tendency of this method to misplace posterior mass and overestimate the predictive uncertainty.

2.5.6 Hierarchical model

We know that the true decision boundary in the spheres problem is expressed by all weights being set to the *same* value, at least when we ignore normalization. In this particular case, we know this because we use a synthetic problem that we have designed ourselves. With real problems, however, we might still know *something* about the dependency structure of the parameter space. In the Bayesian framework, one way to express such prior knowledge is using a *hierarchical prior*. For example, we can define a *hyperprior* on the mean of the weight prior:

$$\mu \sim \mathcal{N}\big(0, \sigma_\mu^2\big), \tag{2.56}$$

$$w_i \sim \mathcal{N}\big(\mu, \sigma_w^2\big). \tag{2.57}$$

In other words, we assume the weights have been sampled from a distribution with an unknown (potentially non-zero) mean, and do inference for this mean as well. This induces a correlation between the weights in the prior, the strength of which depends on σ_μ and σ_w, as demonstrated in Fig. 2.9. Smaller σ_w expresses the belief that the weights do not deviate much from the mean, and vice versa.

We can then run MCMC on this hierarchical model. In practice, however, MCMC might mix slowly due to μ and w_i's being heavily correlated, which is why an equivalent *non-centered*

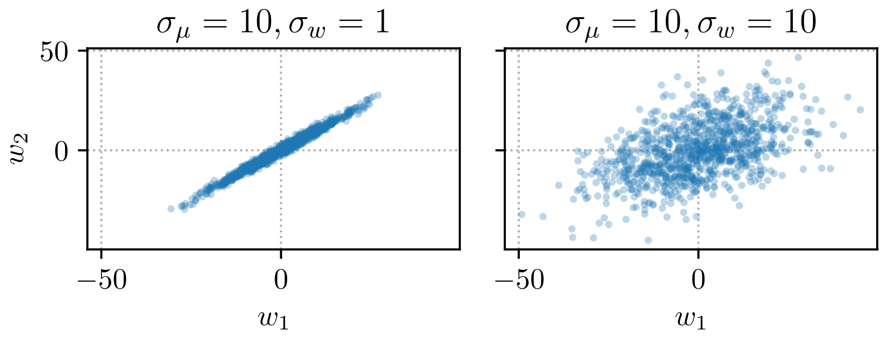


Figure 2.9: Samples from a hierarchical prior. As defined by Eqs. (2.56) to (2.57), changing the σ_w value.

parametrization can be used (Betancourt and Girolami, 2015):

$$\mu \sim \mathcal{N}(0, \sigma_\mu^2), \tag{2.58}$$

$$\hat{w}_i \sim \mathcal{N}(0, \sigma_w^2), \tag{2.59}$$

$$w_i = \mu + \hat{w}_i. \tag{2.60}$$

This new parametrization allows the sampler to explore the parameter space more efficiently by sampling from two independent distributions, all while preserving the desired correlations between w_i's.

In our experiments we use HMC to sample from the non-centered model. We use $\sigma_\mu = 10$ and vary σ_w. The bias parameter has a simple non-hierarchical Gaussian prior with $\sigma_b = 10$. Results are summarized in Table 2.2. To ensure that the prior is sensible, i.e. that the target weights are indeed all equal, we no longer standardize the features. This explains why the baseline results in this section are marginally worse in comparison to the MCMC results in Table 2.1.

We see that the hierarchical model drastically improves upon the non-hierarchical baseline for both values of σ_w. In both cases, adversarial optimization is no longer able to discover misclassified points. For $\sigma_w = 1$ (stronger belief that weight values are close to one another) the adversarial optimization is not able to find points with increased uncertainty at all. While the latter is hardly surprising given that we are almost giving away the solution, the former shows that even when the prior allows for high variation in the weights, it usefully regularizes the model. These results showcase the value of hierarchical models that is well-known in Bayesian literature (Gelman and Hill, 2006), even when the hyperpriors used are wide and non-informative.

Instead of fixing σ_w, one could set a hyper-prior over it as well, e.g. an inverse gamma

Table 2.2: Results for the Bayesian logistic regression with a hierarchical prior. Run on the spheres problem using elementwise squared features, a non-centered hierarchical prior as defined in Eqs. (2.58) to (2.60), and MCMC inference. σ_m is fixed to 10. Confidence defined as $p(y_{true} \,|\, \mathbf{x})$. Error defined as $p(1 - y_{true} \,|\, \mathbf{x})$. Adversarial dataset produced by perturbing validation points as described in Section 2.3.7.

	Validation		Adversarial	
Model	Log likelihood	Confidence	Log likelihood	Max. error
Baseline	−0.114	0.892	−0.879	0.635
Hierarchical, $\sigma_w = 10$	−0.002	0.998	−0.355	0.363
Hierarchical, $\sigma_w = 1$	−0.002	0.998	−0.002	0.002

distribution. This way we would not *prescribe* how much the weights can diverge from the mean, but would let the model learn this from data.

2.5.7 Transferability of adversarial points

In this section we test whether the adversarial points found for one bootstrap or MCMC ensemble transfer to *new* ensembles obtained using the same method. More precisely, we take the following steps:

Step 1: Fit a set of models with bootstrap / sample from the posterior using MCMC.

Step 2: Run adversarial optimization, saving the adversarial points found.

Step 3: Fit/sample *another* ensemble using the same training dataset.

Step 4: Evaluate the new ensemble on the previously saved adversarial points.

The results are shown in Table 2.3. We see that for both bootstrap and MCMC methods the adversarial points do indeed transfer — the new ensembles stay uncertain and/or wrong for the adversarial points found for a different ensemble. This suggests that adversarial optimization not only exploits a particular ensemble, but discovers points for which the training data and/or the inductive bias in the model are insufficient to restrict the weights.

The predictive uncertainty of the new bootstrap ensemble increases, but marginally. We explain this by our observations in Section 2.5.3: *all* of the bootstrap ensembles will have "wrong" weights in directions in which the MAP solution is wrong given a fixed, finite training data sample, and these are exactly the directions that adversarial optimization targets. As a result, the same adversarial points will generalize to *all* bootstrap ensembles for the same fixed dataset.

Predictions of the new MCMC ensemble, on the other hand, become *completely* uncertain on the same points: $p(1 - y_{\text{true}} \,|\, \mathbf{x}) \approx 0.5$. This suggests that for MCMC the adversarial optimization exploits the finiteness of the posterior sample, not the method itself. In other

Table 2.3: Resampling results for Bayesian logistic regression with elementwise squared features. One ensemble is trained using the corresponding method (*original* results), and adversarial points are found for it. A different ensemble is obtained using the same method, and is evaluated on the same adversarial points (*resampled* results). Error defined as $p(1 - y_{true} \,|\, \mathbf{x})$. Adversarial dataset produced by perturbing validation points as described in Section 2.3.7.

	Log likelihood		Max. adv. error	
Model	Original	Resampled	Original	Resampled
Bootstrap	-1.259	-1.225	0.870	0.863
MCMC	-0.737	-0.533	0.612	0.504

words, unlike bootstrap, even if the adversary knows we are doing MCMC, they are unable to find directions in the input space with underestimated model uncertainty. In addition, this suggests that using more samples and/or improving mixing is likely to make the MCMC model even more robust.

2.6 Experiments: learned features

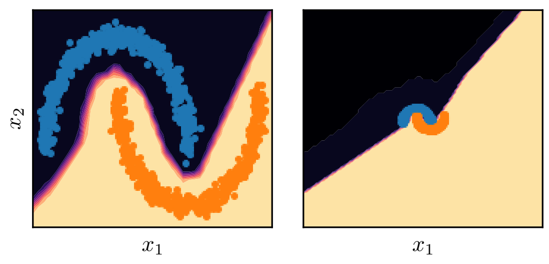
While feasible for the simple sphere problem considered in the previous section, for most problems of interest in modern machine learning (e.g. natural image classification) it is difficult to design useful features by hand. We now move on to experiments that rely on *learning* input features using neural networks.

2.6.1 Planar

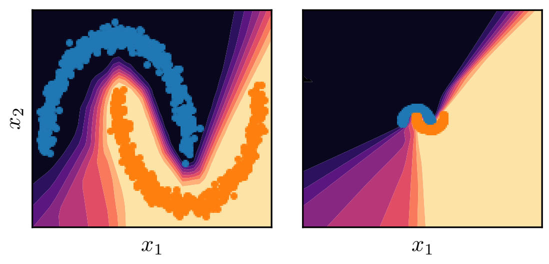
To begin with, we visualize the predictive uncertainty of hybrid models on a *planar* binary classification task, comparing to the "ideal" predictive uncertainty of full MCMC inference, and to the baseline predictive uncertainty of a MAP estimate. We use a small ReLU neural network with 2 hidden layers of size 10. We use the NUTS sampler, as in prior experiments. The predictive uncertainties are visualized in Fig. 2.10.

For points close to the training data the predictive uncertainties are comparable for all the methods. This is not the case for predictions further away from the training data, where the MAP model is unreasonably confident, while the full MCMC model becomes increasingly uncertain. The uncertainty does not increase in all directions, however: this is determined by a combination of the model architecture and the prior, as chosen by the practitioner.

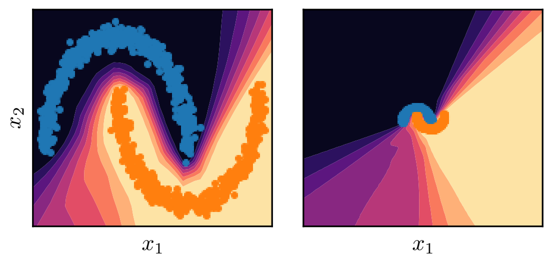
The predictive uncertainties of the hybrid models, i.e. ones where we use the MAP approximation for the initial layer(s) and do MCMC inference for the final layer(s), improve upon the complete confidence of the maximum likelihood model on points far away from



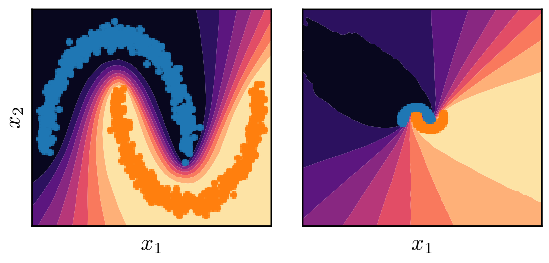
(a) MAP



(b) MCMC for the last layer



(c) MCMC for the last two layers



(d) MCMC for all layers

Figure 2.10: The predictive surface of a ReLU network when using a MAP approximation, MCMC inference, or hybrid inference. The network has 2 hidden layers of size 10. Training data is also plotted, colored by class. The right half of each subplot is the zoomed-out version of the left half.

the training data. This could explain why the hybrid methods can be sufficient for some downstream applications (Snoek et al., 2015; Bauer et al., 2017; Bradshaw et al., 2017). At the same time, the uncertainties are clearly underestimated when compared to the full Bayesian model. In the next sections we try to understand if these visual differences manifest in adversarial robustness.

2.6.2 Concentric spheres

We now turn back to the spheres problem, but instead of prescribing a feature space, we attempt to learn one using a neural network. We use a neural network with 4 hidden layers: the first three of size 1000, and the last of size $D = 500$. In line with Gilmer et al. (2018) we use ReLU activations and batch normalization (Ioffe and Szegedy, 2015). We train the neural network by maximum likelihood. We use 10^8 training examples (10^6 optimization steps with a batch size of 100, generating data online), and 10^6 validation examples. We use the Adam (Kingma and Ba, 2015) optimizer. The final model makes zero errors on the validation set (error rate $< 10^{-6}$).

We then use the output from the layer before last as a representation to learn on a *held-out*, smaller concentric spheres dataset, repeating experiments from Section 2.5. We standardize

Table 2.4: Results for Bayesian logistic regression on the concentric spheres problem with neural representations trained by maximum likelihood. Confidence defined as $p(y_{true} \mid \mathbf{x})$. Error defined as $p(1 - y_{true} \mid \mathbf{x})$. Adversarial dataset produced by perturbing validation points as described in Section 2.3.7.

	Validation		Adversarial	
Model	Log likelihood	Confidence	Log likelihood	Max. error
MAP	−0.000	1.000	−31.355	1.000
SVI	−0.000	1.000	−99.646	1.000
MCMC	−0.000	1.000	−89.099	1.000

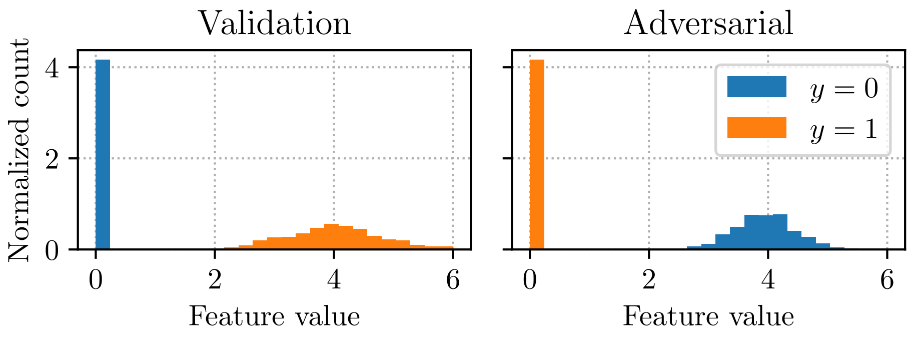


Figure 2.11: Values of an arbitrary feature in a neural representation for the concentric spheres problem colored by class.

the features to have zero mean and unit variance, removing the features with near-zero variance before passing the representations to the Bayesian logistic regression model. Results are summarized in Table 2.4.

As with the elementwise squared features, adversarial optimization finds points on the manifold that the MAP model misclassifies with high confidence, which replicates the findings of Gilmer et al. (2018). The adversarial log likelihood values are an order of magnitude lower than corresponding values in Table 2.1, suggesting a model is wrong with extreme confidence for a significant proportion of adversarial points. However, unlike the encouraging results when using the handcrafted features, neither SVI nor MCMC improve upon the MAP results when used on top of learned neural representations.

Trying to understand the negative results of the hybrid models, in Fig. 2.11 we plot the values of a particular neural feature for validation points and for adversarial points, colored by the true class. As we can see, this single feature allows the linear model to confidently separate the two classes. As a result, a Bayesian model will infer a posterior with a lot of mass put on the positive values of the weight corresponding to this feature. At the same time, in the right half of Fig. 2.11, we see that adversarial optimization discovers input points with *reversed* feature values for the two classes. In other words, *the representation*

Table 2.5: Results for Bayesian logistic regression with CNN representations on images. † for models that use a representation pre-trained on ImageNet. LL for log likelihood. Confidence defined as $p(y_{true} \,|\, \mathbf{x})$. Error defined as $p(1 - y_{true} \,|\, \mathbf{x})$. Adversarial dataset produced by perturbing validation points as described in Section 2.3.7.

	Validation			Adversarial		
Model	Accuracy	LL	Confidence	Accuracy	LL	Max. error
MAP	0.941	−0.187	0.936	0.033	−16.717	1.000
MCMC	0.941	−0.187	0.935	0.038	−16.834	1.000
SVI	0.941	−0.172	0.934	0.004	−10.846	1.000
MAP†	0.932	−0.164	0.911	0.000	−32.776	1.000
MCMC†	0.933	−0.164	0.910	0.001	−31.386	1.000
SVI†	0.936	−0.169	0.913	0.002	−31.413	1.000

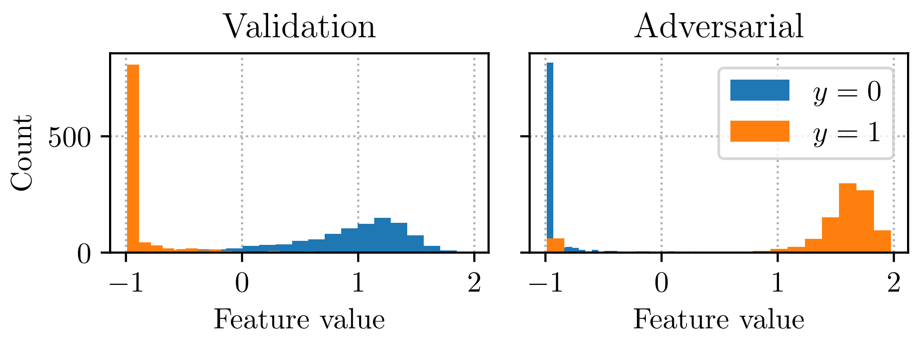


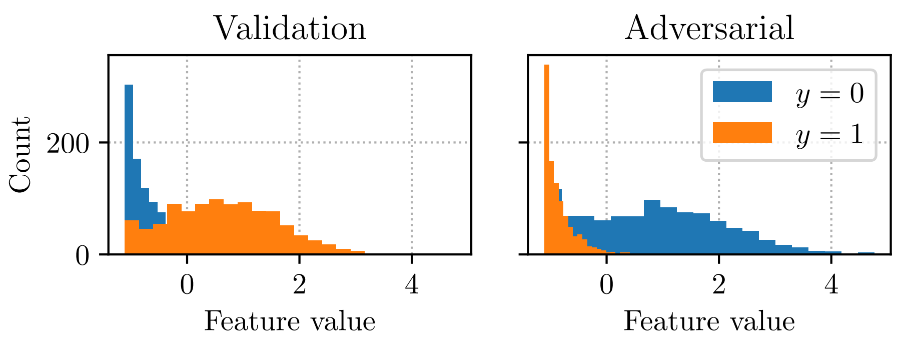
Figure 2.12: Values of an arbitrary feature in a CNN representation trained directly on the classification task.

network itself remains vulnerable, and even if we could do *exact* Bayesian inference we would not able to detect adversarial inputs in this representation space.

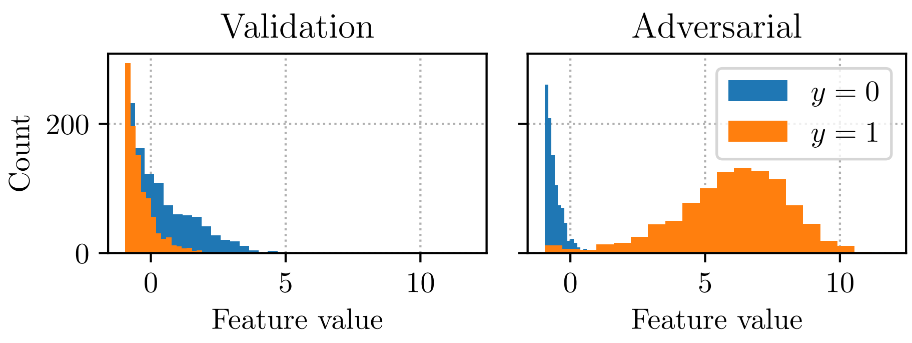
2.6.3 Images

To confirm the findings above in a more realistic setting, we move on from the synthetic spheres dataset to an image classification task. We use a convolutional neural network (CNN) to learn image representations, and, as before, use the Bayesian logistic regression in the learned representation space. In addition, we experiment with using a representation pre-trained on a *different* dataset (ImageNet, Russakovsky et al., 2015), which is a common approach in practice. Results are shown in Table 2.5.

We see that the results are similarly negative. The Bayesian logistic regression can not detect the adversarial points when used on top of the CNN representation space, which itself remains vulnerable as shown in Fig. 2.12. The same happens when using a pre-trained representation, even though Fig. 2.13 shows that the change of the feature values for adversarial points is not as drastic.



(a) Feature 34



(b) Feature 113

Figure 2.13: Values of features in a CNN representation pre-trained on ImageNet. Features with the worst discrepancy between validation and adversarial distributions are shown.

2.7 Discussion

In this chapter we have studied adversarial robustness of Bayesian logistic regression, demonstrating that, unlike its non-Bayesian counterpart, the Bayesian model no longer makes *confident* errors on adversarial points. While this is shown in an idealized setting with a synthetic dataset, to our knowledge we are the first to demonstrate that accurate Bayesian inference and informative priors can *completely eliminate* calibration errors. The use of complex neural network models and real datasets in prior work made it difficult to observe the same level of impact.

We have shown that these results do not translate to hybrid models, where the feature space, instead of being prescribed, is learned by a neural network. Such methods have been used successfully in prior work, and we present a setup where their uncertainty estimates are insufficient. This suggests that practitioners should be careful when using Bayesian approximations based on marginalizing over a *subset* of parameters, especially when adversarial robustness is important. An interesting future direction is to identify the properties of the representation space that would make the hybrid model robust. For example, in their subsequent work Watson et al. (2021) show that promoting *smoothness* and *diversity* of the features results in better predictive uncertainty of a hybrid model on out-of-distribution data.

The results of MCMC and SVI are comparable in our benchmarks. This is encouraging, because, unlike classic MCMC methods, SVI can scale to large models and datasets. At the same time, the performance of SVI would suffer if the model was more complex, and its posterior would not be well matched by a Gaussian distribution. Richer variational families based on normalizing flows (Chapter 3) can be used, but can significantly increase computational cost.

Laplace approximation is often used as simple method to fit a parametric distribution, but our observations re-iterate that Laplace approximation can not be applied to problems where the discovered posterior mode is not indicative of the rest of the posterior. Similarly, the baseline non-Bayesian ensembling method (bootstrap) has not worked well in this problem. Alternative non-Bayesian ensemble methods have shown promising results (Lakshminarayanan et al., 2017), and exploring these methods would be a useful extension of our work.

Exploring models tailored to *linearly separable* problems is another interesting direction for future work. For example, *Expectation Propagation* (EP) and *Assumed Density Filtering* (ADF) inference methods can be used with a *hard step* classifier (Minka and Picard, 2001), which is the limiting case of a logistic classifier when the weight norm goes to infinity. This could help eliminate the residual uncertainty of the Bayesian logistic regression model on validation points.

We re-iterate that in this chapter we only consider the *model calibration* aspect of the adversarial phenomenon. To many readers solving adversarial examples might mean "not changing the predicted class *or* becoming uncertain after making *imperceptible* changes to the input", which we believe has solely to do with the inductive bias of the model. If the logistic regression model fully captured the *rotational symmetry* present in the spheres problem, a *single* sample per class would be sufficient to learn the perfect decision boundary in an arbitrary dimension. Similarly, modern neural network image models have no notion of "imperceptible" changes, and adversarial examples are a manifestation of the inability of the model to learn to be invariant to such changes given limited data.

We demonstrate one way to induce the rotation symmetry in the spheres problem through the use of a hierarchical prior, and show that it eliminates the calibration errors completely. In realistic problems with deep neural networks, the choice of prior will only have a stronger effect on the uncertainties reported by Bayesian methods, and exploring the families of priors that can express symmetries and invariances in *real* problems is an important direction in

Bayesian deep learning.

Before we can design models that *are* able to generalize in ways that we expect, the best we can hope for is that our models *know when they do not know*, i.e. know when they lack the data to make confident predictions. It is undesirable for the model to become uncertain for perfectly valid inputs, but we believe it is strictly more desirable than making *wrong* predictions *confidently*. The squared features capture *some* symmetry inherent to the spheres problem, but the uncaptured symmetry manifests in model uncertainty. It is unlikely that we will ever be able to model *all* symmetries in real problems, hence capturing model uncertainty will continue to be an important aspect of building reliable models.

Chapter 3

Neural spline flows: flexible models for density estimation

A normalizing flow models a potentially complex distribution as a parametric transformation of a simple base distribution, turning the density estimation problem into a function approximation problem. To be able to match the target distribution with a normalizing flow, we need to parametrize an *invertible* function that is *flexible enough* to express the target function. In this work we propose novel, flexible parametrizations of invertible functions, and experimentally demonstrate their effectiveness when used in a normalizing flow.

This chapter is based on the following papers (* indicates joint-first authorship):

- Cubic-Spline Flows. Conor Durkan*, Artur Bekasov*, Iain Murray, George Papamakarios. Presented at the First Workshop on Invertible Neural Nets and Normalizing Flows (INNF), part of the Thirty-sixth International Conference on Machine Learning (ICML), 2019. Selected for a contributed talk. (Durkan et al., 2019a)
- Neural Spline Flows. Conor Durkan*, Artur Bekasov*, Iain Murray, George Papamakarios. In the proceedings of the Thirty-third Conference on Neural Information Processing Systems (NeurIPS), 2019. (Durkan et al., 2019b)

Author's contributions This work is the result of a collaboration between Artur, Conor, Iain and George. Artur and Conor have collaborated closely on deriving and implementing the proposed and baseline methods, which is reflected in joint-first authorship. Conor run tabular density estimation and variational autoencoder experiments, and contributed most to writing the papers, which is reflected in the ordering of the first authors. Artur run generative image modeling experiments, and implemented substantial parts of the normalizing flow framework. George contributed to designing and implementing the normalizing flow framework, and played a major part in shaping and directing the work. Iain proposed the rational-quadratic spline idea, and advised on the project.

This chapter is written solely by Artur, and contains no material written by collaborators, with the exception of the experimental details in Section 3.6, which were copied verbatim from the *Neural Spline Flows* paper. The chapter combines the two papers into a single narrative, provides a more extensive background and motivation, and introduces the methods in more detail.

3.1 Introduction

Density estimation is one of the fundamental problems in statistics and machine learning, where we aim to fit the distribution that the given dataset is likely to have been sampled from, and thus capture the stochastic process that generates the data. For continuous data, this distribution can be characterized by its *density function*, which is what we aim to approximate. We can then sample *novel* points from the process, compute the density of unseen points, or evaluate expectations with respect to the density. (Papamakarios, 2019)

Modern machine learning practitioners are interested in modeling densities of complex, high-dimensional data like images or audio. Classic density estimation methods struggle to fit such densities: histograms and kernel density estimation suffer from the curse of dimensionality, simple parametric density models are limited in their expressivity, while the cost of mixture models becomes excessive when using sufficiently many components.

Recently, the idea of a *normalizing flow* (Tabak and Turner, 2013; Rippel and Adams, 2013) has emerged as a parametric density model that can model complex, high-dimensional distributions. A normalizing flow models the density via a parametric, *invertible* transformation of some simple analytic density, such as a standard normal. We can sample from the flow by transforming the samples from the simple *base* density, and evaluate the density of an arbitrary point using the change-of-variables formula. The latter means the likelihood of the model is readily available and can be used as a principled criterion for training and model comparison.

The expressivity of the flow depends on the flexibility of the invertible transformation. Parametrizing flexible invertible transformations is challenging, and is an active area of research (Papamakarios et al., 2019a). In particular, the *autoregressive* and *coupling* transformations have attracted a lot of attention in literature. Both make use of the expressive power of neural networks, and both admit an analytic inverse and a cheap-to-compute determinant of the Jacobian, which is required to compute the density by the change-of-variables formula (Section 3.2.2). The autoregressive transformation is appealing as a *universal approximator* in the limit of infinite parameters, but computing the inverse of coupling transformation is considerably cheaper, even though it is often less expressive. Normalizing flows built by stacking multiple autoregressive or coupling layers demonstrate excellent performance on density estimation tasks (Huang et al., 2018; Kingma and Dhariwal, 2018).

Both autoregressive and coupling layers rely on parametrizing invertible *univariate* functions. Early work used simple additive/affine functions that can only shift and scale the input (Kingma et al., 2016; Papamakarios et al., 2017; Dinh et al., 2017). As a result, modeling complex densities would require stacking *many* layers, increasing computational cost, and potentially resulting in vanishing/exploding gradients (Koehler et al., 2021). For example, a state-of-the-art image flow architecture (Glow, Kingma and Dhariwal, 2018) uses up to *192* coupling layers. Later work proposed more expressive univariate function parametrizations (Ho et al., 2019; Jaini et al., 2019), including the method by Huang et al. (NAF, 2018) that uses a *neural network with positive weights* to parametrize the monotonic function, and as a result is universal in the limit of infinite parameters. However, the more expressive parametrizations like NAF generally do not provide an *analytic* inverse, requiring numerical approximation to invert the flow.

Müller et al. (2018) proposed an alternative, which is to use parametric monotonic *polynomial splines*, i.e. monotonic *piecewise polynomial* functions. Such functions can be arbitrarily flexible in the limit of infinitely many "pieces", yet the simplicity of each individual function means they still provide an analytic inverse and an easy-to-compute derivative. Müller et al. propose monotonic parametrizations of *linear* and *square* polynomial splines, using the resulting flows to learn proposal distributions for importance sampling.

In our work we continue this line of research, building upon the work of Müller et al. to propose two novel spline-based normalizing flow architectures that make use of autoregressive/coupling layers. We then demonstrate the improved flexibility of the novel parametrizations experimentally.

3.1.1 Contributions

In particular, we make the following contributions:

- We propose and implement *cubic* polynomial spline flows that use the method of Steffen (1990) to parametrize a monotonic cubic polynomial spline.
- We propose and implement *rational-quadratic* spline flows, using the monotonic rational-quadratic spline parametrization of Gregory and Delbourgo (1982).

- We propose the use of *linear tails* for adapting spline-based flows to work with unbounded data, overcoming the numerical instability of the sigmoid/logit transformation proposed by Müller et al. (2018).
- We combine the proposed methods into a novel class of flows that we refer to as *neural spline flows* (NSF), and evaluate their effectiveness on tabular density estimation, generative modeling of images, and inference in variational autoencoders.
- We show that NSF achieves state-of-the-art performance in tabular density estimation, in some cases allowing the coupling flows to outperform autoregressive flows.
- We demonstrate greatly improved parameter efficiency of NSF when compared to a state-of-the-art image flow (Glow, Kingma and Dhariwal, 2018), owing to the vastly reduced number of layers required to achieve comparable performance.

3.2 Background

In this section we introduce and motivate the density estimation task, and discuss the drawbacks of the traditional density estimation methods. We then introduce the normalizing flow idea and state-of-the-art methods for constructing flexible normalizing flow models.

3.2.1 Density estimation

The general task of density estimation is defined as follows. Given a dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{N}$, we assume the data have been sampled from some underlying probability distribution: $\mathbf{x}_i \sim P^*$. If the distribution P^* is continuous, it can be defined by its *density function* p^*. The density $p^*(\mathbf{x})$ is defined as the probability of a point sampled from P^* falling in the infinitesimal region around $\mathbf{x}$.[1] In other words, $p^*(\mathbf{x})$ determines how much probability *mass* there is in the vicinity of $\mathbf{x}$. Importantly, p^* is *not* the probability of sampling $\mathbf{x}$, which is always zero in a continuous space. Instead, we integrate the density to obtain a probability of $\mathbf{x}$ lying in some continuous *region* D:

$$P^*(\mathbf{x} \in D) = \int_D p^*(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{3.1}$$

To define a valid distribution, the density function must be non-negative everywhere, and must be normalized, i.e. must integrate to one over its domain:

$$p^*(\mathbf{x}) \geq 0 \text{ for all } \mathbf{x}, \qquad \int p^*(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1. \tag{3.2}$$

[1]We discuss this in more detail when talking about densities on manifolds in Section 4.2.1.

The goal of density estimation is to recover the density function p^* given the finite sample $\mathcal{D}$. Sometimes we are interested in estimating a *conditional* density $p^*(\mathbf{x} \,|\, \mathbf{y})$ for some *context* variable $\mathbf{y}$, having observed a set of $(\mathbf{x}_i, \mathbf{y}_i)$ pairs instead.

Conceptually, by recovering the data density we capture the stochastic *generative* process that has produced the data. More practically, however, an accurate model of data density enables a broad set of applications. For example, we can use the density model to sample *novel* data from the distribution, either directly if supported by the density estimator, or using Markov chain Monte Carlo (MCMC) methods. We can evaluate *expectations* of arbitrary functions with respect to the density. We can perform *anomaly detection*, spotting input points that are unlikely to have been sampled from the data distribution.[2] We can perform lossless data compression using arithmetic coding (MacKay, 2003, Chapter 6), which can achieve near-perfect compression as the density model approaches true data density.

Moreover, density models are an important aspect of many machine learning methods. Variational inference, for example, fits a density model to the Bayesian posterior (Hoffman et al., 2013). The *variational autoencoder* (VAE, Kingma and Welling, 2014), a latent variable model fit using variational inference, relies on two density models: the *prior* and the conditional *posterior*. In *likelihood-free* inference, we approximate the posterior density *or* the likelihood (Papamakarios, 2019). Both *importance sampling* and MCMC, popular methods for sampling from arbitrary distributions, rely on the *proposal* density (Müller et al., 2018).

A *histogram* is a simple, yet widely popular density model which partitions the space into a set of non-overlapping bins, and sets the density in each bin to be proportional to number of datapoints that fall into said bin. *Kernel density estimation* (KDE, Murphy, 2012, Section 14.7.2) is another popular, non-parametric density estimation method, which is similar in spirit to the nearest neighbors approach to prediction. In KDE, the density of a point is defined by its average "distance" to the points in the observed dataset. The "distance" is defined by the *kernel* function k, which controls what form of "smoothing" we want to apply to the estimate. The KDE density is then defined as

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} k(\mathbf{x} - \mathbf{x}_i). \tag{3.3}$$

While conceptually simple, both histograms and KDE suffer from the *curse of dimensionality.*

[2] One has to be careful when using densities directly to detect out-of-distribution points in *high-dimensional* spaces. Paradoxically, for high-dimensional distributions the high-density points are often *not* located in regions that we will sample in more often, which can lead to unexpected results (Nalisnick et al., 2019a). The notions of a *typical set* and *typicality* are more intuitive in this case (Nalisnick et al., 2019b).

For histograms, it is manifested in the fact that the number of bins required to cover the space grows *exponentially* with dimensionality. For KDE, as for nearest neighbors, the distances become more difficult to reason about as the dimensionality increases, and the kernel/kernel bandwidth selection becomes a major challenge (Heidenreich et al., 2013).

We can fit high-dimensional data with a simple analytic density, e.g. a multivariate Gaussian, or a *mixture* of such densities. The *Gaussian mixture model* (GMM, Murphy, 2012, Section 11.2.1) fits a convex combination of multivariate Gaussians, each parametrized by its mean and covariance matrix. The GMM can fit *any* density given enough mixture components, and is a strong baseline for simpler density estimation problems (Uria et al., 2013). At the same time, its computational cost scales with the number of components, and many components are needed to represent simple densities (for example, a uniform density in a unit hypercube) that can be easily summarized by alternative models.

3.2.2 Normalizing flows

We now introduce the *normalizing flow* (Tabak and Turner, 2013; Rippel and Adams, 2013), a flexible parametric density model. A normalizing flow defines a density as an *invertible* transformation of some simple *base* density. Consider transforming the points $\mathbf{z} \in \mathbb{R}^D$ sampled from some distribution π with a function $\mathbf{f}$:

$$\mathbf{x} = \mathbf{f}(\mathbf{z}) \qquad \mathbf{z} \sim \pi(\mathbf{z}). \tag{3.4}$$

Unless $\mathbf{f}$ is an identity function, the transformed points $\mathbf{x}$ will clearly no longer be distributed according to π. To find the distribution of $\mathbf{x}$, we can use the change-of-variables formula (Murphy, 2012, Eq. 2.89):

$$\begin{aligned} p(\mathbf{x}) &= \pi(\mathbf{z})|\det \mathbf{J}_\mathbf{f}(\mathbf{z})|^{-1} && (3.5)\\ &= \pi(\mathbf{z})\,|\det \mathbf{J}_{\mathbf{f}^{-1}}(\mathbf{x})| && (3.6)\\ &= \pi\big(\mathbf{f}^{-1}(\mathbf{x})\big)\,|\det \mathbf{J}_{\mathbf{f}^{-1}}(\mathbf{x})|\,, && (3.7)\end{aligned}$$

where $\mathbf{f}^{-1}$ is the inverse of $\mathbf{f}$ s.t. $\mathbf{f}^{-1}(\mathbf{f}(\mathbf{z})) = \mathbf{z}$, and $\mathbf{J}$ is the $D \times D$ Jacobian matrix:

$$\mathbf{J}_\mathbf{f}(\mathbf{z})_{ij} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{z}_j}. \tag{3.8}$$

We apply the *inverse function theorem* (Spivak, 1971) to go from Eq. (3.5) to Eq. (3.6).

In practice, we choose a simple base density π, one that is easy to evaluate and sample from. The standard normal is a common choice. The idea behind a normalizing flow is that even if π is a simple distribution like the standard normal, the flow can model an *arbitrarily complex* $p(\mathbf{x})$, provided that $\mathbf{f}$ is sufficiently expressive.

If we *parametrize* the transformation function $\mathbf{f}_\theta$ we can effectively *optimize over the space of distributions* by optimizing over the transformation parameters θ. In particular, we can attempt to match a density p^* by minimizing a divergence loss between the target distribution and the distribution p_θ modeled by the flow. The Kullback–Leibler (KL) divergence loss is a common choice:

$$\begin{aligned}\mathcal{L}(\theta) &= D_{\mathrm{KL}}(p^*(\mathbf{x}) \,\|\, p_\theta(\mathbf{x})) &(3.9)\\ &= \mathbb{E}_{p^*(\mathbf{x})}\left[\log \frac{p^*(\mathbf{x})}{p_\theta(\mathbf{x})}\right] &(3.10)\\ &= -\mathbb{E}_{p^*(\mathbf{x})}[\log p_\theta(\mathbf{x})] + \text{const.} &(3.11)\end{aligned}$$

In the context of density estimation, we do not have access to the true p^* (this is what we are aiming to estimate), but we assume that the training data have been sampled from it. This is why we can use the training points $\mathbf{x}^{(i)}$ to compute the Monte Carlo estimate of Eq. (3.11):

$$\begin{aligned}\hat{\mathcal{L}}(\theta) &= -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta\left(\mathbf{x}^{(i)}\right) &(3.12)\\ &\approx -\mathbb{E}_{p^*(\mathbf{x})}[\log p_\theta(\mathbf{x})]. &(3.13)\end{aligned}$$

In other words, by minimizing $\hat{\mathcal{L}}$ we are maximizing the probability of the data under the flow distribution, or *maximizing the likelihood* of the flow parameters given the data. This is why we refer to this procedure as *maximum likelihood* training.

Eq. (3.7) is only defined if $\mathbf{f}^{-1}$ exists, i.e. the function $\mathbf{f}$ must be *invertible* (or *bijective*). Moreover, both $\mathbf{f}$ and its inverse $\mathbf{f}^{-1}$ must be *differentiable*, and must have a tractable $\det \mathbf{J}_\mathbf{f}$. Note that we only need $\mathbf{f}^{-1}$ to compute the likelihood during training, as per Eq. (3.7). This is why we often parametrize $\mathbf{h}_\theta(\mathbf{x}) = \mathbf{f}_\theta^{-1}(\mathbf{x})$ instead, which explains why we refer to the model as a *normalizing* flow: we can say we are learning a "sampler" $\mathbf{f}$, but we can also say we are learning a *normalizing* transformation $\mathbf{h}$, i.e. a transformation that turns the target distribution into a standard *normal*. Conceptually, it makes no difference which direction we parametrize: we can simply invert $\mathbf{f}_\theta$ to arrive at $\mathbf{f}_\theta^{-1}$, and vice versa, as $\left(\mathbf{f}_\theta^{-1}\right)^{-1} = \mathbf{f}_\theta$. In practice, however, the inverse operation can be computationally expensive.

Designing *expressive* parametrizations $\mathbf{h}_\theta/\mathbf{f}_\theta$ with all the desirable properties is not straightforward, and is an active area of research (Papamakarios et al., 2019a). Neural networks are expressive, differentiable function approximators, which makes it appealing to try to use them to parametrize the invertible transformation in a normalizing flow. Unfortunately, we can not use neural networks directly, as they are not restricted to stay invertible during training, even if we use invertible non-linearities like the sigmoid or the leaky ReLU. Moreover, to compute the $\det \mathbf{J}_\mathbf{f}$ for a free-form neural network $\mathbf{f}$, we need to compute the determinants of all the free-form weight matrices. The cost of computing the determinant for an arbitrary $D \times D$ matrix scales as $\mathcal{O}(D^3)$, making the computation increasingly expensive for large neural networks.

Behrmann et al. (2019) propose a method for regularizing a residual neural network to stay invertible during training, and a method to efficiently approximate the $\det \mathbf{J}_\mathbf{f}$ for said network. Unfortunately, doing so involves significant technical difficulty, and results in a model that lacks certain desirable properties of a normalizing flow. In particular, there is no analytic inverse available (numerical inversion via fixed-point iteration must be performed), and we can not compute *exact* densities under the model. A more common solution is to construct neural network architectures that are restricted to express invertible functions and have an easy-to-compute $\det \mathbf{J}_\mathbf{f}$.

3.2.3 Autoregressive transform

We now consider one way to use neural networks to parametrize an invertible function: the *autoregressive transformation* (Kingma et al., 2016; Papamakarios et al., 2017). It works by transforming the input *elementwise*, conditioning each transformation on *preceding* dimensions:

$$\mathbf{x}_i = f_{\phi^{(i)}}(\mathbf{z}_i) \quad \text{where} \quad \phi^{(i)} = \mathbf{g}_\theta(\mathbf{z}_{1:i-1}). \tag{3.14}$$

In Eq. (3.14) $\mathbf{g}_\theta$ is the trainable *conditioner* function that outputs the parameters of the univariate transformations $f_{\phi^{(i)}}$. Note that for $i = 1$ the transformation is unconditional.

Evaluating the overall transformation is then as trivial as computing all $\phi^{(i)}$'s using the corresponding values of $\mathbf{z}$, and evaluating $f_{\phi^{(i)}}$ for each dimension i. This process can be parallelized: all transformations $f_{\phi^{(i)}}$ are independent given the input vector $\mathbf{z}$. In particular, when using a neural network to parametrize $\mathbf{g}_\theta$, we can use the method proposed by Germain et al. (MADE, 2015) to compute all $\phi^{(i)}$'s in a single forward pass, which is done by applying specially constructed masks to the weight matrices.

We can invert the autoregressive transformation in Eq. (3.14) by sequentially inverting each of the elementwise functions, using the $\mathbf{z}_i$'s computed in earlier steps to recover the transformation parameters:

$$\mathbf{z}_i = f^{-1}_{\phi^{(i)}}(\mathbf{x}_i) \quad \text{where} \quad \phi^{(i)} = \mathbf{g}_\theta(\mathbf{z}_{1:i-1}). \tag{3.15}$$

This means we only need to keep each univariate function f invertible to make the overall transformation $\mathbf{f}$ invertible. Moreover, because each output variable only depends on *earlier* input variables in the ordering, the Jacobian matrix will be triangular:

$$\mathbf{J}_{\mathbf{f}}(\mathbf{z})_{ij} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{z}_j} = 0 \quad \text{if} \quad i < j. \tag{3.16}$$

The determinant of a triangular matrix is the product of its diagonal entries, meaning that for the autoregressive transformation:

$$\det \mathbf{J}_{\mathbf{f}}(\mathbf{z}) = \prod_{i=1}^{D} \frac{\partial f_{\phi^{(i)}}}{\partial \mathbf{z}_i}. \tag{3.17}$$

This means that we only need to compute the derivatives of the elementwise functions to compute the $\det \mathbf{J}_{\mathbf{f}}$. The fact that we never have to invert $\mathbf{g}_\theta$ *or* compute its Jacobian means we can use a complex, non-invertible model like a neural network to parametrize it.

Autoregressive layers are especially appealing because we know that the autoregressive flow with a uniform base distribution is a universal approximator, which means that we can model *any* valid distribution in the limit of infinite parameters. We now briefly outline the proof of this statement. First note that according to the product rule of probability any multivariate distribution can be decomposed into a sequence of univariate conditional densities:

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{x}_1)p(\mathbf{x}_2 \,|\, \mathbf{x}_1)p(\mathbf{x}_3 \,|\, \mathbf{x}_1, \mathbf{x}_2)... && (3.18) \\ &= \prod_{i=1}^{D} p(\mathbf{x}_i \,|\, \mathbf{x}_{1:i-1}). && (3.19) \end{aligned}$$

Any univariate density, in turn, can be expressed as a transformation of the uniform distribution on $(0, 1)$. Consider $F(x)$, the *cumulative density function (CDF)* of some target density $p(x)$, which is defined as

$$F(x) = \Pr\bigl(x' \leq x\bigr) = \int_{-\infty}^{x} p\bigl(x'\bigr) \,\mathrm{d}x'. \tag{3.20}$$

We can show that if we transform the samples from the uniform distribution with the *inverse* CDF of $p(x)$

$$\tilde{x} = F^{-1}(u) \quad \text{where} \quad u \sim \mathcal{U}(0, 1), \tag{3.21}$$

then the CDF of $\tilde{x}$ will be equal to $F(x)$:

$$F_{\tilde{x}}(x) = \Pr\big(F^{-1}(u) \leq x\big) \tag{3.22}$$
$$= \Pr(u \leq F(x)) \tag{3.23}$$
$$= F(x), \tag{3.24}$$

which in turn implies that $p(\tilde{x}) = p(x)$, i.e. we recover the target distribution. To go from Eq. (3.23) to Eq. (3.24) we use the CDF of the uniform distribution:

$$F_u(y) = \Pr(u \leq y) = y \quad \text{if} \quad u \sim \mathcal{U}(0,1). \tag{3.25}$$

This completes the proof: we can represent any density in $\mathbb{R}^D$ by autoregressively transforming a uniform base distribution in the $(0,1)^D$ cube, as soon as the univariate transforms $f_{\phi^{(i)}}$ can match the (inverse) conditional CDFs of the target distribution. □

The forward direction of the autoregressive layer can be computed efficiently: values of all output dimensions can be computed in parallel. This is not the case for the inverse: as per Eq. (3.15), to computing inverse for i-th dimension we need to have inverted all the previous dimensions in the ordering. Such operation can only be performed sequentially, and its computational cost scales linearly with the number of dimensions. This means that if we parametrize $\mathbf{f}_\theta^{-1}$ with an autoregressive flow, we can cheaply evaluate the densities, but sampling from the flow is expensive. If we parametrize $\mathbf{f}_\theta$, on the other hand, computing densities of arbitrary points is expensive, but we can cheaply sample from the flow and compute densities *of the samples*, given that we know which points in the $\mathbf{z}$ space have produced them. This asymmetry is a considerable disadvantage of the autoregressive layer.

3.2.4 Coupling transform

One way we can speed up the inverse of the autoregressive transform is to transform multiple variables at once. This is the main idea behind the *coupling layer* (Dinh et al., 2017), which elementwise-transforms a *block* of variables, conditioning the transformation on the remainder of the variables. More precisely, assuming the dimensions have been ordered, for d s.t. $1 < d < D$ we compute $\mathbf{x} = \mathbf{f}(\mathbf{z})$ as follows:

$$\phi = \left\{\phi^{(i)}\right\}_{i=1}^{d} = \mathbf{g}_\theta(\mathbf{z}_{d:D}) \tag{3.26}$$
$$\mathbf{x}_{1:d-1} = \left[f_{\phi^{(1)}}(\mathbf{z}_1) \ \cdots \ f_{\phi^{(d-1)}}(\mathbf{z}_{d-1})\right]^\top \tag{3.27}$$
$$\mathbf{x}_{d:D} = \mathbf{z}_{d:D} \tag{3.28}$$

Typically $d = \lfloor D/2 \rfloor + 1$, i.e. we transform a half of the variables, conditioned on the other half. Note that the variables $\mathbf{z}_{d:D}$ that we condition the transformation on are themselves *not* transformed. While this clearly limiting, the transformation of the other variables can now be parallelized, both in the forward and in the inverse direction. In particular, to compute the inverse $\mathbf{z} = \mathbf{f}^{-1}(\mathbf{x})$ we simply go back through Eqs. (3.26) to (3.28):

$$\mathbf{z}_{d:D} = \mathbf{x}_{d:D} \tag{3.29}$$

$$\phi = \left\{\phi^{(i)}\right\}_{i=1}^{d} = \mathbf{g}_\theta(\mathbf{z}_{d:D}) \tag{3.30}$$

$$\mathbf{z}_{1:d-1} = \left[f^{-1}_{\phi^{(1)}}(\mathbf{x}_1) \ \cdots \ f^{-1}_{\phi^{(d-1)}}(\mathbf{x}_{d-1})\right]^\top \tag{3.31}$$

This means that the coupling layer is equally fast in both directions. As with the autoregressive layer, the inverse still only requires inverting the univariate functions, and never the $\mathbf{g}_\theta$. Moreover, the Jacobian of the transformation is still lower-triangular, taking the following form:

$$\mathbf{J}_\mathbf{f}(\mathbf{z}) = \begin{bmatrix} \operatorname{diag}\left(f'_{1:d-1}\right) & \mathbf{0} \\ \mathbf{J}_{\mathbf{g}_\theta}(\mathbf{z}_{d:D}) & \mathbb{I} \end{bmatrix} \quad \text{where} \quad f'_i = \frac{\partial f_{\phi^{(i)}}}{\partial \mathbf{z}_i}. \tag{3.32}$$

This means that the determinant of the Jacobian is also the product of diagonal terms, in this case:

$$\det \mathbf{J}_\mathbf{f}(\mathbf{z}) = \prod_{i=1}^{d-1} \frac{\partial f_{\phi^{(i)}}}{\partial \mathbf{z}_i}. \tag{3.33}$$

Given that a subset of the variables are not transformed, clearly a single coupling layer can not represent all transformations, even if the elementwise transform f is arbitrarily flexible. This is the trade-off that comes with an efficient inverse. At the same time, recently Teshima et al. (2020) have shown that when combining the coupling layer with an invertible *affine* layer (see Section 3.2.6), we also arrive at a universal approximator.

There is a spectrum between a coupling layer and an autoregressive layer: in general we can split the variables into M *blocks*, and transform one block at a time conditioned on prior blocks in some ordering. The coupling layer corresponds to one extreme with $M = 2$, and the autoregressive layer corresponds to the other extreme with $M = D$, i.e. each block contains one variable only. There has been little work exploring other points on the spectrum, however.

3.2.5 Stacking multiple transforms

The expressivity of autoregressive and coupling layers will depend both on the expressivity of f, and on the ordering of variables. For the coupling layer the ordering is especially critical: it will determine which subset of the variables is transformed. The universality proof of the autoregressive layer holds for any ordering, but in practice the assumption that f is arbitrarily flexible will not hold. The variable ordering will determine the complexity of the univariate conditional distributions to be matched in the autoregressive layer.

One simple way to increase the expressivity of a normalizing flow is to stack *multiple* transformation layers. This is similar in spirit to neural networks, where relatively inexpressive layers are stacked to obtain an expressive function. Stacking is also well-defined in normalizing flows because a sequence of invertible transformations stays invertible. The inverse of a composite function $\mathbf{x} = \mathbf{f}(\mathbf{z}) = \mathbf{f}_2(\mathbf{f}_1(\mathbf{z})) = (\mathbf{f}_2 \circ \mathbf{f}_1)(\mathbf{z})$ is defined as

$$\mathbf{z} = \mathbf{f}^{-1}(\mathbf{x}) = (\mathbf{f}_2 \circ \mathbf{f}_1)^{-1}(\mathbf{x}) = \left(\mathbf{f}_1^{-1} \circ \mathbf{f}_2^{-1}\right)(\mathbf{x}), \tag{3.34}$$

meaning that if ${\mathbf{f}_1}^{-1}$ and ${\mathbf{f}_2}^{-1}$ are well defined, then so is $\mathbf{f}^{-1}$. By the same reasoning, a sequence of differentiable transforms is also differentiable. Moreover, the $\det \mathbf{J}$ of the composite function is defined in terms of $\det \mathbf{J}$'s of the individual functions:

$$\det \mathbf{J}_{\mathbf{f}}(\mathbf{z}) = \det \mathbf{J}_{\mathbf{f}_2 \circ \mathbf{f}_1}(\mathbf{z}) = \det \mathbf{J}_{\mathbf{f}_1}(\mathbf{z}) \det \mathbf{J}_{\mathbf{f}_2}(\mathbf{f}_1(\mathbf{z})), \tag{3.35}$$

which means that if $\det \mathbf{J}_{\mathbf{f}_1}$ and $\det \mathbf{J}_{\mathbf{f}_2}$ are cheap to compute, so will be computing $\det \mathbf{J}_{\mathbf{f}}$.

3.2.6 Invertible linear layers

When stacking multiple layers in a normalizing flow, it is useful to change the ordering of variables between layers. In earlier work this is done by *reversing* the ordering in each subsequent layer, starting with either the original dataset ordering (Kingma et al., 2016; Papamakarios et al., 2017), or a handcrafted ordering informed by the input structure (Dinh et al., 2017). For the coupling layer in particular this ensures that all variables are eventually transformed.

Note that reversing the ordering of dimensions is equivalent to applying a particular *fixed* permutation matrix to the input vector. Can we *learn* this permutation matrix? While the permutation operation is clearly non-differentiable, the set of permutation matrices is a subset of the general set of *invertible* matrices. If we could find a differentiable parametrization of invertible matrices, we could parametrize a set of transformations that contains permutations.

The affine transformation $y = \mathbf{M}x$ is not invertible in general, only if $\mathbf{M}$ is non-singular.

Even if we somehow regularize $\mathbf{M}$ to prevent it from becoming singular during training, $\det \mathbf{J}$ of the affine transformation is equal to $\det \mathbf{M}$, which costs $\mathcal{O}\big(D^3\big)$ to compute for a $D \times D$ matrix $\mathbf{M}$. The same cost in incurred when computing $\mathbf{M}^{-1}$ for the flow inverse: $x = \mathbf{M}^{-1}y$.

Instead, Kingma and Dhariwal (2018) propose to parametrize an invertible matrix via its PLU *decomposition*, which alleviates the problems above. In particular, they propose parametrizing two matrices: a lower-triangular matrix $\mathbf{L}$ with *positive diagonal values*, and an upper-triangular matrix $\mathbf{U}$ with ones on the diagonal. For some fixed permutation matrix $\mathbf{P}$, the resulting matrix $\mathbf{M} = \mathbf{PLU}$ is guaranteed to be invertible: $\mathbf{M}$ is invertible if $\det \mathbf{M} \neq 0$, and

$$\det \mathbf{M} = \underbrace{\det \mathbf{P}}_{1} \det \mathbf{L} \underbrace{\det \mathbf{U}}_{1} = \prod_{i=1}^{D} \mathbf{L}_{ii} > 0, \tag{3.36}$$

where the inequality holds as we restrict the diagonal of $\mathbf{L}$ to be positive. Eq. (3.36) demonstrates another benefit of this parametrization, which is that computing the $\det \mathbf{M}$ is now a $\mathcal{O}(D)$ operation. Moreover, computing $\mathbf{M}^{-1}$ is also considerably cheaper:

$$\mathbf{M}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}^{\top}, \tag{3.37}$$

where computing $\mathbf{L}^{-1}$ and $\mathbf{U}^{-1}$ for triangular $\mathbf{L}$ and $\mathbf{U}$ is $\mathcal{O}\big(D^2\big)$ when using back/forward substitution. In practice, we sample a permutation matrix $\mathbf{P}$ before training, and only optimize $\mathbf{L}$ and $\mathbf{U}$. The fixed $\mathbf{P}$ is a limitation of the PLU parametrization, because it restricts the set of invertible matrices we can represent.

Alternative matrix decompositions can be used. Hoogeboom et al. (2019b) propose to use the QR decomposition $\mathbf{M} = \mathbf{QR}$, parametrizing an *orthogonal* matrix $\mathbf{Q}$, and an upper-triangular matrix $\mathbf{R}$ with a positive diagonal. The QR parametrization, unlike the PLU with fixed $\mathbf{P}$, can represent *all* invertible matrices $\mathbf{M}$. One can also consider using the SVD decomposition $\mathbf{M} = \mathbf{U\Sigma V}^{\top}$, where $\mathbf{V}$ and $\mathbf{U}$ are orthogonal, and $\mathbf{\Sigma}$ is diagonal with positive values. The main benefit of the SVD parametrization is a cheaper inverse when compared to PLU and QR parametrizations:

$$\mathbf{M}^{-1} = \mathbf{V\Sigma}^{-1}\mathbf{U}^{\top}, \tag{3.38}$$

where $\mathbf{\Sigma}^{-1}$ is trivial for the diagonal matrix $\mathbf{\Sigma}$.

While the QR and SVD parametrizations are attractive in theory, in practice parametrizing the orthogonal matrices can be challenging. Differentiable parametrizations of orthogonal matrices exist: for example, we can use the *Householder transformations* (Tomczak and

Welling, 2016) to parametrize any $D \times D$ orthogonal matrix $\mathbf{M}$ as a sequence of D reflections. However, most such parametrizations come with a expressivity vs. cost trade-off, and can be expensive when fully general. We refer the reader to (Papamakarios et al., 2019a, Appendix A) for more detail.

3.2.7 Invertible univariate transformations

We now discuss the parametrization of the univariate function f. The parametrization must be restricted to invertible functions, must be differentiable, and the more flexible it is, the more complex densities the normalizing flow will be able to model. The *additive* function is a simple parametrization that has been used in early work by Dinh et al. (2015):

$$f_\phi(\mathbf{z}_i) = \mathbf{z}_i + t \quad \text{where} \quad \phi = \{t\}\,. \tag{3.39}$$

The derivative of this additive function is a constant: $\frac{\partial f_\phi}{\partial \mathbf{z}_i} = 1$. While this is convenient for computing the density, this makes the transformation *volume preserving*. This means that the transformation can not *expand* or *contract* density, only move it around, which is limiting.

A generalization of the additive function proposed by Dinh et al. (2017) is the *affine* function, which also *scales* the input:

$$f_\phi(\mathbf{z}_i) = \exp(s) \cdot \mathbf{z}_i + t \quad \text{where} \quad \phi = \{s, t\}\,. \tag{3.40}$$

The exp is a simple transformation that restricts the multiplier to positive values, guaranteeing invertibility. More numerically stable alternatives to exp are often used in practice, for example $\text{softplus}(s) + \epsilon$ for $\epsilon > 0$.

While the affine transformation is no longer volume preserving, and hence more expressive than the additive one, it still is of limited flexibility. For example, if the base distribution is a standard normal and we use an autoregressive layer with affine univariate functions, we can only translate and scale a standard normal in each univariate transformation. This means we can only represent distributions that can be factorized into a sequence of normal conditional distributions.

Alternative ways of parametrizing invertible univariate functions have been considered. Neural autoregressive flow (NAF, Huang et al., 2018) parametrizes a neural network with positive weights and monotonic activation functions. The function modeled by such neural network is guaranteed to be monotonically increasing, and we can represent *any* monotonic function in the limit of infinite weights. Ho et al. (2019) propose parametrizing a mixture of logistic CDFs, noting that a convex combination of monotonic functions is also monotonic.

Sum-of-squares polynomial flow (SOS, Jaini et al., 2019) parametrizes coefficients of a monotonic polynomial of an arbitrary degree by rewriting it as a sum of squares. While these approaches greatly enhance the expressivity of a flow, they do not provide an analytic inverse, meaning that we need to perform inefficient numerical inversion using methods like bisection search.

3.2.8 Spline transforms

An alternative way to define a complex univariate function is by using a *spline*, i.e. as a *piecewise* function. To do so we divide some finite input domain $[a, b]$ into K non-overlapping pieces, or *bins*, where bin k is defined by its boundary coordinates $[x_k, x_{k+1}]$, s.t.

$$[a, b] = [x_0, x_1] \cup [x_1, x_2] \cup ... \cup [x_{K-2}, x_{K-1}] \cup [x_{K-1}, x_K] \tag{3.41}$$

$$a = x_0 \leq x_1 \leq ... \leq x_{K-1} \leq x_K = b. \tag{3.42}$$

We then transform the input x with a function corresponding to the bin that x belongs to:

$$f(x) = \begin{cases} f_0(x) & \text{if } x_0 \leq x < x_1 \\ f_1(x) & \text{if } x_1 \leq x < x_2 \\ & ... \\ f_{K-1}(x) & \text{if } x_{K-1} \leq x \leq x_K \end{cases} \tag{3.43}$$

While each function f_k can be simple (for example, a polynomial), concatenating K such functions into the spline f allows us to model increasingly complex functions as K increases. All the bin boundaries must be sorted as per Eq. (3.42), which means that in practice we can apply *binary search* to bin the input x with $\mathcal{O}(\log K)$ computational cost.

Müller et al. (2018) propose to use the splines to parametrize the univariate functions in a coupling layer. In particular, they explore *linear* and *quadratic* polynomial splines, i.e. splines where f_k are polynomials of the first and the second degree. In the linear spline, the transformation for bin k is defined as

$$f_k(x) = a_k x + b_k. \tag{3.44}$$

In the quadratic spline it is defined as

$$f_k(x) = a_k x^2 + b_k x + c_k. \tag{3.45}$$

The coefficients of the polynomials can not be arbitrary, however: we need to keep the function f invertible is we want to use it in the autoregressive/coupling layer. Müller et al. propose an elegant parametrization that keeps the linear and quadratic splines *monotonically increasing*. Instead of f itself they parametrize its *derivative* f', and integrate said derivative to evaluate f. By the fundamental theorem of calculus

$$f(x) = \int_a^x f'(t)\,\mathrm{d}t + f(a). \tag{3.46}$$

Müller et al. also fix $[a, b] = [0, 1]$, $f(0) = 0$ and $f(1) = 1$, so this simplifies to

$$f(x) = \int_0^x f'(t)\,\mathrm{d}t. \tag{3.47}$$

This means we can restrict f to be monotonically increasing on the interval by simply keeping f' *positive* on the interval. If the function is monotonically increasing, by definition $f(x_0) \leq f(x_1) \leq ... \leq f(x_{K-1}) \leq f(x_K)$, and the inverse is computed as:

$$f^{-1}(y) = \begin{cases} f_0^{-1}(y) & \text{if } f(x_0) \leq y < f(x_1) \\ f_1^{-1}(y) & \text{if } f(x_1) \leq y < f(x_2) \\ & \dots \\ f_{K-1}^{-1}(y) & \text{if } f(x_{K-1}) \leq y \leq f(x_K) \end{cases} \tag{3.48}$$

This means that the inverse can be computed analytically, with no numeric inversion required. Moreover, with Müller et al.'s parametrization the derivative f' is readily available for computing the Jacobian determinant.

Note that f' will be piecewise constant for the linear spline, and piecewise linear for the quadratic spline, which is easy to see by differentiating Eq. (3.44) and Eq. (3.45). Both piecewise constant and piecewise linear functions are straightforward to integrate exactly: we simply sum the areas of a sequence of rectangles (for piecewise constant) or right trapezoids (for piecewise linear). The computational cost of such integration scales linearly with K.

The linear spline is a universal approximator in the limit of infinite bins: for an infinitesimal bin width, to represent any function *exactly* one simply needs to set the slope of the line a_k to the derivative of the function. In practice, of course, we work with a finite number of bins K, and the flexibility of the function in each bin matters. Moreover, it is desirable to minimize the value of K while maintaining flexibility, as the computational cost of the parametrization scales with K.

The set of quadratic functions is a superset of the set of the linear functions, so the quadratic parametrization is strictly more flexible. However, it comes with additional benefits. First, it allows us to *learn* the bin boundaries $[x_k, x_{k+1}]$, in addition to the functions within said boundaries, enhancing expressivity for a fixed K. As discussed by Müller et al. (2018, Appendix B), we can not do the same for the linear spline, and instead have to split the input domain into K *fixed*, equally sized bins. And second, the derivative of the quadratic spline is continuous, which Müller et al. argue helps optimization.

3.2.9 Modeling discretized data

The standard normalizing flow is only well-defined in a continuous space. Fitting a continuous normalizing flow to discrete data is not meaningful, and will lead to degeneracies: a discrete distribution has no volume in continuous space, which allows a normalizing flow to "cheat" by placing narrow density spikes on the training datapoints. As the spikes get narrower and the densities approach infinity, the likelihood also approaches infinity, even though the normalizing flow is not capturing the target distribution in any meaningful way. Discrete normalizing flows have been proposed by Hoogeboom et al. (2019a) and Tran et al. (2019), but come with their own set of trade-offs.

An alternative solution is based on the observation that many datasets of interest are not *actually* discrete, rather they contain *discretized* representations of the underlying continuous signal. For example, the stored pixel intensities of natural images are *quantized* representations of the true, continuous color intensities. To undo the quantization, i.e. to *dequantize* the data, we can define a dequantization distribution $q(\tilde{\mathbf{x}} \,|\, \mathbf{x})$, where $\tilde{\mathbf{x}}$ is the unknown continuous signal and $\mathbf{x}$ is the discretized observation. Intuitively, $q(\tilde{\mathbf{x}} \,|\, \mathbf{x})$ defines the relative likelihood of $\tilde{\mathbf{x}}$ being the continuous signal that has produced the discrete observation $\mathbf{x}$ after quantization. The continuous dequantized distribution $p(\tilde{\mathbf{x}})$ is obtained by marginalizing over the discrete observations $\mathbf{x}$:

$$p(\tilde{\mathbf{x}}) = \sum_{\mathbf{x}} P(\mathbf{x}) q(\tilde{\mathbf{x}} \,|\, \mathbf{x}). \tag{3.49}$$

The distribution $p(\tilde{\mathbf{x}})$ can be fit with a continuous normalizing flow using the standard maximum likelihood loss in Eq. (3.12). To obtain the samples from $p(\tilde{\mathbf{x}})$ for computing the loss, we sample from $q\big(\tilde{\mathbf{x}} \,|\, \mathbf{x}^{(i)}\big)$ given the training points $\mathbf{x}^{(i)}$.

Theis et al. (2016) analyze how the density fit using this procedure relates to the target discrete probability $P(\mathbf{x})$. Consider the KL objective that we are maximizing when fitting the flow density $q(\tilde{\mathbf{x}})$ to match $p(\tilde{\mathbf{x}})$:

$$\begin{aligned}
\mathbb{E}_{p(\tilde{\mathbf{x}})}[\log q(\tilde{\mathbf{x}})] &= \int p(\tilde{\mathbf{x}}) \log q(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}} && (3.50)\\
&= \int \sum_{\mathbf{x}} P(\mathbf{x}) q(\tilde{\mathbf{x}} \,|\, \mathbf{x}) \log q(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}} && (3.51)\\
&= \sum_{\mathbf{x}} P(\mathbf{x}) \int q(\tilde{\mathbf{x}} \,|\, \mathbf{x}) \log q(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}} && (3.52)\\
&\leq \sum_{\mathbf{x}} P(\mathbf{x}) \log Q(\mathbf{x}), && (3.53)
\end{aligned}$$

where we use the Jensen's inequality to go from Eq. (3.52) to Eq. (3.53), and define $Q(\mathbf{x})$ as

$$Q(\mathbf{x}) = \int q(\tilde{\mathbf{x}} \,|\, \mathbf{x}) q(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}}. \tag{3.54}$$

In other words, when maximizing the standard likelihood-based objective using the dequantized data, we are maximizing a lower bound on the *discrete likelihood* for a probability mass function Q, which is implicitly defined by the continuous normalizing flow and the dequantization distribution.

As pointed out by Theis et al., the negative base-2 log of Q is equivalent to the average number of bits required to losslessly compress the (discretized) data with an entropy coding scheme that uses Q (Shannon, 1948). It is common to report this negative discrete log-likelihood in *bits per dimension* (BPD), dividing the base-2 log value by the number of input dimensions.

When it comes to natural image data, we commonly store the images with 8-bit color depth and 3 color channels (RGB): $\mathbf{x} \in \{0, ..., 255\}^{D \times D \times 3}$. We assume that the quantization is a simple *floor* operation $\mathbf{x} = \lfloor \tilde{\mathbf{x}} \rfloor$, i.e. the continuous color intensity $\tilde{x}$ is mapped to the greatest integer x that is *less or equal to* $\tilde{x}$. The support of $q(\tilde{\mathbf{x}} \,|\, \mathbf{x})$ in this case is a unit hypercube where $\mathbf{x} = \lfloor \tilde{\mathbf{x}} \rfloor$, and it is common to assume that $q(\tilde{\mathbf{x}} \,|\, \mathbf{x})$ is *uniform* in this hypercube.

Using the uniform dequantization makes it trivial to sample from $p(\tilde{\mathbf{x}})$: we simply add uniform noise to the training data $\tilde{\mathbf{x}} = \mathbf{x}^{(i)} + \mathbf{u}$, where $\mathbf{u}$ is a sample from a uniform distribution in $[0, 1)^{D \times D \times 3}$. In other words, to fit an implicit discrete distribution to the data we simply fit a continuous normalizing flow $q(\tilde{\mathbf{x}})$ to *noisy* training images. Moreover, when using uniform noise, the implicit probability mass function Q is defined as an integral over the density in the corresponding unit hypercube:

$$Q(\mathbf{x}) = \int_{[0,1)^{D \times D \times 3}} q(\mathbf{x} + \mathbf{u})\, \mathrm{d}\mathbf{u}. \tag{3.55}$$

This means we can easily compute a Monte Carlo estimate of Eq. (3.55), and hence of BPD, by taking the mean density of noisy test images, where only one noisy sample per test image is often used in practice.

The disadvantage of the uniform dequantization is that the peculiar continuous distribution defined by it — a set of hypercubes with uniform density — might be difficult to fit with a normalizing flow designed to model smooth densities. To alleviate this, Ho et al. (2019) propose to parametrize the dequantization density $q(\tilde{\mathbf{x}} \,|\, \mathbf{x})$ with a smaller normalizing flow and *learn* it jointly with the main flow. Ho et al. propose a training procedure based on variational inference, referring to the resulting method as *variational dequantization*, and empirically demonstrate improved performance when compared to uniform dequantization.

3.3 Neural spline flows

In this work, we build upon the spline flows of Müller et al. (2018). First we explore the *cubic* polynomial spline flows, the natural evolution of the square polynomial spline flows. We then consider using *rational* functions instead of a polynomials in a spline, proposing *rational-quadratic* spline flows. We also consider an alternative way of adapting spline flows to work with unbounded data. Finally, we combine the proposed methods into *neural spline flows*, a novel class of flexible density estimators.

3.3.1 Cubic splines

One way to further increase the flexibility of a quadratic polynomial spline is to simply increase the order of the polynomial. This results in a *cubic* spline, where the function in each bin is defined as

$$f_k(x) = a_k + b_k x + c_k x^2 + d_k x^3. \tag{3.56}$$

Beyond the potentially increased flexibility for a fixed number of bins, the cubic can be made second-order continuous. As discussed above, in their work Müller et al. (2018) note that first-order continuity of their quadratic spline parametrization is likely beneficial, as we use the gradients of the function during optimization. However, through the Jacobian determinant term in Eq. (3.7) the *gradients themselves* are a part of the loss, meaning that second-order gradients are also used during optimization, and hence them being continuous could also aid optimization.

We use the method of Steffen (1990) to restrict the cubic spline to be monotonically increasing. Instead of parametrizing $a_k, ..., d_k$ directly, Steffen parametrizes a set of $K + 1$

monotonically increasing *knots* $\{(x_k, y_k)\}_{k=0}^{K}$, such that

$$(x_0, y_0) = (0, 0), \quad (x_K, y_K) = (1, 1) \tag{3.57}$$

$$x_k < x_{k+1} \text{ and } y_k < y_{k+1} \text{ for } k = 0 : K - 1. \tag{3.58}$$

We can parametrize the knots via a set of *widths* $w_k = x_{k+1} - x_k$, and a set of heights $h_k = y_{k+1} - y_k$, both sets of size K. The knots are then defined by the cumulative sums of weights and heights:

$$x_k = \sum_{k'=0}^{k} w_{k'} \qquad y_k = \sum_{k'=0}^{k} h_{k'}. \tag{3.59}$$

To satisfy Eqs. (3.57) and (3.58), we ensure that all the weights and heights are positive, and that both sets sum to 1:

$$w_k > 0 \quad \text{and} \quad h_k > 0 \quad \text{for} \quad k = 0 : K - 1 \tag{3.60}$$

$$\sum_{k=0}^{K-1} w_k = 1, \quad \sum_{k=0}^{K-1} h_k = 1. \tag{3.61}$$

We can convert an arbitrary parameter vector $\theta \in \mathbb{R}^K$ to a vector of weights (or heights, equivalently) that satisfy the definition above by simply passing it through a *softmax* function:

$$w_k = \frac{\exp(\theta_k)}{\sum_{k'=0}^{K-1} \exp(\theta_{k'})}. \tag{3.62}$$

Having parametrized the knots, Steffen defines a cubic polynomial that smoothly interpolates between these knots, while keeping the function monotonically increasing. In other words, the function is defined by the set of points that lie on it (the knots), and the rest of the function is “filled in” by connecting these knots using cubic functions. In particular, each cubic function is defined as

$$f_k(\xi) = \alpha_k^{(0)} + \alpha_k^{(1)}\xi + \alpha_k^{(2)}\xi^2 + \alpha_k^{(3)}\xi^3, \tag{3.63}$$

where $\xi = x - x_k$ is the input in the “local” coordinate system of bin k. The coefficients are given by

$$\alpha_k^{(0)} = y_k \qquad \alpha_k^{(1)} = d_k \tag{3.64}$$

$$\alpha_k^{(2)} = \frac{3s_k - 2d_k - d_{k+1}}{w_k} \tag{3.65}$$

$$\alpha_k^{(3)} = \frac{d_k + d_{d+1} - 2s_k}{w_k^2}, \tag{3.66}$$

where w_k is the bin width as defined above, $s_k = h_k/w_k$ is slope of the line joining consecutive knots, and d_k is the derivative at the knot. All the derivatives apart from d_0 and d_K are prescribed:

$$d_k = \min(p_k, 2\min(s_{k-1}, s_k)), \qquad \text{where} \quad p_k = \frac{s_{k-1}w_k + s_k w_{k-1}}{w_{k-1} + w_k}. \tag{3.67}$$

The boundary derivatives d_0 and d_K are free parameters that can be learned, but have to be restricted to stay positive. Together with the widths w_k and the heights h_k this results in $2K + 2$ total learnable parameters for K bins. Intuitively, during optimization we move the knots to match the target function.

The derivative of the function f_k is trivially:

$$\frac{\partial f_k(\xi)}{\partial x} = \alpha_k^{(1)} + 2\alpha_k^{(2)}\xi + 3\alpha_k^{(3)}\xi^2. \tag{3.68}$$

Inverting f_k involves solving a cubic equation. More precisely, if we wish to compute $\xi = f_k^{-1}(y)$, we need to solve the following equation for ξ:

$$(\alpha_k^{(0)} - y) + \alpha_k^{(1)}\xi + \alpha_k^{(2)}\xi^2 + \alpha_k^{(3)}\xi^3 = 0 \tag{3.69}$$

Solving a cubic equation is considerably more difficult than solving a quadratic equation, especially doing so in a numerically stable fashion (Blinn, 2007). We have found the standard trigonometric and hyperbolic methods, as well as the Cardano's formula, to be too numerically unstable in practice. We settled on using a version of the method by Peters (2016), which we adapted to better deal with cases where the cubic approaches a quadratic ($\alpha_k^{(3)} \to 0$) or even a linear ($\alpha_k^{(3)} \to 0$, $\alpha_k^{(2)} \to 0$) function.

In general, a cubic equation can have up to three roots. Knowing that the parametrization keeps the function monotonic, only *one* of these roots will lie within the boundaries of the bin k. In practice, we compute all the roots, and select the one lying in the required interval.

3.3.2 Rational-quadratic splines

A alternative approach to increase the flexibility of a quadratic polynomial is to consider a fraction of *two* quadratic polynomials. Such fractions are known as *rational* functions, and the rational *quadratic* function takes the following form:

$$f_k(x) = \frac{\alpha_k^{(0)} + \alpha_k^{(1)} x + \alpha_k^{(2)} x^2}{\beta_k^{(0)} + \beta_k^{(1)} x + \beta_k^{(2)} x^2}. \tag{3.70}$$

We use the method of Gregory and Delbourgo (1982) to parametrize a monotonic rational-quadratic spline. In line with monotonic cubic spline parametrization, we define a sequence of knots $\{(x_k, y_k)\}_{k=0}^{K}$ such that Eqs. (3.57) and (3.58) hold. However, in this case none of the derivatives at the knots $\{d_k\}_{k=0}^{K}$ are prescribed. Instead we can learn the derivatives together with the knots, restricting them to they stay positive to preserve monotonicity. Given the knots and the derivatives, the method of Gregory and Delbourgo interpolates between the knots with a *monotonic* rational quadratic spline. In particular, they parametrize the transformation in bin k as a function of the transformed input x:

$$f_k(x) = g_k(\zeta) = \frac{\alpha_k(\zeta)}{\beta_k(\zeta)}, \tag{3.71}$$

where $\zeta = \frac{x - x_k}{w_k} \in [0, 1]$, and

$$\alpha_k(\zeta) = s_k y_{k+1} \zeta^2 + (y_k d_{k+1} + y_{k+1} d_k)\zeta(1-\zeta) + s_k y_k (1-\zeta)^2 \tag{3.72}$$

$$\beta_k(\zeta) = s_k \zeta^2 + (d_{d+1} + d_k)\zeta(1-\zeta) + s_k(1-\zeta)^2. \tag{3.73}$$

Gregory and Delbourgo note that g_k can be re-written as

$$g_k(\zeta) = y_k + \frac{h_k\big[s_k\zeta^2 + d_k\zeta(1-\zeta)\big]}{s_k + (d_{k+1} + d_k - 2s_k)\zeta(1-\zeta)}, \tag{3.74}$$

which is a more numerically stable form, especially for small s_k. The w_k, h_k and s_k in the formulas above are the bin widths, heights and slopes as defined in Section 3.3.1. Gregory and Delbourgo show that the spline defined by Eq. (3.74) will interpolate between the knots and match the corresponding derivatives d_k at each knot, all while staying monotonic on the $[0, 1]$ interval.

Differentiating the Eq. (3.74) results in:

$$\frac{\partial f_k(x)}{\partial x} = \frac{\partial g_k(\zeta)}{\partial \zeta}\frac{\partial \zeta}{\partial x} = \frac{s_k^2\big[d_{k+1}\zeta^2 + 2s_k\zeta(1-\zeta) + d_k(1-\zeta)^2\big]}{[s_k + [d_{k+1} + d_k - 2s_k]\zeta(1-\zeta)]^2}. \tag{3.75}$$

To compute the inverse $x = f_k^{-1}(y)$, we note that in the end f_k is a rational quadratic with a particular parametrization of the coefficients. Using f_k's definitions in Eqs. (3.70) and (3.71) we find the equation that needs to be solved for x:

$$q(x) = \alpha(\zeta(x)) - y\beta(\zeta(x)) \tag{3.76}$$

$$= \underbrace{\left(\alpha_k^{(2)} - \beta_k^{(2)} y\right)}_{a} x^2 + \underbrace{\left(\alpha_k^{(1)} - \beta_k^{(1)} y\right)}_{b} x + \underbrace{\left(\alpha_k^{(0)} - \beta_k^{(0)} y\right)}_{c} = 0 \tag{3.77}$$

As we can see, inverting the rational quadratic only requires solving a *quadratic* equation. We can rearrange Eq. (3.74) to find the coefficients of Eq. (3.77) in terms of the knots and derivatives:

$$a = h_k(s_k - d_k) + (y - y_k)(d_{k+1} + d_k - 2s_k) \tag{3.78}$$

$$b = h_k d_k - (y - y_k)(d_{k+1} + d_k - 2s_k) \tag{3.79}$$

$$c = -s_k(y - y_k) \tag{3.80}$$

This is an important benefit of the parametrization: while the flexibility is increased when compared to the quadratic polynomial, the inverse only requires solving a quadratic equation, which can be done with fewer numerical difficulties when compared to the cubic equation.

In general, a quadratic equation can have up to two roots. We are always interested in one for which the rational-quadratic has a *positive derivative*, as the function is restricted to be monotonic. To identify the root in a general case, note that the total derivative of a constant function is always zero. Hence if $q(x, y(x)) = 0$, then

$$\frac{\mathrm{d}q(x, y(x))}{\mathrm{d}x} = \frac{\partial q}{\partial x} + \frac{\partial q}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{3.81}$$

$$= \frac{\partial q}{\partial x} + \underbrace{\beta(\zeta(x))\frac{\mathrm{d}y}{\mathrm{d}x}}_{>0} = 0 \tag{3.82}$$

where $\frac{\mathrm{d}y}{\mathrm{d}x} > 0$ as the function is monotonic on the interval, and $\beta(\zeta(x)) > 0$ by its definition in Eq. (3.73). To satisfy the Eq. (3.82), $\frac{\partial q}{\partial x} > 0$ must hold for x to be a valid solution. When using the discriminant formula, the following root always has $\frac{\partial q}{\partial x} > 0$:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, \tag{3.83}$$

where the latter form is more numerically stable, especially when $a \to 0$ (near-linear case) and/or $c \to 0$.

3.3.3 Modeling unbounded data

In function parametrizations based on splines the input domain must be finite. This is a problem if we want to model an unbounded target distribution. Müller et al. recognize this, and propose a solution: we can map the unbounded input to some finite range using an invertible function, and use the inverse of the function to undo the mapping having applied the transformation. For example, when working in $[0, 1]$ they propose to use the *sigmoid* function and its inverse — the *logit* function. We can use the sigmoid function σ to map the input from $\mathbb{R}$ to $[0, 1]$, and then apply the $[0, 1] \to [0, 1]$ transformation f:

$$\tilde{y} = f(\tilde{x}), \quad \text{where} \quad \tilde{x} = \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.84}$$

Finally, we use σ^{-1} (the logit function) to map the output of f from $[0, 1]$ back to $\mathbb{R}$:

$$y = \sigma^{-1}(\tilde{y}) = \log \frac{\tilde{y}}{1 - \tilde{y}}. \tag{3.85}$$

While the approach is sound in theory, in our preliminary experiments we have observed it to be unstable due to numerical issues. In particular, in 32-bit floating point precision (common in GPUs) the tails of $\sigma(x)$ underflow for x outside of the approximate range of $[-13, 13]$. When $\sigma(x)$ underflows (i.e. the output becomes *exactly* 0 or 1) its gradient vanishes and the function becomes non-invertible, which means that the transformed density is ill-defined.

We propose an alternative solution, which is to use the *identity* transformation outside the range of the spline. This is equivalent to adding infinitely wide boundary bins or "tails" to the spline, where the transformation is $f(x) = x$. To preserve the first-order continuity, we match the derivatives at the boundary knots by setting $d_0 = d_{K-1} = f'(x) = 1$. This is trivial to do for both the cubic and the rational-quadratic splines, where d_0 and d_{K-1} are free parameters. The inverse of the tails is trivially the identity as well, and the derivative is a constant.

While the method above fixes the potential numerical issues of the Müller et al.'s approach, its obvious deficiency is that the transformation happens in a finite interval. However, we first note that this interval can be arbitrary: a spline that is defined on $[0, 1]$ can trivially be adapted to work on $[-B, B]$ for any $B > 0$:

$$y = B - 2B\tilde{y}, \quad \text{where} \quad \tilde{y} = f(\tilde{x}), \quad \text{where} \quad \tilde{x} = \frac{x + B}{2B}. \tag{3.86}$$

More importantly, however, if the spline autoregressive/coupling layer is preceded by an invertible affine layer, the affine layer can learn to map the input range that would most benefit from a non-trivial transformation to the required interval. In our experiments, we rely on both: we pick a B that we believe is appropriate for the range of the data, and use the invertible affine layers to allow the model to further adjust the effective range of the transformation if needed.

3.3.4 The proposed model

We combine the methods introduced above to obtain a class of models that we name *neural spline flows* (NSF). Such models use spline-based autoregressive/coupling layers with either quadratic (Q-NSF), cubic (C-NSF) or rational-quadratic (RQ-NSF) splines. Spline layers use the linear tails to model unbounded data, and each spline layer is preceded by an invertible linear layer. The proposed model resembles a standard feed-forward neural network: it is a sequence of linear transformations, each followed by an elementwise non-linear transformation, albeit the latter is *learned* and has a non-trivial dependency structure.

3.4 Experiments

The goal of the experiments in this section is to measure the performance of spline based normalizing flows, and compare them to other state-of-the-art normalizing flow architectures. In particular, we aim to understand whether the cubic and rational-quadratic splines improve upon the quadratic spline baseline. We use three benchmarks for our experiments: density estimation on tabular data, generative modeling of high-dimensional natural image data, and variational inference in a VAE.

We model the function $\mathbf{g}_\theta$ with a residual neural network (He et al., 2016a) with pre-activation residual blocks (He et al., 2016b). For autoregressive transformations we use the ResMADE architecture outlined by Nash and Durkan (2019). Preliminary results indicated only minor differences in setting the tail bound b within the range $[1, 5]$, so we fix a value $B = 3$ across experiments, and find this to work robustly. We also fix the number of bins $K = 8$ across our experiments unless stated otherwise, and apart from preliminary experiments where $K = 10$. We implement all invertible linear transformations using the PLU decomposition, sampling and fixing a random permutation matrix $\mathbf{P}$ at the beginning of training, and initializing the product $\mathbf{LU}$ to the identity. For all non-image experiments, we define a flow "step" as the composition of an invertible linear transformation with either a coupling or autoregressive transform, and use 10 steps per flow in all experiments, unless otherwise noted. All flows use a standard normal base distribution: $\pi(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbb{I})$.

We train the models using the Adam optimizer (Kingma and Ba, 2015), annealing the learning rate according to a cosine schedule (Loshchilov and Hutter, 2017). In some cases,

Table 3.1: Tabular density estimation results of quadratic and cubic spline flows. Test log likelihood (in nats) for UCI datasets and BSDS300; higher is better. Error bars correspond to two standard deviations.

Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Quadratic	0.65 ± 0.01	13.13 ± 0.02	-14.95 ± 0.02	-9.18 ± 0.43	157.49 ± 0.28
Cubic	0.65 ± 0.01	13.14 ± 0.02	-14.59 ± 0.02	-9.06 ± 0.44	157.24 ± 0.28

Table 3.2: Tabular density estimation results of Q-NSF and RQ-NSF. Test log likelihood (in nats) for UCI datasets and BSDS300. Error bars correspond to two standard deviations. FFJORD†, NAF†, Block-NAF†, and SOS† report error bars across repeated runs rather than across the test set. Superscript* indicates results are taken from the existing literature. For validation results which can be used for comparison during model development, see Table 3.7 in Section 3.6.1.

Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
FFJORD*†	0.46 ± 0.01	8.59 ± 0.12	-14.92 ± 0.08	-10.43 ± 0.04	157.40 ± 0.19
Glow	0.42 ± 0.01	12.24 ± 0.03	-16.99 ± 0.02	-10.55 ± 0.45	156.95 ± 0.28
Q-NSF (C)	0.64 ± 0.01	12.80 ± 0.02	-15.35 ± 0.02	-9.35 ± 0.44	157.65 ± 0.28
RQ-NSF (C)	0.64 ± 0.01	13.09 ± 0.02	-14.75 ± 0.03	-9.67 ± 0.47	157.54 ± 0.28
MAF	0.45 ± 0.01	12.35 ± 0.02	-17.03 ± 0.02	-10.92 ± 0.46	156.95 ± 0.28
NAF*†	0.62 ± 0.01	11.96 ± 0.33	-15.09 ± 0.40	-8.86 ± 0.15	157.73 ± 0.04
Block-NAF*†	0.61 ± 0.01	12.06 ± 0.09	-14.71 ± 0.38	-8.95 ± 0.07	157.36 ± 0.03
SOS*†	0.60 ± 0.01	11.99 ± 0.41	-15.15 ± 0.10	-8.90 ± 0.11	157.48 ± 0.41
Q-NSF (AR)	0.66 ± 0.01	12.91 ± 0.02	-14.67 ± 0.03	-9.72 ± 0.47	157.42 ± 0.28
RQ-NSF (AR)	0.66 ± 0.01	13.09 ± 0.02	-14.01 ± 0.03	-9.22 ± 0.48	157.31 ± 0.28

we find applying dropout (Srivastava et al., 2014) in the residual blocks to be beneficial for regularization. Full experimental details are provided in Section 3.6. Code reproducing the experiments is available online at `https://github.com/bayesiains/nsf`.

3.4.1 Density estimation

We begin with density estimation on tabular data. We use a battery of tabular datasets from the UCI repository (Dua and Karra Taniskidou, 2017) and BSDS3000 — a tabular dataset derived from natural images (Martin et al., 2001). We follow the experimental setup and pre-processing of Papamakarios et al. (2017), using the data provided by Papamakarios (2018).

Experiments we run for our preliminary paper compare the cubic spline to the quadratic spline when used in a coupling layer. In these experiments we use the sigmoid before and the logit after each coupling layer, as proposed by Müller et al. (2018). We also only use five flow layers for the Miniboone dataset to alleviate overfitting. The test likelihoods of both models are shown in Table 3.1.

As we can see, in a controlled comparison the cubic spline does improve upon the quadratic spline for most of the datasets. At the same time, we find that the cubic spline leaves a lot to be desired when it comes to numerical stability: careful tuning of optimization hyperparameters becomes critical to prevent numerical issues. Due to these issues we move on to the rational-quadratic for further experiments with tabular data.

In our main paper we aim to understand whether the rational-quadratic spline can also improve upon the quadratic spline, and whether it can do so without the additional numerical issues of the cubic. We perform a controlled comparison for both coupling layers and autoregressive layers. We also include state-of-the-art coupling/autoregressive flows in our comparison, and train two NSF baselines that use affine univariate functions instead of splines. We refer to the baseline models as Glow and MAF: NSF (C) with an affine transformation corresponds to Glow (Kingma and Dhariwal, 2018), while affine NSF (AR) is equivalent to MAF Papamakarios et al. (2017) with an addition of LU layers. Details of these experiments and final hyperparameters settings are provided in Section 3.6. The test likelihoods are shown in Table 3.2.

We see that the rational-quadratic also beats the quadratic, for both coupling layers and autoregressive layers. Moreover, we observe the rational-quadratic to be more numerically stable and less sensitive to hyperparameters than the cubic. Both rational-quadratic and quadratic spline models beat the affine baselines by a significant margin, exposing the limited flexibility of these models. Moreover, both spline models achieve *state-of-the-art results* for normalizing flow models on three out of five datasets. Surprisingly, spline flows that use the *coupling* layers beat state-of-the-art *autoregressive* models on some of the datasets. The latter finding goes against the common wisdom that autoregressive flows should be strictly more flexible than flows with coupling layers.

3.4.2 Image modeling

We now run the experiments on generative modeling of high-dimensional natural image datasets. In this section, we focus solely on flows with a cheap inverse that use coupling layers.

In our preliminary work, we fit a Glow-like cubic spline image model to FashionMNIST images (Xiao et al., 2017). The samples shown in Fig. 3.1 suggest that the trained model captures the target distribution well. However, as in tabular density estimation experiments, we find the cubic splines to be numerically unstable. In addition to issues during training, we have found it difficult to avoid numerical artifacts when sampling, which stemmed from numerical issues when solving the cubic equation. As a result, we again move on to the rational-quadratic splines for further experiments.

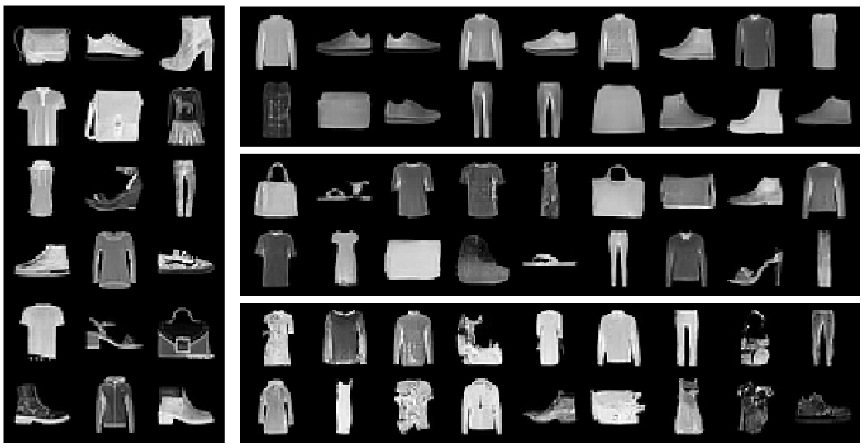


Figure 3.1: Fashion-MNIST samples from C-NSF (C) image models. **Left**: Training data. **Right**: Unconditional samples from the cubic-spline flow. We re-scale the standard normal base distribution during sampling, as common in literature. The standard deviation of the used spherical normal base distribution is referred to as *temperature*. Samples for three temperatures are shown in separate blocks, top-to-bottom: $0.5, 0.75, 1.0$.

To test the model on higher-dimensional, more complex datasets, we use the CIFAR-10 (Krizhevsky, 2009) and downsampled 64×64 ImageNet (van den Oord et al., 2016; Russakovsky et al., 2015) datasets, both with original 8-bit color depth and with reduced 5-bit color depth. We use Glow-like architectures with rational-quadratic coupling transforms. We compare our results quantitatively in terms of bits-per-dimension (BPD) to the state-of-the-art normalizing flow image model (Glow, Kingma and Dhariwal, 2018). We also train a baseline Glow model that uses affine transforms instead of rational-quadratic splines, but is otherwise equivalent to RQ-NSF (C). We provide all details for these experiments in Section 3.6. Quantitative results are shown in Table 3.3, and the samples are shown in Fig. 3.2.

Qualitatively, the samples from our model resemble the images in the training set: the high-level structure, colors and textures are all captured well, even though visual fidelity could be improved. Quantitatively, the RQ-NSF (C) image flow comfortably beats the baseline on three out of four datasets, with a marginal increase in the number of parameters. The baseline flow matches the RQ-NSF (C) performance on the CIFAR-10 5-bit dataset. We conjecture that this is the “easiest” dataset of the four, hence it benefits least from increased flow flexibility. In fact, we have experienced overfitting problems with this dataset, further indicating that the lack of expressivity is no longer the issue.

When compared to the state-of-the-art image flow (Glow), we see that RQ-NSF(Q) achieves competitive results, but does so using a much smaller number of layers, and an *order of magnitude* fewer parameters as a result. We are unable to run the RQ-NSF (C) with the

Table 3.3: Generative image modeling results for RQ-NSF (C). Test-set bits per dimension (*BPD*, lower is better) and parameter count (*Params*) for CIFAR-10 and ImageNet64 models. Superscript* indicates results taken from the existing literature.

	CIFAR-10 5-bit		CIFAR-10 8-bit		ImageNet64 5-bit		ImageNet64 8-bit	
Model	BPD	Params	BPD	Params	BPD	Params	BPD	Params
Baseline	1.70	5.2M	3.41	11.1M	1.81	14.3M	3.91	14.3M
RQ-NSF (C)	1.70	5.3M	3.38	11.8M	1.77	15.6M	3.82	15.6M
Glow*	1.67	44.0M	3.35	44.0M	1.76	110.9M	3.81	110.9M

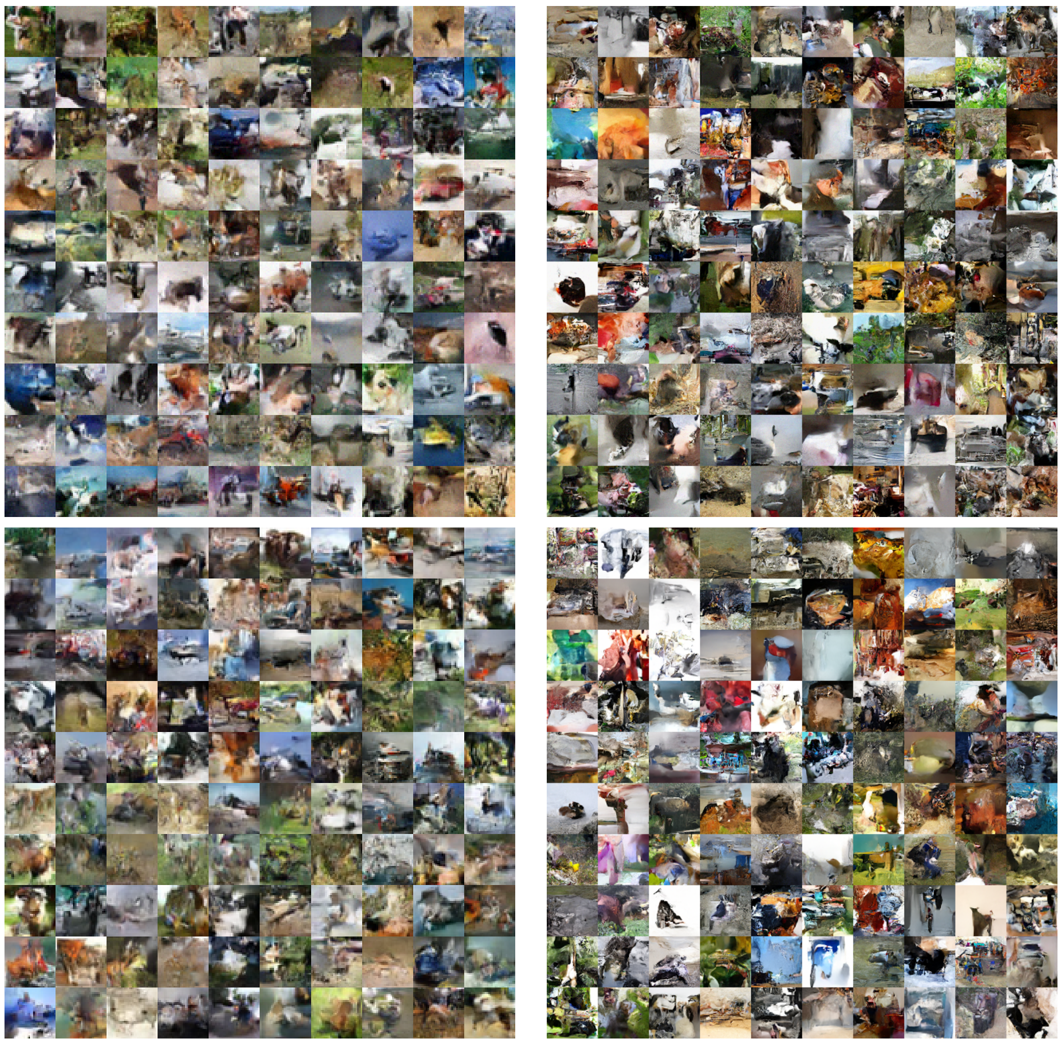


Figure 3.2: Samples from RQ-NSF (C) image models. **Top:** Reduced 5-bit color depth. **Bottom:** Original 8-bit color depth. **Left:** CIFAR-10. **Right:** ImageNet64.

number of parameters comparable to Glow due to lack of considerable GPU resources required.

Table 3.4: Variational autoencoder results of RQ-NSF. Test-set results (in nats) for the evidence lower bound (ELBO) and importance-weighted estimate of the log likelihood (computed as by Burda et al. (2016) using 1000 importance samples). Error bars correspond to two standard deviations.

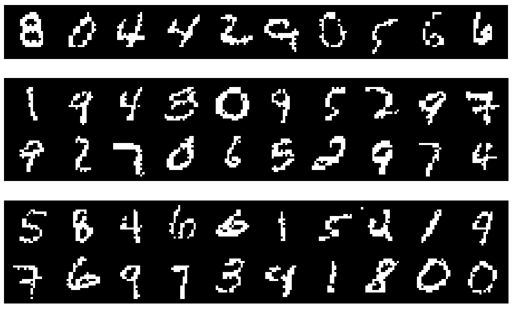
	MNIST		EMNIST	
Posterior/Prior	ELBO	$\log p(\mathbf{x})$	ELBO	$\log p(\mathbf{x})$
Baseline	-85.61 ± 0.51	-81.31 ± 0.43	-125.89 ± 0.41	-120.88 ± 0.38
Glow	-82.25 ± 0.46	-79.72 ± 0.42	-120.04 ± 0.40	-117.54 ± 0.38
RQ-NSF (C)	-82.08 ± 0.46	-79.63 ± 0.42	-119.74 ± 0.40	-117.35 ± 0.38
IAF/MAF	-82.56 ± 0.48	-79.95 ± 0.43	-119.85 ± 0.40	-117.47 ± 0.38
RQ-NSF (AR)	-82.14 ± 0.47	-79.71 ± 0.43	-119.49 ± 0.40	-117.28 ± 0.38

3.4.3 Variational autoencoder

Finally, we use the rational-quadratic spline flow to model the prior and the posterior in a VAE. In particular, we use the VAE to model the images in the MNIST (LeCun and Cortes, 2010) and the EMNIST (Cohen et al., 2017) datasets. We use the method of Salakhutdinov and Murray (2008) to *dynamically* binarize the images: we treat the (normalized) pixel intensities as Bernoulli probabilities, and sample binary values from the pixel-wise distributions dynamically during training. This allows us to use the Bernoulli likelihood in the VAE.

We evaluate using either the RQ-NSF (C) and the RQ-NSF (AR) as the prior and the posterior density models, comparing to the flows that use affine transformations instead. The affine flows in this case correspond to Glow (Kingma and Dhariwal, 2018) when using the coupling layer, and to IAF/MAF (Kingma et al., 2016; Papamakarios et al., 2017) when using the autoregressive layer. We also compare to the commonly used baseline where the prior is a standard normal and a diagonal normal is used to model the posterior. Further details of the VAE models evaluated are provided in Section 3.6. Quantitative results are provided in Table 3.4. We compare the models in terms of their evidence lower bound (ELBO), as well as importance-weighted estimate of their log likelihood (Burda et al., 2016). Samples from the trained VAEs are shown in Fig. 3.3, and demonstrate that the VAEs that use rational-quadratic spline flows have indeed captured the data distribution.

All of the VAEs that use normalizing flows improve upon the baseline that uses normal distributions. Using the normal distributions for the prior and posterior is still common in practice, and our results demonstrate a clear benefit of using more flexible density models. At the same time, in this case we see little benefit from using the rational-quadratic splines instead of simple affine transformations: all of the results are well within the error bars. Previous sections provide evidence that rational-quadratic spline flows are (sometimes


(a) MNIST

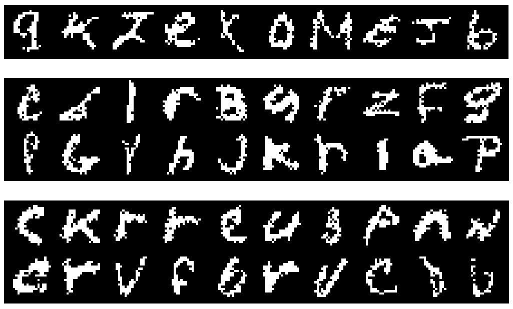

(b) EMNIST-letters

Figure 3.3: Samples from a VAE with a RQ-NSF prior and posterior. Top to bottom: training data, RQ-NSF (C), RQ-NSF (AR).

significantly) more flexible than affine flows, which is why we conjecture that the affine flows are sufficient to model the latent space in the VAE models considered, hence we do not benefit from increasing the flexibility further.

3.5 Discussion

Normalizing flows based on autoregressive/coupling layers allow us to exploit the expressive power of neural networks to build flexible, high-dimensional density estimators. However, we can not make full use of this expressive power if the neural networks parametrize elementwise transformations that themselves have limited flexibility. This is seen in our experimental results, where flows based on polynomial or rational-quadratic splines consistently outperform affine baselines.

RQ-NSF achieves state-of-the-art results on several tabular density estimation datasets. Moreover, when using the rational-quadratic spline in a coupling layer, the flows perform on-par with, and sometimes *outperform*, the autoregressive flows with simpler transformations. When compared with Glow (Kingma and Dhariwal, 2018), RQ-NSF requires an order of magnitude fewer parameters to achieve the same level of results. This suggests that RQ-NSF (C) approaches a *universal* density model: it has the expressivity of an autoregressive flow, yet also provides fast sampling, allowing to use RQ-NSF (C) in any density estimation context.

Increased flexibility is of most benefit when the target density is sufficiently complex, *and* when we have enough data to learn it. While RQ-NSF is strictly more flexible than Q-NSF, in tabular density estimation we observe the most significant improvement on Power, Gas, and Hepmass, the datasets with the highest ratio of the number of datapoints to dimensionality (see Tables 3.5 and 3.6). In generative image modeling, the improvement of RQ-NSF upon the affine baseline is most significant on the ImageNet dataset, which has an order of magnitude more datapoints than CIFAR-10, and contains many more object classes.

On CIFAR-10, and its 5-bit variant in particular, RQ-NSF model does not improve upon the affine baseline. We also observe no significant improvement when using RQ-NSF instead of an affine baseline in a VAE, which likely means that the prior/posterior is already well captured by this baseline. Nevertheless, with careful hyperparameter tuning, *at worst* RQ-NSF performs on-par with flows based on quadratic splines and affine transformations.

Despite achieving high likelihoods on image data, the image samples in Fig. 3.2 lack visual acuity. This is often the case with likelihood-based models (van den Oord et al., 2016; Kingma and Dhariwal, 2018; Ho et al., 2019), samples from which do not achieve the perceptual quality of e.g. GAN-based models (Brock et al., 2019). It is not clear if this is due to the likelihood-based models not being flexible enough and/or not having the right inductive bias, or due to the fundamental properties of likelihood-based training objectives, whose relationship with sample quality is a complex one (Theis et al., 2016). Running a RQ-NSF model of size comparable to Glow would be an interesting extension of our work, although one requiring considerable GPU resources.

At the same time, we have experienced issues with overfitting when modeling some of the image datasets with RQ-NSF, having to increase the dropout rate and reduce the model size considerably. This suggests that while RQ-NSF is a flexible model, it might not provide the right inductive bias for modeling natural image data. Ho et al. (2019) show that other aspects of the flow, such as the dequantization distribution and the used neural network architecture, are more important: Ho et al.'s model significantly outperforms Glow, with ablations showing that the use of variational dequantization and *self-attention* (Vaswani et al., 2017) layers bring the biggest improvement.

Flows based on cubic splines also demonstrate increased flexibility when compared to the quadratic splines, but simultaneously result in numerical issues, both during optimization and when inverting the flow for sampling. This suggests that increasing the order of a polynomial in a spline beyond quadratic is not a good way forward, and alternative approaches to increasing flexibility must be considered. Switching to rational functions proved fruitful, but other options likely exist.

Finally, we note that flexible, differentiable monotonic function parametrizations are also useful outside of density estimation. For example, in the context of supervised learning, they can be used to warp the output space of a Gaussian process (GP), allowing for non-Gaussian processes with non-Gaussian noise (Snelson et al., 2004). Alternatively, using invertible architectures can reduce the memory requirements of backpropagation, as we can recompute the activations instead of storing them (Gomez et al., 2017; MacKay et al., 2018). We hope our density estimation results together with these potential applications will motivate the community to propose novel monotonic function parametrizations.

3.5.1 `nflows`: normalizing flows in PyTorch

During this project we have developed a framework for building normalizing flow models in PyTorch (Paszke et al., 2019). The framework has been used to implement the flow models for experiments in (Durkan et al., 2019a) and (Durkan et al., 2019b). After the publication, the framework has been released as a standalone Python package under the name of `nflows` (`https://github.com/bayesiains/nflows`). Since the release of the package, Artur has been its primary maintainer. To date the package has amassed more than 500 "stars" on GitHub, and has been cited by at least 16 subsequent papers through its DOI (`https://doi.org/10.5281/zenodo.4296287`).

The package is based on three core concepts/classes: a `Distribution`, a `Transform`, and a `Flow`. The `Distribution` instances are the continuous probability distributions for which we can evaluate the (log) density, and which we can sample from. The `Transform` instances are invertible transformations for which we can evaluate the forward pass, the inverse pass, and the $\log |\det \mathbf{J}|$. Finally, the `Flow` is a subclass of `Distribution` that is defined as a combination of a (base) `Distribution` and a `Transform`.

The framework provides a set of `Transform` implementations, including affine/spline-based autoregressive and coupling transforms; invertible affine transforms that use LU, QR or SVD parametrization; activation normalization and 1×1 convolution from (Kingma and Dhariwal, 2018); etc. Moreover, we encourage the external contributors to add new types of transformations. For example, Antoine Wehenkel has recently worked with Artur to add the *unconstrained monotonic neural networks* (UMNN, Wehenkel and Louppe, 2019) to `nflows`.

Alternative frameworks for building normalizing flows exist. *TensorFlow Probability* (TFP, Dillon et al., 2017) provides a similar framework, with implementations of many bijective transformations. In the world of PyTorch, the probabilistic programming framework *Pyro* (Bingham et al., 2019) provides a set of distribution transformations, which are in the process of being factored out into a separate *FlowTorch* package (`https://flowtorch.ai`). All of the frameworks mentioned above now include the rational-quadratic spline implementation. At the time of working on NSF, however, these frameworks had a limited set of transformations implemented. Moreover, these more established frameworks tend to err on the side of generality when it comes to accepted tensor shapes, bijectivity requirements for transformations, domain constraints, etc. `nflows`, while less general, has a minimal API that makes it trivial to construct and train a state-of-the-art normalizing flow, which likely explains the continued interest in it.

3.6 Appendix: Experimental details

3.6.1 Tabular density estimation

Model selection is performed using the standard validation splits for these datasets. We clip the norm of gradients to the range $[-5, 5]$, and find this helps stabilize training. We modify MAF by replacing permutations with invertible linear layers. Hyperparameter settings are shown for coupling flows in Table 3.5 and autoregressive flows in Table 3.6. We include the dimensionality and number of training data points in each table for reference. For higher dimensional datasets such as Hepmass and BSDS300, we found increasing the number of coupling layers beneficial. This was not necessary for Miniboone, where overfitting was an issue due to the low number of data points.

Table 3.5: Hyperparameters for density-estimation results using coupling layers in Section 3.4.1.

	Power	Gas	Hepmass	Miniboone	BSDS300
Dimension	6	8	21	43	63
Train data points	1,615,917	852,174	315,123	29,556	1,000,000
Batch size	512	512	256	128	512
Training steps	400,000	400,000	400,000	200,000	400,000
Learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
Flow steps	10	10	20	10	20
Residual blocks	2	2	1	1	1
Hidden features	256	256	128	32	128
Bins	8	8	8	4	8
Dropout	0.0	0.1	0.2	0.2	0.2

Table 3.6: Hyperparameters for density-estimation results using autoregressive layers in Section 3.4.1.

	Power	Gas	Hepmass	Miniboone	BSDS300
Dimension	6	8	21	43	63
Train data points	1,615,917	852,174	315,123	29,556	1,000,000
Batch size	512	512	512	64	512
Training steps	400,000	400,000	400,000	250,000	400,000
Learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
Flow steps	10	10	10	10	10
Residual blocks	2	2	2	1	2
Hidden features	256	256	256	64	512
Bins	8	8	8	4	8
Dropout	0.0	0.1	0.2	0.2	0.2

3.6.2 Generative modeling of images

For image-modeling experiments we use a Glow-like model architecture introduced by Kingma and Dhariwal (2018, Section 3). This involves stacking multiple steps for each level

Table 3.7: Validation log likelihood (in nats) for UCI datasets and BSDS300, with error bars corresponding to two standard deviations.

Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
RQ-NSF (C)	0.65 ± 0.01	13.08 ± 0.02	-14.75 ± 0.06	-9.03 ± 0.43	172.51 ± 0.60
RQ-NSF (AR)	0.67 ± 0.01	13.08 ± 0.02	-13.82 ± 0.05	-8.63 ± 0.41	172.5 ± 0.59

in the multi-scale architecture of Dinh et al. (2017), where each step consists of an actnorm layer, an invertible 1×1 convolution and a coupling transform. For our RQ-NSF (C) model, we make the following modifications to the original Glow model: we replace affine coupling transforms with rational-quadratic coupling transforms, we go back to residual convolutional networks as used in RealNVP (Dinh et al., 2017), and we use an additional 1×1 convolution at the end of each level of transforms. The baseline model is the same as RQ-NSF (C), except that it uses affine coupling transforms instead of rational-quadratic ones. For CIFAR-10 experiments we do not factor out dimensions at the end of each level, but still use the squeezing operation to trade spatial resolution for depth.

For all experiments we use 3 residual blocks and batch normalization (Ioffe and Szegedy, 2015) in the residual networks which parameterize the coupling transforms. We use 7 steps per level for all experiments, resulting in a total of 21 coupling transforms for CIFAR-10, and 28 coupling transform for ImageNet64 (Glow models used by Kingma and Dhariwal (2018) use 96 and 192 affine coupling transforms for CIFAR-10 and ImageNet64 respectively).

We use the Adam (Kingma and Ba, 2015) optimizer with default β_1 and β_2 values. An initial learning rate of 0.0005 is annealed to 0 following a cosine schedule (Loshchilov and Hutter, 2017). We train for 100,000 steps for 5-bit experiments, and for 200,000 steps for 8-bit experiments. To track the performance of our models, we split off 1% of the training data to use as a development set. Due to the resource requirements of the experiments, we perform a limited manual hyper-parameter exploration. Final values are reported in Table 3.8.

We use a single NVIDIA Tesla P100 GPU card per CIFAR-10 experiment, and two such cards per ImageNet64 experiment. Training for 200,000 steps takes about 5 days with this setup.

Table 3.8: Hyperparameters for generative image-modeling results in Section 3.4.2.

Dataset		Batch Size	Levels	Hidden Channels	Bins	Dropout
CIFAR-10	5-bit	512	3	64	2	0.2
	8-bit	512	3	96	4	0.2
ImageNet64	5-bit	256	4	96	8	0.1
	8-bit	256	4	96	8	0.0

3.6.3 Improving the variational autoencoder

We use the Adam optimizer (Kingma and Ba, 2015) with default hyperparameters, annealing an initial learning rate of 0.0005 to 0 using a cosine schedule (Loshchilov and Hutter, 2017) over 150,000 training steps with batch size 256. We use a 'warm-up' phase for the KL divergence term of the loss, where the multiplier for this term is initialized to 0.5 and linearly increased to 1 over the first 10% of training. This modification initially reduces the penalty incurred by the approximate posterior in deviating from the prior, and similar schemes have been shown to improve VAE training dynamics (Rezende and Viola, 2018). Model selection is performed using a held-out validation set of 10,000 samples for MNIST, and 20,000 samples for EMNIST.

We use 32 latent features, and residual nets use 2 blocks, with 64 latent features for coupling layers, and 128 latent features for autoregressive layers. Both coupling and autoregressive flows use 10 steps. As with the tabular density-estimation experiments, we modify IAF (Kingma et al., 2016) and MAF (Papamakarios et al., 2017) by replacing permutations with invertible linear layers using an LU-decomposition. All NSF models use 8 bins. The encoder and decoder architectures are set up exactly as described by Nash and Durkan (2019), and are similar to those used in IAF (Kingma et al., 2016) and NAF (Huang et al., 2018).

Conditioning the approximate posterior distribution $q(\mathbf{z} \,|\, \mathbf{x})$ follows a multi-stage procedure. First, the encoder computes a context vector $\mathbf{h}$ of dimension 64 as a function of the input $\mathbf{x}$. This vector is then mapped to the mean and diagonal covariance of a Gaussian distribution in the latent space. Then, $\mathbf{h}$ is also given as input to the residual nets in each of the flow's coupling or autoregressive layers, where it is concatenated with the input $\mathbf{z}$, mapped to the required number of hidden features, and also used to modulate the additive update of each residual block with a sigmoid gate. We found this scheme to work well across experiments.

Chapter 4

Nested dropout flows: lower-dimensional structure in density estimation

In this chapter we consider the problem of learning a density given data that lie on or close to a lower-dimensional manifold. We first review existing work on normalizing flows that fit densities on manifolds, and then propose a training regime for a standard normalizing flow based on the *nested dropout* idea of Rippel et al. (2014). The additional nested dropout objective allows extracting a sequence of densities on manifolds from the full-dimensional flow fit to the data. Unlike the alternatives, our method defines a valid density in the ambient space, and does not rely on prior knowledge of the intrinsic data dimension. We experimentally demonstrate that our method captures lower-dimensional structure in the data, but also identify trade-offs.

This chapter is based on *Ordering Dimensions with Nested Dropout Normalizing Flows* (Bekasov and Murray, 2020), a paper presented at the Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models, a part of the Thirty-seventh International Conference on Machine Learning (ICML). The paper was selected for a spotlight presentation at the workshop.

This chapter extends the paper: we provide a broader background and a deeper review of related work, and include more experimental results that help evaluate the captured manifolds and understand the likelihood-reconstruction trade-off.

4.1 Introduction

In Chapter 3 we have introduced a normalizing flow as a method for parametrizing a complex, high-dimensional distribution via a sequence of invertible transformations of a simple base distribution. Unlike most other popular flexible generative models (e.g. Kingma and Welling, 2014; Goodfellow et al., 2014a; Du and Mordatch, 2019), flows have a tractable likelihood, which simplifies both training and model comparison. It is also easier to use tractable representations of distributions in other models and algorithms, for example when modeling the prior and the posterior in a *variational autoencoder* (VAE, Kingma and Welling, 2014).

We are often interested in capturing a density of structured, high-dimensional data, such as images or audio. It is common to assume that such data do not span the high-dimensional observation space, but lie on (or close to) a *lower-dimensional manifold*. In other words, we assume that the number of hidden *factors* that underlie each observation is smaller than the number of variables that represent the observation. For example, in images of handwritten digits in the MNIST dataset such factors include the position of the digit, its orientation and size, or the thickness of the stroke (Hinton et al., 1997).

The low-dimensional manifold assumption is directly built into many generative models. For example, in the *generative adversarial network* (GAN, Goodfellow et al., 2014a) the generative process is defined as a transformation of samples from a *low-dimensional* noise distribution. This restricts the density modeled by a GAN to lie on a low-dimensional manifold. A similar effect is achieved by prescribing a low-dimensional latent space in a VAE.

In the classic treatment of normalizing flows, we restrict the transformation between the base distribution (the *latent* space) and the data distribution (the *data* space) to be *bijective*, i.e. a one-to-one correspondence. Such transformation can only exist if the dimension of the latent space *matches* the dimension of the observation space. Moreover, if the base distribution itself is non-degenerate (has volume in the latent space), no *invertible* function can transform it to a distribution on a lower-dimensional manifold, because this requires collapsing the density. As a result, standard normalizing flows are ill-suited to modeling data on lower-dimensional manifolds.

We can use a more general form of the change-of-variables formula to extend the definition of a normalizing flow to *injective* transformations, i.e. transformations that are bijective only when restricted to a subset of points in the ambient space, for instance to points that lie on some low-dimensional manifold. When the manifold is *known*, Gemici et al. (2016) and Rezende et al. (2020) model a density in a lower-dimensional space using a standard normalizing flow, and embed the density in the observation space using a prescribed injective map. When the data manifold is *unknown* and/or can not be expressed analytically, Kumar et al. (2020) learn the injective flow transformation end-to-end, while Brehmer and Cranmer (2020) learn the injective map *and* a low-dimensional flow simultaneously.

The methods above fix the manifold dimension before training, and hence rely on prior knowledge of the intrinsic data dimension. In this work we propose an alternative method that simultaneously fits a *sequence* of injective flows, i.e. a sequence of lower-dimensional densities, each lying on a manifold of different dimension. Our methods trains a standard, *full-dimensional* flow, but induces an *ordering* on the latent variables, such that taking the first k of them will reconstruct the data with small square error. We induce such ordering using the *nested dropout* idea of Rippel et al. (2014). Having fit the flow, we recover a density on a k-dimensional manifold by sampling k latent variables.

For a linear flow, our approach recovers a solution to *principal components analysis* (PCA). For a non-linear flow, our experimental results suggest that the method also captures the principal modes of variation. Besides not requiring prior knowledge of the intrinsic data dimension, our method retains a valid density in the ambient space, does not require specialized normalizing flow architectures, and is simple to implement. At the same time, we note that the normalizing flow architecture has an impact on how effective the additional nested dropout objective can be, while the additional objective itself can have a negative effect on the likelihood of the full-dimensional flow. We hope that our work will motivate further research into flows that can naturally represent low-dimensional structure.

4.2 Background

We are often interested in modeling densities of high-dimensional data like images or audio. Naively modeling the density of such data with (say) a histogram is problematic due to the *curse of dimensionality.* Consider covering a unit hypercube with evenly spaced points, such that the distance between two adjacent points (*sampling precision*) is $\epsilon \in (0, 1)$. In this case ϵ^{-D} points are required to cover a D-dimensional hypercube.[1] For example, for $\epsilon = 0.01$ in 1D we only need $0.01^{-1} = 100$ points. For the same ϵ, if we go up to as few as 10 dimensions, we need $0.01^{-10} = 10^{20}$ points, an intractable number. Modern image datasets can have thousands of dimensions: small 32×32 pixel images with 3 channel color contain $32 \times 32 \times 3 = 3072$ dimensions.

This is why additional assumptions must be made about a high-dimensional density to make density estimation feasible. Often we make assumptions about the smoothness of the density, or the presence of symmetries/invariances in it (Papamakarios, 2019). In this

[1]The unit hypercube is split into a number of smaller non-overlapping hypercubes. Volume of a D-dimensional hypercube with side ϵ is ϵ^D, so the number of non-overlapping ϵ-sided hypercubes that fit into a unit hypercube is $1^D/\epsilon^D = 1/\epsilon^D = \epsilon^{-D}$.

chapter we focus on another common assumption: the one of *low intrinsic dimensionality*, or the *manifold hypothesis*.

4.2.1 Manifold hypothesis and density estimation

In machine learning, and generative modeling in particular, we often assume that each datapoint has a small number of underlying latent “factors”, where each factor can control a group of variables (Hinton et al., 1997). Another way to express the same assumption is to say that the data lie on a lower-dimensional, potentially non-linear *manifold* that is embedded in the higher-dimensional observation space, which is why we refer to this assumption as the *manifold hypothesis* (Bengio et al., 2013, Section 8).

The manifold hypothesis is typically implemented by using information bottlenecks in machine learning model architectures. For example, if the hidden dimension of an autoencoder architecture is set to be smaller than the data dimension, the output from the decoder is forced to lie on a manifold of corresponding dimension (Kramer, 1991).

When it comes to density estimation, employing the manifold hypothesis is difficult. This is because the density of data that lie on a lower-dimensional manifold is *ill-defined* in the ambient space. To see why, we use the definition of a probability density at a point $\mathbf{x}$

$$p(\mathbf{x}) = \lim_{V(\Delta(\mathbf{x}))\to 0} \frac{P(\Delta(\mathbf{x}))}{V(\Delta(\mathbf{x}))}, \tag{4.1}$$

where $\Delta(\mathbf{x})$ is a continuous region centered at $\mathbf{x}$, $P(\mathcal{A})$ is the probability of a sampled a point landing in a region $\mathcal{A}$, and $V(\mathcal{A})$ is the volume of a region $\mathcal{A}$. In other words, $p(\mathbf{x})$ is the probability of a sample landing in an infinitesimal region in the vicinity of $\mathbf{x}$. This limit must be finite for a density to integrate to one while staying positive. However, if $\mathbf{x}$ are restricted to a lower-dimensional manifold, we can take a region $\Delta(\mathbf{x})$ *on the manifold*, in which case $V(\Delta(\mathbf{x})) = 0$ and $P(\Delta(\mathbf{x})) > 0$, and hence $p(\mathbf{x}) = \infty$. In other words, there exist regions that have zero volume in the ambient space, but that contain non-zero probability mass. For points off the manifold, on the other hand, $p(\mathbf{x}) = 0$, due to $P(\Delta(\mathbf{x})) = 0$.

A standard, “ambient” normalizing flow can never represent such a degenerate density: the transformation that turns a density defined in the ambient space into a density on a manifold *can not be invertible*: it requires collapsing probability density into regions with zero volume, which is a lossy transformation. At most, the flow can fit a smooth approximation of such a low-dimensional density, and we are likely to experience numerical issues during training as the optimization pushes the flow towards non-invertible configurations.

4.2.2 Representation learning

While the discussion above has focused on using the manifold hypothesis as a way to make density estimation tractable for high-dimensional problems, learning manifolds has a useful side-effect. Having learned the low-dimensional underlying factors for the data, we can use them as *representations* of data in downstream tasks. For example, we can train a classifier in the representation space instead of the high-dimensional data space, which could potentially simplify the classification problem (Bengio et al., 2013). This approach is particularly useful in problems where little labeled data is available, as the representations could be learned on unlabeled data.

Representation learning, and manifold learning in particular, are important subfields of machine learning, with a long history (Cayton, 2005) and recent developments (van den Oord et al., 2018; Chen et al., 2020). Normalizing flows are rarely the method of choice when representation learning is the primary objective, where simpler methods like deterministic autoencoders work well, and are easier to implement and train. Like representation learning methods based on VAEs and GANs, however, an advantage of normalizing flows is that they, besides providing representations, can *sample* in the learned latent space. Looking at the samples is an easy way to debug the learned representations, verifying that they indeed capture the principal factors of variation in the data.

The continuous manifold hypothesis is only one possible assumption that is useful for learning representations. For example, *natural clustering* is an assumption that the data is split into separate groups or categories, in which case group identity is an important underlying factor. Other possible assumptions include factor sparsity, disentanglement of factors, or simplicity of factor dependencies (Bengio et al., 2013). Making normalizing flows work with assumptions beyond the manifold hypothesis is an exciting direction for future research.

4.2.3 Normalizing flows on manifolds

The standard normalizing flow setup can be adapted to allow modeling data on lower-dimensional manifolds. While the density in the ambient space is not well-defined, as discussed above, we attempt to model the density of the subset of points that lie on the manifold. In other words, for a particular manifold $\mathcal{M} \subset \mathbb{R}^D$ the density is only defined for points $\mathbf{x} \in \mathcal{M}$, is undefined for other points in $\mathbb{R}^D$, and is only normalized on the manifold, *not* on $\mathbb{R}^D$:

$$\int_{\mathcal{M}} p(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1. \tag{4.2}$$

Flows on manifolds have to employ an alternative change-of-variables formula, one which, unlike Eq. (3.7), is not restricted to bijective transformations. Let $\mathbf{f}\colon \mathcal{Z} \to \mathcal{X}$ be an *injective* (one-to-one, but not bijective) differentiable mapping between a lower-dimensional latent space $\mathcal{Z}$ and the data space $\mathcal{X}$, i.e. $\mathbf{x} = \mathbf{f}(\mathbf{z})$ for $\mathbf{x} \in \mathcal{X}$ and $\mathbf{z} \in \mathcal{Z}$. The function $\mathbf{f}$ transforms an infinitesimal hypercube $\mathrm{d}\mathbf{z}$ at point $\mathbf{z} \in \mathcal{Z}$ to a corresponding *parallelepiped* on the manifold embedded in $\mathcal{X}$. The volume of this parallelepiped is $\det\Big(\mathbf{J}_\mathbf{f}(\mathbf{z})^\top \mathbf{J}_\mathbf{f}(\mathbf{z})\Big)^{\frac{1}{2}}$, where $\mathbf{J}_\mathbf{f}$ is the Jacobian of the mapping $\mathbf{f}$ (Ben-Israel, 1999). The density of points on the manifold is then defined as

$$p(\mathbf{x}) = \pi(\mathbf{z}) \det\Big(\mathbf{J}_\mathbf{f}(\mathbf{z})^\top \mathbf{J}_\mathbf{f}(\mathbf{z})\Big)^{-\frac{1}{2}}, \tag{4.3}$$

where $\mathbf{z}$ is the unique point for which $\mathbf{x} = f(\mathbf{z})$, and π is the base density on $\mathcal{Z}$. The density $p(\mathbf{x})$ is only defined for points on the manifold $\mathbf{x} \in \mathcal{M}$, where $\mathcal{M} = \{\mathbf{f}(\mathbf{z}) : \mathbf{z} \in \mathcal{Z}\}$, and is only normalized on the same manifold.

When $\mathbf{f}$ is bijective, a property of a square determinant matrix $\mathbf{J}_\mathbf{f}(\mathbf{z})$ implies that

$$\det\Big(\mathbf{J}_\mathbf{f}(\mathbf{z})^\top \mathbf{J}_\mathbf{f}(\mathbf{z})\Big) = \det(\mathbf{J}_\mathbf{f}(\mathbf{z}))^2, \tag{4.4}$$

and, as expected, we recover Eq. (3.7) from Eq. (4.3):

$$\begin{aligned} p(\mathbf{x}) &= \pi(\mathbf{z}) \big(\det \mathbf{J}_\mathbf{f}(\mathbf{z})^2\big)^{-\frac{1}{2}} && (4.5)\\ &= \pi(\mathbf{z}) \left|\det \mathbf{J}_\mathbf{f}(\mathbf{z})^{-1}\right| && (4.6)\\ &= \pi\big(\mathbf{f}^{-1}(\mathbf{x})\big) \left|\det \mathbf{J}_{\mathbf{f}^{-1}}(\mathbf{x})\right|. && (4.7) \end{aligned}$$

Eq. (4.3) means that we can model a density on a manifold with a normalizing flow by parametrizing an *injective* transformation $\mathbf{f}_\theta$ which transforms a lower-dimensional base density π into a density $p(\mathbf{x})$ on a manifold embedded in the data space. While this approach is sound in theory, implementing it in practice is difficult. In particular, evaluating Eq. (4.3) is expensive, even for transformations with a cheap $\det \mathbf{J}_\mathbf{f}$. This is a major concern, as tractable likelihood is an important feature of standard normalizing flow models.

In subsequent sections we review several methods that aim to overcome this difficulty and fit normalizing flows on manifolds. We start with methods that learn densities on *prescribed*, analytic manifolds, and move on to methods that simultaneously *learn* the manifold, using architectures and/or approximate optimization objectives to make training tractable.

4.2.4 Prescribed manifolds

Gemici et al. (2016) use Eq. (4.3) to fit a normalizing flow on the sphere[2] $\mathbb{S}^D \subset \mathbb{R}^{D+1}$. The

[2]Here and in the rest of the chapter we use the term "sphere" to refer to the $(D-1)$-dimensional surface

authors define an injective[3] function $\mathbf{f}$ that maps the Euclidean space $\mathbb{R}^D$ to the sphere $\mathbb{S}^D \subset \mathbb{R}^{D+1}$. The function $\mathbf{f}^{-1}$ (defined for points on the sphere) is used to project an analytic base density on $\mathbb{S}^D$ (e.g. uniform) to a density on $\mathbb{R}^D$. Any bijective transformation $\mathbf{g}\colon \mathbb{R}^D \to \mathbb{R}^D$ could then be used to transform the simple base density to a potentially complex, multi-modal density. Finally the authors use $\mathbf{f}$ to project the enriched density back to $\mathbb{S}^D$.

The approach of Gemici et al. allows fitting a normalizing flow on any pre-defined manifold, as soon as the map $\mathbf{f}$ is known. However, if the manifold prescribed by $\mathbf{f}$ is not *homeomorphic* to $\mathbb{R}^D$, the method is prone to numerical instability. In fact, $\mathbb{S}^D$ is *not* homeomorphic to $\mathbb{R}^D$, hence the $\mathbf{f}$ proposed by Gemici et al. cannot be well-behaved for all points on the manifold. The inverse of $\mathbf{f}$ is equivalent to the well-known *stereographic projection* of the unit hypersphere:

$$\mathbf{f}^{-1}(\mathbf{x}) = \frac{\mathbf{x}_{1:D}}{1 - \mathbf{x}_{D+1}}, \tag{4.8}$$

and it will not be well-defined for some points $\mathbf{x} \in \mathbb{S}^D \subset \mathbb{R}^{D+1}$ on the sphere: the output will approach infinity as $\mathbf{x}_{D+1} \to 1$.

Rezende et al. (2020) consider an alternative that avoids said numerical issues. They first propose robust methods based on spline flows (Chapter 3) for modeling densities on the circle $\mathbb{S}^1$, and use them to build flows on the sphere $\mathbb{S}^D$ *recursively*. At each step of the recursion, like Gemici et al. (2016), a transformation is applied in an intermediate space, but instead of $\mathbb{R}^D$ the authors use the *cylinder* $\mathbb{S}^{D-1} \times [-1, 1]$. Analytic maps $T_{s\to c}$ and $T_{c\to s}$ are defined that transform the sphere to the cylinder and back, respectively, which the authors show to be more numerically stable than equivalent transformations between the sphere and the Euclidean space. The density on the cylinder is modeled autoregressively using two flows: a 1-dimensional flow on the cylinder height $h \in [-1, 1]$, and a conditional flow on the sphere $\mathbb{S}^{D-1}$. The latter is itself defined via a density on a lower-dimensional cylinder, continuing the recursion all the way down to $\mathbb{S}^1$.

Rezende et al. also parameterize a density on a D-dimensional torus $\mathbb{T}^D$ as a product of conditional densities on $\mathbb{S}^1$. Their methods are less prone to numerical problems than those proposed by Gemici et al., but extending them to other types of manifolds is less trivial than simply writing down a manifold map $\mathbf{f}$.

of a D-dimensional ball.

[3]Gemici et al. refer to $\mathbf{f}$ as a *homeomorphism*: a bijective, continuous function that also has a continuous inverse (is *bicontinuous*). Note that $\mathbf{f}$ is only a homeomorphism when treated as a transformation between $\mathbb{R}^D$ and $\mathbb{S}^D$. When viewed as a transformation between $\mathbb{R}^D$ and $\mathbb{R}^{D+1}$ it is injective, not bijective.

Continuous manifold flows

Using *continuous* normalizing flows presents an attractive alternative when it comes to density estimation on prescribed manifolds. Consider a continuous normalizing flow based on a *neural ordinary differential equation* (Neural ODE, Chen et al., 2018):

$$\frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathbf{f}_\theta(\mathbf{h}(t), t) \tag{4.9}$$

Given the input $\mathbf{z}$, we set the initial condition $\mathbf{h}(0) = \mathbf{z}$, and use a numerical ODE solver to compute $\mathbf{x} = \mathbf{h}(1)$. Under some mild smoothness assumptions on $\mathbf{f}_\theta$, the resulting $\mathbf{z} \to \mathbf{x}$ transformation is guaranteed to be invertible (Chen et al., 2018). Moreover, $\mathbf{f}_\theta$ itself does *not* need to be bijective, meaning that a free-form neural network can be used to parametrize it.

Mathieu and Nickel (2020), Lou et al. (2020) and Falorsi and Forré (2020) have all concurrently proposed to extend the setup above to model densities on prescribed manifolds. The core idea is to use some analytic base density $\pi(\mathbf{z})$ on the manifold (e.g. a uniform density on $\mathbb{S}^D$), and restrict the output of $\mathbf{f}_\theta$ to lie in the *tangent space* of said manifold. When paired with a manifold-aware ODE solver (e.g. a *projective* solver that projects each step onto the manifold), $\mathbf{x}$ will be restricted to stay on the presribed manifold, and will have a well-defined density on it. The main benefit of continuous manifold flows when compared to the alternatives is that they only require a *local* map of the manifold, not a *global* one. The latter, as discussed above, can be difficult to define for some manifolds without running into numerical difficulties.

4.2.5 Learned manifolds

Methods above assume that we know the data manifold a priori and can define it analytically. This is the case in applications where the variables are representing, for example, angles, axes or directions (Mardia and Jupp, 2009). For example, directions from earth to stars could be represented as points on the sphere $\mathbb{S}^2 \subset \mathbb{R}^3$, if the distances to the stars are ignored or unknown. We can use the methods above to model a probability density of such directions.

Unfortunately, often we only know that there is *some* manifold structure in the problem, but do not know what it is, and/or cannot define it analytically. This is the case with the manifold of natural images, which is a commonly used benchmark for generative models. Both Kumar et al. (2020) and Brehmer and Cranmer (2020) propose methods that allow *learning* a potentially complex, non-linear manifold *and* a density on that manifold at the same time.

Injective flows

Kumar et al. (2020) parameterize the transformation $\mathbf{f}$ with a neural network, attempting to train a *generative neural sampler* (Goodfellow et al., 2014a; Nowozin et al., 2016) with a lower-dimensional latent space. The authors keep $\mathbf{f}$ (a neural network with an arbitrary architecture) injective by penalizing solutions where the singular values of $\mathbf{J_f}$ approach zero. This is achieved by penalizing low values of $\|\mathbf{J_f v}\|_2$ for non-zero vectors $\mathbf{v}$, effectively discouraging settings of $\mathbf{J_f}$ for which $\mathbf{J_f v} = \mathbf{0}$ for some non-zero vector $\mathbf{v}$.

Training $\mathbf{f}$ by optimizing Eq. (4.3) exactly is also too computationally expensive: it involves computing a $D \times K$ Jacobian $\mathbf{J_f}$ for a potentially large neural network. In automatic differentiation frameworks computing such $\mathbf{J_f}$ involves K iterations of backpropagation. Moreover, computing the determinant of the resulting $D \times D$ matrix is an $\mathcal{O}(D^3)$ operation. Instead, Kumar et al. derive a *lower-bound* on the log likelihood, which provides a tractable training objective. In addition, to avoid solving the inverse problem for each $\mathbf{x}$, i.e. finding $\mathbf{z}$ s.t. $\mathbf{x} = \mathbf{f}(\mathbf{z})$, the authors train an *encoder* $\mathbf{h}\colon \mathcal{X} \to \mathcal{Z}$ s.t. $\mathbf{f}(\mathbf{h}(\mathbf{x})) \approx \mathbf{x}$ for points $\mathbf{x}$ on the manifold.

Following the required approximations, the training regime of Kumar et al. resembles the one of a deterministic *regularized autoencoder* (RAE, Ghosh et al., 2020). Both injective flows and RAEs, beyond the standard reconstruction error loss, regularize the latent space of the encoder, and encourage the *smoothness*/injectivity of the generator/decoder $\mathbf{f}$ by penalizing the Jacobian norm or an approximation to it.

In the end, by making the flow injective, we lose a tractable analytic inverse of $\mathbf{f}$, and effectively arrive at a different model class. At the same time, this is also the case for certain standard normalizing flow architectures, where we have to resort to numerical methods (like bijective search or fixed-point iteration) to compute the inverse (Huang et al., 2018; van den Berg et al., 2018; Behrmann et al., 2019; Wehenkel and Louppe, 2019). If we parametrize $\mathbf{f}$ (the noise-to-data transformation) using these flow architectures, this allows for cheap sampling, but density evaluation is expensive, same as with injective flows.

Manifold-learning flows

Brehmer and Cranmer (2020) propose an alternative way of parametrizing a flexible injective transformation $\mathbf{f}$. They parametrize a bijective transformation in the ambient space $\mathbf{g}\colon \mathbb{R}^D \to \mathbb{R}^D$ with a standard normalizing flow. The transformation $\mathbf{f}$ is then defined by padding the input to $\mathbf{g}$ with zeros, which is a simple way to push the lower-dimensional point $\mathbf{z}$ into the ambient space $\mathbb{R}^D$:

$$\mathbf{x} = \mathbf{f}(\mathbf{z}) = \mathbf{g}\Big([\mathbf{z}\ \mathbf{0}]^\top\Big), \tag{4.10}$$

where $\mathbf{z} \in \mathbb{R}^K$ and $[\mathbf{z}\ \mathbf{0}]^\top \in \mathbb{R}^D$. Such a transformation $\mathbf{f}$ is injective by construction (see Section 4.3.2), hence additional penalties on non-injective configurations are not needed. For points $\mathbf{x}$ on the manifold, $\mathbf{g}^{-1}$ can be used to *exactly* solve the inverse problem, i.e. find $\mathbf{z}$ s.t. $\mathbf{x} = \mathbf{f}(\mathbf{z})$, but the authors also evaluate training a separate encoder as done by Kumar et al. (2020).

Like in the method by Gemici et al. (2016), additional bijective transformations in the latent space can be used to enrich the density on the manifold, which is cheaper than making $\mathbf{f}_\theta$ more expressive, because it is done in the lower-dimensional space $\mathcal{Z}$. The approximate lower bound on Eq. (4.3) by Kumar et al. (2020) could be used for efficient likelihood evaluation and training, but Brehmer and Cranmer also discuss other possible approximations.

Brehmer and Cranmer propose a way of dealing with points *off* the manifold. Any point $\mathbf{x}$ can be projected onto the manifold by computing $\tilde{\mathbf{z}} = \mathbf{f}^{-1}(\mathbf{x})$, taking the corresponding dimensions $\mathbf{z} = \tilde{\mathbf{z}}_{1:K}$, and reconstructing $\tilde{\mathbf{x}} = \mathbf{f}(\mathbf{z})$. This way we can obtain "denoised" datapoints $\tilde{\mathbf{x}}$, or use the reconstruction error $\|\mathbf{x} - \tilde{\mathbf{x}}\|$ as a metric for detecting anomalous or out-of-distribution points.

At the same time, Brehmer and Cranmer point out the fundamental problem with training manifold flows by maximizing the likelihood on the manifold as the sole objective. The authors argue that comparing two densities that lie on *different* manifolds is fundamentally meaningless. As an extreme example, they consider the situation where the learned manifold is perpendicular to the true manifold. If this were the case, all data would be projected to a small region of high density, allowing for large likelihoods even without discovering the true manifold at all. To alleviate such pathological scenarios Brehmer and Cranmer augment the likelihood objective with additional reconstruction error, resulting in the overall objective not unlike the one that Kumar et al. (2020) arrive at by putting an approximate lower-bound on the likelihood objective.

Kim et al. (2020) propose *SoftFlow*, a method that learns a density of *noisy* data, conditioned on the level of the noise. In particular, Kim et al. push the data into the ambient space by adding noise to it, where the noise magnitude itself is a random variable:

$$\tilde{\mathbf{x}}_i \sim \mathcal{N}\big(\mathbf{x}_i, c_i^2 \mathbb{I}\big), \qquad c_i \sim \mathcal{U}(a, b). \tag{4.11}$$

They then train a conditional flow to match $p(\tilde{\mathbf{x}} \,|\, c)$, i.e. learn the density of the noisy data $\tilde{\mathbf{x}}$ conditioned on the known magnitude of the noise c. In other words, Kim et al. learn a sequence of flows that model increasingly noisy versions of the dataset. To sample data on the manifold, they propose to sample from $p(\tilde{\mathbf{x}} \,|\, c)$ with a small c, or even $c = 0$. SoftFlow demonstrates good generative performance on synthetic 2D data and 3D point clouds, but it is not clear whether these results generalize to higher-dimensional density estimation problems. Moreover, when setting $c = 0$, we are still asking a bijective flow to represent a density with no volume in the ambient space, something that it can not do *exactly*, as discussed in Section 4.2.1.

4.3 Nested dropout flows

In this section we propose *nested dropout flows* (NDF), a training regime for a normalizing flow that allows extracting low-dimensional densities from the fit full-dimensional density. We first review *nested dropout* (Rippel et al., 2014), a method for ordering dimensions in a standard autoencoder, and show how nested dropout can also be used to order latent dimensions in a normalizing flow. We then compare and contrast NDF with alternative manifold normalizing flows.

4.3.1 Nested dropout

The proposed method relies on the idea of *nested dropout*. Nested dropout was introduced by Rippel et al. (2014) in the context of autoencoders, with a goal of imposing an ordering on representation dimensions, such that information is pushed into earlier dimensions in the ordering.

As in standard dropout (Srivastava et al., 2014), nested dropout randomly "drops" (i.e. zeros out) representation dimensions during training, but only does so for the output of the encoder, not for its intermediate activations. Moreover, rather than dropping features independently, we sample an index $k \sim p_k(\cdot)$, and drop dimensions $k+1 \dots K$, i.e. all dimensions above that index, where K is the latent dimension.

We define $\mathbf{z}_{\downarrow k} = \begin{bmatrix} \mathbf{z}_{1:k} & \mathbf{0}_{(D-k)} \end{bmatrix}^\top$ as a representation $\mathbf{z}$ where all dimensions above the k-th have been dropped. We then define a low-dimensional reconstruction of a given datapoint $\mathbf{x}$ for a given dimension k as

$$\hat{\mathbf{x}}_{\downarrow k} = \mathbf{g}_\theta(\mathbf{f}_\theta(\mathbf{x})_{\downarrow k}), \tag{4.12}$$

where $\mathbf{f}_\theta$ and $\mathbf{g}_\theta$ are the encoder and the decoder, respectively, with parameters θ.

To train the autoencoder for a given dataset $\{\mathbf{x}_n\}_{n=1}^N$, we minimize the following objective:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{k \sim p_k}[d(\mathbf{x}_n, \hat{\mathbf{x}}_{n\downarrow k})], \tag{4.13}$$

where d is the chosen distance metric, typically L_2 distance. This setup encourages the autoencoder to put more information into dimensions that correspond to lower indices, as shown both theoretically and empirically by Rippel et al. (2014).

The theoretical analysis of Rippel et al. applies to any sampling distribution p_k, as soon as it has support over all latent indices. The original work uses a geometric distribution, parametrized as $p_k(k) = (1-p)^{k-1}p$ for a given Bernoulli probability $p \in (0, 1]$, which is a hyper-parameter. We follow the choice of Rippel et al. in this work and use a geometric distribution, but note that in a different application of nested dropout Xu et al. (2021) have observed limited effect when using an alternative (uniform) sampling distribution.

Rippel et al. show that when used with a *linear autoencoder* (LAE), nested dropout recovers principal components of the data, i.e. is equivalent to PCA. In their more recent work, Bao et al. (2020) show that the convergence of a gradient-based optimizer is slow when training the LAE with nested dropout, explained by the fact that the Hessian of the loss is ill-conditioned when close to the global minimum. Bao et al. use a *deterministic* variant of nested dropout with a LAE, where instead of sampling from p_k they minimize the *expected* reconstruction error with respect to it. This expectation has an analytic form for the LAE, which allows reducing the variance of the gradients. Bao et al. also explore an alternative representation regularization scheme that recovers PCA: *non-uniform* L_2, but it also suffers from slow convergence when used with a LAE. Experimenting with the non-uniform L_2 and other potential representation regularization schemes would be an interesting extension of our work.

Characterizing the solution when using nested dropout with a *non-linear* autoencoder is more difficult. Nevertheless, Rippel et al. demonstrate empirically that the learned dimension ordering is useful for several downstream tasks. In particular, it allows performing efficient *retrieval* by constructing a binary *trie/prefix tree* from binarized, ordered latent representations of the items in a database. It also allows performing *adaptive* lossy compression, where a compressed item can be decoded *progressively* for an arbitrary number of bits received.

4.3.2 Nested dropout flows

We now apply nested dropout to normalizing flows, aiming to replicate the alignment of the latent space observed by Rippel et al. (2014). We start with the standard training objective of a normalizing flow, which is to maximize the likelihood of its parameters given in Eq. (4.7). For a given a dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$, we fit the flow parameters θ by minimizing the negative log likelihood objective:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \tag{4.14}$$

$$= -\sum_{i=1}^{N} \log \pi\big(\mathbf{f}_\theta^{-1}(\mathbf{x}_i)\big) + \log \left|\det \mathbf{J}_{\mathbf{f}_\theta^{-1}}(\mathbf{x}_i)\right|. \tag{4.15}$$

We can redefine a lower-dimensional reconstruction in Eq. (4.12) for a normalizing flow by treating the flow as a type of an autoencoder where the function $\mathbf{f}_\theta^{-1}$ is the invertible "encoder", and its inverse $\mathbf{f}_\theta$ is the invertible "decoder". The reconstruction is then defined as

$$\tilde{\mathbf{x}}_{\downarrow k} = \mathbf{f}_\theta\big(\mathbf{f}_\theta^{-1}(\mathbf{x})_{\downarrow k}\big). \tag{4.16}$$

We combine Eqs. (4.13), (4.14) and (4.16) into a novel objective, which specifies that the flow should achieve high log likelihood, but also give good reconstructions from low-dimensional parts of its representation:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} -\log p_\theta(\mathbf{x}_i) + \lambda\, \mathbb{E}_{k\sim p_k}[d(\mathbf{x}_i, \tilde{\mathbf{x}}_{i\downarrow k})], \tag{4.17}$$

where λ is a hyper-parameter that balances the two losses. Intuitively, we optimize the flow to match the target density in the data space *and* align the latent space to ensure that each of k low-dimensional manifolds is close to the data. We call the normalizing flow trained with this objective a *nested dropout flow* (NDF).

In line with Rippel et al., we estimate the expectation in Eq. (4.17) with a single Monte Carlo sample: for each image $\mathbf{x}_i$ in a batch, we sample a single value of $k \sim p_k$, project the image onto the corresponding manifold, and compute the reconstruction error. Using multiple samples of k would reduce the variance of the estimator, but will considerably slow down the training: a full pass through the flow transformation is required to compute $\tilde{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{z}_{\downarrow k})$ for each k, even though $\mathbf{z} = \mathbf{f}_\theta^{-1}(\mathbf{x})$ could be cached and re-used for all samples.

Having trained the NDF, to sample from the k-dimensional manifold we define a function $\mathbf{h}_{\theta,k} : \mathcal{Z}_k \rightarrow \mathcal{X}$ that uses the trained $\mathbf{f}_\theta$ to transform samples from a lower-dimensional base distribution to samples on the manifold:

$$\mathbf{x} = \mathbf{h}_{\theta,k}(\tilde{\mathbf{z}}) = \mathbf{f}_\theta\Big([\tilde{\mathbf{z}}\ \mathbf{0}]^\top\Big), \quad \text{where} \quad \tilde{\mathbf{z}} \sim \pi_k(\tilde{\mathbf{z}}), \tag{4.18}$$

and π_k is the k-dimensional marginal base distribution. When π is a standard normal, π_k is simply a k-dimensional standard normal. As $(D-k)$ dimensions of the input to $\mathbf{f}_\theta$ are set to a constant (zero), the sample $\mathbf{x}$ is guaranteed to lie on a k-dimensional manifold, a manifold that the nested dropout objective pushes towards the data during training. We recover the standard, ambient flow when $k = D$. For $k < D$, the samples will come from a valid density on the k-dimensional manifold, as the transformation $\mathbf{h}_{\theta,k}$ is *injective* (one-to-one, but not a correspondence).

To show that $\mathbf{h}_{\theta,k}$ is indeed injective, we note that the zero-padding operation in Eq. (4.18) is invertible: if $\mathbf{z} = [\tilde{\mathbf{z}}\ \mathbf{0}]^\top$, then $\tilde{\mathbf{z}} = \mathbf{z}_{\downarrow k}$, meaning that each sample $\tilde{\mathbf{z}}$ is associated with a unique $\mathbf{z}$. Each $\mathbf{z}$, in turn, is associated with a unique $\mathbf{x}$, as $\mathbf{f}_\theta$ is a bijection between $\mathcal{Z}$ and $\mathcal{X}$. As a result, $\mathbf{h}_{\theta,k}$ is necessarily *one-to-one.* The codomain of $\mathbf{h}_{\theta,k}$, i.e. the modeled manifold, is a *subset* of points with zeros at the corresponding latent dimensions, i.e. $\mathbf{x} \in \mathcal{M}_k \subset \mathcal{X}$ where

$$\mathcal{M}_k = \left\{ \mathbf{x}' \in \mathcal{X} : \mathbf{f}_\theta^{-1}(\mathbf{x}')_{k+1:D} = \mathbf{0} \right\}, \tag{4.19}$$

where $\mathcal{M}_k$ is a strict subset when $k < D$. By definition, a function is injective if it maps points in its domain to a *subset* of points in its codomain, and each output point corresponds to *at most* one input point. We have shown that both properties hold for $\mathbf{h}_{\theta,k}$. □

Like Brehmer and Cranmer (2020), we can use the definition of the manifold in Eq. (4.19) to check whether any given point $\mathbf{x}$ lies on said manifold, which is only the case if $\mathbf{f}_\theta^{-1}(\mathbf{x})_{k+1:D} = \mathbf{0}$. Similarly, dropping all but k latent dimensions of a point and passing the resulting vector through $\mathbf{f}_\theta$ we can project the point onto the k-dimensional manifold, which is equivalent to computing the lower-dimensional reconstruction for the nested dropout loss as done in Eq. (4.16).

To compute the manifold density of a point $\mathbf{x}$, we must use the general change-of-variables formula in Eq. (4.3), which involves "inverting" $\mathbf{h}_{\theta,k}$, i.e. finding $\tilde{\mathbf{z}}$ s.t. $\mathbf{x} = \mathbf{h}_{\theta,k}(\tilde{\mathbf{z}})$ and computing $\det\left(\mathbf{J}_{\mathbf{h}_{\theta,k}}(\tilde{\mathbf{z}})^\top \mathbf{J}_{\mathbf{h}_{\theta,k}}(\tilde{\mathbf{z}})\right)$. The former is trivial, assuming $\mathbf{x}$ lies on the manifold: $\tilde{\mathbf{z}} = \mathbf{f}_\theta^{-1}(\mathbf{x})_{1:k}$. For the latter, we note that the Jacobian of the zero-padding transformation is an identity function padded with zeros, which means that

$$\mathbf{J}_{\mathbf{h}_{\theta,k}}(\tilde{\mathbf{z}}) = \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z}) \mathbf{J}_{\tilde{\mathbf{z}} \to \mathbf{z}}(\tilde{\mathbf{z}}) = \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z}) \begin{bmatrix} \mathbb{I}_k \\ \mathbf{0} \end{bmatrix}, \tag{4.20}$$

and hence

$$\det\left(\mathbf{J}_{\mathbf{h}_{\theta,k}}(\tilde{\mathbf{z}})^\top \mathbf{J}_{\mathbf{h}_{\theta,k}}(\tilde{\mathbf{z}})\right) = \det\left(\begin{bmatrix} \mathbb{I}_k & \mathbf{0} \end{bmatrix} \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z})^\top \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z}) \begin{bmatrix} \mathbb{I}_k \\ \mathbf{0} \end{bmatrix} \right). \tag{4.21}$$

In our case, $\mathbf{f}_\theta$ is parametrized using a sequence of standard normalizing flow layers (coupling layers, invertible affine layers, etc.). Such layers were designed to enable cheap computation

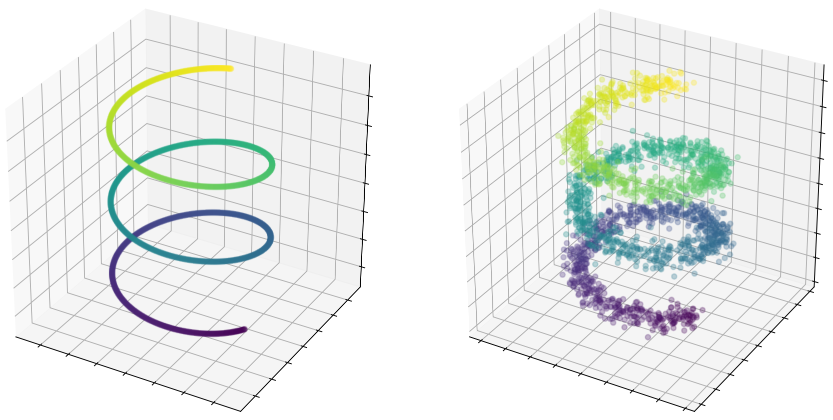

(a) No noise (b) With noise

Figure 4.1: Data on the manifold vs. data near the manifold. A 1-dimensional line embedded in $\mathbb{R}^3$ via a non-linear map, with/without Gaussian noise added in the ambient space. With no noise, data lie on the manifold, hence as discussed in Section 4.2.1 their density is not defined in $\mathbb{R}^3$, only on the 1-dimensional manifold. With noise, data lie *near* the manifold, and have a well-defined density in $\mathbb{R}^3$. NDF is best suited for dataset (b).

of $\det \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z})$, *not* $\det\left(\mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z})^\top \mathbf{J}_{\mathbf{f}_\theta}(\mathbf{z})\right)$. In practice, this means that we need to compute the Jacobians of the individual layers *explicitly*, multiply them, and compute the determinant of the resulting $D \times D$ matrix using one of the general $\mathcal{O}\left(D^3\right)$ algorithms. In the end, NDF in no way overcomes the common problem of manifold flows, which is that computing the *exact* density on the manifold is expensive, or even intractable.

4.3.3 Compared to alternative manifold flows

The fundamental difference between NDF and alternative manifold flows introduced in Section 4.2.3 is that in NDF we do not assume that the target density *actually* lies on the manifold. Instead we assume that the density is well-defined in the ambient space, but has *lower-dimensional structure*, as visualized in Fig. 4.1. Based on the popularity and success of PCA, we conjecture that this assumption holds in many datasets of interest, where the data lie close to a lower-dimensional manifold, not *exactly* on it. If the assumption holds, the standard log-likelihood training objective is well defined, and can be used for training the full-dimensional flow. In NDF, we simply augment the likelihood-based objective to allow extracting the low-dimensional densities from the trained ambient flow.

There are parallels between our training procedure and noise level sampling of SoftFlow (Kim et al., 2020). Like ours, their method does not assume that we know the true manifold dimension. In fact, with SoftFlow we can not control this dimension at all, instead relying on the model to find the single best fit. As mentioned in Section 4.2.3, both Brehmer and Cranmer (2020, Eq. 22) and Kumar et al. (2020, Eq. 6) arrive at an objective that 1) minimizes the reconstruction error, and 2) maximizes the likelihood on the manifold. Our loss in Eq. (4.17) is similar in spirit, but has important distinctions.

First, when training an NDF we are not actually maximizing the likelihood *on the manifold*. Instead, we maximize the likelihood of the ambient flow, and expect that, when projected onto a manifold that fits the data well, the modeled density on the manifold will resemble the true density. On one hand, this allows us to avoid the expensive evaluation of the density on the manifold through Eq. (4.3) during training. On the other hand, this means we can not guarantee that the true density on the manifold will indeed be captured. Brehmer and Cranmer, on the other hand, optimize the density on the manifold, albeit only training a part of their injective transformation for this part of the objective. Kumar et al. resort to training their injective flow using a series of lower bounds on the true manifold density.

Second, our method does not rely on the prior knowledge of the intrinsic data dimension. Brehmer and Cranmer discuss this as a potential drawback of their method in problems where the true manifold dimension is unknown. With their model, as well as with the model of Kumar et al., this would warrant a brute-force search for the right latent dimension K, re-training the flow for each considered value and choosing one based on performance on held-out data. Like PCA, NDF makes it possible to pick the latent dimension at test time.

Finally, while Brehmer and Cranmer use two separate flows, one for each part of the loss, we train a single flow for both. This potentially simplifies the implementation, but also results in a bigger model: Brehmer and Cranmer can use a low-dimensional flow to model the density on the manifold, and a bigger flow to learn the manifold itself, where latter is *not* optimized for manifold density. At the same time, in line with Brehmer and Cranmer, the ability of NDF to trivially "invert" the injective transformation allows us to avoid training a separate encoder as done by Kumar et al.

4.3.4 Greedy sorting baseline

The reader might wonder if the additional training objective is strictly necessary. Can we take a flow trained with the standard log likelihood objective, and align the latent space *afterwards*? This leads to a simple baseline, where we order the dimensions of a standard, pre-trained flow by sorting the dimensions *greedily* according to reconstruction error. Pseudocode describing the approach is given in Algorithm 4.1. At each iteration we go through all unselected dimensions, and select the one which, when included, results in the smallest MSE on validation data. It is known that if the reconstruction error is

Algorithm 4.1 Sorting dimensions greedily by their MSE contribution. Takes a validation set $\mathcal{D}$ and a data dimension D. Outputs *sorted*, a sorted list of dimension indices.

```
1:  $unsorted \leftarrow$ set with numbers 1 to $D$
2:  $sorted \leftarrow$ empty list
3:  while $unsorted$ is not empty do
4:      $min \leftarrow$ max float value
5:      $next \leftarrow$ empty
6:      for all $idx$ in $unsorted$ do
7:          $dims \leftarrow sorted$
8:          $dims$.append(idx)
9:          $mse \leftarrow$ reconstruction MSE on $\mathcal{D}$ for when retaining $dims$ dimensions
10:         if $mse < min$ then
11:             $min \leftarrow mse$
12:             $next \leftarrow idx$
13:     $unsorted$.remove(next)
14:     $sorted$.append(next)
```

submodular[4] as a function of the set of selected dimensions, a greedy algorithm will discover a near-optimal ordering (Nemhauser et al., 1978).

The method assumes that the latent space is already axis-aligned in the pre-trained flow, and it is only a matter of finding the correct ordering of dimensions. We hypothesize that this is *not* the case, because the standard flow training regime has no incentive to align latent dimensions in any way, assuming a spherically-symmetric distribution like a standard normal is used.

Moreover, at each iteration we need to compute the MSE on a potentially large validation set, and the number of iterations scales as $\mathcal{O}(D^2)$, i.e. quadratically in the number of dimensions. This makes the computational cost prohibitive for higher-dimensional problems. Nevertheless, we use the greedy approach as a baseline in our experiments. To make experiments manageable, we restrict the size of validation set that we evaluate the MSE on.

4.4 Experiments

We now run experiments to evaluate NDF empirically. Our goal is to understand whether the additional nested dropout objective is indeed aligning the latent space of the flow in a way that allows us to obtain meaningful samples from low-dimensional manifolds. We first turn to a simple synthetic 3-dimensional dataset, where PCA is the optimal dimensionality reduction method. We then move on to a high-dimensional natural image density estimation

[4]Informally, a set function is submodular if there are *diminishing returns*: as the input set size *increases* the gain in function value from adding a single element to the set *decreases*.

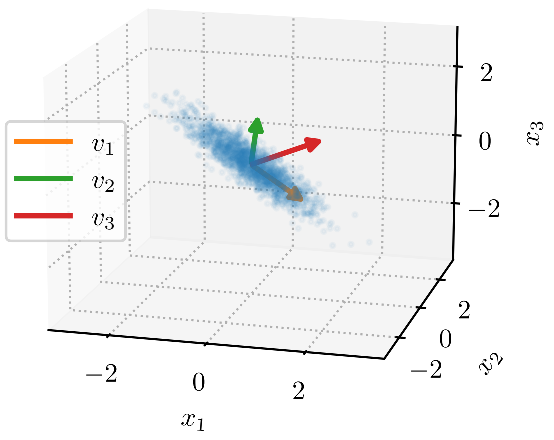


Figure 4.2: Synthetic data, sampled from a transformed normal distribution in 3D. The three eigenvectors of the covariance are plotted (rescaled to aid readability). The corresponding eigenvalues are: $\lambda_1 = 1$, $\lambda_2 = 0.1$, $\lambda_3 = 0.01$. The data distribution looks like a "fuzzy" elliptical disc embedded in the three-dimensional space, and can be projected onto a planar manifold with a small re-construction error.

task using Fashion-MNIST images.

4.4.1 Synthetic dataset

We first consider a simple synthetic dataset, where we sample points from a centered 3-dimensional normal distribution that is scaled along the axis, and then rotated. The eigenvalues of the covariance are 1, 0.1, and 0.01, so the target distribution looks like a fuzzy elliptical disc embedded in 3-dimensional ambient space, as visualized in Fig. 4.2. We sample 10^4 points for training, and another 10^4 points for evaluation.

We use a simple flow with a standard normal base distribution and a *single* invertible *linear* transformation. Such a flow can express the target distribution exactly by scaling/rotating the base distribution as required. However, due to the spherical symmetry of the standard normal distribution, the likelihood objective is invariant to a permutation (or indeed to any orthogonal transformation) of the latent dimensions. As a result, the standard training does not incentivize the model to align/order the latent dimensions in any way.

The invertible linear transformation is parameterized by either an LU decomposition with a random, fixed permutation matrix $\mathbf{P}$ (Kingma and Dhariwal, 2018), or a QR decomposition (Hoogeboom et al., 2019b) with the orthogonal matrix $\mathbf{Q}$ parameterized by 3 Householder transformations (Tomczak and Welling, 2016). We train the flow using the

Table 4.1: Nested dropout results on synthetic data. Average test log-likelihood (in nats) and reconstruction MSE when projecting down to 1 or 2 dimensions for the synthetic dataset, comparing baseline flows to the same flows trained with the additional nested dropout objective. Mean ± 2 standard deviations over 10 different initializations. Nested dropout moves the flow reconstruction results closer to PCA numbers, although with larger variance for the LU parametrization.

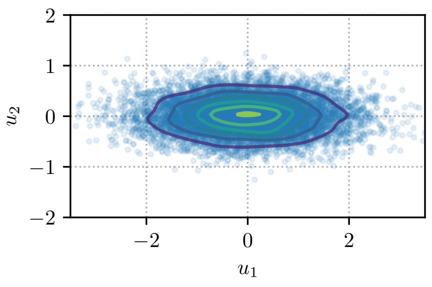
	LL	MSE(2)	MSE(1)
PCA	–	0.003	0.037
QR (BASELINE)	-0.804 ± 0.000	0.240 ± 0.053	0.321 ± 0.025
QR (NDF)	-0.804 ± 0.000	0.003 ± 0.000	0.037 ± 0.000
LU (BASELINE)	-0.804 ± 0.000	0.051 ± 0.042	0.257 ± 0.130
LU (NDF)	-0.805 ± 0.001	0.008 ± 0.007	0.037 ± 0.001

Adam optimizer (Kingma and Ba, 2015) for 30×10^3 iterations with a batch size of 500. We set the reconstruction coefficient to $\lambda = 20$ for nested dropout runs, with p_k set to a geometric distribution with $p = 0.33$, unless stated otherwise.

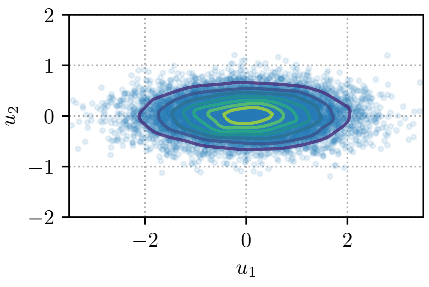
Likelihood and reconstruction results of the trained flows are summarized in Table 4.1. We use PCA as a baseline for reconstruction numbers, computing the corresponding principal components. With both the QR and the LU flows the additional nested dropout loss improves the low-dimensional reconstruction errors. Moreover, the QR flow trained with nested dropout matches the optimal PCA reconstruction results, with no impact on the test likelihood of the flow.

With the LU flow, the nested dropout objective negatively impacts the test likelihood, in addition to not pushing the reconstruction error all the way to PCA results. The results for this parametrization also have a higher variance, which is likely due to the fixed permutation matrix $\mathbf{P}$: it is initialized to a random permutation matrix at the beginning of each run, and is not learned. The matrix $\mathbf{P}$ defines the subset of transformations that the LU layer can represent, and determines the optimization surface for the matrices $\mathbf{L}$ and $\mathbf{U}$. At the same time, we see that the LU parametrization is biased towards solutions with better reconstruction results, even when training without the nested dropout objective.

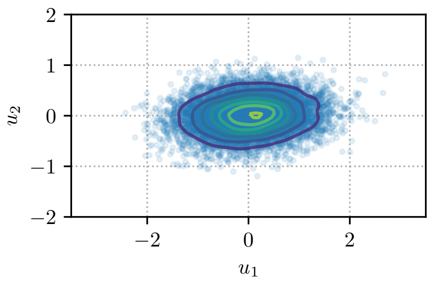
In Fig. 4.3 we analyze the samples from the NDF manifold. We sample from NDF with $k = 2$ using Eq. (4.18), and project the samples onto the first two principal vectors computed by PCA for the training data. We do the same for validation data. A mismatch between the principal components of the two sets of datapoints would suggest that the true manifold density has not been captured, which is what we see for the baseline flow. For NDF, the empirical density of sample projections resembles the validation data, indicating that the manifold density of NDF is not unreasonable.



(a) Validation data



(b) NDF samples, $k = 2$

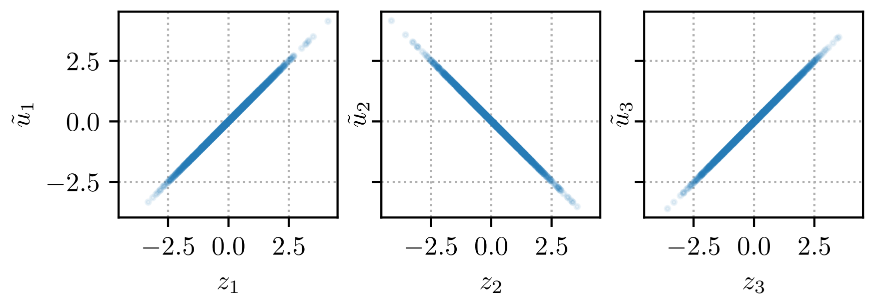


(c) Baseline samples, $k = 2$

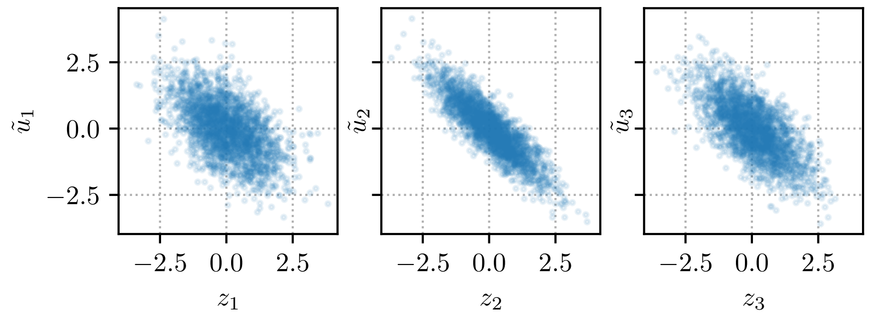
Figure 4.3: Projections of the low-dimensional NDF samples onto the principal vectors of the data. Flows fit to synthetic data in Fig. 4.2. We sample from the trained NDF for $k = 2$, and project the samples onto the first two principal vectors of PCA computed for the training data. We plot the resulting two-dimensional points, as well as their kernel density estimate. We do the same for validation data. For NDF the two densities are similar, which is not the case for the baseline flow. This suggests that the two-dimensional manifold density fit by NDF is not unreasonable given the data.

Finally, we aim to understand whether NDF is *equivalent* to PCA for this dataset. We do so empirically, measuring the correlation between the latent variables of NDF and the principal components of PCA. We plot the latent variables with corresponding principal components in Fig. 4.4. Unlike for the baseline flow, the correlation between these values is *perfect* for NDF, assuming we normalize the principal components (divide each by the square root of the corresponding eigenvalue). At the same time, we see that the correlation is *negative* (≈ -1) for one of the components. This is expected: the loss of PCA is invariant to reversing the direction of any of the component vectors. The objective of PCA is to find a vector $\mathbf{v}^*$ that minimizes the reconstruction error, or, equivalently, maximizes the variance of the principal components:

$$\mathbf{v}^* = \operatorname*{argmax}_{\mathbf{v}} (\mathbf{X}\mathbf{v})^\top (\mathbf{X}\mathbf{v}), \quad \text{s.t.} \quad \|\mathbf{v}\| = 1, \tag{4.22}$$



(a) NDF



(b) Baseline

Figure 4.4: Correlation between latent variables of NDF and principal components of PCA. Flows fit to synthetic data in Fig. 4.2. For the validation set, we plot the latent variables z_i and the principal components $\tilde{u}_i$. The latter are *normalized*: we divide each component by the square root of the corresponding eigenvalue. As we can see, the NDF values are *perfectly* correlated with the corresponding principal components, suggesting that NDF indeed fits PCA in this case. The correlation is negative for the second variable — the PCA loss is invariant to reversing the direction of any of the component vectors. This correlation is significantly weaker for the baseline flow trained without nested dropout.

where we assume that the data matrix $\mathbf{X} = [\mathbf{x}_1 \;\; \mathbf{x}_2 \;\; \ldots \;\; \mathbf{x}_N]^\top$ is centered (the mean of each column is zero). Defining $\tilde{\mathbf{v}} = -\mathbf{v}$, if $\|\mathbf{v}\| = 1$ then trivially $\|\tilde{\mathbf{v}}\| = 1$, and the PCA objective is unchanged:

$$(\mathbf{X}\tilde{\mathbf{v}})^\top(\mathbf{X}\tilde{\mathbf{v}}) = -\mathbf{v}\mathbf{X}^\top\mathbf{X}-\mathbf{v} \tag{4.23}$$

$$= (\mathbf{X}\mathbf{v})^\top(\mathbf{X}\mathbf{v}). \; \square \tag{4.24}$$

4.4.2 Images

To evaluate the method on real, high-dimensional data, we fit a normalizing flow to Fashion-MNIST images (Xiao et al., 2017). To simplify the flow architecture, we zero-pad the images by 2 pixels on each side, giving 32×32 images, which means the observation space

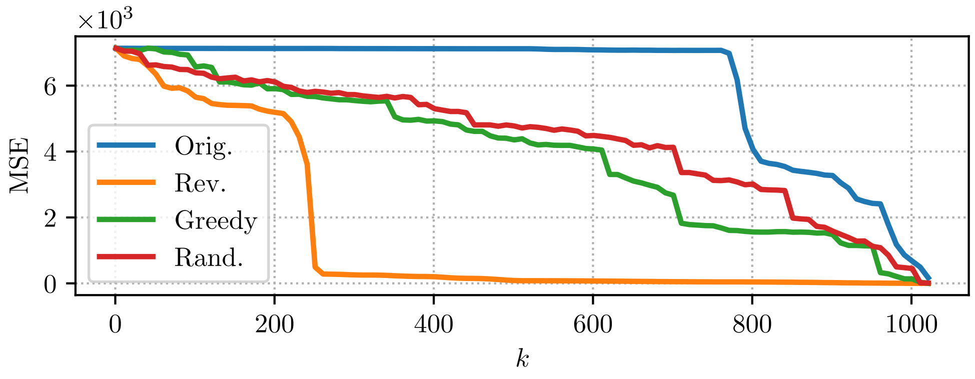


Figure 4.5: Impact of dimension ordering on the flow reconstruction MSE. Mean squared error of Fashion-MNIST reconstructions for RQ-NSF (C) flow against the number of retained dimensions k, varying the order in which the dimensions are dropped. *Greedy* for the order that results from sorting the dimensions greedily according to their MSE contribution, as described in Section 4.3.4. The reversed ordering is clearly preferable. The greedy ordering shows surprisingly poor results.

is 1024-dimensional. We use the provided test set, but split the training set into 50 000 training images and 10 000 validation images. We dequantize the images by adding uniform noise $\mathbf{U} \in [0, 1)^{32\times 32}$ (as discussed in Section 3.2.9).

We use a RQ-NSF (C) image flow (Section 3.3 and Durkan et al., 2019b) containing 3 multi-scale levels with 4 transformation steps per level, where each step consists of an activation normalization layer, a 1x1 convolution, and a rational-quadratic coupling transform. Residual convolutional networks with 3 blocks and 128 hidden channels parameterize the 4-bin rational-quadratic splines in the coupling transforms. We train all models for 100×10^3 iterations, with a batch size of 256. We anneal the learning rate of the Adam optimizer from 5×10^{-4} down to zero according to a cosine schedule (Loshchilov and Hutter, 2017, Eq. 5).

Ordering in a multi-scale architecture

The multi-scale architecture with variable splitting, introduced by Dinh et al. (2017) and used in RQ-NSF (C), already induces a partial ordering on variables, even without the additional nested dropout objective. In the multi-scale architecture some variables undergo more transformations than others. Each time the variables are “split-off”, an invertible *squeezing* transformation is undertaken, which reduces the spatial resolution of the image, offloading finer detail to additional channels. The end result is that the variables that undergo more transformations are eventually transformed at a “coarser scale”, i.e. by transformations conditioned on variables that are further away spatially. Dinh et al. (2017,

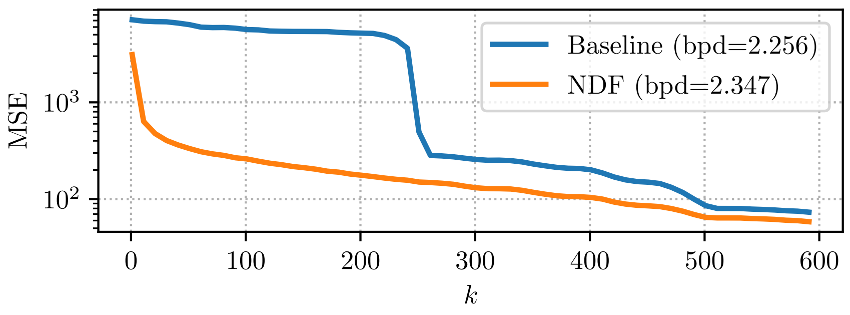


Figure 4.6: Impact of nested dropout on the flow reconstruction MSE. Mean squared error of Fashion-MNIST reconstructions for RQ-NSF (C) flow against the number of retained dimensions k. Comparing a flow trained with nested dropout to the baseline without the additional loss. *bpd* is the negative test log-likelihood in bits per dimension (lower is better). Training with nested dropout results in better reconstruction numbers, but at a cost the lower likelihood.

Appendix D) demonstrate that this by itself causes a subset of the variables to contain higher-level semantic information.

We demonstrate the effect of this on reconstruction errors in Fig. 4.5. There is a sudden drop in reconstruction error after including ≈250 dimensions from the reversed order. In our implementation of the multi-scale transform, it is this last quarter of the variables that are transformed at all 3 multi-scale levels. This makes the results consistent with the intuition above.

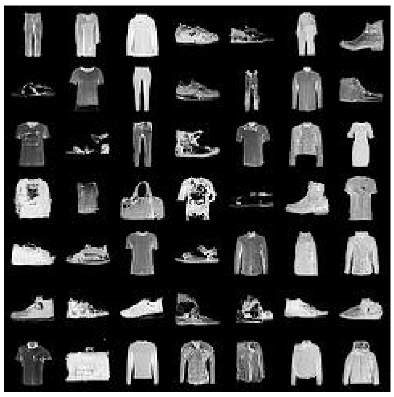
We attempt to improve the ordering by sorting the dimensions greedily according to Algorithm 4.1. This performs much worse than the reversed ordering, and, surprisingly, barely beats the random ordering. It could be due to only the small subset of the validation set being used, and/or because the problem at hand is not actually submodular (as discussed in Section 4.3.4).

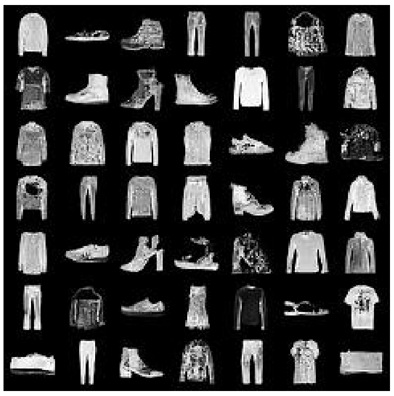
We use the reverse order as the baseline order in all further experiments. In addition, we use the reverse order when indexing dimensions for training with nested dropout. For example, for $k = 1$ we would only retain the *last* dimension in the latent vector, for $k = 2$ the last two dimensions, and so on.

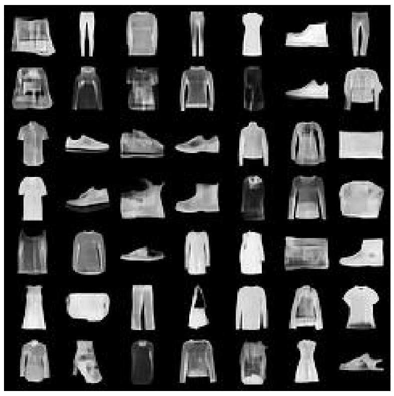
Training with nested dropout

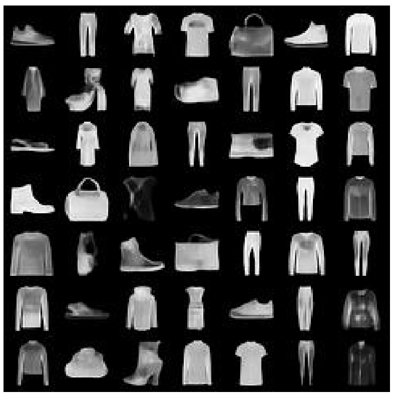
We now evaluate the additional nested dropout objective, aiming to improve upon the baseline RQ-NSF (C) model in terms of reconstruction error. We set the reconstruction coefficient $\lambda = 10^{-3}$, and we set $p = 10^{-3}$ for the geometric distribution p_k.

(a) Training data


(b) Baseline


(c) NDF ($k = 1024$)


(d) NDF ($k = 128$)


(e) NDF ($k = 64$)

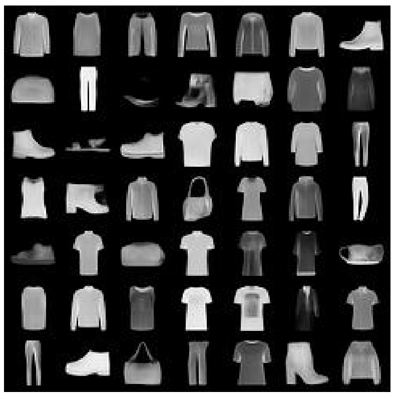

(f) NDF ($k = 16$)

Figure 4.7: Fashion-MNIST image samples from the NDF manifolds. Training data and samples from a baseline flow with no nested dropout are shown for comparison. We sample from the full-dimensional NDF ($D = 1024$ for the padded FashionMNIST), and from the manifolds with corresponding dimension k. We see that the smaller the manifold dimension k, the less "low-level" information (textures, detailing, etc.) is present in the samples, suggesting NDF has indeed captured the principal modes of variation.

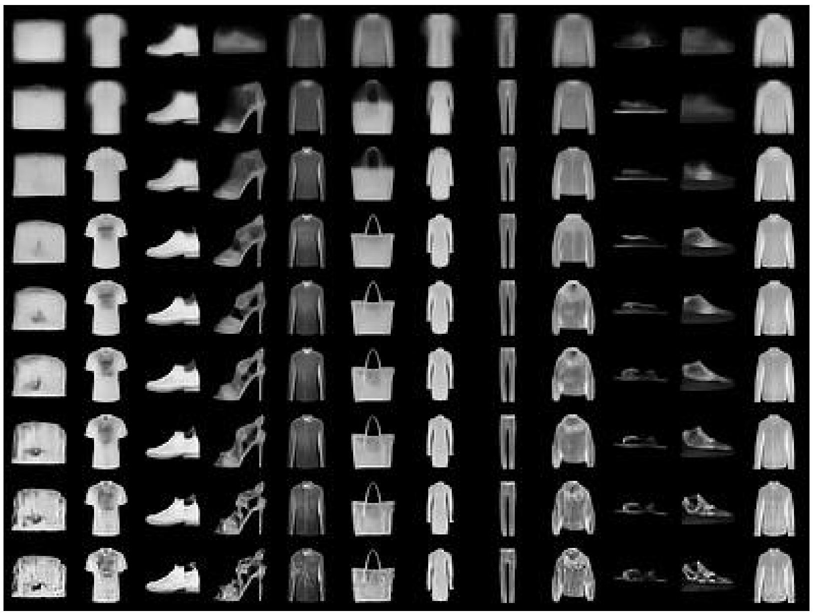


Figure 4.8: NDF samples when increasing the number of dimensions retained k. The number of dimensions retained k increases in $[2, 4, 8, 16, 32, 64, 128, 256, 512]$ top-to-bottom. The values in the retained dimensions stay *fixed* across rows. Finer details appear as more dimensions are sampled, while the object identity and higher-level features stay consistent. This is not the case for the baseline flow, indicating the lack of similar alignment of the latent space.

As shown in Fig. 4.6, the reconstruction errors are reduced significantly when training with nested dropout, in particular for $k < 250$. At the same time, the test likelihood of the flow suffers, being reduced by $\approx 4\%$ for NDF, hinting at a conflict between the two objectives. We study this conflict further in later experiments.

We show the samples from the densities on manifolds learned by NDF in Fig. 4.7. We vary the manifold dimension k, and compare the NDF samples to the training data and to samples from the baseline flow. With $k = 1024$ (retaining all dimensions), there is no clear difference between the NDF and baseline flow samples, even though the likelihoods of the two models differ. Both methods capture the data distribution reasonably well judging by their samples. This is not unexpected: Theis et al. (2016) note that the relationship between the perceptual quality of samples and likelihood can be complex.

However, as we reduce k, the low-dimensional samples from NDF contain less detail and high-frequency information, but retain the high-level features and sample diversity. This is the "regularization" effect that we seek when applying the manifold hypothesis, and suggests the manifold densities learned by NDF indeed match the data well.

Aiming to better understand the nature of the manifolds learned by NDF, in Fig. 4.8 we again plot the manifold samples varying the manifold dimension k, but this time *fixing* the retained latent values. In other words, when increasing k we sample additional latent values, keeping the previously sampled values unchanged. We see that as we sample more dimensions, the object identity and high-level properties stay fixed, but the object becomes less generic and gains detail. For example, the generic t-shirt for $k = 2$ becomes a t-shirt with *a* print for $k = 16$, which becomes a t-shirt with a certain material texture and a *particular* print for $k = 256$.

In Fig. 4.9 we plot lower-dimensional projections and reconstructions of validation images. For NDF, the story resembles the one in Fig. 4.8: as we project onto manifolds with fewer dimensions retained k, the object in the image sheds detail, but its identity and high-level properties stay consistent. In comparison, the object identity consistently flips with $k < 128$ for the baseline flow trained without nested dropout. Together with Fig. 4.8, these results suggest that NDF has indeed captured semantically meaningful manifolds, in addition to densities on them.

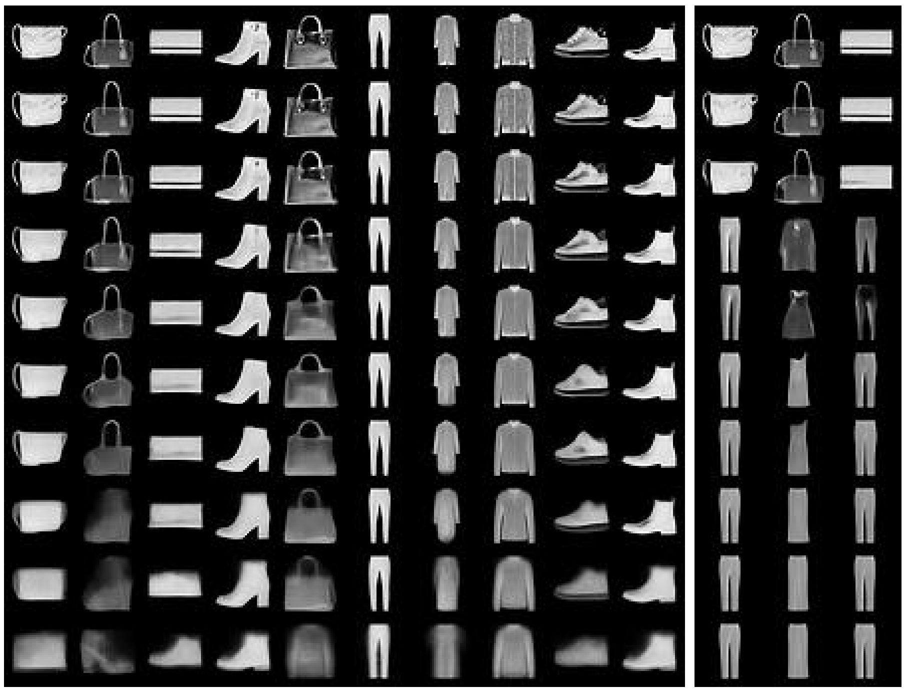
The likelihood-reconstruction trade-off

We now revisit the trade-off between the likelihood and the reconstruction error of NDF, noted in previous experiments. To understand the impact of the hyper-parameters on this trade-off, we perform a limited grid-search for λ and p, starting with the baseline values used in previous experiments, and perturbing each hyper-parameter independently. The resulting likelihoods and reconstruction errors are shown in Fig. 4.10.

The lower value of p (dropping more dimensions on average during training) marginally improves the MSE for $k < 200$, but generally this hyper-parameter has a limited impact on the results. This is consistent with the observation made by Xu et al. (2021) that the form and parameters of the distribution p_k are not critical to the success of the method.

The effect of perturbing λ is more pronounced. The larger value causes a noticeable improvement in reconstruction results, while the lower value has the opposite effect. Surprisingly, all 4 hyper-parameter settings improve upon the baseline in terms of test likelihood. This could be due to non-trivial interactions between λ and p, or simply due to a large variation across runs. All but one hyper-parameter setting, including the one with the best reconstruction results, still lose to the baseline flow in terms of test likelihood (as reported in Fig. 4.6).

Finally, we aim to understand the role of optimization in the likelihood-reconstruction trade-off. We start with a flow trained *without* the additional nested dropout penalty and *fine-tune* it using the combined nested dropout objective. The likelihood and reconstruction curves are shown in Fig. 4.11. We can see that very quickly after adding the nested dropout penalty the optimization settles into a region with much better reconstruction numbers,



(a) NDF (b) Baseline

Figure 4.9: Lower-dimensional projections of Fashion-MNIST images on the NDF manifolds. The input images are given in the top row, followed by lower-dimensional projections, decaying k in $[512, 256, 128, 64, 32, 16, 8, 4, 2]$ top-to-bottom. The identity of the projected object stays consistent for NDF, with textures and detailing disappearing for lower-dimensional projections. This suggests the high-level information is indeed pushed to the desired latent dimensions in NDF. This is not the case for the baseline, where the object identity always flips.

but worse likelihoods, further suggesting that the two objectives are at odds with each other for the used flow architecture.

4.5 Discussion

Nested dropout is a simple way to encourage a normalizing flow to represent data as closely as possible in the top, *principal* elements of its latent representation. Given the large redundancy in the way flows parametrize distributions, we can train flows with similar likelihoods that align the latent space. The goal of this alignment in this work is to allow extracting densities on low-dimensional manifolds from the trained ambient flow. Our experiments show that the manifold densities learned by NDF are meaningful, both for the low-dimensional synthetic dataset and the high-dimensional Fashion-MNIST image dataset.

For linear flows with an unrestricted parametrization, we know we can rotate the latent

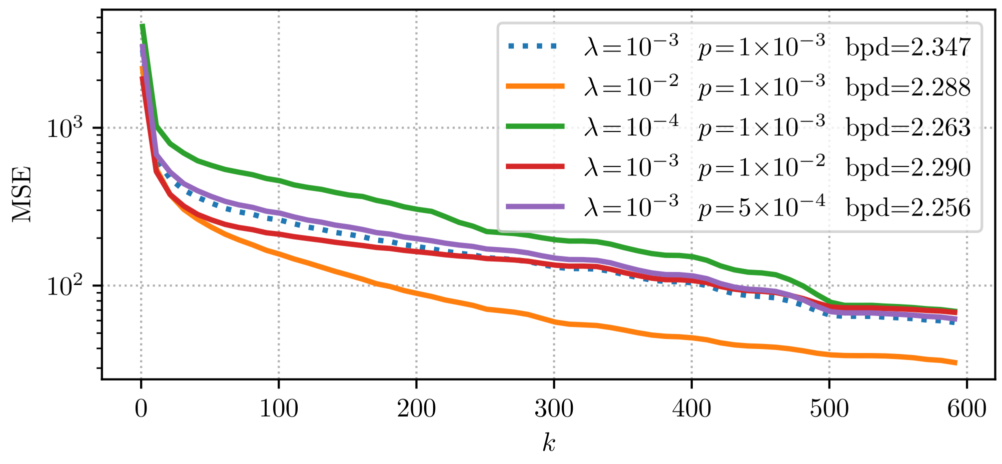


Figure 4.10: Effect of NDF hyper-parameters. Mean squared error of Fashion-MNIST reconstructions for RQ-NSF (C) flow trained with nested dropout against the number of retained dimensions k, varying hyper-parameters λ and p. The dotted line for the baseline nested dropout model. *bpd* is the negative test log-likelihood in bits per dimension (lower is better). The effect of changing p is limited, the effect of changing λ is as expected: higher values lead to better reconstruction results, but lower likelihoods. All runs improve on the baseline in terms of bpd: the interactions between hyperparameters are likely complex.

space without losing generality, and hence observe no impact on the likelihood. For more realistic flows, we saw some loss of model likelihood when encouraging the flow to order its representation. This suggests that existing flows have some inductive bias *against* the desired alignment. It is likely that for some parametrizations the desired ordering might come naturally and with no likelihood cost, while other parametrizations might find it difficult to represent the desired ordering while *also* matching the target density well. We have studied the impact of hyper-parameters and optimization on this trade-off, but more research is needed to understand the role of the flow parametrization, as well as the inherent biases for useful orderings in flow architectures.

Unlike other manifold flows, NDF does not explicitly fit the density on the manifold. Instead, it aims to match the ambient density *and* the low-dimensional structure using two separate terms in its loss. We conjecture that the density on the manifold will be approximated well when projecting the well-matched ambient density on the well-matched manifold, and our experimental results provide evidence that this is indeed the case. On one hand, this allows us to avoid the expensive density evaluation or approximation during training, and to use standard normalizing flow frameworks and architectures. On the other hand, it is difficult to characterize the relationship between the manifold density fit by our method and the true manifold density, and hence put any guarantees/bounds on the fit. Theoretical and/or

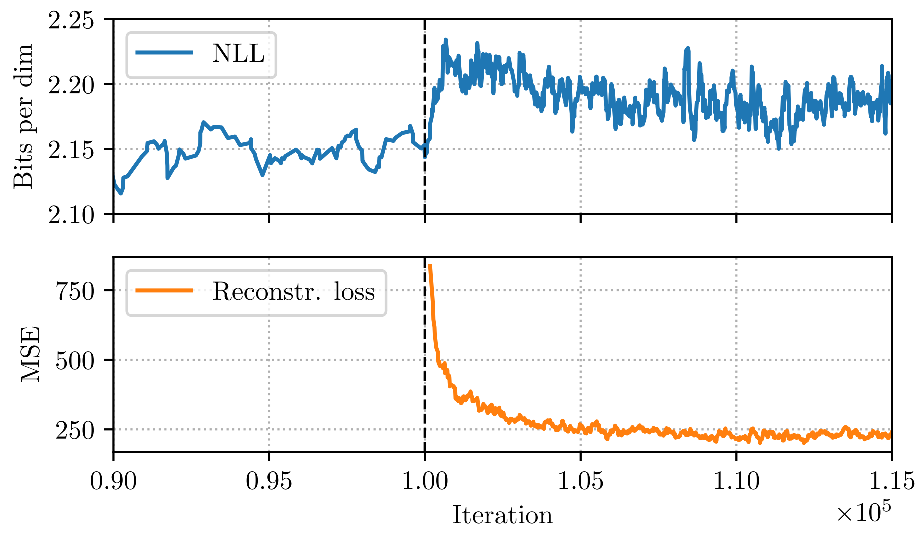


Figure 4.11: Effect of fine-tuning a trained flow after adding the nested dropout objective. Fine-tuning a baseline RQ-NSF(C) model with $\lambda = 10^{-3}$ and $p = 10^{-3}$. Dashed line indicates the iteration at which the fine-tuning begins. Training log likelihood in bits per dimension and reconstruction mean squared error are plotted. After adding the nested dropout objective the model quickly settles in a region with better reconstruction numbers, but lower likelihoods.

empirical study of this relationship would be an interesting extension of our work.

In NDF the flow must be evaluated in both directions on every iteration to compute the likelihood and the reconstruction error. In our experiments the training was typically slower by a factor of two when using architectures with one-pass sampling. NDF with flow architectures *without* a cheap inverse, e.g. auto-regressive flows, would be impractical.

NDF, like other contemporary manifold flows, does not provide a cheap-to-compute density on the manifold, which requires computing the determinant of a $D \times D$ matrix for a D-dimensional data space. When evaluating densities on the manifold is important, subsequently proposed *conformal embedding flow* (CEF, Ross and Cresswell, 2021) should be considered. CEF is based on injective transformations designed to have an easy-to-compute $\det \mathbf{J}^\top \mathbf{J}$, and hence provides tractable densities on the manifold.

When data *really* lie on a low-dimensional manifold, densities in data space are ill-defined, and so is the NDF training objective. If that is the case, and the dimension of the manifold is known, the methods of Kumar et al. (2020) and Brehmer and Cranmer (2020) are more appropriate. However, PCA and other dimensionality reduction methods are often applied when data are not *really* restricted to a subspace of fixed dimension, or when this dimension is unknown. We hope that nested dropout flows will prove useful in similar circumstances.

Chapter 5

Discussion

The probabilistic approach to machine learning allows us to use the language of probability to formalize and reason about learning problems. Probabilistic methods explicitly account for the uncertainty inherent to natural processes, and the uncertainty resulting from learning with finite data. While attractive conceptually, the probabilistic approach is often difficult to implement in practice, especially when it comes to complex, *high-dimensional* learning problems. Classic probabilistic methods make naive assumptions about the modeled processes and/or struggle to scale to large, high-dimensional datasets, which means that practitioners often simplify or ignore important aspects of the probabilistic approach.

5.1 Probabilistic prediction

As our first contribution, the experiments in Chapter 2 demonstrate the importance of careful Bayesian treatment in *probabilistic prediction* with high-dimensional data. The success of neural networks trained by maximum likelihood is often used as a justification against developing accurate, scalable approximate inference methods and complex, hierarchical priors that faithfully express our prior belief about the underlying function. This justification is being questioned as underestimated model uncertainty manifests in overconfident predictions for out-of-distribution and adversarial points. Our work demonstrates that principled probabilistic treatment is important for training trustworthy models that *know when they do not know.*

In particular, our experiments show how *informative* priors and *accurate* methods for estimating the posterior eliminate the calibration errors on adversarial points for a simple *linear* model. Increasing the flexibility of the model only increases the importance of using proper priors and accurate inference schemes. Assuming a fixed dataset, extending the set of functions that the model can represent amplifies the effect of prior-induced "regularization" and the negative impact of underestimated model uncertainty — there are more parameter settings consistent with the data that will not be captured by naive inference methods. In other words, principled probabilistic treatment is no less important for expressive models like neural networks, and is likely to bring a similar improvement to their predictive uncertainty.

To this end, there are several exciting directions in the Bayesian neural network literature. Instead of setting priors in the neural network weight space directly, Hafner et al. (2019) propose to define a prior on the *input-output relationship* instead, while Sun et al. (2019), Pearce et al. (2019) and Karaletsos and Bui (2020) propose to define priors in the *function space* of neural networks, searching for architectures and/or weight priors that correspond to a particular Gaussian Process kernel. Finally, Atanov et al. (2019) train a generative model of the neural network weights, using it as a prior for novel tasks.

When it comes to approximate inference, Izmailov et al. (2021) study the properties of neural network posteriors by running standard HMC on modern neural network architectures, Cobb and Jalaian (2021) develop *scalable* HMC methods for accurate and practical inference in BNNs, while Tomczak et al. (2021) propose a novel variational lower bound for BNNs, demonstrating improved performance of the cheap mean-field approximation. Importantly, Nado et al. (2021), Band et al. (2021) and Osband et al. (2021) develop benchmarks and implement baselines in order to stimulate the development and principled evaluation of novel BNN methods.

5.2 Density estimation

As our next contribution, in Chapter 3 we propose a class of flexible models for high-dimensional *density estimation*. Density estimation is a principled formalization of unsupervised learning in the context of probabilistic machine learning. Estimating densities of high-dimensional data is hard, however, and practitioners often consider more practical, non-probabilistic formulations of unsupervised learning. In this thesis, we propose NSF — a novel class of *normalizing flow* models that use flexible *rational-quadratic* monotonic splines. Such flexible parametrizations allow NSF to fully exploit the expressive power of neural networks to parametrize complex, high-dimensional densities, demonstrating state-of-the-art performance on several density estimation benchmarks.

We believe an important implication of our results is that the proposed coupling NSF approaches a *universal* density model: it is expressive and parameter-efficient, is differentiable and hence trainable by gradient-based methods, provides tractable densities, and is cheap to sample from. Representing and manipulating densities is fundamental to probabilistic modeling and reasoning, and, besides estimating the data density directly, density models are useful in likelihood-free inference (Papamakarios et al., 2019b), variational inference (Rezende and Mohamed, 2015), sampling algorithms (Müller et al., 2018), and for building generative classifiers (Schott et al., 2019). We believe that NSF is a step towards a plug-and-play density model that works well in many density estimation contexts.

While normalizing flows demonstrate excellent density estimation results, the perceptual quality of their samples is lacking when compared to alternative generative methods like GANs. Recently proposed *denoising diffusion models* (Ho et al., 2020) resemble normalizing flows, but instead *fix* the data-to-noise direction to a prescribed diffusion process, and learn a sequence of conditional densities that iteratively "de-noises" the samples from the base distribution. Diffusion models are trained by maximum likelihood, but their samples are competitive with the samples of state-of-the-art GAN models (Dhariwal and Nichol, 2021).

At the same time, diffusion models demonstrate underwhelming density estimation results when compared to alternative likelihood-based models. Similarly, training normalizing flows with an alternative objective (e.g. the Jensen–Shannon divergence, as used in GANs) improves their sample quality, but negatively impacts their likelihoods (Grover et al., 2018). Studying the tension between the likelihood-based training objectives and sample quality will be an important topic for future research.

As our final contribution, in Chapter 4 we propose a novel method for fitting normalizing flows to data with lower-dimensional structure. Assumptions about the target density are necessary to make high-dimensional density estimation tractable, and to build models with the desired inductive bias. The manifold hypothesis is an important assumption used in many machine learning methods, but it is difficult to use with likelihood-based density models: the standard training objectives become ill-defined when data are restricted to a lower-dimensional manifold.

Unlike the existing manifold flows, NDF is based on the assumption that many types of real data (including images) are not *restricted* to a manifold, even if they do have non-trivial lower-dimensional structure. In this case, nested dropout can align the latent space of a flow, thus capturing the lower-dimensional structure of the density, as indicated by the samples from our model.

NDF allows us to avoid the conceptual and practical complications of training injective flows, but is less principled: we never *actually* fit the density on the manifold. Studying the NDF manifold *densities* is an important direction for future work, aided by the fact that said densities, while expensive, *can* be computed. It is also important to understand if the performance of NDF transfers to more complex, higher-dimensional image datasets like ImageNet (Russakovsky et al., 2015), or alternative data modalities like speech or general time series.

5.3 Bayesian inference and density estimation

While Bayesian inference and density estimation have been treated separately in this thesis, there are several interesting lines of research in the intersection of the two topics.

As mentioned before, density models like normalizing flows can be used to define flexible variational families in the context of variational inference (Rezende and Mohamed, 2015). When it comes to neural network posteriors, the target densities are *extremely* high-dimensional, so a normalizing flow with a *single* invertible linear layer — equivalent to fitting a full-covariance Gaussian if the base distribution is Gaussian — becomes too expensive. Existing work by Louizos and Welling (2017) reparametrizes the neural network to allow fitting the normalizing flow in a lower-dimensional parameter space. Developing similar ways of incorporating structure and independence assumptions into normalizing flows is important if we want to use them for variational inference with complex models like neural networks.

Finally, in Chapters 3 and 4 we fit the normalizing flow parameters by maximum likelihood. As demonstrated in Chapter 2, this can lead to undesirable effects, which can be alleviated by setting proper priors and performing Bayesian inference to compute the posterior. Pope et al. (2020) note that modern normalizing flows *also* suffer from adversarial examples, where minimal perturbations to the valid input can significantly decrease its likelihood. Trippe and Turner (2018) explore *Bayesian normalizing flows*, and observe improved performance on a set of *conditional* density estimation tasks. Can we improve adversarial robustness and/or reduce overfitting observed in Chapter 4 through careful probabilistic treatment of normalizing flows? Unfortunately, setting priors on the parameters of neural normalizing flows and doing accurate inference for them is no less difficult than it is for other neural networks. We hope that future developments in Bayesian deep learning can enable a Bayesian treatment of normalizing flows as well.

Bibliography

Andrei Atanov, Arsenii Ashukha, Kirill Struminsky, Dmitry P. Vetrov, and Max Welling. The deep weight prior. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ByGuynAct7. *(Cited on page 123.)*

Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283. PMLR, 2018. URL http://proceedings.mlr.press/v80/athalye18a.html. *(Cited on page 35.)*

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through Bayesian deep Q-networks. In *2018 Information Theory and Applications Workshop, ITA 2018, San Diego, CA, USA, February 11-16, 2018*, pages 1–9. IEEE, 2018. doi: 10.1109/ITA.2018.8503252. URL https://doi.org/10.1109/ITA.2018.8503252. *(Cited on page 24.)*

Neil Band, Tim GJ Rudner, Qixuan Feng, Angelos Filos, Zachary Nado, Michael W Dusenberry, Ghassen Jerfel, Dustin Tran, and Yarin Gal. Benchmarking Bayesian deep learning on diabetic retinopathy detection tasks. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021. *(Cited on page 123.)*

Xuchan Bao, James Lucas, Sushant Sachdeva, and Roger B Grosse. Regularized linear autoencoders recover the principal components, eventually. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6971–6981. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4dd9cec1c21bc54eecb53786a2c5fa09-Paper.pdf. *(Cited on page 104.)*

Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartłomiej Świątkowski, Bernhard Schölkopf, and Richard E. Turner. Discriminative k-shot learning using probabilistic models. *arXiv preprint*, 2017. URL http://arxiv.org/abs/1706.00326v2. *(Cited on pages 20, 24, 32, and 51.)*

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR, 2019. URL http://proceedings.mlr.press/v97/behrmann19a.html. *(Cited on pages 64 and 101.)*

Artur Bekasov and Iain Murray. Bayesian adversarial spheres: Bayesian inference and adversarial examples in a noiseless setting. *Bayesian Deep Learning Workshop at the Conference on Neural Information Processing Systems (NeurIPS)*, 2018. URL http://arxiv.org/abs/1811.12335v1. *(Cited on page 18.)*

Artur Bekasov and Iain Murray. Ordering dimensions with nested dropout normalizing flows. *Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models (INNF+) at the International Conference on Machine Learning (ICML)*, 2020. URL http://arxiv.org/abs/2006.08777v1. *(Cited on page 93.)*

Adi Ben-Israel. The change-of-variables formula using matrix volume. *SIAM Journal on Matrix Analysis and Applications*, 21(1):300–312, 1999. doi: 10.1137/S0895479895296896. URL https://doi.org/10.1137/S0895479895296896. *(Cited on page 98.)*

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8): 1798–1828, 2013. *(Cited on pages 96 and 97.)*

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380. *(Cited on page 35.)*

Michael Betancourt and Mark Girolami. Hamiltonian Monte Carlo for hierarchical models. *Current trends in Bayesian methodology with applications*, 79(30):2–4, 2015. *(Cited on page 48.)*

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019. URL http://jmlr.org/papers/v20/18-403.html. *(Cited on page 89.)*

James F. Blinn. How to solve a cubic equation, part 5: Back to numerics. *IEEE Computer Graphics and Applications*, 27(3):78–89, 2007. *(Cited on page 77.)*

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 1613–1622, 2015. *(Cited on pages 15, 19, and 23.)*

Ondrej Bohdal, Yongxin Yang, and Timothy M. Hospedales. Meta-calibration: Meta-learning of model calibration using differentiable expected calibration error. *arXiv preprint*, 2021. URL `https://arxiv.org/abs/2106.09613`. *(Cited on page 23.)*

John Bradshaw, Alexander G. de G. Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. *arXiv preprint*, 2017. URL `http://arxiv.org/abs/1707.02476v1`. *(Cited on pages 21, 24, 25, 32, and 51.)*

Johann Brehmer and Kyle Cranmer. Flows for simultaneous manifold learning and density estimation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/051928341be67dcba03f0e04104d9047-Abstract.html`. *(Cited on pages 17, 95, 100, 101, 102, 106, 107, 108, and 121.)*

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1xsqj09Fm`. *(Cited on page 88.)*

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`. *(Cited on page 14.)*

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning*

Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016. URL http://arxiv.org/abs/1509.00519. *(Cited on page 86.)*

Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005. *(Cited on page 97.)*

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html. *(Cited on page 100.)*

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 32, pages 1683–1691, 2014. *(Cited on page 24.)*

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020. URL http://proceedings.mlr.press/v119/chen20j.html. *(Cited on page 97.)*

Adam D. Cobb and Brian Jalaian. Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 675–685. PMLR, 2021. URL https://proceedings.mlr.press/v161/cobb21a.html. *(Cited on page 123.)*

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: An extension of MNIST to handwritten letters, 2017. URL http://arxiv.org/abs/1702.05373. *(Cited on page 86.)*

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020. doi: 10.1073/pnas.1912789117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1912789117. *(Cited on page 14.)*

Prafulla Dhariwal and Alex Nichol. Diffusion models beat GANs on image synthesis. *arXiv preprint*, 2021. URL https://arxiv.org/abs/2105.05233v4. *(Cited on page 124.)*

Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave

Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. Tensorflow distributions. *arXiv preprint*, 2017. URL http://arxiv.org/abs/1711.10604. *(Cited on page 89.)*

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1410.8516. *(Cited on page 70.)*

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx. *(Cited on pages 16, 59, 66, 68, 70, 91, and 114.)*

Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint*, 2019. URL http://arxiv.org/abs/1903.08689v3. *(Cited on page 94.)*

Dheeru Dua and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. *(Cited on page 82.)*

Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216 – 222, 1987. ISSN 0370-2693. doi: https://doi.org/10.1016/0370-2693(87)91197-X. URL http://www.sciencedirect.com/science/article/pii/037026938791197X. *(Cited on pages 31 and 41.)*

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. *Workshop on Invertible Neural Nets and Normalizing Flows (INNF) at the International Conference on Machine Learning (ICML)*, 2019a. URL http://arxiv.org/abs/1906.02145v1. *(Cited on pages 57 and 89.)*

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7511–7522. Curran Associates, Inc., 2019b. URL http://papers.nips.cc/paper/8969-neural-spline-flows.pdf. *(Cited on pages 57, 89, and 114.)*

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018. doi: 10.1109/CVPR.2018.00175. *(Cited on page 25.)*

Luca Falorsi and Patrick Forré. Neural ordinary differential equations on manifolds. *arXiv preprint*, 2020. URL `https://arxiv.org/abs/2006.06663`. *(Cited on page 100.)*

Yarin Gal. *Uncertainty in Deep Learning.* PhD thesis, University of Cambridge, 2016. *(Cited on pages 18 and 19.)*

Yarin Gal and Lewis Smith. Sufficient conditions for idealised models to have no adversarial examples: a theoretical and empirical study with Bayesian neural networks. *arXiv preprint*, 2018. URL `http://arxiv.org/abs/1806.00667v3`. *(Cited on pages 15, 19, 20, and 25.)*

Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006. *(Cited on page 48.)*

Mevlana C. Gemici, Danilo Rezende, and Shakir Mohamed. Normalizing flows on Riemannian manifolds. *arXiv preprint*, 2016. URL `http://arxiv.org/abs/1611.02304v2`. *(Cited on pages 94, 98, 99, and 102.)*

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: masked autoencoder for distribution estimation. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 881–889. JMLR.org, 2015. URL `http://proceedings.mlr.press/v37/germain15.html`. *(Cited on page 64.)*

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015. ISSN 1476-4687. doi: 10.1038/nature14541. URL `https://doi.org/10.1038/nature14541`. *(Cited on page 14.)*

Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Scholkopf. From variational to deterministic autoencoders. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1g7tpEYDS`. *(Cited on page 101.)*

Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian J. Goodfellow. Adversarial spheres. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=SkthlLkPf`. *(Cited on pages 20, 21, 33, 36, 37, 39, 51, and 52.)*

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf. *(Cited on page 88.)*

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. *(Cited on page 14.)*

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014a. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets. *(Cited on pages 94 and 101.)*

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014b. *(Cited on pages 15, 19, 25, 26, and 40.)*

Alex Graves. Practical variational inference for neural networks. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2348–2356, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993. *(Cited on pages 15 and 23.)*

John A. Gregory and Roger Delbourgo. Piecewise rational quadratic interpolation to monotonic data. *IMA Journal of Numerical Analysis*, 2(2):123–130, 1982. *(Cited on pages 59 and 78.)*

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-GAN: Combining maximum likelihood and adversarial learning in generative models. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3069–3076. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17409. *(Cited on page 124.)*

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 2017. URL http://proceedings.mlr.press/v70/guo17a.html. *(Cited on pages 15 and 22.)*

Danijar Hafner, Dustin Tran, Timothy P. Lillicrap, Alex Irpan, and James Davidson. Noise contrastive priors for functional uncertainty. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 905–914. AUAI Press, 2019. URL http://proceedings.mlr.press/v115/hafner20a.html. *(Cited on page 123.)*

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001a. *(Cited on page 31.)*

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001b. *(Cited on page 13.)*

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016a. *(Cited on page 81.)*

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *European Conference on Computer Vision*, 2016b. *(Cited on page 81.)*

Nils-Bastian Heidenreich, Anja Schindler, and Stefan Sperlich. Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advances in Statistical Analysis*, 97(4):403–433, 2013. ISSN 1863-818X. doi: 10.1007/s10182-013-0216-y. URL https://doi.org/10.1007/s10182-013-0216-y. *(Cited on page 62.)*

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *Proceedings of the International Conference on Machine Learning*, 2019. *(Cited on page 39.)*

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1861–1869, 2015. *(Cited on page 24.)*

Geoffrey E. Hinton, Peter Dayan, and Michael Revow. Modeling the manifolds of images of handwritten digits. *IEEE Trans. Neural Networks*, 8(1):65–74, 1997. doi: 10.1109/72.554192. URL https://doi.org/10.1109/72.554192. *(Cited on pages 94 and 96.)*

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach,* *California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2722–2730. PMLR, 2019. URL http://proceedings.mlr.press/v97/ho19a.html. *(Cited on pages 16, 59, 70, 75, and 88.)*

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html. *(Cited on page 124.)*

Matthew D. Hoffman and Andrew Gelman. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(47): 1593–1623, 2014. URL http://jmlr.org/papers/v15/hoffman14a.html. *(Cited on page 31.)*

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013. *(Cited on pages 29 and 61.)*

Emiel Hoogeboom, Jorn W. T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12134–12144, 2019a. URL https://proceedings.neurips.cc/paper/2019/hash/9e9a30b74c49d07d8150c8c83b1ccf07-Abstract.html. *(Cited on page 73.)*

Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2771–2780. PMLR, 2019b. URL http://proceedings.mlr.press/v97/hoogeboom19a.html. *(Cited on pages 69 and 110.)*

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900208. *(Cited on page 14.)*

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron C. Courville. Neural autoregressive flows. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stock-*

holm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 2018. URL http://proceedings.mlr.press/v80/huang18d.html. *(Cited on pages 16, 59, 70, 92, and 101.)*

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html. *(Cited on pages 39, 51, and 91.)*

Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson. What are Bayesian neural network posteriors really like? In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4629–4640. PMLR, 2021. URL https://proceedings.mlr.press/v139/izmailov21a.html. *(Cited on page 123.)*

Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3009–3018. PMLR, 2019. URL https://proceedings.mlr.press/v97/jaini19a.html. *(Cited on pages 59 and 71.)*

Pasi Jylänki, Aapo Nummenmaa, and Aki Vehtari. Expectation propagation for neural networks with sparsity-promoting priors. *The Journal of Machine Learning Research*, 15 (1):1849–1901, 2014. *(Cited on page 24.)*

Theofanis Karaletsos and Thang D. Bui. Hierarchical Gaussian process priors for Bayesian neural network weights. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c70341de2c112a6b3496aec1f631dddd-Abstract.html. *(Cited on page 123.)*

Archit Karandikar, Nicholas Cain, Dustin Tran, Balaji Lakshminarayanan, Jonathon Shlens, Michael C. Mozer, and Becca Roelofs. Soft calibration objectives for neural networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29768–29779, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f8905bd3df64ace64a68e154ba72f24c-Abstract.html. *(Cited on page 23.)*

Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017. *(Cited on page 19.)*

Hyeongju Kim, Hyeonseung Lee, Woo Hyun Kang, Joun Yeop Lee, and Nam Soo Kim. Softflow: Probabilistic framework for normalizing flow on manifolds. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16388–16397. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf. *(Cited on pages 102 and 107.)*

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980. *(Cited on pages 33, 51, 81, 91, 92, and 111.)*

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114. *(Cited on pages 61 and 94.)*

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2575–2583, 2015. *(Cited on page 29.)*

Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4743–4751, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819. *(Cited on pages 16, 59, 64, 68, 86, and 92.)*

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc., 2018. *(Cited on pages 16, 59, 60, 69, 83, 84, 86, 87, 88, 89, 90, 91, and 110.)*

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of* *the National Academy of Sciences*, 114(13):3521–3526, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1611835114. URL http://www.pnas.org/content/114/13/3521. *(Cited on pages 23 and 30.)*

Frederic Koehler, Viraj Mehta, and Andrej Risteski. Representational aspects of depth and conditioning in normalizing flows. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5628–5636. PMLR, 2021. URL https://proceedings.mlr.press/v139/koehler21a.html. *(Cited on pages 16 and 59.)*

Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. doi: https://doi.org/10.1002/aic.690370209. URL https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209. *(Cited on page 96.)*

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. *(Cited on pages 37 and 84.)*

Abhishek Kumar, Ben Poole, and Kevin Murphy. Regularized autoencoders via relaxed injective probability flow. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR, 2020. URL http://proceedings.mlr.press/v108/kumar20a.html. *(Cited on pages 17, 94, 100, 101, 102, 108, and 121.)*

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJGU3Rodl. *(Cited on page 25.)*

Malte Kuss and Carl Edward Rasmussen. Assessing approximate inference for binary Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704, 2005. URL http://www.jmlr.org/papers/volume6/kuss05a/kuss05a.pdf. *(Cited on page 46.)*

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6402–6413, 2017. *(Cited on page 55.)*

Yann LeCun and Corinna Cortes. MNIST handwritten digit database, 2010. URL http://yann.lecun.com/exdb/mnist/. *(Cited on pages 13 and 86.)*

Yingzhen Li and Yarin Gal. Dropout inference in Bayesian neural networks with alphadivergences. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017. *(Cited on page 25.)*

Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2323–2331, 2015. *(Cited on page 24.)*

Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.324. URL https://doi.org/10.1109/ICCV.2017.324. *(Cited on page 23.)*

William A. Link and Mitchell J. Eaton. On thinning of chains in MCMC. *Methods in Ecology and Evolution*, 3(1):112–115, 2012. doi: https://doi.org/10.1111/j.2041-210X.2011.00131.x. URL https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/j.2041-210X.2011.00131.x. *(Cited on page 42.)*

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx. *(Cited on pages 35, 81, 91, 92, and 114.)*

Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser-Nam Lim, and Christopher De Sa. Neural manifold ordinary differential equations. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/cbf8710b43df3f2c1553e649403426df-Abstract.html. *(Cited on page 100.)*

Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix Gaussian posteriors. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1708–1716. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/louizos16.html. *(Cited on page 23.)*

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017. *(Cited on pages 19, 23, and 125.)*

David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. URL http://www.cambridge.org/0521642981. Available from http://www.inference.phy.cam.ac.uk/mackay/itila/. *(Cited on pages 29, 30, and 61.)*

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992a. *(Cited on pages 15, 18, and 23.)*

David JC MacKay. *Bayesian methods for adaptive models.* PhD thesis, California Institute of Technology, 1992b. *(Cited on page 23.)*

Matthew MacKay, Paul Vicol, Jimmy Ba, and Roger B. Grosse. Reversible recurrent neural networks. In *NeurIPS*, pages 9043–9054, 2018. URL http://papers.nips.cc/paper/8117-reversible-recurrent-neural-networks. *(Cited on page 88.)*

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb. *(Cited on page 33.)*

K.V. Mardia and P.E. Jupp. *Directional Statistics.* Wiley Series in Probability and Statistics. Wiley, 2009. ISBN 9780470317815. URL https://books.google.co.uk/books?id=PTNiCm4Q-M0C. *(Cited on page 100.)*

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423 vol.2, 2001. doi: 10.1109/ICCV.2001.937655. *(Cited on page 82.)*

Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1aa3d9c6ce672447e1e5d0f1b5207e85-Abstract.html. *(Cited on page 100.)*

Thomas P. Minka and Rosalind Picard. *A Family of Algorithms for Approximate Bayesian Inference.* PhD thesis, Massachusetts Institute of Technology, USA, 2001. AAI0803033. *(Cited on page 55.)*

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. *(Cited on page 25.)*

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2017. doi: 10.1109/CVPR.2017.17. *(Cited on page 25.)*

Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip H. S. Torr, and Puneet K. Dokania. Calibrating deep neural networks using focal loss. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/aeb7b30ef1d024a76f21a1d40e30c302-Abstract.html`. *(Cited on page 23.)*

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029. *(Cited on pages 14, 20, 27, 32, 40, 45, 61, and 62.)*

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *arXiv preprint*, 2018. URL `http://arxiv.org/abs/1808.03856v3`. *(Cited on pages 16, 59, 60, 61, 71, 72, 73, 75, 80, 82, and 124.)*

Zachary Nado, Neil Band, Mark Collier, Josip Djolonga, Michael Dusenberry, Sebastian Farquhar, Angelos Filos, Marton Havasi, Rodolphe Jenatton, Ghassen Jerfel, Jeremiah Liu, Zelda Mariet, Jeremy Nixon, Shreyas Padhy, Jie Ren, Tim Rudner, Yeming Wen, Florian Wenzel, Kevin Murphy, D. Sculley, Balaji Lakshminarayanan, Jasper Snoek, Yarin Gal, and Dustin Tran. Uncertainty Baselines: Benchmarks for uncertainty & robustness in deep learning. *arXiv preprint*, 2021. URL `https://arxiv.org/abs/2106.04015v3`. *(Cited on page 123.)*

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using Bayesian binning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015:2901–2907, 2015. ISSN 2159-5399. URL `https://pubmed.ncbi.nlm.nih.gov/25927013`. 25927013[pmid]. *(Cited on page 22.)*

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=H1xwNhCcYm`. *(Cited on page 61.)*

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using typicality. *arXiv preprint*, 2019b. URL `https://arxiv.org/abs/1906.02994v2`. *(Cited on page 61.)*

Charlie Nash and Conor Durkan. Autoregressive energy machines. *International Conference on Machine Learning*, 2019. *(Cited on pages 81 and 92.)*

Radford M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Dept. of Computer Science, University of Toronto, 1994. *(Cited on pages 15, 18, 23, and 24.)*

Radford M. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, 2011. *(Cited on page 41.)*

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions — I. *Math. Program.*, 14(1):265–294, 1978. ISSN 0025-5610. doi: 10.1007/BF01588971. URL https://doi.org/10.1007/BF01588971. *(Cited on page 109.)*

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/cedebb6e872f539bef8c3f919874e9d7-Paper.pdf. *(Cited on page 101.)*

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Botao Hao, Morteza Ibrahimi, Dieterich Lawson, Xiuyuan Lu, Brendan O'Donoghue, and Benjamin Van Roy. Evaluating predictive distributions: Does Bayesian deep learning work?, 2021. *(Cited on page 123.)*

George Papamakarios. Preprocessed datasets for MAF experiments, 2018. URL https://zenodo.org/record/1161203. *(Cited on page 82.)*

George Papamakarios. *Neural Density Estimation and Likelihood-free Inference*. PhD thesis, Centre for Doctoral Training in Data Science, University of Edinburgh, 2019. *(Cited on pages 14, 17, 58, 61, and 95.)*

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2338–2347, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/6c1da886822c67822bcf3679d04369fa-Abstract.html. *(Cited on pages 16, 59, 64, 68, 82, 83, 86, and 92.)*

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint*, 2019a. URL http://arxiv.org/abs/1912.02762v1. *(Cited on pages 58, 63, and 70.)*

George Papamakarios, David C. Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 837–848. PMLR, 2019b. URL http://proceedings.mlr.press/v89/papamakarios19a.html. *(Cited on page 124.)*

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. *(Cited on page 89.)*

Tim Pearce, Russell Tsuchida, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. Expressive priors in Bayesian neural networks: Kernel combinations and periodic functions. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 134–144. AUAI Press, 2019. URL http://proceedings.mlr.press/v115/pearce20a.html. *(Cited on page 123.)*

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. *(Cited on page 37.)*

Christoph Peters. How to solve a cubic equation, revisited. http://momentsingraphics.de/?p=105, 2016. Accessed: 2019-04-17. *(Cited on page 77.)*

Phillip Pope, Yogesh Balaji, and Soheil Feizi. Adversarial robustness of flow-based generative models. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 3795–3805. PMLR, 2020. URL http://proceedings.mlr.press/v108/pope20a.html. *(Cited on page 125.)*

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia

Li, Razvan Pascanu, Peter W. Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5690–5701, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/9e82757e9a1c12cb710ad680db11f6f1-Abstract.html. *(Cited on page 14.)*

Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 814–822. JMLR.org, 2014. URL http://proceedings.mlr.press/v33/ranganath14.html. *(Cited on page 29.)*

Ambrish Rawat, Martin Wistuba, and Maria-Irina Nicolae. Adversarial phenomenon in the eyes of Bayesian deep learning. *arXiv preprint*, 2017. URL http://arxiv.org/abs/1711.08244v1. *(Cited on pages 19, 20, and 25.)*

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML)*, volume 37, pages 1530–1538, 2015. *(Cited on pages 45, 124, and 125.)*

Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint*, 2018. URL http://arxiv.org/abs/1810.00597. *(Cited on page 92.)*

Danilo Jimenez Rezende, George Papamakarios, Sebastien Racaniere, Michael Albergo, Gurtej Kanwar, Phiala Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8083–8092. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/rezende20a.html. *(Cited on pages 94 and 99.)*

Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint*, 2013. URL http://arxiv.org/abs/1302.5125. *(Cited on pages 16, 58, and 62.)*

Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. Learning ordered representations with nested dropout. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1746–1754. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/rippel14.html. *(Cited on pages 17, 93, 95, 103, 104, and 105.)*

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable Laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Skdvd2xAZ. *(Cited on pages 23 and 30.)*

Brendan Leigh Ross and Jesse C Cresswell. Tractable density estimation on learned manifolds with conformal embedding flows. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=DqU-rIHy4Eh. *(Cited on page 121.)*

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. *(Cited on pages 13, 53, 84, and 124.)*

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Found. Trends Mach. Learn.*, 11(1):1–96, 2018. ISSN 1935-8237. doi: 10.1561/2200000070. URL https://doi.org/10.1561/2200000070. *(Cited on page 24.)*

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 872–879, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390266. URL https://doi.org/10.1145/1390156.1390266. *(Cited on page 86.)*

Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on MNIST. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=S1EHOsC9tX. *(Cited on page 124.)*

David F Shanno. On Broyden-Fletcher-Goldfarb-Shanno method. *Journal of Optimization Theory and Applications*, 46(1):87–94, 1985. *(Cited on page 28.)*

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. *(Cited on page 74.)*

Edward Snelson, Zoubin Ghahramani, and Carl Rasmussen. Warped Gaussian processes. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004. URL https://proceedings.neurips.cc/paper/2003/file/6b5754d737784b51ec5075c0dc437bf0-Paper.pdf. *(Cited on page 88.)*

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2171–2180, 2015. *(Cited on pages 20, 24, 32, and 51.)*

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/076a0c97d09cf1a0ec3e19c7f2529f2b-Paper.pdf. *(Cited on page 24.)*

M. Spivak. *Calculus On Manifolds: A Modern Approach To Classical Theorems Of Advanced Calculus*. Mathematics monograph series. Avalon Publishing, 1971. ISBN 9780813346120. URL https://books.google.co.uk/books?id=POIJJJcCyUkC. *(Cited on page 62.)*

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014. URL http://dl.acm.org/citation.cfm?id=2627435.2670313. *(Cited on pages 82 and 103.)*

Matthias Steffen. A simple method for monotonic interpolation in one dimension. *Astronomy and Astrophysics*, 239:443, 1990. *(Cited on pages 59, 75, and 76.)*

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger B. Grosse. Functional variational Bayesian neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rkxacs0qY7. *(Cited on page 123.)*

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6199. *(Cited on pages 15, 19, and 25.)*

Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. In *International Joint Conference on Neural Networks*, 2015. *(Cited on page 26.)*

E. G. Tabak and Cristina V. Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. doi: https://doi.org/10.1002/cpa.21423. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.21423. *(Cited on pages 16, 58, and 62.)*

Thomas Tanay and Lewis Griffin. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint*, 2016. URL http://arxiv.org/abs/1608.07690v1. *(Cited on pages 26 and 41.)*

Takeshi Teshima, Isao Ishikawa, Koichi Tojo, Kenta Oono, Masahiro Ikeda, and Masashi Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/2290a7385ed77cc5592dc2153229f082-Abstract.html. *(Cited on page 67.)*

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.01844. *(Cited on pages 73, 74, 88, and 117.)*

Jakub M. Tomczak and Max Welling. Improving variational auto-encoders using Householder flow. *arXiv preprint*, 2016. URL http://arxiv.org/abs/1611.09630v4. *(Cited on pages 69 and 110.)*

Marcin B. Tomczak, Siddharth Swaroop, Andrew Y. K. Foong, and Richard E Turner. Collapsed variational bounds for Bayesian neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ykN3tbJ0qmX. *(Cited on page 123.)*

Dustin Tran, Keyon Vafa, Kumar Krishna Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14692–14701, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/e046ede63264b10130007afca077877f-Abstract.html. *(Cited on page 73.)*

Brian L Trippe and Richard E Turner. Conditional density estimation with Bayesian normalising flows. *arXiv preprint*, 2018. URL https://arxiv.org/abs/1802.04908. *(Cited on page 125.)*

Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Pro-*

cessing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pages 2175–2183, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/53adaf494dc89ef7196d73636eb2451b-Abstract.html. *(Cited on page 62.)*

Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 393–402. AUAI Press, 2018. URL http://auai.org/uai2018/proceedings/papers/156.pdf. *(Cited on page 101.)*

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *International Conference on Machine Learning*, 2016. *(Cited on pages 84 and 88.)*

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint*, 2018. URL http://arxiv.org/abs/1807.03748v2. *(Cited on page 97.)*

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. *(Cited on page 88.)*

Bastiaan S. Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant CNNs for digital pathology. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II*, volume 11071 of *Lecture Notes in Computer Science*, pages 210–218. Springer, 2018. doi: 10.1007/978-3-030-00934-2_24. URL https://doi.org/10.1007/978-3-030-00934-2_24. *(Cited on page 13.)*

Joe Watson, Jihao Andreas Lin, Pascal Klink, Joni Pajarinen, and Jan Peters. Latent derivative Bayesian last layer networks. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1198–1206. PMLR, 2021. URL https://proceedings.mlr.press/v130/watson21a.html. *(Cited on page 55.)*

Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks.

In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1543–1553, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2a084e55c87b1ebcdaad1f62fdbbac8e-Abstract.html. *(Cited on pages 89 and 101.)*

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10402–10413, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/488e4104520c6aab692863cc1dba45af-Abstract.html. *(Cited on page 26.)*

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195. *(Cited on page 24.)*

Tom White, Mattie Tesfaldet, Samaneh Azadi, Daphne Ippolito, Lia Coleman, and David Ha. Machine learning for creativity and design: A workshop at the Conference on Neural Information Processing Systems (NeurIPS). https://neuripscreativityworkshop.github.io/2021/, 2021. *(Cited on page 14.)*

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint*, 2017. URL http://arxiv.org/abs/1708.07747v2. *(Cited on pages 83 and 113.)*

Yilun Xu, Yang Song, Sahaj Garg, Linyuan Gong, Rui Shu, Aditya Grover, and Stefano Ermon. Anytime sampling for autoregressive models via ordered autoencoding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=TSRTzJnuEBS. *(Cited on pages 104 and 118.)*

Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint*, 2017. URL http://arxiv.org/abs/1712.07107. *(Cited on page 25.)*

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016. *(Cited on page 39.)*

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017. *(Cited on page 23.)*