Neural Networks for Inference, Inference for Neural Networks

Stefan Webb

Wadham College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2018

Abstract

Bayesian statistics is a powerful framework for modeling the world and reasoning over uncertainty. It provides a principled method for representing our prior knowledge, and updating that knowledge in the light of new information. Traditional Bayesian statistics, however, has been limited to simple models. Two of the main limiting factors for this are the expressiveness and flexibility of the probability distributions used, and the computational restrictions in performing inference and model learning. In this thesis, we consider how neural networks (NNs) can be used to assist with both of these problems. In particular, we will look at how NNs can assist in the inference process and how we can perform inference over flexible NN models.

NNs are helpful for Bayesian inference in generative models. They are useful for producing the flexible variational families required for successful variational inference (VI), as well as learning distributed representations of model variables for guiding the distributional relationships of the model. Inference, on the other hand, is useful for quantifying uncertainty in NN discriminative models. For instance, with "Bayesian NNs" one can use inference to learn a distribution over the NNs parameters and hence quantify our uncertainty over the model's predictions. However, this increased flexibility in modeling and inference comes with challenges in inference and representation.

We present three pieces of original work in this thesis towards solving these challenges. We produce an algorithm for constructing the factorization of variational approximations in an optimal way to improve the fidelity and scalability of VI. We develop a framework for distributed Bayesian learning that is particularly useful for large Bayesian NNs and is less prone to the stale gradient of non-Bayesian approaches. We finish by considering an example of how Bayesian inference can be applied to NNs in a non-standard context by reinterpreting the problem of estimating NN robustness as an inference problem.

Acknowledgements

I would like to thank first and foremost my girlfriend, Anjuli, and my parents back in Australia. Without your love and support I could not have gotten through this arduous time. I would also like to extend a special thanks to Tom Rainforth, whose mentorship was instrumental in bringing our work to publication—you have taught me a lot. I am grateful to my supervisors, Pawan and Yee Whye, for their guidance and direction, and my colleagues and collaborators in Engineering Science and Statistics for providing a stimulating intellectual environment. Lastly, I am thankful to Steve Roberts and Niki Trigoni, and the financial support provided by the AIMS CDT program and the EPSRC, for providing the opportunity to study at Oxford and become a published researcher, all the while experiencing many wonderful things—music, theatre, debates, speeches, and meeting the love of my life.

Contents

1 Introduction

Bayesian statistics [Gelman et al., 2013; Ghahramani, 2015] is a powerful framework for modeling the world and reasoning over uncertainty. Its central thesis is that the world can be understood through appropriately chosen probabilistic models, with parameters and variables inferred or learnt through data. Moreover, it provides a principled method for representing our prior knowledge, and updating that knowledge in the light of new information. Traditional Bayesian statistics, however, has been limited to simple models, frequently employing conjugacy assumptions in the distribution of the model to make inference tractable. Two of the main limiting factors for this are the lack of flexibility in the probability distributions used, and the computational restrictions in performing inference and model learning. In this thesis, we consider how neural networks (NNs) can be used to assist with both of these problems. In particular, we will look at how NNs can assist in the inference process and how we can perform inference over flexible NN models.

NNs are helpful for Bayesian inference in generative models. Variational inference (VI) is a family of inference algorithms widely used in Bayesian modeling whose distinguishing feature is reframing inference as an optimization problem. The success of VI methods depend on having a flexible family of distributions to approximate the posterior. NN density estimators can be effectively utilized to construct the required flexible distribution families. For instance, take the example of normalizing flows [Rezende and Mohamed, 2015], a class of methods for NN distribution parametrization, that transforms a simple Gaussian noise source into a more complex distribution by the application of learnable bijections with tractable Jacobians. Such parametrizations have been shown to improve inference in variational autoencoders [Kingma et al., 2016].

Conversely, inference is useful for flexible model learning in NNs. Typically the conditional distributions comprising a Bayesian model are simple distributions with parametric forms. Amortized VI (to be explained in Ch 2), however, enables model learning of models with arbitrary distributions, using inference to estimate the update to the model parameters. In this way, we can include model terms that, for example, use NNs to regress the values of a variable's parents

to the parameters of its conditional distribution. For instance, in the variational autoencoder (VAE) [Kingma and Welling, 2014], a NN is used in the model to learn how to decode the latent variable to the parameters of the distribution over images.

More broadly, inference is useful for analyzing NN models. Take the example of "Bayesian NNs," which provide a bridge between discriminative and generative models, and are formed from standard NN architectures by placing a prior distribution over the model parameters. Learning is reformulated as inference over the posterior of the parameters and, subsequently, one can thus represent and reason over our uncertainty about the NN model's outputs using this posterior. Another type of uncertainty is encountered during distributed learning of NNs, for which the worker nodes have incomplete and potentially out-of-date information on the progress of their fellow workers. As we will show in Ch 4, inference can be used to quantify this uncertainty and improve the robustness of distributed learning in NN models to stale gradients.

1.1 Overview

The increased flexibility in modeling and inference comes at the price of challenges in inference and representation. In this section, we give an outline of the remainder of the thesis, and connect how it relates to tackling these challenges.

In Ch 2, we provide the context for our work by giving a high-level overview of Bayesian modeling, inference, and representation. We first delineate discriminative and generative probabilistic models, and the importance of latent variable generative models for Bayesian inference. Probabilistic models differ in whether they model joint or conditional distributions, and in whether they contain latent variables. Deep learning is based, primarily, on discriminative models that are formed from conditional distributions where all variables are observed, and is well suited to scenarios that have masses of training data and little prior knowledge. Bayesian modeling, on the other hand, is based on models that are typically generative, or rather, based on joint distributions, and contain latent (unobserved) variables, whose value we must infer. These models are better suited for scenarios where we would like to use our prior knowledge to learn efficiently when big data is lacking. Generative models are required for more advanced tasks beyond classification and regression, such as anomaly detection, learning concepts, causal discovery, and disentangling factors of variation in our data. They allow us to learn from incomplete data, in an unsupervised or semi-supervised fashion, in contrast to fully-observed discriminative models.

We next explain three different types of inference methods—Markov chain Monte Carlo (MCMC) inference, variational inference (VI), and expectation propagation (EP)—and how each

is suited for different problems. MCMC inference is based on constructing a Markov chain that converges to the target distribution, typically, the posterior [Andrieu et al., 2003]. By simulating the Markov chain, one can draw approximate samples from the posterior for Bayesian inference, with time complexity that is of polynomial order in the dimension of the distribution, an advantage over other inference methods. Despite having relatively well-understood theoretical properties, the performance of MCMC is often tricky to characterize in practice, such as determining when the chains have approximately converged to the target distribution. Variational inference (VI) is another class of inference methods, which reframes inference as an optimization problem, learning an approximation to the posterior by minimizing the KL-divergence between it and the posterior [Jordan et al., 1999]. Relative to MCMC, the theoretical properties of VI are not well understood. Despite this, it converges more quickly in practice than MCMC methods, and scales to large data [Hoffman et al., 2013]. EP is a type of variational inference (in the broader sense of the term) that reverses the direction of the KL-divergence and has the advantage of naturally being suited for distributed learning.

We note that these three families of inference methods are not specific to Bayesian inference. MCMC allows us to draw samples from an unnormalized distribution, not necessarily the posterior. VI and EP are, likewise, general methods for matching a family of distributions to a fixed one that is not necessarily the posterior. Consider the goal of estimating an arbitrary expectation of a function under a target distribution. These inference methods can be used for approximating this expectation as follows. Using MCMC, one can draw approximate samples from the target and form a Monte Carlo (MC) estimate of the expectation. Using VI, the expectation can be estimated using a so-called importance sampling (IS) estimate, evaluating the function under samples from the variational approximation, weighting the terms according to how well the density of the variational approximation matches the target. These inference methods can thus be used for approximating probabilistic expectations, which is a more general aim than that of calculating an expectation over the posterior distribution. Indeed, this is how MCMC is applied in our work of Ch 4.

We also introduce modern and amortized VI, extensions of classical VI that operate on a much broader class of models. Amortized inference is of special interest, and learns an approximation, $q_\phi(\mathbf{z} \mid \mathbf{x})$, to the posterior known as an "inference network," that, unlike classical VI, is explicitly a function of the observed variables $\mathbf{x}$, amortizing the cost of performing inference over inference problems similar to those encountered during learning. We explain how they can be used to learn to perform inference without making any conjugacy assumptions or performing model-specific

derivations, and how they enable learning of deterministic parameters in the model. We explain two central challenges to both: reducing the variance of the gradient estimates, and producing flexible variational families, and how these can be tackled with NN techniques. For instance, in the work of [Mnih and Gregor, 2014], a dense feedforward NN is used to learn how to regress the observed datum to appropriately scale a control variate.

Concluding Ch 2, we delineate two facets of representation, factorization and parametrization, focusing on the latter. We describe how to construct NN distribution parametrizations for inclusion in either the model or the variational approximation, especially useful under amortized VI schemes. These NN distribution parametrizations can be used for powerful representation learning of the model variables. More broadly, they permit the whole suite of deep learning architectures to be utilized, including special techniques like attention [Eslami et al., 2016] and memory [Bornschein et al., 2017]. Our idealized NN distribution parametrization would satisfy a number of properties. It would have sampling and scoring (calculation of its density or mass function) with constant time complexity with respect to the dimension of the distribution. It would also be a universal density estimator; in other words, there would exist a sequence of NN density estimators within the family under consideration that converges in distribution to any target distribution (that satisfies some additional technical constraints) with the specified domain. It is difficult to satisfy all these properties simultaneously, however, and we describe various techniques and the tradeoffs they make in this section. For instance, inverse autoregressive flow (IAF) [Kingma et al., 2016] is a distribution parameterization with $O(1)$ time complexity for sampling, and $O(D)$ time complexity for scoring arbitrary samples, where D is the dimension of the distribution. Masked autoregressive flow (MAF) [Papamakarios et al., 2017], on the other hand, a related method, has $O(D)$ time complexity for sampling, and $O(1)$ time complexity for scoring arbitrary samples. Because of this, IAF is better suited for constructing inference networks, whereas MAF is more appropriate for density estimation. It is unlikely that either is a universal density estimator, motivating the development of more recent techniques.

Our development of amortized VI is continued in Ch 3, in which we present a novel algorithm for designing the structure of inference networks in a principled fashion that is guaranteed to be optimal in a technical sense [Webb et al., 2018a]. The fidelity with which the inference network is able to represent the true posterior effects the bias and variance of inference amortization. Moreover, an inadequate inference network has negative consequences for model learning as well, restricting the complexity of the resulting model that is learnt. Unfortunately, the structure of an inference network is typically formed in a heuristic fashion by inverting the edges of the

generative model, and is not guaranteed to be a structure which the true posterior factorizes over. If the true posterior does not factorize over the structure we have chosen for our inference network, the latter cannot represent the true posterior, even in the limit of universal density estimators over the individual factors. This motivates our algorithm, which takes as input the graphical model structure of a generative model, and outputs a graphical model structure for the posterior. The output is optimal in the sense that it does not mislead us about the conditional independencies expressed by the input—we say that it is faithful to the posterior, or equivalently that it is an I-map for the posterior [Koller and Friedman, 2009]. The output is also locally optimal in the sense that, while it is not guaranteed to have the least edges out of all I-maps for the posterior, the removal of a single edge makes it unfaithful to the posterior—it is a minimal I-map. We demonstrate the utility of model learning and inference amortization on several models with minimally faithful inference networks, comparing to heuristic and fully-connected variants. Looking to the future, we believe our method will prove a crucial component of automated universal inference in probabilistic programming languages.

The idea that inference is useful for analyzing the properties of discriminative NNs is developed in Ch 4 and 5. One instance of this is found in applying Bayesian inference to discriminative Bayesian NN models for reasoning about our uncertainty over the model's parameters (and thus, predictions) during and after learning. Consider the distributed learning of neural networks, such as by asynchronous SGD (A-SGD) [Dean et al., 2012] and elastic averaging SGD (EASGD) [Zhang et al., 2014]. In these methods, a worker node sends gradient updates on the model parameters to the master node based on its knowledge of the progress of the other workers. A form of uncertainty thus presents itself during learning due to the inevitably out-of-date knowledge each worker has about the other workers—the so-called "stale gradient problem." In Ch 4, we present a novel distributed Bayesian learning framework to ameliorate this deficiency [Hasenclever et al., 2016]. Rather than passing deterministic gradient updates, it passes variational approximations between workers—i.e., the messages are distributions rather than point estimates—and in this way is able to reconcile the uncertainty during learning. After learning, it is able to capture the uncertainty over the model predictions using the same variational approximation to the posterior over the model parameters. It is based on a modification of the standard EP [Gelman et al., 2014] algorithm. While our framework is a general one for Bayesian learning, it has a particular efficacy for learning Bayesian NNs, which we typically desire to learn from big data. EP has the advantage over other forms of inference for this problem by naturally being formulated for distributed learning.

Another instance of inference being useful for analyzing NNs is developed in Ch 5. It has been known for several years that NN classifiers can be tricked into misclassifying an input by the addition of a small amount of carefully constructed noise [Szegedy et al., 2013]. More generally, we would like our discriminative NN models to satisfy interpretable properties, e.g., the output does not deviate too greatly from a reference function, or that it satisfies the laws of physics within a given tolerance in the case of a control policy. The vulnerability of NNs to adversarial inputs is a serious issue for NN classifiers deployed in applications like medical image analysis and self-driving cars, where failure can result in financial loss or death. Towards this problem, we develop a novel measure of neural network robustness, framing the problem as the estimation of an expectation [Webb et al., 2018b]. We note that the event of failure is commonly a rare one, and make a connection to the rare event estimation literature of statistical inference. We show how an algorithm from this literature, adaptive multi-level splitting (AMLS) [Guyader et al., 2011], an MCMC inference algorithm, is able to estimate our robustness metric with low bias and variance on a variety of datasets and models. As a consequence of framing the problem as an inference task, our method scales more favourably than traditional purely optimization-based approaches, scaling linearly in the cost of the forward operation of the NN classifier. Also, by basing our method on MCMC based inference rather than VI, we are able to reliably calculate our metric for problems with high-dimensional inputs, which is common in image classification models. This work illustrates how inference methods apply to more general problems than Bayesian inference.

In Ch 6, we conclude with some thoughts on the application of the ideas in this thesis to future work. We sketch out how our novel robustness metric for NNs can be extended to scale to larger models and input-dimensionality, and how one can measure the "total NN robustness" for adversarial properties (in contrast to the per-datum robustness). We also suggest a method for robust training motivated by our work based on generating counter-examples by sampling methods, another connection of inference methods to neural networks. Finally, we elaborate on our suggestion that the NaMI algorithm of Ch 3 can be used to automate the design of inference networks in deep probabilistic programming language (PPLs). Recently developed deep PPLs like Pyro [Bingham et al., 2018] and Edward [Tran et al., 2016] combine deep learning frameworks like PyTorch and TensorFlow with simple abstractions for probabilistic modeling and inference, typically with a focus on amortized VI. Unfortunately, to perform VI in these conceptions of probabilistic programming the user is required to be particularly knowledgable in the details of VI, designing the inference network by hand based on heuristics and past experience. It would be desirable if the construction of the inference network could be automated. This

is one big hurdle in producing automated universal inference, which can effectively operate on any latent variable probabilistic model without the intervention of a user in the details of how inference is applied. We believe our NaMI algorithm can be used, together with recent developments in NN distribution representation, amortized VI, and the "poutine" abstraction mechanisms of Pyro, to automate the design of inference networks, improving upon existing schemes and advancing the goals of probabilistic programming.

1.2 Publications

The work presented in this thesis has been published, or has been accepted for publication at the following venues:

- Chapter 3: **Webb, Stefan**, Golinski, Adam, Zinkov, Robert, Narayanaswamy, Siddharth, Rainforth, Tom, Teh, Yee Whye, and Wood, Frank. Faithful Inversion of Generative Models for Effective Amortized Inference. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montreal, Canada*.

- Chapter 4: Hasenclever, Leonard, **Webb, Stefan**, Lienart, Thibaut, Vollmer, Sebastian, Lakshminarayanan, Balaji, Blundell, Charles, Tom, and Teh, Yee Whye. Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server. *Journal of Machine Learning Research 18 (2017) 1-37*.

- Chapter 5: **Webb, Stefan**, Rainforth, Tom, Teh, Yee Whye, and Pawan Kumar, M. A Statistical Approach to Assessing Neural Network Robustness. To appear in *Proceedings of the Seventh International Conference on Learning Representations (ICLR2019), New Orleans*.

This thesis is presented as an *integrated* thesis, in which my publications are included in their camera-ready form, that is, as they appear in the proceedings from their publication venues. Following each publication chapter is a signed statement of authorship detailing my contributions.

2
Literature review

In this chapter, we give a high-level overview of probabilistic modeling, Bayesian inference, and NN distribution representation. We first describe discriminative and generative models, and why latent-variable generative models are of special interest in probabilistic machine learning. We then explain the three different types of inference methods used in our work—Markov chain Monte Carlo, amortized variational inference (VI), and expectation propagation (EP)—how each uses or is applied to NNs, and why they are well-suited for the applications given here. We finally discuss NN distribution representations, especially as they relate to amortized VI, drawing a distinction between factorization and parametrization, and giving a summary of methods used for constructing such representations.

2.1 Probabilistic modeling

Let us begin with a useful taxonomy of probabilistic models to provide a context for how NNs are useful for inference and vice versa. Define by $\mathbf{y}$ the variables of interest to be predicted, and by $\mathbf{x}$ data that are (hopefully) informative about the outputs. For instance, $\mathbf{y}$ may be a vector of attributes relating to a real estate property such as postcode, number of bedrooms, number of bathrooms, etc., and $\mathbf{x}$ may be the property market value. Or, $\mathbf{x}$ may be the pixel values of an image containing a single object, and $\mathbf{y}$ a label describing the identity of the object in the image. Define by $\mathbf{z}$ a variable with similar meaning to $\mathbf{y}$, with the convention that $\mathbf{z}$ is typically a latent (unobserved) variable, whereas $\mathbf{y}$ typically denotes one that is observed. Probabilistic models can be roughly divided into two branches depending on whether the goal is to match a joint or a conditional distribution.

2.1.1 Discriminative models

In the discriminative modeling paradigm, the statistician assumes a conditional distribution, $p_\phi(\mathbf{y} \mid \mathbf{x})$, known as the *discriminative model*, relating the likelihood of observing the output given the value of the data and the parameters, ϕ. In this paradigm, both $\mathbf{y}$ and $\mathbf{x}$ are observed, and the model typically fit by the method of maximum likelihood: given a dataset $\mathcal{D} = \{\mathbf{y}_n, \mathbf{x}_n\}_{n=1}^N$, find

$$\phi^* = \text{argmax}_{\phi \in \Phi} \mathbb{E}_{(\mathbf{y},\mathbf{x}) \sim p_\mathcal{D}} \left[\ln \left(p_\phi \left(\mathbf{y} \mid \mathbf{x} \right) \right) \right]. \tag{2.1}$$

where $p_{\mathcal{D}}$ denotes the empirical data distribution. This is equivalent to minimizing the KL-divergence between the empirical distribution and model,

$$\phi^* = \text{argmin}_{\phi \in \Phi} \text{KL}\{p_{\mathcal{D}}(\mathbf{y}, \mathbf{x}) \parallel p_\phi(\mathbf{y} \mid \mathbf{x}) p_{\mathcal{D}}(\mathbf{x})\}.$$

This type of learning is known as *supervised learning*, since both the inputs and outputs are observed. Discriminative models are typically understood through a frequentist statistical lens, viewing the parameters as fixed and the data as random.

Traditional deep learning models [Goodfellow et al., 2016] assume that the parameters ϕ of the discriminative distribution are the output of a neural network (NN) inputting the features, $\phi = f_\theta(\mathbf{x})$, with its own parameters θ, learnt by stochastic gradient ascent on (2.1).

2.1.2 Generative models

In the generative modeling paradigm, on the other hand, the statistician assumes a joint distribution, $p_\phi(\mathbf{x})$ or $p_\phi(\mathbf{x}, \mathbf{z})$, known as the *generative model*, that directly models the observed data, $\mathbf{x}$, and, optionally, features $\mathbf{z}$.

Generative models of the form $p_\phi(\mathbf{x})$ are often referred to as autoregressive generative models, since the distribution is commonly represented by the chain rule as,

$$p_\phi(\mathbf{x}) = \prod_{n=1}^{N} p_\phi(x_n \mid \mathbf{x}_{\prec n}),$$

where $\mathbf{x}_{\prec n} \triangleq \{x_1, \ldots, x_{n-1}\}$, although we note that any factorization suffices. For this reason, we will refer to them as *fully-observed generative models*. A representative example is PixelRNN [van den Oord et al., 2016c], which represents a joint distribution over the pixels of an image parametrizing the ϕ of each factor using the output of a cleverly constructed RNN.

For fully-observed generative models, the objective is typically to learn $p_\phi(\mathbf{x})$ from the data by the method of maximum likelihood, which we again can interpret as minimizing a KL-divergence. This is a form of supervised learning.

Generative models of the form $p_\phi(\mathbf{x}, \mathbf{z})$ are referred to as *latent variable generative models*, and it is assumed that the latent variables, $\mathbf{z}$, in some way capture the dynamics of the process giving rise to $\mathbf{x}$. Latent variable generative models include both traditional statistical models like latent dirichlet allocation [Blei et al., 2003] and discrete Bayesian networks [Koller and Friedman, 2009], as well as modern "deep" generative models like the variational autoencoder [Kingma and Welling, 2014] and attend-infer-repeat [Eslami et al., 2016].

Latent variable generative models are most conveniently understood through a Bayesian statistical lens, where the data $\mathbf{x}$ is fixed, and the statistical parameters, $\mathbf{z}$, are random, and

must be inferred through the process of inference. The objective is typically to approximate the posterior, $p(\mathbf{z} \mid \mathbf{x})$, either through approximate samples, or an approximation, $q_\psi(\mathbf{z})$ or $q_\psi(\mathbf{z} \mid \mathbf{x})$. The model can also be learnt by maximizing (indirectly) the marginal log-likelihood, $p_\phi(\mathbf{x})$, when amortized VI is used. Typically $\mathbf{x}$ is observed and $\mathbf{z}$ latent, or unobserved, leading to *unsupervised learning*, although it is also possible to make $\mathbf{z}$ partially-observed for *semi-supervised learning* [Kingma et al., 2014; Siddharth et al., 2017], or fully-observed for *supervised learning*.

Latent variable models that condition on an observed context, $\mathbf{y}$, with conditional distribution $p(\mathbf{x}, \mathbf{z} \mid \mathbf{y})$ are also considered to be generative models, provided that part of $\mathbf{z}$ is latent, and the distribution is defined in a way that $\mathbf{z}$ gives rise to $\mathbf{x}$. We refer to these models as *conditional generative models*.

2.1.3 Implicit models

We further divide generative models into those that are “explicit" (or “prescribed”) versus those that are “implicit” [Mohamed and Lakshminarayanan, 2016]. The models of the previous section are said to be *explicit*, since we directly model an exact functional form for the likelihood, or scoring process, $p_\theta(\mathbf{x})$. In an *implicit* generative model, on the other hand, we directly model the sampling process for $\mathbf{x}$, and this only indirectly defines $p_\theta(\mathbf{x})$ (which is typically intractable).

One well known implicit generative model is the generative adversarial network (GAN) [Goodfellow et al., 2014a]. In its simplest incarnation, it models the sampling process of an image datum, $\mathbf{x}$, as follows: first sample $\mathbf{z} \sim N(\mathbf{0}, I_{D\times D})$ for some $D \ll N$ from a standard multivariate normal, then calculate $\mathbf{x} = f_\theta(\mathbf{z})$, where f_θ is a shallow densely-connected feedforward NN.

The learning objective of implicit generative modeling is to match the assumed sampling process of the model to that of the empirical data distribution. In the simplest case, this can be interpreted as minimizing the Jensen–Shannon divergence between the data distribution and the model (see [Goodfellow et al., 2014a, Theorem 1]),

$$\theta^* = \text{argmin}_{\theta\in\Theta}\text{JSD}\{p_{\mathcal{D}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})\} \tag{2.2}$$

where

$$\text{JSD}\{p \parallel q\} \triangleq \frac{1}{2}\text{KL}\left\{p \,\middle\|\, \frac{p+q}{2}\right\} + \frac{1}{2}\text{KL}\left\{q \,\middle\|\, \frac{p+q}{2}\right\}.$$

It is not possible to directly optimize (2.2), as it depends on the implicit and intractable density, $p_\theta(\mathbf{x})$. Instead, a discriminator function is introduced whose purpose is to estimate the intractable likelihood ratio terms required by (2.2).

Implicit models may be necessary for situations where it is only feasible to simulate the sampling process of the data, such as in stochastic simulators for scientific modeling [van de Meent et al., 2018]. Nonetheless, there is theoretical and empirical evidence that implicit models such as GANs learn distributions with low-dimensional support [Arora et al., 2017; Arora and Zhang, 2017], and thus presumably are limited in their capability to generalize beyond the observed data. We do not consider implicit models further in this thesis.

Let us make clear the distinction between model, objective, algorithm, and framework, as we define them, which is also relevant to our later discussion of variational autoencoders (VAEs). We define the GAN model as, e.g., the sampling process described above. Alternatives exist, such as using a convolutional architecture [Radford et al., 2015] or an invertible neural network [Grover et al., 2018]. By the objective, we mean the divergence metric or other measure used to match the sampling process to the data distribution, such as the Jenson–Shannon divergence described above. Again, there are other possibilities that can be used independently of the model, such as the Wasserstein distance [Arjovsky et al., 2017]. By algorithm, we mean the optimization problem and method of optimization to optimize the objective. For instance, for GANs this typically involves SGD with adaptive stepsizes [Kingma and Ba, 2014] on two separate optimization problems involving a discriminator function that determines how well the sampling process is able to match the data distribution. By framework, we mean the combination of these three elements. When we simply refer to GAN or VAE, we will take that to mean the GAN or VAE model, and not the whole framework. It is common in the literature to conflate the term GAN with the complete framework. However, we believe it is important to at least distinguish the model from the learning or inferential procedure—many models can be used with many separate learning procedures, both of which are developed in parallel in the literature.

2.1.4 Discussion

Discriminative models have lower asymptotic error than generative models for regression and classification tasks, presumably because they do not require any assumption on the structure of $p(\mathbf{x})$. Indeed, deep NN discriminative models, parametrized by flexible NN function approximators and learnt from big data have produced state-of-the-art results in classification tasks [Krizhevsky et al., 2012; Hinton et al., 2012]. Generative models, however, often approach their higher asymptotic error faster [Ng and Jordan, 2002] and are to be preferred in low-data regimes.

Moreover, generative models can solve a number of AI tasks requiring a deeper level of "intelligence" than the regression and classification afforded by discriminative models. Most

obviously, generative models can sample new data points. This is more than a curiosity and has been used for practical application such as supersampling of images [Ledig et al., 2017]. Latent variable generative models are particularly useful and can be applied to tasks beyond simply associating inputs to outputs [Mohamed and Rezende, 2017] such as,

- recognizing the identity and location of objects [Eslami et al., 2016]
- predicting future states of the world [Kosiorek et al., 2018]
- disentangling the factors of variation giving rise to our observations [Siddharth et al., 2017]
- forming concepts in an unsupervised manner that are useful for reasoning and decision making [Lake et al., 2015]
- generating plans for the future [Igl et al., 2018]
- causal modeling and discovery [Louizos et al., 2017]

When the conditional distributions comprising a generative model are parametrized by NNs, the models are termed *deep generative models*. This poses a challenge for model learning, as we cannot analytically marginalize out the latent variables to produce the marginal, $p_\phi(\mathbf{x})$. In the following section we will see how model learning can be performed using the amortized VI paradigm. Amortized VI also makes use of NNs for inference.

Discriminative models do not provide any mechanism to express our uncertainty over the predictions. However, we can do so by reinterpreting the model weights as random variables. Suppose we have a discriminative classifier, $p(\mathbf{y} \mid \mathbf{x}; \phi)$. By placing a prior on the parameters, $p(\phi)$, we can transform the discriminative model into one that is conditionally generative, $p(\mathbf{y}, \phi \mid \mathbf{x}) = p(\mathbf{y} \mid \mathbf{x}, \phi)p(\phi)$. Instead of learning the parameters by, e.g., stochastic gradient descent, the problem is transformed to one of inference. As to be explained, one can draw approximate samples, $\{\phi_m\}_{m=1}^M$, from the posterior, $p(\phi \mid \mathbf{x}, \mathbf{y})$, by an MCMC method, and use those samples to produce M different distributions, $p(\mathbf{y} \mid \mathbf{x}, \phi_m)$ characterizing our uncertainty over the prediction distribution. Alternatively, by classical VI or EP, one can learn an approximation, $q_\psi(\phi)$, to the posterior, and use this to similarly characterize the uncertainty over prediction distributions. When the model is parametrized by a NN, this bridge from discriminative to generative models is known as a *Bayesian NN*, and will be used in Ch 4 for distributed learning of NNs.

In order to exploit the low asymptotic error of discriminative models, we may wish to train a model on more data than fits on a single machine. In these situations, we require learning systems

that can distribute and coordinate the work of learning across several machines. In one such setup, the model is replicated across W worker nodes, each of which receives a (possibly overlapping) portion of the data, and a master node. In the asynchronous SGD (A-SGD) [Dean et al., 2012] algorithm, each worker node requests the most recent parameters from the master, and after receiving this performs a gradient update on a minibatch from its portion of the data, sending back the resulting update to the parameters to the master node. While a worker is computing the gradient with respect to the current minibatch, other workers have potentially interacted with the master to update the current parameters, and thus the worker in question is working with out-of-date information. On average, each worker's update will be $W - 1$ steps behind. This is known as the stale-gradient problem. Again, this is a problem that can be addressed by Bayesian inference. We develop in Ch 4, a distributed Bayesian learning framework that lessens the severity of the stale-gradient problem. From a high-level, it does this by exchanging distributions over the parameters from the workers to the master, rather than simple gradient updates.

Thus, NNs are important components of both discriminative and generative models. They are powerful function approximators that are able to learn hierarchical, distributed, and often sparse, representations of their inputs [Bengio, 2009]. Deep NN discriminative models are the state-of-the-art in many classification and regression tasks. NNs are useful not just for modeling, but also for inference, as we will describe in the next two sections. Latent variable generative models require inference, and a scalable, generic inference framework known as amortized VI makes use of NNs for this purpose. We will also describe how inference can be useful for analyzing discriminative NN models.

2.2 Bayesian inference

We have discussed how latent variable generative models are of particular interest for probabilistic machine learning in that they enable the solution of many tasks beyond associating inputs and outputs. The inclusion of latent variables, however, necessitates inferential procedures for recovering their likely value. In this section, we describe two main families of approximate inference algorithms, and their relationship to our work.

Recall that a latent variable generative model supposes latent (unobserved) random variables, $\mathbf{z}$, that give rise to observed random variables, $\mathbf{x}$, through a generative process, $p(\mathbf{x}, \mathbf{z})$. The *prior*, $p(\mathbf{z})$, reflects our beliefs about the uncertainty in $\mathbf{z}$ before having made any observations. After

having observed $\mathbf{x}$, our beliefs can be updated to reflect this knowledge using Bayes' rule,

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})}{p(\mathbf{x})},$$

calculating what is called the *posterior* distribution. In Bayesian statistics, the posterior is considered to contain complete information about the unknown quantities $\mathbf{z}$ given knowledge of our observations, and is used to calculate any quantity of interest, $f(\cdot)$, about the latent generative process via

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] = \int f(\mathbf{z})p(\mathbf{z} \mid \mathbf{x})d\mathbf{z}, \tag{2.3}$$

which is known as performing *inference* on the model.

We speak of the generative model as a forward process. Typically it factors as $p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$, and so can be sampled from by ancestral sampling by first sampling $\mathbf{z}$, and then sampling $\mathbf{x}$ given $\mathbf{z}$. The calculation of the posterior is then, in a sense, the inverse process. We are given $\mathbf{x}$ and want to invert the model so that we can, e.g., sample from $\mathbf{z}$ given $\mathbf{x}$.

One central challenge of Bayesian statistics is to perform this inversion process in order to effect inference. The inversion itself is considered synonymous with the term inference. Given that the numerator is specified by the model, the difficulty is in calculating the denominator,

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

known as the *evidence*. This integration can only be done analytically—known as performing *exact inference*—for simple models, such as those for which $p(\mathbf{z} \mid \mathbf{x})$ is in the same distribution family as $p(\mathbf{z})$. We must be content with approximations in most interesting situations.

We now describe two families of approximate inference procedures, Markov chain Monte Carlo (MCMC) and variational inference (VI) that will be used in subsequent chapters. MCMC is a family of methods that treats inference as a sampling problem, and lets us draw approximate samples from the posterior to directly approximate (2.3). VI is another family of approximate inference methods that treats inference as an optimization problem, and learns an approximation to the posterior, which can be used to approximate (2.3) by, e.g., importance sampling.

2.2.1 MCMC

MCMC [Andrieu et al., 2003; Koller and Friedman, 2009, Ch 12.3] is a framework for generating approximate samples from a (potentially unnormalized) target distribution when we cannot do so directly. When used for Bayesian inference, the target distribution is the unnormalized posterior. MCMC involves constructing an iterative process, a so-called *Markov chain*, that samples from distributions that become closer and closer to the target as the process proceeds. Formally,

Definition 2.1. *A (discrete-time)* Markov chain *is a stochastic process satisfying the Markov property. That is, it is a countably infinite collection of random variables,*

$$\mathcal{M} = \{x_0, x_1, x_2, \ldots\}$$

where $x_0 \sim p^{(0)}(\cdot)$ *is drawn from an initial state distribution, the variables belong to the same space,* $x_t \in \mathcal{X}$*, and the dynamics of the process satisfy the Markovian transition model,* $g(x_t \mid \mathbf{x}_{\prec x_t}) = g(x_t \mid x_{t-1})$*, for* $t \geq 1$*.*

To generate a (truncated) Markov chain trajectory, we first sample $x_0 \sim p^{(0)}(\cdot)$. Then, we iterate the process by sampling $x_1 \mid x_0 \sim g(\cdot \mid x_0)$. This procedure is repeated T times, sampling $x_t \sim g(\cdot \mid x_{t-1})$, to produce the trajectory, $\hat{\mathcal{M}} = \{x_0, x_1, \ldots, x_T\}$.

Denote the distribution of x_t as $p^{(t)}(\cdot)$. From the chain dynamics, this distribution is defined recursively as,

$$p^{(t)}(x') = \int_{\mathcal{X}} p^{(t-1)}(x) g(x' \mid x) dx.$$

Assuming the process converges, one would expect

$$\begin{aligned} p^{(t)}(x') &\approx p^{(t+1)}(x') \\ &= \int_{\mathcal{X}} p^{(t)}(x) g(x' \mid x) dx \end{aligned}$$

with, intuitively, equality holding in the limit $t \to \infty$. This equilibrium is known as the stationary distribution,

Definition 2.2. *A distribution,* $\pi(\cdot)$*, is a stationary distribution for a Markov chain with transition* g *if it satisfies,*

$$\pi(x') = \int_{\mathcal{X}} \pi(x) g(x' \mid x) dx,$$

for all $x' \in \mathcal{X}$*.*

If we can construct a Markov chain that converges to a stationary distribution, $\pi(\cdot)$, that is equal to the desired target, then approximate samples can be drawn from the target by simulating the Markov chain trajectory. In the context of Bayesian inference, the trajectory can be simulated to sample $\{\mathbf{z}_n\}_{n=1}^N$ correlated samples that are approximately distributed according to the posterior, which can then be used to estimate (2.3) by naive Monte Carlo (MC),

$$\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})} [f(\mathbf{z})] \approx \sum_{n=1}^{N} f(\mathbf{z}_n).$$

In general, however, there is no guarantee that a stationary distribution exists for a Markov chain, that is, that the chain converges, and even if it does, that the stationary distribution is unique. In the case of the latter, the stationary distribution reached depends upon the starting distribution, $p^{(0)}(\cdot)$.

Fortunately, some (rather technical) sufficient conditions are known for providing the existence of a unique stationary distribution. In the case of when $\mathcal{X}$ is a finite-space, a Markov chain converges to a unique stationary distribution if it is irreducible and aperiodic [Andrieu et al., 2003]. From a high level, the property of irreducibility means that any state is reachable from any other state at all times, and the property of aperiodicity means that the chain does not get trapped in cycles. Analogous technical conditions can be defined when $\mathcal{X}$ is a continuous space [Stachurski, 2009, Theorem 8.2.14].

How can we construct Markov chains such that the target is the unique stationary distribution? One family of methods is based on constructing Markov transitions that are reversible,

Definition 2.3. *A Markov chain with transition, g is* reversible *if there exists a unique distribution π such that, for all $x, x' \in \mathcal{X}$,*

$$\pi(x)g(x' \mid x) = \pi(x')g(x \mid x')$$

This equation is termed detailed balance*.*

It can be shown very simply that if the Markov chain with transition g is irreducible and aperiodic (or the equivalent conditions for a continuous-space) and it satisfies detailed balance, then π is the unique stationary distribution. Integrating both sides of the detailed balance we have,

$$\begin{aligned}\int_{\mathcal{X}} \pi(x)g(x' \mid x)dx &= \int_{\mathcal{X}} \pi(x')g(x \mid x')dx' \\ &= \pi(x'),\end{aligned}$$

and so by definition, π must be the stationary distribution. By satisfying the sufficient conditions, the Markov chain converges to this unique distribution.

Metropolis–Hastings (MH) [Gilks et al., 1995] is one such MCMC algorithm based on this idea. We will use it in Ch 4 to draw samples from the potentially rare event of failure in a neural network. It works by breaking up the Markov transition,

$$g(x' \mid x) = g'(x' \mid x)A(x', x) \tag{2.4}$$

into a proposal, $g'(x' \mid x)$, and an accept-reject step with probability $A(x', x)$ to correct for the discrepancy between the proposal and a valid transition.

Substituting (2.4) into the detailed balance and rearranging,

$$\frac{A(x', x)}{A(x, x')} = \frac{\pi(x')}{\pi(x)} \frac{g'(x \mid x')}{g(x' \mid x)}. \tag{2.5}$$

Fixing the proposal, the acceptance probability can be defined as,

$$A(x', x) = \min \left\{ 1, \frac{\pi(x')}{\pi(x)} \frac{g'(x \mid x')}{g(x' \mid x)} \right\}, \tag{2.6}$$

to satisfy (2.5). So, to apply the MH algorithm, we decide upon a proposal, $g'(x' \mid x)$, and simulate the trajectory as previously described, using (2.4) and (2.6). It can be shown that a wide variety of proposals satisfy the technical conditions for the chain to have a unique stationary distribution (the existence is guaranteed by detailed balance) [Andrieu et al., 2003].

An application of MH is given in Ch 5 [Webb et al., 2018b], where we apply it to perform inference on NNs. Let us describe the context. We often desire that a discriminative NN model satisfies a certain property. For instance, it could be that the network should not change its classification in some neighbourhood of an input, or that the network tracks a reference function with some degree of fidelity. In such scenarios, it is possible to quantify the property with a function, $s(\cdot)$, for which the property of interest is violated if and only if $s(\mathbf{x}) \geq 0$ for $\mathbf{x} \in \mathcal{X}$ in some subset, $\mathcal{X}$, of the input domain.

Given a distribution $p(\cdot)$ over $\mathcal{X}$, we define a metric of how robust the NN is to the property as the probability that the property is violated under this input model,

$$\mathcal{I}[s, p] = \int_{\mathcal{X}} \mathbb{1}_{s(\mathbf{x}) \geq 0} p(\mathbf{x}) d\mathbf{x}. \tag{2.7}$$

The problem is that the event of failure is typically a rare one, and we cannot reliably estimate this integral using a naive MC estimate, drawing samples from $p(\mathbf{x})$. We apply an algorithm from the rare event estimation literature that breaks up the estimation of (2.7) into multiple successive events that are not rare, conditioned on the previous one, and MH is used at each step to sample from the event at that iteration. From a high level, multiple Markov chains are simulated with MH, using a cascade of target distributions, progressively directing the chains to the rare event of failure.

In passing, we note that, whilst it is beyond the scope of this thesis to give a detailed survey of MCMC methods, there are a wealth beyond the simple example of MH. For instance, Hamiltonian Monte Carlo (HMC) [Neal, 2011] is a type of MCMC method that uses Hamiltonian mechanics to reduce the correlation between successive samples relative to MH. In practice, this involves simulating Hamiltonian dynamics on an augmented space, using an accept-reject step to compensate for the bias introduced by the numerical solution to the diffferential equation.

By both using gradient information and making larger transitions in an augmented space, the mixing time is lessened. We discuss this further in §6.2.1.

2.2.2 Classical variational inference

Another class of approaches known collectively as *variational inference* (VI) [Jordan et al., 1999; Blei et al., 2016; Zhang et al., 2018], in its simplest form, supposes a family of distributions, $\mathcal{Q}$, over $\mathbf{z}$ and aims to find the member,

$$q^*_{\text{VI}}(\mathbf{z};\mathbf{x}) \triangleq \text{argmin}_{q\in\mathcal{Q}}\text{KL}\left\{q(\mathbf{z})||p(\mathbf{z}\mid\mathbf{x})\right\} \tag{2.8}$$

that is closest to the posterior in the KL-divergence sense. We refer to this formulation as *classical VI*. Once found, $q^*_{\text{VI}}(\mathbf{z};\mathbf{x})$ can be sampled from to directly estimate (2.3) by naive MC, or else combined with another MC approach such as importance sampling. In this way, the problem of inference is transformed into a problem of optimization.

We use the notation $q^*_{\text{VI}}(\mathbf{z};\mathbf{x})$ to denote that the optimal member of the variational family is implicitly a function of $\mathbf{x}$ through the optimization problem, despite that no $q \in \mathcal{Q}$ is a function of $\mathbf{x}$.

It is not possible, however, to directly optimize (2.8). The KL-divergence requires the calculation of the evidence, seen by expanding the divergence,

$$\text{KL}\left\{q(\mathbf{z})||p(\mathbf{z}\mid\mathbf{x})\right\} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\ln q(\mathbf{z})\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\ln p(\mathbf{x},\mathbf{z})\right] + \ln(p(\mathbf{x}))$$

which, circularly, is the problem that we are trying to solve by (2.8)!

The solution is to optimize an alternative objective—clearly the log-evidence term, $\ln(p(\mathbf{x}))$, is not a function of q and can be subtracted. Negating the resulting expression gives,

$$\mathcal{L}_{\text{ELBO}}\{q,\mathbf{x}\} \triangleq \mathbb{E}_q\left[\ln(p(\mathbf{x},\mathbf{z}))\right] - \mathbb{E}_q\left[\ln(q(\mathbf{z}))\right],$$

an expression known as the *evidence lower bound* (ELBO). Thus solving (2.8) is equivalent to minimizing the negative ELBO,

$$q^*(\mathbf{z}) \triangleq \text{argmin}_{q\in\mathcal{Q}} - \mathcal{L}_{\text{ELBO}}\{q,\mathbf{x}\}. \tag{2.9}$$

The ELBO can be given several useful interpretation. Firstly, interpreting the two terms in its definition, we see that $\mathbb{E}\left[\ln(p(\mathbf{x},\mathbf{z}))\right]$ rewards joint configurations of $(\mathbf{x},\mathbf{z})$ with high probability, permitting $\mathbf{x}$ to be explained or reconstructed, whereas $H(q) = -\mathbb{E}\left[\ln(q(\mathbf{z}))\right]$ rewards q with high entropy, i.e., being less informative, thus providing a regularizing effect.

A second interpretation of the ELBO is given by rewriting it as,

$$\mathcal{L}_{\text{ELBO}}\{q,\mathbf{x}\} = \mathbb{E}\left[\ln(p(\mathbf{x}\mid\mathbf{z}))\right] - \text{KL}\left\{q(\mathbf{z})||p(\mathbf{z})\right\}.$$

The first term, $\mathbb{E}\left[\ln(p(\mathbf{x} \mid \mathbf{z}))\right]$, is again a reconstruction term that rewards q that explain $\mathbf{x}$ well, and the second term penalizes q that deviate from the prior, $p(\mathbf{z})$, that is, that are as simple as the prior, thus providing a regularizing effect.

The reason why the ELBO is called the evidence lower bound, is that it lower-bounds the log-evidence,

$$\begin{aligned}\ln\left(p\left(\mathbf{x}\right)\right) &= \mathrm{KL}\left\{q(\mathbf{z})||p(\mathbf{z} \mid \mathbf{x})\right\} + \mathcal{L}_{\mathrm{ELBO}} \\ &\geq \mathcal{L}_{\mathrm{ELBO}},\end{aligned}$$

noting that the KL-divergence is always nonnegative. This fact will prove useful in the next section.

In classical variational inference, the simplest variational family, $\mathcal{Q}$ used is the *mean-field variational family*,

$$q(\mathbf{z}) \triangleq \prod_{d=1}^{D} q_d(z_d),$$

assuming each latent variable z_d is independent of the remainder.

One simple algorithm to solve (2.9) is coordinate ascent variational inference (CAVI) [Blei et al., 2016]. It works by iteratively optimizing each q_d in the mean-field family, holding the others fixed. Consider the dth latent variable z_d. The optimal $q_d(\cdot)$ is,

$$q_d^*(z_d) \propto \exp\left\{\mathbb{E}_{q_{-d}}\left[\ln\left(p(z_d \mid \mathbf{z}_{-d}, \mathbf{x})\right)\right]\right\}, \tag{2.10}$$

where $\mathbf{z}_{-d} \triangleq \{z_1, z_2, \ldots, z_{d-1}, z_{d+1}, \ldots, z_D\}$, $q_{-d}(\mathbf{z}_{-d}) \triangleq \prod_{i \neq d} q_i(z_i)$, and $p(z_d \mid \mathbf{z}_{-\mathbf{d}}, \mathbf{x})$ is known as the *complete conditional*. Applying this algorithm, we sample a d at random, update z_d using (2.10), and repeat until the ELBO has satisfied some termination criterion, such as that its relative change is less than a small constant, δ.

A requirement of this method is that the optimal expression for $q_d^*(\cdot)$ is calculable in closed-form, for which it turns out is possible when the complete conditional belongs to the exponential family. Other classical variational inference algorithms require similar assumptions of conjugacy and often require lengthy model specific derivations to determine their update equations. This is one motivation for amortized VI, to be explained in 2.2.5.

2.2.3 Expectation propagation

Expectation propagation [Minka, 2001; Opper and Winther, 2005; Gelman et al., 2014] is a type of variational inference, in the broader sense of the term, that differs from classical

VI (and amortized VI in its simplest form) by finding the member of the variational family minimizing the reverse KL-divergence,

$$q^*_{\text{EP}} \triangleq \arg\min_{q\in\mathcal{Q}} \text{KL}\left\{p(\mathbf{z} \mid \mathbf{x})||q(\mathbf{z})\right\} \tag{2.11}$$

rather than the forward KL-divergence of (2.8). In the method, it is assumed we have a factorization,

$$p(\mathbf{z} \mid \mathbf{x}) \propto \prod_{k=1}^{K} f_k(\mathbf{z})$$

and that the variational approximation factors as,

$$q(\mathbf{z}) \propto \prod_{k=1}^{K} g_k(\mathbf{z}).$$

The terms, $\{f_k\}$ are known as the target pieces, and the terms $\{g_k\}$ the site approximations. It is further assumed that the sites belong (up to normalization) to the same exponential family, for example,

$$g_k(\mathbf{z}) \propto h(\mathbf{z})g(\nu_k)\exp(\nu_k^T u(\mathbf{z}))$$

for $k = 1, 2, \ldots, K$.

The algorithm works as follows. We choose a site, g_k, and form the so-called *cavity distribution*,

$$q_{-k}(\mathbf{z}) \propto \frac{q(\mathbf{z})}{g_k(\mathbf{z})},$$

which can be done by simply subtracting the natural parameters,

$$q_{-k}(\mathbf{z}) \propto h(\mathbf{z})g(\nu_{-k})\exp(\nu_{-k}^T u(\mathbf{z})),$$

where $\nu_{-k} = \nu - \nu_k$. Then we form the *tilted distribution* as,

$$q_{\backslash k}(\mathbf{z}) \propto f_k(\mathbf{z})q_{-k}(\mathbf{z}).$$

Effectively, in doing so, we have replaced the effect of the kth site approximation with the kth target piece.

Finally, we update the site, g_k, so that, KL $\left\{q_{\backslash k}(\mathbf{z})||q(\mathbf{z})\right\}$ is minimized. It turns out that when q belongs to an exponential family, the solution to this is equivalent to,

$$\mathbb{E}_{q_{\backslash k}}[u(\mathbf{z})] = \mathbb{E}_q[u(\mathbf{z})],$$

that is, that the expected sufficient statistics match. This method depends on being able to determine the sufficient statistics of q under the tilted distribution analytically, which depends in

general on the form of $\{f_k\}$. This procedure is then repeated for the other site approximations until convergence is attained.

We note that despite minimizing the local divergences in the scope of each site, there is no theoretical guarantee that the global divergence of (2.11) is minimized. However, EP has been demonstrated to converge in practice on many problems.

EP can be thought of as a message passing algorithm, with each site approximation passing a message to all other sites at each step. Indeed, loopy belief propagation [Koller and Friedman, 2009, Ch 11], a message passing algorithm on graphical models, is a specific case of EP. We note that coordinate ascent VI can also be thought of as a message passing algorithm—the solution for $q_d(\cdot)$ can be thought of as the calculation of the message for "node d" given the current state of information from the other $D-1$ nodes. This message is then passed to the other nodes, repeating the process synchronously or asynchronously.

In Ch 4, we give an application of an extension of the basic EP algorithm presented here to distributed Bayesian learning. Our framework is particularly useful for learning Bayesian NNs. NN discriminative models achieve state-of-the-art performance in many tasks when trained on big data. However, special methods are required to train these models when the model parameters and/or the dataset is too large to fit on a single machine. Our work focuses on the latter case, and divides the data between a number of worker nodes, where there is a single target piece per division of the data. Each worker then calculates its site update using a modification of EP that is designed to be robust to lags in communication from the other workers. After convergence, we can take the mean of the variational approximation as a point estimate of the parameters, and use the variance of the same approximation to estimate our uncertainty in the outputs.

2.2.4 Modern variational inference

Classical VI suffers several shortcomings. Restrictive assumptions must be placed on the form of the distributions in the model, usually conditions of conjugacy, and lengthy derivations are often required to determine the closed-form update equations. In addition, classical VI does not permit model learning other than by lifting the problem to be one of inference, and requires optimization each time a new $\mathbf{x}$ is encountered. These considerations motivate the framework of modern VI [Rezende et al., 2014; Kingma and Welling, 2014; Zhang et al., 2018].

First, let us modify the problem setup. Assume without loss of generality that the model is a function of deterministic parameters, ϕ, and that it decomposes as,

$$p_\phi(\mathcal{D}, \mathbf{z}) \triangleq \prod_{n=1}^{N} p'_\phi(\mathbf{x}_n, \mathbf{z}).$$

We modify the optimization problem accordingly to,

$$\min_{\phi,\psi} \mathbb{E}_{\mathbf{x}_n \in \mathcal{D}} \left[\mathbb{E}_{\mathbf{z} \in q_\psi(\cdot)} \left[\ln \left(q_\psi \left(\mathbf{z} \right) \right) - \ln \left(p_\phi \left(\mathbf{x}_n, \mathbf{z} \right) \right) \right] \right].$$

Modern VI, also known as Monte Carlo VI, optimizes this by gradient descent with respect to $\theta = \{\psi, \phi\}$, estimating the gradients by the method of Monte Carlo. This poses two problems. Firstly, how can we take the expectation over $\mathbf{x}_n \sim \mathcal{D}$ given that the dataset may be large? Secondly, how can we take the gradient with respect to θ, which involves exchanging a derivative and an expectation?

The first problem can be solved by taking a stochastic gradient, that is, approximating the expectation over the N data points by a minibatch of $N' << N$ samples from $\mathcal{D}$. We note that stochastic gradients have been previously used in forms of classical VI [Hoffman et al., 2013]. Also, the minibatch size must be large for effective parameter updates in this simple setting when the variational family is not directly a function of $\mathbf{x}$. (The case of when the variational family is a function of $\mathbf{x}$ is known as amortization and will be discussed in the sequel.) When the dataset is on the order of 1000 samples or less, we can forego minibatching.

The second problem is more difficult, and will be dealt with using one of two tricks.

Score-function estimators. Ignoring for the moment the expectation over $\mathbf{x}$, we want to take the gradient of

$$\mathcal{L}(\mathbf{x}) \triangleq \int \left(\ln \left(p_\phi \left(\mathbf{x}, \mathbf{z} \right) \right) - \ln \left(q_\psi \left(\mathbf{z} \right) \right) \right) q_\psi(\mathbf{z}) d\mathbf{z}.$$

The gradient with respect to the model parameters is,

$$\begin{aligned} \nabla_\phi \mathcal{L}(\mathbf{x}) &= \int \nabla_\phi \ln \left(p_\phi(\mathbf{x}, \mathbf{z}) \right) q_\psi(\mathbf{z}) d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z} \sim q(\cdot)} \left[\nabla_\phi \ln \left(p_\phi(\mathbf{x}, \mathbf{z}) \right) \right]. \end{aligned}$$

The gradient with respect to the inference network parameters is,

$$\begin{aligned} \nabla_\psi \mathcal{L}(\mathbf{x}) &= \int \left(\ln \left(p_\phi \left(\mathbf{x}, \mathbf{z} \right) \right) \nabla_\psi q_\psi \left(\mathbf{z} \right) - \nabla_\psi \left(q_\psi(\mathbf{z}) \ln \left(q_\psi \left(\mathbf{z} \right) \right) \right) \right) d\mathbf{z} \\ &= \int \left(\ln p_\phi \left(\mathbf{x}, \mathbf{z} \right) - \ln \left(q_\psi \left(\mathbf{z} \right) \right) \right) \nabla_\psi q_\psi \left(\mathbf{z} \right) d\mathbf{z} \\ &= \int \left(\ln p_\phi \left(\mathbf{x}, \mathbf{z} \right) - \ln \left(q_\psi \left(\mathbf{z} \right) \right) \right) q_\psi(\mathbf{z}) \nabla_\psi \ln \left(q_\psi \left(\mathbf{z} \right) \right) d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot)} \left[\left(\ln \left(p_\phi \left(\mathbf{x}, \mathbf{z} \right) \right) - \ln \left(q_\psi \left(\mathbf{z} \right) \right) \right) \nabla_\psi \ln \left(q_\psi(\mathbf{z}) \right) \right], \end{aligned}$$

where we have used the fact that the expected score is zero, $\mathbb{E}_p[\nabla \ln p(\mathbf{x})] = 0$, and the so-called "log-derivative trick" that $\nabla p(\mathbf{x}) = p(\mathbf{x})\nabla \ln(p(\mathbf{x}))$. Estimators for the inference network gradient based on the log-derivative trick are known as score-function estimators [Schulman et al., 2015].

Both gradients can be estimated by naive MC,

$$\nabla_\phi \mathcal{L}(\mathbf{x}) \approx \frac{1}{M}\sum_{m=1}^{M} \nabla_\phi \ln\left(p_\phi\left(\mathbf{x}, \mathbf{z}_m\right)\right)$$
$$\nabla_\psi \mathcal{L}(\mathbf{x}) \approx \frac{1}{M}\sum_{m=1}^{M} \hat{L}_m \nabla_\psi \ln\left(q_\psi\left(\mathbf{z}_m\right)\right),$$

where $\hat{L}_m = \ln\left(p_\phi(\mathbf{x}, \mathbf{z}_m)\right) - \ln\left(q_\psi(\mathbf{z}_m)\right)$ is known as the learning signal.

The estimate for the model gradient is unproblematic. However, the estimate of the inference network gradient has high variance due to the high variability of the learning signal—is has unbounded magnitude and will typically be large during the initial phase of learning when $q_\psi(\mathbf{z}_m)$ diverges greatly from $p_\phi(\mathbf{x} \mid \mathbf{z}_m$. Indeed, it was originally surmised that score-function estimators were infeasible for learning due to the high variance encountered.

Using the fact that the expected score is zero, a term, κ, that is not a function of the latent variable (but may be a function of the input) known as a *control variate* can be subtracted from the learning signal without effecting equality,

$$\nabla_\psi \mathcal{L}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\psi(\cdot)}\left[(\hat{L} - \kappa(\mathbf{x}))\nabla_\psi \ln\left(q_\psi(\mathbf{z})\right)\right].$$

An effective control variate is highly correlated with the learning signal. One method uses a feedforward NN that tracks the learning signal [Mnih and Gregor, 2014].

Other control variates have been devised. For instance, in the method of Mnih and Rezende [2016], a so-called *Monte Carlo objective*,

$$\mathcal{L}^K(\mathbf{x}) \triangleq \mathbb{E}_{\prod_k q_\psi(\cdot)}\left[\ln\left(\frac{1}{K}\sum_{k=1}^{K}\frac{p_\phi(\mathbf{x}, \mathbf{z}_k)}{q_\psi(\mathbf{z}_k)}\right)\right], \tag{2.12}$$

is optimized. This multi-sample objective is provably a tighter bound on the log-evidence, tightening as $K \to \infty$ [Burda et al., 2016]. Repeating an analogous procedure to the single-sample objective (see [Mnih and Rezende, 2016, Appendix D]),

$$\nabla_\phi \mathcal{L}^K(\mathbf{x}) \approx \sum_{k=1}^{K} \tilde{w}_k \nabla_\phi \ln(p_\phi(\mathbf{x}, \mathbf{z}_k))$$
$$\nabla_\psi \mathcal{L}^K(\mathbf{x}) \approx \sum_{k=1}^{K}\left(\hat{L}(\mathbf{z}_{1:K})\nabla_\psi q(\mathbf{z}_k) - \tilde{w}_k \nabla_\psi \ln\left(q(\mathbf{z}_k)\right)\right),$$

where

$$\tilde{w}_k \triangleq \frac{p_\phi(\mathbf{x}, \mathbf{z}_k)/q_\psi(\mathbf{z}_k)}{\sum_{k=1}^K p_\phi(\mathbf{x}, \mathbf{z}_k)/q_\psi(\mathbf{z}_k)}$$

is a normalized importance weighting term, and

$$\hat{L}(\mathbf{z}_{1:K}) = \ln\left(\frac{1}{K}\sum_{k=1}^K \frac{p_\phi(\mathbf{x}, \mathbf{z}_k)}{q_\psi(\mathbf{z}_k)}\right)$$

is analogous to the learning signal for the single-sample objective. Similary to the single-sample case, the gradient estimate for ϕ poses no problem, being the convex combination of log-gradient terms. However, the gradient estimate for ψ is expected to have high variance from two sources.

Firstly, and similarly to the single-sample case, the learning signal is unbounded in magnitude and is expected to be large during the initial phase of learning when all samples from the proposal explain the data poorly. Secondly, and differently from the single-sample case, the same learning signal is applied to all samples from the proposal—consequently, the gradient for a sample that scores highly under the model is not given any more weight than another sample that scores poorly, within the set of K proposal samples. This is in contrast to the second term that assigns credit to the gradients by multiplying by normalized importance weights.

Mnih and Rezende [2016] propose to remedy these two sources of variance by subtracting a baseline based on previous estimates of $\hat{L}(\mathbf{z}_{1:K})$, as well as subtracting a term from the learning signal for each $\mathbf{z}_k$ that is highly correlated with the learning signal but is only a function of $\{\mathbf{z}_j\}_{j\neq k}$.

For score-function estimators, control variates are essential to so reduce the variance on the gradient estimates that model learning is feasible.

Pathwise estimators. Another technique for estimating the parameter gradients is based on the following trick. Suppose that we can express our latent variable $\mathbf{z}$ in terms of a function, f_ψ, and some random noise, ϵ, that is not a function of the parameters. That is, suppose $\mathbf{z} = f_\psi(\epsilon)$. For example, if $\mathbf{z}$ has a diagonal multivariate normal distribution, we could use $\mathbf{z} = \mu_\psi(\mathbf{x}) + \sigma_\psi(\mathbf{x}) \otimes \epsilon$, where ϵ is i.i.d. standard Gaussian noise. Using this so-called "reparametrization trick," we can now conveniently exchange the order of differentiation and expectation to estimate the inference parameters' gradient,

$$\begin{aligned}
\nabla_\psi \mathcal{L}(\mathbf{x}) &= \nabla_\psi \mathbb{E}_{p(\epsilon)}\left[\ln q_\psi(\mathbf{z}(\epsilon)) - \ln p_\phi(\mathbf{x}, \mathbf{z}(\epsilon))\right] \\
&= \mathbb{E}_{p(\epsilon)}\left[\nabla_\psi \left(\ln q_\psi(\mathbf{z}(\epsilon)) - \ln p_\phi(\mathbf{x}, \mathbf{z}(\epsilon))\right)\right] \\
&\approx \frac{1}{M}\sum_{m=1}^M \nabla_\psi \left(\ln q_\psi(\mathbf{z}(\epsilon_m)) - \ln p_\phi(\mathbf{x}, \mathbf{z}(\epsilon_m))\right).
\end{aligned}$$

The estimator for the gradient with respect to ϕ is the same as for score-function estimators. Reparametrization gradients exist for a variety of continuous distributions [Figurnov et al., 2018], including relaxations of discrete-valued ones [Maddison et al., 2017b; Jang et al., 2017]. They are easily formed for normalizing flows, to be discussed in the next section. Truely discrete distributions, however, require score-function estimates at present, due to the requirement that we must be able to differentiate through the log-density with respect to its input. Although there does not exist a proof, it is commonly held wisdom that pathwise estimators have significantly lower variance than score-function ones (using control variates) in practice [Ruiz et al., 2016].

Modern VI thus enables scalable inference with the use of stochastic gradients, and generic inference by the use of Monte Carlo gradient estimates, only requiring sampling, the calculation of certain gradient terms (which can be performed by automatic differentiation in a modern deep learning software framework), and techniques for reducing the variance of the MC gradients. As a side-product, we get model learning "for free." NNs can be used in both the model and the variational family, although this is atypical in the basic setup where the variational family only implicitly conditions on the data.

Although not widely used in practice, alternatives to the KL-divergence have been developed. Take the example of Rényi's α-divergence [Li and Turner, 2016], defined as

$$D_\alpha \left\{p||q\right\} \triangleq \frac{1}{\alpha - 1} \ln \int_{\mathcal{X}} p(\mathbf{x})^\alpha q(\mathbf{x})^{1-\alpha} d\mathbf{x},$$

where $\alpha > 0$, $\alpha \neq 1$ for two distributions on $\mathcal{X}$. This metric recovers the KL-divergence in the limit as $\alpha \to 1$ and a function of the Hellinger distance for $\alpha = 0.5$, amongst others. Replacing the KL-divergence in the derivation of the ELBO results in the new objective,

$$\mathcal{L}_\alpha \triangleq \frac{1}{\alpha - 1} \ln \mathbb{E}_{q_\psi} \left[\left(\frac{p_\phi(\mathbf{x}, \mathbf{z})}{q_\psi(\mathbf{z})} \right)^{1-\alpha} \right],$$

which is optimized similarly to the ELBO.Interestingly, the procedure also works for $\alpha < 0$, in which case $\mathcal{L}_\alpha$ is an upper bound on the model evidence.

Similar extensions to modern VI have been developed for the more general family of f-divergences [Bamler et al., 2017], as well as the Wasserstein distance [Ranganath et al., 2016; Liu and Wang, 2016]. It remains an open question in which scenarios such measures should be prefered to the KL-divergence.

In another line of research, the objective itself has been modified. In the importance weighted autoencoder, the variational family is used as a proposal for an importance weighted estimate of the marginal log-likelihood. This is the Monte Carlo objective of (2.12), albeit used

with a pathwise rather than a score-function estimator. In the filtering variational objective [Maddison et al., 2017a; Le et al., 2018; Naesseth et al., 2018], the variational family is used as a proposal for sequential Monte Carlo, the algorithm in a sense being "differentiated through." Such methods have the advantage of providing a different bias/variance tradeoff for the model parameter gradient estimates, compensating for some of the lack of expressivity in the variational family under consideration.

2.2.5 Amortized variational inference

In some circumstances, we require to solve a large number of related inference problems, or require to solve, at a later time, additional inference problems similar to the one at hand. For instance, consider the VAE model,

$$p_\phi(\mathcal{D}, \mathbf{z}) = \prod_{d=1}^{D} p_\phi(\mathbf{x}_d \mid \mathbf{z}_d) p(\mathbf{z}_d),$$

where $\mathcal{D}$ is typically a dataset of images, and p_ϕ is a decoder that reconstructs a distribution over the image $\mathbf{x}_d$ given a latent code, or encoding, $\mathbf{z}_d$. Notice that each latent $\mathbf{z}_d$ is local to each datum $\mathbf{x}_d$, and that the inference problems are connected through the shared weights ϕ. From a graphical model perspective, the $\{\mathbf{x}_d, \mathbf{z}_d\}$ exist on a plate, and ϕ is a global variable outside the plate. We ought to share the statistical strength across similar data while learning ϕ, in the process solving a number of related inference problems.

Amortized inference [Dayan et al., 1995; Rezende et al., 2014; Kingma and Welling, 2014] extends modern VI for this purpose by making the variational family explicitly a function of the data, forming what is known as an *inference network*. In the case of the VAE, the inference network takes the form,

$$q_\psi(\mathbf{z} \mid \mathcal{D}) = \prod_{d=1}^{D} q_\psi(\mathbf{z}_d \mid \mathbf{x}_d).$$

Plugging this definition into, e.g., the ELBO, and performing learning as per modern VI allows us to simultaneously solve the related inference problems for each datum whilst performing model learning on the global parameters. After learning, we can approximate the posterior, $p_\phi(\mathbf{z}'|\mathbf{x}', \mathcal{D})$, for a datum $\mathbf{x}'$ not seen during learning but similar to the dataset with the inference network, $q_\psi(\mathbf{z}'|\mathbf{x}')$.

NNs are not typically used with non-amortized modern VI (althought they are not precluded from doing so). In contrast, NNs *are* widely used with amortized VI, both for constructing the required inference network approximating the posterior—which must necessary model the complex relationship between the variable that is conditioned on and the distribution of the latent

variables—and for modeling. Modern amortized VI has opened up a new class of models known as *deep generative models* including the VAE and others [Gregor et al., 2015; Eslami et al., 2016; Bornschein et al., 2017] that make use of deep-learning innovations for discriminative modeling.

The performance of both inference amortization and model learning depends on how closely our inference network can approximate the true posterior for the current model parameters, ϕ. Clearly this is true for inference amortization: the quality of inference amortization can be measured by $\mathrm{KL}\left\{q(\mathbf{z} \mid \mathbf{x}) || p(\mathbf{z} \mid \mathbf{x})\right\}$. With regards to model learning, we see from the relation $\ln(p(\mathbf{x})) = \mathrm{KL}\left\{q(\mathbf{z} \mid \mathbf{x}) || p(\mathbf{z} \mid \mathbf{x})\right\} + \mathcal{L}_{\mathrm{ELBo}}(\mathbf{x})$ that the gradient update to the model parameters will be more in the direction of the true marginal log-likelihood the smaller is the gap between the inference network and true posterior. If the inference network is poorly constructed so this gap can never be sufficiently small, the model learning will be biased. In the next section and the following chapter, we discuss two aspects of suitably designing inference networks. Both the factorization of the distribution of q and the individual distributions for each factor must be appropriately chosen for amortized VI to work well in practice.

Amortized inference can also be motivated from a cognitive perspective [Gershman and Goodman, 2014]. The human brain appears to be able to cache and compose the results of previous inferences to answer new queries. It can be demonstrated from simple experiments that the answer to a query can be predicted from a person's answer to a simpler query if that simpler query occured first and requires a subinference of the more complex one. *Amortization expresses the experimentally observable interaction between inference and memory*.

At a high level, amortization is conceptually different from distillation [Hinton et al., 2015]. Distillation involves the compression of the "knowledge" in an ensemble of models trained by supervised learning on the same task. The motivation is typically to speed up inference on the deployed model without degrading performance. In contrast, in amortization, the knowledge is "distilled" across multiple tasks for a single model.

2.2.6 Discussion

In this section, we have introduced the three inference methods used in the following three paper chapters. Here, we discuss the relative merits and disadvantages of each.

In Ch 3, we present an algorithm for designing better inference networks in order to improve both inference amortization and model learning. Amortized VI is a general purpose and scalable inference method. It has the advantage over classical VI and EP of not requiring model-specific derivations, and the advantage over MCMC of faster convergence in practice. Moreover, the

inference amortization can be used to perform model learning of deterministic parameters, which allows us to incorporate powerful NNs into the model—take the example of the VAE, which uses a NN to learn a compact distributed representation from its high-dimensional image observations. Amortized VI requires flexible NN density estimators to be effective, which we outline in the next section.

EP is well suited for the distributed Bayesian learning framework of Ch 4 for a number of reasons. The reverse direction of the KL-divergence of the EP objective in (2.11), means that the mismatch between $p(\mathbf{z}|\mathbf{x})$ and $q(\mathbf{z})$ is not penalized when $q(\mathbf{z}) = 0$. The consequence of this is that when q is a simple parametric form such as a multivariate normal distribution, it will tend to fit one of the modes of $p(\mathbf{z}|\mathbf{x})$. In contrast, the forward KL-divergence of the classical VI objective in (2.8) penalizes mismatch from p for all $p(\mathbf{z}|\mathbf{x}) > 0$. This tends to lead to solutions that are "mass covering" and overdispersed, with a mode between the true modes of p, if p is multimodal. Consider our application of learning Bayesian NNs. We learn a variational approximation to the model parameters, and afterwards take the mean of this multivariate normal variational approximation as our point estimate of the parameters. If this does not match a mode of the posterior then the solution will be poor. This is one reason that EP is well-suited for this application over classical VI. Also, the natural interpretation of the algorithm as one based on message passing makes it well-suited for *distributed* learning. We note that MCMC has also been used for learning of Bayesian NNs [Neal, 2012].

MCMC is well suited to the application of measuring NN robustness in Ch 5 because the problem is often defined on a large dimensional input space and MCMC scales relatively favourably in the input dimension. There is another family of methods for rare-event estimation based on importance sampling known as the *cross-entropy method* [De Boer et al., 2005]. However, as we discovered, they do not scale in practice to the large input dimensions of NN image classifier problems, which can be in the thousands. The method requires estimating a proposal density for importance sampling at each level of event rareness, and suffers from the curse of dimensionality. In contrast, adaptive multi-level splitting (AMLS) [Guyader et al., 2011], which is based on MCMC inherits the favourable scaling of the MCMC technique chosen. For instance, it is known that MH with a random walk proposal takes $O(D^2)$ computation time for the Markov chain to reach a nearly independent point, where D is the input dimension [Neal, 2011]. On the other hand, our method also inherits the challenging aspects of MCMC, namely that it is hard to assess convergence in the trajectory of a Markov chain and to construct a Markov transition with quick convergence. Our strategy is simply to run the chains for a long burn-in

time, although we do suggest in Ch 6 how one might construct a faster Markov transition, which would be required for scaling the method to ImageNet size models.

We point out that the inference methods presented apply more broadly than to just Bayesian inference. MCMC can be used to sample from an unnormalized density, not necessarily the posterior. VI in its many forms can be used to match a density estimator to a target distribution, which is not necessarily the posterior. The techniques can be used to estimate a probabilistic expectation, which does not necessarily have to be over a posterior distribution. Indeed, this is how inference is used in Ch 5, where the target distribution is defined by the region of failure of the NN under the property to be verified.

On a historical note, many of the inference methods introduced in this chapter have their genesis in the statistical physics literature. MCMC methods were originally introduced to sample from models arising in statistical physics [Metropolis et al., 1953]. Also, variational inference with the mean-field approach was introduced to model spin glasses, which are types of disordered magnets [Opper and Saad, 2001]. Dayan et al. [1995] were the first to introduce the concept of the inference network, or "recognition model" as they termed it. They draw a connection between the alternative explanations a latent variable generative model makes of the data to the configurations of a physical system, and connect learning to the principle of minimum energy. Their model, the Helmholtz machine is learnt by the wake-sleep algorithm, which differs from algorithms such as SGD on the ELBO in that it optimizes a separate objective for the model and inference network.

2.3 Neural distribution representation

Recall that one of the main criteria determining the performance of amortized VI is whether there exists an inference network $q_\psi(\mathbf{z} \mid \mathbf{x})$ from the family $\psi \in \Psi$ such that $\mathrm{KL}\{q_\psi(\mathbf{z}|\mathbf{x}) \parallel p_\phi(\mathbf{z}|\mathbf{x})\} \approx 0$ for each ϕ during optimization of the ELBO objective. NNs are useful for amortized VI in that they can be used to represent flexible families of variational approximations in order that this holds. NNs distribution representations are also useful for modeling, since they permit us to learn distributed representations of our model variables, making use of the powerful function approximation power of deep architectures.

In this section, we give an overview of the details of different techniques for constructing such distribution representations from NNs. We begin with a discussion of two facets of representation which are, roughly speaking, the dependency structure of the representation and the conditional distribution of each variable. We then show how to construct univariate conditional distributions with simple parametric forms, like the Gaussian or Bernoulli distribution, whose parameters

are a complex nonlinear mapping of the parent variables. Next, we discuss how this method can be extended to multivariate conditional distributions. Following this, we show how to construct multimodal distributions from simple parametric forms, i.e. mixture models. And finally, we describe a set of methods known as normalizing flow for producing a rich family of representations from random noise and bijective transformations.

2.3.1 Factorization and parametrization

Denote the target distribution to be represented as $p^*(\mathbf{x} \mid \mathbf{y})$, where the input variables are $\mathbf{x} = \{x_1, x_2, \ldots, x_N\} \in \mathcal{X}$, the variables being conditioned upon, $\mathbf{y} = \{y_1, y_2, \ldots, y_M\} \in \mathcal{Y}$, and it may be the case that $\mathbf{y} = \emptyset$. Both $\mathbf{x}$ and $\mathbf{y}$ may comprise continuous or discrete valued variables (or a mixture), and we use the term *distribution* to encompass both densities and mass functions. We denote our representations as belonging to a family of distributions, $\mathcal{P} = \{p_\phi(\mathbf{x} \mid \mathbf{y}) \mid \phi \in \Phi\}$, with learnable parameters ϕ, hopefully containing a member close to $p^*(\mathbf{x} \mid \mathbf{y})$.

We draw a distinction between two facets of representation: *factorization* and *parametrization*. Suppose our representation is of the form,

$$p_\phi(\mathbf{x} \mid \mathbf{y}) = \prod_{n=1}^{N} p_{\phi_n}(x_n \mid \mathbf{x}'_n, \mathbf{y}) \tag{2.13}$$

where each $\mathbf{x}'_n$ is an arbitrary subset of $\mathbf{x} \setminus \{x_n\}$. "Factorization" refers to the way that (2.13) is broken up into factors, $\{p_{\phi_n}\}$, and reflects the assumptions on the structure in a distribution. On the other hand, "parametrization," as we define it, refers to the specification of the functional forms for each factor p_{ϕ_n}.

In the case of (2.13), each factor, p_{ϕ_n}, is a conditional distribution—such factorizations are known as *Bayesian networks* (BNs) (more strictly, *conditional Bayesian networks* when $\mathbf{y} \neq \emptyset$). Another type of factorization,

$$p_\phi(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z(\mathbf{y})} \prod_{n=1}^{N'} \psi_n(\mathbf{x}_n, \mathbf{y}) \tag{2.14}$$

where $\mathbf{x}_n$ is an arbitrary subset of $\mathbf{x}$, expresses p_ϕ as a product of N' unnormalized factors, ψ_n (with $N' \neq N$ in general) whose normalization constant is,

$$Z(\mathbf{y}) = \int_{\mathcal{X}} \prod_n \psi_n(\mathbf{x}_n, \mathbf{y}) d\mathbf{x}. \tag{2.15}$$

This type of factorization is known as a *conditional random field* (CRF) when $\mathbf{y} \neq \emptyset$ and a *Markov random field* (MRF), otherwise.

The theory of probabilistic graphical models (PGMs) [Koller and Friedman, 2009] can be used to associate a graph with these two types of factorizations, and from that read off

conditional independence relationships that hold in any distribution factoring according to these graphical structures. For instance, in a Bayesian network, (using the notation of (2.13)) we can associate a directed acyclic graph with an edge $x_n \leftarrow z$ if and only if $z \in \mathbf{x}'_n$ or $z \in \mathbf{y}$. The conditional independence relationships,

$$x_n \perp \textsc{NonDescendants}_{\mathcal{G}}(x_n) \mid \textsc{Parents}_{\mathcal{G}}(x_n), \quad n = 1, 2, \ldots, N,$$

hold in any distribution factoring over the same BN structure, $\mathcal{G}$. In fact, there are many more conditional independence relationships one can read off the graph using the concept of *d-separation* (see [Koller and Friedman, 2009] and [Webb et al., 2018a, A.1]). Conversely, any distribution that satisfies these conditional independence relationships must factor according to the BN structure. Thus, due to this equivalency, one can think of the graph as a compact structure for representing the dependency structure of the distribution.

Importance of the factorization. Let us restrict our attention to BNs and consider the importance of the factorization of our representation. If the factorization chosen for p_ϕ implies conditional independence relationships that do not hold in the target distribution, p^*, then it is not possible that $p_\phi(\mathbf{x}|\mathbf{y}) = p^*(\mathbf{x} \mid \mathbf{y})$ for any choice of the factors, even when each factor is a universal distribution estimator, i.e., is able to arbitrarily closely approximate any conditional distribution over the domain of its inputs. This occurs, roughly speaking, when the graphical structure is missing edges. In the context of amortized VI, if the factorization chosen for the inference network, $q_\psi(\cdot|\mathbf{x})$, makes conditional assumptions that do not hold in the true posterior, $p_\phi(\cdot|\mathbf{x})$, then it is likely that $\mathrm{KL}\{q_\psi(\mathbf{z}|\mathbf{x}) \parallel p_\phi(\mathbf{z}|\mathbf{x})\}$ cannot be small, even holding ϕ fixed and performing multiple SGD steps on the ELBO for ψ. We say that the inference network is *faithful* to the posterior when its corresponding BN structure does not mislead us about the conditional independence assumptions made by the posterior. An unfaithful inference network results in not just poor inference amortization, but also poor model learning, as $q_\psi(\mathbf{z}|\mathbf{x})$ is used to estimate the gradient of the ELBO for the model parameters, ϕ.

Thus, a poorly chosen structure for our inference network biases and increases the variance of model learning and inference amortization. Existing practice is to design this structure heuristically, typically from inverting the edges from the generative model. Unfortunately, this does not in general produce a faithful structure. This motivates our development of the NaMI algorithm in Ch 5 [Webb et al., 2018a], which inputs a BN graphical structure for the generative model and outputs a provably faithful BN structure for the true posterior. The algorithm can be trivially generalized to operate on MRFs and factor graphs. The output is not only faithful, but

minimally faithful, in the sense that the removal of even a single edge from the output renders it unfaithful to the posterior. We demonstrate experimental gains in model learning and inference amortization by using a faithful over an unfaithful inverse, and, perhaps surprisingly, using a minimally faithful over a non-minimally faithful inverse.

As an aside, let us mention the importance of the factorization for modeling. The factorization chosen expresses the assumptions made on the relationships between the variables of the model. We can use this to encode our prior knowledge of which variables directly effect which others, this being often more clearly justified than any prior assumptions on the explicit functional form of the relationships between variables.

Importance of the parametrization. Having considered the importance of the factorization, let us now consider the importance of the parametrization. The first step in producing a flexible family of representations for a target is to ensure the factorization is adequate. Nonetheless, even with an adequate factorization, if each factor is restricted in its representational power it is likely that there does not exist a $\phi \in \Phi$ for which $p_\phi(\mathbf{x}|\mathbf{y}) = p^*(\mathbf{x} \mid \mathbf{y})$ approximately holds. In the context of amortized VI, we note that having factors with simple parametric forms like the Gaussian or Bernoulli distribution in the generative model does not imply that those factors for the corresponding variables in the posterior are adequately approximated by the same parametric forms. NNs, being powerful function approximators, are crucial for constructing the flexible parametrizations required for effective amortized VI. Also, in both inference networks and modeling, NNs can be used to learn lower-dimensional distributed representations of, e.g., perceptual inputs such as images and text. Operating at a higher level of abstraction, these learnt embeddings are more effective for expressing the relationships between variables.

An ideal parametrization would have tractable scoring, that is, calculation of $\ln(p_\phi(\mathbf{x} \mid \mathbf{y}))$, as well as tractable sampling, in addition to both operations being numerically stable. Tractability in this instance is taken to mean that the running cost of these operations is $O(1)$ rather than, say, $O(N)$, and that the constant factor is not prohibitively large. Typically, $O(N)$ complexity is not considered intractable in computer science. However, in our context it is, since we are often working at the limits of our computational resources. For example, consider training a VAE model with the standard independent Gaussian inference network, which has $O(1)$ computational complexity for sampling. Suppose that training takes 12 hours. Compare this to, for example, training with a fully-connected Gaussian inference network parametrized by MADE, which has $O(N)$ computation complexity for sampling. Controlling the architectures,

the constant factor is roughly equal, so that training would take over a year if we assume the dataset is MNIST, with $N = 784$!

An ideal parametrization would also be able to represent a wide class of distributions for a given $\mathcal{X}$ and $\mathcal{Y}$, that is, it would be a *universal density estimator* (in the case of continuous variables). Let us define this concept more precisely. A universal density estimator is a family of densities $\{p_\phi(\cdot|\mathbf{y}) \mid \phi \in \Phi\}$ for which for every $\epsilon > 0$ and multivariate continuously differentiable CDF, $F(\mathbf{x}|\mathbf{y})$, there exists a member $\phi^* \in \Phi$, such that $|F(\mathbf{x}|\mathbf{y}) - F_{p_{\phi^*}}(\mathbf{x}|\mathbf{y})| < \epsilon$ for all $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$. The dimensionality of the particular ϕ^* must clearly increase as ϵ decreases.

In practice, however, we must trade off between these objectives, as we now discuss. The remainder of this section describes the technical details of several methods for constructing NN density estimators.

2.3.2 Univariate factors

Let us consider a univariate target distribution, $p^*(x)$. The simplest construction of a distribution representation for this case, consists of a univariate parametric form with deterministic parameters. For example, if $\mathcal{X} = \mathbb{R}$, one could use the normal density, $p_\phi(x) = \mathcal{N}(x \mid \mu, \sigma)$, with learnable parameters $\phi \triangleq (\mu, \sigma)$. If $\mathcal{X} = \{1, 2, \ldots, D\}$, one could use a categorical distribution, $p_\phi(x) = \prod_{d=1}^{D} \phi_d^{\mathbb{1}\{x=d\}}$, with $\sum_d \phi_d = 1$ and $0 \leq \phi_d \leq 1$.

Sampling is trivial for common parametric univariate distributions, and is typically based on either inverse transform sampling (e.g., the exponential distribution), a transformation from another distribution with simpler sampling (e.g. the Box–Muller algorithm [Box, 1958] for sampling from a bivariate normal distribution), of which inverse transform sampling is a specific case, or rejection sampling with a cleverly constructed proposal (e.g., sampling from a Gamma distribution by the method of Marsaglia and Tsang [2000]).

When there are variables, $\mathbf{y} \neq \emptyset$, to condition on, this method can be extended with the use of NNs. We first suppose a family of parametric forms for $\mathbf{x}$ and set $\phi = f_\theta(\mathbf{y})$ for a neural network f_θ. For example, for $\mathcal{X} = \mathbb{R}$, we could use $p_\phi(x \mid \mathbf{y}) = \mathcal{N}(x \mid \mu_\theta(\mathbf{y}), \sigma_\theta(\mathbf{y}))$, where the neural network outputs the parameters to the normal distribution, $f_\theta(\mathbf{y}) = (\mu_\theta(\mathbf{y}), \sigma_\theta(\mathbf{y}))$.

Sampling is no more difficult than the non-conditional case—one simply calculates the parameters to the distribution using the neural network and the value of the conditioned variables and then samples from the parametric form using the standard techniques.

As many neural networks are universal function approximators given sufficient capacity, this allows a great flexibility in how the variables to condition on, $\mathbf{y}$, influence the distribution of $\mathbf{x}$.

One should think of the networks as learning a representation of $\mathbf{y}$, and should use deep learning practice to guide their design. For instance, if $\mathbf{y}$ is an image, it seems sensible to use a standard CNN architecture for image classification in regressing $\mathbf{y}$ onto ϕ.

Nonetheless, we are still greatly constraining the representational power by using a simple form for x. In our running example of a normally distributed family, $p_\phi(x \mid \mathbf{y})$ cannot be heavy-tailed or multimodal, always having the simple bell-shape of a normal distribution. More importantly, most interesting distributions are multivariate, $N > 1$, and we thus desire representations for this case.

2.3.3 Autoregressive representations

In many deep generative models, there is a vector of latent variables $\mathbf{z}$ that are independent in the prior, $p(\mathbf{z}) = \prod_m p(z_m)$. This is the case in the VAE model. However, for the inference network to be faithful, they must typically be completely dependent. Thus for effective amortized VI, we require NN distribution representations for multivariate distributions.

Now let us consider representing a multivariate distribution over $p_\phi(\mathbf{x} \mid \mathbf{y})$ for $N > 1$. By the chain rule of probability,

$$p_\phi(\mathbf{x} \mid \mathbf{y}) = \prod_{n=1}^{N} p_{\phi_n}(x_n \mid \mathbf{x}_{\prec n}, \mathbf{y}), \tag{2.16}$$

where $\mathbf{x}_{\prec n} \triangleq \{x_1, \ldots, x_{n-1}\}$.

One approach would be to apply the approach of §2.3.2 to each univariate factor, p_n. This would require N separate neural networks, $f_{\theta_n}(\mathbf{x}_{\prec n}, \mathbf{y})$, to compute the parameters for the parametric form of each variable, x_n. It would seem advantageous if we could somehow share learning across these neural networks—from an optimization point of view, weight sharing reduces the variance in the gradients during learning.

An autoregressive neural network is one such approach. It is defined as a neural network, $f_\theta(\mathbf{x}, \mathbf{y})$, that outputs a vector $\phi = (\phi_1, \phi_2, \ldots, \phi_N)$, where the nth element of the output, ϕ_n, only depends on the previous inputs, $\mathbf{x}_{\prec n}$, and conditioned variables, $\mathbf{y}$. The output of an autoregressive network can be used as the parameters for N simple parametric factors.

We discuss several types of autoregressive neural networks in this section. They will also be used in 2.3.5 to define invertible transformations of simple random variables in order to produce richer distributions.

Masked autoencoder for density estimation. The masked autoencoder for density estimation (MADE) [Germain et al., 2015] is a densely connected feedforward network that cleverly

masks the weights of the network so that the nth output, f_n, is a function only of the previous inputs, $\mathbf{x}_{\prec n}$. Let us make this more concrete.

Suppose our factors take scalar parameters, $\phi_n \in \mathbb{R}$, and consider a dense feedforward network with a single hidden layer with H units,

$$\begin{aligned}\mathbf{h} &= f\left(U\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\end{pmatrix} + \mathbf{b}\right)\\ \phi &= V\mathbf{h} + c,\end{aligned}$$

where $U \sim H \times (N+M)$, $\mathbf{b} \sim H$, $V \sim N \times H$, $c \sim N$, and f is a non-linearity such as ReLU.

The idea of MADE is to multiply U and V elementwise by binary-valued masking matrices B and C, respectively, such that the connections are pruned in exactly the right way to make the network satisfy the autoregressive property,

$$\begin{aligned}\mathbf{h} &= f\left((B \otimes U)\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\end{pmatrix} + \mathbf{b}\right)\\ \phi &= (C \otimes V)\mathbf{h} + c.\end{aligned}$$

The construction of the masking matrices is as follows. Indices are associated with each element of the input vector, the hidden units, and the output vector. The input vector, $(\mathbf{y}, \mathbf{x})$, is associated with the "input indices," $\mathbf{m}^{(x)} \triangleq (\mathbf{0}_M, 1, 2, \ldots, N)^T$. The hidden units, $\mathbf{h}$, are associated with the "hidden indices," $\mathbf{m}^{(h)}$, by producing a sequence of H linearly spaced numbers from 1 to N and rounding to nearest integer. The outputs, ϕ, are associated with the "output indices," $\mathbf{m}^{(\phi)} \triangleq (1, 2, \ldots, N)^T$.

We set,

$$B_{ij} = \begin{cases}1, & m_j^{(x)} < m_i^{(h)}\\ 0, & \text{otherwise}\end{cases}$$
$$C_{ij} = \begin{cases}1, & m_j^{(h)} \leq m_i^{(\phi)}\\ 0, & \text{otherwise}\end{cases}.$$

Now, why does this construction satisfy the autoregressive property? It is helpful to imagine the active connections tracing a path backwards from a given output, ϕ_k, to the inputs. The output ϕ_k is connected to all hidden units with index less than or equal to k. The hidden units with index k are connected to inputs with index $k-1$ or lower, the hidden units with index $k-1$ are connected to inputs with index $k-2$ or lower, and so on. So in total, ϕ_k is connected to inputs with index $k-1$ or lower. But by construction, these are exactly $(\mathbf{x}_{\prec k}, \mathbf{y})$, and so the network is autoregressive.

This simple presentation is easily extended to the more general case. When there are multiple parameters for each factor, e.g. $\phi_k \in \mathbb{R}^2$, we can make the output of the autoregressive network

a longer vector that is reshaped, modifying the masking matrix appropriately. Restrictions on the valid range of ϕ_k can be enforced by applying an output activation function,

$$\phi = g((C \otimes V)\mathbf{h} + c).$$

The feedforward network can be extended to have multiple hidden layers by introducing additional masking matrices with the same construction, and it has been found advantageous to introduce skip connections between the inputs and outputs [Paige and Wood, 2016], requiring an additional masking matrix.

Sampling $\mathbf{x}$ now requires N passes of the network as follows. Note that ϕ_1 only depends on $\mathbf{y}$. Thus a single pass through the network with any value for the $\mathbf{x}$ reveals ϕ_1. Using this we sample $x_1 \sim p_{\phi_1}(\cdot)$. Next, another pass through the network with our sampled value for x_1 and any value for $x_2, \ldots, x_N$ reveals ϕ_2. Using this we can sample x_2 and so on until the entire $\mathbf{x}$ has been sampled.

One trick to make this procedure easily implementable is to fix the random noise vector used to sample $\mathbf{x}$. For instance, if $\mathbf{x} = \mu + \sigma\mathbf{z}$, where $\phi = (\mu, \sigma)$, and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}_N, \mathbf{1}_N)$, we can use the same $\mathbf{z}$ at each iteration of the sampling procedure, and sample the entire $\mathbf{x}$ at each step. The sample, $\mathbf{x}$, should not change beyond N iterations, a simple sanity check that the algorithm has been implemented correctly.

The MADE model is capable of representing fully-connected multivariate distributions where each variable's conditional distribution has a simple parametric form, sharing parameters between the factors corresponding to each variable. However, it is limited by the simple form for each factor and in that sampling is not a tractable operation, having complexity $O(N)$. On the other hand, the density can be calculated with a single forward pass of the NN, resulting in $O(1)$ scoring.

Recurrent neural networks There are scenarios where we would like to model distributions with varying dimensionality. For instance, one possible faithful inference network structure for a hidden Markov model (HMM) structured generative model contains factors, $q_\psi(z_t \mid z_{t-1}, x_{\succ t})$, where $x_{\succ t} = \{x_{t+1}, x_{t+2}, \ldots, x_T\}$ (see Krishnan et al. [2017]). We would like to share the weights of the inference network over time steps, t, and hence require a function that is function of the variably lengthed $\{x_{\succ t}\}$. Recurrent neural networks (RNNs) [Rumelhart et al., 1986; Graves, 2012] are one such choice to design these types of inference networks.

A RNN is a NN that is designed to operate on a (possibly) variable-length sequence. In our case, the sequence will be the elements of the input, $\{x_1, x_2, \ldots, x_N\}$. The RNN defines

a hidden state, $\mathbf{h}^{(n)} \sim H$ for each position in the sequence, containing a representation of the sequence seen thus far.

The hidden state is calculated as the output of a transition function,

$$\mathbf{h}^{(n)} = f_\theta(\mathbf{h}^{(n-1)}, x_n, \mathbf{y}), \;\; n = 1, \ldots, N-1, \tag{2.17}$$

that learns to update the current representation based on the one of the previous time-step, the input at current time-step, and, in our case, the variables $\mathbf{y}$ that the distribution conditions upon.

The transition function is typically a shallow dense feedforward network, such as,

$$f_\theta(\mathbf{h}^{(n-1)}, x_n, \mathbf{y}) \triangleq \tanh\left(U\mathbf{h}^{(n-1)} + V\begin{pmatrix} x_n \\ \mathbf{y} \end{pmatrix} + \mathbf{b}\right).$$

where $\theta = \{U, V, \mathbf{b}\}$ are learnable parameters.

The hidden state, $\mathbf{h}^{(n)}$ is directly a function of x_n, and indirectly a function of $\mathbf{x}_{\prec n}$ via $\mathbf{h}^{(n-1)}$. Therefore, we can define the parameters,

$$\phi_n \triangleq g_\theta(W\mathbf{h}^{(n-1)} + \mathbf{c}), \;\; n = 1, \ldots, N,$$

and use ϕ_n to parametrize the factor p_n of (2.16), safe in the knowledge that ϕ_n depends only on $(\mathbf{x}_{\prec n}, \mathbf{y})$.

The initial hidden state, $\mathbf{h}^{(0)}$ is typically set to zero or made a learnable parameter. In our case, however, we require it to be a function of $\mathbf{y}$ and none of the input variables. For this we could define an "initialization function,"

$$\mathbf{h}^{(0)} \triangleq \tanh\left(V'\mathbf{y} + \mathbf{b}\right).$$

We thus see that the RNN accomplishes representation learning of variable-length sequences by sharing weights across time-steps, and can be interpreted as an autoregressive NN that can be used to represent joint distributions.

The simple RNN transition function of (2.17) suffers from the "vanishing gradient problem" [Pascanu et al., 2013] for longer sequences, where the gradient signal diminishes to a degree that learning long-term dependencies is infeasible. Gating RNNs such as the LSTM [Hochreiter and Schmidhuber, 1997; Gers et al., 2000; Gers and Schmidhuber, 2000; Graves and Schmidhuber, 2005] and GRU [Cho et al., 2014; Chung et al., 2014, 2015] effectively solve this issue, and are widely used in practice.

RNNs have been used to parametrize inference networks for sequential models where length of the sequence is not fixed [Eslami et al., 2016; Krishnan et al., 2017]. This is their main

advantage, working on variable-length sequences, as both sampling and scoring are operations with undesirable $O(N)$ complexity.

In recent developments, "recurrent" NNs making use of convolutional operators have been developed [Bradbury et al., 2016; van den Oord et al., 2016a; Kalchbrenner et al., 2016; van den Oord et al., 2016b] and shown to have comparable performance to standard RNNs while offering improved parallelization (for scoring but not sampling).

2.3.4 Mixture models

Many modeling situations naturally call for representing multi-modal distributions. For instance, a distribution over word embeddings would likely need to be multi-modal to represent our uncertainty over the particular meaning of a word being used given its context. Mixture models [Bishop, 1994] are a simple means of constructing multi-modal distributions from simpler ones.

Suppose a family of distributions, $\mathcal{P} = \{p_\phi(\mathbf{x} \mid \mathbf{y}) \mid \phi \in \Phi\}$. A mixture model with K components is a convex combination of distributions,

$$p_\phi(\mathbf{x} \mid \mathbf{y}) = \sum_{k=1}^{K} \alpha_k p_{\phi_k}(\mathbf{x} \mid \mathbf{y})$$

where $p_{\phi_k} \in \mathcal{P}$ and the mixture weights, $\sum_{k=1}^{K} \alpha_k = 1$. This is clearly still a valid distribution—it corresponds to the marginal of $p_\phi(\mathbf{x}, k \mid \mathbf{y}) \triangleq p_{\phi_k}(\mathbf{x} \mid \mathbf{y})p(k)$ over k.

To sample from a mixture model, we first sample a component index

$$k' \sim \text{Categorical}(\alpha_1, \alpha_2, \ldots, \alpha_K)$$

with probabilities given by the mixture weights, then sample from the corresponding component, p_{ϕ_k}.

Whether sampling and scoring is tractable for the mixture model distribution in whole depends on the tractability for each mixture component. In particular, if both are tractable for all components then they are tractable for the the mixture model. This can be used to construct more complex distribution representations from any of the tools outlaid in this chapter, preserving their properties of tractability. For instance, Paige and Wood [2016] construct a multi-modal proposal for importance sampling as a mixture of MADE networks.

A limitation of mixture models is that we must fix the maximum nodes in advance, which may result in components being unused if K is too large for the data, or an inadequate fit if K is too small. It would be preferable if our representation adapted its number of modes during learning. Moreover, for a fixed K, the flexibility of a mixture model depends on the flexibility of each component, p_{ϕ_k}, which means the concept has not solved the central challenge of this chapter.

2.3.5 Normalizing flows

So far we have described representations that use a simple parametric form for each factor $p(x_n|x_{\prec n}, \mathbf{y})$, or a mixture of these. An arbitrary continuous distribution, however, can possess a much more complex functional form, and we would indeed expect it to—even though the model may be expressed with simple parametric forms, this does not entail that the true posterior under a given factorization comprises the same parametric forms.

Normalizing flows provide one avenue to produce more complex representations. Often it is convenient to represent the sampling process of some distribution as a transformation of a simpler one [Tabak and Turner, 2013; Rezende and Mohamed, 2015]. For instance, if $U \sim \text{Unif}(0, 1)$, then $W = \lambda(-\ln(U))^{1/k} \sim \text{Weibull}(k, \lambda)$. To sample from W we simply sample u from U and evaluate W with $U = u$. Normalizing flows extend this concept by making the transformation a learnable function, g_θ, typically over vector-valued random variables. Given a simple source of randomness such as a draw $Z \sim N(\mathbf{0}, I)$ from the standard multivariate normal distribution, one samples from the more complex distribution $\mathbf{x} \sim p(\cdot \mid \mathbf{y})$ as $\mathbf{x} = g_\theta(\mathbf{z}; \mathbf{y})$.

How does the density $p_X(\mathbf{x} \mid \mathbf{y})$ relate to the density $p_Z(\mathbf{z})$? When g is a bijective, differentiable function in $\mathbf{z}$, we can apply a multivariate change-of-variables,

$$\begin{aligned} p_X(\mathbf{x} \mid \mathbf{y}) &= \left|\det\left(\frac{d\mathbf{z}}{d\mathbf{x}}\right)\right| p_Z(\mathbf{z}) \\ &= \left|\det\left(\frac{d\mathbf{x}}{d\mathbf{z}}\right)\right|^{-1} p_Z(\mathbf{z}), \end{aligned}$$

where $d\mathbf{z}/d\mathbf{x}$ denotes the Jacobian J with $J_{ij} \triangleq \partial g_i/\partial x_j$ (and similarly for $d\mathbf{x}/d\mathbf{z}$), and the last line follows by the inverse function theorem. In log-space this is,

$$\ln(p_X(\mathbf{x} \mid \mathbf{y})) = \ln(p_Z(\mathbf{z})) - \ln\left(\left|\det\left(\frac{d\mathbf{x}}{d\mathbf{z}}\right)\right|\right).$$

Provided we can easily calculate the determinant of the Jacobian, the transformed variable is easy to score. Research in normalizing flows aims to construct powerful bijective and differentiable transforms for which this holds.

There are several strategies used for constructing bijections with tractable $\det(J)$. Firstly, it is possible to calculate $\det(J)$ without explicitly using Laplace's formula or performing an expensive matrix decomposition in some restricted cases [Rezende and Mohamed, 2015; van den Berg et al., 2018]. Another strategy is to construct volume-preserving bijections, for which $\det(J) = \pm 1$ [Dinh et al., 2014; Tomczak and Welling, 2016]. Yet another strategy is to construct the transformation so that J is a triangular matrix and thus $\det(J)$ the product of the diagonal

[Kingma et al., 2016; Papamakarios et al., 2017; Huang et al., 2018]. In recent work, the issue of calculating $\det(J)$ is sidestepped by making the transformation and its inverse the (numerical) solution to an ordinary differential equation [Chen et al., 2018].

Layers of normalizing flows can be cascaded to produce more powerful transformations. Suppose we have L bijective continuous transforms applied in sequence,

$$\mathbf{z}_L = g_{\theta_L} \circ g_{\theta_{L-1}} \circ \cdots \circ g_{\theta_1}(\mathbf{z}_0)$$

where $\circ$ denotes function composition, $\mathbf{z}_0$ is the base noise, and $\mathbf{z}_L = \mathbf{x}$ the output. By applying the change-of-variables recursively,

$$\ln(p_X(\mathbf{x} \mid \mathbf{y})) = \ln(p_Z(\mathbf{z}_0)) - \sum_{l=1}^{L} \ln\left(\left|\det\left(\frac{d\mathbf{z}_l}{d\mathbf{z}_{l-1}}\right)\right|\right).$$

This comes at the expense of more challenging optimization due to a deeper architecture and more parameters. Provided the reparametrization trick (explained in Section 2.2.4, "Pathwise estimators") is applicable to the base distribution, it is easy to produce reparametrization gradients for a normalizing flow using the chain-rule (which is taken care of by automatic differentiation). Note that we will use the term "normalizing flow" to denote both the distribution resulting from applying a single or cascade of bijective transformations to samples from a base distribution, as well as the transformations themselves (which will be clear from the context).

Normalizing flows with L discrete layers are referred to as finite flows. As the number of layers approaches infinity, we have an infinitesimal, or continuous, flow, for which the flow dynamics is described by an ordinary differential equation,

$$\frac{\partial}{\partial t}\mathbf{z}_t = g_\theta(\mathbf{z}_t).$$

When a normalizing flow is used to parametrize an inference network for amortized variational inference, it is not a requirement of most schemes that scoring of arbitrary samples, which requires inverting the flow, is a well-defined operation. For instance, the ELBO (Ch 2.2) minimizes $\mathrm{KL}\{q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\}$ up to a constant. This only requires tractable sampling and scoring these samples, that is, calculate $\ln q(\mathbf{z} \mid \mathbf{x})$ for $\mathbf{z} \sim q(\cdot \mid \mathbf{x})$. Even if the inverse of the flow transformation is undefined, numerically unstable, or computationally expensive, we can cache the inputs $\mathbf{z}_0$ that gave rise to each $\mathbf{x} = \mathbf{z}_L$, and use this to calculate $p_Z(\mathbf{z})$. It is even possible in many cases to calculate and cache $\ln(|\det(\frac{\partial z}{\partial z'})|)$ during sampling.

On the other hand, if the normalizing flow is used for density estimation of $p(\cdot)$ by $q_\theta(\cdot)$ and the objective is, e.g., the negative log-likelihood, or $\mathrm{KL}\{p(\mathbf{x})||q(\mathbf{x})\}$ up to a constant, then we require to be able to draw samples from the target $p(\cdot)$, and that scoring of arbitrary samples

be tractable on the distribution estimator $q_\theta(\cdot)$. Moreover, sampling from $q_\theta(\cdot)$ is not required for learning (although it may be required for a downstream task like using q_θ as a proposal for importance sampling [Paige and Wood, 2016]). Thus we see that the direction of the KL-divergence in the objective used to match distributions effects the requirements we place on the forward and inverse operations of the normalizing flow used.

There are two means of representing a conditional distribution, $p(\mathbf{x} \mid \mathbf{y})$, with normalizing flow. Firstly, we can simply make each layer of flow's transformation, $g_\theta^{(l)}$ take $\mathbf{y}$ as an input. An alternative strategy is to make the first layer take $\mathbf{y}$ as an input, modify the transformations so that $g_\theta^{(l)} = (\mathbf{z}_l, \mathbf{h}_l)$, each layer outputs additionally a "hidden state," $\mathbf{h}_l$, and feed this into subsequent layers after the 0th one.

2.3.6 Discussion

In this section, we have drawn a distinction between a distribution's factorization and parametrization, and have focused on the latter, describing how NNs can be used to construct representations, both for inference and modeling purposes. In the following chapter, we focus on the factorization aspect of distribution representation, and describe a novel algorithm for constructing optimal factorizations for inference networks with respect to a generative model. When combined with the NN distribution representations presented in this section, such inference networks can be used by amortized VI to perform effective scalable and generic Bayesian inference.

3

Faithful Inversion of Generative Models for Effective Amortized Inference

Faithful Inversion of Generative Models for Effective Amortized Inference

Stefan Webb[*] University of Oxford
Adam Goliński University of Oxford
Robert Zinkov UBC
N. Siddharth University of Oxford
Tom Rainforth University of Oxford
Yee Whye Teh University of Oxford
Frank Wood UBC

Abstract

Inference amortization methods share information across multiple posterior-inference problems, allowing each to be carried out more efficiently. Generally, they require the inversion of the dependency structure in the generative model, as the modeller must learn a mapping from observations to distributions approximating the posterior. Previous approaches have involved inverting the dependency structure in a heuristic way that fails to capture these dependencies correctly, thereby limiting the achievable accuracy of the resulting approximations. We introduce an algorithm for faithfully, and minimally, inverting the graphical model structure of any generative model. Such inverses have two crucial properties: a) they do not encode any independence assertions that are absent from the model and b) they are local maxima for the number of true independencies encoded. We prove the correctness of our approach and empirically show that the resulting minimally faithful inverses lead to better inference amortization than existing heuristic approaches.

1 Introduction

Evidence from human cognition suggests that the brain reuses the results of past inferences to speed up subsequent related queries (Gershman & Goodman, 2014). In the context of Bayesian statistics, it is reasonable to expect that, given a generative model, $p(\mathbf{x}, \mathbf{z})$, over data $\mathbf{x}$ and latent variables $\mathbf{z}$, inference on $p(\mathbf{z} \mid \mathbf{x}_1)$ is informative about inference on $p(\mathbf{z} \mid \mathbf{x}_2)$ for two related inputs, $\mathbf{x}_1$ and $\mathbf{x}_2$. Several algorithms (Kingma & Welling, 2014; Rezende et al., 2014; Stuhlmüller et al., 2013; Paige & Wood, 2016; Le et al., 2017, 2018; Maddison et al., 2017a; Naesseth et al., 2018) have been developed with this insight to perform *amortized inference* by learning an inference artefact $q(\mathbf{z} \mid \mathbf{x})$, which takes as input the values of the observed variables, and—typically with the use of neural network architectures—return a distribution over the latent variables approximating the posterior. These inference artefacts are known variously as inference networks, recognition models, probabilistic encoders, and guide programs; we will adopt the term *inference networks* throughout.

Along with conventional fixed-model settings (Stuhlmüller et al., 2013; Le et al., 2017; Ritchie et al., 2016; Paige & Wood, 2016), a common application of inference amortization is in the training of variational auto-encoders (VAEs) (Kingma & Welling, 2014), for which the inference network is simultaneously learned alongside a generative model. It is well documented that deficiencies in the expressiveness or training of the inference network can also have a knock-on effect on the learned generative model in such contexts (Burda et al., 2016; Cremer et al., 2017, 2018; Rainforth et al., 2018), meaning that poorly chosen coarse-grained structures can be particularly damaging.

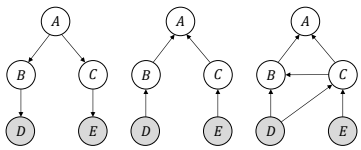
Implicit in the factorization of the generative model and inference network in both fixed and learned model settings are probabilistic graphical models, commonly Bayesian networks (BNs), encoding dependency structures. We refer to these as the *coarse-grain* structure, in opposition to the *fine-grain* structure of the neural networks that form each inference (and generative) network factor. In this sense, amortized inference can be framed as the problem of graphical model inversion—how to invert the graphical model of the generative model to give a graphical model approximating the posterior.

[*]Correspondence to info@stefanwebb.me

Many models from the deep generative modeling literature can be represented as BNs (Krishnan et al., 2017; Gan et al., 2015; Neal, 1990; Kingma & Welling, 2014; Germain et al., 2015; van den Oord et al., 2016b,a), and fall within this framework.

In this paper, we borrow ideas from the probabilistic graphical models literature, to address the previously open problem of how best to automate the design of the coarse-grain structure of the inference network (Ritchie et al., 2016). Typically, the inverse graphical model is formed heuristically. At the simplest level, some methods just invert the edges in the BN for the generative model, removing edges between observed variables (Kingma & Welling, 2014; Gan et al., 2015; Ranganath et al., 2015). In a more principled, but still heuristic, approach, Stuhlmüller et al. (2013); Paige & Wood (2016) construct the inference network by inverting the edges and additionally connecting the parents of children in the original graph (both of which are a subset of a variable's Markov blanket; see Appendix C).

In general, these heuristic methods introduce conditional independencies into the inference network that are not present in the original distribution. Consequently, they cannot represent the true posterior even in the limit of infinite neural network capacities. Take the simple generative model with branching structure of Figure 1a. The inference network formed by Stuhlmüller's method inverts the edges of the model as in Figure 1b. However, an inference network that is able to represent the true posterior requires extra edges between the branches, as in Figure 1c.



(a) (b) (c)

Figure 1: (a) Generative model BN; (b) Inverse BN by Stuhlmüller's Algorithm; (c) *Faithful* inverse BN by our algorithm.

Another approach, taken by Le et al. (2017), is to use a fully connected BN for the inverse graphical model, such that every random choice made by the inference network depends on every previous one. Though such a model is expressive enough to correctly represent the data given infinite capacity and training time, it ignores substantial available information from the forward model, inevitably leading to reduced performance for finite training budgets and/or network capacities.

In this paper, we develop a tractable framework to remedy these deficiencies: the ***Na***tural ***M***inimal ***I***-map generator (NaMI). Given an arbitrary BN structure, NaMI can be used to construct an inverse BN structure that is provably both *faithful* and *minimal*. It is faithful in that it contains sufficient edges to avoid encoding conditional independencies absent from the model. It is minimal in that it does not contain any unnecessary edges; i.e., removing any edge would result in an unfaithful structure.

NaMI chiefly draws upon variable elimination (Koller & Friedman, 2009, Ch 9,10), a well-known algorithm from the graphical model literature for performing exact inference on discrete factor graphs. The key idea in the operation of NaMI is to simulate variable elimination steps as a tool for successively determining a minimal, faithful, and natural inverse structure, which can then be used to parametrize an inference network. NaMI further draws on ideas such as the min-fill heuristic (Fishelson & Geiger, 2004), to choose the ordering in which variable elimination is simulated, which in turn influences the structure of the generated inverse.

To summarize, our key contributions are:

i) framing generative model learning through amortized variational inference as a graphical model inversion problem, and
ii) using the simulation of exact inference algorithms to construct an algorithm for generating provably minimally faithful inverses.

Our work thus highlights the importance of constructing both minimal and faithful inverses, while providing the first approach to produce inverses satisfying these properties.

2 Method

Our algorithm builds upon the tools of *probabilistic graphical models*— a summary for unfamiliar readers is given in Appendix A.

2.1 General idea

Amortized inference algorithms make use of inference networks that approximate the posterior. To be able to represent the posterior accurately, the distribution of the inference network should not encode independence assertions that are absent from the generative model. An inference network that did

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

encode additional independencies could not represent the true posterior, even in the non-parametric limit, with neural network factors whose capacity approaches infinity.

Let us define a *stochastic inverse* for a generative model $p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$ that factors according to a BN structure $\mathcal{G}$ to be a factorization of $q(\mathbf{z}|\mathbf{x})q(\mathbf{x})$ over $\mathcal{H}$ (Stuhlmüller et al., 2013; Paige & Wood, 2016). The $q(\mathbf{z}|\mathbf{x})$ part of the stochastic inverse will define the factorization, or rather, coarse-grain structure, of the inference network. Recall from §1 that this involved two characteristics. We first require $\mathcal{H}$ to be an *I-map* for $\mathcal{G}$:

Definition 1. *Let $\mathcal{G}$ and $\mathcal{H}$ be two BN structures. Denote the set of all conditional independence assertions made by a graph, $\mathcal{K}$, as $\mathcal{I}(\mathcal{K})$. We say $\mathcal{H}$ is an* I-map *for $\mathcal{G}$ if $\mathcal{I}(\mathcal{H}) \subseteq \mathcal{I}(\mathcal{G})$.*

To be an I-map for $\mathcal{G}$, $\mathcal{H}$ may not encode all the independencies that $\mathcal{G}$ does, but it must not mislead us by encoding independencies not present in $\mathcal{G}$. We term such inverses as being *faithful*. While the aforementioned heuristic methods *do not* in general produce faithful inverses, using either a fully-connected inverse, or our method, does.

Second, since a fully-connected graph encodes no conditional independencies and is therefore suboptimal, we require in addition that $\mathcal{H}$ be a *minimal I-map* for $\mathcal{G}$:

Definition 2. *A graph $\mathcal{K}$ is a* minimal I-map *for a set of independencies $\mathcal{I}$ if it is an I-map for $\mathcal{I}$ and if removal of even a single edge from $\mathcal{K}$ renders it not an I-map.*

We call such inverses *minimally faithful*, which roughly means that the inverse is a local optimum in the number of true independence assertions it encodes.

There will be many minimally faithful inverses for $\mathcal{G}$, each with a varying number of edges. Our algorithm produces a *natural inverse* in the sense that it either inverts the order of the random choices from that of the generative model (when it is run in the topological mode), or it preserves the ordering of the random choices (when it is run in reverse topological mode):

Definition 3. *A stochastic inverse $\mathcal{H}$ for $\mathcal{G}$ over variables $\mathcal{X}$ is a* natural inverse *if either, for all $X \in \mathcal{X}$ there are no edges in $\mathcal{H}$ from X to its descendants in $\mathcal{G}$, or, for all $X \in \mathcal{X}$ there are no edges in $\mathcal{H}$ from X to its ancestors in $\mathcal{G}$.*

Essentially, a natural inverse is one for which if we were to perform ancestral sampling, the variables would be sampled in either a topological or reverse-topological ordering, relative to the original model. Consider the inverse networks of $\mathcal{G}$ shown in Figure 2. $\mathcal{H}_1$ is not a natural inverse of $\mathcal{G}$, since there is both an edge $A \to C$ from a parent to a child, and an edge $C \to B$ from a child to a parent, relative to $\mathcal{G}$. However, $\mathcal{H}_2$ and $\mathcal{H}_3$ are natural, as they correspond respectively to the reverse-topological and topological orderings C, B, A and B, A, C.

Most heuristic methods, including those of (Stuhlmüller et al., 2013; Paige & Wood, 2016), produce (unfaithful) natural inverses that invert the order of the random choices, giving a reverse-topological ordering.

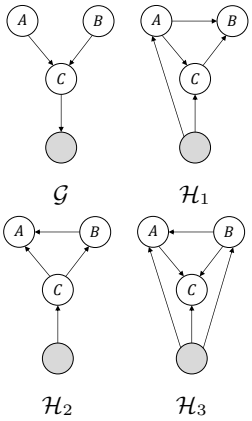


Figure 2: Illustrating definition of naturalness.

2.2 Obtaining a natural minimally faithful inverse

We now present NaMI's graph inversion procedure that given an arbitrary BN structure, $\mathcal{G}$, produces a natural minimal I-map, $\mathcal{H}$. We illustrate the procedure step-by-step on the example given in Figure 3. Here H and J are observed, as indicated by the shaded nodes. Thus, our latent variables are $\mathbf{Z} = \{D, I, G, S, L\}$, our data is $\mathbf{X} = \{H, J\}$, and a factorization for $p(\mathbf{z} \mid \mathbf{x})$ is desired.

The NaMI graph-inversion algorithm is traced in Table 1. Each step incrementally constructs two graphs: an *induced graph* $\mathcal{J}$ and a *stochastic inverse* $\mathcal{H}$. The induced graph is an undirected graph whose maximally connected subgraphs, or *cliques*, correspond to the scopes of the intermediate factors produced by simulating variable elimination. The stochastic inverse represents our eventual target which encodes the inverse dependency structure. It is constructed using information from the partially-constructed induced graph. Specifically, NaMI goes through the following steps for this example.

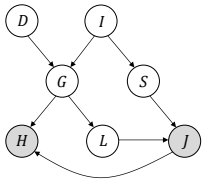
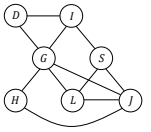
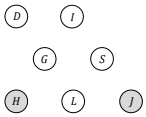
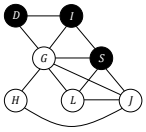
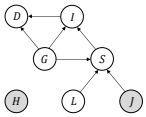
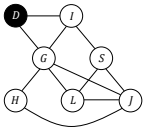
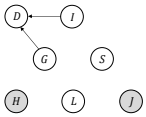
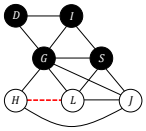
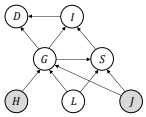
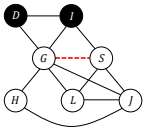
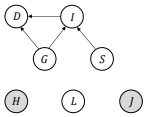
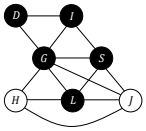
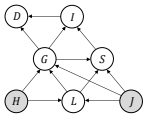


Figure 3: Example BN

STEP 0: The partial induced graph and stochastic inverse are initialized. The initial induced graph is formed by taking the directed graph for the forward model, $\mathcal{G}$, removing the directionality of the

Table 1: Tracing the NaMI algorithm on example from Figure 3. S is the set of "frontier" variables that are considered for elimination, $v \in S$ the variable eliminated at each step chosen by the greedy min-fill heuristic, $\mathcal{J}$ the partially constructed induced graph *after* each step with black nodes indicating a eliminated variables, and $\mathcal{H}$ the partially constructed stochastic inverse.

STEP	S	v	$\mathcal{J}$	$\mathcal{H}$	STEP	S	v	$\mathcal{J}$	$\mathcal{H}$
0	$\emptyset$	$\emptyset$			3	G, S	S		
1	D, I	D			4	G	G		
2	I	I			5	L	L		

edges, and adding additional edges between variables that share a child in $\mathcal{G}$—in this example, edges $D - I$, $S - L$ and $G - J$. This process is known as *moralization*. The stochastic inverse begins as disconnected variables, and edges are added to it at each step.

STEP 1: The frontier set of variables to consider for elimination, S, is initialized to the latent variables having no latent parents in $\mathcal{G}$, that is, D, I. To choose which variable to eliminate first, we apply the greedy min-fill heuristic, which is to choose the (possibly non-unique) variable that adds the fewest edges to the induced graph $\mathcal{J}$ in order to produce as compact an inverse as possible under the topological ordering. Specifically, noting that the cliques of $\mathcal{J}$ correspond to the scopes of intermediate factors during variable elimination, we want to avoid producing intermediate factors which would require us to add additional edges to $\mathcal{J}$, as doing so will in turn induce additional edges in $\mathcal{H}$ at future steps. For this example, if we were to eliminate D, that would produce an intermediate factor, $\psi_D(D, I, G)$, while if we were to eliminate I, that would produce an intermediate factor, $\psi_I(I, D, G, S)$. Choosing to eliminate would I thus requires adding an edge G–S to the induced graph, as there is no clique I, D, G, S in the current state of $\mathcal{J}$. Conversely, eliminating D does not require adding extra edges to $\mathcal{J}$ and so we choose to eliminate D.

The elimination of D is simulated by marking its node in $\mathcal{J}$. The parents of D in the inverse $\mathcal{H}$ are set to be its nonmarked neighbours in $\mathcal{J}$, that is, I and G. D is then removed from the frontier, and any non-observed children in $\mathcal{G}$ of D whose parents have all been marked added to it—in this case, there are none as the only child of D, G, still has an unmarked parent I.

STEP 2: Variable I is the sole member of the frontier and is chosen for elimination. The elimination of I is simulated by marking its node in $\mathcal{J}$ *and* adding the additional edge G–S. This is required because elimination of I requires the addition of a factor, $\psi_I(I, G, S)$, that is not currently present in $\mathcal{J}$. The parents of I in the inverse $\mathcal{H}$ are set to be its nonmarked neighbours in $\mathcal{J}$, G and S. I is then removed from the frontier. Now, G and S are children of I, and both their parents D and I have been marked. Therefore, they are added to the frontier.

STEP 3-5: The process is continued until the end of the fifth step when all the latent variables, D, I, S, G, L, have been eliminated and the frontier is empty. At this point, $\mathcal{H}$ represents a factorization $p(\mathbf{z} \mid \mathbf{x})$, and we stop here as only a factorization for the posterior is required for amortized inference. Note, however, that it is possible to continue simulating steps of variable elimination on the observed variables to complete the factorization as $p(\mathbf{z} \mid \mathbf{x})p(\mathbf{x})$.

An important point to note is that NaMI's graph inversion can be run in one of two modes. The "topological mode," which we previously implicitly considered, simulates variable elimination in a topological ordering, producing an inverse that reverses the order of the random choices from the generative model. Conversely, NaMI's graph inversion can also be run in "reverse topological

mode," which simulates variable elimination in a reverse topological ordering, producing an inverse that preserves the order of random choices in the generative model. We will refer to these approaches as *forward-NaMI* and *reverse-NaMI* respectively in the rest of the paper. The rationale for these two modes is that, though they both produce minimally faithful inverses, one may be substantially more compact than the other, remembering that minimality only ensures a local optimum. For an arbitrary graph, it cannot be said in advance which ordering will produce the more compact inverse. However, as the cost of running the inversion algorithm is low, it is generally feasible to try and pick the one producing a better solution.

Algorithm 1 NaMI Graph Inversion

1: **Input:** BN structure $\mathcal{G}$, latent variables $\mathcal{Z}$, TOPMODE?
2: $\mathcal{J} \leftarrow$ MORALIZE($\mathcal{G}$)
3: Set all vertices of $\mathcal{J}$ to be unmarked
4: $\mathcal{H} \leftarrow \{$VARIABLES($\mathcal{G}$), $\emptyset\}$, i.e. unconnected graph
5: UPSTREAM $\leftarrow$ "parent" if TOPMODE? else "child"
6: DOWNSTREAM $\leftarrow$ "child" if TOPMODE? else "parent"
7: $S \leftarrow$ all latent variables without UPSTREAM latents in $\mathcal{G}$
8: **while** $S \neq \emptyset$ **do**
9: Select $v \in S$ according to min-fill criterion
10: Add edges in $\mathcal{J}$ between unmarked neighbours of v
11: Make unmarked neighbours of $v \in \mathcal{J}$, v's parents in $\mathcal{H}$
12: Mark v and remove from S
13: **for** unmarked latents DOWNSTREAM u of v in $\mathcal{G}$ **do**
14: Add u to S if all its UPSTREAM latents in $\mathcal{G}$ are marked
15: **end for**
16: **end while**
17: **return** $\mathcal{H}$

The general NaMI graph-reversal procedure is given in Algorithm 1. It is further backed up by the following formal demonstration of correctness, the proof for which is given in Appendix F.

Theorem 1. *The Natural Minimal I-Map Generator of Algorithm 1 produces inverse factorizations that are natural and minimally faithful.*

We further note that NaMI's graph reversal has a running time of order $O(nc)$ where n is the number of latent variables in the graph and $c << n$ is the size of the largest clique in the induced graph. We consequently see that it can be run cheaply for practical problems: the computational cost of generating the inverse is generally dominated by that of training the resulting inference network itself. See Appendix F for more details.

2.3 Using the faithful inverse

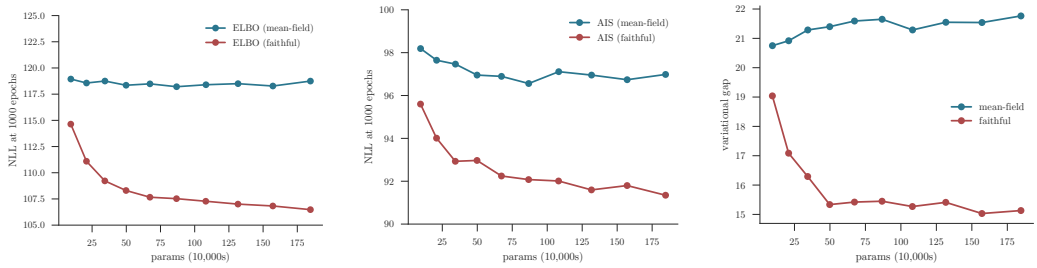
Once we have obtained the faithful inverse structure $\mathcal{H}$, the next step is to use it to learn an inference network, $q_\psi(\mathbf{z} \mid \mathbf{x})$. For this, we use the factorization given by $\mathcal{H}$. Let τ denote the reverse of the order in which variables were selected for elimination by Line 9 in Algorithm 1, such that τ is a permutation of $1, \ldots, n$ and $\tau(n)$ is the first variable eliminated. $\mathcal{H}$ encodes the factorization

$$q_\psi(\mathbf{z} \mid \mathbf{x}) = \prod_{i=1}^{n} q_i(z_{\tau(i)} \mid \mathrm{Pa}_{\mathcal{H}}(z_{\tau(i)})) \tag{1}$$

where $\mathrm{Pa}_{\mathcal{H}}(z_{\tau(i)}) \subseteq \{\mathbf{x}, z_{\tau(1)}, \ldots, z_{\tau(i-1)}\}$ indicates the parents of $z_{\tau(i)}$ in $\mathcal{H}$. For each factor q_i, we must decide both the class of distributions for $z_{\tau(i)} \mid \mathrm{Pa}_{\mathcal{H}}(z_{\tau(i)})$, and how the parameters for that class are calculated. Once learned, we can both sample from, and evaluate the density of, the inference network for a given dataset by considering each factor in turn.

The most natural choice for the class of distributions for each factor is to use the same distribution family as the corresponding variable in the generative model, such that the supports of these distributions match. For instance, continuing the example from Figure 3, if $D \sim N(0, 1)$ in the generative model, then a normal distribution would also be used for $D \mid I, G$ in the inference network. To establish the mapping from data to the parameters to this distribution, we train neural networks using stochastic gradient ascent methods. For instance, we could set $D \mid \{I = i, G = g\} \sim N(\mu_\varphi(i, g), \sigma_\varphi(i, g))$, where μ_φ and σ_φ are two densely connected feedforward networks, with learnable parameters φ. In general, it will be important to choose architectures which well match the problem at hand. For example, when perceptual inputs such as images and language are present in the conditioning variables, it is advantageous to first embed them to a lower-dimensional representation using, for example, convolutional neural networks.

Matching the distribution families in the inference network and generative model, whilst a simple and often adequate approximation, can be suboptimal. For example, suppose that for a normally distributed variable in the generative model, the true conditional distribution in the posterior for that variable is multimodal. In this case, using a (single mode) normal factor in the inference network would not suffice. One could straightforwardly instead use, for example, either a mixture of Gaussians, or, normalizing flows (Rezende & Mohamed, 2015; Kingma et al., 2016), to parametrize each inference network factor in order to improve expressivity, at the cost of additional implementational



(a) (b) (c)

Figure 4: Results for the relaxed Bernoulli VAE with 30 latent units, compared after 1000 epochs of learning the: (a) negative ELBO, and (b) negative AIS estimates, varying inference network factorizations and capacities (total number of parameters); (c) An estimate of the variational gap, that is, the difference between marginal log-likelihood and the ELBO.

complexity. In particular, if one were to use a provably universal density estimator to parameterize each inference network factor, such as that introduced in Huang et al. (2018), the resulting NaMI inverse would constitute a universal density estimator of the true posterior.

After the inference network has been parametrized, it can be trained in number of different ways, depending on the final use case of the network. For example, in the context of amortized stochastic variational inference (SVI) methods such as VAEs (Kingma & Welling, 2014; Rezende et al., 2014), the model $p_\theta(\mathbf{x}, \mathbf{z})$ is learned along with the inference network $q_\psi(\mathbf{z} \mid \mathbf{x})$ by optimizing a lower bound on the marginal loglikelihood of the data, $\mathcal{L}_{ELBO} = \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}, \mathbf{z}) - \ln q_\psi(\mathbf{z} \mid \mathbf{x})\right]$. Stochastic gradient ascent can then be used to optimize $\mathcal{L}_{ELBO}$ in the same way a standard VAE, simulating from $q_\psi(z|x)$ by considering each factor in turn and using reparameterization (Kingma & Welling, 2014) when the individual factors permit doing so.

A distinct training approach is provided when the model $p(\mathbf{x}, \mathbf{z})$ is fixed (Papamakarios & Murray, 2015). Here a proposal is learnt for either importance sampling (Le et al., 2017) or sequential Monte Carlo (Paige & Wood, 2016) by using stochastic gradient ascent to minimize the reverse KL-divergence between the inference network $q_\psi(\mathbf{z} \mid \mathbf{x})$ and the true posterior $p(\mathbf{z} \mid \mathbf{x})$. Up to a constant, the objective is given by $\mathcal{L}_{IC} = \mathbb{E}_{p(\mathbf{x},\mathbf{z})}\left[-\ln q_\psi(\mathbf{z} \mid \mathbf{x})\right]$.

Using a minimally faithful inverse structure typically improves the best inference network attainable and the finite time training performance for both these settings, compared with previous naive approaches. In the VAE setting, this can further have a knock-on effect on the quality of the learned model $p_\theta(\mathbf{x}, \mathbf{z})$, both because a better inference network will give lower variance updates of the generative network (Rainforth et al., 2018) and because restrictions in the expressiveness of the inference network lead to similar restrictions in the generative network (Cremer et al., 2017, 2018).

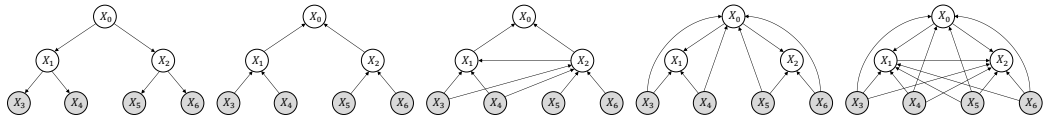
In deep generative models, the BNs may be much larger than the examples shown here. However, typically at the macro-level, where we collapse each vector to a single node, they are quite simple. When we invert this type of collapsed graph, we must do so with the understanding that the distribution over a vector-valued node in the inverse must express dependencies between all its elements in order for the inference network to be faithful.

3 Experiments

We now consider the empirical impact of using NaMI compared with previous approaches. In §3.1, we highlight the importance of using a faithful inverse in the VAE context, demonstrating that doing so results in a tighter variational bound and a higher log-likelihood. In §3.2, we use NaMI in the fixed-model setting. Here our results demonstrate the importance of using both a faithful and minimal inverse on the efficiency of the learned inference network. Low-level details on the experimental setups can be found in Appendix D and an implementation at `https://git.io/fxVQu`.

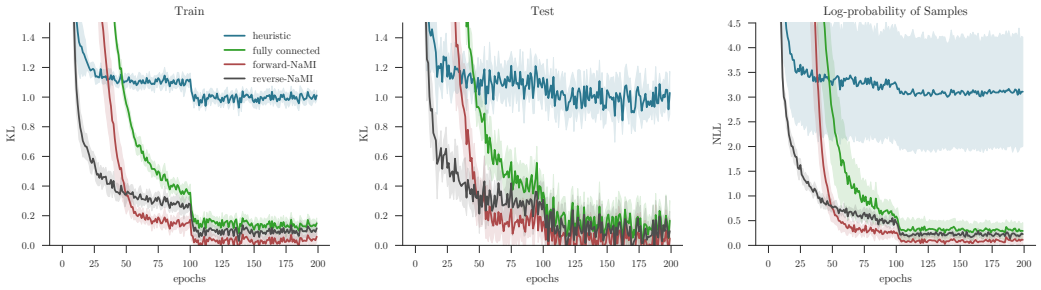
3.1 Relaxed Bernoulli VAEs

Prior work has shown that more expressive inference networks give an improvement in amortized SVI on sigmoid belief networks and standard VAEs, relative to using the mean-field approximation (Uria et al., 2016; Maaløe et al., 2016; Rezende & Mohamed, 2015; Kingma et al., 2016). Krishnan et al. (2017) report similar results when using more expressive inverses in deep linear-chain state-space models. It is straightforward to see that any minimally faithful inverse for the standard VAE



(a) (b) (c) (d) (e)

Figure 5: (a) BN structure for a binary tree with $d = 3$; (b) Stuhlmüller's heuristic inverse; (c) Natural minimally faithful inverse produced by NaMI in topological mode; (d) Most compact inverse when $d > 3$, given by running NaMI in reverse topological mode; (e) Fully connected inverse.



(a) (b) (c)

Figure 6: Results for binary tree Gaussian BNs with depth $d = 5$, comparing inference network factorizations in the compiled inference setting. The KL divergence from the analytical posterior estimated to the inference network on the training and test sets are shown in (a) and (b) respectively. (c) shows the average negative log-likelihood of inference network samples under the analytical posterior, conditioning on five held-out data sets. The results are averaged over 10 runs and 0.75 standard deviations indicated. The drop at 100 epochs is due to decimating the learning rate.

framework (Kingma & Welling, 2014) has a fully connected clique over the latent variables so that the inference network can take account of the explaining-away effects between the latent variables in the generative model. As such, both forward-NaMI and backward-NaMI produce the same inverse.

The relaxed Bernoulli VAE (Maddison et al., 2017b; Jang et al., 2017) is a VAE variation that replaces both the prior on the latents and the distribution over the latents given the observations with the relaxed Bernoulli distribution (also known as the Concrete distribution). It can also be understood as a "deep" continuous relaxation of sigmoid belief networks.

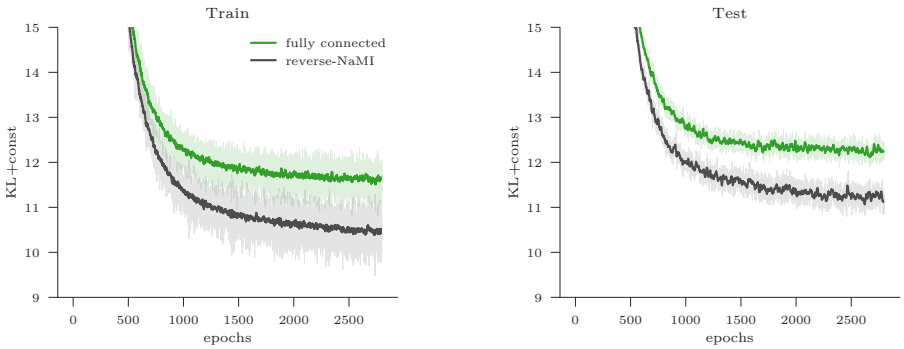
We learn a relaxed Bernoulli VAE with 30 latent variables on MNIST, comparing a faithful inference network (parametrized with MADE (Germain et al., 2015)) to the mean-field approximation, after 1000 epochs of learning for ten different sizes of inference network, keeping the size of the generative network fixed. We note that the mean-field inference network has the same structure as the heuristic one that reverses the edges from the generative model. A tight bound on the marginal likelihood is estimated with annealed importance sampling (AIS) (Neal, 1998; Wu et al., 2017).

The results shown in Figure 4 indicate that using a faithful inverse on this model produces a significant improvement in learning over the mean-field inverse. Note that the x-axis indicates the number of parameters in the inference network. We observe that for *every* capacity level, the faithful inference network has a lower negative ELBO and AIS estimate than that of the mean-field inference network. In Figure 4c, the variational gap is observed to decrease (or rather, the variational bound tightens) for the faithful inverse as its capacity is increased, whereas it increases for the mean-field inverse. This example illustrates the inadequacy of the mean-field approximation in certain classes of models, in that it can result in significantly underutilizing the capacity of the model.

3.2 Binary-tree Gaussian BNs

Gaussian BNs are a class of models in which the conditional distribution of each variable is normally distributed, with a fixed variance and a mean that is a fixed linear combination of its parents plus an offset. We consider here Gaussian BNs with a binary-tree structured graph and observed leaves (see Figure 5a for the case of depth, $d = 3$). In this class of models, the exact posterior can be calculated analytically (Koller & Friedman, 2009, §7.2) and so it forms a convenient test-bed for performance.

The heuristic inverses simply invert the edges of the graph (Figure 5b), whereas a natural minimally faithful inverse requires extra edges between subtrees (e.g. Figure 5c) to account for the influence one



(a) (b)

Figure 7: Convergence of reverse KL divergence (used as the training objective) for Bayesian GMM for $K = 3$ clusters and $N = 200$ data points, comparing inference networks with a fixed generative model. The shaded regions indicate 1 standard error in the estimation.

node can have on others through its parent. For this problem, it turns out that running reverse-NaMI (Figure 5d) produces a more compact inverse than forward-NaMI. This, in fact, turns out to be the most compact possible I-map for any $d > 3$. Nonetheless, all three inversion methods have significantly fewer edges than the fully connected inverse (Figure 5e).

The model is fixed and the inference network is learnt from samples from the generative model, minimizing the "reverse" KL-divergence, namely that from the posterior to the inference network $\mathrm{KL}(p_\theta(\mathbf{z}|\mathbf{x})||q_\psi(\mathbf{z}|\mathbf{x}))$, as per (Paige & Wood, 2016). We compared learning across the inverses produced by using Stuhlmüller's heuristic, forward-NaMI, reverse-NaMI, and taking the fully connected inverse. The fully connected inference network was parametrized using MADE (Germain et al., 2015), and the forward-NaMI one with a novel MADE variant that modifies the masking matrix to exactly capture the tree-structured dependencies (see Appendix E.2). As the same MADE approaches cannot be used for heuristic and reverse-NaMI inference networks, these were instead parametrized with a separate neural network for each variable's density function. The inference network sizes were kept constant across approaches.

Results are given in Figure 6 for depth $d = 5$ averaging over 10 runs. Figures 6a and 6b show an estimate of $\mathrm{KL}(p_\theta(\mathbf{z}|\mathbf{x})||q_\psi(\mathbf{z}|\mathbf{x}))$ using the train and test sets respectively. From this, we observe that it is necessary to model at least the edges in an I-map for the inference network to be able to recover the posterior, and convergence is faster with fewer edges in the inference network. Despite the more compact reverse-NaMI inverse converging faster than the forward-NaMI one, the latter seems to converges to a better final solution. This may be because the MADE approach could not be used for the reverse-NaMI inverse, but this is a subject for future investigation nonetheless.

Figure 6c shows the average negative log-likelihood of 200 samples from the inference networks evaluated on the analytical posterior, conditioning on five fixed datasets sampled from the generative model not seen during learning. It is thus a measure of how successful inference amortiziation has been. All three faithful inference networks have significantly lower variance over runs compared to the unfaithful inference network produced by Stuhlmüller's algorithm.

We also observed during other experimentation that if one were to decrease the capacity of all methods, learning remains stable in the natural minimally faithful inverse at a threshold where it becomes unstable in the fully connected case and in Stuhlmüller's inverse.

3.3 Gaussian Mixture Models

Gaussian mixture models (GMMs) are a clustering model where the data $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ is assumed to have been generated from one of K clusters, each of which has a Gaussian distribution with parameters $\{\mu_j, \Sigma_j\}$, $j = 1, 2, \ldots, K$. Each datum, x_i is associated with a corresponding index, $z_i \in \{1, \ldots, K\}$ that gives the identity of that datum's cluster. The indices, $\mathbf{z}' = \{z_i\}$ are drawn i.i.d. from a categorical distribution with parameter ϕ. Prior distributions are placed on $\theta = \{\mu_1, \Sigma_1, \ldots, \mu_K, \Sigma_K\}$ and ϕ, so that the latent variables are $\mathbf{z} = \{\mathbf{z}', \theta, \phi\}$. The goal of inference is then to determine the posterior $p(\mathbf{z} \mid \mathbf{x})$, or some statistic of it.

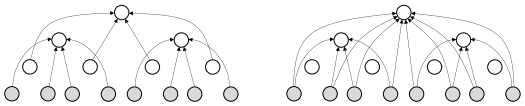
As per the previous experiment, this falls into the fixed-model setting. We factor the fully-connected inverse as, $q(\theta|x)q(\phi|\theta, \mathbf{x})q(\mathbf{z}'|\phi, \theta, \mathbf{x})$. It turns out that applying reverse-NaMI de-

couples the dependence between the indices, $\mathbf{z}'$, and produces a much more compact factorization, $q(\theta|\mathbf{x}, \phi) \prod_i^N q(z_i|x_i, \phi, \theta) q(\phi|\mathbf{x})$, than either the fully-connected or forward-NaMI inverses for this model. The inverse structure produced by Stuhlmüller's heuristic algorithm is very similar to the reverse-NaMI structure for this problem and is omitted.

We train our amortization artifact over datasets with $N = 200$ samples and $K = 3$ clusters. The inference network terms with distributions over vectors were parametrized by MADE, and we compare the results for the fully-connected and reverse-NaMI inverses. We hold the neural network capacities constant across methods and average over 10 runs, the results for which are shown in Figure 7. We see that learning is faster for the minimally faithful reverse-NaMI method, relative to the fully-connected inverse, and converges to a better solution, in agreement with the other experiments.

3.4 Minimal and Non-minimal Faithful Inverses

To further examine the hypothesis that a non-minimal faithful inverse has slower learning and converges to a worse solution relative to a minimal one, we performed the setup of Experiment 3.2 with depth d = 4, comparing the forward-NaMI network to two additional networks that added 12 and 16 connections to forward-NaMI (holding the total capacity fixed). The additional edges are shown in Figure 8. Note the regular forward-NaMI edges are omitted for visual clarity.



(a) 12 skips edges (b) 16 skips edges

Figure 8: Additional edges over forward-NaMI.

Figure 9 shows the average negative log likelihood (NLL) under the true posterior for samples generated by the inference network, based on 5 datasets not seen during training. It appears that the more edges are added beyond minimality, the slower is the initial learning and convergence is to a worse solution.

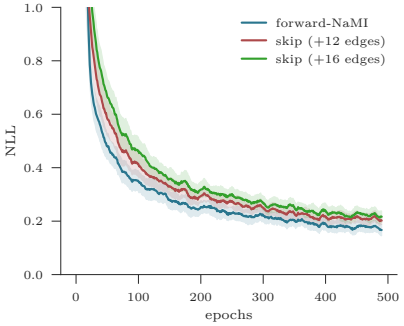


Figure 9: Average NLL of inference network samples under analytical posterior.

To further explain why minimality is crucial, we note that adding additional edges beyond minimality means that there will be factors that condition on variables whose probabilistic influence is blocked by the other variables. This effectively adds an input of random noise into these factors, which is why we then see slower learning and convergence to a worse solution.

4 Discussion

We have presented NaMI, a tractable framework that, given the BN structure for a generative model, produces a natural factorization for its inverse that is a minimal I-map for the model. We have argued that this should be used to guide the design of the coarse-grain structure of the inference network in amortized inference. Having empirically analyzed the implications of using NaMI, we find that it learns better inference networks than previous heuristic approaches. We further found that, in the context of VAEs, improved inference networks have a knock-on effect on the generative network, improving the generative networks as well.

Our framework opens new possibilities for learning structured deep generative models that combine traditional Bayesian modeling by probabilistic graphical models with deep neural networks. This allows us to leverage our typically strong knowledge of which variables effect which others, while not overly relying on our weak knowledge of the exact functional form these relationships take.

To see this, note that if we forgo the niceties of making mean-field assumptions, we can impose arbitrary structure on a generative model simply by controlling its parameterization. The only requirement on the generative network to evaluate the ELBO is that we can evaluate the network density at a given input. Recent advances in normalizing flows (Huang et al., 2018; Chen et al., 2018) mean it is possible to construct flexible and general purpose distributions that satisfy this requirement and are amenable to application of dependency constraints from our graphical model. This obviates the need to make assumptions such as conjugacy as done by, for example, Johnson et al. (2016).

NaMI provides a critical component to constructing such a framework, as it allows one to ensure that the inference network respects the structural assumptions imposed on the generative network, without which a tight variational bound cannot be achieved.

Acknowledgments

We would like to thank (in alphabetical order) Rob Cornish, Rahul Krishnan, Brooks Paige, and Hongseok Yang for their thoughtful help and suggestions.

SW and AG gratefully acknowledge support from the EPSRC AIMS CDT through grant EP/L015987/2. RZ acknowledges support under DARPA D3M, under Cooperative Agreement FA8750-17-2-0093. NS was supported by EPSRC/MURI grant EP/N019474/1. TR and YWT are supported in part by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 617071. TR further acknowledges support of the ERC StG IDIU. FW was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1, DARPA PPAML through the U.S. AFRL under Cooperative Agreement FA8750-14-2-0006, an Intel Big Data Center grant, and DARPA D3M, under Cooperative Agreement FA8750-17-2-0093.

References

Burda, Yuri, Grosse, Roger, and Salakhutdinov, Ruslan. Importance weighted autoencoders. *International Conference on Learning Representations*, 2016.

Chen, Tian Qi, Rubanova, Yulia, Bettencourt, Jesse, and Duvenaud, David. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Cremer, Chris, Morris, Quaid, and Duvenaud, David. Reinterpreting importance-weighted autoencoders. *International Conference on Learning Representations Workshop Track*, 2017.

Cremer, Chris, Li, Xuechen, and Duvenaud, David. Inference suboptimality in variational autoencoders. *Proceedings of the International Conference on Machine Learning*, 2018.

Fishelson, Maáyan and Geiger, Dan. Optimizing exact genetic linkage computations. *Journal of Computational Biology*, 11(2-3):263–275, 2004.

Gan, Zhe, Li, Chunyuan, Henao, Ricardo, Carlson, David E, and Carin, Lawrence. Deep temporal sigmoid belief networks for sequence modeling. *Advances in Neural Information Processing Systems*, 2015.

Germain, Mathieu, Gregor, Karol, Murray, Iain, and Larochelle, Hugo. MADE: masked autoencoder for distribution estimation. *Proceedings of the International Conference on Machine Learning*, 2015.

Gershman, Samuel J and Goodman, Noah D. Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 2014.

Huang, Chin-Wei, Krueger, David, Lacoste, Alexandre, and Courville, Aaron. Neural Autoregressive Flows. *Proceedings of the International Conference on Machine Learning*, 2018.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with Gumbel-softmax. *International Conference on Learning Representations*, 2017.

Johnson, Matthew J, Duvenaud, David, Wiltschko, Alexander B, Datta, Sandeep R, and Adams, Ryan P. Composing graphical models with neural networks for structured representations and fast inference. *arXiv preprint arXiv:1603.06277v2 [stat.ML]*, 2016.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Improving variational inference with Inverse Autoregressive Flow. *Advances in Neural Information Processing Systems*, 2016.

Koller, Daphne and Friedman, Nir. *Probabilistic Graphical Models*. MIT Press, 2009. ISBN 9780262013192.

Krishnan, Rahul G, Shalit, Uri, and Sontag, David. Structured inference networks for nonlinear state space models. *Proceedings of the national conference on Artificial intelligence (AAAI)*, 2017.

Le, Tuan Anh, Baydin, Atilim Gunes, and Wood, Frank. Inference compilation and universal probabilistic programming. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2017.

Le, Tuan Anh, Igl, Maximilian, Jin, Tom, Rainforth, Tom, and Wood, Frank. Auto-encoding Sequential Monte Carlo. In *International Conference on Learning Representations*, 2018.

Maaløe, Lars, Sønderby, Casper Kaae, Sønderby, Søren Kaae, and Winther, Ole. Auxiliary deep generative models. In *Proceedings of the International Conference on Machine Learning*, 2016.

Maddison, Chris J, Lawson, John, Tucker, George, Heess, Nicolas, Norouzi, Mohammad, Mnih, Andriy, Doucet, Arnaud, and Teh, Yee. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, 2017a.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The Concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017b.

Naesseth, Christian A, Linderman, Scott W, Ranganath, Rajesh, and Blei, David M. Variational Sequential Monte Carlo. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2018.

Neal, Radford M. Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, 1990.

Neal, Radford M. Annealed Importance Sampling (technical report 9805 (revised)). *Department of Statistics, University of Toronto*, 1998.

Paige, Brooks and Wood, Frank. Inference networks for Sequential Monte Carlo in graphical models. In *Proceedings of the International Conference on Machine Learning*, 2016.

Papamakarios, George and Murray, Iain. Distilling intractable generative models. In *Probabilistic Integration Workshop at Neural Information Processing Systems*, 2015.

Rainforth, Tom, Kosiorek, Adam R, Le, Tuan Anh, Maddison, Chris J, Igl, Maximilian, Wood, Frank, and Teh, Yee Whye. Tighter variational bounds are not necessarily better. *Proceedings of the International Conference on Machine Learning*, 2018.

Ranganath, Rajesh, Tang, Linpeng, Charlin, Laurent, and Blei, David M. Deep exponential families. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2015.

Rezende, Danilo and Mohamed, Shakir. Variational inference with normalizing flows. In *Proceedings of the International Conference on Machine Learning*, 2015.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the International Conference on Machine Learning*, 2014.

Ritchie, Daniel, Horsfall, Paul, and Goodman, Noah D. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

Stuhlmüller, Andreas, Taylor, Jacob, and Goodman, Noah. Learning stochastic inverses. In *Advances in Neural Information Processing Systems*, 2013.

Uria, Benigno, Côté, Marc-Alexandre, Gregor, Karol, Murray, Iain, and Larochelle, Hugo. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

van den Oord, Aaron, Kalchbrenner, Nal, Espeholt, Lasse, Vinyals, Oriol, Graves, Alex, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, 2016a.

van den Oord, Aaron, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2016b.

Wu, Yuhuai, Burda, Yuri, Salakhutdinov, Ruslan, and Grosse, Roger. On the quantitative analysis of decoder-based generative models. In *International Conference on Learning Representations*, 2017.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Faithful Inversion of Generative Models for Effective Amortized Inference
Publication Status	X Published □Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Webb, Stefan, Golinski, Adam, Zinkov, Robert, N., Siddharth, Rainforth, Tom, Teh, Yee Whye, and Wood, Frank. Faithful Inversion of Generative Models for Effective Amortized Inference. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montreal, Canada*.

Student Confirmation

Student Name:	Stefan Webb		
Contribution to the Paper	Conceived idea for work, developed NaMI algorithm after suggestion from supervisor, wrote first draft of paper, performed experiments in 3.1, 3.2, and 3.4. Made posters for workshops and conference, camera ready version, and released source code repository.		
Signature		Date	12/12/2018

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof Yee Whye Teh		
Supervisor comments		
Signature	Date	

4

Distributed Bayesian Learning with SNEP and the Posterior Server

Journal of Machine Learning Research 18 (2017) 1-37 Submitted 9/16; Revised 7/17; Published 10/17

Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server

Leonard Hasenclever HASENCLEVER@STATS.OX.AC.UK
Department of Statistics, University of Oxford, United Kingdom

Stefan Webb STEFANW@ROBOTS.OX.AC.UK
Department of Engineering Sciences, University of Oxford, United Kingdom

Thibaut Lienart LIENART@STATS.OX.AC.UK
Department of Statistics, University of Oxford, United Kingdom

Sebastian Vollmer VOLLMER@STATS.OX.AC.UK
Department of Statistics, University of Oxford, United Kingdom

Balaji Lakshminarayanan BALAJILN@GOOGLE.COM
DeepMind

Charles Blundell CBLUNDELL@GOOGLE.COM
DeepMind

Yee Whye Teh Y.W.TEH@STATS.OX.AC.UK
Department of Statistics, University of Oxford, United Kingdom

Editor: Manfred Opper

Abstract

This paper makes two contributions to Bayesian machine learning algorithms. Firstly, we propose stochastic natural gradient expectation propagation (SNEP), a novel alternative to expectation propagation (EP), a popular variational inference algorithm. SNEP is a black box variational algorithm, in that it does not require any simplifying assumptions on the distribution of interest, beyond the existence of some Monte Carlo sampler for estimating the moments of the EP tilted distributions. Further, as opposed to EP which has no guarantee of convergence, SNEP can be shown to be convergent, even when using Monte Carlo moment estimates. Secondly, we propose a novel architecture for distributed Bayesian learning which we call the posterior server. The posterior server allows scalable and robust Bayesian learning in cases where a data set is stored in a distributed manner across a cluster, with each compute node containing a disjoint subset of data. An independent Monte Carlo sampler is run on each compute node, with direct access only to the local data subset, but which targets an approximation to the global posterior distribution given all data across the whole cluster. This is achieved by using a distributed asynchronous implementation of SNEP to pass messages across the cluster. We demonstrate SNEP and the posterior server on distributed Bayesian learning of logistic regression and neural networks.

Keywords: Distributed Learning, Large Scale Learning, Deep Learning, Bayesian Learning, Variational Inference, Expectation Propagation, Stochastic Approximation, Natural Gradient, Markov chain Monte Carlo, Parameter Server, Posterior Server.

1. Introduction

Algorithms and systems for enabling machine learning from large scale data sets are becoming increasingly important in the era of Big Data. This has driven many developments, including various forms of stochastic gradient descent, parallel and distributed learning systems, use of GPUs, sketching, random Fourier features, divide-and-conquer methods, as well as various approximation schemes. These large scale machine learning systems have in turn driven significant advances across many data-oriented sciences and technologies, ranging from the biological sciences, neuroscience, social sciences, signal processing, speech processing, natural language processing, computer vision etc.

In this paper we will consider methods for large scale *Bayesian* machine learning. As opposed to the more common empirical risk minimisation or maximum likelihood approaches, where learning is phrased as finding a set of parameters optimal with respect to a data set and to a loss or likelihood function, Bayesian machine learning rests upon probabilistic models which capture the dependencies among all observed and unobserved variables, and where learning is phrased as computing the posterior distribution over unobserved variables (including both latent variables and model parameters) given the observed data.

The Bayesian framework can more fully capture the uncertainty in learnt parameters and prevent overfitting. In principle this allows the use of more complex and larger scale models. However, Bayesian approaches are generally more computationally intensive than optimisation-based ones, and have to date not led to methods which are as scalable.

For complex models, exact computation of the posterior distribution is intractable and approximate schemes such as variational inference (VI) (Wainwright and Jordan, 2008), Markov chain Monte Carlo (MCMC) (Gilks et al., 1996) and sequential Monte Carlo (Doucet et al., 2001) are needed. Scalable methods in both traditions include: stochastic variational inference (Hoffman et al., 2013, Mnih and Gregor, 2014, Rezende et al., 2014) which apply minibatch stochastic gradient descent (Robbins and Monro, 1951) to optimise the variational objective function, stochastic gradient MCMC (Welling and Teh, 2011, Patterson and Teh, 2013, Ding et al., 2014, Teh et al., 2015, Leimkuhler, B. and Shang, X., 2016, Ma et al., 2015, Li et al., 2016) which uses minibatch stochastic gradients within MCMC, austerity MCMC (Korattikara et al., 2014, Bardenet et al., 2014) which uses data subsampling to reduce computational cost of Metropolis-Hastings acceptance steps, and embarrassingly parallel MCMC (Huang and Gelman, 2005, Scott et al., 2013, Wang and Dunson, 2013, Neiswanger et al., 2014) which distribute data across a cluster, runs independent MCMC samplers on each worker and combines samples across the cluster only at the end to reduce network communication costs. In addition, standard learning schemes have also been successfully deployed in large scale settings. A successful example that is particularly relevant to our work is expectation propagation (EP), introduced by (Minka, 2001) and (Opper and Winther, 2000) which iteratively constructs approximations to a factorized distribution. EP is at the heart of the TrueSkill XBox player rating and matching system (Herbrich et al., 2007).

Our work builds upon prior work on using EP for performing distributed Bayesian learning (Xu et al., 2014, Gelman et al., 2014). In this framework, a data set is partitioned into disjoint subsets with each subset stored on a worker node in a cluster. Learning is performed at each worker based on the data subset there using MCMC sampling. As

opposed to embarrassingly parallel MCMC methods which only communicate the samples to the master at the end of learning, EP is used to communicate messages (infrequently) across the cluster. These messages coordinate the samplers such that the target distributions (which coincidentally are the tilted distributions in EP) of all samplers on all workers share certain moments, e.g. means and variances, hence the name sampling via moment sharing (SMS) coined by (Xu et al., 2014). At convergence, it can be shown that the target distributions of the samplers also share moments with the EP approximation to the global posterior distribution given all data, hence the target distributions on the workers can themselves be treated as approximations to the global posterior.

While SMS works well on simpler models like Bayesian logistic regression, we have found that it did not work for more complex, high-dimensional, and non-convex models like Bayesian deep neural networks. This is due to the non-convergence of EP, particularly as the moments of the tilted distributions needed by EP are estimated using MCMC sampling, with estimation noise that further compounds the well-known lack of convergence guarantees for EP, and the fact that extremely long MCMC runs are needed for the samplers to equilibrate due to the complex posterior distribution in these models.

Our first contribution is thus the development of stochastic natural-gradient EP (SNEP), an alternative algorithm to power EP (a generalisation of EP) (Minka, 2004) which optimises the same variational objective function. SNEP is a double-loop algorithm with convergence guarantees. The inner loop is a stochastic natural-gradient descent algorithm which tolerates estimation noise, so that SNEP is convergent even with moments estimated using MCMC samplers. Our derivation of SNEP improves upon the derivation of the convergent EP algorithm of (Heskes and Zoeter, 2002) in that ours works for a more general class of models, we make explicit the underlying variational objective function that is being optimised, and ours uses a natural-gradient descent algorithm (Amari and Nagaoka, 2001) more tolerant of Monte Carlo noise.

Building upon the development of SNEP, our second contribution is a distributed Bayesian learning architecture which we call the posterior server. In analogy to the parameter server (Ahmed et al., 2012) which maintains and serves the parameter vector to a cluster of workers, the posterior server maintains and serves (an approximation to) the posterior distribution. Figure 1 gives a schematic for the steps involved. Each worker has a subset of data, from which we get a likelihood function. It also maintains a tractable approximation of the likelihood and a cavity distribution which is effectively a conditional distribution over the parameters given all data on other workers. An MCMC sampler targets the normalised product of the cavity distribution and the (true) likelihood, and forms a stochastic estimate of the required moments, which is in turn used to update the likelihood approximation using stochastic natural-gradient descent. Each worker communicates with the posterior server asynchronously and in a non-blocking manner, sending the current likelihood approximation and receiving the new cavity distribution. This communication protocol makes more efficient use of computational resources on workers than SMS, which requires either synchronous or blocking asynchronous protocols.

Note that our set-up of the distributed learning problem is that each worker has access to a subset of data, and no worker has access to all data. This situation might occur in situations other than large scale learning. For example, when working with sensitive patient data which cannot be shared directly, we might still want to be able to make use

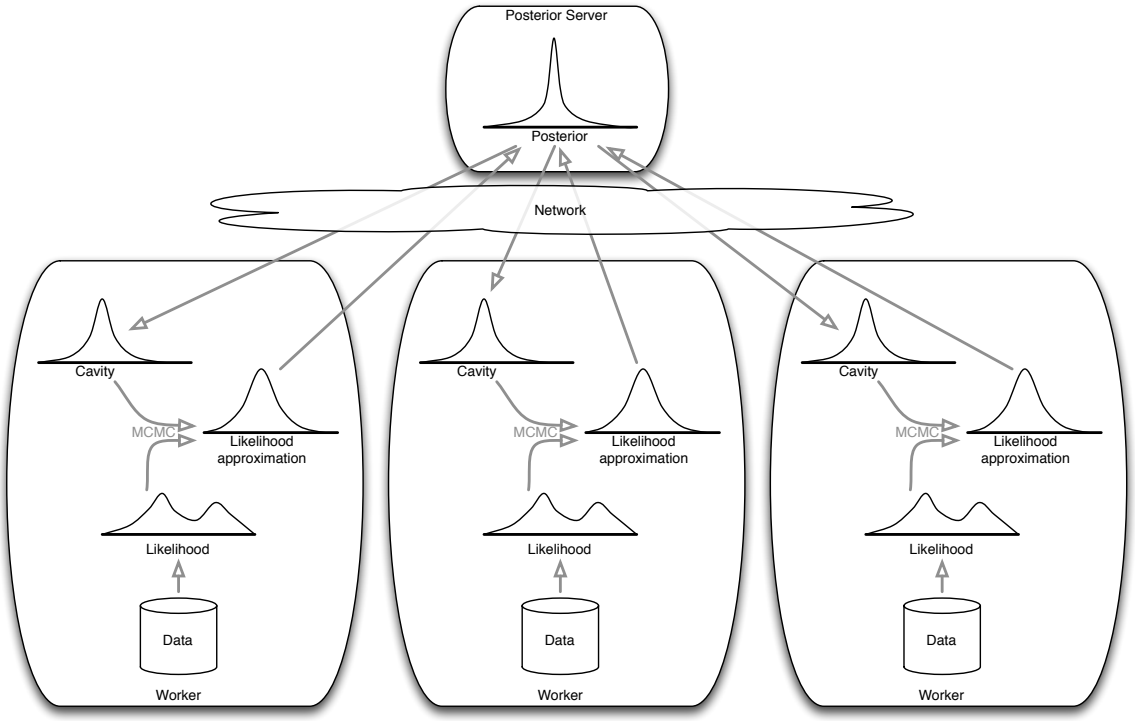


Figure 1: The posterior server.

of all available data across multiple sites to improve inference. Typically known as divide-and-conquer or consensus inference (Zhang et al., 2015b, Zhao et al., 2016, Kleiner et al., 2014, Battey et al., 2015), it is also well-known that this situation is harder than typical distributed learning settings where it is assumed that all data is accessible on all workers, which are settings assumed in DistBelief networks (Dean et al., 2012) and elastic-averaging SGD (Zhang et al., 2015a), two state-of-the-art distributed learning algorithms for neural networks.

In the following, Section 2 describes our set up of distributed Bayesian learning and reviews the necessary background on exponential families and convex duality. Section 3 formulates EP and power EP within the framework of variational inference. Our contributions are contained in Section 4 deriving SNEP and Section 5 describing the posterior server architecture. Section B describes additional techniques we used to make the method work on more complex problems like neural networks. We demonstrate the approach Bayesian logistic regression and Bayesian neural networks in Section 6. Section 7 concludes with a summary and discussion of future work.

2. Problem Set-up and Background

In this section we set-up the problem of distributed Bayesian learning, using the framework of variational inference in exponential families. For an excellent introduction to exponential families and variational inference we refer the interested reader to (Wainwright and Jordan, 2008). Section 2.1 reviews exponential families and convex duality while introducing nota-

tion used throughout the paper. Readers familiar with these concepts can skim ahead to section 2.2.

2.1 Exponential Families and Convex Duality

.

Consider an exponential family described by a d-dimensional sufficient statistics function $s(x)$. A member p_θ of this exponential family is parameterized by a natural parameter $\theta \in \mathbb{R}^d$, and has density (with respect to some base measure, say Lebesgue),

$$p_\theta(x) = \exp\left(\theta^\top s(x) - A(\theta)\right),$$
$$A(\theta) = \log \int \exp\left(\theta^\top s(x)\right) dx.$$

The log partition function $A(\theta)$ is convex and finite on the natural domain of the exponential family,

$$\Theta := \{\theta : A(\theta) < \infty\} \subset \mathbb{R}^d,$$

which is a convex subset of $\mathbb{R}^d$.

Associated with any distribution p and the d-dimensional sufficient statistics function $s(x)$ is a mean parameter,

$$\mu := \mathbb{E}_p[s(x)],$$

where $\mathbb{E}_p$ denotes the expectation operator with respect to p. The set of valid mean parameters $\mathcal{M}$ is a closed convex set, which we refer to as the mean domain,

$$\mathcal{M} = \{\mu : \exists \text{ distribution } p \text{ with } \mu = \mathbb{E}_p[s(x)]\} \subset \mathbb{R}^d$$

Given a natural parameter $\theta \in \Theta$, the exponential family member p_θ is also associated with a mean parameter $\mu = \mathbb{E}_\theta[s(x)]$ (where $\mathbb{E}_\theta$ denotes the expectation with respect to p_θ), which we can write as a function of the natural parameters, $\mu(\theta)$. If the exponential family is minimal[1], then the mapping $\theta \mapsto \nabla A(\theta)$ is one-to-one and onto the interior of $\mathcal{M}$, and maps θ to the mean parameter, $\mu(\theta) = \nabla A(\theta)$.

We will assume that our exponential family of interest is minimal.

The convex conjugate of $A(\theta)$ is,

$$A^*(\mu) := \sup_{\theta \in \Theta} \theta^\top \mu - A(\theta).$$

Evaluated at the mean parameter $\mu(\theta)$, the conjugate is the negative entropy of p_θ,

$$A^*(\mu(\theta)) = \mathbb{E}_\theta[\log p_\theta(x)].$$

1. The exponential family is minimal if the d 1-dimensional functions making up the sufficient statistics function $s(x)$ are linearly independent, i.e. $\theta^\top s(x) = 0$ for all x implies $\theta = 0$.

Conversely, we have

$$A(\theta) = \sup_{\mu \in \mathcal{M}} \theta^\top \mu - A^*(\mu)$$

and that the natural parameter associated with μ is $\theta = \theta(\mu) = \nabla A^*(\mu)$. In the following we will make extensive use of the duality between A and A^* and between θ and μ described above.

It is useful to write down formulae for the KL divergence between two exponential family distributions, parameterized by natural and mean parameter pairs θ, μ and θ', μ' respectively:

$$\begin{aligned}
\mathrm{KL}(p_\theta \| p_{\theta'}) &= \mathbb{E}_\theta[\log p_\theta(x) - \log p_{\theta'}(x)] \\
&= \mathbb{E}_\theta[\theta^\top s(x) - A(\theta) - (\theta')^\top s(x) + A(\theta')] \\
&= \mu^\top(\theta - \theta') - A(\theta) + A(\theta') \\
&= A^*(\mu) + A(\theta') - \mu^\top \theta' \\
&= A^*(\mu) - A^*(\mu') + (\mu' - \mu)^\top \theta'. \qquad (1)
\end{aligned}$$

We will write $\mathrm{KL}(\theta \| \theta'), \mathrm{KL}(\mu \| \theta')$ etc. to refer to the same KL divergence between the same two distributions.

As an example, for a diagonal covariance Gaussian of dimension $d/2$, we have

$$\begin{aligned}
p(x) &= \prod_{j=1}^{d/2} \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{1}{2\sigma_j^2}(x_j - u_j)^2\right) \\
&= \exp\left(\sum_{j=1}^{d/2} (u_j \sigma_j^{-2})(x_j) + (-\sigma_j^{-2})(\tfrac{1}{2}x_j^2) - \tfrac{1}{2}(u_j^2 \sigma_j^{-2} + \log(2\pi\sigma_j^2))\right)
\end{aligned}$$

So the sufficient statistics are x_j and $\frac{1}{2}x_j^2$, mean parameters are $\mu_{j1} = u_j$ and $\mu_{j2} = \frac{1}{2}(u_j^2 + \sigma_j^2)$, natural parameters are $\theta_{j1} = u_j \sigma_j^{-2}$ and $\theta_{j2} = -\sigma_j^{-2}$, and

$$\begin{aligned}
A(\theta) &= \sum_{j=1}^{d/2} \frac{1}{2}(u_j^2 \sigma_j^{-2} + \log(2\pi\sigma_j^2)) \\
A^*(\mu) &= \sum_{j=1}^{d/2} \frac{-1}{2}(1 + \log(2\pi\sigma_j^2))
\end{aligned}$$

The conversions between natural and mean parameters are:

$$\begin{aligned}
u_j &= \mu_{j1} = -\theta_{j1}\theta_{j2}^{-1} & \theta_{j1} &= \mu_{j1}(2\mu_{j2} - \mu_{j1}^2)^{-1} & \mu_{j1} &= -\theta_{j1}\theta_{j2}^{-1} \\
\sigma_j^2 &= 2(\mu_{j2} - \mu_{j1}^2) = -\theta_{j2}^{-1} & \theta_{j2} &= -(2\mu_{j2} - \mu_{j1}^2)^{-1} & \mu_{j2} &= \frac{1}{2}(\theta_{j1}^2 \theta_{j2}^{-2} - \theta_{j2}^{-1})
\end{aligned}$$

We will use a diagonal covariance Gaussian as our exponential family in most of our experiments, due to the high-dimensionality of the models used.

2.2 Distributed Bayesian Learning

We assume that our model is parameterized by a high-dimensional parameter vector x. Let the prior distribution $p_0(x)$ be a member of a tractable and minimal exponential family distribution, with natural parameter $\theta_0 \in \Theta \subset \mathbb{R}^d$, sufficient statistics function $s(x)$ and log partition function $A(\theta_0)$. Specifically, we will take $p_0(x)$ to be a diagonal covariance Gaussian. We refer to this exponential family as the *base exponential family.*

We assume that our training data set is spread across a cluster of n compute nodes or workers, with log likelihood $\ell_i(x)$ on compute node $i = 1, \ldots, n$. For example, if we let $\{S_i\}$ be a partition of the data indices, each compute node i could store the corresponding subset of the data $D_i = \{y_c\}_{c \in S_i}$, so that the log likelihood $\ell_i(x)$ is a sum over terms, each corresponding to the log density of one data point stored on node i,

$$\ell_i(x) = \sum_{c \in S_i} \log p(y_c \mid x).$$

Covariates and input vectors can be easily incorporated in the above. The target posterior distribution is then,

$$\tilde{p}(x) := p(x \mid \{D_i\}_{i=1}^n) \propto p_0(x) \exp\left(\sum_{i=1}^n \ell_i(x)\right). \tag{2}$$

Using neural networks as an example, x corresponds to all learnable weights and biases in a network, $\log p(y_c \mid x)$ gives the probability of the class of data item c given the corresponding input vector, and the Gaussian prior corresponds to weight decay.

The learning task is then to compute the posterior distribution. For example, we may want to estimate the posterior mean or variance of the model parameters x, or we may want to draw samples distributed according to $\tilde{p}(x)$, using these to predict on test data by averaging the predictive densities over the samples as a Monte Carlo estimate of the marginal predictive density. In the rest of the paper we will aim to obtain these efficiently but approximately.

3. Variational Inference in an Extended Exponential Family

In general, the likelihood functions are intractable and approximations are necessary. In this paper we will formulate the learning task as variational inference in an extended exponential family. In particular, we will consider a class of variational methods known as *power expectation propagation* (power EP) (Minka, 2004).

3.1 Extended Exponential Family

To start with, we may trivially formulate the target posterior distribution as an *extended exponential family distribution* with sufficient statistics $\tilde{s}(x) := [s(x), \ell_1(x), \ldots, \ell_n(x)]$ and natural parameters $\tilde{\theta} := [\theta_0, \mathbf{1}_n]$ where $\mathbf{1}_n$ is a vector of 1's of length n:

$$\tilde{p}(x) = \exp\left(\tilde{\theta}^\top \tilde{s}(x) - \tilde{A}(\tilde{\theta})\right).$$

The extended log partition function $\tilde{A}(\tilde{\theta})$ is (up to a constant) the log marginal probability of the data,

$$\begin{aligned}\tilde{A}(\tilde{\theta}) = \log \int \exp\left(\tilde{\theta}^\top \tilde{s}(x)\right) dx &= \log \int \exp\left(\theta_0^\top s(x) + \sum_{i=1}^n \ell_i(x)\right) dx \\ &= \log \mathbb{E}_{\theta_0}\left[\exp\left(\sum_{i=1}^n \ell_i(x)\right)\right] + A(\theta_0) \\ &= \log p(\{D_i\}_{i=1}^n) + A(\theta_0).\end{aligned}$$

Denoting the convex conjugate by $\tilde{A}^*(\tilde{\mu})$ and the extended mean domain by $\tilde{\mathcal{M}} \subset \mathbb{R}^{d+n}$, the problem of posterior computation can be expressed as the following concave variational maximization problem:

$$\max_{\tilde{\mu} \in \tilde{\mathcal{M}}} \tilde{\theta}^\top \tilde{\mu} - \tilde{A}^*(\tilde{\mu}). \tag{3}$$

For example, if the prior exponential family is a diagonal covariance Gaussian, then the optimal mean parameter is $\tilde{\mu}^* := [\mu^*, \nu_1^*, \dots \nu_n^*]$, where $\mu^* := \mathbb{E}_{\tilde{\theta}}[s(x)] \in \mathbb{R}^d$ corresponds to the posterior means and variances of the model parameters x, while $\nu_i^* := \mathbb{E}_{\tilde{\theta}}[\ell_i(x)]$ is the posterior expectation of the ith log likelihood $\ell_i(x)$. Hence the extended mean parameters capture the important aspects of the posterior distribution.

3.2 Power Expectation Propagation

In this section we derive another class of variational approximations corresponding to a generalization of expectation propagation (EP) (Minka, 2001) called power EP (Minka, 2004). The derivation is a straightforward generalisation of the variational formulation of Wainwright and Jordan (2008) from EP to power EP.

For each worker node i let $\beta_i > 0$ be a given positive real number. Typically, we take $\beta_i = 1$ which corresponds to standard EP, while $\beta_i \to \infty$ corresponds to variational Bayes (Wiegerinck and Heskes, 2003, Minka, 2004). In the formulation of Wainwright and Jordan (2008), EP involves two approximations; both associated with simpler exponential families, which we refer to as *locally extended exponential families* (or sometimes *local exponential families*). For each i, let the ith locally extended exponential family be associated with the sufficient statistics function $s_i(x) := [s(x), \ell_i(x)]$. Let Θ_i, $\mathcal{M}_i$, A_i, A_i^* be the associated (local) natural domain, mean domain, log partition function and negative entropy respectively. A distribution in this locally extended exponential family with natural parameter $[\theta_i, \eta_i] \in \Theta_i$ has the form

$$p_{\theta_i,\eta_i}(x) = \exp\left(\theta_i^\top s(x) + \eta_i \ell_i(x) - A_i(\theta, \eta_i)\right), \tag{4}$$

which is a distribution obtained by tilting a tractable distribution with density proportional to $\exp(\theta_i^\top s(x))$ by a single intractable likelihood $\exp(\ell_i(x))$ raised to the power of η_i. This family can be thought of as treating the ith likelihood term exactly, while approximating all other likelihood terms by projecting them onto the tractable base exponential family,

the hope being that this family is still tractable while being closer to the true posterior distribution $\tilde{p}(x)$.

For the first approximation, the extended negative entropy $\tilde{A}^*$ is approximated using a tree-like approximation constructed using only the locally extended negative entropies,

$$\tilde{A}^*([\mu, \nu_1, \ldots, \nu_n]) \approx A^*(\mu) + \sum_{i=1}^{n} \beta_i (A_i^*(\mu, \nu_i) - A^*(\mu)).$$

This approximation is related to the Bethe entropy of loopy belief propagation (Yedidia et al., 2001) and fractional belief propagation (Wiegerinck and Heskes, 2003). Secondly, the extended mean domain $\tilde{\mathcal{M}}$ is approximated by a local mean domain,

$$\mathcal{L} := \{[\mu, \nu_1, \ldots, \nu_n] : [\mu, \nu_i] \in \mathcal{M}_i \text{ for all } i = 1, \ldots, n\}. \tag{5}$$

The local mean domain is an outer bound, $\mathcal{L} \supset \tilde{\mathcal{M}}$. Intuitively, the constraints described by $\tilde{\mathcal{M}}$ are replaced by the weaker constraints described by the local mean domains. We assume that working with the local exponential families in these ways will be more tractable than working with the full extended exponential family.

Let $\mu_0 \in \mathcal{M}$ be a mean parameter in the base exponential family. For the ith local exponential family, we denote a mean parameter by $[\mu_i, \nu_i] \in \mathcal{M}_i$, and require the marginalization constraint $\mu_i = \mu_0$. The power EP variational problem, which is not in general concave, is,

$$\begin{aligned} \max_{\mu_0, [\mu_i, \nu_i]_{i=1}^n} \quad & \theta_0^\top \mu_0 + \sum_{i=1}^{n} 1 \cdot \nu_i - A^*(\mu_0) - \sum_{i=1}^{n} \beta_i (A_i^*(\mu_i, \nu_i) - A^*(\mu_i)) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \\ & \mu_0 = \mu_i \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{6}$$

Note in particular that the entropy and mean domain in the original variational problem (3) have been replaced by their respective approximations.

3.3 Deriving EP and Power EP Updates

In this section, for completeness' sake, we derive the updates for EP and power EP as fixed-point equations that solve the variational problem. Readers familiar with this derivation can skip ahead to section 3.4.

First we introduce Lagrange multipliers λ_i for the equality constraints $\mu_0 = \mu_i$, so that the above is equivalent to,

$$\begin{aligned} \max_{\mu_0, [\mu_i, \nu_i]_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \quad & \underbrace{\theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^{n} \Big(\nu_i - \lambda_i^\top (\mu_i - \mu_0) - \beta_i (A_i^*(\mu_i, \nu_i) - A^*(\mu_i)) \Big)}_{=: L(\mu_0, [\mu_i, \nu_i, \lambda_i]_{i=1}^n)} \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{7}$$

where the domain of λ_i is $\mathbb{R}^d$. Let the Lagrangian above be denoted by $L(\mu_0, [\mu_i, \nu_i, \lambda_i]_{i=1}^n)$. The Karush-Kuhn-Tucker (KKT) conditions of the above variational problem have to be satisfied at an optimum, and simply involve setting the derivatives with respect to $\mu_0, \mu_i, \nu_i, \lambda_i$ to zero:

$$\frac{dL}{d\lambda_i} = 0 : \qquad \mu_i = \mu_0 \tag{8a}$$

$$\frac{dL}{d\mu_0} = 0 : \qquad \theta_0 - \nabla A^*(\mu_0) + \sum_{j=1}^n \lambda_j = 0$$

$$\theta_0 + \sum_{j=1}^n \lambda_j = \nabla A^*(\mu_0) \tag{8b}$$

$$\frac{dL}{d\mu_i} = 0 : \qquad -\lambda_i - \beta_i \nabla_{\mu_i} A_i^*(\mu_i, \nu_i) + \beta_i \nabla A^*(\mu_i) = 0$$

$$\theta_0 + \sum_{j=1}^n \lambda_j - \beta_i^{-1}\lambda_i = \nabla_{\mu_i} A_i^*(\mu_i, \nu_i) \tag{8c}$$

$$\frac{dL}{d\nu_i} = 0 : \qquad \beta_i^{-1} = \nabla_{\nu_i} A_i^*(\mu_i, \nu_i) \tag{8d}$$

Equation (8b) shows that an optimal μ_0 has to be the mean parameter corresponding to a (base) exponential family distribution with natural parameter $\theta_0 + \sum_{j=1}^n \lambda_j$. Specifically, the posterior distribution can be approximated as,

$$\tilde{p}(x) \propto p_0(x) \exp\left(\sum_{j=1}^n \ell_j(x)\right) \approx \exp\left(\left(\theta_0 + \sum_{j=1}^n \lambda_j\right)^\top s(x)\right). \tag{9}$$

In other words, we can interpret λ_j as the natural parameter of an exponential family approximation to the likelihood factor $\exp(\ell_j(x))$.

Further, from (8c) and (8d) above, we see that the optimal $[\mu_i, \nu_i]$ is the mean parameter associated with the local posterior distribution,

$$p_i(x) \propto \exp\left(\left(\theta_0 + \sum_{j=1}^n \lambda_j - \beta_i^{-1}\lambda_i\right)^\top s(x) + \beta_i^{-1}\ell_i(x)\right). \tag{10}$$

For standard EP, where $\beta_i = 1$, we get,

$$p_i(x) \propto \exp\left(\left(\theta_0 + \sum_{j\neq i} \lambda_j\right)^\top s(x) + \ell_i(x)\right).$$

The above local posterior distribution is known as the tilted distribution in EP, with the term $(\theta_0 + \sum_{j=1}^n \lambda_j - \beta_i^{-1}\lambda_i)^\top s(x)$ corresponding to the cavity distribution with (the β_i^{-1}th power of) the exponential family approximation to the ith likelihood removed, and replaced

by (the β_i^{-1}th power of) the likelihood factor itself. Each step of EP involves first computing $[\mu_i, \nu_i]$ as the mean parameter of the tilted distribution, then updating λ_i, using (8a) and (8b):

$$\lambda_i^{\text{new}} = \nabla A^*(\mu_i) - \theta_0 - \sum_{j \neq i} \lambda_j, \tag{11}$$

where we recall that $\nabla A^*(\mu_i)$ computes the natural parameter corresponding to the mean parameter μ_i in the base exponential family. The EP update ensures that the expectation of the sufficient statistics function under the tilted distribution $p_i(x)$ and under its exponential family approximation (with natural parameters $\theta_0 + \sum_{j=1}^n \lambda_j - \beta_i^{-1}\lambda_i + \beta_i^{-1}\lambda_i^{\text{new}}$) match. At convergence, this ensures that the expectations under all the tilted distributions and under the approximated posterior distribution (9) match.

Note that the (power) EP updates are derived as fixed point equations and have no guarantees of convergence. In practice, damped updates can be used to avoid oscillations without affecting the fixed points:

$$\lambda_i^{\text{new}} = \alpha\lambda_i + (1-\alpha)\left(\nabla A^*(\mu_i) - \theta_0 - \sum_{j \neq i} \lambda_j\right). \tag{12}$$

where $\alpha \in [0, 1)$ is a damping factor. Another potential issue is that λ_i^{new} as computed may not lie in the natural domain Θ, in which case higher damping factors ensuring that it does can be used, or the update can be discarded.

3.4 Computing Mean Parameters

Assuming that the base exponential family is tractable, the above steps of EP involve additions, subtractions, and conversions between mean and natural parameters so are easy to compute. The only difficulty is in the computation of the mean parameters (expectations of the sufficient statistics function) of the tilted distributions, which involve the log likelihoods $\ell_i(x)$. In typical applications of EP to graphical models, these are either obtained analytically or using numerical quadrature.

In more complex and general scenarios, both analytic or quadrature-based methods are ruled out. A successful and common class of methods for calculating expectations under otherwise intractable distributions is Monte Carlo. A number of papers have proposed such an approach, including Barthelmé and Chopin (2011), Heess et al. (2013), Gelman et al. (2014) using importance sampling and Xu et al. (2014), Gelman et al. (2014) using Markov chain Monte Carlo (MCMC). In addition, Heess et al. (2013), Eslami et al. (2014), Jitkrittum et al. (2015) use learning techniques to speed-up the process by directly predicting the natural parameters given properties of the tilted distribution.

We have explored the sampling via moment sharing (SMS) algorithm (Xu et al., 2014) in the context of distributed Bayesian learning of deep neural networks. Unfortunately, the SMS algorithm did not work even for relatively small neural networks and data sets, partly because of the high dimensionality and partly because the Monte Carlo estimation is inherently stochastic, both of which we found affected the convergence of the EP fixed point equations.

4. Stochastic Natural-gradient Expectation Propagation

In this section we will derive a novel convergent stochastic approximation based alternative to EP and power EP, which optimises the same variational objective but is significantly more tolerant of Monte Carlo noise. Our algorithm is derived using a modified variational objective with additional auxiliary variables but with the same optima as the original problem, and solving the dual problem using a stochastic approximation algorithm (Robbins and Monro, 1951). In the following we will use "EP" to refer to both EP and power EP for simplicity.

4.1 Auxiliary Variational Problem

For each i, we introduce an auxiliary natural parameter vector $\theta'_i \in \Theta$, and introduce a term $-\sum_{i=1}^n \mathrm{KL}(p_{\mu_i} \| p_{\theta'_i})$ into the variational objective (7), where p_{μ_i} and $p_{\theta'_i}$ are the base exponential family distribution given by mean parameter μ_i and natural parameter θ'_i, respectively. This results in a lower bound on the original objective and is reminiscent of the EM algorithm (Dempster et al., 1977, Neal and Hinton, 1999) and of typical variational Bayes approximations (Beal, 2003, Wainwright and Jordan, 2008). Plugging the relevant formula for the KL divergence (1) into (7), we have the resulting variational problem

$$\begin{aligned} \max_{\mu_0, [\mu_i, \nu_i, \theta'_i]_{i=1}^n} \quad & \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \Big(\nu_i - \beta_i \Big(A_i^*(\mu_i, \nu_i) - \mu_i^\top \theta'_i + A(\theta'_i) \Big) \Big) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \\ & \theta'_i \in \Theta \quad \text{for } i = 1, \ldots, n \\ & \mu_0 = \mu_i \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{13}$$

Maximizing over θ'_i while keeping the other variables fixed will simply set $\theta'_i = \nabla A^*(\mu_i)$, so that the KL terms vanish and resulting in the original problem (6). On the other hand, the constrained maximization over the original parameters $\mu_0, [\mu_i, \nu_i]_{i=1}^n$ requires introducing Lagrange multipliers,

$$\begin{aligned} \max_{[\theta'_i]_{i=1}^n} \max_{\mu_0, [\mu_i, \nu_i]_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \quad & \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \Big(\nu_i - \lambda_i^\top (\mu_i - \mu_0) - \beta_i \Big(A_i^*(\mu_i, \nu_i) - \mu_i^\top \theta'_i + A(\theta'_i) \Big) \Big) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \\ & \theta'_i \in \Theta \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{14}$$

We can solve this inner constrained optimization by transforming to the convex dual. Noticing that the Lagrangian is concave in $\mu_0, [\mu_i, \nu_i]_{i=1}^n$ and that Slater's condition holds, the

duality gap is zero and we have,

$$\max_{[\theta_i']_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \max_{\mu_0,[\mu_i,\nu_i]_{i=1}^n} \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \Big(\nu_i - \lambda_i^\top(\mu_i - \mu_0) - \beta_i\Big(A_i^*(\mu_i,\nu_i) - \mu_i^\top\theta_i' + A(\theta_i')\Big)\Big)$$
$$\begin{aligned} \text{subject to} \qquad \mu_0 &\in \mathcal{M} \\ [\mu_i,\nu_i] &\in \mathcal{M}_i \quad \text{for } i = 1,\dots,n \\ \theta_i' &\in \Theta \quad\;\; \text{for } i = 1,\dots,n \end{aligned} \tag{15}$$

Now maximizing over $\mu_0, [\mu_i,\nu_i]_{i=1}^n$, we have the equivalent dual problem, whose objective function we will denote with $L([\theta_j',\lambda_j]_{j=1}^n)$,

$$\max_{[\theta_i']_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \underbrace{A\left(\theta_0 + \sum_{i=1}^n \lambda_i\right) + \sum_{i=1}^n \beta_i\left(A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right) - A(\theta_i')\right)}_{=:L([\theta_j',\lambda_j]_{j=1}^n)}$$
$$\text{subject to} \quad \theta_i' \in \Theta \quad \text{for } i = 1,\dots,n \tag{16}$$

In summary, our algorithm is coordinate maximization of (13), where we alternate between maximizing $[\theta_i']_{i=1}^n$ given $\mu_0, [\mu_i,\nu_i]_{i=1}^n$, and maximizing $\mu_0, [\mu_i,\nu_i]_{i=1}^n$ given $[\theta_i']_{i=1}^n$. To solve the second maximization, we transform to the dual and minimize (16) with respect to the Lagrange multipliers $[\lambda_i]_{i=1}^n$. This requires an inner iteration loop involving stochastic natural gradient descent, which is described next. The structure of our derived algorithm is the same as the convergent double loop EP algorithm of (Heskes and Zoeter, 2002). Our derivation is interesting and useful in that a single global variational problem is described, rather than a sequence of variational problems each obtained by minorizing around the current parameters and then maximizing, as in the minorization-maximization (MM) algorithm) (Hunter and Lange, 2004) and the CCCP algorithm (Yuille, 2002). Our algorithm is also different in the choice of stochastic natural gradient descent inner loop.

4.2 Stochastic Natural Gradient Descent

The gradient of the dual objective $L([\theta_j',\lambda_j]_{j=1}^n)$ is,

$$\frac{dL}{d\lambda_i} = \nabla A\left(\theta_0 + \sum_{j=1}^n \lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right) \tag{17}$$

where we used the notation $\nabla_{\theta_i} A_i$ for the partial derivative of $A_i(\cdot,\cdot)$ with respect to its first (d-dimensional) argument. While this gradient is the direction of steepest descent in a Euclidean geometry it does not take into account the geometry of the base exponential family. In other words, the gradient is not covariant[2] (MacKay, 1999) and the corresponding gradient descent algorithm is not expected to perform well.

2. A covariant gradient descent algorithm roughly means that the size of the gradient in a direction positively covaries with the length scale in that direction hence with the desired step size. An example of a non-covariant gradient descent algorithm is steepest descent with standard Euclidean geometry, say for a 2D quadratic loss function that is long in the x-axis and short in the y-axis. The gradient will be close to pointing vertically up or down, but the best descent direction points horizontally instead.

A better approach is to use natural gradient descent (Amari et al., 1996, Amari and Nagaoka, 2001) or, equivalently, mirror descent (Beck and Teboulle, 2003, Raskutti and Mukherjee, 2015) instead. These methods treat the parameter space as a Riemannian manifold where the metric tensor is given by the Fisher information. The direction of steepest descent in this geometry is given by the gradient preconditioned by the inverse metric. Natural gradient descent has appealing theoretical properties as it incorporates second order information to be more covariant. As we will see below we can take advantage of natural gradient descent at no additional computational cost by reparametrising. As noted in the previous section, the Lagrange multiplier λ_i can be interpreted as the natural parameters of the base exponential family approximation to the likelihood $\exp(\ell_i(x))$. Reparameterising using the corresponding mean parameter γ_i instead, with $\lambda_i = \nabla A^*(\gamma_i)$, the gradient is,

$$\begin{aligned}\frac{dL}{d\gamma_i} &= \frac{d\lambda_i}{d\gamma_i}\left(\nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right)\right) \\ &= \nabla^2 A^*(\gamma_i)\left(\nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right)\right) \end{aligned} \tag{18}$$

The appropriate metric in the mean parameter space is simply $\nabla^2 A^*(\gamma_i)$, so that its inverse cancels the $\nabla^2 A^*(\gamma_i)$ term (Raskutti and Mukherjee, 2015), and the natural gradient update is simply,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t\left(\nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1}\right) - \nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j^{(t)}\right)\right) \tag{19}$$

where the corresponding natural parameter is given as a function of the mean parameter, $\lambda_i^{(t)} := \nabla A^*(\gamma_i^{(t)})$, and ϵ_t is the step size at iteration t. These updates can be performed in series or parallel fashion. In our distributed Bayesian learning setting they are performed in an asynchronous distributed fashion (Section 5).

Recall that derivatives of the log partition function compute the mean parameters from the natural parameters. Hence the first term of the natural gradient step is the mean parameter of the current local tilted distribution,

$$p_i^{(t)}(x) = \exp\left((\theta_i' - \beta_i^{-1}\lambda_i^{(t)})^\top s(x) + \beta_i^{-1}\ell_i(x) - A_i(\theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1})\right), \tag{20}$$

while the second term is the mean parameter of the current exponential family approximation to the global posterior. Their difference gives the update for the mean parameter of the likelihood approximation. The gradient is zero when both terms are equal, precisely the condition from which the EP fixed point equation is derived.

In general, the first term cannot be obtained in closed form, and we instead use a Markov chain Monte Carlo (MCMC) estimate, leading to a stochastic natural-gradient descent algorithm. Specifically, let $\mathcal{K}_i(\cdot \mid x; \theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i)$ be a Markov chain kernel with previous state x whose invariant distribution is the local tilted distribution (20). Let $x_i^{(t)}$ be

the state of the Markov chain at iteration t. The next state $x_i^{(t+1)}$ is obtained by simulating from the Markov chain, using the current values of the parameters,

$$x_i^{(t+1)} \sim \mathcal{K}_i\left(\cdot \mid x_i^{(t)}; \theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i\right). \tag{21}$$

In summary, the stochastic natural gradient update is,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t \left(s\left(x_i^{(t+1)}\right) - \nabla A\left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)}\right)\right) \tag{22}$$

We refer to the resulting algorithm as "Stochastic Natural-gradient EP", or SNEP for short.

Technically, the stochastic natural-gradient descent requires unbiased estimates of gradients, and the mean parameter estimates obtained using MCMC updates are only unbiased if the Markov chain equilibrates in between gradient updates. In practice, we find that using MCMC samples works well regardless, an observation consistent with past experiences with stochastic approximation algorithms in machine learning (Neal and Hinton, 1999, Tieleman, 2008).

The stochastic natural gradient descent algorithm above serves as the inner loop of our double loop algorithm, and solves for the constrained maximization of the mean parameters $\mu_0, [\mu_i, \nu_i]_{i=1}^n$ given auxiliary parameters $[\theta_i']_{i=1}^n$. Returning now to the outer loop update, maximizing θ_i' involves simply setting it to be $\nabla A^*(\mu_i)$, the natural parameter corresponding to the mean parameter μ_i of the local tilted distribution (see the discussion after Equation 13). Assuming that the inner loop has converged, this and the mean parameters of all other tilted distributions would be equal to μ_0, so that the t'th outer loop update is,

$$(\theta_i')^{(t'+1)} = \nabla A^*(\mu_i) = \nabla A^*(\mu_0) = \theta_0 + \sum_{j=1}^{n} \lambda_j^{(\infty)} \tag{23}$$

where $\lambda_j^{(\infty)}$ is the converged value of the dual parameters in the inner loop. In practice, we simply perform the outer loop update infrequently (and before the inner loop has fully converged). In our experiments we do not see instabilities resulting from this, an observation also consistent with past experiences with double loop algorithms (Teh and Welling, 2002, Yuille, 2002, Heskes and Zoeter, 2002).

In the extreme case where the outer loop update is performed after every inner loop update, we can roll both updates into the following update,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t \left(\nabla_{\theta_i} A_i \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1}\right) - \nabla A\left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)}\right)\right) \tag{24}$$

It is interesting to contrast the above with a damped EP update, which in our notation is given by,

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \epsilon_t \left(\nabla A^* \left(\nabla_{\theta_i} A_i \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1}\right)\right) - \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)}\right)\right) \tag{25}$$

Note that $\nabla A^*(\cdot)$ converts from mean parameters to natural parameters, so that the first term in parentheses can be read as first computing the moments (mean parameters) of the tilted distribution then converting into the corresponding natural parameter. Hence each term in the damped EP update is obtained by converting the corresponding term in (24) from mean to natural parameters, i.e. applying $\nabla A^*(\cdot)$. In other words the update (24) can be thought of as a mean parameter space version of a damped EP update. When mean parameters are estimated with Monte Carlo noise, updating using mean parameters rather than natural parameters leads to more stable behaviour; see Section 6.1 for empirical results supporting this claim.

4.3 Discussion and Related Works

While we developed SNEP in the context of distributed Bayesian learning, it is clear that it is generally applicable, since the distribution (2) targeted is simply a product of factors, each of which is approximated by a factor in the base exponential family, precisely the setting of EP. SNEP can therefore be used in place of EP in situations where Monte Carlo moment estimates are used, including graphical models (Heess et al., 2013, Eslami et al., 2014, Jitkrittum et al., 2015, Lienart et al., 2015), hierarchical Bayesian models (Gelman et al., 2014), and approximate Bayesian computation (Barthelmé and Chopin, 2011).

Since Minka (2001), there have been a substantial number of extensions and alternatives to EP proposed. Stochastic EP (Li et al., 2015) and averaged EP (Dehaene and Barthelmé, 2015) assume that all factors can be well approximated by the *same* exponential family factor via parameter tying. This saves memory storage and was shown to work well. This is an orthogonal idea to our work, and it is possible to apply it to SNEP as well in the future, although in the distributed learning setting considered in this paper each worker must keep a copy of the parameters anyway, which means it would not reduce memory requirements. Convergent EP (Heskes and Zoeter, 2002) is also a double loop coordinate descent algorithm with convergence guarantees but SNEP differs in that the inner loop is a stochastic natural-gradient descent which is more robust to the use of Monte Carlo estimated moments. We note here that stochastic natural-gradient descent has also been used in Stochastic Variational Inference (Hoffman et al., 2013).

SNEP can be considered a black-box variational inference algorithm, as the only requirement on the model is the existence of MCMC samplers targeting the tilted distributions. Black-box methods have recently been developed for variational Bayes (Ranganath et al., 2014, Black-box Variational Inference or BBVI) and for power EP (Hernandez-Lobato et al., 2016, Black-box Alpha or BB-α). BB-α and SNEP are both methods for Bayesian learning based on power EP and both operate in the mean parameter space. BB-α uses the parameter tying idea of stochastic and averaged EP to construct a further approximated objective function which is then optimised using a convergent single-loop algorithm. As a result the fixed points of the algorithm are different from power EP. Single loop algorithms are generally preferrable as it is well known that double loop algorithms can be slow if the inner loop is run until convergence. In our experiments, we do not run the inner loop of SNEP until convergence, and have found empirically that even a single iteration of the inner loop per outer loop update does not affect convergence or stability. In fact, on a two-layer feedforward neural network, SNEP converges in less than one tenth of the number of epochs

and a fraction of the running times of those quoted by Hernandez-Lobato et al. (2016). In both BBVI and BB-α, naive Monte Carlo estimators are used, with samples drawn from the approximating distribution. The resulting estimators can have high variance, requiring techniques for variance reduction. In contrast, SNEP uses MCMC samplers targeting the tilted distribution. It is generally accepted that, in high-dimensional settings, MCMC samplers often have lower variance than naive Monte Carlo and as a result work better, with the tradeoff being that MCMC samplers need to equilibrate and produce correlated samples.

Our application of SNEP to distributed Bayesian learning applies one exponential family approximation per subset of data on each worker node. This contrasts with typical applications of EP and variational inference in general, which applies one approximation per data item. This is made possible due to the black-box flexibility of our approach, since the likelihood associated with a data subset is more complex than for a single data item. In cases where the subset is itself quite large, and the Bernstein-von Mises theorem holds, the likelihood will be close to a Gaussian, so that if we use a (full-covariance) Gaussian as the base exponential family, the approximation will introduce negligible biases. For a recent study of EP in the large data limit, see (Dehaene and Barthelmé, 2015). We can think of our approach as a hybrid which interpolates between a pure variational approach (when $n = N$) and a pure MCMC approach (when $n = 1$), with smaller n corresponding to less bias introduced by approximations but higher variance/computational costs.

In our experiments, we apply SNEP to deep neural networks. Deep learning has been extremely successful in a large number of different domains. However, one drawback of neural networks trained by stochastic optimisation techniques is that they do not provide uncertainty estimates. Both black-box variational methods and stochastic gradient MCMC methods can be applied to neural networks yielding uncertainty estimates. Probabilistic backpropagation (Hernandez-Lobato and Adams, 2015, PBP) relies on assumed density filtering (ADF), a sequential form of EP. Other approaches are based on mean field variational inference, which optimises a lower bound of the marginal likelihood. Graves (2011) proposed an algorithm using a biased Monte Carlo estimate of this lower bound to optimise it. More recently, this approach has been extended by Blundell et al. (2015, Bayes by Backprop) who obtain unbiased Monte Carlo estimates of the same objective using the reparametrization trick (Kingma and Welling, 2014).

5. The Posterior Server

Our development of SNEP is motivated by the problem of distributed Bayesian learning outlined in Section 2.2, where each log likelihood term $\ell_i(x)$ corresponds to the log probability/density of the data subset on worker node i. Using SNEP, each worker node iteratively learns an exponential family approximation of $\ell_i(x)$, with a master node coordinating the learning across workers. We call the master node the *posterior server*, as it maintains and serves the exponential family approximation of the posterior distribution, obtained by combining the prior with the likelihood approximations at the workers.

In more detail, the posterior server maintains the natural parameter $\theta_{\text{posterior}} := \theta_0 + \sum_{j=1}^{n} \lambda_j$ of the global posterior approximation, while each worker node i maintains the mean and natural parameters γ_i, λ_i of the likelihood approximation and the state of the MCMC

sampler x_i. It also maintains the auxiliary parameter θ_i' used in the outer loop. Learning at the worker proceeds by alternating between the MCMC (21) and inner loop updates (22), with periodic outer loop updates (23). These updates require access to the data subset at the node, as well as the cavity distribution, with natural parameters $\theta_{-i} := \theta_0 + \sum_{j \neq i} \lambda_j$, which is obtained by getting $\theta_{\text{posterior}}$ from the posterior server and subtracting the local λ_i.

Communication between the worker node and the posterior server involves the worker first sending the posterior server the difference $\Delta_i := \lambda_i^{\text{new}} - \lambda_i^{\text{old}}$ between the current natural parameters λ_i^{new} and the one during the previous communication with the server, λ_i^{old}. The posterior server updates its global posterior approximation via $\theta_{\text{posterior}}^{\text{new}} = \theta_{\text{posterior}}^{\text{old}} + \Delta_i$, and sends the new value $\theta_{\text{posterior}}^{\text{new}}$ back to the worker. The worker in turn uses this to update the cavity, $\theta_{-i}^{\text{new}} = \theta_{\text{posterior}}^{\text{new}} - \lambda_i^{\text{new}}$.

The pseudocode for the overall algorithm is given in Algorithm 1. Note that all communications are performed asynchronously and in a non-blocking fashion. In particular, Steps 11-17 are performed in a separate coroutine from the main loop (Steps 7-18), and Step 15 can happen several iterations of the main loop after Step 14. This is so that compute nodes can spend most of their time learning (Steps 8-10) and do not have to wait for network communications to complete. However, this means that the previous copy of the natural parameter λ_i used when the last message was sent to the master (Step 13) needs to be stored, in addition to the most recent one. We also note that faster compute nodes need not wait for slower ones since they each learn their own separate likelihood approximation parameters. It would be interesting for future research to explore adaptive methods to allow faster compute nodes to increase the data subsets that they learn from, and slower ones to decrease, to balance the learning progress across compute nodes more evenly.

5.1 Discussion and Related Works

Our naming of the posterior server contrasts with that of the parameter server (Ahmed et al., 2012) which is typically used for maximum likelihood (or minimum empirical risk) estimation of model parameters. Note however that our algorithmic contribution is effectively orthogonal to (Ahmed et al., 2012), who proposed a generic and robust computational architecture for distributed machine learning. We believe it is possible to implement our algorithm using the parameter server software framework.

One of the difficulties of the parameter server architecture is that learning happens at the level of parameters, with a single set of parameters being maintained across the cluster. Since the data subsets on workers are disjoint, the learning on each worker tends to make the local copy of the parameters diverge from those on the parameter server and on other workers. As a result, frequent synchronisation with the parameter server is necessary to prevent stale parameters and gradients. As an example, in the DistBelief method (Dean et al., 2012), experiments were conducted where the communication with the master was performed after every iteration, which can significantly slow down the learning process. On the other hand, one of the interesting aspects of the posterior server is that it lifts learning from the level of parameters to the level of distributions over parameters. As a result each worker can maintain a distinct parameter set in the MCMC sampler and a distinct likelihood approximation (since the likelihoods on different workers are indeed different as they have different data subsets) without requiring frequent communication with the posterior server.

Algorithm 1 Posterior Server: Distributed Bayesian Learning via SNEP

for each compute node $i = 1, \ldots, n$ **asynchronously do**
  let $\gamma_i^{(1)}$ be the initial mean parameter of local likelihood approximation.
  let $\lambda_i^{\text{old}} := \lambda_i^{(1)} := \nabla A^*(\gamma_i^{(1)})$ be the initial natural parameter of local likelihood approximation.
  let $\theta_{-i} := \theta_0 + \sum_{j \neq i} \lambda_j^{(1)}$ be the initial natural parameter of cavity distribution
  let $\theta_i' := \theta_{-i} + \lambda_i^{(1)}$ be the initial auxiliary parameter.
  let $x_i^{(1)} \sim p_{\theta_{-i} + \lambda_i^{(1)}}$ be the initial state of MCMC sampler.
  for $t = 1, 2, \ldots$ until convergence **do**
    update local state via MCMC:

$$x_i^{(t+1)} \sim \mathcal{K}_i \left(\cdot \mid x_i^{(t)}; \theta_i' - \beta_i^{-1} \lambda_i^{(t)}, \beta_i^{-1} \right)$$

    update local likelihood approximation:

$$\gamma_i^{(t+1)} := \gamma_i^{(t)} + \epsilon_t \left(s(x_i^{(t+1)}) - \nabla A \left(\theta_{-i} + \lambda_i^{(t)} \right) \right)$$
$$\lambda_i^{(t+1)} := \nabla A^*(\gamma_i^{(t+1)})$$

    every N_{outer} iterations **do**: update auxiliary parameter:

$$\theta_i' := \theta_{-i} + \lambda_i^{(t)}$$

    every N_{sync} iterations **asynchronously do:** communicate with posterior server:
      let $\Delta_i := \lambda_i^{(t)} - \lambda_i^{\text{old}}$.
      update $\lambda_i^{\text{old}} := \lambda_i^{(t)}$.
      send Δ_i to posterior server.
      receive $\theta_{\text{posterior}}$ from posterior server.
      update $\theta_{-i} := \theta_{\text{posterior}} - \lambda_i^{\text{old}}$.
  end for
end for
for the posterior server **do**
  let $\theta_{\text{posterior}} := \theta_0 + \sum_{j=1}^{n} \lambda_i^{(1)}$ be the initial natural parameter of the posterior approximation.
  maintain a queue of messages from workers.
  for each message Δ_i received from some worker i **do**
    update $\theta_{\text{posterior}} := \theta_{\text{posterior}} + \Delta_i$.
    send $\theta_{\text{posterior}}$ to worker i.
  end for
end for

The only role of communication here is for the cavity distributions, which can be thought of as a way for the system to focus the learning happening on the workers on the relevant regions of the parameter space (Xu et al., 2014). Empirically, the precise parameterisation of the cavity distribution is not very important. For an extreme example, suppose both the prior and likelihood terms are close to being Gaussians and the tractable family is also Gaussian. Then the likelihood approximation will not in fact depend on the cavity distribution at all, and will converge to the true likelihood on each worker independently. See also Zhang et al. (2015a) for elastic-averaging SGD, a similar idea of allowing each worker a separate parameter vector, using an ADMM-like methodology.

6. Experiments

In this section we present our experiments on SNEP and the posterior server. Our implementation, which is available at `http://bigbayes.github.com/PosteriorServer`, is in the Julia[3] technical computing language. In our setup, the master and workers are run on separate system processes (that is, in separate memory spaces) on a many-core computing cluster, making use of the high-level parallel programming abstractions provided by Julia. Other architectures are possible; for example, multiple threads sharing an address space, multiple GPUs, or nodes communicating over a network using MPI. For the MCMC sampler on workers, we chose an adaptive version of stochastic gradient Langevin dynamics (SGLD) (Welling and Teh, 2011) related to Adam (Kingma and Ba, 2015), which is more computationally scalable to larger neural network models and data sets than standard MCMC (see Appendix B for details). Our neural network models are implemented in the Mocha[4] deep learning Julia package.

6.1 Comparison to SMS on Bayesian Logistic Regression

In this section we compare SNEP against *Sampling via Moment Sharing* (SMS) (Xu et al., 2014), a related algorithm for distributed Bayesian learning whereby each worker has a separate MCMC sampler and coordination across workers is achieved using plain EP. SMS was originally proposed to scale up MCMC methods and as such it assumes that MCMC chains can be run for many iterations to convergence in between communications with the master. SMS uses the MCMC iterates to estimate the moments required for EP updates and so, as discussed in the introduction, requires a large number of iterations to produce the low Monte Carlo noise for EP updates to work properly.

We illustrate below the differing dynamics of SMS and SNEP when applied to a Bayesian logistic regression model with simulated data. We took a similar setup as in the SMS paper (Xu et al., 2014), generating a dataset $\mathcal{D} = \{(z_c, y_c)\}_{c=1}^{N}$ where covariates $z_c \in \mathbb{R}^d$ and response $y_c \in \{0, 1\}$. The conditional distribution of y_c given z_c and weights x is

$$p(y_c = 1 \mid z_c, x) = \sigma(z_c^\top x) \tag{26}$$

where σ denotes the logistic function. We used a Gaussian prior $p_0(x) = \mathcal{N}(x; 0_d, 10 I_d)$ on x and the aim is to construct an approximation to the posterior $p(x \mid \mathcal{D})$. We generated

3. `http://julialang.org`.
4. `https://github.com/pluskid/Mocha.jl`.

$N = 50000$ data points with $d = 50$ using iid draws for the covariates, $z_c \sim \mathcal{N}(\mu, \Sigma)$ where $\mu \in [0, 1]^d$ and $\Sigma = PP^\top$ with $P \in [-1, 1]^{d \times d}$, with entries drawn uniformly at random for both P and μ. The generating weight vector x^* is drawn from the prior $\mathcal{N}(x; 0_d, 10I_d)$. The labels y_c are then sampled according to the model.

Both algorithms were run with three workers each with one third of the data. SNEP is run with 1 inner loop iteration per synchronisation with the master (for the purpose of comparing against SMS). Varying numbers of MCMC samples per inner loop iteration were used for both algorithms, to investigate the effect of Monte Carlo noise on the performances of the algorithms (low number of samples meaning high noise and both lower performance and lower computational cost). The damping for SMS and the learning rate for SNEP were tuned for best performances. As the base exponential family, we used a full-covariance Gaussian. We compared the predictive RMSE $\sqrt{\sum_c |\hat{p}_c - y_c|^2 / N}$ obtained with both methods over time where $\hat{p}_c = \sigma(z_c^T \overline{x})$ where $\overline{x}$ is the currently estimated posterior mean. We also compared the relative difference between the estimated posterior mean and that estimated from a long run of the No-U-Turn sampler (Hoffman and Gelman, 2014) as implemented in Stan (Carpenter et al., 2017) with 4 chains and 50000 iterations.

Figure 2 illustrates the performances of both algorithms and how they are influenced by the number of MCMC samples used per iteration. It can be observed that SNEP is more robust and better performing than SMS in the presence of noise. In general, we found that the number of samples needed for SMS to perform well grows with the dimensionality of the problem. In models with very high dimensionality and multi-modality (e.g., neural networks), the number of samples needed per step is very large leading to iterations that are computationally impractical, whereas SNEP can afford to use many fewer samples.

6.2 Bayesian Neural Networks

In this section we report experimental results applying SNEP and the posterior server to distributed Bayesian learning of neural networks. We have found that the SMS algorithm exhibited significant instabilities and was not suitable for these larger scale problems. Instead our aim here is to explore the behaviour of SNEP when varying various key hyperparameters. We used diagonal covariance Gaussians as the approximating exponential family due to computational cost considerations. While a diagonal Gaussian approximation can be quite poor for the full Bayesian posterior, past works have shown that they can produce good predictive performances (Hernandez-Lobato and Adams, 2015, Blundell et al., 2015, Kirkpatrick et al., 2017). Hence our aim is to evaluate the predictive performance of SNEP, treated as a distributed learning algorithm for neural networks, and comparing it to Adam (Kingma and Ba, 2015), a state-of-the-art stochastic gradient descent (SGD) algorithm with access to the whole data set on a single computer, as well as several state-of-the-art distributed SGD algorithms: asynchronous SGD (A-SGD) (Dean et al., 2012) and elastic averaging SGD (EASGD) (Zhang et al., 2015a).

In our experiments we used stochastic gradient Langevin dynamics with an preconditioning scheme reminiscent of Adam as the MCMC sampler. The preconditioning scheme is the same as the one proposed by (Li et al., 2016) except for the addition of a debiasing reminiscent of Adam. We found that sometimes this adaptive scheme lead to large injected noise, especially at the start of training which was detrimental to learning. To mitigate

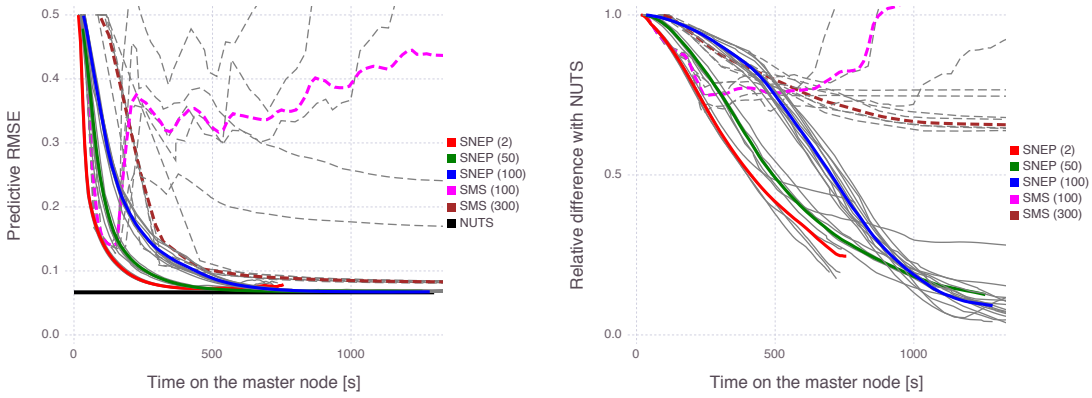


Figure 2: (left) Comparison of the predictive RMSE obtained with SMS (dashed-lines) and SNEP (full lines) versus the time on the master node when varying the number of MCMC samples per iteration (numbers reported in the legend). When too few samples are used for the moment estimates, SMS becomes unstable whilst SNEP does not suffer from this. SNEP is also consistently faster and more accurate than SMS. The accuracy obtained with the posterior mean estimated with a long run of Stan/NUTS is also displayed as comparison (horizontal line). (right) Relative difference between the posterior means estimated using Stan/NUTS and using SMS (dashed lines) or SNEP (full lines). Each coloured line corresponds to an average over several runs (represented in light grey).

this effect we limited the standard deviation of the injected noise to be at most ϵ in our experiments. For further details see Appendix B.

6.2.1 MNIST, Two Layer Fully Connected Network

In this section we look at training a fully connected neural network on the MNIST data set[5], which consists of 60000 training images of handwritten digits of size 28×28 and 10000 test images, the task being to classify each image into one of ten classes. The network has two hidden layers with 500 and 300 rectified linear units (ReLUs) respectively and softmax output units.

In the first set of experiments, we varied a number of hyperparameters of the learning regime while keeping the rest kept at default values, to investigate the sensitivity of the learning to these hyperparameters. The default values were chosen by hand in a rough initial round of experimentation as follows: the minibatch size is 100, the learning rate for SNEP was 0.02 and step size for the adaptive SGLD sampler was 0.001, the number of inner loop iterations per communication with master is $N_{sync} = 10$, the number of inner loop iterations per outer loop update is $N_{outer} = 10$, and the parameters were initialised with the popular Xavier initialisation (Glorot and Bengio, 2010). We set a minimum variance of 0.01 for the likelihood approximations, as we found that the estimated variances are otherwise too small due to the diagonal Gaussian approximation. Unless otherwise stated, we used four workers to explore the distributed behaviour. Test curves were produced by evaluating

5. `http://yann.lecun.com/exdb/mnist/`.

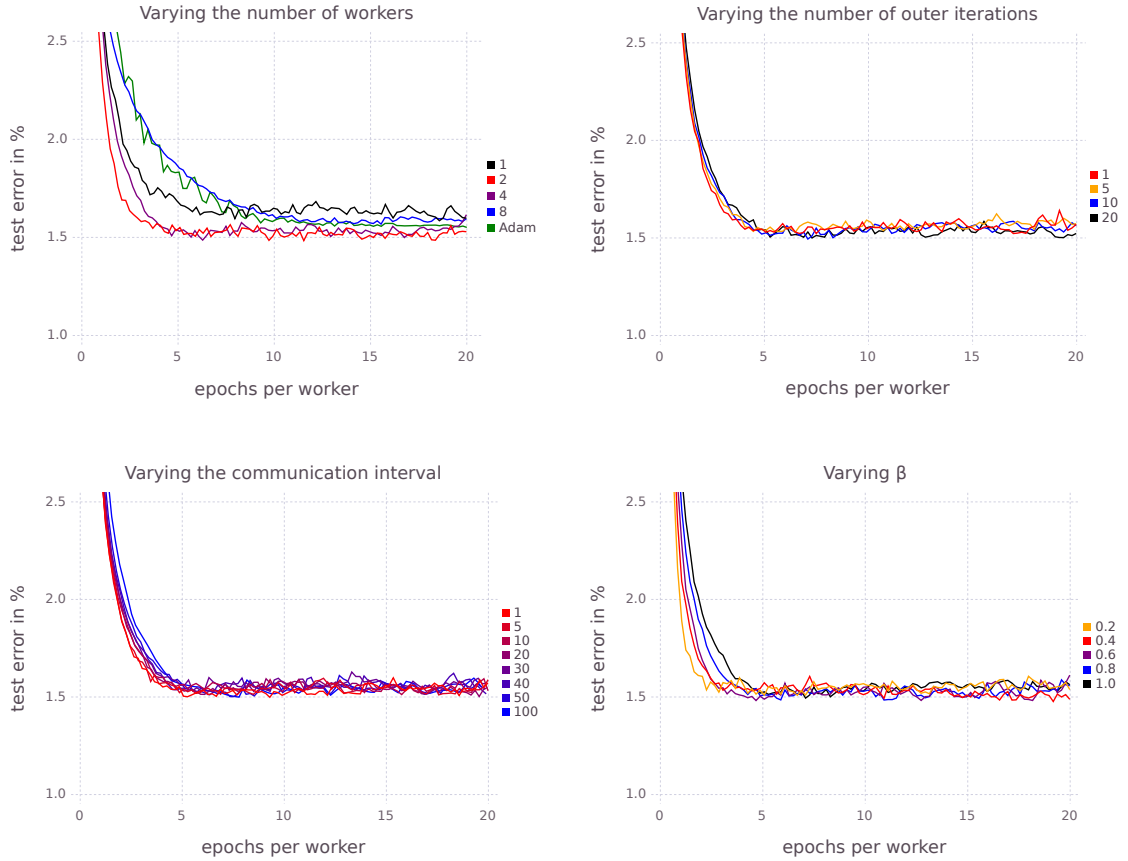


Figure 3: Results on a two-layer densely connected model on MNIST varying the parameters of SNEP and comparing against SGD, A-SGD, and EASGD. Four workers were used in the distributed methods in the experiments that held the number of workers fixed. This demonstrates that SNEP is insensitive to the full convergence of the outer loop, is robust to large communication intervals, that convergence is faster for a lower β (on this data set), and that SNEP produces competitive results quickly. Detailed analyses of sub-figures (left to right, top to bottom) are reported in the main text.

networks based on the approximate posterior means at the master. Each reported curve in the figures is an average over ten repetitions. In our experiments we used one CPU core per worker process. See Figure 3 for the results of this set of experiments.

First, we examined the behaviour of SNEP as N_{worker} is varied. We see that two workers perform significantly better than one worker or Adam. However, performance deteriorates as the number of workers is increased further. This is possibly due to smaller number of data items per worker which makes the variational approximation more severe. Another factor we found is that because we imposed a minimum variance in for the likelihood approximations, using more workers gives tighter cavity distributions which can affect the scale of the natural-gradients and slow convergence significantly.

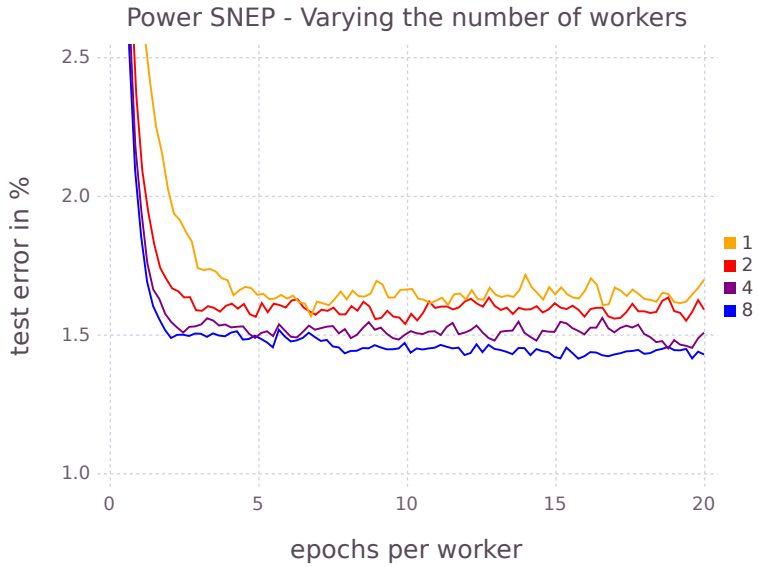


Figure 4: Results on a two-layer densely connected model on MNIST for power SNEP ($\beta = 1/N_{workers}$) varying the number of workers. There is a clear improvement in terms of convergence speed and final accuracy for more workers.

Second, we investigated the effect of varying N_{sync} and N_{outer}. Figure 3 demonstrates that SNEP is insensitive to these hyperparameters over reasonably large range of values. Note in particular that infrequent communications with the posterior server (up to once every 100 iterations in this experiment) did not significantly deteriorate the learning process at all. In fact, the related SMS algorithm (Xu et al., 2014) effectively involves communications with the master once every thousands of iterations.

Third, we investigated the effect of varying the β parameter. Interestingly, lower values of β sped up convergence significantly. The literature on power EP links the β parameter with locally minimising $\alpha-$divergences (where $\beta = 1/\alpha$). Lower values of β correspond to local approximations that are likely to capture more of the full posterior rather than matching one mode (Minka, 2005). Another factor is that given the minimum variance setting, lower values of β increase the variance of the cavity distribution allowing for more local exploration. Generally, In our experiments we found that a heuristic of setting $\beta = 1/N_{workers}$ worked particularly well for MNIST. We call the resulting algorithm power SNEP (p-SNEP). With this parameter setting we see a clear advantage for more workers (see figure 4). More workers give better results in terms of the test error and convergence is fastest for eight workers. Note that eight workers reach a test error of 1.5% after about two epochs, which is much faster than our experiments with $\beta = 1.0$.

Finally, we compared SNEP with two distributed SGD algorithms, asynchronous SGD (A-SGD, also known as Downpour) (Dean et al., 2012) and elastic averaging SGD (EASGD) (Zhang et al., 2015a). Our implementations differ slightly from those given in the papers. For A-SGD, both the workers and master performed Adam updates with independently

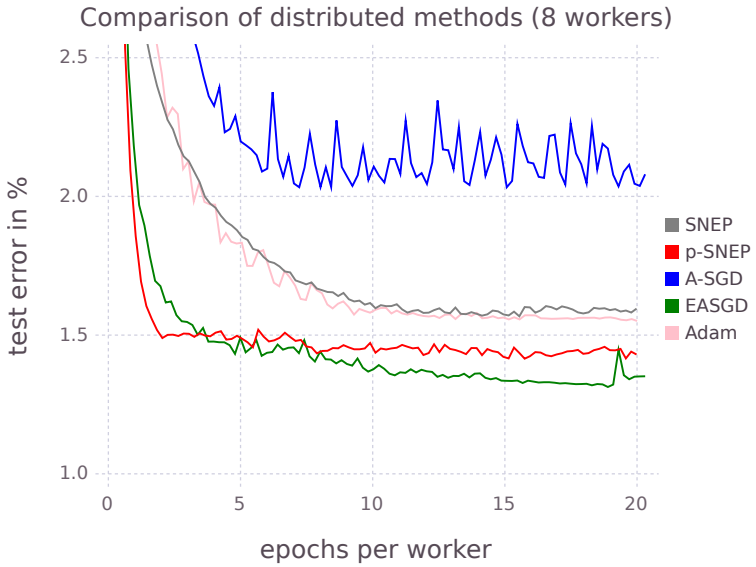


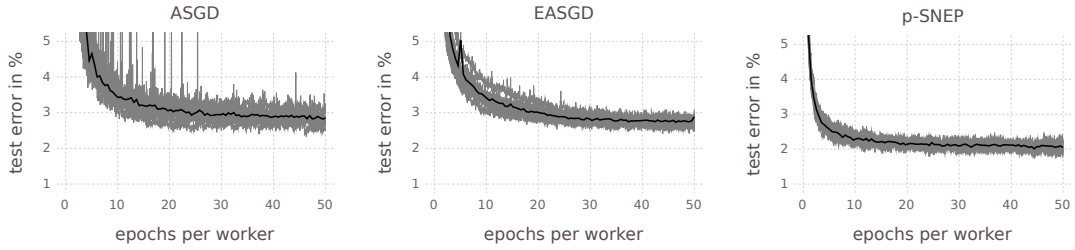
Figure 5: Results on a two-layer densely connected model on MNIST comparing SNEP and p-SNEP to asychronous SGD and elastic averaging SGD.

chosen step size constants. The authors of EASGD do not address the case of distributing the data among the workers. Nonetheless, we found this posed no problem to the algorithm. All hyperparameters were tuned to optimise performances. We used the same length of communication intervals $N_{sync} = 10$ and four workers for all algorithms. As shown in figure 5, SNEP (with $\beta = 1$) and p-SNEP (with $\beta = 1/N_{workers}$) achieve performances comparable to these state-of-the-art methods, both in terms of speed and final test accuracies. SNEP is comparable to Adam and better than A-SGD. p-SNEP outperforms A-SGD, Adam and SNEP, and is slightly faster initially than EASGD but achieves slightly higher test errors. It is important to note that we are tackling a harder problem with SNEP/p-SNEP, that of Bayesian learning rather than optimisation.

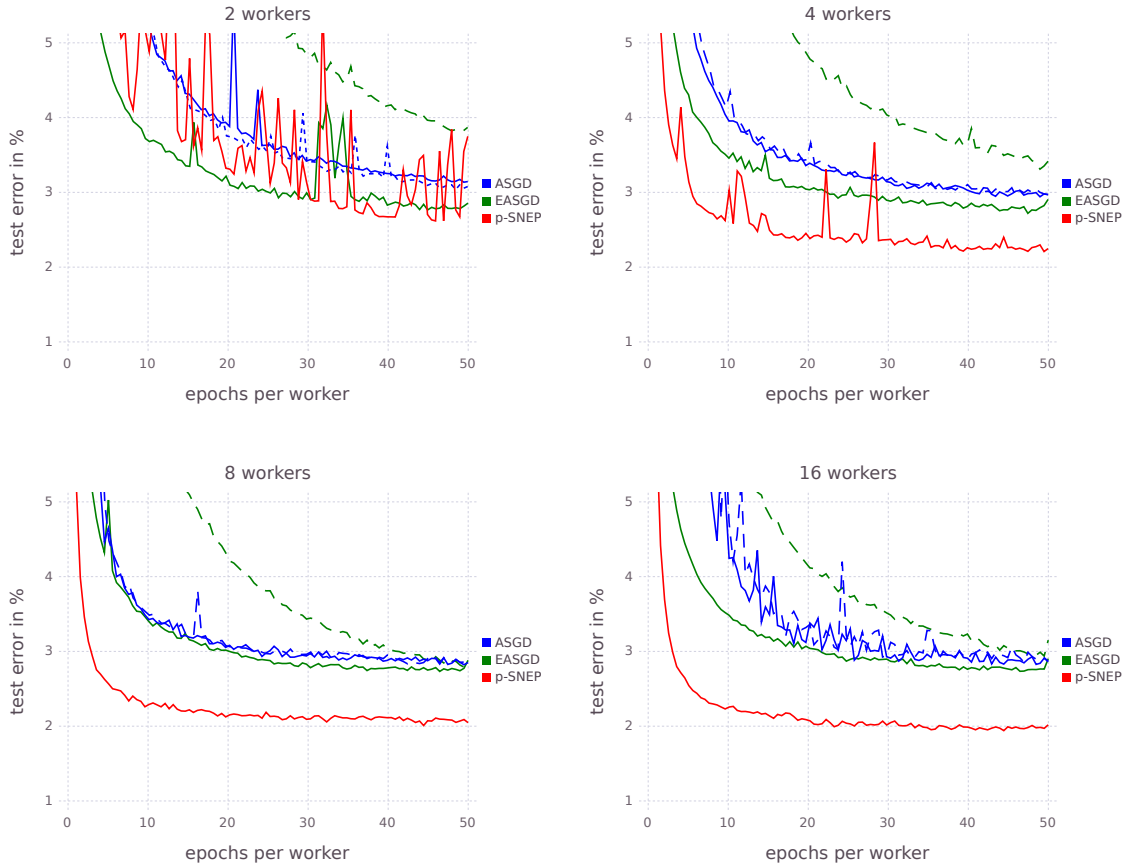
6.2.2 MNIST, Very Deep Fully Connected Network

Deeper models pose a much harder learning problem—as the depth increases, the number of saddle points increases exponentially (Dauphin et al., 2014). Moreover, learning often gets stuck not at critical points but in large plateaus of the error surface (Lipton, 2016). It is, therefore, interesting to evaluate (p-)SNEP against standard methods on a deeper fully-connected network, and we chose the architecture from Neelakantan et al. (2016), which has 20 hidden layers of 50 units each.

The experimental setup was similar to the previous section. The distributed methods used a communication intervals of $N_{sync} = 10$ and a varying number of workers. For p-SNEP, we optimized the prior variance, scaling of the injected noise, learning rate, step size, and beta hyperparameters by a course grid search over five orders of magnitude.



(a)



(b)

Figure 6: Results on deep narrow model for MNIST. (a) Comparing distributed methods for 8 workers. p-SNEP attains an extra percent of accuracy with less variability in the runs. The solid line is the average over 10 runs. (b) Varying the number of workers, averaged over 10 runs. The dashed line indicates result with no prelearning phase for A-SGD and EASGD, whereas for those methods the solid line indicates an Adam pre-learning phase of a third of an epoch.

Similarly, for A-SGD we optimized the Adam step size constants, separately on the workers and master, for EASGD the moving rate and Adam step size constants, and for SGD the Adam step size constants and scaling of injected noise, from no noise to three orders of magnitude. All methods were initialized by the Xavier method, and run with and without a non-distributed pre-learning phase of a third of an epoch. We found that pre-learning was essential for EASGD, while it was unnecessary for A-SGD and p-SNEP. In fact, for SNEP, too long a pre-learning phase is actually harmful to the final solution reached, which we suggest is due to the reduced exploration between workers. We only report results for p-SNEP without pre-learning (as per the previous section).

The same Adam step size constant was found to be optimal across all methods, 0.00065. For p-SNEP, the prior variance was set to 0.05, β to 1/8, and the learning rate to 0.04. For EASGD, the moving rate was set to 0.25. For A-SGD, the worker step sizes were set to 0.00065 and the master step size to a third of this; it must be slower to account for the staleness of the parameters. For SGD, we discovered that a small amount of noise, the same constant as is optimal for p-SNEP, reduces the variance of the runs around their mean by helping some runs avoid getting stuck in bad solutions. However, SGD did not converge to within the plot limits within the time set for experiments and so is omitted in the results shown in Figure 6a.

p-SNEP was found to significantly outperform A-SGD and EASGD, obtaining an extra percent of accuracy relative to other methods after fifty epochs for 8 and 16 workers. From the perspective of traditional neural network learning, our algorithm has two advantages for learning deep models: the addition of noise, and a principled method of regularizing the parameters. As in Neelakantan et al. (2016), we found it was important to add noise to the SGD methods. Although with a good initialization, added noise only improved the worst runs and not the best, smoothing out convergence. We conjecture that the added noise helps learning escape the difficult saddle points and plateaus. For this model, a strong prior variance was found to be important for p-SNEP to extract the full capacity of the model. We conjecture that this allows the gradients to flow back through the network more easily by forcing the parameters of the top layers to be small. Surprisingly, we found L2 regularization did not help with A-SGD and EASGD. p-SNEP also clearly benefits from additional workers in this setting, in contrast to A-SGD and EASGD (see Figure 6b).

It is interesting to note that this model has about one sixth the parameters of the previous two layer model. Yet, due to the deeper architecture and ability to successfully navigate the difficult learning landscape, p-SNEP is able to obtain similar test accuracies.

6.2.3 CIFAR-10, Convolutional Network

We also experimented with distributed Bayesian learning of convolutional neural networks, applying these to the CIFAR-10 data set (Krizhevsky, 2009) which consists of 50000 training instances and 10000 test instances from 10 classes, each instance being a 32x32 colour natural image. The network used is the one described in Alex Krizhesky's CIFAR tutorial[6] and consists of 8 layers: a first convolutional layer followed by max-pooling and local response normalization, a second convolutional layer also followed by max-pooling and local response normalization, and a third convolutional layer followed by a fully connected layer.

6. `https://code.google.com/p/cuda-convnet/wiki/Methodology`

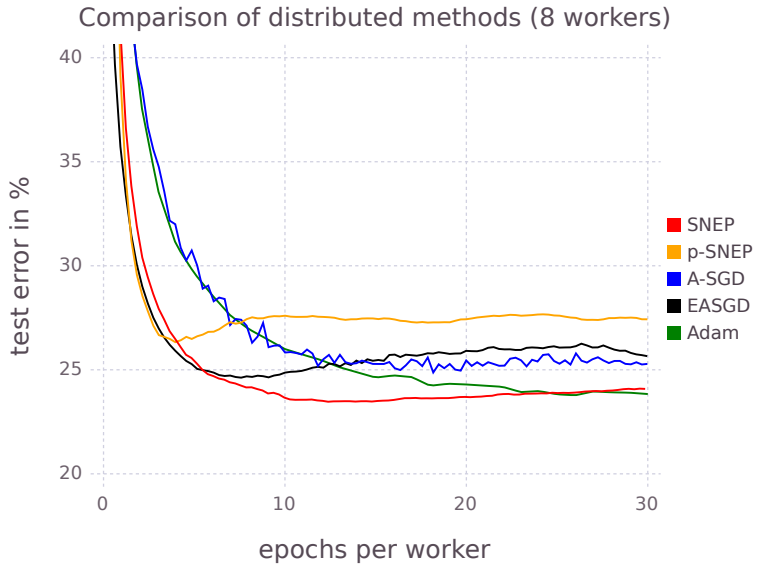


Figure 7: Learning curves varying the number of workers (averaged over three runs). The green line shows Adam with default parameters for comparison. Best viewed in colour.

For SNEP, we used the following settings: $N_{sync} = 10$, $N_{outer} = 10$, mini-batch size 100, weights initialised with the Xavier initialisation, and SNEP learning rate and adaptive SGLD step sizes of 0.02. We did not investigate improving performance by building in invariances using data perturbations or having multiple learning phases with different learning rates. Learning curves averaged over 3 runs are shown in Figure 7. All distributed algorithms are shown with hand-tuned optimal hyperparameter settings. On this data set SNEP outperforms other distributed algorithms. It converges as quickly as EASGD and p-SNEP but reaches a better final test accuracy. p-SNEP converges very fast but to a worse local optimum. It is not clear why p-SNEP performed better on previous experiments but not here; understanding the optimal choice of β is an interesting area for future research as we have observed that this parameter has a strong influence on the performance of the algorithm.

7. Discussion

In this paper, we have proposed a novel alternative to expectation propagation called stochastic natural-gradient expectation propagation (SNEP). SNEP is demonstrably convergent, even when using Monte Carlo estimates of the moments/mean parameters of tilted distributions. Experimentally, we find that SNEP converges more efficiently and more stably than other methods considered, particularly when Monte Carlo noise is high. Using SNEP, we have proposed the posterior server architecture for distributed Bayesian learning using an asynchronous non-blocking message-passing protocol. The architecture uses a

separate MCMC sampler on each worker, and SNEP to coordinate the samplers across the cluster so that the target distributions agree on the moments which characterise the base exponential family. In contrast with typical maximum likelihood parameter server architectures, the posterior server allows each worker to learn separate variational parameters, and as a result requires less frequent synchronisation across the cluster. We believe that this insight can allow for significant advances to distributed learning, although more work is still needed to make this reality. Finally, we applied SNEP and the posterior server to distributed Bayesian learning of both fully-connected and convolutional neural networks, where we showed performances on par with or better than state-of-the-art distributed and non-distributed optimisation algorithms. In conclusion, we have demonstrated that Bayesian learning of large complex models can be achieved efficiently and effectively in a distributing computing setting, and can achieve state-of-the-art performances on learning neural networks.

Our work leaves open a number of interesting avenues of future research, which we have discussed throughout the paper. It would be interesting to develop other novel convergent alternatives to EP with potentially faster convergence (e.g. using a single loop instead of double loop algorithm), and to apply such methods to other settings where Monte Carlo estimates are used within EP. It would also be interesting to further investigate combining SNEP with ideas from other projects such as the parameter tying idea of stochastic and averaged EP. We also believe that further explorations of learning regimes and hyperparameters, as well as of larger data sets, is called for, and will demonstrate further improvements to the method. These explorations can include using samples obtained at workers to form predictive probabilities, evaluating the algorithms using test log probabilities and on their abilities to quantify uncertainties, understanding the role of the β parameter, and combining SNEP with simulated annealing for optimisation. Another interesting area for application is on-device machine learning which falls naturally into our distributed data set-up. It would be interesting to compare SNEP to existing approaches such as federated learning (McMahan et al., 2016) We have also noted that our application of SNEP to neural network learning necessitates a diagonal covariance Gaussian approximation for computational reasons. Further work can consider richer posterior approximations, such as block-diagonal covariances or the use of matrix variate Gaussian distributions (Louizos and Welling, 2016). Ultimately, we believe these explorations will demonstrate the utility of a distributed Bayesian approach to learning.

Acknowledgments

LH is funded by the EPSRC OxWaSP CDT through grant EP/L016710/1. SW gratefully acknowledges support from the EPSRC AIMS CDT through grant EP/L015987/2. TL gratefully acknowledges funding from the Scatcherd European scholarship and EPSRC Grant EP/L505031/1. SJV thanks EPSRC for funding through EPSRC Grants EP/N000188/1 and EP/K009850/1. YWT gratefully acknowledges EPSRC for research funding through grant EP/K009362/1, and the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no.

617071. The authors thank Daniel Hernández-Lobato, Yingzhen Li, Thang Bui and Rich Turner for valuable feedback on the preprint and suggestions about tied EP factors.

Appendix A. Relationship of Ideal Variational Problem in the Extended Exponential Family to Variational Inference

As expected, the ideal variational problem in the extended exponential family in (3) is intractable and approximations are needed for tractability, with different approximations leading to different variational approaches. For readers unfamiliar with this framework, it might be illuminating to link the above framework to standard mean field variational inference, which corresponds to approximating the mean domain $\tilde{\mathcal{M}}$ by the set of moments achievable by some family of factorised distributions q over x. The standard evidence lower bound on the log marginal probability (also known as the variational free energy) is,

$$\begin{aligned} p(\{D_i\}_{i=1}^n) &\geq \mathbb{E}_q[\log p(x, \{D_i\}_{i=1}^n) - \log q(x)] \\ &= \mathbb{E}_q\left[\theta_0^\top s(x) - A(\theta_0) + \sum_{i=1}^n \ell_i(x)\right] - \mathbb{E}_q\left[\log q(x)\right] \\ &= \tilde{\theta}^\top \tilde{\mu} - A(\theta_0) - \mathbb{E}_q\left[\log q(x)\right] \end{aligned}$$

where we have introduced $\mu = \mathbb{E}_q[s(x)]$ and $\nu_i = \mathbb{E}_q[\ell_i(x)]$ as the moments of q, and we have used the definition of the extended natural and mean parameters $\tilde{\theta}$, $\tilde{\mu}$. The second term is a constant not dependent on q, while the third term above is the negative entropy, so that the above is equivalent to (3) except that the optimisation is only over a family of factorised distributions q.

Appendix B. Additional Techniques for Bayesian Neural Networks

In our experiments we investigated the use of SNEP and the posterior server for distributed Bayesian learning of neural networks. To get the method working well on the notoriously complicated posterior distributions for neural networks, a number of additional techniques are needed, which we describe here.

B.1 Adaptive Stochastic Gradient Langevin Dynamics

Most of the computational costs associated with the algorithm involve the MCMC updates to the state x_i. When the number of data points stored on each compute node is large, standard MCMC updates are infeasible as each update requires computations involving every data point. In our experiments we used the stochastic gradient Langevin dynamics (SGLD) algorithm proposed by Welling and Teh (2011) which scales well to large data sets.

SGLD uses mini-batches of data to provide unbiased estimates of gradients which are used in a time discretized Langevin dynamics simulation whose stationary distribution is the desired tilted distribution (20). SGLD injects noise in every step. As the discretization stepsize decreases the noise due to the stochastic gradients is eventually dominated by the injected noise and hence neglibible. Likewise the discretization introduces errors which tend to zero as the discretization step sizes decreases to zero; see (Teh et al., 2015, Vollmer

et al., 2016). Recall that the data points on compute node i is $D_i = \{y_c\}_{c \in S_i}$, and the log likelihood is

$$\ell_i(x_i) = \sum_{c \in S_i} \log p(y_c \mid x_i).$$

Let $B^{(t)} \subset S_i$ be a mini-batch of data, chosen uniformly at random with fixed size. Each SGLD update is,

$$x_i^{(t+1)} = x_i^{(t)} + \kappa_t (M^{(t)})^{-1} \left(\nabla s(x_i^{(t)})^\top (\theta_i' - \beta_i^{-1} \lambda_i^{(t)}) + \beta_i^{-1} \frac{|S_i|}{|B^{(t)}|} \sum_{c \in B^{(t)}} \nabla \log p(y_c | x_i^{(t)}) \right) + \eta_t,$$
$$\eta_t \sim \mathcal{N}(0, 2\kappa_t (M^{(t)})^{-1}). \tag{27}$$

The term inside the parentheses is an unbiased estimate of the gradient of the log density of the tilted distribution (20), κ_t is the discretization step size, $M^{(t)}$ is (an adaptive) diagonal mass matrix, while η_t is an injected normally-distributed noise, which prevents SGLD from converging to a mode of the distribution and distinguishes it from stochastic gradient descent. See Welling and Teh (2011) for details.

In (27), $M^{(t)}$ is a mass matrix which effectively controls the length scale of updates to each dimension of x_i. It is well known that in neural networks the length scales of gradients differ significantly across different parameters and adaptation of learning rates specific to each parameter is crucial for successful deployment of stochastic gradient descent learning. We have found that this is the case for SGLD as well, and used an adaptation scheme for the mass matrix reminiscent of Adam (Kingma and Ba, 2015). The adaptation scheme is the same as the one proposed by (Li et al., 2016) except for the addition of a debiasing reminiscent of Adam. At iteration t let g_t be the stochastic gradient estimate in (27). We use a diagonal mass matrix M_t that is updated according to the following update equations:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t \odot g_t$$
$$\mathrm{diag}(M_t) = \frac{v_t}{1 - \beta^2},$$

where $\odot$ denotes the elementwise product. We used $\beta = 0.999$ in our experiments. The actual adaptive stepsize is then given by

$$\kappa_t = \frac{\epsilon}{\sqrt{M_t} + \delta},$$

where $\epsilon = 10^{-3}$ and $\delta = 10^{-8}$ is added for numerical stability. We found that sometimes this adaptive scheme lead to large injected noise, specially at the start of training which was detrimental to learning. To mitigate this effect we limited the standard deviation of injected noise to be at most ϵ in our experiments. An alternative solution to this problem could be the approach proposed by Lu et al. (2017).

B.2 Shifting MCMC States After Communication with Posterior Server

After each communication with the posterior server, the target distribution of the MCMC sampler on the worker, say i, is changed, because of θ_{-i} being updated in Step 16 of

Algorithm 1. Assuming that the MCMC sampler had previously converged, it will now not be so anymore because of this shift in the target distribution, and a number of burn-in iterations may be needed before the mean parameter estimates can be used for SNEP updates again.

For Gaussian base exponential families, we can shift the MCMC state along with the target distribution when θ_{-i} is updated in the following way. Suppose μ_i^{old}, Σ_i^{old}, μ_i^{new}, Σ_i^{new} are the means and covariances of the approximate Gaussian posterior before and after the update to θ_{-i} (with natural parameters $\lambda_i + \theta_{-i}^{\text{old}}$, $\lambda_i + \theta_{-i}^{\text{new}}$ respectively where λ_i is the current natural parameter of the Gaussian likelihood approximation). Suppose x_i^{old} is the MCMC state before the update. Then we shift the MCMC state as follows:

$$x_i^{\text{new}} = \mu_i^{\text{new}} + (\Sigma_i^{\text{new}})^{\frac{1}{2}} (\Sigma_i^{\text{old}})^{-\frac{1}{2}} (x_i^{\text{old}} - \mu_i^{\text{old}}). \tag{28}$$

The idea is that x_i^{new} should be at the same location relative to the new Gaussian approximation to the posterior as x_i^{old} is relative to the old Gaussian approximation. We have found that no burn-in is needed with this shift in the MCMC state after each communication.

References

A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2012.

S. Amari and H. Nagaoka. *Methods of Information Geometry*. American Mathematical Society, 2001.

S. Amari, A. Cichocki, and H. Yang. A new algorithm for blind signal separation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 757–763, 1996.

R. Bardenet, A. Doucet, and C. Holmes. Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach. In *International Conference on Machine Learning (ICML)*, 2014.

S. Barthelmé and N. Chopin. ABC-EP: Expectation propagation for likelihood-free Bayesian computation. In *International Conference on Machine Learning (ICML)*, 2011.

H. Battey, J. Fan, H. Liu, J. Lu, and Z. Zhu. Distributed estimation and inference with statistical guarantees. *ArXiv:1509.05457*, 2015.

M. J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Network. In *International Conference on Machine Learning (ICML)*, pages 1613–1622, 2015.

B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017.

Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2933–2941, 2014.

J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1223–1231, 2012.

G. Dehaene and S. Barthelmé. Expectation propagation in the large-data limit. *ArXiv:1503.08060*, 2015.

A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. New York: Springer-Verlag, May 2001.

S. M. A. Eslami, D. Tarlow, P. Kohli, and J. Winn. Just-in-time learning for fast and flexible inference. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

A. Gelman, A. Vehtari, P. Jylänki, T. Sivula, D. Tran, S. Sahai, P. Blomstedt, J. P. Cunningham, D. Schiminovich, and C. Robert. Expectation propagation as a way of life: A framework for Bayesian inference on partitioned data. ArXiv:1412.4869, 2014.

W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

A. Graves. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2348–2356, 2011.

N. Heess, D. Tarlow, and J. Winn. Learning to pass expectation propagation messages. In *Advances In Neural Information Processing Systems (NIPS)*, 2013.

R. Herbrich, T. Minka, and T. Graepel. Trueskill$^{\text{TM}}$: A Bayesian skill rating system. *Advances in Neural Information Processing Systems (NIPS)*, 19:569, 2007.

J. M. Hernandez-Lobato and Ryan Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *International Conference on Machine Learning (ICML)*, pages 1861–1869, 2015.

J. M. Hernandez-Lobato, Y. Li, M. Rowland, T. Bui, D. Hernandez-Lobato, and R. Turner. Black-Box Alpha Divergence Minimization. In *International Conference on Machine Learning (ICML)*, pages 1511–1520, 2016.

T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 18, 2002.

M. Hoffman, D. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013.

M. D. Hoffman and A. Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15, 2014.

Z. Huang and A. Gelman. Sampling for Bayesian computation with large datasets. Technical report, Department of Statistics, Columbia University, 2005.

D. R. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 2004.

W. Jitkrittum, A. Gretton, N. Heess, S. M. Ali Eslami, B. Lakshminarayanan, D. Sejdinovic, and Z. Szabó. Kernel-based just-in-time learning for passing expectation propagation messages. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society B*, 76, 2014.

A. Korattikara, Y. Chen, and M. Welling. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *International Conference on Machine Learning (ICML)*, 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Leimkuhler, B. and Shang, X. Adaptive Thermostats for Noisy Gradient Systems. *SIAM Journal on Scientific Computing*, 38(2):A712–A736, 2016.

C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient Langevin dynamics for deep neural networks. In *AAAI Conference on Artificial Intelligence*, 2016.

Y. Li, J. M. Hernandez-Lobato, and R. E. Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

T. Lienart, Y. W. Teh, and A. Doucet. Expectation particle belief propagation. In *Advances In Neural Information Processing Systems (NIPS)*, 2015.

Z. C. Lipton. Stuck in a what? adventures in weight space. *arXiv preprint arXiv:1602.07320*, 2016.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1708–1716, 2016.

X. Lu, V. Perrone, L. Hasenclever, Y. W. Teh, and S. J. Vollmer. Relativistic Monte Carlo . In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1236–1245, 2017.

Y.-A. Ma, T. Chen, and E. B. Fox. A complete recipe for stochastic gradient MCMC. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

D. J. C. MacKay. Maximum likelihood and covariant algorithms for independent component analysis. http://www.inference.org.uk/mackay/abstracts/ica.html, 1999.

H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

T. Minka. Power EP. Technical report, Microsoft Research, 2004.

T. Minka. Divergence measures and message passing. Technical report, Microsoft Research Cambridge, January 2005. URL `https://www.microsoft.com/en-us/research/publication/divergence-measures-and-message-passing/`.

T. P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2001.

A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning (ICML)*, 2014.

R. M. Neal and G.E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1999.

A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2016.

W. Neiswanger, C. Wang, and E. P. Xing. Asymptotically exact, embarrassingly parallel MCMC. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

M. Opper and O. Winther. Gaussian Processes for Classification: Mean-Field Algorithms. *Neural Computation*, 12(11):2655–2684, 2000.

S. Patterson and Y. W. Teh. Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

G. Raskutti and S. Mukherjee. The information geometry of mirror descent. *IEEE Transactions on Information Theory*, 61:1451–1457, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning (ICML)*, 2014.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

S. L. Scott, A. W. Blocker, F. V. Bonassi, H. Chipman, E. George, and R. McCulloch. Bayes and big data: the consensus Monte Carlo algorithm. In *Bayes 250*, 2013.

Y. W. Teh and M. Welling. The unified propagation and scaling algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2002.

Y. W. Teh, A. H. Thiéry, and S. J. Vollmer. Consistency and fluctuations for stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 2015.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning (ICML)*, 2008.

S. J. Vollmer, K. C. Zygalakis, and Y. W. Teh. Exploration of the (non-)asymptotic bias and variance of stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 2016.

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

X. Wang and D. B. Dunson. Parallelizing MCMC via Weierstrass sampler, 2013. arXiv:1312.4605.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning (ICML)*, 2011.

W. Wiegerinck and T. Heskes. Fractional belief propagation. *Advances in Neural Information Processing Systems (NIPS)*, 2003.

M. Xu, B. Lakshminarayanan, Y. W. Teh, J. Zhu, and B. Zhang. Distributed Bayesian posterior sampling via moment sharing. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 689–695, 2001.

A.L. Yuille. CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.

S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. *Advances in Neural Information Processing Systems (NIPS)*, 2015a.

Y. Zhang, J. C. Duchi, and M. J. Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 2015b.

T. Zhao, G. Cheng, and H. Liu. A partially linear framework for massive heterogeneous data. *The Annals of Statistics*, 44(4):1400–1437, 08 2016.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server
Publication Status	X Published □Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Hasenclever, Leonard, Webb, Stefan, Lienart, Thibaut, Vollmer, Sebastian, L., Balaji, Blundell, Charles, Tom, and Teh, Yee Whye. Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server. *Journal of Machine Learning Research 18 (2017) 1-37*.

Student Confirmation

Student Name:	Stefan Webb		
Contribution to the Paper	Developed code framework for running experiments. Implemented non-Bayesian distributed baselines A-SGD and EASGD and compared SNEP to these, e.g. Figure 5. Solely responsible for conceiving and running the experiments on the deep model in 6.2.2 and writing this section. Some of the writing in 6.2.1. Evaluated a number of ideas for improvement of SNEP such as tying the parameters and other variants of the likelihood update.		
Signature		Date	12/12/2018

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof Yee Whye Teh		
Supervisor comments		
Signature	Date	

5

A Statistical Approach to Assessing Neural Network Robustness

A STATISTICAL APPROACH TO ASSESSING NEURAL NETWORK ROBUSTNESS

Stefan Webb*
Department of Engineering Science
University of Oxford

Tom Rainforth, Yee Whye Teh
Department of Statistics
University of Oxford

M. Pawan Kumar
Department of Engineering Science
University of Oxford,
Alan Turing Institute

ABSTRACT

We present a new approach to assessing the robustness of neural networks based on estimating the proportion of inputs for which a property is violated. Specifically, we estimate the probability of the event that the property is violated under an input model. Our approach critically varies from the formal verification framework in that when the property can be violated, it provides an informative notion of *how* robust the network is, rather than just the conventional assertion that the network is not verifiable. Furthermore, it provides an ability to scale to larger networks than formal verification approaches. Though the framework still provides a formal guarantee of satisfiability whenever it successfully finds one or more violations, these advantages do come at the cost of only providing a statistical estimate of unsatisfiability whenever no violation is found. Key to the practical success of our approach is an adaptation of multi-level splitting, a Monte Carlo approach for estimating the probability of rare events, to our statistical robustness framework. We demonstrate that our approach is able to emulate formal verification procedures on benchmark problems, while scaling to larger networks and providing reliable additional information in the form of accurate estimates of the violation probability.

1 INTRODUCTION

The robustness of deep neural networks must be guaranteed in mission-critical applications where their failure could have severe real-world implications. This motivates the study of neural network verification, in which one wishes to assert whether certain inputs in a given subdomain of the network might lead to important properties being violated (Zakrzewski, 2001; Bunel et al., 2018). For example, in a classification task, one might want to ensure that small perturbations of the inputs do not lead to incorrect class labels being predicted (Szegedy et al., 2013; Goodfellow et al., 2015).

The classic approach to such verification has focused on answering the binary question of whether there exist any counterexamples that violate the property of interest. We argue that this approach has two major drawbacks. Firstly, it provides no notion of how robust a network is whenever a counterexample can be found. Secondly, it creates a computational problem whenever no counterexamples exist, as formally verifying this can be very costly and does not currently scale to the size of networks used in many applications.

To give a demonstrative example, consider a neural network for classifying objects in the path of an autonomous vehicle. It will almost certainly be infeasible to train such a network that is perfectly robust to misclassification. Furthermore, because the network will most likely need to be of significant size to be effective, it is unlikely to be tractable to formally verify the network is perfectly robust, even if such a network exists. Despite this, it is still critically important to assess the robustness of the network, so that manufacturers can decide whether it is safe to deploy.

*Author for correspondence at info@stefanwebb.me

To address the shortfalls of the classic approach, we develop a new measure of intrinsic robustness of neural networks based on the *probability* that a property is violated under an input distribution model. Our measure is based on two key insights. The first is that for many, if not most, applications, full formal verification is neither necessary nor realistically achievable, such that one actually desires a notion of *how* robust a network is to a set of inputs, not just a binary answer as to whether it is robust or not. The second is that most practical applications have some acceptable level of risk, such that it is sufficient to show that the probability of a violation is below a certain threshold, rather than confirm that this probability is exactly zero.

By providing a probability of violation, our approach is able to address the needs of applications such as our autonomous vehicle example. If the network is not perfectly robust, it provides an explicit measure of exactly how robust the network is. If the network is perfectly robust, it is still able to tractability assert that a violation event is "probably-unsatisfiable". That is it is able to statistically conclude that the violation probability is below some tolerance threshold to true zero, even for large networks for which formal verification would not be possible.

Calculating the probability of violation is still itself a computationally challenging task, corresponding to estimating the value of an intractable integral. In particular, in most cases, violations of the target property constitute (potentially extremely) rare events. Consequently, the simple approach of constructing a direct Monte Carlo estimate by sampling from the input model and evaluating the property will be expensive and only viable when the event is relatively common. To address this, we adapt an algorithm from the Monte Carlo literature, adaptive multi-level splitting (AMLS) (Guyader et al., 2011; Nowozin, 2015), to our network verification setting. AMLS is explicitly designed for prediction of rare events and our adaptation means that we are able to reliably estimate the probability of violation, even when the true value is extremely small.

Our resulting framework is easy to implement, scales linearly in the cost of the forward operation of the neural network, and is agnostic both to the network architecture and input model. Assumptions such as piecewise linearity, Lipschitz continuity, or a specific network form are not required. Furthermore, it produces a diversity of samples which violate the property as a side-product. To summarize, our main contributions are:

- Reframing neural network verification as the estimation of the probability of a violation, thereby providing a more informative robustness metric for non-verifiable networks;
- Adaptation of the AMLS method to our verification framework to allow the tractable estimation of our metric for large networks and rare events;
- Validation of our approach on several models and datasets from the literature.

2 Related work

The literature on neural network robustness follows two main threads. In the optimization community, researchers seek to formally prove that a property holds for a neural network by framing it as a satisfiability problem (Zakrzewski, 2001), which we refer to as the classical approach to verification. Such methods have only been successfully scaled beyond one hidden layer networks for piecewise linear networks (Cheng et al., 2017; Katz et al., 2017), and even then these solutions do not scale to, for example, common image classification architectures with input dimensions in the hundreds, or apply to networks with nonlinear activation functions (Bunel et al., 2018). Other work has sought approximate solutions in the same general framework but still does not scale to larger networks (Pulina & Tacchella, 2010; Xiang et al., 2018; Huang et al., 2017b). As the problem is NP-hard (Katz et al., 2017), it is unlikely that an algorithm exists with runtime scaling polynomially in the number of network nodes.

In the deep learning community, research has focused on constructing and defending against adversarial attacks, and by estimating the robustness of networks to such attacks. Weng et al. (2018b) recently constructed a measure for robustness to adversarial attacks estimating a lower bound on the minimum adversarial distortion, that is the smallest perturbation required to create an adversarial example. Though the approach scales to large networks, the estimate of the lower bound is often demonstratively incorrect: it is often higher than an upper bound on the minimum adversarial distortion (Goodfellow, 2018). Other drawbacks of the method are that it cannot be applied to networks that are not Lipschitz continuous, it requires an expensive gradient computation for each class per sample, does not produce adversarial examples, and cannot be applied to non-adversarial proper-

ties. The minimum adversarial distortion is also itself a somewhat unsatisfying metric for many applications, as it conveys little information about the prevalence of adversarial examples.

In other work spanning both communities (Gehr et al., 2018; Weng et al., 2018a; Wong & Kolter, 2018), researchers have relaxed the satisfiability problem of classical verification, and are able to produce certificates-of-robustness for some samples (but not all that are robust) by giving a lower-bound on the minimal adversarial distortion. Despite these methods scaling beyond formal verification, we note that this is still a binary measure of robustness with limited informativeness.

3 MOTIVATING EXAMPLES

To help elucidate our problem setting, we consider the ACASXU dataset (Katz et al., 2017) from the formal verification literature. A neural network is trained to predict one of five correct steering decisions, such as "hard left," "soft left," etc., for an unmanned aircraft to avoid collision with a second aircraft. The inputs $\mathbf{x}$ describe the positions, orientations, velocities, etc. of the two aircraft. Ten interpretable properties are specified along with corresponding constraints on the inputs, for which violations correspond to events causing collisions. Each of these properties is encoded in a function, s, such that it is violated when $s(\mathbf{x}) \geq 0$. The formal verification problem asks the question, "Does there exist an input $\mathbf{x} \in \mathcal{E} \subseteq \mathcal{X}$ in a constrained subset, $\mathcal{E}$, of the domain such that the property is violated?" If there exists a counterexample violating the property, we say that the property is satisfiable (SAT), and otherwise, unsatisfiable (UNSAT).

Another example is provided by adversarial properties from the deep learning literature on datasets such as MNIST. Consider a neural network $f_\theta(\mathbf{x}) = \text{Softmax}(\mathbf{z}(\mathbf{x}))$ that classifies images, $\mathbf{x}$, into C classes, where the output of f gives the probability of each class. Let δ be a small perturbation in an l_p-ball of radius ϵ, that is, $\|\delta\|_p < \epsilon$. Then $\mathbf{x} = \mathbf{x}' + \delta$ is an adversarial example for $\mathbf{x}'$ if $\arg\max_i \mathbf{z}(\mathbf{x})_i \neq \arg\max_i \mathbf{z}(\mathbf{x}')_i$, i.e. the perturbation changes the prediction. Here, the property function is $s(\mathbf{x}) = \max_{i \neq c} (\mathbf{z}(\mathbf{x})_i - \mathbf{z}(\mathbf{x})_c)$, where $c = \arg\max_j \mathbf{z}(\mathbf{x}')_j$ and $s(\mathbf{x}) \geq 0$ indicates that $\mathbf{x}$ is an adversarial example. Our approach subsumes adversarial properties as a specific case.

4 ROBUSTNESS METRIC

The framework for our robustness metric is very general, requiring only a) a neural network f_θ, b) a property function $s(\mathbf{x}; f, \phi)$, and c) an input model $p(\mathbf{x})$. Together these define an integration problem, with the main practical challenge being the estimation of this integral. Consequently, the method can be used for any neural network. The only requirement is that we can evaluate the property function, which typically involves a forward pass of the neural network.

The property function, $s(\mathbf{x}; f_\theta, \phi)$, is a deterministic function of the input $\mathbf{x}$, the trained network f_θ, and problem specific parameters ϕ. For instance, in the MNIST example, $\phi = \arg\max_i f_\theta(\mathbf{x}')_i$ is the true output of the unperturbed input. Informally, the property reflects how badly the network is performing with respect to a particular property. More precisely, the event

$$E \triangleq \{s(\mathbf{x}; f_\theta, \phi) \geq 0\} \tag{1}$$

represents the property being violated. Predicting the occurrence of these, typically rare, events will be the focus of our work. We will omit the dependency on f_θ and ϕ from here on for notional conciseness, noting that these are assumed to be fixed and known for verification problems.

The input model, $p(\mathbf{x})$, is a distribution over the subset of the input domain that we are considering for counterexamples. For instance, for the MNIST example we could use $p(\mathbf{x}; \mathbf{x}') \propto \mathbb{1}(\|\mathbf{x} - \mathbf{x}'\|_p \leq \epsilon)$ to consider uniform perturbations to the input around an l_p-norm ball with radius ϵ. More generally, the input model can be used to place restrictions on the input domain and potentially also to reflect that certain violations might be more damaging than others.

Together, the property function and input model specify the probability of failure through the integral

$$\mathcal{I}[p, s] \triangleq P_{X \sim p(\cdot)}(s(X) \geq 0) = \int_{\mathcal{X}} \mathbb{1}_{\{s(\mathbf{x}) \geq 0\}} p(\mathbf{x}) d\mathbf{x}. \tag{2}$$

This integral forms our measure of robustness. The integral being equal to exactly zero corresponds to the classical notion of a formally verifiable network. Critically though, it also provides a measure for *how* robust a non-formally-verifiable network is.

5 METRIC ESTIMATION

Our primary goal is to estimate (2) in order to obtain a measure of robustness. Ideally, we also wish to generate example inputs which violate the property. Unfortunately, the event E is typically very rare in verification scenarios. Consequently, the estimating the integral directly using Monte Carlo,

$$\hat{P}_{X\sim p(\cdot)}\left(s(X)\geq 0\right)=\frac{1}{N}\sum\nolimits_{n=1}^{N}\mathbb{1}_{\{s(\mathbf{x}_n)\geq 0\}},\quad \text{where}\quad \mathbf{x}_n\overset{\text{i.i.d.}}{\sim}p(\cdot) \tag{3}$$

is typically not feasible for real problems, requiring an impractically large number of samples to achieve a reasonable accuracy. Even when E is not a rare event, we desire to estimate the probability using as few forward passes of the neural network as possible to reduce computation. Furthermore, the dimensionality of $\mathbf{x}$ is typically large for practical problems, such that it is essential to employ a method that scales well in dimensionality. Consequently many of the methods commonly employed for such problems, such as the cross-entropy method (Rubinstein, 1997; De Boer et al., 2005), are inappropriate due to relying on importance sampling, which is well known to scale poorly.

As we will demonstrate empirically, a less well known but highly effective method from the statistics literature, adaptive multi-level splitting (AMLS) (Kahn & Harris, 1951; Guyader et al., 2011), can be readily adapted to address all the aforementioned computational challenges. Specifically, AMLS is explicitly designed for estimating the probability of rare events and our adaptation is able to give highly accurate estimates even when the E is very rare. Furthermore, as will be explained later, AMLS also allows the use of MCMC transitions, meaning that our approach is able to scale effectively in the dimensionality of $\mathbf{x}$. A further desirable property of AMLS is that it produces property-violating examples as a side product, namely, it produces samples from the distribution

$$\pi(\mathbf{x})\triangleq p(\mathbf{x}\mid E)=p(\mathbf{x})\mathbb{1}_{\{s(\mathbf{x})\geq 0\}}/\mathcal{I}\left[p,s\right]. \tag{4}$$

Such samples could, in theory, be used to perform robust learning, in a similar spirit to Goodfellow et al. (2015) and Madry et al. (2017).

5.1 MULTI-LEVEL SPLITTING

Multi-level splitting (Kahn & Harris, 1951) divides the problem of predicting the probability of a rare event into several simpler ones. Specifically, we construct a sequence of intermediate targets,

$$\pi_k(\mathbf{x})\triangleq p(\mathbf{x}\mid\{s(\mathbf{x})\geq L_k\})\propto p(\mathbf{x})\mathbb{1}_{\{s(\mathbf{x})\geq L_k\}},\quad k=0,1,2,\ldots,K,$$

for levels, $-\infty=L_0<L_1<L_2<\cdots<L_K=0$, to bridge the gap between the input model $p(\mathbf{x})$ and the target $\pi(\mathbf{x})$. For any choice of the intermediate levels, we can now represent equation (2) through the following factorization,

$$\begin{aligned}P_{X\sim p(\cdot)}\left(s(X)\geq 0\right)&=\prod\nolimits_{k=1}^{K}P\left(s(X)\geq L_k\mid s(X)\geq L_{k-1}\right)=\prod\nolimits_{k=1}^{K}P_k,\\ &\text{where}\quad P_k\triangleq\mathbb{E}_{X\sim\pi_{k-1}(\cdot)}\left[\mathbb{1}_{\{s(X)\geq L_k\}}\right].\end{aligned} \tag{5}$$

Provided consecutive levels are sufficiently close, we will be able to reliably estimate each P_k by making use of the samples from one level to initialize the estimation of the next.

Our approach starts by first drawing N samples, $\{\mathbf{x}_n^{(0)}\}_{n=1}^N$, from $\pi_0(\cdot)=p(\cdot)$, noting that this can be done exactly because the perturbation model is known. These samples can then be used to estimate P_1 using simple Monte Carlo,

$$P_1\approx\hat{P}_1\triangleq\frac{1}{N}\sum\nolimits_{n=1}^{N}\mathbb{1}_{\{s(\mathbf{x}_n^{(0)})\geq L_1\}}\quad\text{where}\quad\mathbf{x}_n^{(0)}\sim\pi_0(\cdot).$$

In other words, P_1 is the fraction of these samples whose property is greater than L_1. Critically, by ensuring the value of L_1 is sufficiently small for $\{s(\mathbf{x}_n)\geq L_1\}$ to be a common event, we can ensure $\hat{P}_1$ is a reliable estimate for moderate numbers of samples N.

To estimate the other P_k, we need to be able to draw samples from $\pi_{k-1}(\cdot)$. For this we note that if $\{\mathbf{x}_n^{(k-2)}\}_{n=1}^N$ are distributed according to $\pi_{k-2}(\cdot)$, then the subset of these samples for which $s(\mathbf{x}_n^{(k-2)})\geq L_{k-1}$ are distributed according to $\pi_{k-1}(\cdot)$. Furthermore, setting L_{k-1} up to ensure this event is not rare means a significant proportion of the samples will satisfy this property.

Algorithm 1 Adaptive multi-level splitting with termination criterion

Input: Input model $p(\mathbf{x})$, sample quantile ρ, MH proposal $g(\mathbf{x}'|\mathbf{x})$, number of MH steps M, termination threshold $\log(P_{\text{min}})$
Sample $\{\mathbf{x}_n^{(0)}\}_{n=1}^N$ i.i.d. from $p(\cdot)$
Initialize $L \leftarrow -\infty, \quad L_{\text{prev}} \leftarrow -\infty, \quad \log(\mathcal{I}) \leftarrow 0, \quad k \leftarrow 0$
while $L < 0$ **do**
  $k \leftarrow k + 1$
  Evaluate and sort $\{s(\mathbf{x}_n^{(k-1)})\}_{n=1}^N$ in descending order
  $L_k \leftarrow \min\{0, s(\mathbf{x}_{\lfloor \rho N \rfloor}^{(k-1)})\}$ ▷ Updating the level
  $\hat{P}_k \leftarrow \#\{\mathbf{x}_n^{(k-1)} \mid s(\mathbf{x}_n^{(k-1)}) \geq L\} \,/\, N$
  $\log(\mathcal{I}) \leftarrow \log(\mathcal{I}) + \log(\hat{P}_k)$ ▷ Updating integral estimate
  if $\log(\mathcal{I}) < \log(P_{\text{min}})$ **then return** $(\emptyset, -\infty)$ **end if** ▷ Final estimate will be less than $\log(P_{\text{min}})$
  Initialize $\{\mathbf{x}_n^{(k)}\}_{n=1}^N$ by resampling with replacement N times from $\{\mathbf{x}_n^{(k-1)} \mid s(\mathbf{x}_n^{(k-1)}) \geq L\}$
  Apply M MH updates separately to each $\mathbf{x}_n^{(k)}$ using $g(\mathbf{x}'|\mathbf{x})$
  [Optional] Adapt $g(\mathbf{x}'|\mathbf{x})$ based on MH acceptance rates
end while
return $(\{\mathbf{x}_n^{(k)}\}_{n=1}^N, \log(\mathcal{I}))$

To avoid our set of samples shrinking from one level to the next, it is necessary to carry out a rejuvenation step to convert this smaller set of starting samples to a full set of size N for the next level. To do this, we first carry out a uniform resampling with replacement from the set of samples satisfying $s(\mathbf{x}_n^{(k-1)}) \geq L_k$ to generate a new set of N samples which are distributed according to $\pi_k(\cdot)$, but with a large number of duplicated samples. Starting with these samples, we then successively apply M Metropolis–Hastings (MH) transitions targeting $\pi_k(\cdot)$ separately to each sample to produce a fresh new set of samples $\{\mathbf{x}_n^{(k)}\}_{n=1}^N$ (see Appendix A for full details). These samples can then in turn be used to form a Monte Carlo estimate for P_k,

$$P_k \approx \hat{P}_k \triangleq \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}_{\{s(\mathbf{x}_n^{(k-1)}) \geq L_k\}} \quad \text{where} \quad \mathbf{x}_n^{(k-1)} \sim \pi_{k-1}(\cdot), \tag{6}$$

along with providing the initializations for the next level. Running more MH transitions decreases the correlations between the set of samples, improving the performance of the estimator.

We have thus far omitted to discuss how the levels L_k are set, other than asserting the need for the levels to be sufficiently close to allow reliable estimation of each P_k. Presuming that we are also free to choose the number of levels K, there is inevitably a trade-off between ensuring that each $\{s(X) \geq L_k\}$ is not rare given $\{s(X) \geq L_{k-1}\}$, and keeping the number of levels small to reduce computational costs and avoid the build-up of errors. AMLS (Guyader et al., 2011) builds on the basic multi-level splitting process, providing an elegant way of controlling this trade-off by adaptively selecting the level to be the minimum of 0 and some quantile of the property under the current samples. The approach terminates when the level reaches zero, such that $L_K = 0$ and K is a dynamic parameter chosen implicitly by the adaptive process.

Choosing the ρth quantile of the values of the property results in discarding a fraction $(1 - \rho)$ of the chains at each step of the algorithm. This allows explicit control of the rarity of the events to keep them at a manageable level. We note that if all the sample property values are distinct, then this approach gives $P_k = \rho$, $\forall k < K$. To give intuition to this, we can think about splitting up $\log(\mathcal{I})$ into chunks of size $\log(\rho)$. For any value of $\log(\mathcal{I})$, there is always a unique pair of values $\{K, P_K\}$ such that $\log(\mathcal{I}) = K \log(\rho) + \log(P_K)$, $K \geq 0$ and $P_K < \rho$. Therefore the problem of estimating $\mathcal{I}$ is equivalent to that of estimating K and P_K.

5.2 Termination Criterion

The application of AMLS to our verification problem presents a significant complicating factor in that the true probability of our rare event might be exactly zero. Whenever this is the case, the basic AMLS approach outlined in (Guyader et al., 2011) will never terminate as the quantile of the property will never rise above zero; the algorithm simply produces closer and closer intermediate levels as it waits for the event to occur.

To deal with this, we introduce a termination criterion based on the observation that AMLS's running estimate for $\mathcal{I}$ monotonically decreases during running. Namely, we introduce a threshold probability, P_{min}, below which the estimates will be treated as being numerically zero. We then terminate the algorithm if $\mathcal{I} < P_{\text{min}}$ and return $\mathcal{I} = 0$, safe in the knowledge that even if the algorithm would eventually generate a finite estimate for $\mathcal{I}$, this estimate is guaranteed to be less than P_{min}.

Putting everything together, gives the complete method as shown in Algorithm 1. See Appendix B for low-level implementation details.

6 EXPERIMENTS

6.1 EMULATION OF FORMAL VERIFICATION

In our first experiment, we aim to test whether our robustness estimation framework is able to effectively emulate formal verification approaches, while providing additional robustness information for SAT properties. In particular, we want to test whether it reliably identifies properties as being UNSAT, for which $\mathcal{I} = 0$, or SAT, for which $\mathcal{I} > 0$. We note that the method still provides a formal demonstration for SAT properties because having a non-zero estimate for $\mathcal{I}$ indicates that at least one counterexample has been found. Critically, it further provides a measure for how robust SAT properties are, through its estimate for $\mathcal{I}$.

We used the COLLISIONDETECTION dataset introduced in the formal verification literature by (Ehlers, 2017). It consists of a neural network with six inputs that has been trained to classify two car trajectories as colliding or non-colliding. The architecture has 40 linear nodes in the first layer, followed by a layer of max pooling, a ReLU layer with 19 hidden units, and an output layer with 2 hidden units. Along with the dataset, 500 properties are specified for verification, of which 172 are SAT and 328 UNSAT. This dataset was chosen because the model is small enough so that the properties can be formally verified. These formal verification methods do not calculate the value of $\mathcal{I}$, but rather confirm the existence of a counterexample for which $s(\mathbf{x}) > 0$.

We ran our approach on all 500 properties, setting $\rho = 0.1$, $N = 10^4$, $M = 1000$ (the choice of these hyperparameters will be justified in the next subsection), and using a uniform distribution over the input constraints as the perturbation model, along with a uniform random walk proposal. We compared our metric estimation approach against the naive Monte Carlo estimate using 10^{10} samples. The generated estimates of $\mathcal{I}$ for all SAT properties are shown in Figure 1a.

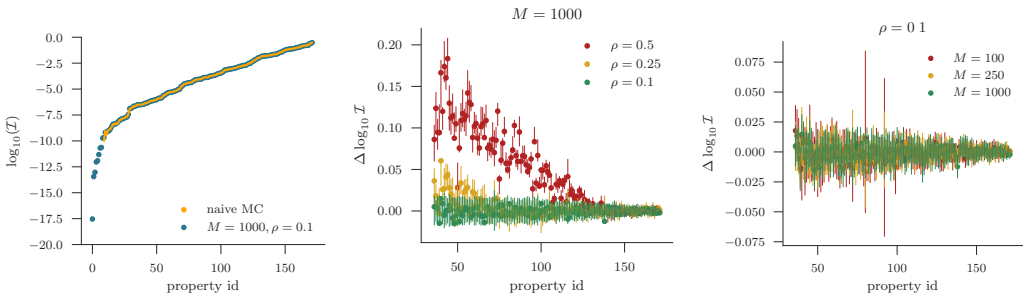
Both our approach and the naive MC baseline correctly identified all of the UNSAT properties by estimating $\mathcal{I}$ as exactly zero. However, despite using substantially more samples, naive MC failed to find a counterexample for 8 of the rarest SAT properties, thereby identifying them as UNSAT, whereas our approach found a counterexample for all the SAT properties. As shown in Figure 1a, the variances in the estimates for $\mathcal{I}$ of our approach were also very low and matched the unbiased MC baseline estimates for the more commonly violated properties, for which the latter approach still gives reliable, albeit less efficient, estimates. Along with the improved ability to predict rare events, our approach was also significantly faster than naive MC throughout, with a speed up of several orders of magnitude for properties where the event is not rare—a single run with naive MC took about 3 minutes, whereas a typical run of ours took around 3 seconds.

6.2 SENSITIVITY TO PARAMETER SETTINGS

As demonstrated by Bréhier et al. (2015), AMLS is unbiased under the assumption that perfect sampling from the targets, $\{\pi_k\}_{k=1}^{K-1}$, is possible, and that the cumulative distribution function of $s(X)$ is continuous. In practice, finite mixing rates of the Markov chains and the dependence between the initialization points for each target means that sampling is less than perfect, but improves with larger values of M and N. The variance, on the other hand, theoretically strictly decreases with larger values of N and ρ (Bréhier et al., 2015).

In practice, we found that while larger values of M and N were always beneficial, setting ρ too high introduced biases into the estimate, with $\rho = 0.1$ empirically providing a good trade-off between bias and variance. Furthermore, this provides faster run times than large values of ρ, noting that the smaller values of ρ lead to larger gaps in the levels.

To investigate the effect of the parameters more formally, we further ran AMLS on the SAT properties of COLLISIONDETECTION, varying $\rho \in \{0.1, 0.25, 0.5\}$, $N \in \{10^3, 10^4, 10^5\}$ and



(a) (b) (c)

Figure 1: (a) Estimate of $\mathcal{I}$ for all SAT properties of COLLISIONDETECTION problem. Error bars indicating $\pm$ three standard errors from 30 runs are included here and throughout, but the variance of the estimates was so small that these are barely visible. We can further conclude low bias of our method for the properties where naive MC estimation was feasible, due to the fact that naive MC produces unbiased (but potentially high variance) estimates. (b) Mean AMLS estimate relative to naive MC estimate for different ρ holding $M = 1000$ fixed, for those properties with $\log_{10} \mathcal{I} > -6.5$ such that they could be estimated accurately. The bias decreases both as ρ and the rareness of the event decrease. (c) As per (b) but with varying M and holding $\rho = 0.1$ fixed.

$M \in \{100, 250, 1000\}$, again comparing to the naive MC estimate for 10^{10} samples. We found that the value of N did not make a discernible difference in this range regardless of the values for ρ and M, and thus all presented results correspond to setting $N = 10^4$. As shown in Figure 1b, we found that the setting of ρ made a noticeable difference to the estimates for the relatively rarer events. All the same, these differences were small relative to the differences between properties. As shown in Figure 1c, the value of M made little difference when $\rho = 0.1$,. Interesting though, we found that the value of M was important for different values of ρ, as shown in Appendix C.1, with larger values of M giving better results as expected.

6.3 CONVERGENCE WITH HIGHER-DIMENSIONAL INPUTS AND LARGER NETWORKS

To validate the algorithm on a higher-dimensional problem, we first tested adversarial properties on the MNIST and CIFAR−10 datasets using a dense ReLU network with two hidden-layer of size 256. An l_∞-norm ball perturbation around the data point with width ϵ was used as the uniform input model, with $\epsilon = 1$ representing an l_∞-ball filling the entire space (the pixels are scaled to $[0, 1]$), together with a uniform random walk MH proposal. After training the classifiers, multilevel splitting was run on ten samples from the test set at multiple values of ϵ, with $N = 10000$ and $\rho = 0.1$, and $M \in \{100, 250, 1000\}$ for MNIST and $M \in \{100, 250, 500, 1000, 2000\}$ for CIFAR−10. The results for naive MC were also evaluated using 5×10^9 samples—less than the previous experiment as the larger network made estimation more expensive—in the cases where the event was not too rare. This took around twenty minutes per naive MC estimate, versus a few minutes for each AMLS estimate.

As the results were similar across datapoints, we present the result for a single example in the top two rows of Figure 2. As desired, a smooth curve is traced out as ϵ decreases, for which the event E becomes rarer. For MNIST, acceptable accuracy is obtained for $M = 250$ and high accuracy results for $M = 1000$. For CIFAR−10, which has about four times the input dimension of MNIST, larger values of M were required to achieve comparable accuracy. The magnitude of ϵ required to give a certain value of $\log(\mathcal{I})$ is smaller for CIFAR−10 than MNIST, reflecting that adversarial examples for the former are typically more perceptually similar to the datapoint.

To demonstrate that our approach can be employed on large networks, we tested adversarial properties on the CIFAR−100 dataset and a much larger DenseNet architecture (Huang et al., 2017a), with depth and growth-rate 40 (approximately 2×10^6 parameters). Due to the larger model size, we set $N = 300$, the largest minibatch that could be held in memory (a larger N could be used by looping over minibatches). The naive Monte Carlo estimates used 5×10^6 samples for about an hour of computation time per estimate, compared to between five to fifteen minutes for each AMLS estimate. The results are presented in the bottom row of Figure 2, showing that our algorithm agrees with the naive Monte Carlo estimate.

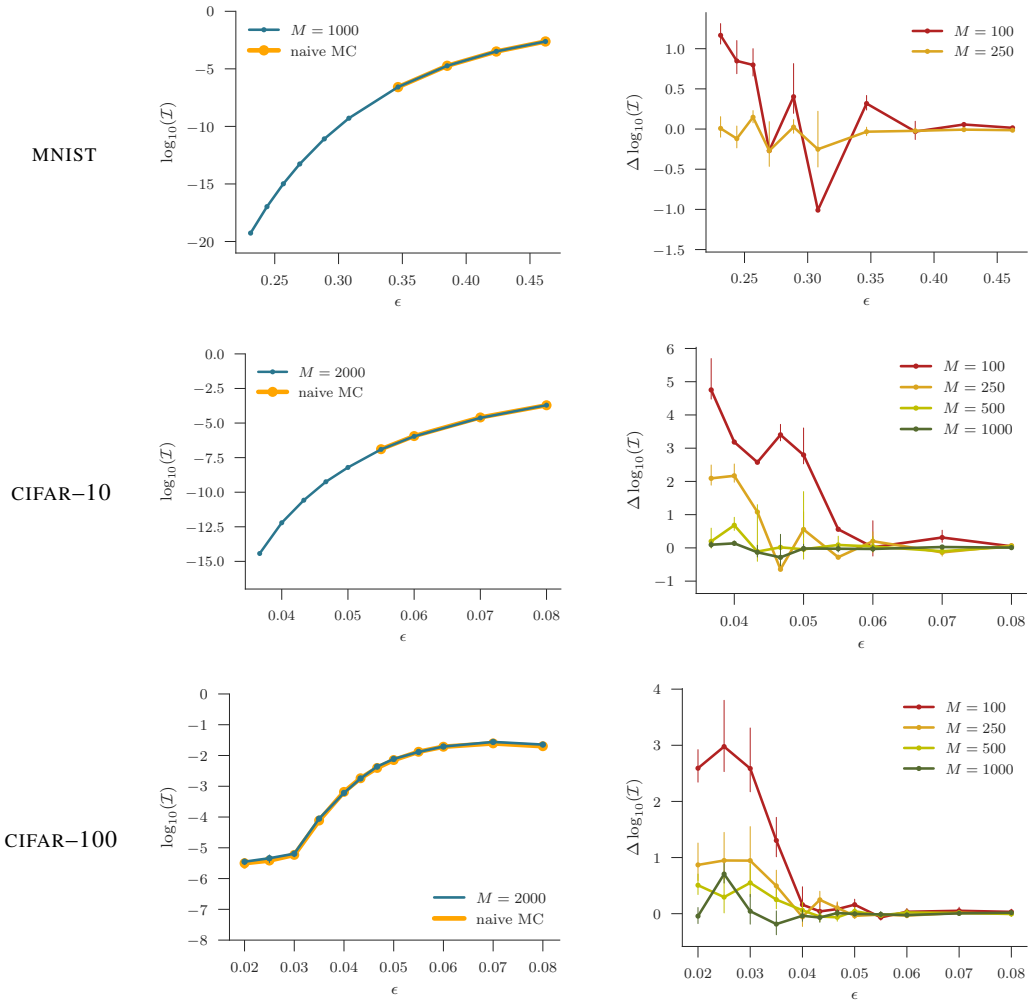
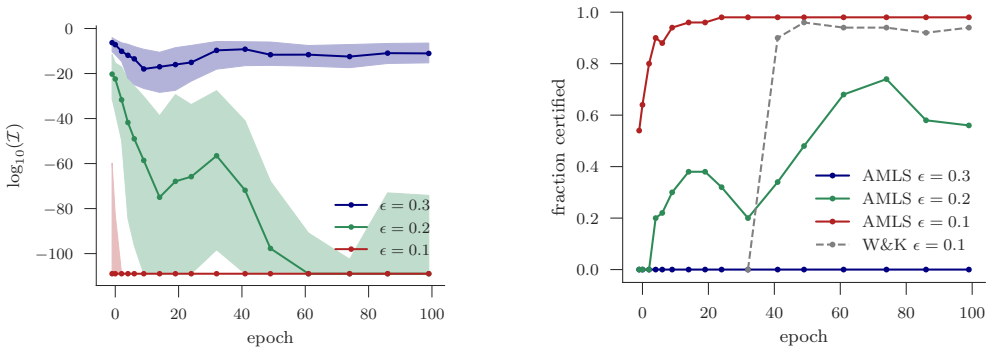


Figure 2: [Left] Estimates for $\mathcal{I}$ on adversarial properties of a single datapoint with $\rho = 0.1$, and $N \in \{10000, 10000, 300\}$ for MNIST/CIFAR–10/CIFAR–100 respectively. As in Figure 1, the error bars from 30 runs are barely visible, highlighting a very low variance in the estimates, while the close matching to the naive MC estimates when ϵ is large enough to make the latter viable, indicate a very low bias. For CIFAR–100 the error bars are shown for the naive estimates, as well, from 10 runs. [Right] The difference in the estimate for the other values of M from $M \in \{1000, 2000, 2000\}$ for MNIST/CIFAR–10/CIFAR–100, respectively. The estimate steadily converges as M increases, with larger M more important for rarer events.

6.4 Robustness of provable defenses during training

We now examine how our robustness metric varies for a ReLU network as that network is trained to be more robust against norm bounded perturbations to the inputs using the method of Wong & Kolter (2018). Roughly speaking, their method works by approximating the set of outputs resulting from perturbations to an input with a convex outer bound, and minimizing the worst case loss over this set. The motivation for this experiment is twofold. Firstly, this training provides a series of networks with ostensibly increasing robustness, allowing us to check if our approach produces robustness estimates consistent with this improvement. Secondly, it allows us to investigate whether the training to improve robustness for one type of adversarial attack helps to protect against others. Specifically, whether training for small perturbation sizes improves robustness to larger perturbations.

We train a CNN model on MNIST for 100 epochs with the standard cross-entropy loss, then train the network for a further 100 epochs using the robust loss of Wong & Kolter (2018), saving a snapshot of the model at each epoch. The architecture is the same as in (Wong & Kolter, 2018), containing



(a) Variation in $\mathcal{I}$ during robustness training

(b) Fraction of datapoints declared UNSAT

Figure 3: (a) Variation in $\mathcal{I}$ during the robustness training of on a CNN model for MNIST for three different perturbation sizes ϵ. Epoch 0 corresponds to the network after conventional training, with further epochs corresponding to iterations of robustness training. The solid line indicates the median over 50 datapoints, and the limits of the shaded regions the 25 and 75 percentiles. Our measure is capped at $P_{\text{min}} = \exp(-250)$. We see that while training improves robustness for $\epsilon = 0.2$, the initial network is already predominantly robust to perturbations of size $\epsilon = 0.1$, while the robustness to perturbations of size $\epsilon = 0.3$ actually starts to decrease after around 20 epochs. (b) Comparing the fraction of 50 datapoints for which Wong & Kolter (2018) produces a certificate-of-robustness for $\epsilon = 0.1$ ("W&K"), versus the fraction of those samples for which $\mathcal{I} = P_{\text{min}}$ for $\epsilon \in \{0.1, 0.2, 0.3\}$ ("AMLS"). Due to very heavy memory requirements, it was computationally infeasible to calculate certificates-of-robustness for $\epsilon = \{0.2, 0.3\}$, and $\epsilon = 0.1$ before epoch 32 with the method of Wong & Kolter (2018). Our metric, however, suffers no such memory issues.

two strided convolutional layers with 16 and 32 channels, followed by two fully connected layers with 100 and 10 hidden units, and ReLU activations throughout. The robustification phase trains the classifier to be robust in an l_∞ ϵ-ball around the inputs, where ϵ is annealed from 0.01 to 0.1 over the first 50 epochs. At a number of epochs during the robust training, we calculate our robustness metric with $\epsilon \in \{0.1, 0.2, 0.3\}$ on 50 samples from the test set. The results are summarized in Figure 3a with additional per-sample results in Appendix C.2. We see that our approach is able to capture variations in the robustnesses of the network.

As the method of Wong & Kolter (2018) returns the maximum value of the property for each sample over a convex outer bound on the perturbations, it is able to produce certificates-of-robustness for some datapoints. If the result returned is less than 0 then no adversarial examples exist in an l_∞ ball of radius ϵ around that datapoint. If the result returned is greater than 0, then the datapoint may or may not be robust in that l_∞ ball, due to fact that it optimizes over an outer bound.

Though we emphasize that the core aim of our approach is in providing richer information for SAT properties, this provides an opportunity to see how well it performs at establishing UNSAT properties relative to a more classical approach. To this end, we compared the fraction of the 50 samples from the test set that are verified by the method of Wong & Kolter (2018), to the fraction that have a negligible volume of adversarial examples, $\mathcal{I} = P_{\text{min}}$, in their l_∞ ϵ-ball neighbourhood. The results are presented in Figure 3b.

Our method forms an upper bound on the fraction of robust samples, which can be made arbitrarily tighter by taking $P_{\text{min}} \to 0$. Wong & Kolter (2018), on the other hand, forms a lower bound on the fraction of robust samples, where the tightness of the bound depends on the tightness of the convex outer bound, which is unknown and cannot be controlled. Though the true value must lie somewhere between the two bounds, our bound still holds physical meaning it its own right in a way that Wong & Kolter (2018) does not: it is the proportion of samples for which the prevalence of violations is less than an a given acceptable threshold P_{min}.

This experiment also highlights an important shortcoming of Wong & Kolter (2018). The memory usage of their procedure depends on how many ReLU activations cross their threshold over perturbations. This is high during initial training for $\epsilon = 0.1$ and indeed the reason why the training procedure starts from $\epsilon = 0.01$ and gradually anneals to $\epsilon = 0.1$. The result is that it is infeasible (the GPU memory is exhausted)—even for this relatively small model—to calculate the maximum

value of the property on the convex outer bound for $\epsilon \in \{0.2, 0.3\}$ at all epochs, and $\epsilon = 0.1$ for epochs before 32. Even in this restricted setting where our metric has been reduced to a binary one, it appears to be more informative than that of Wong & Kolter (2018) for this reason.

7 DISCUSSION

We have introduced a new measure for the intrinsic robustness of a neural network, and have validated its utility on several datasets from the formal verification and deep learning literatures. Our approach was able to exactly emulate formal verification approaches for satisfiable properties and provide high confidence, accurate predictions for properties which were not. The two key advantages it provides over previous approaches are: a) providing an explicit and intuitive measure for *how* robust networks are to satisfiable properties; and b) providing improved scaling over classical approaches for identifying unsatisfiable properties.

Despite providing a more informative measure of how robust a neural network is, our approach may not be appropriate in all circumstances. In situations where there is an explicit and effective adversary, instead of inputs being generated by chance, we may care more about how far away the single closest counterexample is to the input, rather than the general prevalence of counterexamples. Here our method may fail to find counterexamples because they reside on a subset with probability less than P_{min}; the counterexamples may even reside on a subset of the input space with measure zero with respect to the input distribution. On the other hand, there are many practical scenarios, such as those discussed in the introduction, where either it is unrealistic for there to be no counterexamples close to the input, the network (or input space) is too large to realistically permit formal verification, or where potential counterexamples are generated by chance rather than by an adversary. We believe that for these scenarios our approach offers significant advantages to formal verification approaches.

Going forward, one way the efficiency of our approach could be improved further is by using a more efficient base MCMC kernel in our AMLS estimator, that is replace line 12 in Algorithm 1 with a more efficient base inference scheme. The current MH scheme was chosen on the basis of simplicity and the fact it already gave effective empirical performance. However, using more advanced inference approaches, such as gradient-based approaches like Langevin Monte Carlo (LMC) (Rossky et al., 1978) and Hamiltonian Monte Carlo (Neal, 2011), could provide significant speedups by improving the mixing of the Markov chains, thereby reducing the number of required MCMC transitions.

ACKNOWLEDGMENTS

We gratefully acknowledge Sebastian Nowozin for suggesting to us to apply multilevel splitting to the problem of neural network verification. We also thank Rudy Bunel for his help with the COLLISIONDETECTION dataset, and Leonard Berrada for supplying a pretrained DenseNet model.

SW gratefully acknowledges support from the EPSRC AIMS CDT through grant EP/L015987/2. TR and YWT are supported in part by the European Research Council under the European Unions Seventh Framework Programme (FP7/20072013) / ERC grant agreement no. 617071. TR further acknowledges support of the ERC StG IDIU. MPK is supported by EPSRC grants EP/P020658/1 and TU/B/000048.

REFERENCES

Charles-Edouard Bréhier, Tony Lelièvre, and Mathias Rousset. Analysis of adaptive multilevel splitting algorithms in an idealized case. *ESAIM: Probability and Statistics*, 19:361–394, 2015.

Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. A unified view of piecewise linear neural network verification. *arXiv preprint arXiv:1711.00455v3 [cs.AI]*, 2018.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Verification of binarized neural networks. *arXiv preprint arXiv:1710.03107*, 2017.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 269–286. Springer, 2017.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

Ian Goodfellow. Gradient masking causes clever to overestimate adversarial perturbation size. *arXiv preprint arXiv:1804.07870*, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2015.

Arnaud Guyader, Nicolas Hengartner, and Eric Matzner-Løber. Simulation and estimation of extreme quantiles and extreme probabilities. *Applied Mathematics & Optimization*, 64(2):171–196, 2011.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, pp. 3, 2017a.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pp. 3–29. Springer, 2017b.

H. Kahn and T.E. Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30, 1951.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

Sebastian Nowozin. Multilevel splitting. `http://www.nowozin.net/sebastian/blog/multilevel-splitting.html`, 2015.

Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pp. 243–257. Springer, 2010.

Gareth O Roberts, Andrew Gelman, Walter R Gilks, et al. Weak convergence and optimal scaling of random walk metropolis algorithms. *The annals of applied probability*, 7(1):110–120, 1997.

PJ Rossky, JD Doll, and HL Friedman. Brownian dynamics as smart monte carlo simulation. *The Journal of Chemical Physics*, 69(10):4628–4633, 1978.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018a.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018b.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5283–5292, 2018.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, (99):1–7, 2018.

Radosiaw R Zakrzewski. Verification of a trained neural network accuracy. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pp. 1657–1662. IEEE, 2001.

APPENDIX

A METROPOLIS–HASTINGS

Metropolis–Hastings (MH) is an MCMC method that allows for sampling when one only has access to an unnormalized version of the target distribution (Gilks et al., 1995). At a high-level, one attempts iteratively proposes local moves from the current location of a sampler and then accepts or rejects this move based on the unnormalized density. Each iteration of this process is known as a MH transition.

The unnormalized targets distributions of interest for our problem are $\gamma_k(\mathbf{x})$ where

$$\pi_k(\mathbf{x}) \propto \gamma_k(\mathbf{x}) = p(\mathbf{x})\mathbb{1}_{\{s(\mathbf{x})\geq L_k\}}, \quad k = 1, 2, \ldots, K.$$

A MH transition now consists of proposing a new sample using a proposal $\mathbf{x}' \sim g(\mathbf{x}' \mid \mathbf{x})$, where $\mathbf{x}$ indicates the current state of the sampler and $\mathbf{x}'$ the proposed state, calculating an acceptance probability,

$$A_k(\mathbf{x}' \mid \mathbf{x}) \triangleq \min\left\{1, \frac{\gamma_k(\mathbf{x}')g(\mathbf{x} \mid \mathbf{x}')}{\gamma_k(\mathbf{x})g(\mathbf{x}' \mid \mathbf{x})}\right\}, \tag{7}$$

and accepting the new sample with probability $A_k(\mathbf{x}' \mid \mathbf{x})$, returning the old sample if the new one is rejected. The proposal, $g(\mathbf{x}' \mid \mathbf{x})$, is a conditional distribution, such as a normal distribution centred at $\mathbf{x}$ with fixed covariance matrix. Successive applications of this transition process generates samples which converge in distribution to the target $\pi_k(\mathbf{x})$ and whose correlation with the starting sample diminishes to zero.

In our approach, these MH steps are applied independently to each sample in the set, while the only samples used for the AMLS algorithm are the final samples produced from the resulting Markov chains.

B IMPLEMENTATION DETAILS

Algorithm 1 has computational cost $O(NMK)$, where the number of levels K will depend on the rareness of the event, with more computation required for rarer ones. Parallelization over N is possible provided that the batches fit into memory, whereas the loops over M and K must be performed sequentially.

One additional change we make from the approach outlined by Guyader et al. (2011) is that we perform MH updates on all chains in Lines 12, rather than only those that were previously killed off. This helps reduce the build up of correlations over multiple levels, improving performance.

Another is that we used an adaptive scheme for $g(\mathbf{x}'|\mathbf{x})$ to aid efficiency. Specifically, our proposal takes the form of a random walk, the radius of which, ϵ', is adapted to keep the acceptance ratio roughly around 0.234 (see Roberts et al. (1997)). Each chain has a separate acceptance ratio that is average across MH steps, and after M MH steps, for those chains whose acceptance ratio is below 0.234 it is halved, and conversely for those above 0.234, multiplied by 1.02.

C ADDITIONAL RESULTS

C.1 VARYING M FOR FIXED ρ ON COLLISIONDETECTION

Whereas the exact value of M within the range considered proved to not be especially important when $\rho = 0.1$, it transpires to have a large impact in the quality of the results for larger values of ρ as shown in Figure 4.

C.2 PER-SAMPLE ROBUSTNESS MEASURE DURING ROBUST TRAINING

Figure 5 illustrates the diverse forms that the per-sample robustness measure can take on the 40 datapoints averaged over in Experiment §5.3. We see that different datapoints have quite varying initial levels of robustness, and that the training helps with some points more than others. In one case, the datapoint was still not robust add the end of training for the target perturbation size $\epsilon = 0.1$.

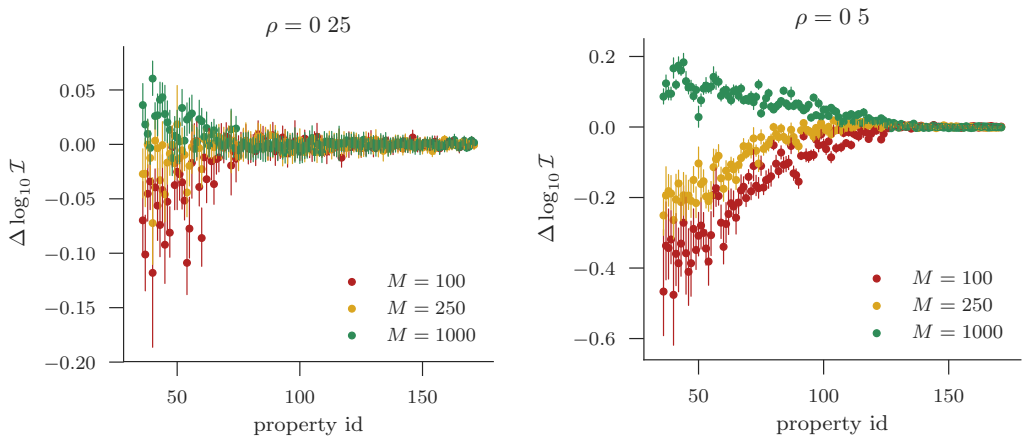


Figure 4: Mean AMLS estimate relative to naive (unbiased) MC estimate for different $M =$ holding ρ fixed to 0.25 (left) and 0.5 (right), for those properties whose naive MC estimate was greater than $\log_{10} \mathcal{I} = -6.5$ such that they could be estimated accurately.

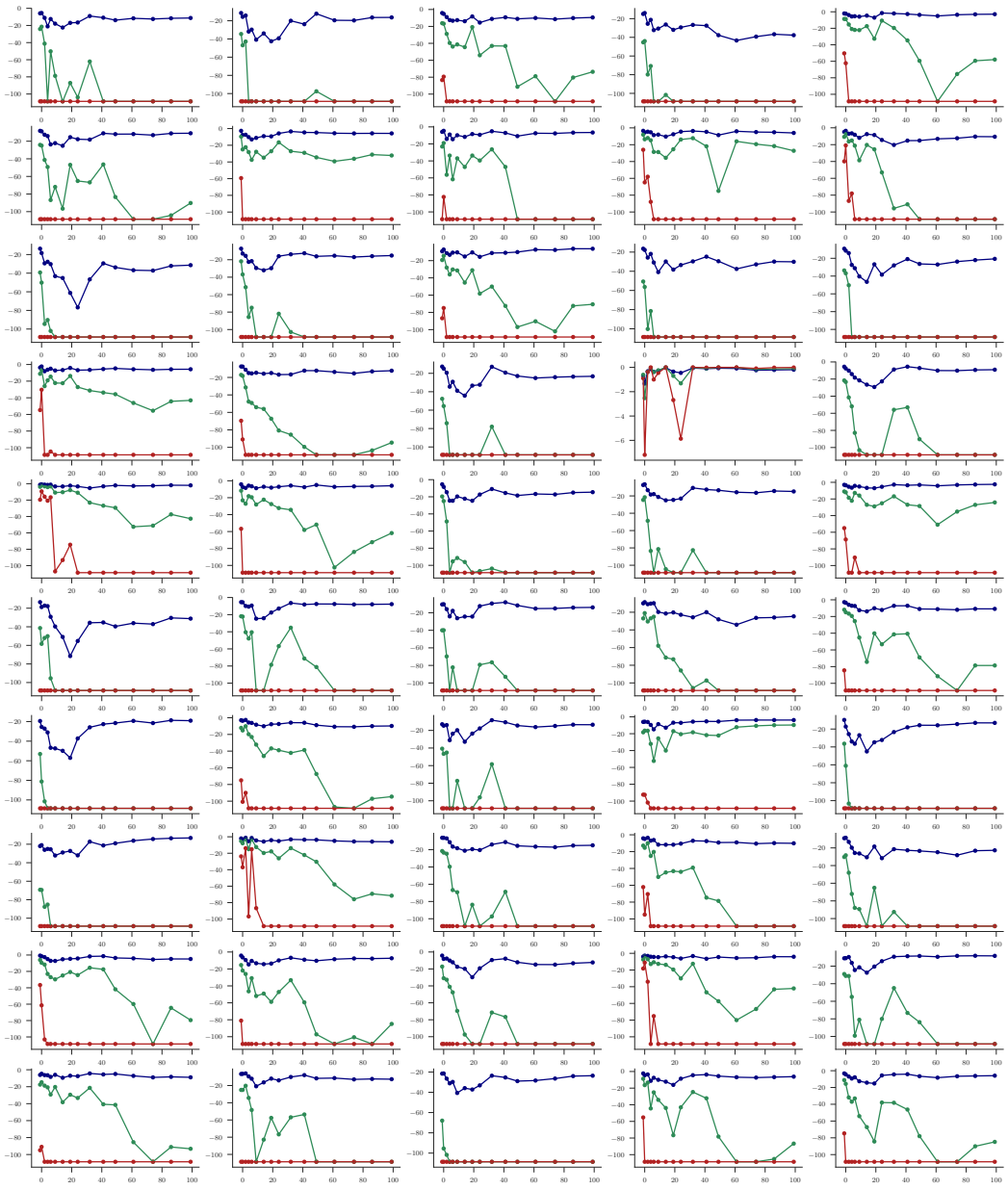


Figure 5: Convergence of individual datapoints used in forming Figure 3.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	A Statistical Approach to Assessing Neural Network Robustness
Publication Status	□Published X Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Webb, Stefan, Rainforth, Tom, Teh, Yee Whye, and Pawan Kumar, M. A Statistical Approach to Assessing Neural Network Robustness. To appear in the *Proceedings of the Seventh International Conference on Learning Representations (ICLR2019), New Orleans*.

Student Confirmation

Student Name:	Stefan Webb		
Contribution to the Paper	Developed idea suggested by supervisor. Performed all experiments. Wrote first draft of paper. Made posters for workshop and conference, camera ready version, and released source code repository.		
Signature		Date	12/12/2018

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof M. Pawan Kumar		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

6 Conclusions

To recap, we have presented three original pieces of work in this thesis. The common theme has been that neural networks and deep learning are crucial for scaling Bayesian inference on generative models, and conversely that traditional Bayesian inference is important for quantifying uncertainty in NN discriminative models. In this chapter, we summarize our contributions and suggest directions for future work.

6.1 Achievements

In Ch 3, we presented the NaMI algorithm for designing the structure of inference networks and demonstrated superiority of NaMI inverses on several models. A review was given in Ch 2 of the other parts of the NaMI framework assumed in Webb et al. [2018a], namely amortized VI and NN density estimators.

In Ch 4, we presented a novel framework for distributed Bayesian learning based on a modification of the standard EP algorithm. We provided evidence that our method was able to efficiently perform distributed learning of Bayesian NNs, suffering less from the stale-gradient problem of (non-Bayesian) distributed SGD algorithms, and outperforming all other methods in learning feedforward networks with many layers of depth.

In Ch 5, we developed a novel statistical measure of neural network robustness based on a traditional inference method and demonstrated its low bias and variance, and improved scalability relative to formal verification methods.

6.2 Future work

6.2.1 A statistical approach to assessing neural network robustness

When our method is applied to adversarial properties, it produces a measure of robustness on the NN *for a given image*. However, it would be desirable if we could produce a metric that measures overall robustness of the neural network, averaging over the data distribution. We believe this is an interesting and easy extension that could be accomplished by modifying the robustness metric to,

$$\mathcal{I}[p, s] \triangleq \int_{\mathcal{X}\times\mathcal{X}'} \mathbb{1}_{\{s(\mathbf{x};\mathbf{x}')\}\geq 0} p(\mathbf{x} \mid \mathbf{x}') p_{\mathcal{D}}(\mathbf{x}') d\mathbf{x} d\mathbf{x}',$$

where the input model, $p(\mathbf{x}, \mathbf{x}')$, can be broken up into two terms: the data distribution $p_{\mathcal{D}}(\cdot)$, and a uniform perturbation around the input, $p(\cdot|\mathbf{x}')$. The property function $s(\cdot)$ now becomes a function of the sampled input, $\mathbf{x}'$. Our algorithm based on AMLS would then be modified to perform the MH sampling over the modified input distribution—effectively the only change is performing approximate sampling from a distribution with double the dimension of the per-sample case.

A second extension is the investigation of improved sampling from the intermediate distributions, $\{\pi_n(\cdot)\}$. The simple Metropolis–Hastings (MH) scheme was found to be very robust and produce low bias/variance estimates in our experiments. However, it may not scale in the dimension, D, of larger datasets such as IMAGENET. Even for CIFAR-10 (with around 4000 dimensions) the required mixing time was found to be quite large in our experiments, being around $M = 2000$ steps to obtain satisfactory convergence to the target distribution. This becomes all the more important with larger models as well, for which both the constant factor of running the forward pass increases, and the number of Markov chains, N, that can be processed in parallel is reduced (due to limited GPU memory).

To improve the mixing time, and thus be able to reduce M without introducing unacceptable bias, we propose to replace the simple MH scheme with Langevin Monte Carlo (LMC) [Neal, 2011]. Theoretically, the computational time to reach a nearly independents sample scales as $O(D^2)$ for MH, whereas for LMC it scales as $O(D^{4/3})$. LMC can be understood as Hamiltonian Monte Carlo (HMC) with a single leapfrog step, or alternatively as a random walk that is informed by gradient information. The computational cost to reach an independent sample does scale slightly better for HMC, as $O(D^{5/4})$, although this is heavily outweighed by the increase in the constant factor due to the many more forward passes of the NN required during the leapfrog steps.

We suggested in a previous version of Webb et al. [2018b] that the adversarial samples produced by our algorithm may be of interest for adversarial training. However, it takes several minutes in typical usage to pass through all the levels to obtain samples from the final π_K. While it may not, therefore, be practical to *directly* use the samples from our algorithm for robustness training, we believe that a robust training method could be formed from the fundamental idea of our work, of sampling adversarial, or near-adversarial, samples by MCMC methods. Let us elaborate.

Many robust training methods can be understood as approximations to the following variant of empirical risk minimization [Madry et al., 2017; Tramèr et al., 2017],

$$f^* = \text{argmax}_{f \in \mathcal{F}} \mathbb{E}_{(\mathbf{x}, y^*) \sim \mathcal{D}} \left[\max_{||\mathbf{x}' - \mathbf{x}||_\infty \leq \epsilon} s(\mathbf{x}'; f, \mathbf{x}, y^*) \right],$$

where y^* is the true class label. For instance, the iterative fast gradient-sign method (I-FGSM) [Goodfellow et al., 2014b] approximates the inner max term by starting from x and taking k steps of size $\alpha = \epsilon/k$ in the direction of the sign of the gradient of the property function, clipping where necessary to keep within the l_∞ ϵ-ball and the valid range of pixels. In another method [Kolter and Wong, 2017; Wong et al., 2018], a convex outer bound, $E \subseteq \{\mathbf{x}' \mid s(\mathbf{x}'; f, \mathbf{x}, y^*) \geq 0\}$ is produced and the max performed over $\mathbf{x}' \in E$.

As pointed out in [Tramèr et al., 2017], if the approximation to the max term is poor, that is,

$$s(\mathbf{x}^*; f, \mathbf{x}, y^*) \ll \max_{||\mathbf{x}'-\mathbf{x}||_\infty \leq \epsilon} s(\mathbf{x}'; f, \mathbf{x}, y^*)$$

where $\mathbf{x}^*$ is the sample produced by the approximation method, then the adversarial training degrades the quality of adversarial samples generated by the model, which then degrades the quality of subsequent adversarial training. The result of this is that the model reaches an equilibrium during training where it can effectively defend against white-box but not black-box attacks, those where the adversarial samples are transferred from another model.

The authors suggest a new method for adversarial training in which they decouple adversarial training from adversarial generation, and use samples from an ensemble of additional source models that are held constant during the adversarial training of the target model (adding a small amount of random noise to each). In this way, the adversarial training does not influence the generation of samples used for the training. This relies on the heuristic that adversarial examples often transfer across models. However, it does seem unsatisfactory in that we are no longer directly solving the objective, and it may not be the case that examples do transfer across all architectures or that the transferable examples are representative of all adversarial examples on the model.

We propose to modify the objective to,

$$f^* = \text{argmax}_{f \in \mathcal{F}} \mathbb{E}_{(\mathbf{x}, y^*) \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{x}' \sim p(\cdot|\mathbf{x})} \left[s(\mathbf{x}'; f, \mathbf{x}, y^*) \right] \right],$$

where, for instance, $p(\mathbf{x}'|\mathbf{x}) \propto \exp(s(\cdot|\mathbf{x})) \mathbb{1}_{\{||\mathbf{x}'-\mathbf{x}||_\infty \leq \epsilon\}}$. We can draw approximate samples from p by LMC.

This method has several advantages. Firstly, we can produce multiple proposal samples per data point directly from the model in question, as opposed to only a single proposal sample from the model like in the I-FGSM method and that of Kolter and Wong [2017], or a number of samples per data point from surrogate models, like Tramèr et al. [2017]. Also, by the reliability of the sampling process, we can ensure that the proposals are diverse and are adversarial or near-adversarial, thus avoiding the poor equilibrium discussed before. As argued in Webb et al.

[2018b], in many cases it makes more sense to define robustness in terms of the prevalence of adversarial examples in the vicinity of a data point rather than the distance to the nearest one, and for this reason we would expect a training algorithm that trains on multiple and diverse adversarial examples per data point to be more effective in reducing our measure of robustness. Our proposed method is also fast, requiring only a few extra gradient steps per minibatch over the I-FGSM.

6.2.2 Faithful inverses for effective amortized inference

A very worthy outstanding challenge of Bayesian statistics is that of producing *universal automated inference*. Such a hypothesized inference scheme is said to be *universal* in that it would be applicable to higher-order probabilistic programs, those with recursion, other stochastic control flow mechanisms, and high-order functions, and, *automated* in that it would not require any intervention by the user after having specified the model and the query of interest in the details of how inference is applied.

A universal inference scheme based on amortized VI and the Anglican PPL [Wood et al., 2014] was attempted in Le et al. [2017]. Their method used particle-based inference with a data driven proposal to learn to perform inference in a universal PPL. As noted by the authors, it suffered from two shortcomings. Firstly, the method did not perform model learning, and as discovered in the experiments on CAPCHA solving, it is difficult to program a correct model ab initio. Secondly, the inference network conditioned on all previous random choices, not only the relevant ones (from the PGM perspective of Webb et al. [2018a]). As demonstrated in our work, this slows learning considerably and leads convergence to a worse solution.

We believe this important work is based on the most promising approach for universal automated inference, that of amortized VI, and that it can be further developed to overcome the hurdles the authors discovered. Firstly, new learning algorithms have been developed that would be suitable for probabilistic programs, with their seqential nature. For instance, filtering variational objectives, in a sense, can differentiate through particle-based inference algorithms in order to perform model learning [Maddison et al., 2017a; Le et al., 2018; Naesseth et al., 2018]. Secondly, the development of the NaMI algorithm provides an intelligent way to automate the design of the structure of the inference network. Currently, either the modeler must design the inference network by hand, or use an unstructured fully connected one (e.g., see the existing autoguides of Pyro [Bingham et al., 2018]).

We intend to integrate the NaMI algorithm into the Pyro PPL for designing the structure of an inference network, and use heuristics for designing the parametrization based on normalizing

flows. PPLs provide the necessary abstractions, in this case *poutines*, to manipulate a representation of the model in order to operate the NaMI algorithm and choose an appropriate factorization.

At first, we intend to demonstrate inference compilation on models with a fixed structure, such as traditional BNs. We believe the importance of a structured inference network increases in the scale of the model—an heuristic approach diverges more from the posterior as the number of variables increases, and a fully connected approach suffers the curse of dimensionality. Some traditional factor graph BNs, for example, those used for medical diagnosis, have thousands of variables, and would be suitable for this experimentation. Performing fast inference in these models is very important in industry.

Some challenges in automating the design of inference networks include automating the design of the NN encoder and decoder architectures and selection of hyperparameters like the learning rates. It is likely that meta-learning approaches are necessary, which learn from their past successes and failures and are able to generalize to novel scenarios.

Bibliography

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875v2 [cs.LG]*, 2017.

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

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017.

Robert Bamler, Cheng Zhang, Manfred Opper, and Stephan Mandt. Perturbative black box variational inference. In *Advances in Neural Information Processing Systems*, pages 5079–5088, 2017.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1): 1–127, 2009.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Christopher M Bishop. Mixture density networks. 1994.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670v2 [stat.CO]*, 2016.

Jörg Bornschein, Andriy Mnih, Daniel Zoran, and Danilo Jimenez Rezende. Variational memory addressing in generative models. In *Advances in Neural Information Processing Systems*, pages 3920–3929, 2017.

George EP Box. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 29:610–611, 1958.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519v3 [cs.LG]*, 2016.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259v2 [cs.CL]*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555v1 [cs.NE*, 2014.

Junyoung Chung, Caglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR, abs/1502.02367*, 2015.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

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

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural*

Information Processing Systems, pages 3225–3233, 2016.

Michael Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. *arXiv preprint arXiv:1805.08498*, 2018.

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

Andrew Gelman, Aki Vehtari, Pasi Jylänki, Christian Robert, Nicolas Chopin, and John P Cunningham. Expectation propagation as a way of life. *arXiv preprint arXiv:1412.4869*, 2014.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 881–889, 2015.

Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Samuel J Gershman and Noah D Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

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

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623v2 [cs.CV]*, 2015.

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Arnaud Guyader, Nicolas Hengartner, and Eric Matzner-Løber. Simulation and estimation of extreme quantiles and extreme probabilities. *Applied Mathematics & Optimization*, 64(2):171–196, 2011.

Leonard Hasenclever, Stefan Webb, Thibaut Lienart, Sebastian Vollmer, Yee Whye Teh, Balaji Lakshminarayanan, and Charles Blundell. Distributed bayesian learning with stochastic natural-gradient expectation propagation and the posterior server. *(In preparation)*, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

1997.

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.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. *arXiv preprint arXiv:1806.02426*, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

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

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099v1 [cs.CL]*, 2016.

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

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4736–4744, 2016.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009. ISBN 9780262013192.

J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 1(2):3, 2017.

Adam R Kosiorek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. *arXiv preprint arXiv:1806.01794*, 2018.

Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *AAAI*, pages 2101–2109, 2017.

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

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *AISTATS*, 2017.

Tuan Anh Le, Maximilian Igl, Tom Jin, Tom Rainforth, and Frank Wood. Auto-encoding sequential monte carlo. In *International Conference on Learning Representations (ICLR)*, 2018.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, volume 2, page 4, 2017.

Yingzhen Li and Richard E. Turner. Rényi divergence variational inference. *arXiv preprint arXiv:1602.02311v3 [stat.ML]*, 2016.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pages 2378–2386, 2016.

Christos Louizos, Uri Shalit, Joris M Mooij, David Sontag, Richard Zemel, and Max Welling. Causal effect inference with deep latent-variable models. In *Advances in Neural Information Processing Systems*, pages 6446–6456, 2017.

Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Teh. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, pages 6576–6586, 2017a.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

George Marsaglia and Wai Wan Tsang. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software (TOMS)*, 26(3):363–372, 2000.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6):1087–1092, 1953.

Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1791–1799, 2014.

Andriy Mnih and Danilo J. Rezende. Variational inference for Monte Carlo objectives. *arXiv preprint arXiv:1602.06725v2 [cs.LG]*, 2016.

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

Shakir Mohamed and Danilo Rezende. Tutorial on deep generative models, 2017. UAI 2017 Australia.

Christian A Naesseth, Scott W Linderman, Rajesh Ranganath, and David M Blei. Variational sequential monte carlo. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

Manfred Opper and David Saad. *Advanced mean field methods: Theory and practice*. MIT press, 2001.

Manfred Opper and Ole Winther. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6(Dec):2177–2204, 2005.

Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, 2016.

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

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1310–1318, 2013.

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

Rajesh Ranganath, Dustin Tran, Jaan Altosaar, and David Blei. Operator variational inference. In *Advances in Neural Information Processing Systems*, pages 496–504, 2016.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and

approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning*, 2014.

Francisco R Ruiz, Michalis Titsias RC AUEB, and David Blei. The generalized reparameterization gradient. In *Advances in neural information processing systems*, pages 460–468, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

N. Siddharth, Brooks Paige, Jan-Willem Van de Meent, Alban Desmaison, Frank Wood, Noah D. Goodman, Pushmeet Kohli, and Philip H. S. Torr. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *Advances in Neural Information Processing Systems*, 2017. URL `http://arxiv.org/abs/1706.00400`.

John Stachurski. *Economic dynamics: theory and computation*. MIT Press, 2009.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

EG Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Jakub M Tomczak and Max Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499v2 [cs.SD]*, 2016a.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016b.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016c.

Stefan Webb, Adam Golinski, Robert Zinkov, N. Siddharth, Tom Rainforth, Yee Whye Teh, and Frank Wood. Faithful inversion of generative models for effective amortized inference. In *Advances in Neural Information Processing Systems*, 2018a.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and Pawan Kumar M. A statistical approach to assessing neural network robustness. *arXiv preprint arXiv:1410.8516*, 2018b.

Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1805.12514*, 2018.

Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

Cheng Zhang, Judith Butepage, Hedvig Kjellstrom, and Stephan Mandt. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. *arXiv preprint arXiv:1412.6651*, 2014.