AN ABSTRACT OF THE DISSERTATION OF

Neale Ratzlaff for the degree of Doctor of Philosophy in Computer Science presented on July 19 2021.

Title: Uncertainty in Deep Learning with Implicit Neural Networks

Abstract approved: _____

Fuxin Li

The ability to extract uncertainties from predictions is crucial for the adoption of deep learning systems to safety-critical applications. Uncertainty estimates can be used as a failure signal, which is necessary for automating complex tasks where safety is a concern. Furthermore, current deep learning systems do not provide uncertainty estimates, and instead can assign high probability to incorrect predictions. To mitigate this problem of overconfidence, this dissertation proposes three approaches that leverage the uncertainty within a *distribution* of models. Specifically, we consider the epistemic uncertainty given by an approximation to the posterior over model parameters. Prior work approximates this posterior by utilizing analytically known distributions, which are inflexible and result in underestimation of the uncertainty. Instead, we propose to use implicit distributions, which are computationally efficient to sample from, and are flexible enough to parameterize a wide range of distributions. The contributions of this thesis show that implicit models enable better uncertainty estimates than prior work, and can be used for open-category prediction, adversarial example detection, and exploration in reinforcement learning.

We begin by showing that implicit generative models with feature-space regularization can be used in the open-category setting to detect input distribution shift, while retaining accuracy on training data. Next, we refine our approach by explicitly encouraging diversity within samples with particle-based variational inference. The uncertainty given by these diverse models is used for exploration in reinforcement learning. We show that

in the model-based setting we can leverage uncertainty as a novelty signal, compelling exploration to poorly understood areas of the environment. Third, we turn to the fundamental problem of approximate Bayesian inference. We develop a framework for generative particle-based variational inference that allows for efficient sampling, places no restrictions on the approximate posterior, and improves our ability to estimate epistemic uncertainty.

Uncertainty in Deep Learning with Implicit Neural Networks

by
Neale Ratzlaff

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 19 2021
Commencement June 2022

Doctor of Philosophy dissertation of Neale Ratzlaff presented on July 19 2021.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Neale Ratzlaff, Author

ACKNOWLEDGEMENTS

First and foremost I thank my advisor Fuxin Li for the support, encouragement, and mentorship. I count myself lucky to have an advisor enthusiastic about letting me pursue my research direction unencumbered, while providing guidance to shape me into a researcher. I consider Fuxin a valuable friend and ally as well as an advisor. I want to express my earnest gratitude for each member of my committee: Tom Dietterich, Alan Fern, Rakesh Bobba, Geoffrey Hollinger, and Stefano Ermon. Tom and Alan in particular have provided consistent, invaluable feedback and guidance. They kept me grounded to the task of solving real problems. I extend a special and sincere thanks to Qinxun Bai. Qinxun became a mentor when I interned at Horizon Robotics, and our collaboration has continued ever since. I cannot possibly enumerate everything I have learned from him. His devotion to details, and his willingness to assist me with any problem at any time have impacted every aspect of my research ability. I also extend my thanks to the excellent researchers Wei Xu, Haonan Yu, Haichao Zhang and Le Zhao from Horizon Robotics. They have my gratitude for contributing to the largest growth period of my PhD. My excellent friend and mentor Donald Heer deserves a special mention for taking an uninterested undergraduate, and providing autonomy and support. I directly credit my interest in AI, work ethic, and technical ability to my time in the TekBots lab. I made lifelong friends at OSU during my PhD, and I'm grateful for their support, clarity, and enthusiasm. A PhD is an arduous process, made significantly easier with good friends. I cannot name all of them but I would be remiss if I did not mention (in alphabetical order) Aayam Shrestha, Alexander Turner, Anurag Koul, Christopher Buss, Lawrence Neal, Matthew Olson, Michael Slater, Robert DeBortoli, Risheek Garrepalli, Saeed Khorram, Taylor Dinkins, and Yilin Yang. I also want to mention my friends and labmates: Ali, Amin, Hung, Jay, Jialin, Mazen, Michael, Tim, Vijay, Wenxuan, Xinyao, and Ziwen. They not only helped tremendously with feedback and advice, but they gracefully listened to the same presentations so many times from me. Last but never least are those outside of my PhD career. Alex Nolan has been my partner through everything, enduring the ups and downs of research, publications, and exams. I am forever grateful for her flexibility, kindness, and support while I pursued my passion. My friends Aniket, Carly, Christian, Cody, Jessica, Jordan, Serena, Spencer, and Trevor deserve

mention for their interest and tolerance as I worked on near-incomprehensible problems. I conclude with perhaps the most important people of all: my wonderful parents Jim and Vicki Ratzlaff, my brothers Chris, Jeff, and Bryan, and sisters Tiffany and Chelsea. They have made me who I am, and I count myself lucky to be related to them.

TABLE OF CONTENTS

Page

1 Introduction 1

1.1 Uncertainty in Deep Learning 1

1.2 Thesis Statement 4

1.3 Thesis Overview 6

2 Related Work 9

2.1 Approximate Bayesian Inference for Neural Networks 9

2.1.1 Markov Chain Monte Carlo 9

2.1.2 Variational Inference 10

2.1.3 Particle-based Variational Inference 11

2.1.4 Comparison to Proposed Work 12

2.2 Frequentist Distributions of Neural Networks 12

2.2.1 Comparison to Proposed Work 13

2.3 Exploration in Reinforcement Learning with Uncertainty 14

2.3.1 Model-Based RL 14

2.3.2 Model-Free RL 15

2.3.3 Comparison to Proposed Work 16

3 Preliminaries 17

3.1 Implicit Generative Models 17

3.1.1 Generative Adversarial Networks 17

3.1.2 Variational Autoencoders 18

3.1.3 Wasserstein Autoencoders 19

3.2 Bayesian Inference 20

3.2.1 Particle-based Variational Inference 21

3.2.2 Bayesian Neural Networks 22

3.2.3 Definition of Epistemic Uncertainty 23

3.3 Reinforcement Learning 24

3.3.1 Problem Statement 24

3.3.2 Definition of Epistemic Uncertainty 25

4 Implicit Generative Modeling with HyperGAN 26

4.1 Introduction 26

4.2 HyperGAN 27

TABLE OF CONTENTS (Continued)

Page

4.2.1 Learning to Generate without Explicit Samples 31

4.3 Experiments 32
4.3.1 Implementation Details 32
4.3.2 Classification Accuracy and Diversity 33
4.3.3 1-D Regression 34
4.3.4 Out of Distribution Detection 35
4.3.5 Ablation Study 38
4.3.6 Exemplar Outlier Examples 40

4.4 Conclusion 41

5 Efficient Exploration in Reinforcement Learning 42

5.1 Introduction 42

5.2 Dynamic Model Uncertainty as Intrinsic Reward 44

5.3 Posterior Approximation via Amortized SVGD 45
5.3.1 Implicit Posterior Generator 45
5.3.2 Training with Amortized SVGD 46
5.3.3 Summary of the Exploration Algorithm 47

5.4 Experimental Results 48
5.4.1 Toy Task: NChain 49
5.4.2 Pure Exploration Results 50
5.4.3 Policy Transfer Experiments 55
5.4.4 Comparison to Model-Free Exploration Approaches 57

5.5 Conclusion 58

5.6 Additional Implementation Details 58

6 Generative Particle-based Variational Inference 62

6.1 Introduction 62

6.2 Generative Particle Variational Inference 64
6.2.1 Reparameterization of the Generator 67
6.2.2 Estimating the Jacobian inverse 69
6.2.3 Summary of the algorithm 70
6.2.4 Comparison with Amortized SVGD 71

6.3 Experiments 73
6.3.1 Likelihood-based Generative Modeling 73

TABLE OF CONTENTS (Continued)

Page

6.3.2 Direct Generative Modeling 84
6.4 Conclusion 89
6.5 Hyperparameter Settings for GPVI 90

7 Improving GPVI 96
7.1 Introduction 96
7.2 Improving Generative Particle-based Variational Inference 97
7.2.1 GPVI 98
7.2.2 A Simplification of the Inverse 98
7.2.3 Practical Computation of the Inverse 101
7.3 Experiments 103
7.3.1 5D Density Estimation 105
7.3.2 Open-Category Prediction 105
7.3.3 Stacked MNIST 108
7.4 Conclusion 109
7.5 Hyperparameter Settings for GPVI+ 110

8 Future Work and Conclusion 116

Bibliography 119

LIST OF FIGURES

Figure Page

1.1 Neural network classifier performance on never-before-seen inputs. Figure (a) shows expected behavior on inputs that resemble training data. While (b) shows prediction on a semantically different input. Predicting a bird as a "7" is a silent failure. 2

4.1 HyperGAN architecture. The mixer transforms $\boldsymbol{z} \sim Z$ into latent codes $\{q_1, \ldots, q_L\}$. The L generators each transform the latent subvector q_i into the parameters of the corresponding layer in the target network. The discriminator D forces $Q(\boldsymbol{s}|\boldsymbol{z})$ to be well-distributed and according to the distribution $\mathcal{P}$. 27

4.2 Results of HyperGAN on the 1D regression task. From left to right, we plot the predictive distribution of 10, 100, and 256 sampled models from a trained HyperGAN. Within each image, the blue line is the target function x^3, the red circles show the noisy observations, the grey line is the learned mean function, and the light blue shaded region denotes ± 3 standard deviations 35

4.3 Out of distribution performance on notMNIST. PDF of the predictive entropy of all approaches. 36

4.4 Out of distribution performance on CIFAR-10. PDF of the predictive entropy of all approaches on the 5 classes of CIFAR-10. 36

4.5 Predictive entropy on FGSM and PGD adversarial examples. HyperGAN generates models that give diverse predictions on adversarial examples. Note that for large ensembles, it is hard to find adversarial examples with small norms e.g. $\epsilon = 0.01$ 38

4.6 Diversity of predictions on adversarial examples. FGSM and PGD examples are created against a network generated by HyperGAN, and tested on 500 more generated networks. FGSM transfers better than PGD, though both attacks fail to cover the distribution learned by HyperGAN 39

LIST OF FIGURES (Continued)

Figure | Page

4.7 Ablation studies of HyperGAN accuracy and diversity on CIFAR-10. (a) HyperGAN without the mixer and without the discriminator, respectively. (b) HyperGAN diversity on CIFAR-10 given a normal training run, with the mixer removed, and with the discriminator removed. Diversity is shown as the standard deviation divided by the norm of the weights, within a population of 100 generated networks. 40

4.8 Example outliers. Top: MNIST examples that HyperGAN assigns high entropy to (outlier examples). Bottom: notMNIST examples that are assigned low entropy (inlier examples) 41

5.1 Dynamics model generator. Independent noise samples $\{z^1, \cdots, z^L\}$ are drawn from a standard Gaussian with diagonal covariance, and input to layer-wise generators $\{f_1, \cdots, f_L\}$. Each generator f_j outputs parameters θ^j for the corresponding j-th layer of the neural network representing the dynamic model. 45

5.2 NChain environment. 49

5.3 Results on the 40-link chain environment. Each line is the mean of three runs, with the shaded regions corresponding to ± 1 standard deviation. Both our method and *MAX* actively reduce uncertainty in the chain, and therefore are able to quickly explore to the end of the chain. ϵ-greedy DQN fails to explore more than 40% of the chain. 50

5.4 Performance on the Acrobot environment. We average five seeds, with error bars representing ± 1 standard deviation. The length of each horizontal bar indicates the number of environment steps each agent/method takes to swing the acrobot to fully horizontal on both (left and right) directions. 52

5.5 We show (a) the U-shaped ant maze. Figures (b-e) show the behavior of our (*Ours*) agent at different stages of training, over 5 seeds. Points are color-coded with blue points occurring at the beginning of the episode, and red points at the end. 53

LIST OF FIGURES (Continued)

Figure Page

5.6 Ant maze performance. (a) performance of each method with mean and ± 1 standard deviation (shaded region) over five seeds. x-axis is the number of steps the ant has moved, y-axis is the percentage of the U-shaped maze that has been explored. Figure (b) shows the proposed intrinsic reward magnitude for each step in the environment, calculated for both our method and *MAX*. 54

5.7 We show (a) the Robotic Hand task in motion. (b) Performance of each method with mean and ± 1 standard deviation (shaded region) over five seeds. x-axis is the number of manipulation steps, y-axis is the number of rotation states of the block that has been explored. Our method (red) explores clearly faster than all other methods. 55

5.8 Policy transfer results. Figure (a) shows the performance of SAC baseline (SAC) and four SAC variants initialized from a 10,000-step exploration policy trained with different intrinsic reward methods, ICM, Disgreement, MAX, and our proposed method (Ours) respectively. Figure (b) shows the performance of downstream SAC policy training initialized with our proposed exploration policy under different settings of exploration length. Init-Nk refers to the SAC agent initialized from our exploration policy which has been trained for N-thousand exploration steps. 56

5.9 Comparison with RND and PSNE on HalfCheetah with external reward (a), and a pure exploration comparison with RND on Ant Maze (b). . . . 58

6.1 Predictive uncertainty of methods for a 1-D regression task. (a) HMC predictive posterior matches the uncertainty in the data; (b) SVGD performs comparably to HMC; (c) Our proposed GPVI performs similarly to SVGD, with the additional capability of sampling new particles during inference; (d) Amortized SVGD overestimates the uncertainty when the data is sparse. 65

6.2 Density estimation with the energy potential functions from Rezende and Mohamed [2015]. 76

6.3 Comparing our helper network with BiCGSTAB in the Bayesian linear regression setting. 79

LIST OF FIGURES (Continued)

Figure Page

6.4 Predictive uncertainty of each method on the 4-class classification task, as measured by the standard deviation between predictions of sampled functions. Regions of high uncertainty are shown as darker, while lighter regions correspond to lower uncertainty. The training data is shown as samples from four unimodal normal distributions. It can be seen that amortized SVGD, BBB and deep ensembles significantly underestimate the uncertainty in regions with no training data 81

6.5 Predictive uncertainty of each method on the 2-class classification task, as measured by the standard deviation between predictions of sampled functions. 82

6.6 GPVI vs GAN for direct generative modeling of 8 Gaussians. 86

6.7 GPVI vs GAN for direct generative modeling of 25 Gaussians. 87

7.1 GPVI+ parameter generation pipeline, by concatenation $\oplus$ of the output of $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$. 99

7.2 How we compute the partial Jacobian matrices $J_{\boldsymbol{g}_1}$ and $J_{\boldsymbol{g}_2}$ for computation of the inverse defined in theorem (7.2.1). 102

7.3 Samples drawn after training on Stacked MNIST after 100 epochs. 109

LIST OF TABLES

Table Page

4.1 HyperGAN diversity. 2-norm statistics on the layers of a population of network parameters sampled from HyperGAN, compared to a 10-network ensemble, as well as 10 samples from APD. Both HyperGAN and the standard models were trained on MNIST to 98% accuracy. Its easy to see that HyperGAN generates more diverse networks under this metric 33

4.2 Classification performance of HyperGAN on MNIST and CIFAR-10. CIFAR-5 refers to a dataset with only the first 5 classes of CIFAR-10. MNIST 5000 and CIFAR-10 5000 refers to training on only 5000 examples of MNIST and CIFAR-10, respectively, which has been used in prior work (e.g. Louizos and Welling [2017]) 34

6.1 Comparison of generative Particle VI approaches for density estimation of 2d and 5d Gaussian distributions. 75

6.2 Bayesian linear regression. Reported error is the L_2 norm of the difference between the learned mean and covariance parameters, and the ground truth after 50000 iterations. 78

6.3 Results for open-category classification on MNIST. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers. 83

6.4 Open-category classification on CIFAR-10. We show results of standard supervised training (Clean), as well as AUC and ECE of each method trained in the open-category setting. 84

6.5 Error metrics for each method in fitting 8 Gaussians. We show the average absolute error across modes, in estimating the mean and standard deviation of each component. 86

6.6 Error metrics for each method in fitting 25 Gaussians. We show the average absolute error across modes, in estimating the mean and standard deviation of each component. 87

6.7 Hyperparameters for density estimation of energy potentials. 91

6.8 Hyperparameters for Bayesian linear regression task 92

LIST OF TABLES (Continued)

Table | Page

6.9 Hyperparameters for 2/4 class classification task 93

6.10 Hyperparameters for MNIST open category task 94

6.11 Hyperparameters for CIFAR-10 open category task 95

6.12 Hyperparameters for Bayesian linear regression task 95

7.1 Comparison of different inversion methods for fitting a 5D diagonal Gaussian distribution. 105

7.2 Results for open-category classification on MNIST. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers. We also compare parameter count of GPVI and GPVI+ auxiliary networks, as well as LinGPVI+ and EnsembleGPVI+. 107

7.3 Results for open-category classification on CIFAR-10. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers. We also compare parameter count of GPVI and GPVI+ auxiliary networks, as well as LinGPVI+ and EnsembleGPVI+. . 108

7.4 Performance on stacked MNIST. We compute the number of modes captured by each generative model, along with the reverse KL divergence between output samples and true samples 109

7.5 Hyperparameters for density estimation. 111

7.6 Hyperparameters for MNIST open category task 113

7.7 Hyperparameters for CIFAR-10 open category task 114

7.8 Hyperparameters for Stacked MNIST task 115

LIST OF ALGORITHMS

Algorithm		Page
1	Exploration with an Implicit Distribution	48
2	Generative Particle VI (GPVI)	70
3	GPVI+ Training Algorithm	104

Chapter 1: Introduction

> All models are wrong but some of them are useful.
>
> *Box, G. E. P. (1976), "Science and statistics", Journal of the American Statistical Association*

When we build models of the world, either mental or mathematical, we know intuitively that there are limits to the variables we can observe – our models may never be perfect. However, the mathematical models we have built have no such understanding. This thesis examines how to imbue models with this intuition.

1.1 Uncertainty in Deep Learning

Modern society has been transformed by the invention of mechanical thinking power. Just thirty years ago, the average adult compressed, stored, and optimized for recall of city maps, basic arithmetic, and vacation memories. Neural circuits once reserved for these tasks have been handed over to navigation apps, smartphone calculators, and sprawling cloud photo libraries. Until recently, tasks involving prediction have resisted the handover. For the most part, humans still drive their own cars, choose their investments, and prescribe medication.

Recently, deep learning based models have been proposed to automate each of these tasks. This poses the problem of validation of complex models: how can designers guarantee the effectiveness of their models in such safety-critical applications? Such systems must be safe to use, or else be designed to fail gracefully in a way that allows operators to respond effectively. Due to the superior predictive power of deep learning systems over

traditional approaches, they are a natural choice for automation in domains like medicine and autonomous driving. In applications where large amounts of data are available, deep neural networks dominate in terms of accuracy and generalization capabilities. While accuracy and generalization performance are critical to their usefulness, a natural question is, how do neural networks behave on novel data. For example, what predictions do we get by prompting a digit classifier with a picture of a bird? Ideally, a failure signal could be efficiently computed from such predictions. The research community has collectively asked this question, and it has become clear that test inputs are indiscriminately treated like samples from the training distribution. On test inputs where this assumption holds, i.e. laboratory conditions, neural networks generalize well. However, in the event of distribution shift, where the new input does not resemble the training data, the associated predictions do not reflect that any such shift has happened. In practice for classification tasks, neural networks assign high probability to a known yet incorrect class. In regression tasks, predictions tend toward the mean of the training set, instead of attempting to extrapolate correctly. This inability to fail gracefully is illustrated in figure 1.1, which implies that if we query our digit classifier on a bird image, the classifier will confidently predict our bird is the ideal number "7".

(a) (b)

Figure 1.1: Neural network classifier performance on never-before-seen inputs. Figure (a) shows expected behavior on inputs that resemble training data. While (b) shows prediction on a semantically different input. Predicting a bird as a "7" is a silent failure.

In many real-world scenarios, models are queried for predictions on novel inputs. Given an anomalous input, we would prefer that the model is *uncertain* about the input, rather than broadcast an incorrect prediction. In autonomous driving, the car's perception system may encounter an obstruction that it has never seen before. The ability to be uncertain about this object could save the life of the driver by pulling over, instead

of assuming the object is benign and running into it. Although this uncertainty is fundamental in human reasoning, it has proven difficult to build models with the same behavior. Neural networks in particular have shown the surprising tendency to give highly confident predictions on data from outside the training distribution.

Despite its importance in human decision-making, uncertainty is difficult to define as an algorithmically computable quantity. In general, the uncertainty that we consider has two components: *aleatoric* and *epistemic* uncertainties. Aleatoric uncertainty stems from the inherent randomness in the environment. This type of uncertainty is irreducible, as the randomness will persist no matter how a learner models it. Epistemic uncertainty, also called model uncertainty, refers to a lack of knowledge about the parameter being estimated. This may be due to limited available data, or else from model misspecification. Epistemic uncertainty is in principle reducible; given a consistent estimator, the epistemic uncertainty about the true value of a parameter will decrease as more data is acquired. For example, if we flip a coin chaotically, the randomness in the coin flip is our aleatoric uncertainty. But if we want to model the posterior mean over many coin flips, this epistemic uncertainty will decrease over time as our estimate concentrates around the true value. This thesis concerns epistemic uncertainty: how can we reason about the true value of neural network parameters given that we have finite data.

Due to the high dimensionality and extreme non-convexity of neural network loss landscapes, it is possible to train randomly initialized networks that achieve similar accuracy, despite having quite different parameters. Ensembles of these diverse networks are more robust to novel data (outliers) and can provide uncertainty estimates over their outputs. The uncertainty associated with a prediction indicates how likely the prediction is to be correct given the observed data. Here we consider parameterized classifiers, that output a set of class-conditional probabilities. In this case we use the entropy in the predictive distribution as our uncertainty statistic. Intuitively, a maximum entropy predictive distribution indicates that no class is more likely than any other. When the classifier is an ensemble of models, individual members may be still be overconfident without sacrificing uncertainty quantification if the predictions are sufficiently diverse. This is analogous to a panel of expert judges, where each judge confidently disagrees in their verdict. Such a panel is clearly uncertain, while remaining individually confident. Hence, diversity in the predictive distributions can be critical to the uncertainty quantification ability of

ensembles. To imbue neural networks with the ability to be uncertain on novel data, we study how to train such diverse neural networks.

1.2 Thesis Statement

In this thesis, we study the problem of uncertainty in deep learning through the lens of ensembles of diverse models. That is, we consider k-class classification functions $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^k$ parameterized by $\boldsymbol{\theta}$. Consider the joint distribution $(\mathcal{X}, \mathcal{Y})$ of data points $\mathcal{X}$ and associated labels $\mathcal{Y}$. We can draw a set of data points of interest $X = \{x_1, x_2, \ldots, x_n\} \subset \mathcal{X}$, and a corresponding set of labels $Y = \{y_1, y_2, \ldots, y_n\} \subset \mathcal{Y}$. For a classifier that is trained to infer $p(y|x)$ given access to limited samples drawn i.i.d. from $(\mathcal{X}, \mathcal{Y})$, we can always find an anomalous input $\bar{x} \in \bar{X},\ \bar{X} \notin supp(X)$ with true label $\bar{y}$. Since $\bar{y} \notin Y$, the classification $\arg\max_y p(y|\bar{x})$ will be necessarily incorrect. Furthermore, the corresponding class probability $\max_y p(y|\bar{x})$ may be arbitrarily close to 1 when $\boldsymbol{f}_{\boldsymbol{\theta}}$ is a nonlinear neural network [Hein et al., 2019]. In this setting, the prediction with respect to $\bar{x}$ is guaranteed to be incorrect, yet it is indistinguishable from predictions on data drawn from X.

We take the view that the predictive distribution $p(y|\bar{x})$ should not assign high probability to any class. In fact, we want this distribution to have high entropy. In practice, this is not the case for nonlinear neural networks trained with maximum likelihood. However, we can improve on such models by using an ensemble. We can always construct an ensemble $q(\boldsymbol{\theta}) = \sum_{i=1}^{M} f_{\boldsymbol{\theta}_i}$. For an ensemble $q(\boldsymbol{\theta})$ with M members, we have M different predictive distributions for a given input. In this case, even if the predictions $\max_y p(y|\bar{x}, \boldsymbol{f}_{\boldsymbol{\theta}_i})$ are close to 1, we can consider the entropy of the average of the predictive distributions. If each classifier is overconfident, but the members "disagree", then the distribution $\frac{1}{M}\sum_{i=1}^{M} p(y|\bar{x}, \boldsymbol{f}_{\boldsymbol{\theta}_i})$ may still have high entropy.

Naive ensembles, where the parameters for each classification function are initialized randomly, have been explored in prior work for uncertainty quantification [Lakshminarayanan et al., 2017b]. These ensembles were found to outperform individual models by a surprising amount, when using predictive entropy as a measure of uncertainty to detect out-of-distribution (OOD) inputs. While ensembles give reasonable uncertainty

estimates, they represent a finite number of solutions that are not regularized to be diverse. Whether an ensemble learns diverse functions, or collapses to a single attractive solution depends on the random initialization and the optimization landscape. Furthermore, ensembles scale linearly in their computational cost, with the number of models in the ensemble limited by computational budget. For sufficiently complex data, we might require many models to estimate uncertainty accurately, and we cannot easily generate another ensemble member.

We take the view that the parameters of any trained classification function represent a sample from the distribution of parameter vectors $p(\Theta)$ that fit the data (with some specified accuracy) e.g. $p(\Theta) \overset{d}{=} p(\boldsymbol{\theta}|X, Y)$. Under this view, a naive ensemble approximates $p(\Theta)$ with an empirical distribution of M samples. We can also find an approximation $p(\tilde{\Theta}) \overset{d}{\approx} p(\Theta)$ with a (possibly) simpler analytically known distribution. For example, we might assume that our target distribution is approximately Gaussian $p(\Theta) \overset{d}{=} \mathcal{N}(\mu, \Sigma)$, but with unknown mean and variance. We could then perform variational inference to find the parameters of $p(\tilde{\Theta})$ that best fit the target $p(\Theta)$. The benefits of variational inference (VI) are twofold; we can encode prior knowledge about the data e.g. we know the form of the target distribution. VI with a tractable approximating distribution also allows us to generate additional samples from $p(\tilde{\Theta})$, something we could not do with an ensemble. Of course, in modern deep learning, we do not *a priori* know the form of $p(\Theta)$, and distributions with analytical forms are likely not flexible enough to be a very close approximation. Indeed, inflexible variational distributions have been shown to underestimate the epistemic uncertainty [Minka, 2001, Bishop, 2006, Snoek et al., 2019] for even low-dimensional functions. We seek to gain the advantages of naive ensembles and variational methods while avoiding the drawbacks. Hence, we seek a representation of $p(\tilde{\Theta})$ that allows for efficient sampling and accurate uncertainty quantification.

Implicit generative models are functions $\boldsymbol{f_\eta} : \mathbb{R}^d \to \mathbb{R}^m$ that define a deterministic transformation of samples $\boldsymbol{z}$ from a simple distribution Z via parameters $\boldsymbol{\eta}$. Given an implicit generative model, we can directly generate data as $\boldsymbol{x} = \boldsymbol{f_\eta}(\boldsymbol{z})$, $\boldsymbol{z} \sim Z$, where $supp(Z) \subseteq \mathbb{R}^d$. We consider the implicit distribution given by samples from $\boldsymbol{f_\eta}(\boldsymbol{z}), \boldsymbol{z} \sim Z$ as an approximation of $p(\Theta)$ if we let $\boldsymbol{f_\eta}$ generate parameter vectors $\boldsymbol{\theta}$. Additionally, $\boldsymbol{f_\eta}(\boldsymbol{z})$ is computationally efficient to sample from and does not assume a constraining parameterization. However, it is not clear how to train an implicit generative model to

approximate $p(\Theta)$, or if the resulting approximation will yield useful uncertainties. This thesis proposes methods to train implicit generative models to do exactly this, and shows that the resulting samples can be used for accurate uncertainty quantification. The main statement of this thesis is as follows:

> *Implicit generative models can be used to learn an approximation of the posterior distribution of neural network parameters that fit the observed data. Samples from this approximate posterior can be effectively used for uncertainty quantification, enabling out-of-distribution input detection, defense against adversarial examples, and exploration of unknown environments.*

A large component of this thesis is about showing not just that implicit generative models can produce diverse models and reasonable uncertainty estimates, but also that the uncertainty quantification is useful for downstream applications. For instance, a reinforcement learning agent tasked with exploration may seek out unknown areas by leveraging its own uncertainty about the environment.

The diversity given by implicit generative models is not only useful for uncertainty estimation, but also for learning about low-probability data. A generative model that can produce diverse samples is also broadly useful for tasks such as density estimation or image generation. Implicit generative models are a key component of modern image generation algorithms e.g. generative adversarial networks (GANs). However, GANs are known to suffer from low sample diversity. This means that if GANs are used to represent human qualities via digital avatars or online shopping applications, low sample diversity implies that low-density demographics are excluded. Hence, this dissertation considers these application domains within the overarching question of how to construct such implicit generative models.

1.3 Thesis Overview

This thesis is organized as a roughly linear road map through our efforts leveraging implicit generative models for uncertainty quantification in neural networks. Each of the main ideas corresponds to a published paper, though in some sections we extend

our published results to connect related topics and show additional applications of our work.

We begin with introducing prior work most relevant to this thesis in chapter 2. This includes a survey of prior methods covering the major topic areas: uncertainty quantification, Bayesian neural networks, and exploration in reinforcement learning. In chapter 3 we give a preliminary introduction to the formulation used in this thesis. The notation, terms, and broad subject ares are introduced here. In order, we cover implicit generative models, Bayesian inference, particle-based variational inference, and exploration in reinforcement learning.

In chapter 4, we introduce our first contribution utilizing implicit generative models for uncertainty quantification. We propose HyperGAN, a generative architecture that transforms a low-dimensional sample of random noise into a full set of parameters for a target neural network architecture. We regularize the generative model with an adversarial discriminator to encourage diversity in generated samples. Our experiments show that HyperGAN generates diverse models that give reasonable uncertainty estimates when used for out-of-distribution data and adversarial example detection.

In chapter 5 we study how implicit models and uncertainty quantification can be used for efficient exploration in reinforcement learning. We demonstrate the effectiveness of implicit generative models in the model-based setting, by leveraging the uncertainty in an implicit distribution over the transition function to drive exploration. Our agent is motivated to explore via an intrinsic reward powered by this uncertainty. We target environments with sparse external rewards, where exploration is difficult. In such environments, an effective agent must efficiently explore a significant portion of the state space, since the true reward function may have negligible (or even zero) support over the state space. Our experiments show that our approach consistently outperforms competing methods regarding data efficiency in pure exploration.

In chapter 6 we introduce a method for constructing an implicit generative model such that its samples match an arbitrary target distribution. Our method *Generative Particle Variational Inference* (GPVI) minimizes the functional gradient of the KL divergence between an implicit distribution given by samples from a generative model, and a target distribution. The benefit over prior work is that GPVI places no restrictions on the

form of the generator function, such as invertibility. In tasks such as Bayesian linear regression, and sampling from energy functionals, we show that GPVI is able to exactly fit arbitrary target distributions. We further apply it to Bayesian neural networks, and show that the uncertainty estimates given by the sampled networks outperform prior work for open-category prediction.

Chapter 7 extends our work on GPVI to GPVI+, and proposes a method to greatly decrease the difficulty of training GPVI, and improve its space-efficiency. We derive a new way to invert the Jacobian of the generator network, reducing the dimensionality of the inversion problem by up to 99%. Our results on low-dimensional density estimation as well as Bayesian neural networks show substantial gains in efficiency as well as an improvement in uncertainty estimation ability.

We conclude in chapter 8 with a reflection on the research presented in this thesis and a discussion of new avenues, such as continual learning, multi-task learning, and transfer learning.

Chapter 2: Related Work

This thesis draws from a variety of fields including Bayesian inference, probabilistic neural networks, ensemble learning, and reinforcement learning.

2.1 Approximate Bayesian Inference for Neural Networks

Bayesian inference provides a powerful framework for reasoning and prediction under uncertainty. As such, there is a vast body of work utilizing Bayesian inference for inferring the parameters of neural networks [Neal, 1994]. Most widely used techniques fall roughly into the categories of the Laplace approximation, Markov chain Monte Carlo (MCMC) and variational inference, which we discuss in the following sections.

The Laplace approximation assumes that the posterior distribution can be approximated well by a Gaussian distribution, where the mean is given by the MAP solution and the covariance is the inverse of the Fisher Information Matrix (inverse of the Hessian). Implementation of this method for neural networks is computationally demanding. Therefore, the following approximations have been proposed. MacKay [1992] used a diagonal approximation to the Hessian to compute the Laplace approximation for Bayesian neural networks. Recently, a more scalable version of the Laplace approximation was proposed by Ritter et al. [2018], by using a Kronecker factored (KFAC) Hessian.

2.1.1 Markov Chain Monte Carlo

Traditionally, MCMC algorithms are employed to simulate drawing samples from arbitrary target distributions. As MCMC methods are known to asymptotically converge given both time and ergodicity, they are useful for both exact and approximate Bayesian inference. Hamiltonian Monte Carlo (HMC) [Neal et al., 2011] evolves a Markov chain under Hamiltonian dynamics, resulting in fast convergence due to less correlated samples. HMC has been extended to gradient-based optimization with SGHMC [Chen et al.,

2014]. The noisy gradient estimator proposed for SGHMC allows for sampling weights for BNNs. Welling and Teh [2011] proposed Stochastic Gradient Langevin Dynamics (SGLD), a first order approximation to Langevin diffusion for BNN training. RECAST [Seedat and Kanan, 2019] integrates cosine annealing with the SGLD step size for efficient exploration of the target posterior distribution.

Ultimately, MCMC based algorithms do not scale well to estimation problems where the distribution in question has millions of parameters. Some MCMC methods like HMC are "full-batch", meaning that we cannot use stochastic gradient approaches which are critical for training large neural networks. Some approaches like SGHMC or RECAST support minibatch updates, but demonstrate slower mixing and their performance is highly dependant on model specification [Yao et al., 2019].

2.1.2 Variational Inference

Variational Inference [Wainwright and Jordan, 2008] transforms intractable Bayesian inference problems into simpler optimization procedures. Unlike MCMC methods, variational inference (VI) assumes that the model family of the posterior distribution is known. VI optimizes for a set of approximating parameters for this distribution, and so can be more efficient than MCMC. VI for neural network parameters was first proposed by Hinton and Van Camp [1993] under the minimum description length framework, but requires a analytic expression of the integral over the variational distribution to perform updates. Hoffman et al. [2013] and Graves [2011] provide more general VI algorithms, using MCMC to compute noisy (yet unbiased) estimates of the gradient of the variational parameters based on subsamples of the data. More recently, variational approximations have been iterated upon extensively. Bayes by Backprop [Blundell et al., 2015] introduced a more efficient gradient estimator using the local reparameterization trick from Kingma et al. [2015] to reduce the number of variational parameters. Probabilistic Backpropagation [Hernández-Lobato and Adams, 2015] makes use of the intermediate gradients computed by the backpropagation algorithm to efficiently update a factorized Gaussian approximate posterior. Because diagonal Gaussian approximate posteriors do not capture correlations between weights, structured approximate posteriors have been recently proposed, making use of matrix Gaussians [Louizos and Welling, 2016], normalizing flows

[Louizos and Welling, 2017, Krueger et al., 2017], and hypernetworks [Pawlowski et al., 2017].

In practice, VI requires approximations to scale to the high dimensionality of neural networks. The most common assumption is that the form of the posterior can be closely approximated by a simple analytically-known distribution. This assumption enables closed-form updates to the variational posterior, and is computationally amenable for drawing samples. However, common distribution choices such as a diagonal Gaussian distribution, are insufficient to capture the complex structure of high dimensional model posterior [Yao et al., 2019]. Even with more structured variational distributions, VI is known to underestimate the uncertainty in the target distribution [Minka, 2001, Bishop, 2006, Yao et al., 2019].

2.1.3 Particle-based Variational Inference

Very recently, particle-based variational inference (ParVI) methods [Liu and Wang, 2016a, Liu et al., 2019, Liu, 2017] have been proposed that represent the variational distribution by a set of finite particles (samples) that are updated through a deterministic optimization process to approximate the posterior. ParVI methods achieve both asymptotic accuracy and computational efficiency. Liu and Wang [2016a], Liu [2017] proposed Stein Variational Gradient Descent (SVGD), that deterministically updates an empirical distribution of particles toward the target distribution via a series of sequentially constructed smooth maps. Each map defines a perturbation of the particles in the direction of steepest descent towards the target distribution under the KL-divergence metric. Liu et al. [2019] cast ParVI as a gradient flow on the space of probability measures $\mathcal{P}_2$ equipped with the W_2 Wasserstein metric, and proposed ParVI methods GFSD and GFSF by smoothing the density and the function respectively. Wasserstein variational gradient descent [Ambrogioni et al., 2018] transports particles to the target distribution by minimizing the optimal transport cost between the particles and the posterior distribution. We make use of ParVI in the following chapters, and seek to improve on its limitations. One such limitation is the fixed number of particles, i.e., the lack of ability to draw new samples beyond the initial set. Prior work, amortized SVGD [Wang and Liu, 2016], proposed to amortize the SVGD gradients to train a neural sam-

pler. While being flexible through drawing samples, amortized ParVI methods cannot match the asymptotic convergence behavior of full ParVI. Another limitation of ParVI is the restriction of the transport maps to an RKHS. The Fisher neural sampler [Hu et al., 2018, Grathwohl et al., 2020] lifts the function space of transportation maps to the space of $\mathcal{L}_2$ neural network functions, and performs minimax optimization to find the optimal perturbation. The benefit over VI and MCMC is that ParVI does not depend on the specific parameterization of the approximate posterior. In general, ParVI is also more efficient than MCMC methods due to its deterministic updates and gradient-based optimization.

2.1.4 Comparison to Proposed Work

The proposed work in this thesis are most similar to variational inference and particle-based variational inference. Our work differs fundamentally from VI in that we do not assume any specific parameterization of the approximate posterior. Additionally, the learned functions in our work are deterministic, and, therefore, do not require the derivation of exact gradients or Monte Carlo sampling. In one proposed contribution, we depend on an adversarial discriminator in the feature space of a generator network to produce diversity. In another, we derive a new method that minimizes the KL divergence between variational and target distributions that does not depend on the specific parameterization of the variational distribution. Both of our proposed approaches utilize implicit generative models, where the approximate posterior is obtained through sampling. Because of this ability to sample, our approaches also differ from particle-based variational inference.

2.2 Frequentist Distributions of Neural Networks

Without taking a probabilistic approach, we can instead use an ensemble of independently trained models to compute the epistemic uncertainty. Given that individual models are diverse with respect to the ensemble, i.e., make uncorrelated errors [Dietterich, 2000], we can average the predicted probabilities to compute the epistemic uncertainty in the ensemble. More recently, Lakshminarayanan et al. [2017a] proposed Deep Ensembles of neural networks, where adversarial training was applied to each member.

While naive ensemble methods remain a capable baseline, other SGD-based approximations have been proposed. These methods often focus on the efficiency of forming an ensemble, as the linear scaling with number of models is expensive for large tasks. Garipov et al. [2018] propose Fast Geometric Ensembling (FGE); using the insight that SGD modes have similar performance on training data yet generalize differently, FGE combines models at multiple local modes. Izmailov et al. [2018] approximates the FGE ensemble by performing stochastic weight averaging of SGD iterates. SWAG [Maddox et al., 2019] extends this work by parameterizing a low-rank Gaussian model with statistics of SGD iterates. While SGD-based approximations remain computationally efficient, their uncertainty estimation capabilities depend on the generalization ability of the particular SGD trajectory. Similar to SWA, Snapshot ensembles Huang et al. [2017] aggregate weights from multiple points in the optimization process of a single model to form an ensemble. A neural network that generates another neural network is sometimes called a *hypernetwork*. The original hypernetwork framework [Ha et al., 2016] describes a larger parent network that supervises the weight updates of a smaller child network. The two networks are trained jointly, and the child network can be repurposed for fast inference.

There are also data driven approaches to generating neural network parameters [Jaderberg et al., 2015, Jia et al., 2016]. In Dynamic filter networks, the convolution filter parameters of the main network are conditioned on the input data, receiving contextual scale and shift updates from an auxiliary network. Generative adversarial networks (GANs) are another possible method for sampling from a distribution of model parameters. For instance, Wang et al. [2018] propose to model the distribution captured by a Bayesian neural network trained with SGLD. Adversarial Posterior Distillation (APD) trains a GAN with parameter vectors sampled from a BNN during training. However, the training samples from the single SGLD chain are inevitably correlated, potentially reducing the diversity of generated networks.

2.2.1 Comparison to Proposed Work

Unlike most methods from the ensemble/SGD-approximation literature, our approaches make use of a generative model that allows us to acquire additional samples. All else

being equal, the ability to cheaply acquire additional samples allows our approaches to achieve a closer approximation to the posterior distribution. The exceptions to this case are SWAG and APD. Each uses a generative model to acquire posterior samples. In the case of SWAG, the generative model is a diagonal Gaussian and is limited by its parameterization, which consists of relatively few samples from the same trajectory through weight-space. Both SWAG and APD rely on a "main" optimization process (SGD/SGLD) to produce diverse iterates. This diversity is limited in practice by the fact that samples only represent one trajectory through weight space. Our approaches do not rely on SGD or SGLD for diversity, instead we construct explicit repulsive forces between samples to encourage diversity.

2.3 Exploration in Reinforcement Learning with Uncertainty

Efficient exploration remains a major challenge in deep reinforcement learning [Fortunato et al., 2017, Burda et al., 2018b, Eysenbach et al., 2018, Burda et al., 2018a] One practical guiding principle for efficient exploration is the reduction of the agent's epistemic uncertainty of the environment. By treating uncertainty as a novelty signal, RL agents can be encouraged to explore unknown parts of their environment.

2.3.1 Model-Based RL

As a high-level approach, a principled way to take actions under uncertainty is by using Thompson sampling [Thompson, 1933]. Thompson sampling, sometimes called probability matching, selects actions according to the probability that they are optimal with respect to a prior function. This sampling approach implicitly incorporates uncertainty, as poorly-understood actions are more likely to be taken under an uninformative prior [Agrawal and Goyal, 2012, Kaufmann et al., 2012]. Thompson sampling has been extended to the MDP setting in Strens [2000], Osband et al. [2013] under the name posterior sampling (PSRL). PSRL assumes a (Gaussian) prior over MDPs, and samples one MDP (conditioned on the history) from this distribution each episode. PSRL then solves for the optimal policy for the sampled MDP, and explores greedily. By sampling a single random MDP each episode, PSRL recovers the Thompson sampling behavior of probability matching, and converges to the true optimal policy as the agent collects

experience.

Without assuming an explicit model of the MDP, we can model the environmental dynamics, and consider the agent's uncertainty in this dynamics model. One way to leverage this uncertainty for exploration is to directly use some measure of the uncertainty as an intrinsic reward. Methods for constructing intrinsic rewards for exploration have become the subject of increased study. One well-known approach is to use the prediction error of an inverse dynamics model as an intrinsic reward [Pathak et al., 2017, Schmidhuber, 1991]. Schmidhuber [1991] and Sun et al. [2011] proposed using the learning progress of the agent as an intrinsic reward. Houthooft et al. [2016] formulate exploration as a variational inference problem, and use Bayesian neural networks (BNNs) to maintain the agent's belief over the transition dynamics. The BNN predictions are used to estimate a form of information gain called compression improvement [Pathak et al., 2019]. [Shyam et al., 2019] induce exploration through intrinsic rewards computed from an ensemble of dynamic models. The Renyi entropy or variance among the ensemble members in next-state predictions is used as the intrinsic reward.

2.3.2 Model-Free RL

To achieve efficient exploration without explicitly modeling the transition function, we can instead take a model-free approach and consider the uncertainty over a more abstract object like the value function. Osband et al. [2017] proposed randomized least-squares value iteration (RLSVI), which samples statistically plausible value functions and employs them for performing value iteration. This approach is extended by Osband et al. [2016] to deep Q-networks (DQN) [Mnih et al., 2013], where a bootstrap ensemble of DQNs is used to approximate the posterior of the value function. The predictions of the ensemble are used as an estimate of the agent's uncertainty, and exploration is performed by model averaging with a uniform prior. Furthermore, Osband et al. [2018] proposed to augment the predictions of a bootstrap DQN agent by adding the contribution from a fixed, untrained prior network. In a similar vein, O'Donoghue [2018] learns a representation of the cumulant generating function of the posterior over the agent's Q values, and derives upper and lower bounds of the agent's epistemic uncertainty with respect to its history. Bayesian deep Q learning has been further proposed in numerous other works,

for sample-efficient reinforcement learning [Azizzadenesheli et al., 2018, Janz et al., 2019, Clements et al., 2019]. As an alternative to the Bayesian treatment, Optimism in the Face of Uncertainty (OFU) approaches upper bound the value function, and act greedily according to the upper bound [Munos, 2014, Strehl et al., 2006, Szita and Szepesvári, 2010]. These algorithms are optimistic about future value in the sense that actions look more profitable than the data suggests. UCB1, UCRL, and UCRL2 [Auer et al., 2002, Ortner and Auer, 2007, Jaksch et al., 2010] provide an exploration bonus proportional to the visitation count for a chosen action in the bandit and MDP settings respectively. Due to the difficulty of estimating state visitation frequency in high dimensional MDPs, pseudo-count methods learn an approximate state count density and apply OFU [Ostrovski et al., 2017, Bellemare et al., 2016]. Proposed methods of estimating pseudo-counts include mixture models [Zhao and Tresp, 2019] and hashing [Tang et al., 2016].

2.3.3 Comparison to Proposed Work

In the model-based RL setting, our work proposes to use an intrinsic reward given by the uncertainty in the agent's dynamics model. Hence, our approach is close in spirit to Shyam et al. [2019] and Pathak et al. [2017], who propose a similar intrinsic reward formulation. The main difference is that our approach represents the dynamics model with an implicit generative model, as opposed to a naive ensemble. Given that RL agents are trained with many similar (non i.i.d.) trajectories and the dynamics may be arbitrarily simple, a naive ensemble of neural networks can quickly overfit – removing any epistemic uncertainty. For this reason, our method makes use of particle-based variational inference, and relies on the repulsive force between particles to ensure diversity in sampled models.

Chapter 3: Preliminaries

This chapter introduces the prerequisite concepts and the notation necessary for this thesis.

3.1 Implicit Generative Models

For every contribution in this thesis we define an implicit generative model that learns to sample from a target distribution of model parameters. To denote distributions, we define a probability space $(\Omega, \mathcal{F}, p)$, where Ω is a sample space, $\mathcal{F}$ is a σ-algebra, and $p(e)$ is the probability of any measurable event $e \in \mathcal{F}$ occurring. While $p : \mathcal{F} \to [0, 1]$ is a function on $\mathcal{F}$ that maps to probabilities, we may use other letters such as q to denote distributions of interest. Our main task is to learn a generator function $\boldsymbol{f} : \mathcal{Z} \to \mathcal{X}$ that transforms samples from a simple probability distribution Z on $\mathcal{Z} \subset \mathbb{R}^d$, to a data space $\mathcal{X} \subseteq \mathbb{R}^m$. We always parameterize $\boldsymbol{f}$ as a neural network with parameters $\boldsymbol{\eta}$. $q(\boldsymbol{x})$ is an approximating distribution, and the distribution $q_{\boldsymbol{f}}(\boldsymbol{x})$ is the implicit distribution represented by samples generated as $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})$. In principle, Z may be any probability distribution, but in practice we draw samples from a standard normal distribution for computational efficiency: $\boldsymbol{z} \sim Z \stackrel{d}{=} \mathcal{N}(0, \mathbf{I}_d)$.

3.1.1 Generative Adversarial Networks

Generative adversarial networks [Goodfellow et al., 2014] learn implicit generative models $\boldsymbol{f}_{\boldsymbol{\eta}}$ to sample from a target data density $p(\boldsymbol{x})$ on $X \subseteq \mathcal{X}$, given access to only a set of samples from $p(\boldsymbol{x})$. The main challenge in using implicit models is evaluating the distribution of samples drawn from $\boldsymbol{f}_{\boldsymbol{\eta}}$, which requires integrating over the input latent variable Z to compute $q_{\boldsymbol{f}}(\boldsymbol{x})$. This can be done when $\boldsymbol{f}_{\boldsymbol{\eta}}$ is invertible, but is in practice often intractable when $\boldsymbol{f}_{\boldsymbol{\eta}}$ is a neural network. The problem of intractability is further compounded when the target distribution is high dimensional, or otherwise lacks an explicit likelihood function. For instance, the distribution of natural images does not

have an explicit likelihood, making evaluation of generated samples difficult. GANs take a likelihood-free approach to circumvent both problems, by training a surrogate to the likelihood function, called a discriminator. The discriminator $D : \mathcal{X} \to \mathbb{R}$ evaluates samples $\boldsymbol{x} \in \mathcal{X}$, and outputs the probability that $\boldsymbol{x}$ was drawn from the target distribution, $p(y = 1|\boldsymbol{x})$. The generator and discriminator are optimized according to the following minimax objective,

$$\min_{\boldsymbol{f_\eta}} \max_{D} \mathbf{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbf{E}_{\boldsymbol{z}\sim Z}[\log 1 - D(\boldsymbol{f_\eta}(\boldsymbol{z}))]. \tag{3.1}$$

Therefore, the goal of the generator is to learn to transform noise samples into data samples that fool the discriminator, thus minimizing the loss on $\boldsymbol{f_\eta}$. The discriminator tries to maximize its ability to tell real images from generated samples. The system reaches optimality when the implicit distribution $q_{\boldsymbol{f}}(\boldsymbol{x})$ matches the target $p(\boldsymbol{x})$. In the original GAN formulation [Goodfellow et al., 2014], the training objective of the discriminator optimizes a lower bound on the likelihood $p(y|\boldsymbol{x})$, and reaches optimality for a fixed generator at $D(\boldsymbol{x}) = \left[\frac{p(\boldsymbol{x})}{p(\boldsymbol{x})+q_{\boldsymbol{f}}(\boldsymbol{x}))}\right]$. GANs have been refined and elaborated on extensively, and remain the state of the art in terms of image-generation quality. GANs are also known to be difficult to train, and suffer from pathologies like mode-collapse, where the generator maps large regions of Z to a single output [Arjovsky and Bottou, 2017].

3.1.2 Variational Autoencoders

Variational autoencoders (VAEs) [Kingma and Welling, 2013] are a family of regularized autoencoders that minimize the negative log-likelihood of the data under an implicit mapping $\boldsymbol{f_\eta} : Z \to \mathcal{X}$. VAEs specify a generative model $\boldsymbol{f_\eta}$ that is trained to map samples from a simple prior Z, to the data space $\mathcal{X}$, yielding the following density,

$$q_{\boldsymbol{f}}(\boldsymbol{x}) = \int_Z q_{\boldsymbol{f}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{z}}(\boldsymbol{z}) d\boldsymbol{z}. \tag{3.2}$$

Distinct from GANs, VAEs are not likelihood-free. They feature no discriminator function. VAEs employ an encoder network, $E : X \to Z$, to map data points to samples from the prior. We denote the conditional distribution of encoded samples as $p_E(\boldsymbol{z}|\boldsymbol{x})$. The

autoencoder structure provides a valid likelihood on $p(\boldsymbol{x})$ and shares the computationally amenable sampling procedure with GANs. To train $\boldsymbol{f_\eta}$ to approximate the target data distribution, VAEs minimize

$$\inf_E -\mathbf{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\mathrm{KL}(p_E(\boldsymbol{z}|\boldsymbol{x})||Z) - \mathbf{E}_{E(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{f}}(\boldsymbol{x}|\boldsymbol{z})\right]\right], \tag{3.3}$$

which is equivalent to maximizing a lower bound on the likelihood. The conditional distribution $\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{z})$ is parameterized by a neural network, but the prior Z is often assumed to be a factorized Gaussian distribution to make the KL term tractable. VAEs have enjoyed success for generative modeling in many domains, but are known to generate blurry examples when applied to image generation. [Zhao et al., 2017].

3.1.3 Wasserstein Autoencoders

Wasserstein Auto-encoders (WAE) [Tolstikhin et al., 2017] approach generative modeling from an optimal transport (OT) framework [Villani, 2008], to minimize the difference between the density given by an implicit generative model $q_{\boldsymbol{f}}(\boldsymbol{x})$ and the data distribution $p(\boldsymbol{x})$. As with VAEs, WAEs define an encoder model $E : X \to \mathcal{Z}$ that maps data samples to points in the latent space. The latent space is encouraged to match the prior distribution $Z \subset \mathcal{Z}$, but unlike VAEs, encoded samples do not parameterize the prior distribution. WAEs minimize the p-Wasserstein distance between the target data distribution and the learned distribution given implicitly through $\boldsymbol{f_\eta}$, $W_p(p(\boldsymbol{x}), q_{\boldsymbol{f}}(\boldsymbol{x}))$. Like VAEs, the discrepancy between the two distributions is minimized via a reconstruction cost on $\boldsymbol{f_\eta}(E(\boldsymbol{x}))$, as well as a regularizer to ensure that the conditional distribution $p_E(\boldsymbol{z}|\boldsymbol{x})$ is well distributed according to Z. In principle, the regularizer $\mathcal{D}_{\boldsymbol{z}}(Z, p_E(\boldsymbol{z}|\boldsymbol{x}))$ could be any distance between probability distributions. WAE uses either MMD or an adversarial discriminator trained with the GAN loss.

$$\inf_{\boldsymbol{f_\eta}} \inf_E \mathbf{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\mathbf{E}_{E(\boldsymbol{z}|\boldsymbol{x})}\left[c(\boldsymbol{x}, \boldsymbol{f_\eta}(\boldsymbol{z}))\right] + \lambda\mathcal{D}_{\boldsymbol{z}}(Z, p_E(\boldsymbol{z}|\boldsymbol{x})), \tag{3.4}$$

where c is any measurable cost function measuring the error between the data $\boldsymbol{x}$ and reconstruction $\boldsymbol{f_\eta}(\boldsymbol{z})$, and λ is a hyperparameter. The key difference between WAE and VAE is that WAE admits deterministic encoders. This means that while the distribution

$p_E(\boldsymbol{z}|\boldsymbol{x})$ is encouraged to match $p_{\boldsymbol{z}}$, it is not explicitly equal. The ability to use degenerate (deterministic) encoders means that we are less likely to map different inputs to the same output. In chapter 4, we propose a method that uses a similar regularization term as WAE to ensure that our generated samples are diverse, by placing an adversarial discriminator in the feature space of a generative model.

3.2 Bayesian Inference

The majority of our contributions are related to the uncertainty in a learned distribution of neural network parameters. This notion of uncertainty brings us firmly into the territory of Bayesian inference. At a high level, Bayesian inference is a way to make inferences about the conditional probabilities of outcomes that result from uncertainty. Under the Bayesian view, we are primarily concerned with computing the posterior probability of some hypothesis $\mathcal{H}$, conditioned on the evidence $\mathcal{D}$. The posterior probability depends on the likelihood of the data under this hypothesis $p(\mathcal{D}|\mathcal{H})$, and the observed evidence $p(\mathcal{D})$. The subjective Bayesian paradigm also specifies our belief about $\mathcal{H}$ through a prior probability $p(\mathcal{H})$. We can relate the posterior probability of $\mathcal{H}$ to the data $\mathcal{D}$ through Bayes rule:

$$p(\mathcal{H}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H})p(\mathcal{H})}{p(\mathcal{D})}, \text{ or equivalently, } \textit{posterior} = \frac{\textit{likelihood} \ \times \ \textit{prior}}{\textit{evidence}}$$

Any question about conditional probabilities can be a Bayesian query, for example:

- What is the probability it will rain the day after it was sunny?
- How likely is my dog to want to go back outside within 5 minutes of letting her in?
- What is the probability of a set of model parameters, given the observed data?

Naturally, we are most concerned with the third point. Specifically, we are interested in what Bayes theorem says about the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, where $\boldsymbol{\theta}$ are model parameters.

3.2.1 Particle-based Variational Inference

Particle-based variational inference (ParVI) was proposed as a family of non-parameteric methods for fitting a target distribution $p(\boldsymbol{x})$. In contrast with standard variational inference, ParVI does not assume an analytic form for the approximate posterior. Instead, ParVI considers the empirical distribution, formed by a set of samples (particles) $q(\boldsymbol{x})$. The goal of ParVI is to learn this empirical distribution such that it converges in distribution to the target distribution. [Liu and Wang, 2016b, Liu et al., 2019, Hu et al., 2018]. ParVI methods deterministically update an empirical distribution of particles toward the target distribution via a series of sequentially constructed smooth maps. Each map defines a perturbation of the particles in the direction of steepest descent towards the target distribution under the KL-divergence metric.

The canonical ParVI algorithm, Stein variational gradient descent (SVGD), represents $q(\boldsymbol{x})$ by a set of particles $\{\boldsymbol{x}_i\}_{i=1}^m$, that are updated iteratively by,

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon \boldsymbol{\phi}^*(\boldsymbol{x_i}), \tag{3.5}$$

where ϵ is a step size, and $\boldsymbol{\phi}^* : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field on the space of particles that corresponds to the optimal direction to perturb particles:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi} \in \mathcal{F}} \left\{ \frac{d}{d\epsilon} \mathrm{KL}(q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x}) \| p(\boldsymbol{x})) \Big|_{\epsilon=0} \right\},$$

where $q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x})$ is the implicit distribution represented by particles updated by (3.5). When $\mathcal{F}$ is chosen to be the unit ball of some RKHS $\mathcal{H}$ with kernel function $k(\cdot,\cdot)$, SVGD gives the following closed form solution for $\boldsymbol{\phi}^*$:

$$\boldsymbol{\phi}^*(\boldsymbol{x}) = \mathbf{E}_{\boldsymbol{x}' \sim q} \left[\nabla_{\boldsymbol{x}'} \log p(\boldsymbol{x}') k(\boldsymbol{x}', \boldsymbol{x}) + \nabla_{\boldsymbol{x}'} k(\boldsymbol{x}', \boldsymbol{x}) \right]. \tag{3.6}$$

This update reduces to standard MAP ascent when $q(\boldsymbol{x})$ consists of just a single particle, and the kernel gradient $\nabla_{\boldsymbol{x}} k(\boldsymbol{x}, \boldsymbol{x}) = 0$. This means that SVGD can be used to interpolate between standard supervised training with a single particle (fitting the posterior mean), and a method for Bayesian inference (fitting the full distribution).

The particle update given in (3.6) can be understood as consisting of two terms: a likeli-

hood term and a repulsive force. The likelihood $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ naturally drives the particles towards regions of high probability under $p(\boldsymbol{x})$, by following an averaged gradient direction, that is composed of contributions from all particles in the system. The repulsive force penalizes similar particle pairs with large $k(\boldsymbol{x}, \cdot)$, ensuring that all particles do not collapse to a single mode.

One downside of SVGD is that while SVGD can be viewed as a way to draw samples from $p(\boldsymbol{x})$, we have no way of drawing additional samples once the SVGD chain has converged. This limitation makes it hard to apply ParVI algorithms like SVGD to tasks like image generation, where we wish to draw a large number of samples. To address this issue, Wang and Liu [2016] proposed amortized SVGD to learn a sampling function for the SVGD approximate posterior. Critically, amortised SVGD does not assume a parametric form of this approximate posterior, but makes use of the particle update (3.6) to train an implicit generative model $\boldsymbol{f}_{\boldsymbol{\eta}}$. Amortized SVGD first samples particles from $\boldsymbol{f}_{\boldsymbol{\eta}}$ and then back-propagates the particle gradients (3.6) through the generator to update the generator parameters. Let $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}),\ \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, I_d)$ be the particle generating process, where $\boldsymbol{f}_{\boldsymbol{\eta}}$ is the generator parameterized by $\boldsymbol{\eta}$. Amortized SVGD updates $\boldsymbol{\eta}$ by

$$\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \epsilon \sum_{i=1}^{m} \frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i)}{\partial \boldsymbol{\eta}} \boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i)), \tag{3.7}$$

where $\boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i))$ is computed by (3.6). In chapter 5 we make use of amortized SVGD for implicit generative modeling in reinforcement learning. In chapter 6 we examine the limitations of the amortized SVGD approximation, and develop a new method for generative particle-based variational inference that addresses these limitations. Further improvements are also explored in chapter 7.

3.2.2 Bayesian Neural Networks

In chapters 5, 6, 7, 8, we perform variational inference via the implicit distribution $q(\boldsymbol{\theta})$. The main objective we optimize is as follows:

$$\min_{\boldsymbol{f}_{\boldsymbol{\eta}}} \mathrm{KL}(q(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})}||p(\boldsymbol{\theta})),$$

where $p(\boldsymbol{\theta})$ is the target distribution, and $\boldsymbol{z}$ is a prior distribution over weights. For BNN inference, we are interested in sampling from the posterior distribution of model parameters conditioned on the data: $p(\boldsymbol{\theta}|D)$. To do this, we train $\boldsymbol{f_\eta}$ to generate parameter vectors $\boldsymbol{\theta}$ that we can evaluate through the likelihood function $\log p(\boldsymbol{\theta})$. Given that we want to model $p(\boldsymbol{\theta}|D)$, we train the generator $\boldsymbol{f_\eta}$ to output $\boldsymbol{\theta}$ such that every sample from $\boldsymbol{f_\eta}$ achieves low log-loss under the data $p(\boldsymbol{\theta})$, $\log p(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{f_\eta}(\boldsymbol{z})}$ is small.

We are often interested in evaluating the "uncertainty" with respect to $\boldsymbol{f_\eta}$. Because epistemic uncertainty is reduced by observing more data, we can measure the predictive uncertainty by marginalizing over $\boldsymbol{f_\eta}$. Given that $\boldsymbol{f_\eta}$ is just a generative model for parameter vectors, we define a *prediction network*, consisting of the architecture F parameterized by samples from $\boldsymbol{f_\eta}$. The prediction network makes predictions with respect to the observed data, providing a way to evaluate samples from $\boldsymbol{f_\eta}$.

3.2.3 Definition of Epistemic Uncertainty

We want to compute reliable estimates of the epistemic uncertainty U from the distribution of samples from $\boldsymbol{f_\eta}$. In practice, the statistic we compute to represent the uncertainty depends on the downstream evaluations of the sampled $\boldsymbol{\theta}$. We denote U_{clf} as the uncertainty that we compute when $\boldsymbol{\theta}$ parameterize classification functions, and U_{reg} denotes the uncertainty when $\boldsymbol{\theta}$ parameterizes regression functions. If the observed data D is a joint distribution of data points and associated labels: $(X, Y) \subseteq (\mathcal{X}, \mathcal{Y})$, then we evaluate samples from $\boldsymbol{f_\eta}$ as $F_{\boldsymbol{f_\eta}(\boldsymbol{z})}(\boldsymbol{x})$, where $\boldsymbol{x} \sim X$.

When $F_{\boldsymbol{f_\eta}(\boldsymbol{z})}$ is a K-way classification function, the predictive uncertainty U_{clf} of $\boldsymbol{f_\eta}$ is given by the Shannon entropy of the predicted probability distribution over classes. We compute U_{clf} with the following Monte Carlo integration:

$$U_{clf}(\boldsymbol{f_\eta}, X) = -\mathbf{E}_{\boldsymbol{z}}\left[\mathbf{E}_{\boldsymbol{x}\sim X}\left[\sum_{c=1}^{K} \sigma F_{\boldsymbol{f_\eta}(\boldsymbol{z})}(\boldsymbol{x})_c \log \sigma F_{\boldsymbol{f_\eta}(\boldsymbol{z})}(\boldsymbol{x})_c\right]\right]. \tag{3.8}$$

Where σ is the softmax function that maps the outputs of $F_{\boldsymbol{f_\eta}(\boldsymbol{z})}$ to the probability simplex, $\sigma(F(x))_i = \frac{e^{F(x)_i}}{\sum_{j=1}^{K} e^{F(x)_j}}$ for $i = 1 \ldots K$.

When $F_{\boldsymbol{f_\eta}(\boldsymbol{z})}$ is a regression function, the epistemic uncertainty U_{reg} is given by the

predictive variance of $F_{\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})}(X)$:

$$U_{reg}(\boldsymbol{f}_{\boldsymbol{\eta}}, X) = \mathbf{E}_{\boldsymbol{z}} \left[\sum_{i=1}^{n} \mathrm{Var} \left[\sigma F_{\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})}(x_i) \right] \right]. \tag{3.9}$$

3.3 Reinforcement Learning

Reinforcement learning (RL) is the problem of learning to maximize returns via sequential interactions with an unknown environment. Superhuman performance on challenging tasks such as the games of Atari [Mnih et al., 2013], Go [Silver et al., 2016], and StarCraft II [Vinyals et al., 2019], has been achieved with RL. While these represent large-scale successes, there is still room for improvement in the fundamental mechanisms of RL. In particular, there is a risk-reward trade-off problem of balancing *exploitation*, where an agent takes actions it knows will immediately give reward, and *exploration*, where an agent bets the possibility of downstream reward on risky actions. Hence, exploration is important to allow the agent to reveal the reward structure of the environment, without focusing on well-understood yet relatively low-return behaviors. The cost of exploration is lower short-term performance, but paying the upfront cost for exploration often results in higher long-term return in scenarios where the reward is sparse.

An inefficient agent may explore by taking random actions at each state. By doing this, the agent may happen on some good behavior, but in complex environments, finding high-reward behavior can take exponentially many time-steps. For example, a rescue agent tasked with finding lost climbers in a cave benefits from efficient, thorough exploration. Rescue agents that explore via a random walk may provably explore the entire cave, but this kind of exploration is very inefficient. Therefore, to efficiently learn a good policy, the agent should maximize return while minimizing the time it takes to find this reward. Many current RL methods rely on random actions to explore, and take millions of examples to converge to a high-reward policy.

3.3.1 Problem Statement

Here we formalize the notions of the environment and agent. Consider a Markov Decision Process (MDP) represented as $(\mathcal{S}, \mathcal{A}, T, r, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action

space, and $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the unknown dynamics model, specifying the probability of transitioning to the next state $s' \in \mathcal{S}$ from the current state $s \in \mathcal{S}$ by taking the action $a \in \mathcal{A}$, as $p(s'|s, a)$. We denote the reward function as $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. $\rho_0 : \mathcal{S} \rightarrow [0, 1]$ is a probability distribution over initial states. A policy is a function $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ that outputs a distribution over actions for a given state s. We focus on finite-horizon MDPs that terminate at time-step L with discount rate γ. We denote the agent's history by H_t, that is, all the experience the agent has collected until time t.

The goal of the agent is to maximize its total expected discounted reward, by learning an optimal policy π^* that maps states to the best possible actions for a given time-step.

3.3.2 Definition of Epistemic Uncertainty

In chapter 5, we propose a method for efficient exploration by leveraging the agent's uncertainty in the environment. We study the model-based RL setting, where the agent learns an approximation to the environment dynamics T. Given a model of the dynamics, an agent can *plan to explore* by identifying regions of the state space where the predicted dynamics do not match the observations. We represent the dynamics as a neural network $F : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, parameterized by samples from an implicit generative model $\boldsymbol{f_\eta} : Z \rightarrow \Theta$. We compute the uncertainty as follows,

$$U(\boldsymbol{f_\eta}, s, a) = \mathbf{E}_{\boldsymbol{z}} \left[\mathrm{Var} \left[F_{\boldsymbol{f_\eta}(\boldsymbol{z})}(s, a) \right] \right]. \tag{3.10}$$

We use the uncertainty in $\boldsymbol{f_\eta}$ as a proxy for state-visitation frequency. If the uncertainty in $\boldsymbol{f_\eta}$ given a state-action pair is low, then we assume the agent has explored there already. Conversely, if the uncertainty is high, then we encourage the agent to visit this state-action pair again.

Chapter 4: Implicit Generative Modeling with HyperGAN

4.1 Introduction

In our first contribution, we build a model from which we can sample sets of parameters for neural networks. The samples should perform approximately equally well on the training data, while giving diverse predictions on inputs far from the training distribution. The goal is to be able to extract uncertainty information from the predictions made by our model. If the predictions of our model agree, then we trust that our model is confident in its prediction. However, if the predictions of samples are diverse on outlier data, then a set of samples in disagreement is indicative of that example being an outlier. In real-world applications, we believe its reasonable that safety-critical decisions should be determined by models that have the ability to be uncertain about their predictions. Models that are always confident yet only often correct, may be dangerous to use in applications where safety is at risk.

To construct a sampler for neural network parameters, we choose a generative architecture where the weights of the generator combined with a sampling distribution form an implicit distribution over the generated parameters. One of the issues in generating weights is the connectivity of the network. Namely, the output of the previous layer becomes the input of the next layer, hence the network weights must be correspondent to yield valid results. In our approach, we sample from a simple multi-dimensional isotropic Gaussian distribution, and propose to transform this sample into multiple different vectors. We call this procedure a *mixer* since it introduces correlations to the otherwise independent components of the noise sample. Then each random vector is used to generate all the weights within one layer of a deep network. The generator is trained with conventional maximum likelihood (classification/regression) with respect to the generated weights, and an adversarial regularization encourages diversity in the samples. In this way, we generate networks that are much larger than the dimensionality of the latent code, making our approach capable of generating all the weights of a deep network

with a single GPU. As an example, in our experiments on CIFAR-10 we start from a 256-dimensional latent vector and generate all $50,000+$ weights in one pass.

Somewhat surprisingly, with just this approach we can already generate complete, multi-layer convolutional networks which do not require additional fine-tuning. We are able to easily sample many well-trained networks from the generator which each achieve low loss on the dataset the generative model is trained on. Moreover, we propose diversity constraints that result in sampled models significantly more diverse than traditional training with multiple random starts, dropout, or adding scaling factors to the weights. We show through a variety of experiments that populations of diverse networks sampled from our model are able to generate reasonable uncertainty estimates by calculating the entropy of the predictive distribution of sampled networks. Our uncertainty estimates allow us to detect out of distribution samples as well as adversarial examples. Our method is straightforward, as well as easy to train and sample from.

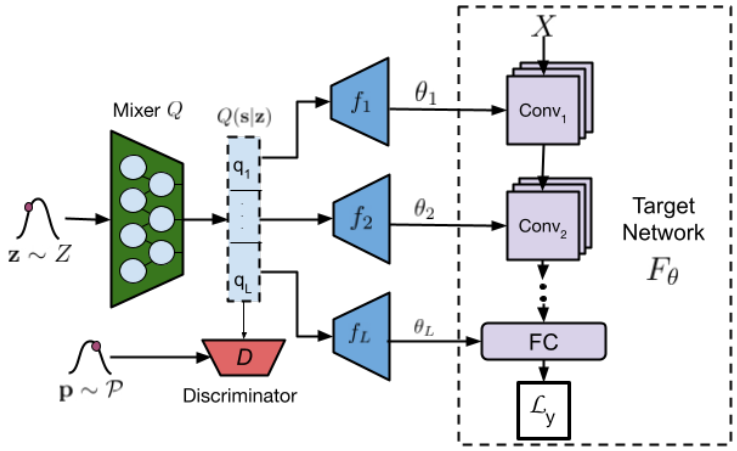


Figure 4.1: HyperGAN architecture. The mixer transforms $\boldsymbol{z} \sim Z$ into latent codes $\{q_1, \ldots, q_L\}$. The L generators each transform the latent subvector q_i into the parameters of the corresponding layer in the target network. The discriminator D forces $Q(\boldsymbol{s}|\boldsymbol{z})$ to be well-distributed and according to the distribution $\mathcal{P}$.

4.2 HyperGAN

Taking note from the original hypernetwork framework for generating neural networks [Ha et al., 2016], we coin our approach HyperGAN. The key idea for HyperGAN is to

leverage a GAN-style sampling approach to directly generate discriminative networks. To do this, the straightforward approach would be to acquire a large set of trained neural networks and use those as training data to the GAN Wang et al. [2018]. However, a large collection of neural networks would be extremely costly to build. Another approach, proposed in Wang et al. [2018] is to train a single Bayesian neural network with SGLD. Intermediate samples from the SGLD chain are then aggregated to form a training set of functions, used as examples to train a GAN. However, such a dataset consists of highly correlated samples, without the necessary diversity to approximate the full distribution.

Instead, we propose to directly optimize the supervised learning objective instead of focusing on reconstruction error on the training for the generator, as in normal GANs. Similar to a standard GAN, we start by drawing a random sample $\boldsymbol{z} \sim Z = \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, where $\mathbf{0}$ is an all-zero vector and $\mathbf{I}_d$ is a $d \times d$ identity matrix. The idea is that this random sample would provide enough diversity to generate diverse parameter vectors. If we can maintain such diversity while optimizing on the supervised learning objective, we can generate networks that all optimize the loss function well, but are sufficiently diverse because they are generated from different Gaussian random vectors.

Figure 5.1 shows the HyperGAN architecture. We begin by defining a neural network as a function $F_{\boldsymbol{\theta}} : X \to Y$ parameterized by $\boldsymbol{\theta}$, consisting of a given architecture with L layers, that maps data points $\boldsymbol{x} \in X$ to corresponding labels $y \in Y$. We further suppose that there exists a distribution of parameters Θ, where samples $\boldsymbol{\theta} \sim \Theta$ achieve arbitrarily high performance on the given task. Under this view, standard training of neural networks amounts to drawing a single sample from Θ. To learn to draw samples from Θ, we modify a standard GAN to output parameter vectors. Distinct from the architecture of the standard GAN, we propose a *Mixer* $Q : Z \to \mathcal{S}$ which is a fully-connected network that maps noise samples $\boldsymbol{z} \sim Z$ to a mixed latent space $\mathcal{S} \subset \mathbb{R}^{Ld}$. The mixer is motivated by the observation that weight parameters between network layers must be strongly correlated as the output of one layer needs to be the input to the next one. Hence, it is likely that some correlations are also needed in the latent vectors that generate those weight parameters. Our Ld-dimensional mixed latent space $Q(\boldsymbol{z})$ contains vectors that are all correlated, which we then partition into L layer embeddings $[q_1, \ldots, q_L]$, each being a d-dimensional vector. Finally, we use L

parallel generators $\boldsymbol{f} = \{f_1(q_1) \dots f_L(q_L)\}$ to generate the parameters $\boldsymbol{\theta}$ for all layers in F. This approach is also memory efficient since the extremely high dimensional space of the weight parameters are now separately connected to multiple latent vectors, instead of fully-connected to the latent space.

After generating a parameter vector $\boldsymbol{\theta}$, we can evaluate the new model $F_{\boldsymbol{\theta}}(\boldsymbol{x})$ on the training set. We define an objective which minimizes the error of generated parameters with respect to a task loss $\mathcal{L}$:

$$\inf_{\boldsymbol{f}_{\boldsymbol{\eta}}, Q} \mathbf{E}_{\boldsymbol{z} \sim Z} \mathbf{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim (X, Y)} \left[\mathcal{L}(F(\boldsymbol{x}; \boldsymbol{f}(Q(\boldsymbol{z}))), \boldsymbol{y}) \right] \tag{4.1}$$

At each training step we generate a different network $\boldsymbol{f}(Q(\boldsymbol{z}))$ from a noise sample $\boldsymbol{z} \sim Z$, and then evaluate the loss function on a mini-batch from the training set. The resulting loss is backpropagated through the generators until $F_{\boldsymbol{\theta}}$ minimizes the target loss $\mathcal{L}$.

The main concern about directly optimizing the formulation in (4.1) would be that the codes sampled from $Q(\boldsymbol{z})$ may collapse to the maximum likelihood estimate, when $\mathcal{L}$ is a log-likelihood. This means that the generators may learn a very narrow approximation of Θ. On the other hand, one can think of the training process as simultaneously starting from many starting points (since we sample different $\boldsymbol{z}$ for each mini-batch) and attempting to obtain an optimum on all of them. Because deep networks are extremely overparameterized and it is often the case that many global optima exist [Choromanska et al., 2015], the optimization may indeed converge to different optima from different random $\boldsymbol{z}$.

The mixer may make the training process easier by building in the required correlations into the latent code, hence improve the chance the optimization converges to different optima from different random $\boldsymbol{z}$. Similar to Wasserstain autoencoders [Tolstikhin et al., 2017], we ensure that the parameters are well distributed by adding an adversarial constraint on the mixed latent space $\mathcal{D}(Q(\boldsymbol{z}))$ and encourage it to not deviate too much from a Gaussian prior $\mathcal{P}$. This constraint is closer to the generated parameters and ensures that $Q(\boldsymbol{z})$ itself does not collapse to always outputting the same latent code. With this

we arrive at the HyperGAN objective:

$$\inf_{\boldsymbol{f},Q} \mathbf{E}_{\boldsymbol{z}\sim Z} \mathbf{E}_{(\boldsymbol{x},y)\sim(X,Y)} \left[\mathcal{L}(F(\boldsymbol{x};\boldsymbol{f}(Q(\boldsymbol{z}))), y)\right] - \beta\mathcal{D}(Q(\boldsymbol{z}), \mathcal{P}) \tag{4.2}$$

Where β is a hyperparameter, and $\mathcal{D}$ is the regularization term which penalizes the distance between the prior and the distribution of latent codes. In practice $\mathcal{D}$ could be any distance function between two distributions. We choose to parameterize $\mathcal{D}$ as a discriminator network D that outputs probabilities, and use the adversarial loss [Goodfellow et al., 2014] to approximate $\mathcal{D}(Q(\boldsymbol{z}), \mathcal{P})$.

$$\mathcal{D} := -\sum_{i=1}^{N} (\log D(p_i) + \log(1 - D(q_i))) \tag{4.3}$$

Note that while $\mathcal{P}$ and Z are both multivariate Gaussians, they are of different dimensionality and covariance.

Note that we find it difficult to learn a discriminator in the output (parameter) space because the dimensionality is high and there is no structure in $\boldsymbol{\theta}$ to be utilized as in images (where CNNs can be trained). Our experiments show that regularizing in the latent space works well, which matches results from recent work in implicit generative models [Tolstikhin et al., 2017].

This framework is general and can be adapted to a variety of tasks and losses. In this work, we show that HyperGAN can operate in both classification and regression settings. For multi-class classification, the generators and mixer are trained with the cross entropy loss function:

$$\begin{aligned} \mathcal{L}_H &= \frac{1}{N}\sum_{i=1}^{N} y_i \log\left(F(x_i;\boldsymbol{\theta})\right) \\ \boldsymbol{\theta} &= \{\boldsymbol{f}_1(q_1), \ldots, \boldsymbol{f}_L(q_L)\} \end{aligned} \tag{4.4}$$

For regression tasks we replace the cross entropy term with the mean squared error

(MSE):

$$\begin{aligned} \mathcal{L}_{MSE} &= \frac{1}{N}\sum_{i=1}^{N}(y_i - F(x_i;\boldsymbol{\theta}))^2 \\ \boldsymbol{\theta} &= \{\boldsymbol{f}_1(q_1), \ldots, \boldsymbol{f}_L(q_L)\} \end{aligned} \tag{4.5}$$

4.2.1 Learning to Generate without Explicit Samples

In generative models such as GAN [Goodfellow et al., 2014] or WAE [Tolstikhin et al., 2017], it's necessary to have a collection of data points to train with, which come from the distribution that is being estimated, so that such training examples can be sampled from the generator. HyperGAN does not have a given set of samples to train with. Instead, it optimizes a supervised learning objective such as maximum likelihood. To draw a connection between this objective and the traditional reconstruction objective in GAN, we note that after training, a generated $\boldsymbol{\theta_f}$ represents the maximum likelihood estimate of $F(\boldsymbol{x};\boldsymbol{\theta})$, which has a well-known link to KL-divergence:

$$\begin{aligned} & \inf_{\boldsymbol{\theta_f}} D_{KL}(p(\boldsymbol{x}|\boldsymbol{\theta})||p(\boldsymbol{x}|\boldsymbol{\theta_f})) \\ = & \inf_{\boldsymbol{\theta_f}} \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta_f})}\left[\log p(\boldsymbol{x}|\boldsymbol{\theta}) - \log p(\boldsymbol{x}|\boldsymbol{\theta_f})\right] \\ = & \inf_{\boldsymbol{\theta_f}} \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta_f})}\left[-\log p(\boldsymbol{x}|\boldsymbol{\theta_f})\right] \end{aligned} \tag{4.6}$$

(4.6) shows that by minimizing the error of the MLE on the log-likelihood, we are indeed minimizing the KL divergence between the unknown true parameter distribution Θ and the distribution of generated samples $\Theta_{\boldsymbol{f}}$. Hence, we can view HyperGAN also as approximating a target distribution of the neural network parameters. However, HyperGAN only assumes the target distribution exists, and update our approximation $\Theta_{\boldsymbol{f}}$ via maximum likelihood to better match the unknown Θ.

4.3 Experiments

We conduct a variety of experiments to test HyperGAN's ability to achieve both high accuracy and obtain accurate uncertainty estimates. First we show classification performance on both MNIST and CIFAR-10 datasets. Next we examine HyperGAN's capability to learn the variance of a simple 1D dataset. We then perform experiments on anomaly detection by testing HyperGAN's ability to discriminate between training data and out-of-distribution examples. For models trained on MNIST we test on the notMNIST dataset. For CIFAR-10 experiments we train on the first 5 classes of CIFAR-10 (airplane, automobile, bird, cat, deer), then test whether the classifier can detect out-of-distribution images from the 5 remaining classes not shown during training. Finally, we test our robustness to adversarial examples as extreme cases of out-of-distribution data.

In the following experiments we compare against APD [Wang et al., 2018], MNF [Louizos and Welling, 2017], and MC Dropout [Gal and Ghahramani, 2016]. We also evaluate standard ensembles as a baseline. The target architecture used is the same across all approaches. For APD we train both MNIST and CIFAR networks with SGLD for 100 epochs, saving intermediate parameters as a training set. We then train a GAN on the SGLD samples given the architectures and hyperparameters specified in [Wang et al., 2018] until convergence. For MNF we use the code provided in [Louizos and Welling, 2017], and we train the model for 100 epochs. MC dropout is trained and sampled from as described in [Gal and Ghahramani, 2016], with a dropout rate of $\pi = 0.5$. In all experiments, unless otherwise stated, we draw 100 networks from the posterior to form the predictive distribution for each approaches.

4.3.1 Implementation Details

HyperGAN models for MNIST and CIFAR-10 take a 256 dimensional sample of Z as input, but have different sized mixed latent spaces. The HyperGAN for the MNIST setting consists of three weight generators, each using a 128 dimensional latent vector as input. The target network for the MNIST experiments is a small two layer convolutional network followed by 1 fully-connected layer, using leaky ReLU activations and 2x2 max pooling after each convolutional layer. Our HyperGAN trained on CIFAR-10 use 5

weight generators and latent dimensonality of 256. The target architecture for CIFAR-10 consists of three convolutional layers, each followed by leaky ReLU and 2x2 max pooling, with 2 fully connected layer after the convolutional layers.

The mixer, generators, and discriminator are each a 2 layer MLP with 512 units in each layer and ReLU nonlinearity. We found that larger networks offered little performance benefit, and ultimately hurt scalability. It should be noted that larger networks do not harm the capability of HyperGAN to model the target distribution. We trained our HyperGAN on MNIST using less than 1.5GB of memory on a single GPU, while CIFAR-10 used just 4GB, making HyperGAN surprisingly scalable.

	HyperGAN			Ensemble			APD		
	Conv1	Conv2	Linear	Conv1	Conv2	Linear	Conv1	Conv2	Linear
Mean	7.49	51.10	22.01	27.05	160.51	5.97	2.63	5.01	17.4
σ	1.59	10.62	6.01	0.31	0.51	0.06	0.22	0.41	1.43

Table 4.1: HyperGAN diversity. 2-norm statistics on the layers of a population of network parameters sampled from HyperGAN, compared to a 10-network ensemble, as well as 10 samples from APD. Both HyperGAN and the standard models were trained on MNIST to 98% accuracy. Its easy to see that HyperGAN generates more diverse networks under this metric

4.3.2 Classification Accuracy and Diversity

First we evaluate the classification accuracy of HyperGAN on MNIST and CIFAR-10. Classification serves as the entrance into our other experiments, as the distribution we want to learn is over parameters which can effectively solve the classification task. We test with both single network samples, and ensembles. For our ensembles we average predictions from M sampled models with the scoring rule $p(y|\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} p_m(y \mid \boldsymbol{x}, \boldsymbol{\theta}_m)$. It should be noted that we did not perform fine tuning, or any additional training on the sampled networks. The results are shown in Table 4.2. We generate ensembles of different sizes and compare against APD [Wang et al., 2018], MNF [Louizos and Welling, 2017], and MC dropout [Gal and Ghahramani, 2016]. For each method we draw 100 samples from the learned posterior to generate the predictive distribution and use the above averaging to compute the classification score.

Method	MNIST	MNIST 5000	CIFAR-5	CIFAR-10	CIFAR-10 5000
1 network	98.64	96.69	84.50	76.32	76.31
5 networks	98.75	97.24	85.51	76.84	76.41
10 networks	99.22	97.33	85.54	77.52	77.12
100 networks	**99.31**	**97.71**	**85.81**	**77.71**	**77.38**
APD	98.61	96.35	83.21	75.62	75.13
MNF	99.30	97.52	84.00	76.71	76.88
MC Dropout	98.73	95.58	84.00	72.75	70.10

Table 4.2: Classification performance of HyperGAN on MNIST and CIFAR-10. CIFAR-5 refers to a dataset with only the first 5 classes of CIFAR-10. MNIST 5000 and CIFAR-10 5000 refers to training on only 5000 examples of MNIST and CIFAR-10, respectively, which has been used in prior work (e.g. Louizos and Welling [2017])

In Table 4.1 we show some statistics of the networks generated by HyperGAN on MNIST. We note that HyperGAN can generate very diverse networks, as the variance of network weights generated by the HyperGAN is significantly higher than standard training from different random initializations, as well as APD. More insights on the diversity of HyperGAN samples can be found in Section 4.3.5.

4.3.3 1-D Regression

We next evaluate the capability of HyperGAN to fit a simple 1D function from noisy samples and generate reasonable uncertainty estimates on regions with few training samples. This dataset was first proposed by [Hernández-Lobato and Adams, 2015], and consists of a training set of 20 points drawn uniformly from the interval $[-4, 4]$. The targets are given by $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 3^2)$. We used the same target architecture as in [Hernández-Lobato and Adams, 2015] and [Louizos and Welling, 2017]: a one layer neural network with 100 hidden units and ReLU nonlinearity trained with MSE. For HyperGAN we use two layer generators, and 128 hidden units across all networks. Because this is a small task, we use only a 64 dimensional latent space.

Figure 4.2 shows that HyperGAN clearly learns the target function and captures the variation in the data. Furthermore, sampling more (100) networks to compose a larger ensemble improves the predicted uncertainty in regions with few training examples.

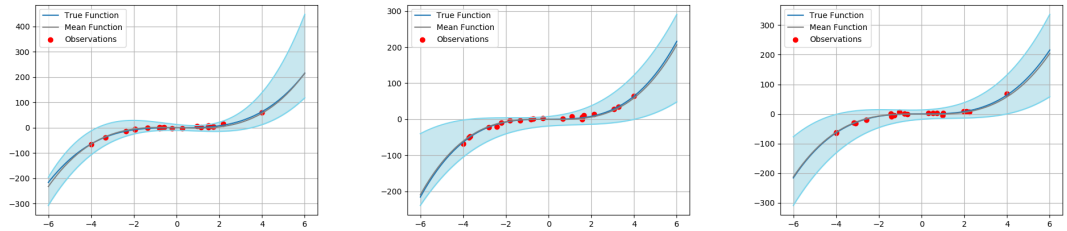


Figure 4.2: Results of HyperGAN on the 1D regression task. From left to right, we plot the predictive distribution of 10, 100, and 256 sampled models from a trained HyperGAN. Within each image, the blue line is the target function x^3, the red circles show the noisy observations, the grey line is the learned mean function, and the light blue shaded region denotes ± 3 standard deviations

4.3.4 Out of Distribution Detection

To measure the uncertainty given on out of distribution data, we measure the total predictive entropy given by HyperGAN-generated ensembles. For MNIST experiments we train a HyperGAN on the MNIST dataset, and test on the notMNIST dataset: a 10-class set of 28x28 grayscale images depicting the letters A - J. In this setting, we measure the entropy in the predictive distribution as our uncertainty statistic. On inlier data, the entropy of HyperGAN's predictive distribution be low, with all networks predicting the true class with high confidence. While on out-of-distribution data, we want to have disagreement across the models drawn from HyperGAN, which corresponds to a high entropy predictive distribution. As with MNIST, we test our CIFAR-10 model by training on the first 5 classes, and using the latter (unseen) 5 classes as out of distribution examples. To build an estimate of the predictive entropy we sample multiple networks from HyperGAN, evaluate them on each example, and measure their predictive entropy. We compare our uncertainty measurements with those of APD, MNF, MC dropout, and standard ensembles. Unless otherwise noted, we compute the entropy based on 100 networks.

Fig. 4.3 shows that HyperGAN is overall less confident on outlier samples than other approaches on the notMNIST dataset. Standard ensembles overfit considerably, as expected. Furthermore, table 4.1 shows that the diversity of standard ensembles is quite low. Fig. 4.4 shows similar behavior on CIFAR-10. Hence HyperGAN can better separate inliers from outliers when out-of-distribution examples are present.

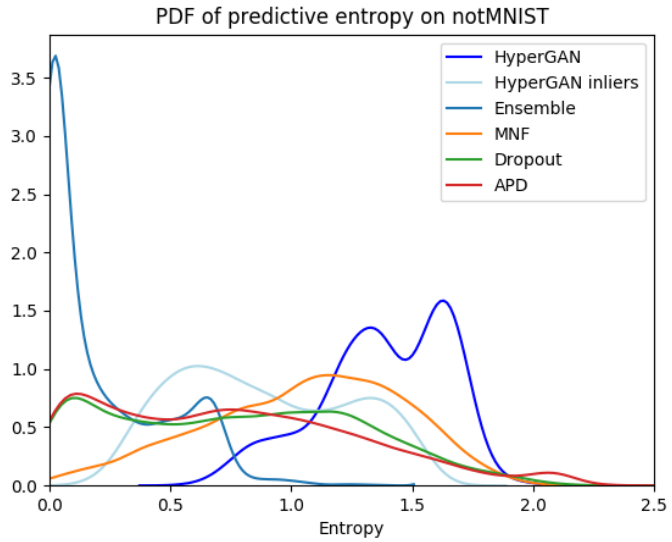


Figure 4.3: Out of distribution performance on notMNIST. PDF of the predictive entropy of all approaches.

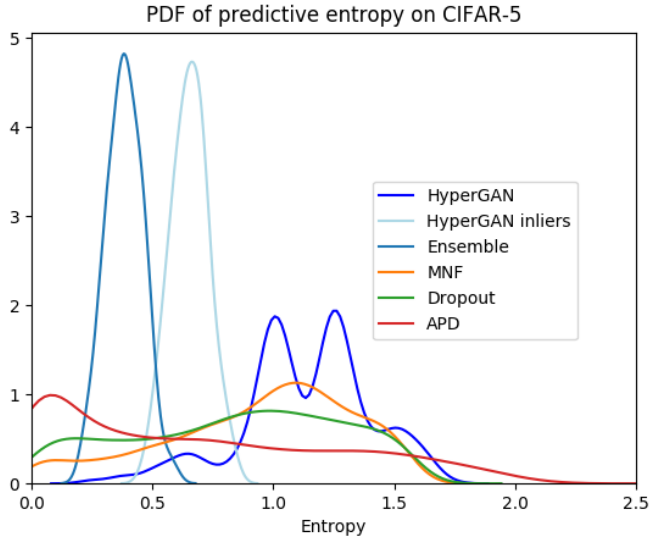


Figure 4.4: Out of distribution performance on CIFAR-10. PDF of the predictive entropy of all approaches on the 5 classes of CIFAR-10.

Adversarial Example Detection

We employ the same experimental setup to the detection of adversarial examples, an extreme type of out-of-distribution data. Adversarial examples are often optimized to

lie within a small neighborhood of a real data point, so that it is hard for humans to detect them visually. They are created by adding perturbations in the direction of the greatest loss with respect to the parameters of the model. Because HyperGAN learns a distribution over parameters, it should be more robust to adversarial attacks with respect to a single set of parameters. We generate adversarial examples using the Fast Gradient Sign method (FGSM) [Goodfellow et al., 2015] and Projected Gradient Descent (PGD) [Madry et al., 2017]. FGSM adds a small perturbation ϵ to the target image in the direction of greatest loss. FGSM is known to underfit to the target model, hence it may transfer better across many similar models. In contrast, PGD takes many steps in the direction of greatest loss, producing a stronger adversarial example, at the risk of overfitting to a single set of parameters. This poses the following challenge: to detect attacks by FGSM and PGD, HyperGAN will need to generate diverse parameters to avoid both attacks.

To detect adversarial examples, we first hypothesize that a single adversarial example will not fool the entire space of parameters learned by HyperGAN. If we then evaluate adversarial examples against many newly generated networks, then we should see a high entropy among predicted probabilities for any individual class.

Adversarial examples have been shown to successfully fool ensembles [Dong et al., 2017], but with HyperGAN one can always generate more models that can be added to the ensemble for the cost of one forward pass, making it hard to fully attack. We compare the performance of HyperGAN with ensembles of $M \in \{5, 10\}$ models trained on MNIST with normal supervised training. We fuse their logits (unnormalized log probabilities) together as $l(\boldsymbol{x}) = \sum_{m=1}^{M} w_n l_m(\boldsymbol{x})$ where w_m is the mth model weighting, and l_m is the logits of the mth model. In all experiments we consider uniformly weighted ensembles. For HyperGAN we sample from the generative model to create as many parameter vectors as we need, then we fuse their resulting logits together. Specifically we test HyperGAN-created ensembles with $M \in \{5, 10, 100, 1000\}$ members each. Adversarial examples are generated by attacking the ensemble directly until the generated image completely fools the whole ensemble. For HyperGAN, we attack the *full* ensemble, but test with a new ensemble of equal size. For other methods we first attack a single model, then test with 100 samples unless otherwise specified.

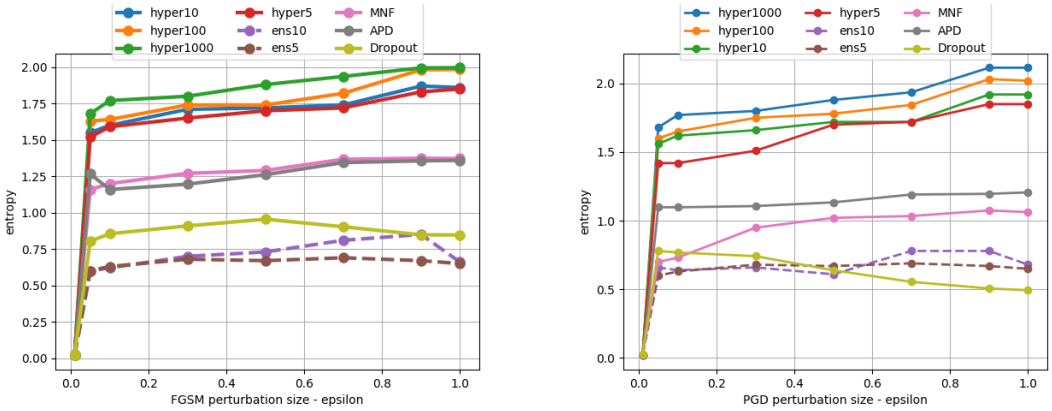


Figure 4.5: Predictive entropy on FGSM and PGD adversarial examples. HyperGAN generates models that give diverse predictions on adversarial examples. Note that for large ensembles, it is hard to find adversarial examples with small norms e.g. $\epsilon = 0.01$

For the purpose of detection, we compute the entropy within the predictive distribution of the ensemble to score the example on how likely it was to be drawn from the training distribution. Figure 4.5 shows that HyperGAN predictions on adversarial examples have high entropy, performing better than other methods as well as standard ensembles. HyperGAN is well-suited to this task as adversarial examples are optimized against a set of parameters - parameters which HyperGAN can change. Because HyperGAN can generate diverse models, it is difficult for an adversarial example to fool the ever-changing ensemble generated by HyperGAN. In some sense, adversarial attacks are always transfer-style attacks with respect to HyperGAN.

As an ablation study, in Fig. 4.6 we show the diversity of the HyperGAN predictions against adversarial examples generated to fool one network. It is shown that while those examples can fool $50\% - 70\%$ of the networks generated by HyperGAN, they usually do not fool all of them.

4.3.5 Ablation Study

The proposed architecture for HyperGAN is motivated by two requirements. First, we want to generate parameters for a target architecture which can solve a task specified by the data and the loss function. Second, we want the generated networks to be diverse.

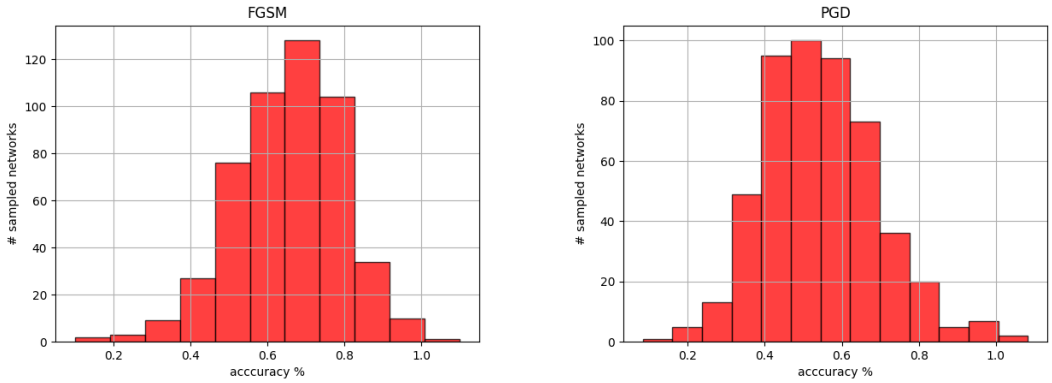


Figure 4.6: Diversity of predictions on adversarial examples. FGSM and PGD examples are created against a network generated by HyperGAN, and tested on 500 more generated networks. FGSM transfers better than PGD, though both attacks fail to cover the distribution learned by HyperGAN

The two specific constructs in the paper are the mixer, which introduces the necessary correlations between generated layers, and an adversarially trained discriminator to enforce that samples from the mixed latent space are well-distributed according to the prior. In this section we test and discuss the validity and effect of these two components by removing each one respectively and check the classification accuracy and the diversity of HyperGAN after the removal.

Training without a discriminator is the simpler of the two experiments. The only modification made to the training procedure is that we remove the distributional constraint on Z. We can see in figure (4.7(a)) that the classification accuracy of the generated networks is unaffected, while the diversity shown in figure (4.7(b)) decreases, showing that by making the mixed latent space well-distributed, the discriminator is having a positive effect in improving the diversity of the generated models. This is similar to the prevention of mode collapse in adversarial [Makhzani et al., 2015] and Wasserstein autoencoders [Tolstikhin et al., 2017]. We can also see that through the training, diversity does indeed decrease, which is also common in GAN training. Hence, early stopping the training when accuracy has just converged may be key to maintaining diversity as well. We would like to study the effect of early stopping more from the theoretical side in future work.

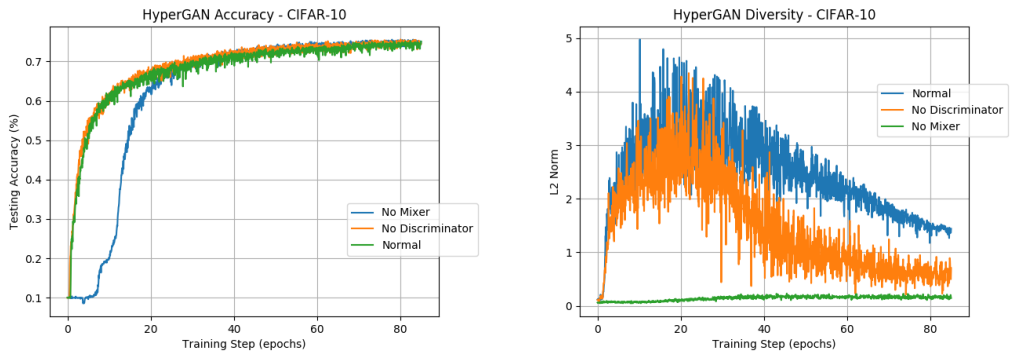


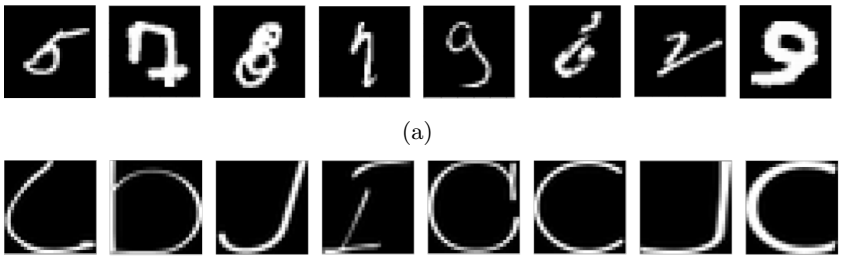
Figure 4.7: Ablation studies of HyperGAN accuracy and diversity on CIFAR-10. (a) HyperGAN without the mixer and without the discriminator, respectively. (b) HyperGAN diversity on CIFAR-10 given a normal training run, with the mixer removed, and with the discriminator removed. Diversity is shown as the standard deviation divided by the norm of the weights, within a population of 100 generated networks.

Next we train HyperGAN without the Mixer Q, or the mixed latent space. In this case, the generator for each layer takes as input an independent d-dimensional sample from a Gaussian distribution. From figure (4.7(a)), we see that even in this case we can obtain similar classification accuracy. However, from figure (4.7(b)) we see that without the mixer, diversity suffers significantly. We hypothesize that without the mixer, a valid optimization trajectory is difficult to find (figure 4.7(a)) also shows that the HyperGAN with no mixer starts with low accuracy for a longer period); when one trajectory is finally found, the optimizer will prioritize classification loss over diversity. When the mixer is included, the built-in correlation between the parameters of different layers may have made optimization easier, hence diverse good optima are found even from different random starts.

4.3.6 Exemplar Outlier Examples

In figure (4.8) we show images of examples which do not behave like other examples from their of their respective distribution. On top are MNIST images which HyperGAN generated ensembles yield a high-entropy predictive distribution. We can see that their semantic content is generally ambiguous and do not fit with the rest of the training data. The bottom row shows notMNIST examples with low-entropy predictive distributions from HyperGAN. It can be seen that these examples look like they could come from

the MNIST training distribution. This serves as a qualitative example of reasonable uncertainty estimates from HyperGAN.



(a)

(b)

Figure 4.8: Example outliers. Top: MNIST examples that HyperGAN assigns high entropy to (outlier examples). Bottom: notMNIST examples that are assigned low entropy (inlier examples)

4.4 Conclusion

In this chapter we proposed HyperGAN, an implicit generative model for learning to sample from the distribution of neural network parameters that fit the data. By training a GAN to learn a probability distribution over neural networks, we can non-deterministically sample diverse, performant networks which we can use to form ensembles that obtain better classification accuracy and uncertainty estimates. Our method is ultimately scalable in terms of the number of networks in the predicting ensemble, requiring just one forward pass to generate a new performant network and a low GPU memory footprint. We have also shown the uncertainty estimates from the generated ensembles are capable of detecting out-of-distribution data and adversarial examples.

Chapter 5: Efficient Exploration in Reinforcement Learning

5.1 Introduction

In this work we focus on the application of uncertainty quantification to exploration in reinforcement learning. We present an algorithm for efficient exploration in unknown environments by leveraging the agent's uncertainty in the dynamics of the environment. As in our previous work, we are primarily interested in the epistemic uncertainty, or the variance in parameters that comes from estimation with finite data. This is in contrast to aleatoric uncertainty, which models the stochastic realizations of an unknown random variable, due to inherent randomness. In this context, the epistemic uncertainty is best thought of as the agent's uncertainty about the environment dynamics. The epistemic uncertainty can thus be reduced by acquiring additional data. Aleatoric uncertainty is due to some stochastic component of the returns or the environment, and is it not reducible given additional data. Exploration methods that do not separate epistemic from aleatoric uncertainty will be inefficient, as they will spend time collecting data to reduce the irreducible aleatoric component of the uncertainty. In the literature, efforts to estimate both components of uncertainty have been used to increase performance in RL agents. Agents seeking to minimize epistemic uncertainty typically explore states where the variance in model predictions/parameters is high, as this reflects a lack of knowledge [O'Donoghue et al., 2017, O'Donoghue, 2018, Osband et al., 2017, 2018, Shyam et al., 2019, Pathak et al., 2019]. Agents that model uncertainty in the returns due to a stochastic environment (aleatoric) seek actions with high probability of receiving reward. Some examples are Distributional RL [Bellemare et al., 2017, Dabney et al., 2018a,b]. Given that epistemic (model) uncertainty can be reduced with more data, modeling this uncertainty is useful for driving efficient exploration. While modeling the aleatoric uncertainty can be used to aid exploitation, i.e. choosing the best action at any time-step. In the following sections, we present a method for efficient exploration that leverages model uncertainty to explore unknown or poorly-understood states. To do so, we learn a distribution over the *transition function*. In our work, we draw samples from

this distribution to evaluate the uncertainty with respect to a given state, and explore accordingly. A key component is how we learn this distribution. The exact form of the transition function is unknown, and it differs between environments. It is important that any distributional representation be flexible and expressive. To this end, we use an implicit distribution to approximate the posterior distribution over transition functions. Implicit distributions are computationally amenable to sampling, and can learn to represent multi-modal distributions. By exploring according to the epistemic uncertainty in the implicit distribution, we show that agents can efficiently explore difficult environments.

We focus on the problem of learning to explore in environments with sparse external rewards. In contrast to our previous work, here we treat outlier data as desirable, and actively seek out data on which our model is uncertain. In environments without external reward, it is important for an effective agent to methodically explore a significant portion of the state space, since there are not enough external signals to indicate how to obtain reward. We follow previous work by constructing an "intrinsic" reward, driven by the uncertainty in the agent's belief of the environment state [Pathak et al., 2019, Shyam et al., 2019]. In this setup, any training signal will be purely a function of the intrinsic reward, and the agent's ability to perform well in these environments is inherently tied to the quality of our uncertainty estimates. Intuitively, agents should explore more thoroughly in states where they are not certain whether exploration could lead to a previously unknown consequence – which could be an unexpected extrinsic reward. However, uncertainty modeling from a deep network has proven to be difficult, with no approach [Snoek et al., 2019] that is proven to be universally applicable.

In this work, we introduce a new framework of Bayesian uncertainty modeling for exploration in deep RL driven by an intrinsic reward. Our framework characterizes the uncertainty in the agent's belief of the environment dynamics in a non-parametric manner to enable flexibility and expressiveness. The main component of our framework is a network generator, each draw of which is a neural network that serves as the dynamics model for RL. Multiple draws approximate a posterior of the dynamics model, and the variance in future state prediction based on this posterior is used as an intrinsic reward for exploration. The generative approach avoids making restrictive distributional assumptions on the likelihood or approximate posterior. Consequently, it provides a significantly

larger model space than previous approaches. Recently, it has been shown [Ratzlaff and Fuxin, 2019] that training such generators can be done in classification problems, and the resulting draws of networks can represent a rich distribution of diverse networks that perform approximately equally well on the classification task.

5.2 Dynamic Model Uncertainty as Intrinsic Reward

To train our generator for the dynamics model, we propose a new algorithm to optimize the KL divergence between the implicit distribution (represented by draws from the generator) and the true posterior of the dynamics model (given the agent's experience) via amortized Stein Variational Gradient Descent (SVGD) [Liu and Wang, 2016b, Feng et al., 2017]. Amortized SVGD allows direct minimization of the KL divergence between the implicit approximate posterior and true posterior without any parametric assumptions, and further projects the infinite dimensional functional gradient computed by SVGD to a finite-dimensional parameter update.

Let $F : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denote a model of the environment dynamics (usually represented by a neural network) that we want to learn based on the agent's experience H. We design a generator module $\boldsymbol{f_\eta} : Z \to \Theta$ that takes a random draw from the normal distribution Z and outputs a sample vector of parameters $\boldsymbol{\theta} \in \Theta$ that determines F (denoted as $F_{\boldsymbol{\theta}}$). If samples from $\boldsymbol{f_\eta}$ represent the posterior distribution $p(F_{\boldsymbol{\theta}}|H)$, then given (s_t, a_t), the uncertainty in the output of the dynamics model can be computed by the following variance among a set of samples $\{\boldsymbol{\theta}_i\}_{i=1}^m$ from $\boldsymbol{f_\eta}$, and used as an intrinsic reward r^{in} for learning an exploration policy,

$$r_t^{in} = \frac{1}{m} \sum\nolimits_{i=1}^{m} \left\| F_{\boldsymbol{\theta}_i}(s_t, a_t) - \frac{1}{m} \sum\nolimits_{\ell=1}^{m} F_{\boldsymbol{\theta}_\ell}(s_t, a_t) \right\|^2 . \tag{5.1}$$

In learning the exploration policy, this intrinsic reward can be computed with either actual rollouts in the environment or simulated rollouts generated by the estimated dynamic model.

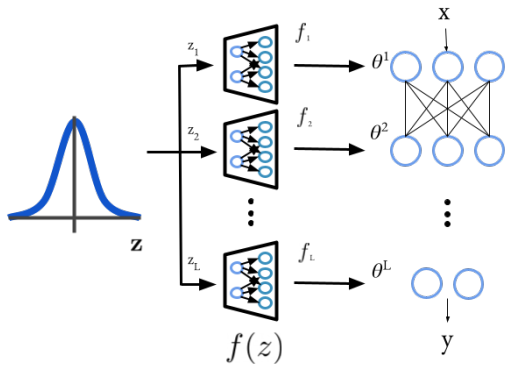


Figure 5.1: Dynamics model generator. Independent noise samples $\{z^1, \cdots, z^L\}$ are drawn from a standard Gaussian with diagonal covariance, and input to layer-wise generators $\{f_1, \cdots, f_L\}$. Each generator f_j outputs parameters θ^j for the corresponding j-th layer of the neural network representing the dynamic model.

5.3 Posterior Approximation via Amortized SVGD

In this section, we introduce the core component of our exploration agent, the dynamic model generator $\boldsymbol{f}$. In the following subsections, we first introduce the design of this generator and then describe its training algorithm in detail. A summary of our algorithm is given in the last subsection.

5.3.1 Implicit Posterior Generator

As shown in Fig. 5.1, the dynamic model is defined as an L-layer neural network function $F_{\boldsymbol{\theta}}(s, a)$, with input (state, action) pair (s, a) and model parameters $\boldsymbol{\theta} = (\theta^1, \cdots, \theta^L)$, where θ^j represents network parameters of the j-th layer. The generator module $\boldsymbol{f}$ consists of exactly L layer-wise generators, $\{f_1, \cdots, f_L\}$, where each f_j takes a random noise vector $\boldsymbol{z}^j \in \mathbb{R}^d$ and outputs the corresponding parameter vector $\theta^j = f_j(z^j; \eta^j)$, where η^j are the parameters of f_j. Note that $\boldsymbol{z}^j$s are generated independently from a d-dimensional standard normal distribution, rather than jointly.

As mentioned in Section. 5.2, this framework has advantages in flexibility and efficiency, comparing with ensemble-based methods [Shyam et al., 2019, Pathak et al., 2019], since it maintains only parameters of the L generators, i.e., $\boldsymbol{\eta} = (\eta^1, \cdots, \eta^L)$, and enables

drawing an arbitrary number of sample networks to approximate the posterior of the dynamic model.

5.3.2 Training with Amortized SVGD

We now introduce the training algorithm of the generator module $\boldsymbol{f_\eta}$. Assuming that the true posterior of the dynamic model given agent's experience H is $p(F|H)$, and the implicit distribution of $F_{\boldsymbol{\theta}}$ captured by $\boldsymbol{f_\eta}$ is $q(F_{\boldsymbol{\theta}})$. We want $q(F_{\boldsymbol{\theta}})$ to be as close as possible to $p(F|H)$, such closeness is commonly measured by the KL divergence $\mathbf{D}_{\mathrm{KL}}\left[q(F_{\boldsymbol{\theta}})\|p(F|H)\right]$. The traditional approach for finding the distribution q that minimizes $\mathbf{D}_{\mathrm{KL}}\left[q(F_{\boldsymbol{\theta}})\|p(F|H)\right]$ is variational inference, where an ELBO is maximized [Blei et al., 2017]. Recently, a nonparametric variational inference framework, Stein Variational Gradient Descent (SVGD) [Liu and Wang, 2016b], was proposed, which represents q with a set of particles rather than making any parametric assumptions, and approximates the functional gradient descent w.r.t. $\mathbf{D}_{\mathrm{KL}}\left[q(F_{\boldsymbol{\theta}})\|p(F|H)\right]$ by iterative particle evolution. We apply SVGD to our sampled network functions, and follow the idea of amortized SVGD [Feng et al., 2017] to project the functional gradients to the parameter space of $\boldsymbol{\eta}$ by back-propagation through the generators.

Given a set of dynamic functions $\{F_{\boldsymbol{\theta}_i}\}_{i=1}^m$ sampled from $\boldsymbol{f_\eta}$, SVGD updates each function by

$$F_{\boldsymbol{\theta}_i} \leftarrow F_{\boldsymbol{\theta}_i} + \epsilon\phi^*(F_{\boldsymbol{\theta}_i}), \qquad i = 1, \cdots, m,$$

where ϵ is a step size, and ϕ^* is the function in the unit ball of a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, that maximally decreases the KL divergence between the distribution q represented by $\{F_{\boldsymbol{\theta}_i}\}_{i=1}^m$ and the target posterior p,

$$\phi^* = \max_{\phi\in\mathcal{H}} \left\{ -\frac{d}{d\epsilon}\mathbf{D}_{\mathrm{KL}}(q||p), \quad s.t. ||\phi||_{\mathcal{H}} \leq 1 \right\}.$$

This optimization problem has a closed form solution,

$$\phi^*(F_{\boldsymbol{\theta}_i}) = \mathbb{E}_{F_{\boldsymbol{\theta}}\sim q}\left[\nabla_{\boldsymbol{\theta}} \log p(F) k(F, F_{\boldsymbol{\theta}_i}) + \nabla_{\boldsymbol{\theta}} k(F, F_{\boldsymbol{\theta}_i})\right], \tag{5.2}$$

where $k(\cdot,\cdot)$ is the positive definite kernel associated with the RKHS. The log-likelihood

term for $F_{\boldsymbol{\theta}}$ corresponds to the negation of the regression loss of future state prediction for all transitions in H, i.e., $\log p(F_{\boldsymbol{\theta}}) = -\sum_{(s,a,s')\in H} L(F_{\boldsymbol{\theta}}(s,a), s')$. Given that each $\boldsymbol{\theta}_i$ is generated by $\boldsymbol{f}(\boldsymbol{z};\boldsymbol{\eta})$, the update rule for $\boldsymbol{\eta}$ can be obtained by by the chain rule,

$$\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \epsilon \sum_{i=1}^{m} \nabla_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{z}_i;\boldsymbol{\eta}) \phi^*(\boldsymbol{\theta}_i), \tag{5.3}$$

where $\phi^*(\boldsymbol{\theta}_i)$ can be computed by (5.2) using empirical expectation from sampled batch $\{\boldsymbol{\theta}_i\}_{i=1}^m$,

$$\phi^*(\boldsymbol{\theta}_i) = \frac{1}{m} \sum_{\ell=1}^{m} \left\{ - \left[\sum_{(s,a,s')\in H} \nabla_{\boldsymbol{\theta}_\ell} L(F_{\boldsymbol{\theta}_\ell}(s,a), s') \right] \right. \tag{5.4}$$

$$\left. \cdot\, k(F_{\boldsymbol{\theta}_\ell(s,a)}, F_{\boldsymbol{\theta}_i(s,a)}) + \nabla_{\boldsymbol{\theta}_\ell} k(F_{\boldsymbol{\theta}_\ell(s,a)}, F_{\boldsymbol{\theta}_i(s,a)}) \right\}, \tag{5.5}$$

where $k(\cdot,\cdot)$ is the RBF kernel evaluated at function outputs, which is in the state space.

5.3.3 Summary of the Exploration Algorithm

To condense what we have proposed so far, we summarize in Algorithm 1 the procedure used to train the generator of dynamic models and the exploration policies.

Our algorithm starts with a buffer H of random transitions and explores for some fixed number of episodes. For each episode, our algorithm samples a set of dynamic models $F_{\boldsymbol{\Theta}} = \{F_{\boldsymbol{\theta}_i}\}$ from the generator $\boldsymbol{f}_{\boldsymbol{\eta}}$, and updates the generator parameters $\boldsymbol{\eta}$ using amortized SVGD (5.3) and (5.5). For the policy update, the intrinsic reward (5.1) is evaluated on the actual experience H and the simulated experience $\widetilde{H}$ generated by $F_{\boldsymbol{\theta}_i}$. The exploration policy is then updated using a model-free RL algorithm on the collected experience H_π and intrinsic rewards R_π. The updated exploration policy is then used to rollout in the environment for T steps so that new transitions are collected and added to the buffer H for subsequent iterations. We repeat the process, until the episode is done.

Algorithm 1 Exploration with an Implicit Distribution

Initialize Generator $\boldsymbol{f}_{\boldsymbol{\eta}}$, parameters T, m
Initialize Policy π, Experience buffer H
while True **do**
  while episode not **done**: **do**
    $F_{\boldsymbol{\Theta}} \leftarrow \boldsymbol{f}(\boldsymbol{z}; \boldsymbol{\eta}), \boldsymbol{z} \sim \mathcal{N}(0, I^d)$
    $\boldsymbol{\eta} \leftarrow$ evaluate (5.3), (5.5) on H
    $H_\pi \leftarrow H \cup \widetilde{H}$,
    $\widetilde{H} \sim \text{MDP}(F_{\boldsymbol{\Theta}})$
    $R_\pi \leftarrow r^{in}(F_{\boldsymbol{\theta}}, s, a | (s, a) \sim H_\pi)$ by (5.1)
    $\pi \leftarrow$ update policy on (H_π, R_π)
    $H_T \leftarrow$ rollout π for T steps
    $H \leftarrow H \cup H_T$
  end
end

5.4 Experimental Results

In this section we conduct experiments to compare our approach to existing state-of-the-art in efficient exploration with intrinsic reward to show the following:

- An agent with an implicit posterior over dynamic models explores more effectively and efficiently than agents using a single model or a static ensemble.
- Agents seeking external reward find better policies when initialized from powerful exploration policies. Our ablation study shows that the better the exploration policy as an initialization, the better the downstream task policy can learn.

To isolate our main claim of the superior exploration efficiency of the proposed method, we first consider exploration tasks agnostic of any external reward. In this setting, the agent explores the environment irrespective of any downstream task. Then, to further investigate the potential of our exploration policies, we consider transferring the learned exploration policy to downstream task policies where a dense external reward is provided. Note that both cases are important for understanding and applying exploration policies. In sparse reward settings, such as a maze, the reward could occur at any location, with-

out informative hints accessible at other locations. Therefore an effective agent must be able to efficiently explore the entire state space in order to consistently find rewards under different task settings. In dense reward settings, the trade-off between exploration and exploitation plays a central role in efficient policy learning. Our experiments show that even for a state-of-the-art model-free algorithm like the Soft Actor-Critic (SAC) [Haarnoja et al., 2018], which already incorporates a strong exploration mechanism (the maximum entropy framework), spending some initial rollouts to learn a powerful exploration policy as an initialization of the task policy still considerably improves the learning efficiency.

5.4.1 Toy Task: NChain

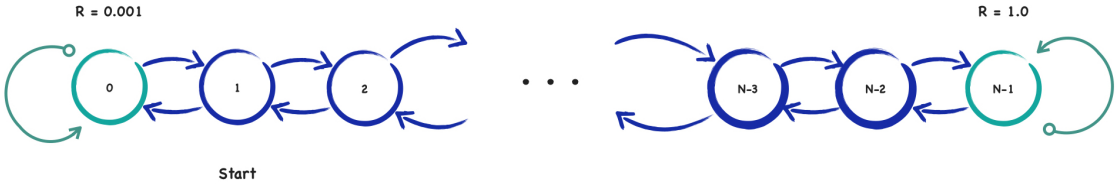


Figure 5.2: NChain environment.

As a sanity check, we first follow *MAX* [Shyam et al., 2019], and evaluate our method on a stochastic version of the toy environment NChain. As shown in Fig. 5.2, the chain is a finite sequence of N states. Each episode starts from state 1 and lasts for $N + 9$ steps. For each step, the agent can move forward to the next state in the chain or backward to the previous state. Attempting to move off the edge of the chain results in the agent staying still. Reward is only afforded to the agent at the edge states: 0.01 for reaching state 0, and 1.0 for reaching state $N - 1$. In addition, there is uncertainty built into the environment: each state is designated as a *flip-state* with probability 0.5. When acting from a flip-state, the agent's actions are reversed, i.e., moving forward will result in movement backward, and vice-versa. Given the (initially) random dynamics and a sufficiently long chain, we expect an agent using an ϵ-greedy exploration strategy to exploit only the small reward of state 0. In contrast, agents with exploration policies which actively reduce uncertainty can efficiently discover all states in the chain. Fig. 5.3 shows that our agent navigates the chain in less than 15 episodes, while the ϵ-greedy

agent (double DQN) does not make meaningful progress.

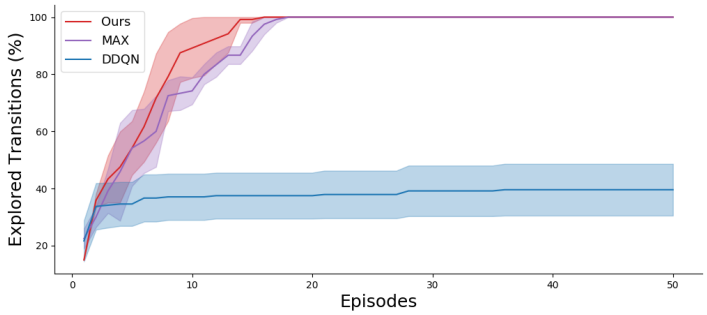


Figure 5.3: Results on the 40-link chain environment. Each line is the mean of three runs, with the shaded regions corresponding to ± 1 standard deviation. Both our method and *MAX* actively reduce uncertainty in the chain, and therefore are able to quickly explore to the end of the chain. ϵ-greedy DQN fails to explore more than 40% of the chain.

5.4.2 Pure Exploration Results

For pure exploration experiments, we consider three challenging continuous control tasks in which efficient exploration is known to be difficult. In each environment, the dynamics are nonlinear and cannot be solved with simpler (efficient) tabular approaches. As explained in the beginning of Section. 5.4, the external reward is completely removed; the agent is motivated purely by the uncertainty in its belief of the environment.

Experimental setup

To validate the effectiveness of our method, we compare with several state-of-the-art formulations of intrinsic reward. Specifically, we conduct experiments comparing the following methods:

- (*Ours*) The proposed intrinsic reward, using the estimated variance of an implicit distribution of the dynamic model.
- (*Random*) Random exploration as a naive baseline.
- (*ICM*) Error between predicted next state and observed next state [Pathak et al., 2017].

- (*Disagreement*) Variance of predictions from an ensemble of dynamic models [Pathak et al., 2019].
- (*MAX*) Jensen-Renyi divergence between predictions from an ensemble of dynamic models [Shyam et al., 2019].

Implementation details

Given our goal is to compare the performance across different intrinsic rewards, we fix the model architecture, training pipeline, and hyper-parameters across all methods. Shared hyper-parameters follow the *MAX* default settings. For the purpose of computing the information gain, dynamic models for *MAX* predict both mean and variance of the next state, while for other methods, dynamic models predict only the mean. Since our method trains a generator of dynamic models instead of a fixed-size ensemble, we fix the number of models we sample from the generator at $m = 32$, which equals the ensemble size for *MAX*, *ICM*, and *Disagreement*. For all experiments, we use SAC as the model-free RL algorithm used to train the exploration policies.

Acrobot Control

Our first environment is a modified continuous control version of the Acrobot. As shown in Figure 5.4, the Acrobot environment begins with a hanging down pendulum which consists of two links connected by an actuated joint. Normally, a discrete action $a \in \{-1, 0, 1\}$ either applies a unit force on the joint in the left or right direction ($a = \pm 1$), or not ($a = 0$). We modify the environment such that a continuous action $a \in [-1, 1]$ applies a force $|a|$ in the corresponding direction.

To focus on efficient exploration, we test the ability of each exploration method to sweep the entire lower hemisphere: positioning the acrobot completely horizontal towards both (left and right) directions. Given this is a relatively simple task and can be solved by random exploration, as shown in Figure 5.4, all four intrinsic reward methods solve it within just hundreds of steps and our method is the most efficient one. The takeaway here is that in relatively simple environments where there might be little room for improvement over state-of-the-art, our method still achieves a better performance due to its flexibility and efficiency in approximating the model posterior. As we will see in

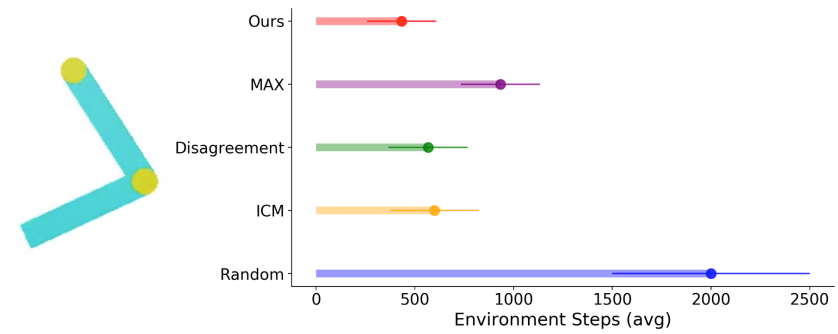


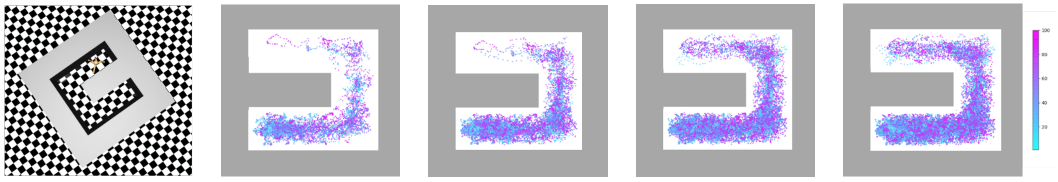
Figure 5.4: Performance on the Acrobot environment. We average five seeds, with error bars representing ±1 standard deviation. The length of each horizontal bar indicates the number of environment steps each agent/method takes to swing the acrobot to fully horizontal on both (left and right) directions.

subsequent experiments, this observation scales well with the increasing difficulty of the environments.

Ant Maze Navigation

Next, we evaluate on the Ant Maze environment. In the Ant control task, the agent provides torques to each of the 8 joints of the ant. The provided observation contains the pose of the torso as well as the angles and velocities of each joint. For the purpose of exploration, we place the Ant in a U-shaped maze, where the goal is to reach the end of the maze, discovering all the states. The agent's performance is measured by the percentage of the maze explored during evaluation. Figure 5.6(a) shows the result of each method over 5 seeds. Our agent consistently navigates to the end of the maze faster than the other competing methods. To see that our agent fully explores the maze environment, we include state visitation diagrams in figure 5.5.

While *MAX* [Shyam et al., 2019] also navigates the maze, the implicit uncertainty modeling scheme in our method allows our agent to better estimate the state novelty, which leads to a considerably faster exploration. To provide a more intuitive understanding of the effect of an intrinsic reward and how it might correlate to the performance, we also plot in Figure 5.6(b) the intrinsic reward observed by our agent at each exploration step, compared with that observed by the *MAX* agent. For fair comparison we plot the intrin-



(a) Ant Maze (b) 2500 Steps (c) 5000 Steps (d) 7500 Steps (e) 10000 Steps

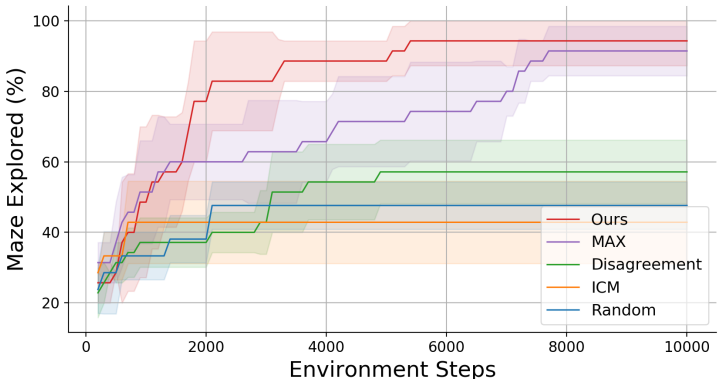
Figure 5.5: We show (a) the U-shaped ant maze. Figures (b-e) show the behavior of our (*Ours*) agent at different stages of training, over 5 seeds. Points are color-coded with blue points occurring at the beginning of the episode, and red points at the end.

sic reward from Eq.(5.1) for both methods. We can see that after step 2,000, predictions from the *MAX* ensemble start to become increasingly similar, leading to a decline in intrinsic reward (Fig. 5.6(b)) as well as a slow-down in exploration speed (Fig. 5.6(a)). We hypothesize this is because in a regular ensemble, all members are updating their gradients on the same experiences without an explicit term to match the real posterior, leading to eventually all agents converging to the same one. By contrast, our intrinsic reward keeps increasing around step 2,000 and remains high as we continue to quickly explore new states in the maze, only starting to decline once we have solved the maze at approximately step 5,000.

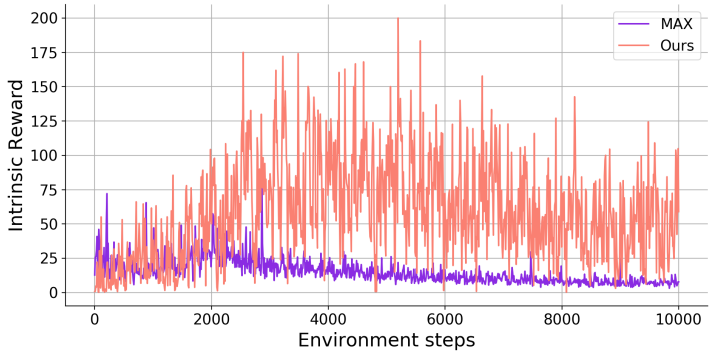
Robotic Manipulation

The final task is an exploration task in a robotic manipulation environment, HandManipulateBlock. As shown in Figure 5.7(a), a robotic hand is given a palm-sized block for manipulation. The agent has actuation control of the 20 joints that make up the hand, and its exploration performance is measured by the percentage of possible rotations of the cube that the agent performs. This is different from the original goal of this environment since we want to evaluate task-agnostic exploration rather than goal-based policies. In particular, the state of the cube is represented by Cartesian coordinates along with a quaternion to represent the rotation. We transform the quaternion to Euler angles and discretize the resulting state space by 45 degree intervals. The agent is evaluated based on how many of the 512 total states are visited.

This task is far more challenging than previous tasks, having a larger state space and



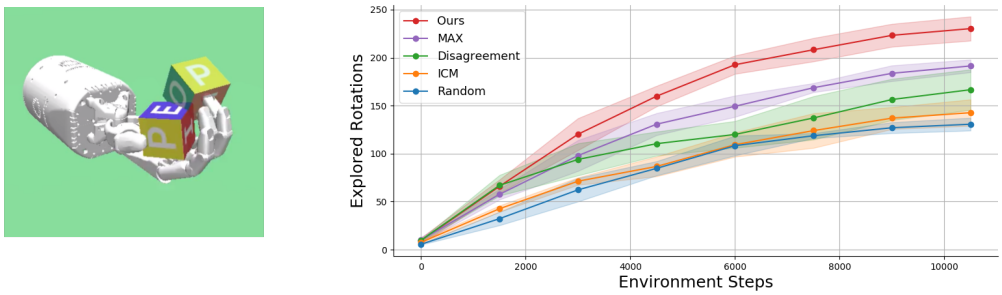
(a) Ant Navigation Task Results



(b) Ant Intrinsic Rewards

Figure 5.6: Ant maze performance. (a) performance of each method with mean and ± 1 standard deviation (shaded region) over five seeds. x-axis is the number of steps the ant has moved, y-axis is the percentage of the U-shaped maze that has been explored. Figure (b) shows the proposed intrinsic reward magnitude for each step in the environment, calculated for both our method and *MAX*.

action space. Additionally, states are more difficult to reach than the Ant Maze environment: requiring manipulation of 20 joints instead of 8. In order to explore in this environment, an agent must also learn how to rotate the block without dropping it. Figure 5.7(b) shows the performance of each method over 5 seeds. This environment proved very challenging for all methods, none succeeded in exploring more than half of the state space. Still, our method performs the best by a clear margin.



(a) Robotic Hand

(b) Manipulation Task Results

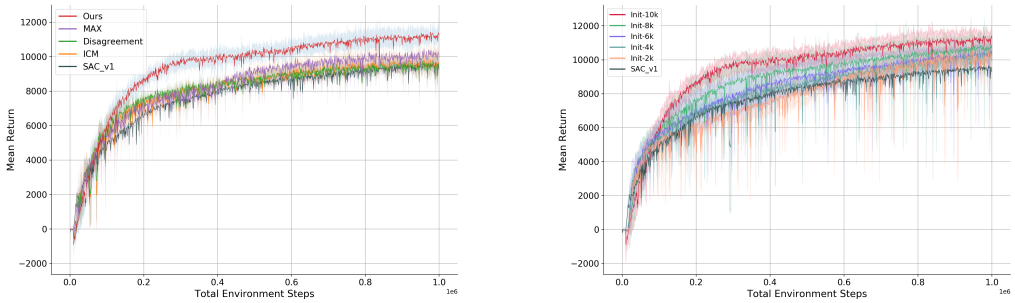
Figure 5.7: We show (a) the Robotic Hand task in motion. (b) Performance of each method with mean and ± 1 standard deviation (shaded region) over five seeds. x-axis is the number of manipulation steps, y-axis is the number of rotation states of the block that has been explored. Our method (red) explores clearly faster than all other methods.

5.4.3 Policy Transfer Experiments

So far, we have demonstrated that the proposed implicit generative modeling of the posterior over dynamic models leads to more effective and efficient pure exploration policies. While the efficiency of pure exploration is important under sparse reward settings, a natural follow-up question is whether a strong pure exploration policy would also be beneficial for downstream tasks where dense rewards are available. We give an answer to this question by performing the following experiments in the widely-used HalfCheetah environment.

We first train a task-agnostic exploration policy following Algorithm 1 for 10,000 environment steps. The trained policy is then used to initialize the (downstream) task policy, followed by standard training of the task policy using external rewards. In particular, we use SAC for both exploration policy and task policy. In our comparison experiments, we initialize the task policy using exploration policies trained by *MAX*, *ICM*, *Disagreement*, and *Ours* respectively, we also include the standard SAC (without exploration policy initialization) as a baseline. For the training of the task policy, we follow the recommended settings for HalfCheetah given in the original SAC method.

Figure 5.8(a) shows the performance of all compared methods on HalfCheetah-v2. We can see that the comparatively small number of initial steps spent on pure exploration



(a) Policy Transfer Performance

(b) Policy Transfer with Varying Exploration Time

Figure 5.8: Policy transfer results. Figure (a) shows the performance of SAC baseline (SAC) and four SAC variants initialized from a 10,000-step exploration policy trained with different intrinsic reward methods, ICM, Disgreement, MAX, and our proposed method (Ours) respectively. Figure (b) shows the performance of downstream SAC policy training initialized with our proposed exploration policy under different settings of exploration length. Init-Nk refers to the SAC agent initialized from our exploration policy which has been trained for N-thousand exploration steps.

pays off when the agent switches to the downstream task. Even though SAC is widely regarded as a strong baseline with maximum entropy-based exploration mechanism, all intrinsic reward methods are able to improve the baseline more or less, by introducing a pure exploration stage before standard training of SAC. We also observe that the stronger the pure exploration policy is, the more it can improve the training efficiency of the downstream task. Task policies initialized with our exploration policy (*Ours*) still perform the best with a clear margin.

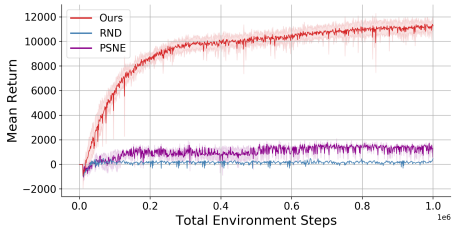
We also conduct an ablation study to better understand the relation between the number of initial exploration steps and the improvement it brings to the downstream task training. We compare multiple variants of *Ours* in Figure 5.8(a), with different numbers of initial exploration steps: 2,000, 4,000, 6,000, 8,000, and 10,000 steps. As shown in Figure 5.8(b), with more than 4,000 steps of initial exploration, our approach is able to improve upon the SAC baseline in the downstream task. The longer our exploration policy is trained, the more beneficial it is to the training of the downstream task. 2,000 steps of initial exploration, on the other hand, harms the downstream task training, since it could be too short to obtain any reasonable exploration policy.

5.4.4 Comparison to Model-Free Exploration Approaches

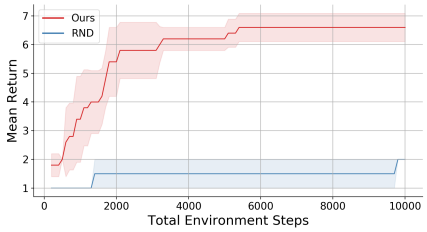
Here we compare our method to two other representative exploration methods. Parameter Space Noise for Exploration (PSNE) Plappert et al. [2017] adds parametric noise to the weights of an agent, similar to Fortunato et al. [2017]. The noise parameters are learned by gradient descent, and the additional stochasticity in the induced policy is responsible for increased exploration ability. Random Network Distillation (RND) is another well-known method Burda et al. [2018b] that introduces a randomly initialized function $\boldsymbol{g} : \mathcal{S} \rightarrow \mathbb{R}^k$ which maps states $\boldsymbol{s}$ to a k-dimensional vector, similar to Osband et al. [2018]. A second function $\hat{\boldsymbol{g}} : \mathcal{S} \rightarrow \mathbb{R}^k$ is trained to match the predictions given by f. The prediction error $\boldsymbol{g}(\hat{\boldsymbol{s}}) - \boldsymbol{g}(\boldsymbol{s})$ is used as an exploration bonus to the reward during training, similar to the psuedo-count based exploration bonus in Bellemare et al. [2016]. RND has been shown to be a strong baseline for both task-specific environments and pure exploration.

In Figure 5.9(a), we first compare our method with PSNE and RND on the HalfCheetah environment, as both can used to learn task-specific policies. For both methods, we use the author provided codes to run our experiments. Because RND is initially designed for discrete actions, we modify the policy to handle continuous action spaces. However, we were unable to recover the reported results from PSNE using the provided code. In Figure 5.9(b), we further compare with RND in the pure exploration setting. We omit PSNE from this experiment, as PSNE does not have intrinsic reward, or another mechanism that can be directly used for pure exploration in the Ant Maze environment. For Ant Maze, each method runs for 10k steps for pure exploration, without external reward.

These methods lack an explicit model of the environment dynamics. It has been shown that model-based methods have a considerable advantage in sample efficiency. RND in particular, takes hundreds of millions of environment steps to achieve its performance. We show in Figure 5.9(a) and 5.9(b) that our method enables superior downstream task performance, and better sample efficiency in exploration, respectively.



(a) Baseline comparison on HalfCheetah



(b) Baseline comparison on Ant Maze

Figure 5.9: Comparison with RND and PSNE on HalfCheetah with external reward (a), and a pure exploration comparison with RND on Ant Maze (b).

5.5 Conclusion

In this work, we introduced a new method for representing the agent's uncertainty of the environment dynamics. Utilizing amortized SVGD, we learned an approximate posterior over dynamics models. We use this approximate posterior to formulate an intrinsic reward based on the uncertainty estimated from samples of this distribution, enabling efficient exploration in difficult environments, Future work in this direction includes investigating more efficient sampling techniques to reduce the computational cost inherent to many model-based algorithms. We would also investigate principled methods of combining intrinsic and external rewards, and how different exploration policies influence the downstream task performance.

5.6 Additional Implementation Details

Here we describe in more detail the various implementation choices we used for our method as well as for the baselines. We describe additional details of the environments that we evaluated in, as well as hyperparameters and algorithm design choices.

Toy Chain Environment

The chain environment is implemented based on the NChain-v0 gym environment. We alter NChain-v0 to contain 40 states instead of 10 to reduce the possibility of solving the environment with random actions. We also modify the stochastic 'slipping' state behavior by fixing the behavior of the states respect to reversing an action. For both our

method and *MAX*, we use ensembles of 5 deterministic neural networks with 4 layers, each is 256 units wide with tanh nonlinearities. As usual, our ensembles are sampled from the generator at each timestep, while *MAX* uses a static ensemble. We generate each layer in the target network with generators composed of two hidden layers, 64 units each with ReLU nonlinearities. Both models are trained by minimizing the regression loss on the observed data. We optimize using Adam with a learning rate of 10^{-4}, and weight decay of 10^{-6}. We use Monte Carlo Tree Search (MCTS) to find exploration policies for use in the environment. We build the tree with 25 iterations of 10 random trajectories, and UCB-1 as the selection criteria. Crucially, when building the tree, we query the dynamic models instead of the simulator, and we compute the corresponding intrinsic reward. For intrinsic rewards, *MAX* uses the Jensen Shannon divergence while our method uses the variance in the predictions within the ensemble. After building the tree we take an action in the real environment according to our selection criteria. There is a small discrepancy between the numbers reported in the MAX paper for the chain environment. This is due to using UCB-1 as the selection criteria instead of Thompson sampling as used in the *MAX* paper. We take actions in the environment based on the children with the highest value. The tree is then discarded after one step, after which, the dynamic models are fit for 10 additional epochs.

Continuous Control Environments

For each method where applicable, we use the method-specific hyperparameters given by the authors. Due to experimenting on potentially different environments, we search for a suitable learning rate which works the best for each method across all tasks. The common details of each exploration method are as follows. Each method uses (or samples) an ensemble of dynamic models to approximate environment dynamics. An ensemble consists of 32 networks with 4 hidden layers, 512 units wide with ReLU nonlinearities, except for *MAX* which uses swish[1]. *ICM*, *Disagreement*, and our method use ensembles of deterministic models, while *MAX* uses probabilistic networks which output a Gaussian distribution over next states. The approximate dynamic models (ensembles/generators) are optimized with Adam, using a minibatch size of 256, a learning rate of 1.0^{-4}, and weight decay of 1.0^{-5}.

[1]Swish refers to the nonlinearity proposed by [Ramachandran et al., 2017] which is expressed as a scaled sigmoid function: $y = x + sigmoid(\beta x)$

For our dynamic model, each layer generator is composed of two hidden layers, 64 units wide and ReLU nonlinearity. The output dimensionality of each generator is equal to the product of the input and output dimensionality of the corresponding layer in the dynamic model. To sample one dynamic model, each generator takes as input an independent draw from $\boldsymbol{z} \sim Z$ where $Z = \mathcal{N}(\mathbf{0}_{32}, \mathbf{I}_{32})$. We sample ensembles of a given size m by instead providing a batch $\{\boldsymbol{z}_i\}_{i=1}^m$ as input. To train the generator such that we can sample accurate transition models, we update according to equation (4) in the main text; we compute the regression error on the data, as well as the repulsive term using an appropriate kernel. For all experiments we use a standard Gaussian kernel $K(F_{\boldsymbol{\theta}_i}, F_{\boldsymbol{\theta}_j}) = \exp\left(-d(F_{\boldsymbol{\theta}_i}, F_{\boldsymbol{\theta}_j})/h\right)$, where $d(F_{\boldsymbol{\theta}_i}, F_{\boldsymbol{\theta}_j}) = \frac{1}{n}\sum_{l=1}^{n} \|F_{\boldsymbol{\theta}_i}(\boldsymbol{x}_l) - F_{\boldsymbol{\theta}_j}(\boldsymbol{x}_l)\|_2^2$ for a training batch $\{\boldsymbol{x}_l\}_{l=1}^n$. Where h is the median of the pairwise distances between sampled particles $\{F_{\boldsymbol{\theta}}\}_{i=1}^m$. Because we sample functions $F_{\boldsymbol{\theta}}$ instead of data points, the pairwise distance is computed by using the likelihood of the data $\boldsymbol{x}$ under the model.

For *MAX*, we use the code provided from [Shyam et al., 2019][2]. Each member in the ensemble of dynamic models is a probabilistic neural network that predicts a Gaussian distribution (with diagonal covariance) over the next state. The exploration policy is trained with SAC, given an experience buffer of rollouts $\bar{H} = \{s, a, s'\} \cup R\pi$ performed by the dynamic models, where R_π is the intrinsic reward: the Jensen-Renyi divergence between next state predictions of the dynamic models. The policy trained with SAC acts in the environment to maximize the intrinsic reward, and in doing so collects additional transitions that serve as training data for the dynamic models for the subsequent training phase.

For *Disagreement* [Pathak et al., 2019], we implement this method under the MAX codebase, following the implementation given by the authors[3]. The intrinsic reward is formulated as the predictive variance of the dynamic models, where the models are represented by a bootstrap ensemble.

Policy Transfer with Warm-up

The exploration policies for our method, *MAX*, *ICM*, and *Disagreement* were trained exactly as in the pure exploration experiments. We trained each exploration policy for

[2]https://github.com/nnaisense/max

[3]https://github.com/pathak22/exploration-by-disagreement

10k steps. We then initialize a new SAC agent with the exploration policy. This agent is initialized within the HalfCheetah environment that includes the external reward. Given that the new agent has an empty replay buffer, we perform a warm-up stage to collect initial data before performing any parameter updates. We collect this initial data by rolling out the pure exploration policy for 10K steps, and storing the observed transitions in the fresh agent's replay buffer. Note that the policy is frozen during the warm-up. After this initial warm-up stage, we allow the fresh agent to train as normal for 1M steps (including the steps taken during warm-up and pure exploration), with respect to the external reward.

Policy Transfer Without Warm-up

The policy transfer experiments without the warm-up stage are similar in that the pure exploration polices are trained for 10K steps, agnostic of the downstream task, then frozen. However, instead of training a new SAC agent on transitions obtained via a warm-up stage, we only transfer the parameters of the pure exploration policy to the fresh SAC agent. Then the transition buffer is cleared, and the agent is trained in the standard setting with external reward for 1M steps (including the steps already taken during pure exploration).

Chapter 6: Generative Particle-based Variational Inference

6.1 Introduction

In this chapter we revisit the topic of particle-based variational inference (ParVI) directly from a Bayesian inference perspective. We examine the approximations made in amortized ParVI methods, and we propose a new generative ParVI approach that improves on these limitations.

Bayesian inference provides a powerful framework for reasoning and prediction under uncertainty. However, computing the posterior is tractable with only a few parametric distributions, making wider applications of Bayesian inference difficult. Traditionally, MCMC and variational inference methods are utilized to provide tractable approximate inference, but these approaches face their own difficulties if the dimensionality of the space is extremely high. For example, a recent case of interest is Bayesian neural networks (BNNs), which applies Bayesian inference to deep neural network training in order to provide a principled way to assess model uncertainty. The goal in this regime is to model the posterior of every parameter in all the weight tensors from every layer of a deep network. However, developing efficient computational techniques for approximating this intractable posterior with extremely high dimensionality remains challenging. In this work we take a closer look at particle-based variational inference. ParVI methods [Liu and Wang, 2016a, Liu et al., 2019, Liu, 2017] have been proposed to represent the variational distribution by a set of particles and update them through a deterministic optimization process to approximate the posterior. While achieving both asymptotic accuracy and computational efficiency, ParVI methods are restricted by the fixed number of particles and lack the ability of drawing new samples beyond the initial set of particles. To address this issue, amortized ParVI methods Wang and Liu [2016] have been proposed to amortize the ParVI gradients in training a neural sampler. While being flexible in drawing samples, in Sec. 4 we show that amortized ParVI methods cannot match the convergence behavior of ParVI methods.

This work proposes a new method for learning to approximately sample from the posterior distribution. We construct a neural sampler that is trained with the functional gradient of the KL-divergence between the empirical sampling distribution and the target distribution, assuming the gradient resides within a reproducing kernel Hilbert space. Our generative ParVI (GPVI) approach maintains the asymptotic performance of ParVI methods while offering the flexibility of a generative sampler. Through carefully constructed experiments, we show that GPVI outperforms previous generative ParVI methods such as amortized SVGD, and is competitive with ParVI as well as gold-standard approaches like Hamiltonian Monte Carlo for fitting both exactly known and intractable target distributions.

In this work, we propose a generative particle variational inference (GPVI) approach that addresses those issues. GPVI trains a neural sampler network by directly estimating the functional gradient with respect to the KL-divergence between the distribution of generated particles and the target distribution, and pulls it back to update the neural sampler. As such it allows the neural sampler to directly generate particles that match the posterior distribution, hence achieving the asymptotic accuracy and computational efficiency of ParVI methods. In figure **??**, we show that the predictive distribution of 1D regression functions sampled from GPVI nearly matches that from the ParVI solution, while amortized SVGD fails.

The main computational challenge lies in a reliable estimate of the functional gradient that involves the inverse of the input-output Jacobian of the neural sampler. Instead of directly computing this term and paying a high computational cost, we introduce a helper network to estimate the inverse Jacobian vector product and train the helper network via gradient descent. By alternating between this gradient step and the gradient update of the sampler network, the computational cost is distributed over the whole training procedure.

In experiments, our proposed approach achieves comparable convergence performance as ParVI methods. It is considerably superior than that of amortized ParVI methods, while still allowing efficient sampling from the posterior. By directly applying our approach as a hypernetwork to generate BNNs, we achieve competitive performance regarding uncertainty estimation.

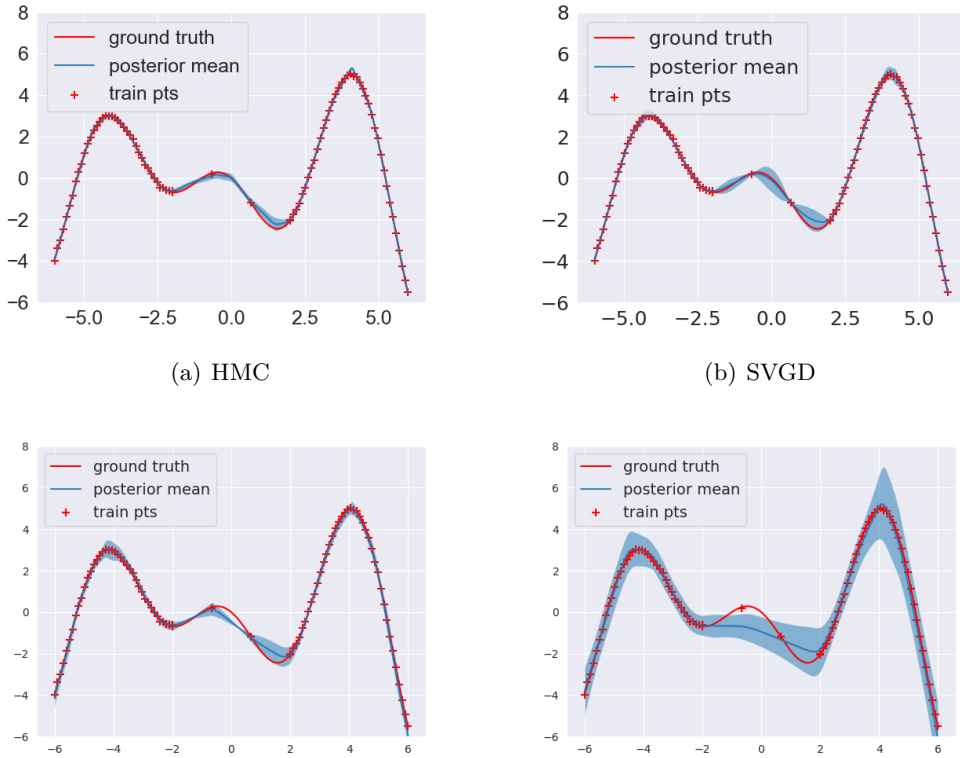
In summary, our contributions are three-fold,

- We propose GPVI, a new variational inference approach that trains a neural sampler to generate particles from any posterior distribution. GPVI estimates the functional gradient and uses it to update the neural sampler. It enjoys the asymptotic accuracy and computational efficiency of ParVI methods. Comparing with existing amortized ParVI methods, our approach enjoys the same efficiency and flexibility while showing considerable advantage in convergence behavior.
- We design careful techniques for efficient gradient estimates that address the challenges in approximating the product between the inverse of the Jacobian and a vector.
- We apply our approach to BNNs and achieve competitive uncertainty estimation quality for deep neural networks.

6.2 Generative Particle Variational Inference

We want to learn a parametric generator: $\boldsymbol{f}_{\boldsymbol{\eta}} : Z \to \mathbb{R}^m$, parameterized by $\boldsymbol{\eta}$, where $Z \subset \mathbb{R}^m$ is the convex space of input noise and $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})$ generates a sample $\boldsymbol{x}$ from an input noise $\boldsymbol{z}$. Let $q(\boldsymbol{x})$ represents the implicit distribution of samples generated by $\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})$, where $\boldsymbol{z} \sim N(\mathbf{0}, I_m)$. Let $p(\boldsymbol{x})$ be the target distribution, we want to solve for $\boldsymbol{f}_{\boldsymbol{\eta}}$ that minimizes the objective $\mathrm{KL}\left(q(\boldsymbol{x}) \| p(\boldsymbol{x})\right)$. Given that a prior over functions that can be represented by $\boldsymbol{f}_{\boldsymbol{\eta}}$ is difficult to specify, we consider an uninformative prior on $\boldsymbol{f}_{\boldsymbol{\eta}}$.

Our approach treats this problem from a functional optimization perspective, by first computing the functional gradient of the objective, and then pulling it back to parameter space through the function parameterization. In the rest of this section we introduce our algorithm in detail and compare it with amortized ParVI approaches.



(a) HMC

(b) SVGD

(c) GPVI

(d) Amortized SVGD

Figure 6.1: Predictive uncertainty of methods for a 1-D regression task. (a) HMC predictive posterior matches the uncertainty in the data; (b) SVGD performs comparably to HMC; (c) Our proposed GPVI performs similarly to SVGD, with the additional capability of sampling new particles during inference; (d) Amortized SVGD overestimates the uncertainty when the data is sparse.

Functional gradient and its pullback

Let $\mathcal{J}(\boldsymbol{f}) = \mathrm{KL}\left(q(\boldsymbol{x}) \| p(\boldsymbol{x})\right)$ be the objective, where $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$. If $\boldsymbol{f}$ is injective, by change of variables for probability measure,

$$q(\boldsymbol{x}) = \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}, \tag{6.1}$$

where $p_{\boldsymbol{z}}(\boldsymbol{z})$ is the distribution from which $\boldsymbol{z}$ is sampled. The minimization objective becomes,

$$\mathcal{J}(\boldsymbol{f}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}\right]. \tag{6.2}$$

Consider some function approximation of $\boldsymbol{f}$, say $\boldsymbol{f} = \boldsymbol{f}_{\boldsymbol{\eta}}$, then the minimization objective becomes,

$$\mathcal{J}(\boldsymbol{\eta}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}}\right)\right|}\right]. \tag{6.3}$$

Directly computing the gradient of $\mathcal{J}(\boldsymbol{\eta})$ with respect to $\boldsymbol{\eta}$ involves not only an inverse of the Jacobian $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}$, but also second derivatives of $\boldsymbol{f}_{\boldsymbol{\eta}}$, which is overly expensive to compute in practice.

We propose, instead, to first compute the functional gradient of (6.2) w.r.t. $\boldsymbol{f}$, i.e., $\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})$, and then back-propagate it through the generator to get the gradient w.r.t. $\boldsymbol{\eta}$, i.e.,

$$\nabla_{\boldsymbol{\eta}}\mathcal{J} = \mathbf{E}_{\boldsymbol{z}}\left[\frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{\eta}}\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})(\boldsymbol{z})\right]. \tag{6.4}$$

The following theorem gives an explicit formula for computing the functional gradient $\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})$ when $\boldsymbol{f}$ is chosen from a Reproducing kernel Hilbert space (RKHS).

Theorem 6.2.1. *Let $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$, where $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$, vector function $\boldsymbol{f} = (f^1, \ldots, f^m) \in \mathcal{H}^d$ with $f^i \in \mathcal{H}$, where $\mathcal{H}$ is the RKHS with kernel $k(\cdot,\cdot)$, $\mathcal{H}^m$ is equipped with inner product $\langle \boldsymbol{f}, \boldsymbol{g}\rangle_{\mathcal{H}^m} = \sum_{i=1}^d \langle f^i, g^i\rangle_{\mathcal{H}}$. For $\mathcal{J}(\boldsymbol{f})$ well-defined by (6.2), we have*

$$\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'}\left[-\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}', \boldsymbol{z}) - \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1}\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})\right]. \tag{6.5}$$

Proof To compute the gradient of $\mathcal{J}(\boldsymbol{f}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}\right]$, for any

$\boldsymbol{\phi} \in T_{\boldsymbol{f}}\mathcal{H}$,

$$
\begin{aligned}
d\mathcal{J}_{\boldsymbol{f}}(\boldsymbol{\phi}) &= \frac{d}{dt}\bigg|_{t=0} \mathcal{J}(\boldsymbol{f} + t\boldsymbol{\phi}) \\
&\overset{①}{=} \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0}(-\log p((\boldsymbol{f}+t\boldsymbol{\phi})(\boldsymbol{z})))\right] - \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0}\log\left|\det\left(\frac{\partial(\boldsymbol{f}+t\boldsymbol{\phi})}{\partial \boldsymbol{z}}\right)\right|\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\frac{d}{dt}\bigg|_{t=0}(f^i + t\phi^i)(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\frac{d}{dt}\bigg|_{t=0}\frac{\partial(\boldsymbol{f}+t\boldsymbol{\phi})}{\partial \boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\phi^i(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)^j_i \frac{\partial \phi^i}{\partial z^j}\right] \\
&\overset{②}{=} \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\langle k(\boldsymbol{z},\cdot),\ \phi^i(\boldsymbol{z})\rangle_{\mathcal{H}}\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left\langle\left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)^j_i \nabla_{z^j}k(\boldsymbol{z},\cdot),\ \phi^i(\boldsymbol{z})\right\rangle_{\mathcal{H}}\right] \\
&= \left\langle \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}k(\boldsymbol{z},\cdot) - \left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)^j_i \nabla_{z^j}k(\boldsymbol{z},\cdot)\right], \phi^i\right\rangle_{\mathcal{H}},
\end{aligned}
$$

the following identities are used in step ① and ②,

$$
d\log|\det A| = \mathrm{Tr}(A^{-1}dA),
$$

$$
\phi^i(\boldsymbol{z}) = \langle k(\boldsymbol{z},\cdot), \phi^i(\boldsymbol{z})\rangle_H,
$$

$$
\nabla_{z^j}\phi^i(\boldsymbol{z}) = \langle \nabla_{z^j}k(\boldsymbol{z},\cdot), \phi^i(\boldsymbol{z})\rangle_H.
$$

By definition of the gradient,

$$
\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'}\left[-\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')}k(\boldsymbol{z}',\boldsymbol{z}) - \left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1}\right)\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}',\boldsymbol{z})\right].
$$

6.2.1 Reparameterization of the Generator

There are two considerations for reparameterizing $\boldsymbol{f}$. Firstly, in order for the inverse $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1}$ in (6.5) to be well-defined, the Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$ should be a square matrix, i.e., input noise $\boldsymbol{z}$ should have the same dimension as the output $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$. In practice,

especially those applications involving BNNs, each $\boldsymbol{x}$ represents parameters of a sampled neural network and therefore can be extremely high dimensional. As a result, a high dimensional $\boldsymbol{f}$ can be computationally prohibitive. Secondly, in order for the change of variables density formula (6.1) to hold, $\boldsymbol{f}$ needs to be injective, which is in general not guaranteed for an arbitrary neural network function.

To overcome the above two concerns, we consider the following parameterization,

$$\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}) = \boldsymbol{g}_{\boldsymbol{\eta}}\left(\boldsymbol{z}^{(:d)}\right) + \lambda \boldsymbol{z}, \quad \forall \boldsymbol{z} \in \mathbb{R}^m, \tag{6.6}$$

where $\boldsymbol{z}^{(:d)} \in \mathbb{R}^d$ denotes the vector consisting of the first d components of $\boldsymbol{m}$, and $\boldsymbol{g}_{\boldsymbol{\eta}} : \mathbb{R}^d \to \mathbb{R}^m$ with parameters $\boldsymbol{\eta}$ is a much slimmer neural network. In our experiments, $\boldsymbol{g}_{\boldsymbol{\eta}}$ is designed with an input dimension d less than 30% the size of m. For high dimensional open-category experiments where $m > 60,000$, we use a d of less than 2% of m. For $\boldsymbol{f}_{\boldsymbol{\eta}}$ defined by (6.6), the Jacobian is,

$$\left[\frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}}\right]_{m\times m} = \left[\left[\frac{\partial \boldsymbol{g}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}^{(:d)}}\right]_{m\times d} \middle| \mathbf{0}_{m\times(m-d)}\right]_{m\times m} + \lambda \boldsymbol{I}_m, \tag{6.7}$$

where λ is a hyper-parameter.

Lemma 6.2.2. *If the domain Z of the input noise $\boldsymbol{z}$ is convex, $\boldsymbol{f}_{\boldsymbol{\eta}}$ is defined as Eq.* (6.6)*, and the Jacobian $J_{\boldsymbol{f}}$ is given by Eq.* (6.7) *then $\boldsymbol{f}_{\boldsymbol{\eta}}$ is injective.*

Proof

For any two different points $\boldsymbol{z}_1, \boldsymbol{z}_2 \in Z$, consider the line segment $\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1), t \in [0,1]$, that lie in Z given the convexity of Z. From the Fundamental Theorem of calculus,

$$\begin{aligned}(\boldsymbol{z}_2 - \boldsymbol{z}_1)^T(\boldsymbol{f}(\boldsymbol{z}_2) - \boldsymbol{f}(\boldsymbol{z}_1)) =& (\boldsymbol{z}_2 - \boldsymbol{z}_1)^T \left(\int_0^1 J_{\boldsymbol{f}}(\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1))dt\right) \cdot (\boldsymbol{z}_2 - \boldsymbol{z}_1) \\ =& \int_0^1 (\boldsymbol{z}_2 - \boldsymbol{z}_1)^T J_{\boldsymbol{f}}(\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1))(\boldsymbol{z}_2 - \boldsymbol{z}_1)dt > 0,\end{aligned}$$

where the last inequality is due to the positive definiteness of $J_{\boldsymbol{f}}$ on Z. Therefore, $\boldsymbol{f}(\boldsymbol{z}_1) \neq \boldsymbol{f}(\boldsymbol{z}_2)$, $\boldsymbol{f}$ is injective. In practice we set λ to be 1.0 and find it sufficient throughout our experiments.

6.2.2 Estimating the Jacobian inverse

The main computational challenge of (6.5) lies in computing the term

$$\left(J_{\boldsymbol{f}}(\boldsymbol{z}')\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}), \tag{6.8}$$

where $J_{\boldsymbol{f}}(\boldsymbol{z}') = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$, especially considering that we need an efficient implementation for batched $\boldsymbol{z}$ and $\boldsymbol{z}'$.

Directly evaluating and storing the full Jacobian $J_{\boldsymbol{f}}(\boldsymbol{z}')$ for each $\boldsymbol{z}'$ of the sampled batch is not acceptable from the standpoint of either time or memory consumption. There exist iterative methods for solving the linear equation system $\boldsymbol{y} = (J_{\boldsymbol{f}}(\boldsymbol{z}'))^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ [Young, 1954, Fletcher, 1976], which involves computing the vector-Jacobian product $J_{\boldsymbol{f}}(\boldsymbol{z}')\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ at each iteration. By alternating between this iterative solver and the gradient update (6.4), it is possible to get a computationally amenable algorithm. However, as shown in figure 6.3.1, such an algorithm does not converge to the target distribution p even for a simple Bayesian linear regression task. This is due to the fact that batches of both $\boldsymbol{z}$ and $\boldsymbol{z}'$ for evaluating (6.5) need to be re-sampled for each gradient update (6.4) to avoid the cumulative sampling error. Therefore, the above alternating procedure of iterative solver for $\boldsymbol{y} = (J_{\boldsymbol{f}}(\boldsymbol{z}'))^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ ends up shooting a moving target for different batches of $\boldsymbol{z}$ and $\boldsymbol{z}'$ at each iterate, which is difficult for convergence.

To overcome this computational challenge, we propose a helper network, denoted by $\boldsymbol{h}_{\boldsymbol{\psi}}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k)$, and parameterized by $\boldsymbol{\psi}$, that consumes both $\boldsymbol{z}'$ and $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ and predicts $(J_{\boldsymbol{f}}(\boldsymbol{z}'))^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$. With the helper network, the functional gradient (6.5) can be computed by,

$$\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'} \left[-\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) \Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}', \boldsymbol{z}) - \boldsymbol{h}_{\boldsymbol{\psi}} \left(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})\right) \right]. \tag{6.9}$$

We use the following loss to train the helper network,

$$\mathcal{L}(\boldsymbol{\eta}) = \| J_{\boldsymbol{f}}(\boldsymbol{z}') \boldsymbol{h}_{\boldsymbol{\psi}}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})) - \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}) \|^2, \tag{6.10}$$

where $J_{\boldsymbol{f}}(\boldsymbol{z}')\boldsymbol{h}_{\boldsymbol{\psi}}$ can be computed by the following formula given the reparameteriza-

tion (6.6) of $\boldsymbol{f}_{\boldsymbol{\eta}}$,

$$\boldsymbol{h}_{\boldsymbol{\psi}}^T \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}'} \right) = \boldsymbol{h}_{\boldsymbol{\psi}}^T \left(\frac{\partial \boldsymbol{g}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}'^{(:d)}} \right) + \lambda \boldsymbol{h}_{\boldsymbol{\psi}}^T, \tag{6.11}$$

where the Vector-Jacobian Product (VJP) $\boldsymbol{h}_{\boldsymbol{\psi}}^T \left(\frac{\partial \boldsymbol{g}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}'^{(:d)}} \right)$ can be computed by one backward pass of the function $\boldsymbol{g}_{\boldsymbol{\eta}}$.

6.2.3 Summary of the algorithm

Both $\boldsymbol{g}_{\boldsymbol{\eta}}$ and $\boldsymbol{h}_{\boldsymbol{\psi}}$ can be trained with stochastic gradient descent (SGD), and our algorithm alternates between the SGD updates of $\boldsymbol{g}_{\boldsymbol{\eta}}$ and $\boldsymbol{h}_{\boldsymbol{\psi}}$. Note that the helper network $\boldsymbol{h}_{\boldsymbol{\psi}}$ only has to chase the update of $\boldsymbol{g}_{\boldsymbol{\eta}}$, but no more extra moving targets due to re-sampling of $\boldsymbol{z}$ and $\boldsymbol{z}'$. Our experiments in Sec. 6.3 show that the helper network is able to efficiently approximate (6.8) which enables the convergence of the gradient update (6.4). Our overall Generative Particle VI (GPVI) algorithm is summarized in Algorithm 2.

Algorithm 2 Generative Particle VI (GPVI)

Initialize generator $\boldsymbol{g}_{\boldsymbol{\eta}}$, helper $\boldsymbol{h}_{\psi}$, and learning rate ϵ

while not converged **do**

1. sample two batches $\{\boldsymbol{z}_i\}, \{\boldsymbol{z}'_i\} \sim N(\boldsymbol{0}, I_m)$
2. compute $k(\boldsymbol{z}', \boldsymbol{z})$ and $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$
3. forward $\boldsymbol{g}_{\boldsymbol{\eta}}$ to compute $\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})$ and $\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}')$ by (6.6)
4. forward $\boldsymbol{h}_{\boldsymbol{\psi}}$ to compute $\boldsymbol{h}_{\boldsymbol{\psi}}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}))$
5. backward $\boldsymbol{g}_{\boldsymbol{\eta}}$ to compute the VJP $\boldsymbol{h}_{\boldsymbol{\psi}}^T \left(\frac{\partial \boldsymbol{g}_{\boldsymbol{\eta}}}{\partial \boldsymbol{z}'^{(1:d)}} \right)$
   and then construct $J_{\boldsymbol{f}}(\boldsymbol{z}')\boldsymbol{h}_{\boldsymbol{\psi}}$ by (6.11),
6. update $\boldsymbol{h}_{\boldsymbol{\psi}}$ by $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \epsilon \nabla_{\boldsymbol{\psi}} \mathcal{L}$,
   where $\nabla_{\boldsymbol{\psi}} \mathcal{L}$ is computed by back-propagating (6.10)
7. compute the functional gradient by (6.9)
8. update $\boldsymbol{\eta}$ by $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} - \epsilon \nabla_{\boldsymbol{\eta}} \mathcal{J}$,
   where $\nabla_{\boldsymbol{\eta}} \mathcal{J}$ is computed by (6.4)

end

6.2.4 Comparison with Amortized SVGD

Stein variational gradient descent (SVGD) represents $q(\boldsymbol{x})$ by a set of particles $\{\boldsymbol{x}_i\}_{i=1}^n$, which are updated iteratively by,

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon \boldsymbol{\phi}^*(\boldsymbol{x_i}), \tag{6.12}$$

where ϵ is a step size and $\boldsymbol{\phi}^* : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field (perturbation) on the space of particles that corresponds to the optimal direction to perturb particles, i.e.,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi} \in \mathcal{F}} \left\{ \frac{d}{d\epsilon} \mathrm{KL}(q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x}) \| p(\boldsymbol{x})) \Big|_{\epsilon=0} \right\},$$

where $q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x})$ denotes the density of particles updated by (6.12) using the perturbation $\boldsymbol{\phi}$, where the density of original particles is $q(x)$. When $\mathcal{F}$ is chosen to be the unit ball of some RKHS $\mathcal{H}$ with kernel function $k(\cdot, \cdot)$, SVGD gives the following closed form solution for $\boldsymbol{\phi}^*$,

$$\boldsymbol{\phi}^*(\boldsymbol{x}) = \mathbf{E}_{\boldsymbol{x}' \sim q} \left[\nabla_{\boldsymbol{x}'} \log p(\boldsymbol{x}') k(\boldsymbol{x}', \boldsymbol{x}) + \nabla_{\boldsymbol{x}'} k(\boldsymbol{x}', \boldsymbol{x}) \right]. \tag{6.13}$$

To turn SVGD into a neural sampler, Amortized SVGD [Wang and Liu, 2016] first samples particles from a generator and then back-propagates the particle gradients (6.13) through the generator to update the generator parameters. Let $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}),\ \boldsymbol{z} \sim N(\mathbf{0}, I_m)$ be the particle generating process, where $\boldsymbol{f}_{\boldsymbol{\eta}}$ is the generator parameterized by $\boldsymbol{\eta}$, amortized SVGD updates $\boldsymbol{\eta}$ by,

$$\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \epsilon \sum_{i=1}^{n} \frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i)}{\partial \boldsymbol{\eta}} \boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i)), \tag{6.14}$$

where $\boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z}_i))$ is computed by (6.13).

The following Lemma gives an explicit view regarding what functional gradient amortized SVGD back-propagates through the generator.

Lemma 6.2.3. *If particles are generated by* $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z}),\ \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$, *eq.* (6.13) *is the*

functional gradient of the objective w.r.t. the perturbation function $\boldsymbol{\phi} : \mathbb{R}^m \to \mathbb{R}^m$*, i.e.,*

$$\nabla_{\boldsymbol{\phi}}\Big|_{\boldsymbol{\phi}=id} KL\left(q(\boldsymbol{x})\|p(\boldsymbol{x})\right) = -\boldsymbol{\phi}^*, \tag{6.15}$$

where $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{f}(\boldsymbol{z}))$ *and* $\boldsymbol{\phi} = (\phi^1, \ldots, \phi^m) \in \mathcal{H}^m$ *with* $\phi^i \in \mathcal{H}$*, where* $\mathcal{H}$ *is the RKHS with kernel* $k(\cdot,\cdot)$*,* $\mathcal{H}^m$ *is equipped with inner product* $\langle \boldsymbol{\phi}, \boldsymbol{\xi}\rangle_{\mathcal{H}^m} = \sum_{i=1}^m \langle \phi^i, \xi^i\rangle_{\mathcal{H}}$*.*

Proof

To compute the gradient of $\mathcal{J}(\boldsymbol{\phi}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{\phi}\left(\boldsymbol{f}(\boldsymbol{z})\right)) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial(\boldsymbol{\phi}\circ\boldsymbol{f})}{\partial \boldsymbol{z}}\right)\right|}\right]$ at $\boldsymbol{\phi} = \mathrm{id}$, for any $\boldsymbol{v} \in T_{\boldsymbol{\phi}}\mathcal{H}$,

$$\begin{aligned}
d\mathcal{J}_{\boldsymbol{\phi}}(\boldsymbol{v}) &= \frac{d}{dt}\Big|_{t=0} \mathcal{J}(\boldsymbol{\phi} + t\boldsymbol{v})\Big|_{\boldsymbol{\phi}=\mathrm{id}} \\
&= \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\Big|_{t=0}\left(-\log p((\boldsymbol{f} + t\boldsymbol{v}\circ\boldsymbol{f})(\boldsymbol{z}))\right)\right] - \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\Big|_{t=0}\log\left|\det\left(\frac{\partial(\boldsymbol{f}+t\boldsymbol{v}\circ\boldsymbol{f})}{\partial\boldsymbol{z}}\right)\right|\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\frac{d}{dt}\Big|_{t=0}(f^i + t(\boldsymbol{v}\circ\boldsymbol{f})^i(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{z}}\right)^{-1}\frac{d}{dt}\Big|_{t=0}\frac{\partial(\boldsymbol{f}+t\boldsymbol{v}\circ\boldsymbol{f})}{\partial\boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} v^i(\boldsymbol{f}(\boldsymbol{z}))\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left(\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{z}}\right)^{-1}\right)^j_i \frac{\partial f^k}{\partial z^j}\frac{\partial v^i}{\partial x^k}\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\langle k(\boldsymbol{f}(\boldsymbol{z}),\cdot),\ v^i(\boldsymbol{f}(\boldsymbol{z}))\rangle_{\mathcal{H}}\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left\langle \nabla_{x^i}k(\boldsymbol{x},\cdot)\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})},\ v^i(\boldsymbol{f}(\boldsymbol{z}))\right\rangle_{\mathcal{H}}\right] \\
&= \left\langle \mathbf{E}_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\left[-\nabla_{x^i}\log p(\boldsymbol{x})k(\boldsymbol{x},\cdot) - \nabla_{x^i}k(\boldsymbol{x},\cdot)\right], v^i\right\rangle_{\mathcal{H}}.
\end{aligned}$$

By definition of the gradient,

$$\nabla_{\boldsymbol{\phi}}\mathcal{J}(\boldsymbol{\phi})(\boldsymbol{x})\Big|_{\boldsymbol{\phi}=\mathrm{id}} = \mathbf{E}_{\boldsymbol{x}'}\left[-\nabla_{\boldsymbol{x}'}\log p(\boldsymbol{x}')k(\boldsymbol{x}',\boldsymbol{x}) - \nabla_{\boldsymbol{x}'}k(\boldsymbol{x}',\boldsymbol{x})\right],$$

where $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$ and $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{z}')$.

Therefore, rather than back-propagating the functional gradient with respect to the generator function as our approach, the amortized SVGD back-propagates the functional gradient with respect to a perturbation function applied after the generator, which is in

general not the steepest descent direction for the generator function $\boldsymbol{f}$, except for the situation $\boldsymbol{f} = \mathrm{id}$, where (6.5) and (6.13) are equivalent because $\boldsymbol{z} = \boldsymbol{x}$ and the Jacobian is identity. Note that such difference in functional gradient computation leads to different choices of RKHS kernel between (6.5) and (6.13).

The amortizing step applied in both methods, i.e., back-propagating some functional gradient to update the generator parameters, is in general not guaranteed to keep the original descent direction of the functional gradient. However, GPVI amortizes the steepest descent direction which is optimal in first order sense, while amortized SVGD amortizes a non-steepest descent direction, which is more likely to result in a non-descending direction after amortizing. In practice, as shown in the next section, our approach consistently outperforms amortized SVGD in approximating the target distribution and capturing model uncertainty.

6.3 Experiments

To demonstrate the effectiveness of our approach for approximate Bayesian inference, we evaluated GPVI in two different settings: learning from a likelihood function, and direct generative modeling. In the former setting, we know how to compute $\log p(\boldsymbol{x})$, and we are able to exactly optimize according to Eq.(6.5). In the direct generative modeling case, we have only samples from the target distribution. This is the setting of image generation, and similar to GANs, we resort to an adversarial discriminator to approximate the likelihood function.

6.3.1 Likelihood-based Generative Modeling

In this setting we evaluate GPVI for both density estimation and Bayesian neural networks. In our density estimation experiment, we trained our generator to draw samples from a target distribution, showing that GPVI learns an accurate posterior fit by maximizing the likelihood of samples. For our BNN experiments, we evaluated on regression, classification, and high dimensional open-category tasks. Our experiments show that among all methods compared, GPVI is the only method to excel in both sampling efficiency and asymptotic performance. We compare GPVI with ParVI methods: SVGD [Liu and Wang, 2016a], GFSF [Liu et al., 2019], and KSD [Hu et al., 2018, Grathwohl

et al., 2020], as well as their corresponding amortized versions. We also compared with Bayes by Backprop (BBB) [Blundell et al., 2015], deep ensembles [Lakshminarayanan et al., 2017b], and HMC [Neal et al., 2011]. We emphasize that our aim is not to only maximize the likelihood of generated samples, rather we want to closely approximate the posterior of parameterized functions given the data. Thus, predictions w.r.t data unseen during training should reflect the epistemic uncertainty of the posterior.

For BNN experiments, we parameterized samples from the target distribution as neural networks with a fixed architecture. In the low dimensional regression and classification settings, we drew 100 samples from the approximate posterior for both training and evaluation, allowing us to compute the predictive mean and variance. For methods without a sampler e.g. ParVI and deep ensembles, we initialized a 100 member ensemble. In the high dimensional open-category tasks, we instead used 10 samples due to the larger computation cost. For methods that utilize a hypernetwork such as GPVI and amortized ParVI, we used Gaussian input noise $\boldsymbol{z} \sim \mathcal{N}(0, I)$ and varied the hypernetwork architecture depending on the task. For ParVI and deep ensembles we randomly initialized each member of the ensemble. In all methods except HMC we used the Adam optimizer [Kingma and Ba, 2014] to train any neural networks.

For HMC we use the same settings for the Bayesian linear regression and BNN tasks. We sampled the momentum from a standard normal distribution, and used 25 leap-frog steps per sample. We ran each experiment for 25K steps in total, with a burn-in of 20K steps, and a step size of 0.0005. We thinned each chain after burn-in, and tuned the number of leapfrog steps and step-size for each experiment. To assess convergence, we checked that the means of each chain were similar, indicating mixing.

Density Estimation

We first evaluate the ability of generative approaches to fit a target distribution from data. We used 2D and 5D zero mean Gaussian distributions with non-diagonal covariances as our target distributions. The target covariance we wish to fit is computed as $\Sigma^* = \Sigma\Sigma^T, \Sigma \sim \mathcal{N}(0, I_m), m \in \{2, 5\}$. We use an MLP with 1 hidden layer to approximately sample from a unimodal Gaussian distribution with covariance Σ^*. Given Gaussian input noise, the variance of the output distribution of the linear generator

Method	Σ error (2d) $\downarrow$	Σ error (5d) $\downarrow$
Amortized SVGD	$0.10 \pm .09$	$0.37 \pm .32$
Amortized GFSF	$0.18 \pm .04$	$0.21 \pm .13$
Amortized KSD	$0.28 \pm .32$	$1.68 \pm .52$
GPVI Exact Jac	$0.15 \pm .09$	$0.49 \pm .17$
GPVI	$0.14 \pm .08$	$0.14 \pm .04$

Table 6.1: Comparison of generative Particle VI approaches for density estimation of 2d and 5d Gaussian distributions.

is computed as $W^T W$, which should match Σ^* after training. As shown in Table 6.1, the estimation problem becomes harder with increased dimension, yet GPVI performs consistently well while amortized ParVI methods suffer more or less from a performance drop and end up inferior to our approach. We also compare GPVI with a variant where the Jacobian and its inverse are explicitly computed. The inverse is computed with the PyTorch [**?**] function `torch.inverse`, which uses LAPACK routines `getrf` and `getri`. We can see that GPVI is competitive with the "Exact-Jac" variant in the 2D setting, and even performs better in the 5D setting. This is unexpected, as explicitly computing the Jacobian and its inverse should be the correct way to compute the functional gradient. We hypothesize that the Jacobian of our generator network is ill-conditioned. In this case any inversion algorithm is more prone to large numerical error. Because our helper network is updated once per training iteration, we may avoid the large gradients that come from numerical error. The increased performance of GPVI over the "Exact-Jac" variant in the 5D setting may be due to this smoother training.

Next, we evaluate GPVI on the four non-Gaussian energy potentials defined in Rezende and Mohamed [2015]. In figure 6.2, we see (from left to right) the target density, GPVI, amortized SVGD, and normalizing flows [Rezende and Mohamed, 2015]. We found that while our method has more parameters, we don't need nearly as deep a model as with normalizing flows. We used just 2 hidden layers for GPVI, while we needed a flow of depth 32 to get comparable performance. We trained each method for 200000 steps with a batch size of 100. To generate plots, we sampled 20000 points from each model. We detail the rest of the hyperparameters in section 6.5. In figures 6.2 we see that GPVI is able to capture the variance of the target distribution, while amortized SVGD samples mostly from the mean. For the Sin Bisect setting, GPVI is the only method that

captures some of both halves of the middle section. But in Sin Split, only normalizing flows capture both halves of the split section.

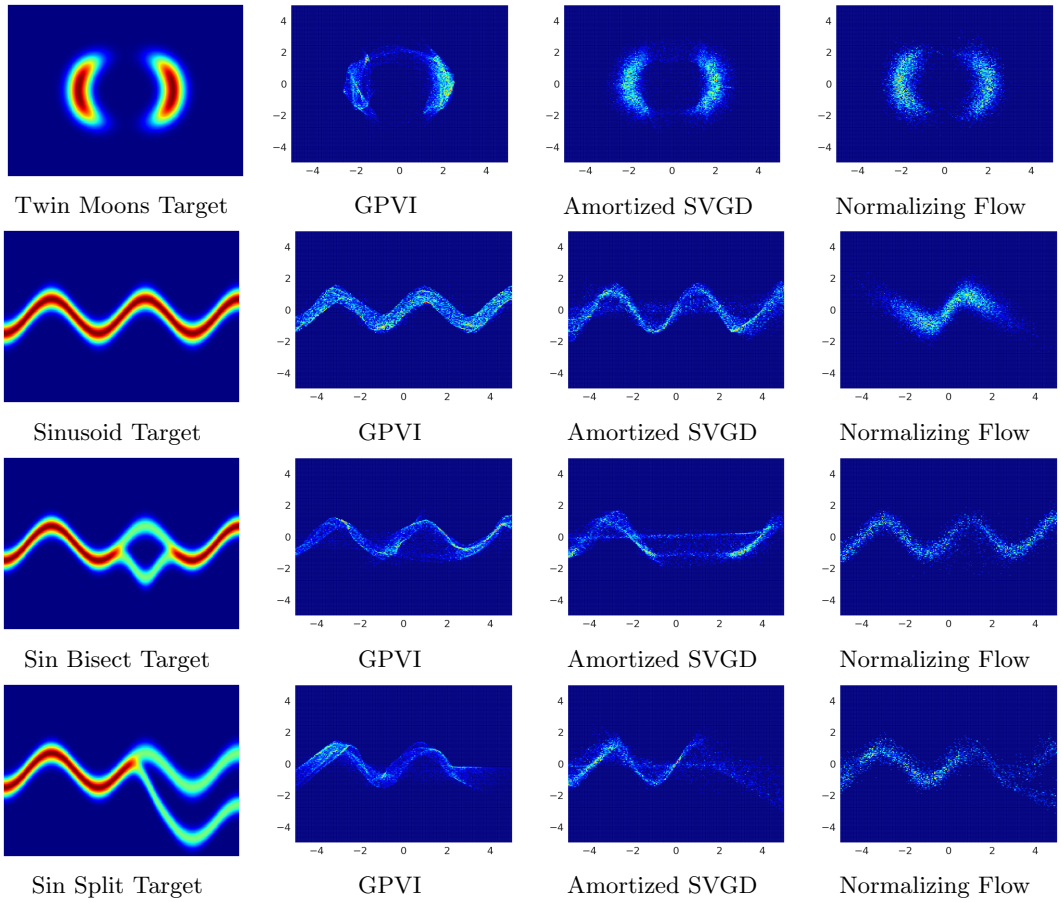


Figure 6.2: Density estimation with the energy potential functions from Rezende and Mohamed [2015].

1D Regression

In figure (6.1), we show quantitative results for GPVI, HMC, SVGD, and amortized SVGD applied to learning a distribution over 1D regression functions. We generate a dataset of 80 samples X with targets Y. We draw 76 samples of X uniformly from $[-6, -2] \cup [2, 6]$, and draw the 4 remaining samples from $[-2, 2]$. The targets Y are

computed as $Y = -(1 + X)\sin(1.2X) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.04)$. For all methods we use 100 posterior samples, and train for 50K iterations. For GPVI and amortized SVGD we use a generator with two linear layers $[32, 32]$ and Gaussian input noise of the same dimension. We can see that HMC and SVGD both give reasonable uncertainty estimates, given the variance in the observed data. GPVI learns an approximate posterior that is competitive with SVGD, while amortized SVGD overestimates the variance.

Bayesian Linear Regression

We evaluated all methods on Bayesian linear regression to investigate how well each method can fit a unimodal normal distribution over linear function weights. The target function is a linear regressor with parameter vector $\boldsymbol{\beta} \in \mathbb{R}^d$, i.e., $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \epsilon$, where $\boldsymbol{X} \sim \mathcal{N}(0, I_d)$, $\epsilon \sim \mathcal{N}(0, I)$, and $\beta^i \sim U(0, 1) + 5$. We chose such linear Gaussian settings where we can explicitly compute the target posterior $p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{N}((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \boldsymbol{X}^T\boldsymbol{X})$ given observations $(\boldsymbol{X}, \boldsymbol{y})$, allowing us to numerically evaluate how well each method fits the target distribution. Each method was trained to regress $\boldsymbol{y}$ from $\boldsymbol{X}$. For GPVI and amortized ParVI methods that make use of a hypernetwork, we used a linear generator with input noise $\boldsymbol{z} \in \mathbb{R}^d$. In this setting, the output of a generator with bias $\boldsymbol{b}$ and weights W follows the distribution $\mathcal{N}(\boldsymbol{b}, WW^T)$, which should match the posterior $p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y})$. For ParVI methods and deep ensembles, we initialized an ensemble of linear regressors with d parameters. We computed the mean and covariance of the learned parameters to measure the quality of posterior fit. For BBB we chose a standard normal prior on the weights, and likewise computed the mean and covariance of weight samples to measure posterior fit. We set $d = 3$ in our experiments. While simple, this quantitative sanity check is crucial before evaluating on more complicated domains where the true posterior is not available. It will be evident in later experiments that the ability to closely approximate the true posterior under this simple setup is indicative of performance on higher dimensional tasks.

Results are shown in table 6.2. Our GPVI outperforms amortized ParVI, BBB, and surprisingly even HMC. GPVI is also very competitive with the best ParVI methods, SVGD and GFSF. Finally, as deep ensembles lacks any mechanism for correctly estimating the covariance, it fails as expected.

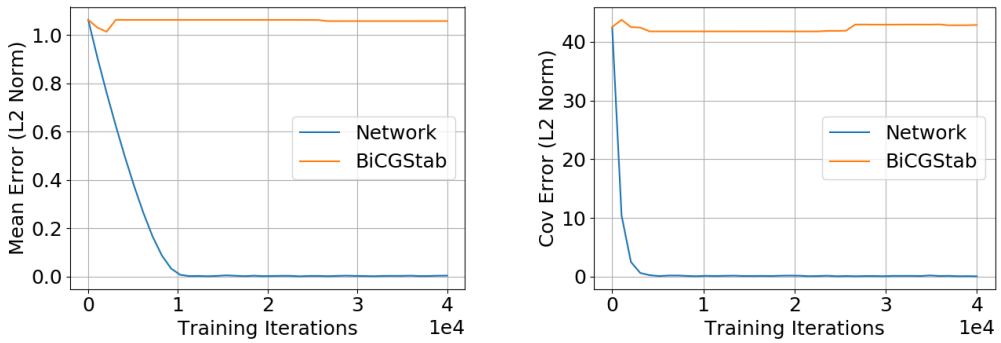
Method	μ error $\downarrow$	Σ error $\downarrow$
SVGD	0.006 $\pm$ .0024	**0.125** $\pm$ **.03**
GFSF	0.003 $\pm$ .0014	0.139 $\pm$ .07
KSD	0.009 $\pm$ .0006	0.373 $\pm$ .11
Amortized SVGD	**0.002** $\pm$ **.0009**	0.158 $\pm$ .04
Amortized GFSF	**0.002** $\pm$ **.0003**	0.209 $\pm$ .05
Amortized KSD	0.004 $\pm$ .0007	0.430 $\pm$ .10
BBB	0.004 $\pm$ .0035	0.303 $\pm$ .04
Deep Ensemble	0.004 $\pm$ .0002	1.0 $\pm$ 0
HMC	0.009 $\pm$ .0003	0.181 $\pm$ .05
GPVI	**0.002** $\pm$ **.0007**	**0.128** $\pm$ **.04**
GPVI Exact Jac	**0.002** $\pm$ **.0004**	**0.106** $\pm$ **.07**

Table 6.2: Bayesian linear regression. Reported error is the L_2 norm of the difference between the learned mean and covariance parameters, and the ground truth after 50000 iterations.

Comparing with Linear Solver for the Inverse Vector Jacobian Product

In the same Bayesian linear regression setting, we justify the use of our helper network $\boldsymbol{h}_\psi$, by comparing its performance to a more traditional linear solver: the stabilized biconjugate gradient method (BiCGSTAB) [Saad, 2003]. BiCGSTAB is a well-known iterative algorithm for solving systems of the form $Ax = b$. It is similar to the conjugate gradient method, but does not require A to be self-adjoint, giving BiCGSTAB wider applicability. In figure 6.3, we see the results of GPVI with the helper network ("Network"), against BiCGSTAB. For BiCGSTAB, we solve $B * B$ systems of the form $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}) = \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ for each iteration of training our generator, where $B * B$ is the effective batch size of $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$. Due to the greatly increased training time from solving $B * B$ independent problems per training iteration, we only run BiCGSTAB for one step, and warm start from the previous solution at each new generator training iteration. In addition to increased training time, BiCGSTAB suffers from instability due to the constantly changing $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$ as well as $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$, due to the resampling of $\boldsymbol{z}$ and $\boldsymbol{z}'$ at each step. As seen in figure 6.3, when using BiCGSTAB, GPVI is unable to fit the target distribution, while using our helper network ("Network") we are able to efficiently minimize the mean and covariance error.

We also considered using a normalizing flow as a replacement for our generator network, as normalizing flows are invertible with lower triangular Jacobians. Unfortunately, the efficiency of normalizing flows comes from setting the input-output dimensionality to be equal to force the Jacobian to be square. When using our method for BNNs, we cannot afford to store a generator with equal input-output dimensionality, making normalizing flows an inefficient choice at best for our own method. Our helper network stands as an efficient, novel solution for explicitly estimating the Jacobian inverse vector product when needed. Nevertheless, we compare GPVI with normalizing flow method for density estimation in the next section.



(a) Mean Error

(b) Cov Error

Figure 6.3: Comparing our helper network with BiCGSTAB in the Bayesian linear regression setting.

Multimodal Classification

While the Bayesian linear regression setting comes with an analytically known target distribution, most problems of interest involve distributions without closed-form representations. Therefore, we further tested on a 2-dimensional, 4-class classification problem, where each class consists of samples from one component of a mixture distribution. The mixture distribution is defined as $p(x) = \sum_{i=1}^{4} \mathcal{N}(\mu_i, 0.3)$, with means $\mu_i \in \{(-2,-2), (-2,2), (2,-2), (2,2)\}$. We assigned labels $y_i \in \{1,2,3,4\}$ according to the index of the mixture component the samples were drawn from. For this task, samples are weight parameters $\boldsymbol{\theta}$'s representing classification functions, which are two-layer neural networks with 10 hidden units in each layer and ReLU activations, denoted by

$f_{\boldsymbol{\eta}} : \mathbb{R}^2 \to \mathbb{R}^4$.

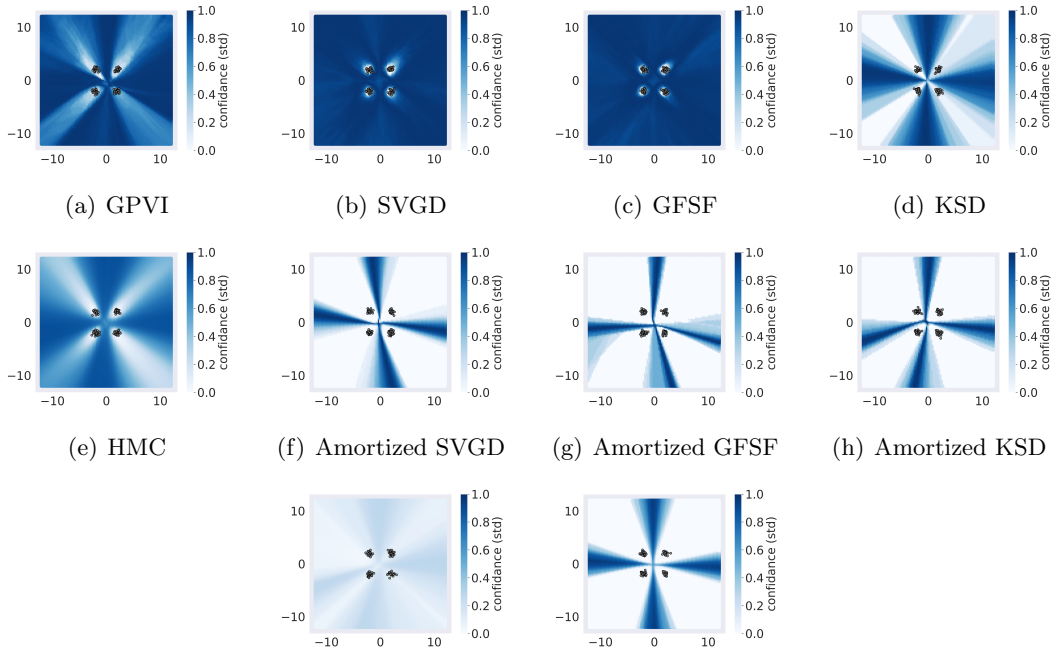
To train each method, we drew a total of 100 training points, and 200 testing points from the target distribution. To evaluate the posterior predictive distribution, we drew points from a grid spaced from $\{-10, 10\}$, then plot the predictive distribution as measured by the standard deviation in predictions among model samples in figure 6.4.

In this setting, the true posterior $p(\boldsymbol{\theta}|\boldsymbol{X}, y)$ is unknown and past work often relies on "gold-standard" approaches like HMC to serve as the ground truth. In the previous Bayesian linear regression tests, however, we saw that HMC was outperformed by GPVI as well as ParVI methods SVGD and GFSF. In the current test, intuitively, a "gold-standard" approach should yield a predictive distribution with high variance (high uncertainty) in regions far from the training data, and low variance (low uncertainty) in regions near each mixture component. As shown in figure 6.4, GPVI and SVGD both have higher uncertainty in no-data regions than HMC, while remaining confident on the training data. On the other hand, BBB and amortized SVGD both dramatically underestimate the uncertainty. The performance of BBB is as expected, as mean-field VI approaches are known to underestimate uncertainty.

We also show results on a simpler two class variant in figure 6.5. In this setting, we generate data in the same way as in the four-class setting, but we use a mixture distribution with two components. Specifically, the mixture distribution is defined as $p(\boldsymbol{x}) = \sum_{i=1}^{2} \mathcal{N}(\mu_i, 0.3)$, with means $\mu_i \in \{(-2, -2), (2, 2)\}$. We assigned labels $y_i \in \{1, 2\}$ according to the index of the mixture component the samples were drawn from. We show the results of each method in figure 6.5. We can see that GPVI again gives better uncertainty estimates than other sampling based approaches. Again, we do not know what the true posterior over classifications looks like, but GPVI and RKHS-based ParVI approaches give uncertainty estimates that closely match our intuition for this problem.

Open Category Prediction

To evaluate the scalability of our approach we turn to large-scale image classification experiments. In this setting, it is not possible to exactly measure the accuracy of posterior



(a) GPVI (b) SVGD (c) GFSF (d) KSD

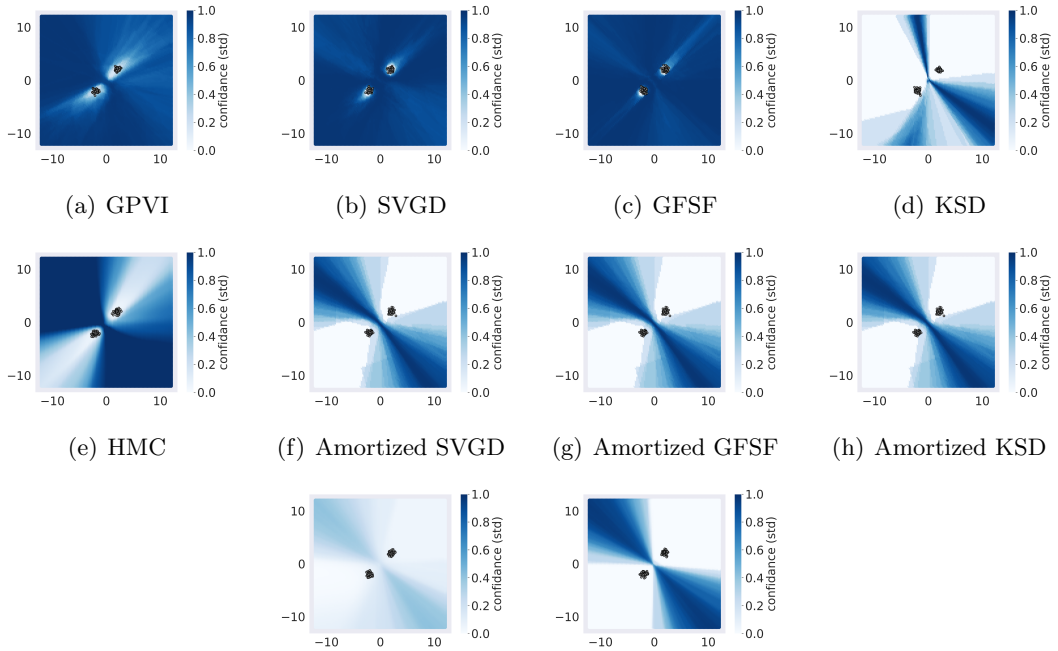
(e) HMC (f) Amortized SVGD (g) Amortized GFSF (h) Amortized KSD

(i) BBB (j) Deep Ensembles

Figure 6.4: Predictive uncertainty of each method on the 4-class classification task, as measured by the standard deviation between predictions of sampled functions. Regions of high uncertainty are shown as darker, while lighter regions correspond to lower uncertainty. The training data is shown as samples from four unimodal normal distributions. It can be seen that amortized SVGD, BBB and deep ensembles significantly underestimate the uncertainty in regions with no training data

fit. We utilize the open-category task to test if our uncertainty estimations can help detect outlier examples. The open category task defines a set of inlier classes that are seen during training, and a set of outlier classes only used for evaluation. While in principle, the content of the outlier classes can be arbitrary, we use semantically similar outlier classes by splitting the training dataset into inlier and outlier classes. This setup is in general more difficult than out-of-distribution experiments where different datasets are used as outlier classes, since distributions of categories from the same dataset may be harder to discriminate.

We evaluated on the MNIST and CIFAR-10 image datasets, following Neal et al. [2018] to split each dataset into 6 inlier classes and 4 outlier classes . We performed standard fully



(a) GPVI (b) SVGD (c) GFSF (d) KSD

(e) HMC (f) Amortized SVGD (g) Amortized GFSF (h) Amortized KSD

(i) BBB (j) Deep Ensembles

Figure 6.5: Predictive uncertainty of each method on the 2-class classification task, as measured by the standard deviation between predictions of sampled functions.

supervised training on the 6 inlier classes, and measured uncertainty in the 4 unseen outlier classes. We evaluated the uncertainty using two widely-used statistics: area under the ROC curve (AUC), and the expected calibration error (ECE). The AUC score measures how well a binary classifier discriminates between predictions made on inlier inputs, vs predictions made on outlier inputs. A perfect AUC score of 1.0 indicates a perfect discrimination, while a score of 0.5 indicates that the two sets of predictions are indistinguishable. ECE partitions predictions into equally sized bins, and computes the L1 difference in expected accuracy and confidence between bins, which represents the calibration error of the bin. ECE computes a weighted average of the calibration error of each bin. Together AUC and ECE tell us how well a model can detect outlier inputs, as well as how well the model fits the training distribution.

MNIST consists of 70,000 grayscale images of handwritten digits at 28x28 resolution, di-

vided into 60,000 training images and 10,000 testing images. We further split the dataset by only using the first six classes for training and testing. The remaining four classes are only used to compute the AUC and ECE statistics. We chose the LeNet-5 classifier architecture for all models, and trained for 100 epochs. Due to the larger computational burden, we only consider 10 samples from each method's approximate posterior for both training and evaluation. Note that our approach is capable of generating many more, but it would be computationally costly to train much more for ensembles and particle VI approaches. For GPVI and amortized ParVI methods we used a 3 layer MLP hypernetwork with layer widths $[256, 512, 1024]$, ReLU activations, and input noise $\boldsymbol{z} \in \mathbb{R}^{256}$. We did not test HMC in this setting, as the computational demand is too high.

Table 6.3 shows the results. All RKHS-based methods (GPVI, ParVI, amortized ParVI) as well as deep ensembles achieve high supervised ("clean") accuracy with the LeNet architecture. BBB underfits slightly, while KSD struggled to achieve competitive accuracy even after 100 epochs. All methods achieved an AUC over 0.95, but SVGD, GFSF and GPVI have the highest AUC scores, respectively. In terms of calibration, GPVI and RKHS-based ParVI methods are the best calibrated. KSD is the worst calibrated model, and deep ensembles/BBB have middling performance.

Method	Clean	AUC↑	ECE ↓
SVGD	99.3	**.989 ± .001**	**.001 ± .0002**
GFSF	99.2	**.988 ± .003**	.002 ± .0003
KSD	97.7	.964 ± .005	.014 ± .0007
Amortized SVGD	99.1	.958 ± .015	**.002 ± .0007**
Amortized GFSF	99.2	.978 ± .005	.004 ± .0013
Amortized KSD	97.7	.951 ± .008	.017 ± .0010
BBB	98.6	.951 ± .008	.014 ± .0027
Deep Ensemble	99.3	.972 ± .002	.008 ± .0060
GPVI	99.3	**.988 ± .001**	**.001 ± .0005**

Table 6.3: Results for open-category classification on MNIST. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers.

CIFAR-10 consists of 60,000 RGB images depicting 10 object classes at 32x32 resolution, divided into 50,000 training images and 10,000 testing images. We adopted the

Method	Clean	AUC ↑	ECE ↓
SVGD	80.3	**.683 ± .008**	.055 ± .004
GFSF	80.6	**.681 ± .004**	.068 ± .012
Amortized SVGD	71.12	.636 ± .018	.073 ± .029
Amortized GFSF	71.09	.583 ± .007	.042 ± .029
BBB	70.0	.649 ± .006	**.016 ± .002**
Deep Ensemble	73.54	.652 ± .018	.033 ± .011
GPVI	76.2	.677 ± .008	**.018 ± .015**

Table 6.4: Open-category classification on CIFAR-10. We show results of standard supervised training (Clean), as well as AUC and ECE of each method trained in the open-category setting.

same 6 inlier / 4 outlier split used in Neal et al. [2018] for the open-category setting. For this task we used a CNN with 3 convolutional layers and two linear layers, which is much smaller than SOTA classifiers for CIFAR-10. Though the classification accuracy would suffer, it allows us to clearly evaluate our method without considering interactions with architectural components such as BatchNorm or residual connections. We also used 10 samples from each method's approximate posterior for training and evaluation. For GPVI and amortized ParVI methods we used the same hypernetwork architecture as in the MNIST setup. Table 6.4 shows the results where it can be seen that our GPVI almost match the performance of SVGD and GFSF in terms of AUC while doing a bit better on ECE.

6.3.2 Direct Generative Modeling

In this section we show how we can use GPVI for direct generative modeling, where we do not have an explicit likelihood function defined. This setting arises naturally when considering data such as natural images; to learn to generate natural images we must learn to approximate the likelihood. Generative adversarial networks (GANs) are an example of a family of methods that does exactly this. Similar to our method, GANs employ an implicit generative model $\boldsymbol{f_\eta}$ that takes as input realizations of a random variable $\boldsymbol{z} \sim Z \subset \mathbb{R}^m$. the input $\boldsymbol{z}$ is mapped to the output data space $X \subseteq \mathbb{R}^m$ as $\boldsymbol{x} = \boldsymbol{f_\eta}(\boldsymbol{z})$. To fit the target distribution, GANs forgo an explicit likelihood in favor of an adversarial discriminator. In GAN [Goodfellow et al., 2014] and DCGAN

[**?**], the discriminator function $D : X \to (0, 1)$ maps data samples to class-conditional probabilities i.e. $p(y = 1|x)$. The discriminator is trained with the following loss to discriminate between generated ($y = 0$) and real ($y = 1$) samples,

$$L(D) = \mathbf{E}_{x \sim p_{data}} \log D(\boldsymbol{x}) + \mathbf{E}_{x \sim \boldsymbol{f_\eta}} \log(1 - D(\boldsymbol{x})), \tag{6.16}$$

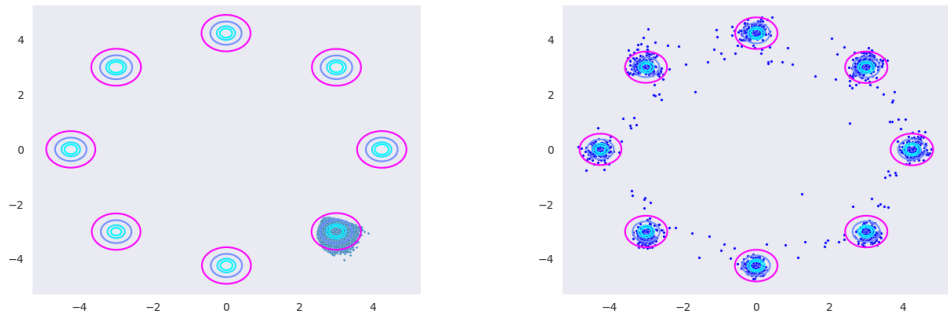
while the generator is trained to generate realistic enough samples to fool the discriminator. At optimality, the discriminator learns a lower bound on the likelihood. To see this, note that $p(\boldsymbol{x}|y) = p(y|\boldsymbol{x}) + p(\boldsymbol{x}) - p(y)$, where $p(y)$ is a constant.

We adapt GPVI to this setting by learning to generate data purely from observed samples. To train GPVI, computing the functional gradient (**??**) requires the evaluation of $\log p(x|y = 1)$, which is unavailable in this setting. We instead opt for a surrogate to the likelihood, and make use of the aforementioned adversarial discriminator as an approximation. In the following experimental settings, we evaluate GPVI as in Sec. 6.3.1 but train a discriminator D to evaluate generated samples. This discriminator is trained with the loss given by (6.16), as in GAN and DCGAN.

Synthetic Data

We evaluate GPVI for learning to fit multimodal 2D data drawn from mixture distributions with 8 and 25 components. We compare GPVI to the corresponding GAN models, to evaluate quantitatively well each method fits the target distribution. Our goal is to show that even with a surrogate likelihood, GPVI learns a close approximation to the data distribution. Given that the GAN generator only maximizes the data probability according to the discriminator, we expect the data distribution learned by GANs will overfit to the modes.

In figure 6.6, we show the results of learning to sample from a mixture of 8 2D Gaussian distributions. We can see that GAN methods suffer from various pathologies that are undesirable in practice. Figure 6.6(a) shows that GAN only fits a single mode of the target distribution, which is consistent with the results reported by Srivastava et al. [2017]. This effect is commonly known as mode-collapse, and is a well known problem in vanilla GANs [Arjovsky and Bottou, 2017]. In contrast, figure 6.6(b) qualitatively show



(a) GAN

(b) GPVI (GAN)

Figure 6.6: GPVI vs GAN for direct generative modeling of 8 Gaussians.

Method	# Modes ↑	|Mean error| ↓	|STD error| ↓
GAN	1	.05	2.62
GPVI (GAN)	8	**.003**	**1.08**

Table 6.5: Error metrics for each method in fitting 8 Gaussians. We show the average absolute error across modes, in estimating the mean and standard deviation of each component.

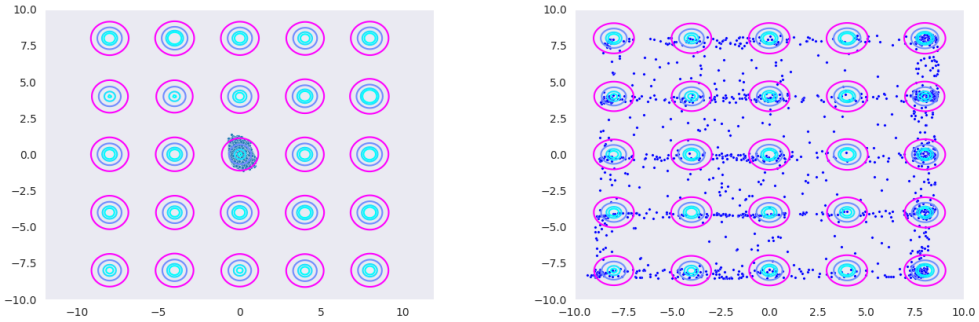
that GPVI is able to consistently fit the target distribution better than corresponding GAN approaches. We also show a quantitative evaluation of the distribution learned by each method in table 6.5. Given that the target distribution is a mixture of Gaussians, we can evaluate an approximating distribution by measure the error in the first two moments. For each method, we sample $20,000$ points, and sort each point into a bin corresponding to the nearest mode. We report the average absolute difference in mean ("Mean Error") and standard deviation ("STD Error") from each component of the target distribution. Table 6.5 shows that GPVI improves on GAN approaches with respect to both metrics. In particular, the gain over standard GAN is significant given that the standard GAN only fits to a single mode.

Next we present results for an analogous experiment on a 25 component mixture distribution. In figure 6.7 we can see that GAN undergoes mode-collapse, only learning to sample from a single component. Qualitatively, while GPVI does not achieve as close an

Method	# Modes ↑	| Mean error | ↓	STD error ↓
GAN	1	.09	1.75
GPVI (GAN)	25	**.004**	**0.30**

Table 6.6: Error metrics for each method in fitting 25 Gaussians. We show the average absolute error across modes, in estimating the mean and standard deviation of each component.

approximation to the target as in the previous setting with 8 components, it still outperforms GAN in terms of resembling the target distribution. Quantitatively, according to table 6.6, GPVI learns a closer approximation to the target distribution the standard GAN approach in this more difficult setting.



(a) DCGAN

(b) GPVI (DCGAN)

Figure 6.7: GPVI vs GAN for direct generative modeling of 25 Gaussians.

Discussion

The quality of generative models can be difficult to assess. For this reason we evaluated GPVI in a range of experimental settings, from fitting explicitly known distributions, to Bayesian neural networks, even to learning from samples with a surrogate likelihood. We believe that the experiments shown in section 6.3 demonstrate that GPVI is a flexible, robust approach to generative modeling.

Much of the focus regarding recent work on Bayesian neural networks concerns their

performance on open-category and out-of-distribution tasks with high dimensional image datasets. Instead, we show that our method closely approximates the target posterior, both in tasks where the posterior is explicitly known, as well as when it is intractable. The Bayesian linear regression and density estimation tasks served as sanity checks. Because the posterior was known explicitly, we could quantitatively test how well each method fit the posterior. For density estimation, GPVI outperforms amortized approaches at matching the target covariance (Table 6.1). In Bayesian linear regression, while the approximation was close for all methods, there was a clear hierarchy in terms of which types of methods produced the tightest approximation (Table 6.2). GPVI and RKHS-based ParVI achieved the best posterior fit overall, and we see in further experiments that quality of fit here is indicative of performance in more difficult tasks.

The four-class classification problem (Fig. 6.4), while seemingly simple, is particularly difficult for most methods we evaluated since many methods tend to overgeneralize to the corners. BBB underestimates the uncertainty as expected [Yao et al., 2019, Minka, 2001, Bishop, 2006]. Notably, amortized SVGD also dramatically underestimates the uncertainty, with a predictive distribution resembling that of a standard ensemble. We believe this is due to the compounding approximation error explained in Section 6.2.4, i.e., naively back-propagating the Stein variational gradient to update the generator is more likely to end up with a non-descent direction compared with back-propagating the exact functional gradient as GPVI. As shown in Fig. 6.4, the posterior approximation of GPVI is tighter, with uncertainty that better matches the data distribution. SVGD performs the best in this task, with high uncertainty everywhere except in regions near observed data. Surprisingly, GPVI and SVGD outperform HMC here, with clearly higher uncertainty near the corners of the sample space. Note that with the functional approximation by neural networks, it is hard to determine if the true posterior exactly matches the intuitively "ideal" uncertainty plot where low variance only shows up around each mixture component. On the other hand, while HMC is guaranteed to converge to the true posterior over time [Durmus et al., 2017], with a fixed computational budget, it is possible that GPVI or SVGD could achieve better performance. GPVI also has the extra benefit of the ability to draw additional samples, which is not possible with ParVI or HMC.

Our experiments in the open-category setting reveal that GPVI consistently has a higher

AUC than other scalable sampling approaches. On the MNIST dataset, GPVI is among the overall top performers, together with two ParVI methods SVGD and GFSF. On CIFAR-10, GPVI is among the top two performers in ECE and performs only slightly behind the two best ParVI methods in AUC. Most importantly, GPVI outperforms with clear margins all amortized ParVI methods on both datasets under all metrics, except for the ECE of MNIST, where both methods achieve very close top scores. This is consistent with all other qualitative and quantitative experiments we have conducted, which again shows the advantage of GPVI over existing generative ParVI methods.

We found that KSD completely failed in CIFAR-10, achieving slightly better than random accuracy. We believe this is due to the way that KSD performs the particle update. Where SVGD has a closed form expression for the particle transportation map, KSD parameterizes it with a critic network that has the input-output dimensionality of the particle parameters. Training this critic naturally becomes difficult when applying to neural network functions.

Finally, we showed that GPVI is even amenable to learning from a surrogate likelihood function. We showed that by integrating an adversarial discriminator with GPVI, we were able to outperform GANs for fitting mixtures of Gaussians. Surprisingly, we found that GPVI greatly improved over standard GAN, both in terms of the mean/STD error metrics as well as a qualitative evaluation.

To summarize the evidence from all experiments, GPVI and RKHS-based ParVI are accurate and versatile approaches to Bayesian deep learning. They consistently outperformed other approaches in all experiments, sometimes by significant margins, such as in the four-class classification setting (Fig. 6.4). Comparing with ParVI methods such as SVGD and GFSF, GPVI has the additional advantage of being able to sample new particles.

6.4 Conclusion

We have presented a new method that fuses the best aspects of parametric VI with non-parametric ParVI. GPVI has asymptotic convergence on par with ParVI. Additionally, GPVI can efficiently draw samples from the posterior. We also presented a

method for efficiently estimating the inverse of the jacobian of a deep network. Our experiments showed that GPVI performs on par with ParVI, and outperforms amortized ParVI and other competing methods in Bayesian linear regression, a classification task, open-category tasks on MNIST and CIFAR-10, and learning to sample from data. In the future we want to explore the efficacy of our method applied to large scale tasks like image generation.

6.5 Hyperparameter Settings for GPVI

In tables 6.7 - 6.12 we detail the hyperparameters chosen for each method in each experimental setting. We refer to the sampler network as $\boldsymbol{g}$, and the predicting classifier as F. We use the same structure of $\boldsymbol{g}$ and input noise for GPVI and amortized ParVI.

Density Estimation (energy potentials)	
Hyperparameter	Value
Common	
Posterior Samples	20000
Learning Rate	$1e-4$
$\boldsymbol{g}$ Hidden Layers	2
$\boldsymbol{g}$ Hidden Width	$[500, 500]$
$\boldsymbol{g}$ Input Noise Stdev	$\sigma \in \{1.0, 2.0, 6.0\}$
Training Steps	$200e3$
Minibatch Size	100
GPVI	
Optimizer	Adam
$\boldsymbol{h}_\psi$ Hidden Width	500
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
Normalizing Flow	
Optimizer	RMSProp
Flow Length	32
Weight Decay	$1e-3$

Base Dist Stdev	$\sigma \in \{1.0, 2.0, 6.0\}$
Flow Architecture	Planar Flow

Table 6.7: Hyperparameters for density estimation of energy potentials.

Bayesian Linear Regression	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	100
Learning Rate	$1e-3$
$\boldsymbol{g}$ Hidden Layers	None
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_3)$
F Hidden Layers	None
Training Steps	$50e3$
Minibatch Size	10
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	10
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	1
Critic Hidden Width	100
Critic Learning Rate	$1e-3$
Critic L_2 Weight	10
BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0

σ_2	0

Table 6.8: Hyperparameters for Bayesian linear regression task

4/2-class Classification	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	100
Learning Rate	$1e-3$
Training Steps	500e3
Minibatch Size	100
$\boldsymbol{g}$ Hidden Layers	2
$\boldsymbol{g}$ Hidden Width	64
$\boldsymbol{g}$ Nonlinearity	ReLU
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{64})$
F Hidden Layers	2
F Hidden Width	10
F Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	544
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	100
Critic Learning Rate	$1e-3$
Critic L_2 Weight	10
BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5

σ_1	1.0
σ_2	$\exp(-6)$

Table 6.9: Hyperparameters for 2/4 class classification task

Open-Category (MNIST)	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	100
Minibatch Size	50
$\boldsymbol{g}$ Learning Rate	$1e-5$
$\boldsymbol{g}$ Hidden Layers	3
$\boldsymbol{g}$ Hidden Width	$[256, 512, 1024]$
$\boldsymbol{g}$ Nonlinearity	ReLU
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{256})$
F Learning Rate	$1e-5$
F Architecture	LeNet-5
F Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	512
Critic Learning Rate	$1e-4$
Critic L_2 Weight	1.0
BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture

Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 6.10: Hyperparameters for MNIST open category task

Open-Category (CIFAR-10)	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	200
Minibatch Size	50
$\boldsymbol{g}$ Learning Rate	$1e-5$
$\boldsymbol{g}$ Hidden Layers	3
$\boldsymbol{g}$ Hidden Width	$[400, 600, 1000]$
$\boldsymbol{g}$ Nonlinearity	ReLU
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{400})$
F Hidden Layers	3 conv, 2 linear
F Hidden Width	$[32, 64, 64, 128, 10]$
F Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	512
Critic Learning Rate	$1e-4$
Critic L_2 Weight	1.0

BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 6.11: Hyperparameters for CIFAR-10 open category task

Direct Generative Modeling	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	64
Learning Rate	$1e-4$
$\boldsymbol{g}$ Hidden Layers	4
$\boldsymbol{g}$ Hidden Width	$[256, 256, 256, 256]$
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_2)$
$\boldsymbol{g}$ Nonlinearity	ReLU
Training Steps	$10e3$
Discriminator Hidden Layers	4
Discriminator Hidden Width	$[256, 256, 256, 256]$
Discriminator Training Iters	5
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	5
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$

Table 6.12: Hyperparameters for Bayesian linear regression task

Chapter 7: Improving GPVI

7.1 Introduction

In this chapter we present an improvement to the efficiency of our previous method GPVI. The benefit of GPVI is that it approximately samples from a posterior distribution, while achieving competitive performance with full ParVI methods. However, GPVI has an efficiency bottleneck in the computation of the functional gradient, which involves the evaluation of the inverse of the Jacobian of the generator. This Jacobian matrix is quadratic in dimensionality as the distribution to be generated; GPVI approximates the inverse of this Jacobian through an auxiliary network, An auxiliary MLP trained to estimate this inverse poses two difficulties. First, inverse estimators are well-known to be sensitive to numerical error, becoming more unstable as the matrix dimension increases. Second, the parameter count of the auxiliary network required to compute the inverse of the Jacobian is on par with the generator itself. This potential instability as well as the space-inefficiency are barriers to the efficient, accurate inference that we seek.

We propose to improve the approximation to the Jacobian inverse, as well as the amount of computational resources required to compute the estimate. Our approach allows for more efficient inference by reducing the number of parameters in the auxiliary network by at least 50% on small problems, and over 87% on larger Bayesian neural network tasks. We achieve this efficiency by utilizing a novel block-matrix inversion theorem that solves for most of the blocks in closed form. We are just left with a square matrix of the same dimension as the input of the generator (dimensionality of the latent space) to invert with the auxiliary network. We show that the resulting minimization problem is more stable as well as cheaper to train. In addition, we show that our method results in better performance for the open-category prediction tasks performed in [Ratzlaff et al., 2021].

We also apply our methods in the direct generative modeling setting, where we only have access to posterior samples, and there is no tractable expression for the density. Here,

we use an adversarial discriminator to approximate a lower bound of the likelihood, and show improvement over standard GANs. The original GAN loss in particular is known to suffer from both mode collapse, and low diversity within the sampled modes. We show through experiments on the stacked MNIST experiment that GPVI+ improves mode-collapse, and fits the target distribution better than GANs.

The contributions of this work are two-fold:

- We propose GPVI+, a generative ParVI algorithm that improves the efficiency and stability of GPVI. We derive a new form for the inverse of an input-output Jacobian matrix, and propose an algorithm to approximate the solution with a network.
- We demonstrate improved uncertainty estimation capabilities for BNNs, as well as improvements to mode-collapse and inter-mode diversity over GANs when applied to image generation.

7.2 Improving Generative Particle-based Variational Inference

In generative particle-based variational inference, our main goal is to learn a generator function $\boldsymbol{f} : Z \to \mathcal{X}$ that maps from a low dimensional noise distribution $Z \in \mathbb{R}^d$, to data space $\mathcal{X} \in \mathbb{R}^m$. We parameterize $\boldsymbol{f}$ as a neural network with parameters $\boldsymbol{\eta}$, and consider the distribution $q(\boldsymbol{x})$ as the implicit distribution of samples generated by $\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{z})$. Z can be any uninformative noise distribution, but in practice we sample $\boldsymbol{z} \sim Z = \mathcal{N}(0, \mathbf{I}_d)$. As this is variational inference, we want to minimize $\mathrm{KL}(q(\boldsymbol{x})||p(\boldsymbol{x}))$, where $p(\boldsymbol{x})$ is the target distribution. In this work, $p(\boldsymbol{x})$ may be specified by a likelihood function for BNN inference, or else by samples for image generation.

7.2.1 GPVI

With GPVI [Ratzlaff et al., 2021] we derived the following functional gradient of the KL objective with respect to $\boldsymbol{f}$.

$$\mathcal{J}(\boldsymbol{f}) = \mathbf{E}_{\boldsymbol{z}} \left[-\log p(\boldsymbol{f}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|} \right]. \tag{7.1}$$

This is the same gradient given in Liu and Wang [2016a], where the generator function in SVGD can be thought of as the identity function. If we instead take $\boldsymbol{f}$ to be a parameterized function drawn from a vector-valued RKHS, then the functional gradient $\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})$ with respect to $\boldsymbol{f}$ is as follows:

$$\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'} \left[-\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) \Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}', \boldsymbol{z}) - \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}) \right]. \tag{7.2}$$

Where $k(\boldsymbol{z}'|\cdot)$ is the kernel induced by the RKHS, in practice this is an RBF kernel with data-dependant bandwidth [Liu and Wang, 2016a]. Similar to amortised SVGD [Wang and Liu, 2016], the GPVI update is passed to the generator parameters $\boldsymbol{\eta}$ via backpropagation:

$$\nabla_{\boldsymbol{\eta}} \mathcal{J} = \mathbf{E}_{\boldsymbol{z}} \left[\frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{\eta}} \nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) \right]. \tag{7.3}$$

7.2.2 A Simplification of the Inverse

Inverting the Jacobian of $\boldsymbol{f}$ is poses two distinct challenges. First, it can be difficult to compute the full Jacobian for arbitrary neural networks. The Vector-Jacobian Product (VJP) computed by the (reverse mode) backward pass reveals only one column of the Jacobian at a time, requiring m backward passes to obtain the full Jacobian. The problem is compounded by the difficulty of computing an inverse of this $m \times m$ matrix, where m may be in the millions.

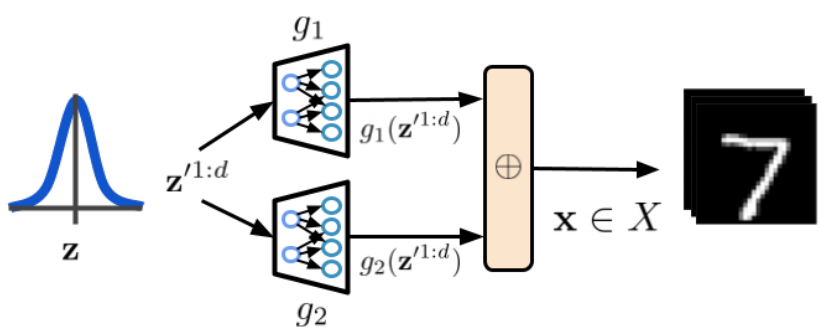
Still, network architectures in practice have non-square Jacobians, and thus have no defined inverse. To avoid limiting the generator architecture to bijections, we consider

the following parameterization of $\boldsymbol{f}$,

$$\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{g}\left(\boldsymbol{z}^{1:d}\right) + \lambda \boldsymbol{z}^{1:m}, \quad \forall \boldsymbol{z} \in \mathbb{R}^m. \tag{7.4}$$

Where $\boldsymbol{z}$ is a vector of components $\left(z^1, \ldots z^d, \ldots z^m\right)$, and $\boldsymbol{z}^{1:d}, \boldsymbol{z}^{1:m}$ are vectors that consist of the first d components, and the full $\boldsymbol{z}$ sample respectively. We show in figure 7.1 how this parameterization is used to generate samples from GPVI. The (non-square) generator function is defined as $\boldsymbol{g} : \mathbb{R}^d \to \mathbb{R}^m$. We can write the Jacobian of $\boldsymbol{f}$ as

$$J_{\boldsymbol{f}}(\boldsymbol{z}) = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right] = \left[\left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}^{1:d}}\right] \middle| \mathbf{0}^{m-d}\right] + \lambda \boldsymbol{I}^m. \tag{7.5}$$



(a) GPVI+ Parameter Generation

Figure 7.1: GPVI+ parameter generation pipeline, by concatenation $\oplus$ of the output of $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$.

To help introduce our method we split $\boldsymbol{g}$ into two functions $\boldsymbol{g}_1 : \mathbb{R}^d \to \mathbb{R}^d$ and $\boldsymbol{g}_2 : \mathbb{R}^d \to \mathbb{R}^{m-d}$, where $\boldsymbol{g}_1, \boldsymbol{g}_2$ may share parameters, or be separate functions entirely. We define the following Jacobian matrices of $\boldsymbol{g}_1, \boldsymbol{g}_2$ with respect to input $\boldsymbol{z}^{1:d}$.

$$J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) = \frac{\partial \boldsymbol{g}_1}{\partial \boldsymbol{z}^{1:d}}, \quad J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}) = \frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{z}^{1:d}}, \tag{7.6}$$

To approximate the inverse $J_{\boldsymbol{f}}(\boldsymbol{z}')^{-1}$, GPVI introduces an auxiliary network $\boldsymbol{h}_{\boldsymbol{\psi}} : \mathbb{R}^m \to \mathbb{R}^m$, that takes inputs $(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}))$, and is trained to output $J_{\boldsymbol{f}}(\boldsymbol{z}')^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$. The auxiliary network is trained jointly with $\boldsymbol{f}$ to minimize the following loss,

$$\min_{\boldsymbol{\psi}} L(\boldsymbol{h}_{\boldsymbol{\psi}}), \qquad L(\boldsymbol{h}_{\boldsymbol{\psi}}) = \| J_{\boldsymbol{f}_{\boldsymbol{\eta}}}(\boldsymbol{z}') \boldsymbol{h}_{\boldsymbol{\psi}}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})) - \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}) \|^2. \tag{7.7}$$

This method is more efficient than computing the full Jacobian via auto-differentiation – then computing its inverse with a linear solver like Conjugate Gradient. But it may still be inefficient for large problems. The main issue lies in the size of $\boldsymbol{h}_{\boldsymbol{\psi}}$, which may have as many parameters as $\boldsymbol{f}_{\boldsymbol{\eta}}$. For BNN inference problems i.e. m is large, training two $\boldsymbol{f}_{\boldsymbol{\eta}}$-sized networks may not be feasible. Additionally, neural network Jacobians are prone to poor conditioning, increasing the sensitivity of the inverse to numerical error. Its therefore advantageous for speed and stability to reduce the complexity of the inverse approximation.

We propose to represent the inverse as a block partitioned matrix, and solve for submatrices in closed form. At the end we are left with a single block of size $d \times d$ to invert with the (much smaller) auxiliary network.

The following theorem gives our main result for computing the inverse of the Jacobian matrix.

Theorem 7.2.1. *Let $\boldsymbol{f}$ be a function $\mathbb{R}^m \to \mathbb{R}^m$, that is parameterized as in Eq.(7.4). Furthermore, let the Jacobian of $\boldsymbol{f}$ w.r.t. input $\boldsymbol{z} \in \mathbb{R}^d$ be given as Eq (7.5), such that*

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \begin{bmatrix} J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}), & \mathbf{0} \\ J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}), & \mathbf{0} \end{bmatrix} + \lambda \mathbf{I} \tag{7.8}$$

Then the inverse of the Jacobian of $\boldsymbol{f}$, evaluated at $\boldsymbol{z}$, is given by the following block-partitioned matrix,

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}^{-1} = \begin{bmatrix} (J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}, & \mathbf{0} \\ -\frac{1}{\lambda} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}, & (1/\lambda)\mathbf{I} \end{bmatrix} \tag{7.9}$$

Proof

We can express the inverse of Eq(7.8) as a block partitioned matrix:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}^{-1} = \begin{bmatrix} C, & E \\ D, & F \end{bmatrix}$$

Once in the block-partitioned form, we know that $(\partial \boldsymbol{f}/\partial \boldsymbol{z})^{-1}(\partial \boldsymbol{f}/\partial \boldsymbol{z}) = \mathbf{I}$, so we can solve for the form of block matrix that yields $\mathbf{I}$,

$$\mathbf{I} = \begin{bmatrix} C, & E \\ D, & F \end{bmatrix} \left(\begin{bmatrix} J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}), & \mathbf{0} \\ J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}), & \mathbf{0} \end{bmatrix} + \lambda \mathbf{I} \right)$$

We can now solve for the submatrices C, D, E, F, and are only left to invert $J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I}$. To see this, note that $E = \mathbf{0}, F = (1/\lambda)\mathbf{I}$, and so the following is true

$$\begin{aligned} \mathbf{I} &= \begin{bmatrix} CJ_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}), & \mathbf{0} \\ DJ_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \frac{1}{\lambda} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}), & \mathbf{0} \end{bmatrix} + \lambda \begin{bmatrix} C, & \mathbf{0} \\ D, & \frac{1}{\lambda}\boldsymbol{I} \end{bmatrix} \\ &= \begin{bmatrix} CJ_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda C, & \mathbf{0} \\ DJ_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \frac{1}{\lambda} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}) + \lambda D, & \mathbf{I} \end{bmatrix} \\ &\rightarrow DJ_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \frac{1}{\lambda} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d}) + \lambda D = 0 \end{aligned}$$

Of the four submatrices necessary to compute the inverse, we have $E = \mathbf{0}, F = (1/\lambda)\mathbf{I}, C = (J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}$, and $D = \frac{1}{\lambda}(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})$. Hence, it is only necessary that we invert $(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})$, where $J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d})$ represents the first d components of $(\partial \boldsymbol{g}/\partial \boldsymbol{z})$ from Eq (7.5).

7.2.3 Practical Computation of the Inverse

While theorem 7.2.1 enables a more efficient computation of the inverse, there are practical challenges in the way of an algorithm. To compute the functional gradient from

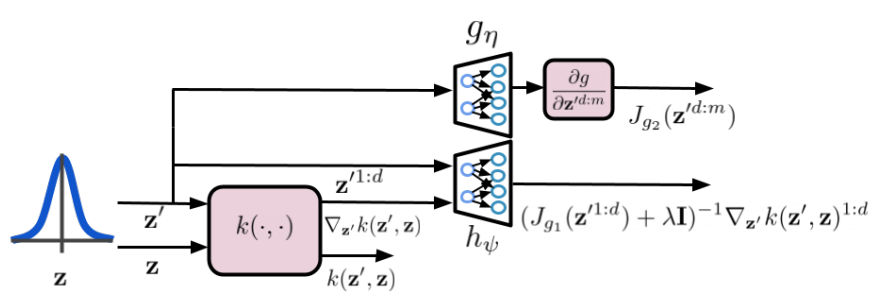
Eq (7.2), our main challenge is the evaluation of $J_{\boldsymbol{f}}(\boldsymbol{z}')^{-1}\nabla k$. For convenience of notation, we note that the kernel gradient ∇k is a m-dimensional vector

$$\nabla k = \left(\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})^1, \ldots \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})^d, \ldots \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})^m\right),$$

where $\nabla k^{1:d}, \nabla k^{d+1:m}$ denote the first d components, and last $m-d$ components of ∇k respectively.

$$\begin{aligned} J_{\boldsymbol{f}}(\boldsymbol{z})^{-1}\,\nabla k &= \begin{bmatrix} (J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}, & \mathbf{0} \\ -\frac{1}{\lambda} J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}, & \frac{\mathbf{I}}{\lambda} \end{bmatrix} \begin{bmatrix} \nabla k^{1:d} \\ \nabla k^{d+1:m} \end{bmatrix} \\ &= \begin{bmatrix} (J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1} \nabla k^{1:d} \\ -\lambda J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1} \nabla k^{1:d} + \frac{1}{\lambda} \nabla k^{d+1:m} \end{bmatrix}, \end{aligned} \tag{7.10}$$

Our two challenges are to compute the top and bottom terms of Eq (7.10). Computing the top term (TOP) is the first priority, as it is also involved in evaluating the bottom term (BOTTOM). We can solve for TOP by making use of the auxiliary network $\boldsymbol{h}_{\boldsymbol{\psi}}$. In this case, $\boldsymbol{h}_{\boldsymbol{\psi}}$ takes as input $\left(\boldsymbol{z}'^{1:d}, \nabla k^{1:d}\right)$, and predicts $(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1} \nabla k^{1:d}$. To train $\boldsymbol{h}_{\boldsymbol{\psi}}$, we use the same loss as in Eq (7.7). Note that $\boldsymbol{h}_{\boldsymbol{\psi}}$ now has a small memory cost, as the target VJP is a vector of size d.



(a) Computation of $J_{\boldsymbol{g}_1}$ and $J_{\boldsymbol{g}_2}$

Figure 7.2: How we compute the partial Jacobian matrices $J_{\boldsymbol{g}_1}$ and $J_{\boldsymbol{g}_2}$ for computation of the inverse defined in theorem (7.2.1).

The second challenge lies in the bottom term of the expression, particularly in how to compute $J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda\mathbf{I})^{-1}\nabla k^{1:d}$. In figure 7.2 we show how the terms of Eq.(7.10) are computed. First, we know that $(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda\mathbf{I})^{-1}\nabla k^{1:d}$ is a vector, and that $J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})$ represents the last $m - d$ dimensions of the Jacobian of $\boldsymbol{g}$. We can then compute the Jacobian-Vector product (JVP):

$$J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})h_\psi(\boldsymbol{z}'^{1:d}, \nabla k^{1:d}).$$

BOTTOM is fully computed with the addition of the last $m-d$ components of $\frac{1}{\lambda}\nabla k$. We emphasize that the full Jacobian is never computed, nor do we invert a matrix larger than d dimensions. An algorithm summarizing our approach, with practical considerations, is shown in Algorithm 3.

7.3 Experiments

We show the efficacy of our approach in two general domains: where the target $p(\boldsymbol{x})$ is defined through a likelihood function, and learning purely from samples. We initially validate our approach through a toy example: fitting an analytically known posterior distribution. We also evaluate our approach for Bayesian neural networks through the same open-category prediction tasks given in Ratzlaff et al. [2021]. For both tasks we show an improvement over GPVI in parameter efficiency, and in most cases a closer approximation of the ParVI solution [Liu and Wang, 2016a, Liu et al., 2019]. Second, we once again follow Ratzlaff et al. [2021], and show that GPVI+ is capable of learning from samples, where the likelihood function is approximated with an adversarial discriminator. Here, we show that GPVI+ can fit distributions of samples without suffering from the various pathologies of implicit generative models such as GANs e.g. mode-collapse. With experiments on stacked MNIST, we provide empirical evidence that GPVI+ samples can fit more modes than DCGANs.

For all tasks, we hold network architectures, learning rates, and other common hyperparameters consistent across methods. An overview of our architectures, hyperparameters, and experiment-specific settings can be found in the following section.

Algorithm 3 GPVI+ Training Algorithm

Init generator $\boldsymbol{f_\eta}$, auxiliary net $\boldsymbol{h_\psi}$, learning rate α, noise dim d, and output dim m

Function `Update_Aux(`$\boldsymbol{h_\psi}$, $\boldsymbol{z'}$, $\boldsymbol{v}_d$`):`

1. Forward $\boldsymbol{h_\psi}(\boldsymbol{z'}, \boldsymbol{v}_k)$ to get $(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}\boldsymbol{v}_d$
2. Backward to get VJP $\boldsymbol{h}_{\boldsymbol{\psi}}^\top J_{\boldsymbol{f_\eta}}(\boldsymbol{z'})$
3. Update $\boldsymbol{h_\psi}$ by $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \alpha \nabla_{\boldsymbol{\psi}} L(\boldsymbol{h_\psi})$

   `/*` $\nabla_{\boldsymbol{\psi}} L$ `is computed by backprop of` (7.7) `*/`

Function `Jac_Inverse(`$\boldsymbol{h_\psi}$, $\boldsymbol{x'}$, $\boldsymbol{z'}$, $\boldsymbol{v}_d$, $\boldsymbol{v}_m$`):`

1. Compute TOP: $(J_{\boldsymbol{g}_1}(\boldsymbol{z}^{1:d}) + \lambda \mathbf{I})^{-1}\boldsymbol{v}_d) = \boldsymbol{h_\psi}(\boldsymbol{z'}, \boldsymbol{v}_d)$
2. Compute BOTTOM: $J_{\boldsymbol{g}_2}(\boldsymbol{z}^{1:d})$ TOP

   `/* This JVP can be done cheaply in JAX, or by a double-backward in PyTorch */`
3. BOTTOM $= -\lambda$BOTTOM$+(1/\lambda)\boldsymbol{v}_m$
4. $J_{\boldsymbol{f}}(\boldsymbol{z'})^{-1}\nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z}) =$ [TOP, BOTTOM]
5. return $J_{\boldsymbol{f}}(\boldsymbol{z'})^{-1}\nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z})$

while Not Done **do**

1. Sample input noise $\boldsymbol{z}, \boldsymbol{z'} \sim N(\mathbf{0}, I_d)$
2. $\boldsymbol{x} = \boldsymbol{f_\eta}(\boldsymbol{z})$
3. $\boldsymbol{x'} = \boldsymbol{f_\eta}(\boldsymbol{z'})$
4. Compute $k(\boldsymbol{z'}, \boldsymbol{z})$ and $\nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z})$
5. $\boldsymbol{v}_d = \nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z})$[1:d]
6. $\boldsymbol{v}_m = \nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z})$[d+1:m]
7. Update_Aux($\boldsymbol{h_\psi}$, $\boldsymbol{x'}$, $\boldsymbol{z'}$, $\boldsymbol{v}_d$)
8. $J_{\boldsymbol{f}}(\boldsymbol{z'})^{-1}\nabla_{\boldsymbol{z'}} k(\boldsymbol{z'}, \boldsymbol{z}) =$ Jac_Inverse($\boldsymbol{h_\psi}$, $\boldsymbol{z'}$, $\boldsymbol{v}_d$, $\boldsymbol{v}_m$)
9. Compute functional gradient $\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})$ by (7.2)
10. Update $\boldsymbol{f_\eta}$ by $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} - \alpha \nabla_{\boldsymbol{\eta}} \mathcal{J}$,

    `/* where` $\nabla_{\boldsymbol{\eta}} \mathcal{J}$ `is computed by` (7.3) `*/`

end

7.3.1 5D Density Estimation

We first evaluate how each generative approach can fit a 0-mean 5-D target Normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$. For each target we randomly initialize a diagonal covariance Σ. We parameterize $\boldsymbol{f}$ as a linear function with parameters $\boldsymbol{\eta}$, where the output distribution of $\boldsymbol{f}$ is given as $\boldsymbol{\eta}\boldsymbol{\eta}^T$. In Table 7.1, we measure the fit of the approximate posterior as the average per-dim error in the estimated Σ, as well as the wall clock time-per-iteration. We compare GPVI+ with GPVI, and a variant of our method that uses the exact inverse of the Jacobian instead of the auxiliary network ("GPVI-Exact"). For all methods, we use input noise $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, I_2)$. We perform each experiment 5 times, and report the mean $\pm$ one standard deviation. We find that GPVI+ performs as expected, learning a closer approximation to the target than GPVI, while taking only slightly more time computationally. In the following experiments, we will see that this extra time is worth the marginal cost, considering the efficiency and performance gain of GPVI+. We note that GPVI-Exact is far too slow to be practical, given that it must explicitly compute the full Jacobian of $\boldsymbol{f}$.

Method	Σ error $\downarrow$	Time (ms)
GPVI Exact	$0.981 \pm .01$	$194.98 \pm .19$
GPVI	$0.954 \pm .05$	$\mathbf{15.67 \pm .06}$
GPVI+	$\mathbf{0.727 \pm .18}$	$18.99 \pm .03$

Table 7.1: Comparison of different inversion methods for fitting a 5D diagonal Gaussian distribution.

7.3.2 Open-Category Prediction

To evaluate the scalability of our approach we turn to large-scale image classification experiments. We utilize the open-category task to test if our uncertainty estimations can help detect outlier examples. The open category task defines a set of inlier classes that are seen during training, and a set of outlier classes only used for evaluation. We evaluated on the MNIST and CIFAR-10 image datasets, following Neal et al. [2018] to split each dataset into 6 inlier classes and 4 outlier classes. We performed standard fully supervised training on the 6 inlier classes, and measured uncertainty in the 4 unseen outlier classes. We evaluated the uncertainty of our models with the AUC and ECE metrics respectively. Where AUC measures the ability of a model to discriminate between inlier and outlier

classes, and ECE measures the calibration error. For the AUC metric, we report the AUC using both the entropy, and the variance of the posterior predictive distribution, as these statistics often favor one method over another.

In light of the computational difficulties that arise when evaluating BNNs on high dimensional datasets, its natural to ask whether we can achieve similar performance at a lower computational cost. One way to do this is to learn a distribution over linear functions that maps a featurized representation to output predictions. In practice, this amounts to jointly training a single deterministic "body" network with standard SGD, and the output linear layer with generative ParVI. We evaluate this method in two settings, by training the last layer with GPVI, and also GPVI+. We evaluate this method, that we call LinGPVI and LinGPVI+ respectively, on the open-category prediction task. We expect that the respective methods will be computationally cheaper than either GPVI and GPVI+, and enable the use of deeper networks for prediction. The possible disadvantage of this strategy is that we may lose critical diversity in the feature representation. Toward recovering some diversity in the feature representation while retaining a computationally cheaper method, we also evaluate a method we call EnsembleGPVI+. In this method, instead of representing the "body" of the network with a single deterministic network as in LinGPVI or LinGPVI+, we use a deep ensemble of randomly initialized networks. The output of the deep ensemble is used as input to the final classification layer that we train with GPVI+. LinGPVI, LinGPVI+ and EnsembleGPVI+ are computationally lighter than GPVI or full GPVI+ in both parameter count and time complexity.

MNIST consists of 70,000 grayscale images of handwritten digits at 28x28 resolution We partition the dataset by only using the first six classes for training and testing. The remaining four classes are used to compute the AUC and ECE statistics. We chose the LeNet-5 classifier architecture for all models, and trained for 100 epochs. For all methods, we draw 10 posterior samples for both training and evaluation. Table 7.2 shows the results of open-category prediction for each method. Our results show that once again, full ParVI methods like SVGD and GFSF are the best performing methods for open-category prediction. Notably, GPVI outperforms GPVI+ here with regard to AUC (entropy), though the gap between them is relatively small. Where GPVI+ excels is in space-efficiency, where we need just 1M parameters for auxiliary network $h_{\boldsymbol{\psi}}$, compared to the 48M parameters used by GPVI. On MNIST, BBB, a mean-field VI

Method	Clean	AUC (Ent)	AUC (Var)	ECE	$\boldsymbol{h_\eta}$ weights
SVGD	99.3	**.987 ± .001**	**.986 ± .003**	**.002 ± .0005**	N/A
GFSF	99.2	**.986 ± .003**	.985 ± .002	**.002 ± .0003**	N/A
BBB	98.6	.951 ± .008	.965 ± .004	.014 ± .0027	N/A
GPVI	99.3	**.986 ± .002**	.980 ± .003	**.002 ± .0005**	48M
LinGPVI+	99.3	.980 ± .004	.977 ± .006	**.002 ± .0002**	**.2M**
EnsembleGPVI+	99.3	.979 ± .007	.984 ± .007	**.002 ± .0004**	**.2M**
GPVI+	99.3	.980 ± .001	.946 ± .009	**.002 ± .0008**	1M

Table 7.2: Results for open-category classification on MNIST. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers. We also compare parameter count of GPVI and GPVI+ auxiliary networks, as well as LinGPVI+ and EnsembleGPVI+.

method has lacking performance under both AUC and ECE statistics. LinGPVI+ and EnsembleGPVI+ are efficient, requiring just 0.2M parameters for the auxiliary network. The AUC of LinGPVI+ performs worse than its counterparts, while EnsembleGPVI+ has competitive performance with regard to AUC (variance), even outperforming GPVI and GPVI+ under this metric.

CIFAR-10 consists of 60,000 RGB images depicting 10 object classes at 32x32 resolution. We adopted the same 6 inlier / 4 outlier split used in the previous MNIST experiment for the open-category setting. We also used the same hypernetwork architecture as in the MNIST setup, drawing 10 samples from each method's approximate posterior for training and evaluation. We report the clean accuracy, AUC (entropy/variance), and ECE statistics in table 7.3. Our CIFAR-10 results show that GPVI+ outperforms GPVI in AUC (entropy), and even outperforms the full ParVI method GFSF. However, GPVI+ is slightly less calibrated than GPVI, while still being better calibrated than GFSF. Once again, the space-efficiency of GPVI+ shines, where we save 69M parameters for our auxiliary network over GPVI. BBB once again has worse AUC performance, but is noticeably better calibrated compared to its performance on MNIST. Again, LinGPVI+ does not fare so well on any metric except $\boldsymbol{h_\eta}$ parameter count, although it does perform slightly

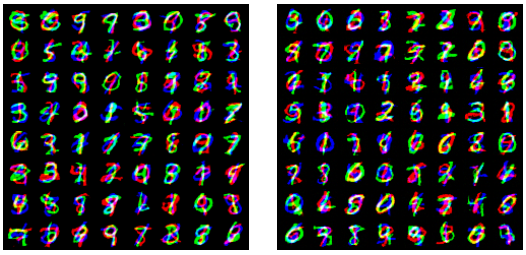
Method	Clean	AUC (Ent)	AUC (Var)	ECE	$\boldsymbol{h_\eta}$ weights
SVGD	80.3	**.688 ± .008**	**.679 ± .007**	.042 ± .004	N/A
GFSF	80.6	.681 ± .004	.657 ± .014	.068 ± .011	N/A
BBB	70.0	.649 ± .006	.599 ± .014	**.016 ± .002**	N/A
GPVI	76.2	.677 ± .008	.629 ± .007	**.018 ± .015**	70M
LinGPVI	72.5	.603 ± .006	.608 ± .040	.136 ± .109	.25M
LinGPVI+	75.0	.621 ± .004	.608 ± .029	.143 ± .122	**.2M**
EnsembleGPVI+	80.7	.615 ± .002	.675 ± .013	.154 ± .191	**.2M**
GPVI+	76.4	**.686 ± .007**	.601 ± .014	.055 ± .025	1M

Table 7.3: Results for open-category classification on CIFAR-10. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers. We also compare parameter count of GPVI and GPVI+ auxiliary networks, as well as LinGPVI+ and EnsembleGPVI+.

better than LinGPVI. EnsembleGPVI+ again performs competitively with SVGD in AUC (variance), making it a viable option for efficient generative ParVI. Its worth noting that LinGPVI, LinGPVI+ and EnsembleGPVI+ are not as well calibrated as the other ParVI methods.

7.3.3 Stacked MNIST

We also train on real image data, to evaluate how well GPVI+ can fit extremely multimodal data. Stacked MNIST consists of 128k RGB images of handwritten digits. Each channel of a stacked MNIST image is random MNIST digit $(0 - 9)$. Hence, stacked MNIST has 1000 modes, where each mode can be identified by querying a pretrained MNIST classifier on each channel. The goal is to train a generative model that not only captures all the modes, but has relatively equal probability of generating any one mode. In figure 7.3 we show the results of training GPVI+ as well as corresponding GAN models on Stacked MNIST. We can see that all GPVI+ variants learn to generate high quality images. GPVI+ also learns to sample relatively even from each mode, while DCGAN is more likely to repeat generated digits. In table 7.4 we sample $100K$ images



(a) DCGAN (b) GPVI+ (DCGAN)

Figure 7.3: Samples drawn after training on Stacked MNIST after 100 epochs.

Method	# Modes ↑	Reverse KL ↓
DCGAN	924	0.253 ± 0.010
GPVI+ (DCGAN)	1000	0.185 ± 0.044

Table 7.4: Performance on stacked MNIST. We compute the number of modes captured by each generative model, along with the reverse KL divergence between output samples and true samples

from each model, and report quantitative metrics such as number of modes fit by each method, as well as the reverse KL divergence between the generated modes and the target modes. To evaluate the number of modes, we pretrain an MNIST classifier to 99.3% accuracy, and classify each channel of the generated stacked MNIST digits. To compute the reverse KL divergence, we find the distribution of modes w.r.t the generated samples q_{modes}, and initialize the "ideal" distribution $p = [\frac{1}{1000}, \frac{1}{1000} \cdots \frac{1}{1000}] \in \mathbb{R}^{1000}$. We then compute $\text{KL}\left[\frac{q_{modes}}{\sum q_{modes}}, p\right]$. The results in table 7.4 show that GPVI with a discriminator trained with the DCGAN loss is able to outperform DCGAN in terms of both number of generated modes, and reverse KL divergence. GPVI+ is able to generate all 1000 modes.

7.4 Conclusion

We proposed GPVI+ to alleviate an inefficiency of training GPVI. GPVI requires an auxiliary network of the same size as its generator network. Accordingly, the size of this auxiliary network will scale with the dimensionality of the target data distribution. Our

method GPVI+ achieves space-efficiency over GPVI by deriving a new way to compute the inverse of the Jacobian of the generator. Our approach reduces the size of the inverse problem from the dimensionality of the generator output, to the dimensionality of its input. The savings in parameter count from our method was over 98% in our largest open-category experimental setting. In both density estimation experiments, as well as open-category prediction for CIFAR-10, we were able to improve on GPVI and even match the full ParVI method GFSF. The limitation of our method is showcased in the density estimation experiment in table 7.1, where our method is slightly slower than GPVI due to the computation of an additional Jacobian-vector product during training. Considering the experimental evidence given by the density estimation and open-category experiments, GPVI+ maintains or exceeds the performance of GPVI, while saving significant space over GPVI. That being said, we would like to explore ways to improve the efficiency of the generator network in the Bayesian neural network setting. We would also like to apply GPVI+ to domains like multi-task and continual learning.

7.5 Hyperparameter Settings for GPVI+

In tables 7.5 - 7.7 we detail the hyperparameters chosen for each method in each experimental setting. We refer to the sampler network as $\boldsymbol{g}$, and the predicting classifier as F. We use the same structure of $\boldsymbol{g}$ and input noise for GPVI and amortized ParVI. In general, we use the same hyperparameters for GPVI and GPVI+, unless otherwise indicated.

Density Estimation	
Hyperparameter	Value
Common	
Posterior Samples	20000
Learning Rate	$1e-4$
$\boldsymbol{g}$ Hidden Layers	2

$\boldsymbol{g}$ Hidden Width	$[500, 500]$
$\boldsymbol{g}$ Input Noise Stdev	$\sigma \in \{1.0, 2.0, 6.0\}$
Training Steps	$200e3$
Minibatch Size	100
GPVI	
Optimizer	Adam
$\boldsymbol{h}_\psi$ Hidden Width	500
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	1.0
GPVI+	
$\boldsymbol{h}_\psi$ Hidden Width	500
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	0.005

Table 7.5: Hyperparameters for density estimation.

Open-Category (MNIST)	
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	100
Minibatch Size	50

$\boldsymbol{g}$ Learning Rate	$1e-5$
$\boldsymbol{g}$ Hidden Layers	3
$\boldsymbol{g}$ Hidden Width	$[256, 512, 1024]$
$\boldsymbol{g}$ Nonlinearity	ReLU
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{256})$
F Learning Rate	$1e-5$
F Architecture	LeNet-5
F Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	1.0
GPVI+	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	0.01
BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 7.6: Hyperparameters for MNIST open category task

Open-Category (CIFAR-10)	
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	200
Minibatch Size	50
$\boldsymbol{g}$ Learning Rate	$1e-5$
$\boldsymbol{g}$ Hidden Layers	3
$\boldsymbol{g}$ Hidden Width	$[400, 600, 1000]$
$\boldsymbol{g}$ Nonlinearity	ReLU
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{400})$
F Hidden Layers	3 conv, 2 linear
F Hidden Width	$[32, 64, 64, 128, 10]$
F Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	1.0
GPVI+	
$\boldsymbol{h}_\psi$ Hidden Width	512

$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	0.001
BBB (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 7.7: Hyperparameters for CIFAR-10 open category task.

Stacked MNIST	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	64
Learning Rate	$1e-4$
$\boldsymbol{g}$ Conv Transpose Layers	3
$\boldsymbol{g}$ Conv Transpose Channels	$[256, 128, 64]$
$\boldsymbol{g}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{64})$
$\boldsymbol{g}$ Nonlinearity	ReLU
Training Steps	$78e3$
Discriminator Conv Layers	3
Discriminator Conv Channels	$[64, 128, 256]$

Discriminator FC Layers	1
Discriminator FC Width	4096
Discriminator Training Iters	5
GPVI+	
$\boldsymbol{h}_\psi$ Hidden Width	512
$\boldsymbol{h}_\psi$ Hidden Layers	3
$\boldsymbol{h}_\psi$ Learning Rate	$1e-4$
λ value	0.01

Table 7.8: Hyperparameters for Stacked MNIST task.

Chapter 8: Future Work and Conclusion

In this dissertation we have shown that implicit generative models can be effectively applied to enable uncertainty estimation in neural networks.

First, we proposed HyperGAN, a generative architecture used to sample from a distribution of model parameters that fit the data. We showed that such a neural sampler could be trained without prohibitive computational overhead, or limiting form of the approximating distribution. Samples from HyperGAN performed as well as their point-estimate counterparts on supervised learning tasks, in addition to providing better uncertainty estimates on out-of-distribution data. When we applied HyperGAN for detecting adversarial examples, we found that HyperGAN was hard to fool with standard white-box attacks. This was due to the fact that HyperGAN can always sample additional models, in addition to the attacked model. Second, we used the lessons learned from HyperGAN, and applied uncertainty quantification through implicit generative models to reinforcement learning. RL represents an area that naturally benefits greatly from models that leverage uncertainty. We showed that implicit distributions of neural networks are greatly useful in the model-based setting. By leveraging the epistemic uncertainty of a distribution over dynamic models to explore environments, our agents were able to explore more efficiently than point-estimate or ensemble-based approaches. In this method we leveraged particle-based varitaional inference to incorporate an explicit diversity term that was missing from HyperGAN. Our method also enables a RL agents to accrue high reward when the reward function is known exactly, by first exploring according to its uncertainty. This initial exploration period is important to collect the most useful data to reduce its own uncertainty in the environment. Third, we proposed a new method for approximate Bayesian inference: GPVI. GPVI is in some sense the culmination of the work presented here. GPVI is a generative counterpart to particle-based variational inference, that allows for acquiring additional samples, without a large performance cost. GPVI also showed improvements over amortized particle-based variational inference in all our evaluations, giving an ideal opportunity to revisit our work

in reinforcement learning. We also found that GPVI was amenable to GAN-style training, where a discriminator is used as a surrogate likelihood function. We found that GPVI with a discriminator was able to fit target distributions much more accurately than standard GANs, without suffering from mode collapse. We further improved on GPVI by reducing the dimensionality of the inversion problem posed by GPVI training. Our improved approach GPVI+ showed dramatic savings in space-efficiency over GPVI, along with improvements to open-category prediction on CIFAR-10. These results are encouraging, and we are eager to apply GPVI+ to higher dimensional image recognition and open-category prediction tasks.

The work above has laid the foundation for an ongoing research agenda. We are interested in both improving our work in uncertainty estimation, and in applying the aforementioned implicit generative models to new tasks. A clear line of improvement lies in the efficiency of Bayesian neural network inference. Currently, our methods use generative models where the number of parameters scale linearly with the output dimensionality. To apply our work to the current state of the art architectures, we must improve the scalability of our methods. We are also interested in application of GPVI(+) to model-free reinforcement learning. We believe that learning an implicit distribution over an agent's value function or policy may allow agents to efficiently explore without resorting to common exploration heuristics. Beyond the applications presented here, we are also interested in multi-task and continual learning. We believe that the implicit generative models presented here are a natural fit for learning a conditional distribution of task-based functions given some environmental input. Furthermore, continual learning represents a component of Bayesian reasoning not explored in our Bayesian neural network experiments. Recognizing and incorporating novel data for downstream predictions should be within the range of capabilities of the systems we have presented. The future of uncertainty-aware deep learning is indeed exciting.

The work presented in this dissertation was published in the following venues.

- **Ratzlaff, Neale**, and Li Fuxin. "HyperGAN: A generative model for diverse, performant neural networks." International Conference on Machine Learning. PMLR, (**ICML, 2019**).

- **Ratzlaff, Neale**, et al. "Implicit generative modeling for efficient exploration."

International Conference on Machine Learning. PMLR, (**ICML, 2020**).

- **Ratzlaff, Neale***, Bai, Qinxun* et al. "Generative Particle Variational Inference via Estimation of Functional Gradients." International Conference on Machine Learning. PMLR, (**ICML, 2021**).

* equal contribution.

Bibliography

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1, 2012.

Luca Ambrogioni, Umut Guclu, Yagmur Gucluturk, and Marcel van Gerven. Wasserstein variational gradient descent: From semi-discrete optimal transport to ensemble variational inference. *arXiv preprint arXiv:1811.02827*, 2018.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9. IEEE, 2018.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018b.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

William R Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*, 2019.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018a.

Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Discovering adversarial examples with momentum. *CoRR*, abs/1710.06081, 2017. URL `http://arxiv.org/abs/1710.06081`.

Alain Durmus, Eric Moulines, and Eero Saksman. On the convergence of hamiltonian monte carlo. *arXiv preprint arXiv:1705.00166*, 2017.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Yihao Feng, Dilin Wang, and Qiang Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

Roger Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical analysis*, pages 73–89. Springer, 1976.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing

model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *arXiv preprint arXiv:1802.10026*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, and Richard Zemel. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, pages 3732–3747. PMLR, 2020.

Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24:2348–2356, 2011.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Rein Houthooft, Xi Chen, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1109–1117. Curran Associates, Inc., 2016.

Tianyang Hu, Zixiang Chen, Hanxi Sun, Jincheng Bai, Mao Ye, and Guang Cheng. Stein neural sampler. *arXiv preprint arXiv:1810.03545*, 2018.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

David Janz, Jiri Hron, Przemysław Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: exploration and uncertainty in temporal difference learning. *Advances in Neural Information Processing Systems*, 32:4507–4516, 2019.

Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, pages 592–600, 2012.

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

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

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local

reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583, 2015.

David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017a.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017b.

Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, and Jun Zhu. Understanding and accelerating particle-based variational inference. In *International Conference on Machine Learning*, pages 4082–4092. PMLR, 2019.

Qiang Liu. Stein variational gradient descent as gradient flow. In *Advances in neural information processing systems*, pages 3115–3123, 2017.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29: 2378–2386, 2016a.

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, 2016b.

Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32:13153–13164, 2019.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv e-prints*, June 2017.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015. URL `http://arxiv.org/abs/1511.05644`.

Thomas Peter Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.

Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open set learning with counterfactual images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 613–628, 2018.

Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1994.

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

Brendan O'Donoghue. Variational bayesian reinforcement learning with regret bounds. *arXiv preprint arXiv:1807.09647*, 2018.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. *arXiv preprint arXiv:1709.05380*, 2017.

P Ortner and R Auer. Logarithmic online regret bounds for undiscounted reinforcement learning. *Advances in Neural Information Processing Systems*, 19:49, 2007.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

Ian Osband, Benjamin Van Roy, Daniel Russo, and Zheng Wen. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8617–8629, 2018.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International Conference on Machine Learning*, pages 5062–5071, 2019.

Nick Pawlowski, Andrew Brock, Matthew CH Lee, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. *arXiv preprint arXiv:1901.11058*, 2019.

Neale Ratzlaff, Qinxun Bai, Li Fuxin, and Wei Xu. Generative particle variational inference via estimation of functional gradients. *arXiv preprint arXiv:2103.01291*, 2021.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

Nabeel Seedat and Christopher Kanan. Towards calibrated and scalable uncertainty representations for neural networks. *arXiv preprint arXiv:1911.00104*, 2019.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788, 2019.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13969–13980, 2019.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. *arXiv preprint arXiv:1705.07761*, 2017.

Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006.

Malcolm Strens. A bayesian framework for reinforcement learning. In *ICML*, volume 2000, pages 943–950, 2000.

Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *International Conference on Artificial General Intelligence*, pages 41–51. Springer, 2011.

István Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *ICML*, 2010.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*, 2016.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein Auto-Encoders. *ArXiv e-prints*, November 2017.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

Dilin Wang and Qiang Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. In *International Conference on Machine Learning (ICML)*, 2018.

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

Jiayu Yao, Weiwei Pan, Soumya Ghosh, and Finale Doshi-Velez. Quality of uncertainty quantification for bayesian neural network inference. *arXiv preprint arXiv:1906.09686*, 2019.

David Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.

Rui Zhao and Volker Tresp. Curiosity-driven experience prioritization via density estimation. *arXiv preprint arXiv:1902.08039*, 2019.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.