Functional Variational Bayesian Neural Networks

Shengyang Sun*†, Guodong Zhang*†, Jiaxin Shi*‡, Roger Grosse†
†University of Toronto, †Vector Institute, ‡Tsinghua University
{ssy, gdzhang, rgrosse}@cs.toronto.edu, shijx15@mails.tsinghua.edu.cn

1 Introduction

Bayesian neural networks (BNNs) [11, 18] have the potential to combine the scalability, flexibility, and predictive performance of neural networks with principled Bayesian uncertainty modeling. However, the practical effectiveness of BNNs is limited by our ability to specify meaningful prior distributions and by the intractability of posterior inference. Choosing a meaningful prior distribution over network weights is difficult because the weights have a complicated relationship to the function computed by the network. Stochastic variational inference is appealing because the update rules resemble ordinary backprop [9, 4], but fitting accurate posterior distributions is difficult due to strong and complicated posterior dependencies [16, 31, 36, 28].

In this paper, we propose to perform variational inference directly on the distribution of functions. Specifically, we introduce functional variational BNNs (fBNNs), where a BNN is trained to produce a distribution of functions with small KL divergence to the true posterior over functions. We prove that the KL divergence between stochastic processes can be expressed as the supremum of marginal KL divergences at finite sets of points. Based on this, we present a novel objective functional ELBO (fELBO). Then we introduce a GAN-like minimax formulation and a VAE-like maximization formulation for functional variational inference. To approximate the marginal KL divergence gradients, we adopt the recently proposed spectral Stein gradient estimator (SSGE) [29].

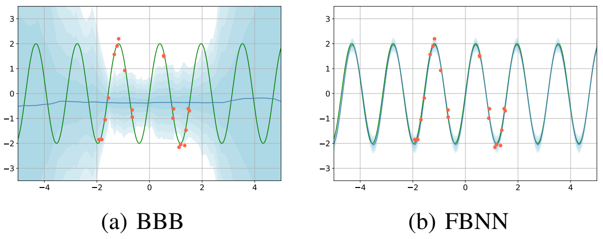


Figure 1: Fitting periodic structures. Blue lines and shaded areas correspond to predictive means and standard deviations.

2 Functional Variational Bayesian Neural Networks

2.1 Functional Evidence Lower Bound (fELBO)

Given dataset $\mathcal{D} = (\mathbf{X}^D, \mathbf{y}^D)$, variational inference in weight space is based on the equivalence between maximizing the ELBO and minimizing the KL divergence from the true posterior. Now we introduce functional variational inference, where the functional ELBO (fELBO) is defined with respect to a variational posterior over functions ($f : \mathcal{X} \to \mathcal{Y}$) rather than over weights.

$$\mathcal{L}(q) := \mathbb{E}_q[\log p(\mathbf{y}^D|f)] - \mathrm{KL}[q(f)||p(f)]. \tag{1}$$

Specifically, we assume a stochastic process prior $p(f)$ over functions. This could be a Gaussian Process, but one could also use stochastic processes without closed-form marginal densities, such as distributions over piecewise linear functions. We consider a variational posterior $q_\phi(f) \in \mathcal{Q}$ defined in terms of a neural network with stochastic weights and/or stochastic inputs. Specifically, we sample a function from q by sampling a random noise ξ and reparameterize $f(\mathbf{x}) = g_\phi(\mathbf{x}, \xi)$.

*Equal contribution.

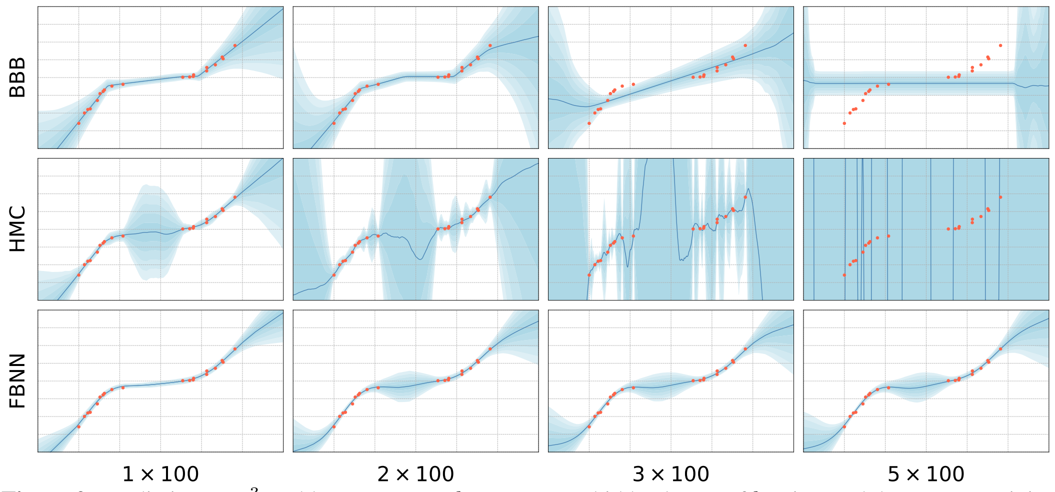


Figure 2: Prediction on x^3 problem. Here $a \times b$ represents a hidden layers of b units. Red dots are 20 training points. Blue line is the mean of final prediction and the shaded areas represents standard derivations. We compare fBNNs, BBB, and HMC [19]. One desideratum of Bayesian models is that they behave gracefully as their capacity is increased [23]. Unfortunately, BNN priors may have undesirable behavior as more units or layers are added. Larger BNNs entail more difficult posterior inference and larger description length for the posterior, causing degeneracy for large networks. Our fBNNs do not have this problem, since the prior is defined directly over the space of functions, and therefore the BNN can be made arbitrarily large without changing the functional variational inference problem.

For example, the standard Bayesian neural networks with Gaussian weight distributions allow this reparameterization trick [13]. Note that because a *single* vector ξ is shared among all input locations, it corresponds to randomness in the *function*, rather than observation variance; hence, the sampling of ξ corresponds to epistemic, rather than aleatoric, uncertainty [6].

In theorem 1, we prove that the functional KL divergence is equal to the supremum of marginal KL divergences over all finite sets of input locations $\mathbf{X} = \mathbf{x}_{1:n}$, which we term *measurement points*. Intuitively, this follows from the Kolmogorov Extension Theorem in appendix A.2 which guarantees that we only need finite marginal distributions to define the stochastic process. A full proof is given in Appendix C.

Theorem 1 (Functional KL Divergence)**.** *For two stochastic processes P, Q, the KL divergence is the supremum of marginal KL divergences over all finite subset of inputs $\mathbf{x}_{1:n}$:*

$$\mathrm{KL}[P\|Q] = \sup_{n,\ \mathbf{x}_{1:n}} \mathrm{KL}[P_{\mathbf{x}_{1:n}} \| Q_{\mathbf{x}_{1:n}}]. \tag{2}$$

fELBO Using this characterization of the functional KL divergence, we rewrite the fELBO:

$$\begin{aligned}\mathcal{L}(q) &= \mathbb{E}_q[\log p(\mathbf{y}^D|f)] - \sup_{\mathbf{X}} \mathrm{KL}[q(\mathbf{f}^{\mathbf{X}})||p(\mathbf{f}^{\mathbf{X}})] \\ &= \inf_{\mathbf{X}} \sum_{(\mathbf{x}^D, y^D)\in\mathcal{D}} \mathbb{E}_q[\log p(y^D|f(\mathbf{x}^D))] - \mathrm{KL}[q(\mathbf{f}^{\mathbf{X}})||p(\mathbf{f}^{\mathbf{X}})] \\ &:= \inf_{\mathbf{X}} \mathcal{L}_{\mathbf{X}}(q).\end{aligned} \tag{3}$$

Hence, maximizing $\mathcal{L}(q)$ can be viewed as a two-player game analogous to a GAN [8]: the generator is maximizing $\mathcal{L}_{\mathbf{X}}(q)$ with respect to q, and the discriminator is minimizing $\mathcal{L}_{\mathbf{X}}(q)$ with respect to $\mathbf{X}$.

Interestingly, we can show that if measurement points contains all training examples, $\mathcal{L}_{\mathbf{X}}(q)$ is a lower bound of the log marginal likelihood $\log p(\mathcal{D})$. A proof is in Appendix D.3.

Theorem 2 (Lower Bound)**.** *If $\mathbf{X}$ contains all training input locations $\mathbf{X}^D$, then*

$$\mathcal{L}_{\mathbf{X}}(q) = \log p(\mathcal{D}) - \mathrm{KL}[q(\mathbf{f}^{\mathbf{X}})\|p(\mathbf{f}^{\mathbf{X}}|\mathcal{D})] \leq \log p(\mathcal{D}).$$

2.2 Functional Variational Inference

Now we present practical algorithms for functional variational inference.

Adversarial Functional Variational Inference Analogously to GANs[2], $\mathcal{L}(q)$ is likely to be infinite if the discriminator is completely unrestricted, because most stochastic process priors assign measure zero to the set of functions representable by a neural network [1]. However, by limiting the capacity of the discriminator, we obtain a meaningful minimax objective which forces the generator to produce samples which resemble the true posterior. Most obviously, we can restrict the discriminator to choose measurement points of size M. Then we have the adversarial functional variational inference:

$$\max_{q\in\mathcal{Q}} \min_{|\mathbf{X}|=M} \mathcal{L}_{\mathbf{X}}(q). \tag{4}$$

To solve this minimax problem, we can adopt the similar inner and outer loop optimization like did in GANs [8]. In the inner loop, we minimize $\mathcal{L}_{\mathbf{X}}(q)$ with respect to $\mathbf{X}$; in the outer loop, we maximize $\mathcal{L}_{\mathbf{X}}(q)$ with respect to the generator q.

Sampling-Based Functional Variational Inference Adversarial functional variational inference plays a minimax game, which is difficult to optimize and takes longer time for convergence [8]. Here we present sampling-based functional variational inference as an alternative, which only jointly optimizes a same objective like VAE [13]. Specifically, we replace the minimization in eq. (4) with a sampling distribution c, and then maximize the expected $\mathcal{L}_{\mathbf{X}}(q)$ under this distribution.

However, without the minimization, the objective will be inclined to overfit on training locations because of the log likelihood term. We therefore let the measurement points $\mathbf{X}$ include random subset $\mathbf{X}^{D_s}$ of training data and denote $\mathbf{X}^M = \mathbf{X}\backslash\mathbf{X}^{D_s}$. Then the sampling-based functional variational inference becomes,

$$\max_{q\in\mathcal{Q}} \mathbb{E}_{\mathcal{D}_s}\mathbb{E}_{\mathbf{X}^M\sim c}\, \mathcal{L}_{\mathbf{X}^M,\mathbf{X}^{D_s}}(q). \tag{5}$$

where $\mathbf{X}^M$ are M points independently drawn from c. Denote $\mathbf{f}^M, \mathbf{f}^D$ as the function values for $\mathbf{X}^M, \mathbf{X}^D$, respectively. By theorem 2, if $\mathbf{X}$ contains all of $\mathbf{X}^D$,

$$\mathcal{L}_{\mathbf{X}^M,\mathbf{X}^D}(q) = \log p(\mathcal{D}) - \mathrm{KL}[q(\mathbf{f}^M, \mathbf{f}^D)\|p(\mathbf{f}^M, \mathbf{f}^D|\mathcal{D})]. \tag{6}$$

Maximizing $\mathcal{L}_{\mathbf{X}^M,\mathbf{X}^D}(q)$ is equivalent to minimizing the KL divergence from the true posterior on points $\mathbf{X}^M, \mathbf{X}^D$. Therefore, another interpretation of sampling-based functional variational inference is that we want better posterior approximation on more "interesting" locations weighted by c.

2.3 KL divergence Gradients

Computing fELBO requires to compute the likelihood term and the KL divergence. Although the likelihood term is tractable, the KL divergence term remains intractable because we don't have the explicit formula for variational posterior $q_\phi(\mathbf{f}^{\mathbf{X}}|\mathbf{X})$. Note $\nabla_\phi \mathrm{KL}[q_\phi(\mathbf{f}^{\mathbf{X}})\|p(\mathbf{f}^{\mathbf{X}})]$ is given by

$$\mathbb{E}_q\left[\nabla_\phi \log q_\phi(\mathbf{f}^{\mathbf{X}})\right] + \mathbb{E}_\xi\left[\nabla_\phi \mathbf{f}^{\mathbf{X}}(\nabla_{\mathbf{f}} \log q(\mathbf{f}^{\mathbf{X}}) - \nabla_{\mathbf{f}} \log p(\mathbf{f}^{\mathbf{X}}))\right]. \tag{7}$$

It is easy to check that the first term in eq. (7) is zero [26]. Besides that, using parametric variational posterior, the gradients $\nabla_\phi \mathbf{f}^{\mathbf{X}}$ can be computed easily by backpropagation. All we left are the log-density derivatives $\nabla_{\mathbf{f}} \log q(\mathbf{f}^{\mathbf{X}})$ and $\nabla_{\mathbf{f}} \log p(\mathbf{f}^{\mathbf{X}})$. For priors with explicit densities like Gaussian Processes [22], Student-t process [27] and Wishart Process [5], $\nabla_{\mathbf{f}} \log p(\mathbf{f}^{\mathbf{X}})$ is analytic.

Spectral Stein Gradient Estimator (SSGE) [29] is a recently proposed method for estimating the log density derivative function of an implicit distribution, i.e. $\nabla_{\mathbf{f}} \log q(\mathbf{f}^{\mathbf{X}})$, only requiring samples from the distribution. Specifically, given a continuous differentiable density $q(\mathbf{x})$, and a positive definite kernel $k(\mathbf{x}, \mathbf{x}')$ in the Stein class [14] of q, they show

$$\nabla_{x_i} \log q(\mathbf{x}) = -\sum_{j=1}^{\infty}\left[\mathbb{E}_q \nabla_{x_i}\psi_j(\mathbf{x})\right]\psi_j(\mathbf{x}), \tag{8}$$

where $\{\psi_j\}_{j\geq 1}$ is a series of eigenfunctions of k described by the Mercer's theorem: $k(\mathbf{x}, \mathbf{x}') = \sum_j \mu_j \psi_j(\mathbf{x})\psi_j(\mathbf{x}')$. Then the Nyström method [3, 34] is used to approximate the eigenfunctions $\psi_j(\mathbf{x})$ and their derivatives. The final estimator is given by truncating the sum in eq. (8) and replacing the expectation by Monte Carlo estimates.

[2]The ordinary GAN objective is typically infinite for an unrestricted discriminator because the generator typically generates from a submanifold of data space.

3 Experiments with Contextual Bandits

Table 1: Contextual bandits regret. Results are relative to the cumulative regret of the Uniform algorithm. Numbers after the algorithm correspond to the network size. We report the mean value and mean rank over multiple datasets.

	FBNN 1×50	FBNN 2×50	FBNN 1×500	FBNN 2×500	MultiTaskGP	BBB 1×50	BBB 1×500
M. Rank	5.875	7.125	**4.875**	**5.0**	5.875	11.5	13.375
M. Value	46.0	47.0	**45.3**	**44.2**	46.5	56.6	68.1
	BBAlphaDiv	ParamNoise	NeuralLinear	LinFullPost	Dropout	RMS	Uniform
M. Rank	16.0	10.125	10.375	9.25	7.625	8.875	16.75
M. Value	87.4	53.0	52.3	NAN	48.3	53.0	100

We compared our fBNNs with the algorithms benchmarked in [25]. We used the multi-task GP of [25] as the prior for the fBNNs. We run the experiments for all algorithms and tasks using the default settings open sourced by [25]. For fBNNs, we also kept the same setting, including batchsize, training epochs and training frequency. Measurement sets consisted of training batches, combined with 10 points sampled from c. We ran each experiment 10 times and report the mean and standard derivation in Table 4. Similarly to [25], we also report the mean rank and mean regret.

As shown in Table 4, fBNNs outperformed other methods by a wide margin. Additionally, the fBNNs maintained consistent performance with various network sizes. By comparison, BBB suffered significant performance degeneration when the hidden size was increased from 50 to 500. This is consistent with our hypothesis that functional variational inference can gracefully handle networks with high capacity. We also show a BO experiment in App E.5, which also needs reliable uncertainty.

Acknowledgements

We thank Ricky T. Q. Chen and Kevin Luk for their helpful comments on this project. SS was supported by a Connaught New Researcher Award and a Connaught Fellowship. GZ was funded by an MRIS Early Researcher Award.

References

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

[2] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[3] Christopher T. Baker. *The Numerical Treatment of Integral Equations*. Clarendon Press, Oxford, 1997.

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

[5] A Philip Dawid. Some matrix-variate distribution theory: notational considerations and a bayesian application. *Biometrika*, 68(1):265–274, 1981.

[6] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Uncertainty decomposition in bayesian neural networks with latent variables. *arXiv preprint arXiv:1706.08495*, 2017.

[7] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. *arXiv preprint arXiv:1806.05975*, 2018.

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

[9] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[10] Robert M. Gray. *Entropy and Infomation Theory*. Springer, 2011.

[11] 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. ACM, 1993.

[12] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

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

[14] Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In *International Conference on Machine Learning*, pages 276–284, 2016.

[15] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[16] 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.

[17] Anton Mallasto and Aasa Feragen. Learning from uncertain curves: The 2-wasserstein metric for gaussian processes. In *Advances in Neural Information Processing Systems*, pages 5660–5670, 2017.

[18] Radford M Neal. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto, 1995.

[19] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

[20] Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003.

[21] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

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

[23] Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Advances in neural information processing systems*, pages 294–300, 2001.

[24] Carl Edward Rasmussen and Christopher KI Williams. Gaussian processes for machine learning. 2006.

[25] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

[26] Geoffrey Roeder, Yuhuai Wu, and David Duvenaud. Sticking the landing: An asymptotically zero-variance gradient estimator for variational inference. *arXiv preprint arXiv:1703.09194*, 2017.

[27] Amar Shah, Andrew Wilson, and Zoubin Ghahramani. Student-t processes as alternatives to gaussian processes. In *Artificial Intelligence and Statistics*, pages 877–885, 2014.

[28] Jiaxin Shi, Shengyang Sun, and Jun Zhu. Kernel implicit variational inference. *arXiv preprint arXiv:1705.10119*, 2017.

[29] Jiaxin Shi, Shengyang Sun, and Jun Zhu. A spectral approach to gradient estimation for implicit distributions. *International Conference on Machine Learning*, 2018.

[30] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised map inference for image super-resolution. *arXiv preprint arXiv:1610.04490*, 2016.

[31] Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pages 1283–1292, 2017.

[32] Shengyang Sun, Guodong Zhang, Chaoqi Wang, Wenyuan Zeng, Jiaman Li, and Roger Grosse. Differentiable compositional kernel learning for gaussian processes. *arXiv preprint arXiv:1806.04326*, 2018.

[33] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. *arXiv preprint arXiv:1703.01968*, 2017.

[34] Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688, 2001.

[35] Andrew G Wilson, Elad Gilboa, Arye Nehorai, and John P Cunningham. Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pages 3626–3634, 2014.

[36] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2017.

A Background

A.1 Variational Inference for Bayesian Neural Networks

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, a Bayesian neural network (BNN) is defined in terms of a prior $p(\mathbf{w})$ on the weights, as well as the likelihood $p(\mathcal{D}|\mathbf{w})$. Variational Bayesian methods [11, 9, 4] attempt to fit an approximate posterior $q(\mathbf{w})$ to maximize the evidence lower bound (ELBO):

$$\mathcal{L}_q = \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{w})] - \mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})]. \tag{9}$$

The most commonly used variational BNN training method is Bayes By Backprop (BBB) [4], which uses a fully factorized Gaussian approximation to the posterior, i.e. $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. Using the reparameterization trick [13], the gradients of ELBO towards μ, σ can be computed by backpropagation, and then be used for updates.

Most commonly, the prior $p(\mathbf{w})$ is chosen for computational convenience; common choices include independent Gaussian or Gaussian mixture distributions. Other weight priors, including log-uniform priors [12, 15] and horseshoe priors [7, 15], have been proposed for particular purposes such as model compression and model selection. However, the relationships of weight space priors to the functions computed by networks are difficult to characterize.

A.2 Stochastic Processes

A stochastic process is defined as a random function $F : \mathcal{X} \to \mathcal{Y}$. For any finite subset $\mathbf{x}_{1:n}$, F can compute its marginal joint distribution over the function values $F(\mathbf{x}_{1:n})$. For example, the Gaussian Process has its marginal distributions as multivariate Gaussians, and the Student-t Process [27] has its marginal distribution as multivariate Student-t distributions.

However, directly representing a stochastic process is difficult because $\mathcal{X}$ is typically infinite. Kolmogorov Extension Theorem [20] provides an alternative using finite dimensional marginal distributions. Specifically, for a collection of joint distributions $\rho_{\mathbf{x}_{1:n}}$, Kolmogorov Extension Theorem states that, ρ defines a random process F such that $\rho_{\mathbf{x}_{1:n}}$ is the marginal distribution of $F(\mathbf{x}_{1:n})$ if ρ satisfies the following two conditions:

Exchangeability For any permutation π of $\{1, \cdots, n\}$, $\rho_{\pi(\mathbf{x}_{1:n})}(\pi(y_{1:n})) = \rho_{\mathbf{x}_{1:n}}(y_{1:n})$.

Consistency For any $1 \le m \le n$, $\rho_{\mathbf{x}_{1:m}}(y_{1:m}) = \int \rho_{\mathbf{x}_{1:n}}(y_{1:n}) dy_{m+1:n}$.

Bayesian neural networks satisfy both exchangeability and consistency, therefore it defines a stochastic process.

B The Algorithm

B.1 Injected Noises for Gaussian Process Priors

For Gaussian Process prior, $p(\mathbf{f}^{\mathbf{X}})$ is multivariate Gaussian distribution, which has an explicit formula. Therefore, we can compute the gradients $\nabla_{\mathbf{f}} \log p_\phi(\mathbf{f}^{\mathbf{X}})$ analytically.

In practice, we found that the GP kernel matrix suffers from statbility issues. To stabilize the gradients computation, we propose to inject a small amount of Gaussian noise on the function values, i.e., to instead estimate the gradients of $\nabla_\phi \mathrm{KL}[q_\phi * p_\gamma \| p * p_\gamma]$, where $p_\gamma = \mathcal{N}(0, \gamma^2)$ is the noise distribution. This is like the instance-noise trick that is commonly used for stabilizing GAN training [30]. Note that injecting the noise on the GP prior is equivalent to have a kernel matrix $\mathbf{K} + \gamma^2 \mathbf{I}$, which have more stable properties. Beyond that, injecting the noise on the parametric variational posterior doesn't affect the reparameterization trick either. Therefore all the previous estimation formulas still applies.

B.2 The Algorithm

Throughout this paper, we mainly investigate sampling-based functional variational inference.

Now we present the whole algorithm for fBNNs in Algorithm 1. Because the log likelihood term has unbiased mini-batch estimations, we estimate $\mathcal{L}_{\mathbf{X}}(q)$ with mini-batch $\mathcal{D}_s$ for fast computation.

Specifically, we are optimizing

$$\sum_{(\mathbf{x},y)\in\mathcal{D}_s} \mathrm{E}_{q_\phi}\left[\log p(y|f(x))\right] - \mathrm{KL}[q(\mathbf{f}^M, \mathbf{f}^{\mathcal{D}_s})\|p(\mathbf{f}^{\mathcal{D}_s}, \mathbf{f}^M)] \tag{10}$$

which is also a proper lower bound of $\log p(\mathcal{D}_s)$.

Algorithm 1 Functional Variational Bayesian Neural Networks (fBNNs)

```
Require: Dataset $\mathcal{D}$, sampling distribution $c$, variational posterior $g(\cdot)$, prior $p$ (explicit or implicit).
1: while $\phi$ not converged do
2:     $\mathbf{X}^M \sim c$; $D_S \subset \mathcal{D}$                                              ▷ sample measurement points
3:     $\mathbf{f}_i = g(\{\mathbf{X}^M, \mathbf{X}^{D_S}\}, \xi_i; \phi),\ i = 1\cdots k.$                  ▷ sample $k$ function values
4:     $\mathbf{g}_1 = \frac{1}{k}\sum_i \sum_{(x,y)} \nabla_\phi \log p(y|\mathbf{f}_i(x))$         ▷ compute log likelihood gradients
5:     $\mathbf{g}_2 = \mathrm{SSGE}(p, \mathbf{f}_{1:k})$                                        ▷ estimate KL gradients
6:     $\phi \leftarrow$ Optimizer$(\phi, \mathbf{g}_1 - \mathbf{g}_2)$                                 ▷ update the parameters
7: end while
```

C Stochastic Processes

C.1 Basic Measure Theory Notations

π-system A π-system is a collection of subsets that is closed under finitely many intersections.

σ-algebra generated by cylinder sets Suppose

$$X \subset \mathbb{R}^{\mathbb{T}} = \{f : f(t) \in \mathbb{R}, t \in \mathbb{T}\}$$

is a set of real-valued functions. Let $\mathcal{B}(\mathbb{R})$ denote the Borel subsets of $\mathbb{R}$. A cylinder subset of X is a finitely restricted set defined as

$$C_{t_1,\cdots,t_n}(B_1, \cdots, B_n) = \{f \in X : f(t_i) \in B_i, 1 \le i \le n\}$$

Each

$$\mathcal{G}_{t_1,\cdots,t_n} = \{C_{t_1,\cdots,t_n}(B_1, \cdots, B_n) : B_i \in \mathcal{B}(\mathbb{R}), 1 \le i \le n\}$$

is a π-system that generates a σ-algebra $\Sigma_{t_1,\cdots,t_n}$. Then the family of subsets

$$\mathcal{F}_X = \bigcup_{n=1}^{\infty} \bigcup_{t_i\in\mathbb{T}, i\le n} \Sigma_{t_1,\cdots,t_n}$$

is an algebra that generates the cylinder σ-algebra for X.

Definition 1 (KL divergence [10])**.** *Given two probability measures q and p on measure space $(\Omega, \mathscr{B})$, the KL divergence of q with respect to p is defined as*

$$\mathrm{KL}[q\|p] = \sup_Q \mathrm{KL}_Q[q\|p]. \tag{11}$$

where the supremum is over all finite measurable partitions Q of Ω.

C.2 PushForward and Kolomogrov Extension Theorem

Definition 2 (Pushforward measure)**.** *Given probability spaces (X, Σ_X, μ) and (Y, Σ_Y, ν), we say that measure ν is a pushforward of μ if $\nu(A) = \mu(T^{-1}(A))$ for a measurable $T : X \to Y$ and any $A \in \Sigma_Y$, denoted by $\nu = \mu \circ T^{-1}$.*

Definition 3 (Canonical projection map)**.** *Let T be an arbitrary index set, and $\{(\Omega_t, \mathcal{F}_t)\}_{t\in T}$ be some collection of measurable spaces. For each subset $J \subset I \subset T$, define $\Omega^J = \prod_{t\in J} \Omega_t$. We call $\pi_{I\to J}$ the canonical projection map from I to J if*

$$\pi_{I\to J}(w) = w|_J \in \Omega^J, \forall w \in \Omega^I. \tag{12}$$

Theorem 3 (Kolmogorov extension theorem)**.** *Let T be an arbitrary index set. $(\Omega, \mathcal{F})$ is a standard measure space, whose product space on T is $(\Omega^T, \mathcal{F}^T)$. For each finite subset $I \subset T$, suppose we have a probability measure μ_I on Ω^I that satisfies the following compatibility relation: For each finite subset $J \subset I \subset T$, we have*

$$\mu_J = \mu_I \circ \pi^{-1}_{I \to J}. \tag{13}$$

Then there exists a unique probability measure μ on Ω^T such that $\forall$ finite subset $I \subset T$,

$$\mu_I = \mu \circ \pi^{-1}_{T \to I}. \tag{14}$$

In Gaussian processes, μ is a Borel measure on a separable Hilbert space $(\mathcal{H}, \mathscr{B}(\mathcal{H}))$. μ_I is a marginal Gaussian measure at any finite set (i.e., I) of input positions [17].

C.3 Proof

We first introduce several lemmas for proving the theorem.

Lemma 4.

$$\mathcal{G}_X = \overset{\infty}{\underset{n=1}{\cup}} \underset{t_i \in \mathbb{T}, i \leq n}{\cup} \mathcal{G}_{t_1, \cdots, t_n}$$

is a π-system. And $\sigma(\mathcal{G}_X)$ generates the same cylinder σ-algebra of X as $\mathcal{F}_X$.

Definition 4. *Let H be a set of functions, define its $\mathbb{T}$-set $\tau(H)$ as*

$$\tau(H) = \mathbb{T} \backslash \{t | t \in \mathbb{T}, \exists S_{\mathbb{T}\backslash\{t\}}, s.t.\ S_{\mathbb{T}\backslash\{t\}} \times \mathbb{R}_t = H\}$$

Here $\times$ represents product-algebra. Intuitively, $\tau(H)$ represents all timesteps that restricts the functions in H. Or we can say if some t doesn't belong to $\tau(H)$, then function values of t can be any number in real line $\mathbb{R}$.

Lemma 5. *For any set $G \in \sigma(\mathcal{G}_X)$, its $\mathbb{T}$-set $\tau(G)$ is countable.*

Proof. We build the sigma-algebra $\sigma(\mathcal{G}_X)$ through an iterative process. We begin by $\mathcal{H}_0 = \mathcal{G}_X$, in each iteration, let

$$\begin{aligned} \mathcal{H}_{i+1} &= \mathcal{H}_i \cup \{H^c | H \in \mathcal{H}_i\}, \\ \mathcal{H}_{i+1} &= \mathcal{H}_{i+1} \cup \{\underset{j}{\cup} H_j | H_1, H_2, \cdots \in \mathcal{H}_i\}. \end{aligned} \tag{15}$$

Recursively applying this extension, we can generate $\sigma(\mathcal{G}_X) = \mathcal{H}_\infty$.

Because $\tau(H^c) = \tau(H)$, and countable union of countable sets is still countable. We know the two updates above keep the countability of its $\mathbb{T}$-set. Therefore, $\forall G \in \sigma(\mathcal{G}_X), \tau(G)$ is countable.

□

Theorem 6. *For two stochastic processes P, M on functional measure space $(\Omega^{\mathbb{T}}, \mathcal{F}^{\mathbb{T}})$, the KL divergence between P and M,*

$$\mathrm{KL}[P \| M] = \sup_{\mathbb{T}_f} \mathrm{KL}[P_{\mathbb{T}_f} \| M_{\mathbb{T}_f}],$$

Where the supremum is over all finite subsets, and $P_{\mathbb{T}_f}, M_{\mathbb{T}_f}$ represent the marginal distribution of P, M on $\mathbb{T}_f$, respectively.

Proof. **Sketches**

1. The cylinder σ-algebra defines a σ-algebra $\sigma(\mathcal{G}_X)$ on the function space. By Lemma 4.
2. Each set in the $\sigma(\mathcal{G}_X)$ is only related to countable number of timesteps. By Lemma 5.
3. Any finite measurable partitions are related to countable number of timesteps, thus correspond to the marginal distribution over a countable subset by an inclusion map. Therefore the KL between stochastic processes reduces to the KL over marginal distributions of countable subsets.

4. KL over marginal distributions of countable subsets can be represented as supremum of KL over marginal distributions over its finite subsets.

By Definition 1,

$$\mathrm{KL}[P\|M] = \sup_{Q_{\Omega^{\mathbb{T}}}} \mathrm{KL}_{Q_{\Omega^{\mathbb{T}}}}[P\|M],$$

where the sup is over all finite measurable partitions of the function space, denoted by $Q_{\Omega^{\mathbb{T}}}$:

$$Q_{\Omega^{\mathbb{T}}} = \{Q^{(1)}_{\Omega^{\mathbb{T}}}, \dots, Q^{(k)}_{\Omega^{\mathbb{T}}} | \bigcup_{i=1}^{k} Q^{(i)}_{\Omega^{\mathbb{T}}} = \Omega^{\mathbb{T}}, Q^{(i)}_{\Omega^{\mathbb{T}}} \in \mathcal{F}^{\mathbb{T}} \text{ are disjoint sets}, k \in \mathbb{N}^{+}\}$$

By Lemma 5, $\tau(Q^{(i)}_{\Omega^{\mathbb{T}}})$ is countable. Therefore, $\mathbb{T}_c := \bigcup_{i=1}^{k} \tau(Q^{(i)}_{\Omega^{\mathbb{T}}})$ is countable.

Consider the canonical projection mapping $\pi_{\mathbb{T}\to\mathbb{T}_c}$, which induces a partition on $\Omega^{\mathbb{T}_c}$, denoted by $Q_{\Omega^{\mathbb{T}_c}}$:

$$Q^{(i)}_{\Omega^{\mathbb{T}_c}} = \pi_{\mathbb{T}\to\mathbb{T}_c}(Q^{(i)}_{\Omega^{\mathbb{T}}}).$$

The pushforward measure defined by this mapping is

$$P_{\mathbb{T}_c} = P \circ \pi^{-1}_{\mathbb{T}\to\mathbb{T}_c}, \quad M_{\mathbb{T}_c} = M \circ \pi^{-1}_{\mathbb{T}\to\mathbb{T}_c}.$$

Then we have

$$\begin{aligned}
\mathrm{KL}[P\|M] &= \sup_{Q_{\Omega^{\mathbb{T}}}} \mathrm{KL}_{Q_{\Omega^{\mathbb{T}}}}[P\|M] && (16)\\
&= \sup_{Q_{\Omega^{\mathbb{T}}}} \sum_i P(Q^{(i)}_{\Omega^{\mathbb{T}}}) \log \frac{P(Q^{(i)}_{\Omega^{\mathbb{T}}})}{M(Q^{(i)}_{\Omega^{\mathbb{T}}})} && (17)\\
&= \sup_{\mathbb{T}_c} \sup_{Q_{\Omega^{\mathbb{T}_c}}} \sum_i P_{\mathbb{T}_c}(Q^{(i)}_{\Omega^{\mathbb{T}_c}}) \log \frac{P_{\mathbb{T}_c}(Q^{(i)}_{\Omega^{\mathbb{T}_c}})}{M_{\mathbb{T}_c}(Q^{(i)}_{\Omega^{\mathbb{T}_c}})} && (18)\\
&= \sup_{\mathbb{T}_c} \mathrm{KL}[P_{\mathbb{T}_c}\|M_{\mathbb{T}_c}] && (19)
\end{aligned}$$

Denote $\mathcal{F}(\mathbb{T}_c)$ as the collection of all finite subsets of $\mathbb{T}_c$. For any finite set $\mathbb{T}_j \in \mathcal{F}(\mathbb{T}_c)$, we denote $P_{\mathbb{T}_j}$ as the pushforward of P on finite set $\mathbb{T}_j$. According to Kolmogorov Extension Theorem (Theorem 3), $P_{\mathbb{T}_j}$ is the marginal distribution of P at $\mathbb{T}_j$. Because $\mathbb{T}_c$ is countable, we have

$$\begin{aligned}
\mathrm{KL}[P\|M] &= \sup_{\mathbb{T}_c} \mathrm{KL}[P_{\mathbb{T}_c}\|M_{\mathbb{T}_c}] && (20)\\
&= \sup_{\mathbb{T}_c} \sup_{\mathbb{T}_j \in \mathcal{F}(\mathbb{T}_c)} \mathrm{KL}[P_{\mathbb{T}_c}\|M_{\mathbb{T}_c}] && (21)
\end{aligned}$$

Note for any finite subset $\mathbb{T}_j$, we can build a finite measurable partition $Q_{\Omega^{\mathbb{T}}}$ such that its $\mathbb{T}_c$ equals to $\mathbb{T}_j$. Let Ω_1, Ω_2 be any two-element partition of Ω. In fact, we can let $Q_{\Omega^{\mathbb{T}}}$ be a partition with $2^{|\mathbb{T}_j|}$ subsets, that each subset is a cylinder subset with timesteps in $\mathbb{T}_j$ and for each timestep, the function values belong to Ω_1 or Ω_2. Therefore, we have shown the set $\{\mathbb{T}_j | \mathbb{T}_j \in \mathcal{F}(\mathbb{T}_c), \mathbb{T}_c\}$ contains all finite subsets.

On the other hand, every element in $\{\mathbb{T}_j | \mathbb{T}_j \in \mathcal{F}(\mathbb{T}_c), \mathbb{T}_c\}$ is a finite subset, we know $\{\mathbb{T}_j | \mathbb{T}_j \in \mathcal{F}(\mathbb{T}_c), \mathbb{T}_c\}$ equals to the collection of all finite subsets.

Therefore we have proven the theorem. □

D Additional Theoretical Results

D.1 Consistency for Gaussian Process

For both adversarial and sampling-based functional variational inference, we cannot enumerate finite subsets of all lengths. However, we conclude that when both the prior and variational process are GPs, the sampling-based method guarantees that the optimal variational posterior matches the true posterior GP.

Theorem 7. *When p and q are both Gaussian Processes, q's family is expressive enough, $M \geq 2$ and the support of c is $\mathcal{X}$, then the following statements are equivalent:*

- *eq. (5) attains its optimum.*
- *the variational posterior process and the true posterior process are identical.*

Proof. Assume the GP form of $p(f(\cdot)|\mathcal{D})$ and $q(f(\cdot))$:

$$p(f(\cdot)|\mathcal{D}): \quad \mathcal{GP}(m_p(\cdot), k_p(\cdot,\cdot)), \tag{22}$$

$$q(f(\cdot)): \quad \mathcal{GP}(m_q(\cdot), k_q(\cdot,\cdot)), \tag{23}$$

where m and k denote the mean and covariance functions, respectively. Because the KL is non-negative, then eq. (5) achieves optimum if and only if $\mathrm{KL}[q_\phi(f(\mathbf{X}^M))\|p(f(\mathbf{X}^M)|\mathcal{D})] = 0$ at any $\mathbf{X}^M = [\mathbf{x}_1, \ldots, \mathbf{x}_M]^\top \in \mathcal{X}^M$, which is

$$\mathcal{N}(m_p(\mathbf{X}), \mathbf{K}_p) = \mathcal{N}(m_q(\mathbf{X}), \mathbf{K}_q), \tag{24}$$

where $m(\mathbf{X}) = [m(\mathbf{x}_1), \ldots, m(\mathbf{x}_M)]^\top$, and $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Given $M \geq 2$, then for $\forall 1 \leq i < j \leq M$, we have $m_p(\mathbf{x}_i) = m_q(\mathbf{x}_i)$, and $k_p(\mathbf{x}_i, \mathbf{x}_j) = k_q(\mathbf{x}_i, \mathbf{x}_j)$. Remember that $\mathbf{X}^M$ is chosen arbitrarily, thus we have

$$m_p(\cdot) = m_q(\cdot), \quad k_p(\cdot,\cdot) = k_q(\cdot,\cdot). \tag{25}$$

Because GPs are uniquely determined by their mean and covariance functions, we arrive at the conclusion. □

D.2 Optimize GP Hyperparameters using Mini-Batch

Consider multivariate Gaussian distribution,

$$\log p(\mathbf{y}|\mathbf{0}, \mathbf{K}_\theta) = -\frac{1}{2}(\mathbf{y}^\top \mathbf{K}_\theta^{-1}\mathbf{y} + \log|\mathbf{K}_\theta|) + \mathrm{const}$$

Its derivative towards $\mathbf{K}_\theta$ is $-\frac{1}{2}\mathbf{K}_\theta^{-1}(\mathbf{K}_\theta - \mathbf{y}\mathbf{y}^\top)\mathbf{K}_\theta^{-1}$, which attains zero only at $\mathbf{K}^\star = \mathbf{y}\mathbf{y}^\top$. Therefore we have

$$\log p(\mathbf{y}|\mathbf{0}, \mathbf{K}_\theta) \leq \log p(\mathbf{y}|\mathbf{0}, \mathbf{K}^\star) \tag{26}$$

Gaussian Process optimization maximizes data likelihood $\log p(\mathbf{y}|\mathbf{0}, \mathbf{K}_\theta)$. According to eq. (26), its maximum is obtained at $\mathbf{K}^\star = \mathbf{y}\mathbf{y}^\top$. In contrast, mini-batch GP optimization maximizes expected subset likelihood $\mathbb{E}_S\left[\log p(\mathbf{y}_S|\mathbf{0}, \mathbf{K}_\theta^S)\right]$, with S being a subset. According to eq. (26), $\log p(\mathbf{y}_S|\mathbf{0}, \mathbf{K}_\theta^S) \leq \log p(\mathbf{y}_S|\mathbf{0}, \mathbf{y}_S\mathbf{y}_S^\top)$. Therefore, its maximum is also obtained at $\mathbf{K}^\star = \mathbf{y}\mathbf{y}^\top$.

Based on the results above, for expressive enough kernels, mini-batch optimized GP reaches the same maximum as full-batch optimized GP, which justifies using mini-batch MLE for hyperparameter optimization.

D.3 Proof for Evidence Lower Bound

This section provides proof for theorem 2.

Let $\mathbf{X}^M = \mathbf{X} \backslash \mathbf{X}^D$ be measurement points which aren't in the training data.

Proof.

$$\mathrm{KL}[q_\phi(\mathbf{f}^D, \mathbf{f}^M)\|p(\mathbf{f}^D, \mathbf{f}^M|\mathcal{D})] = \mathbb{E}_q\left[\log \frac{q_\phi(\mathbf{f}^D, \mathbf{f}^M)}{p(\mathbf{f}^D, \mathbf{f}^M|\mathcal{D})}\right] \tag{27}$$

$$= \mathbb{E}_q\left[\log \frac{q_\phi(\mathbf{f}^D, \mathbf{f}^M)p(\mathcal{D})}{p(\mathbf{y}^D|\mathbf{f}^D)p(\mathbf{f}^D, \mathbf{f}^M)}\right]$$

$$= \log p(\mathcal{D}) - \mathcal{L}_{\mathbf{X}}(q) \tag{28}$$

□

E Additional Experiments

General Setting The variational posterior is represented as a stochastic neural network with independent Gaussian distributions over the weights, i.e. $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$.[3] We always used the ReLU activation function unless otherwise specified. Denote xmin, xmax as dimension-wise minimal and maximal input locations, $\text{xd} = \text{xmax} - \text{xmin}$. We set the sampling distribution $c = \mathbb{U}[\text{xmin} - \text{xd}/2, \text{xmax} + \text{xd}/2]$ unless specific notice. For experiments where we used GP priors, we first fit the GP hyperparameters to maximize the marginal likelihood on subsets of the training examples, and then fixed those hyperparameters to obtain the prior for the fBNNs.

E.1 Implicit Priors

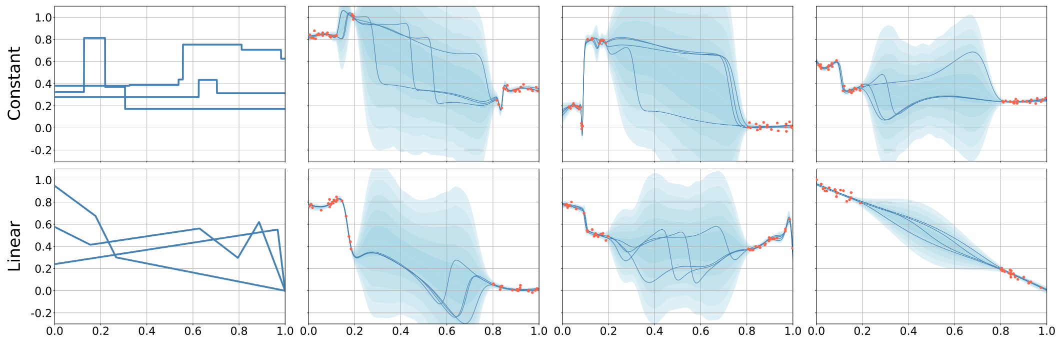


Figure 3: Implicit function priors and variational posteriors. The leftmost column shows 3 prior samples. The other columns are different runs. Red dots are 40 training samples. We plot 4 posterior samples and plot the standard derivation as shaded areas.

Since the fBNN is based on implicit variational inference, the priors need not have convenient expressions for the marginal densities, i.e. we need not limit ourselves to GPs. In this section, we consider two implicit priors: a distribution over piecewise constant functions, and a distribution over piecewise linear functions. **??** provides the detailed explanation on the sampling process. Some samples from these priors are shown in Figure 3.

Concretely, we randomly generate a function $f : [0, 1] \to R$ with the specific structure. To sample piecewise functions, we first sample $n \sim \text{Poisson}(3.)$, then we have $n + 1$ pieces within $[0, 1]$. We uniformly sample n locations from $[0, 1]$ as the changing points. For piecewise constant functions, we uniformly sample $n + 1$ values from $[0, 1]$ as the function values in each piece; For piecewise linear functions, we uniformly sample $n + 1$ values for the values at first $n + 1$ locations, we force $f(1) = 0.$. Then we connect together each piece by a straight line.

In each run of the experiment, we first sampled a random function from the prior, and then sampled 20 locations from $[0, 0.2]$ and 20 locations from $[0.8, 1]$, giving a training set of 40 data points. The standard deviation of observation noise is 0.02. In each iteration, measurement points included all training examples, plus 40 points randomly sampled from $[0, 1]$. We used a fully connected network with 2 hidden layers of 100 units, and tanh activations. (For this experiment, we used tanh rather than ReLU because ReLU would have been unfairly well-suited to the non-smooth priors.) The network was trained for 20,000 iterations. We show the predictive samples and variances of three different runs in Figure 3.

As shown in Figure 3, the fBNN captured the basic structures for both piecewise constant priors and piecewise linear priors, although the posterior samples did not seem to capture the full diversity of possible explanations of the data. Interestingly, even though the tanh activation function encourages smoothness, the network learned to generate functions with sharp transitions.

[3]We note that other choices are possible, since functional variational inference imposes no requirement that epistemic uncertainty be represented in terms of a distribution over the weights of a network. (E.g., in principle, a distribution over functions could be represented with a deterministic network with stochastic inputs.) However, we stick with Gaussian distributions on the weights because it worked as well as anything else we've tried.

E.2 Predictive Performance

To compare with previous work on predictive performance, we next evaluated fBNN in regression experiments with publicly available datasets.

E.2.1 Small Scale Datasets

Following previous work [36], we first experiment with standard regression benchmark datasets from the UCI collection [2]. In particular, we only used the datasets with less than 2000 data points so that we can fit GP hyperparameters using marginal likelihood. The datasets were randomly split into training and validation sets, comprising 90% and 10% of the data respectively. This splitting process was repeated 10 times to reduce randomness.

Table 2: Averaged validation RMSE and log-likelihood for the regression benchmarks.

	Validation RMSE			Validation log-likelihood		
Dataset	BBB	NNG	Ours	BBB	NNG	Ours
Boston	3.171±0.149	2.742±0.125	**2.378±0.104**	-2.602±0.031	-2.446±0.029	**-2.301±0.038**
Concrete	5.678±0.087	5.019±0.127	**4.935±0.180**	-3.149±0.018	**-3.039±0.025**	-3.096±0.016
Energy	0.565±0.018	0.485±0.023	**0.412±0.017**	-1.500±0.006	-1.421±0.005	**-0.684±0.020**
Wine	0.643±0.012	**0.637±0.011**	0.673±0.014	-0.977±0.017	**-0.969±0.014**	-1.040±0.013
Yacht	1.174±0.086	0.979±0.077	**0.607±0.068**	-2.408±0.007	-2.316±0.006	**-1.033±0.033**

For all datasets, we used networks with one hidden layer of 50 hidden units. For all experiments, we first fit GP hyper-parameters using marginal likelihood with a budget of 10,000 iterations. In each experiment, the fBNN was trained for 2,000 epochs. And in each iteration, measurement points included 20 training examples, plus 5 points randomly sampled from the data range.

We compared our method with Bayes By Backprop (BBB) [4] and Noisy Natural Gradient (NNG) [36]. In accordance with [36], we report the standard metrics including root mean square error (RMSE) and validation log-likelihood. The results are summarized in Table 2. On most datasets, our fBNNs outperformed both BBB and NNG, sometimes by a significant margin.

E.2.2 Large Scale Datasets

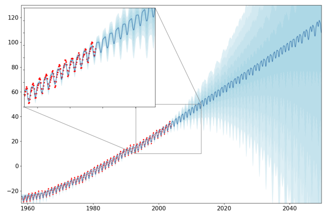
Table 3: Averaged validation RMSE and log-likelihood for the regression benchmarks.

		Test RMSE			Test log-likelihood		
Dataset	N	BP	BBB	FBNN	BP	BBB	FBNN
Naval	11934	(1.3±0.12)E-4	(1.6±0.21)E-4	**(1.1±0.08)E-4**	7.121±0.025	6.950±0.052	**7.153±0.025**
Protein	45730	4.159±0.045	4.331±0.033	**4.142±0.044**	-2.854±0.012	-2.892±0.007	**-2.850±0.012**
Video Mem.	68784	2.178±1.063	**1.879±0.265**	1.994±0.805	-2.421±0.994	-1.999±0.054	**-1.912±0.101**
Video Time	68784	2.761±0.385	3.632±1.974	**2.529±0.248**	-2.512±0.246	-2.390±0.040	**-2.375±0.104**
GPU	241600	21.039±0.607	21.886±0.673	**20.75±0.771**	-4.471±0.029	-4.505±0.031	**-4.456±0.037**
Year	515345	9.394	**9.311**	9.356	-3.672	**-3.646**	-3.669

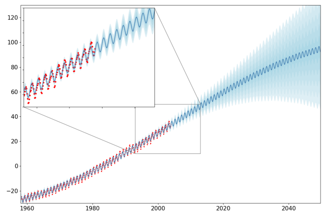
Observe that fBNNs are naturally scalable to large datasets because they access the data only through the expected log-likelihood term, which can be estimated stochastically. In this section, we verify this experimentally. We compared fBNNs, BBB and BP with large scale UCI datasets, including *Naval*, *Protein Structures*, *Video Transcoding (Memory, Time)*, *GPU kernel performance* and *YearPrediction*. We randomly split the datasets into 80% training, 10% validation, and 10% test. We used the validating set to select the hyperparameters and perform early stopping.

Both methods were trained for 80,000 iterations.[4] We used 1 hidden layer with 100 hidden units for all datasets. For the prior of the fBNNs, we use a GP with Neural Kernel Network (NKN) kernels as used in [32]. We note that the GP hyperparameters are fitted using minibatches of size 1000 with 10000 iterations. In each iteration, measurement sets consist of 500 training samples and 5 or 50 points from the sampling distribution c, tuned by validation performance. We ran each experiment 5 times, and report the mean and standard deviation in Table 3.

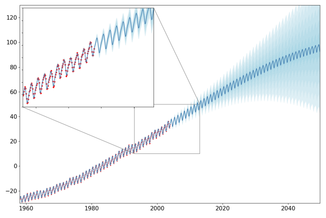
[4]We tune the learning rate from $[0.001, 0.01]$. We tune between not annealing the learning rate or annealing it by 0.1 at 40000 iterations. We evaluate the validating set in each epoch, and select the epoch for testing based on the validation performance.



(a) BBB



(b) FBNN



(c) GP

Figure 4: Prediction on Mauna datasets. We compare BBB, FBNN and GP.

E.3 Contextual Bandits

Table 4: Contextual bandits regret. Results are relative to the cumulative regret of the Uniform algorithm. Numbers after the algorithm correspond to the network size. We report the mean and standard derivation of the mean over 10 trials.

	M. Rank	M. Value	MushRoom	Statlog	Covertype	Financial	Jester	Adult	Census	Wheel
FBNN 1×50	5.875	46.0	21.38 ± 7.00	8.85 ± 4.55	47.16 ± 2.39	9.90 ± 2.40	75.55 ± 5.51	$\mathbf{88.43} \pm \mathbf{1.95}$	51.43 ± 2.34	65.05 ± 20.10
FBNN 2×50	7.125	47.0	24.57 ± 10.81	10.08 ± 5.66	49.04 ± 3.75	11.83 ± 2.95	73.85 ± 6.82	88.81 ± 3.29	50.09 ± 2.74	67.76 ± 25.74
FBNN 3×50	8.125	48.9	34.03 ± 13.95	7.73 ± 4.37	50.14 ± 3.13	14.14 ± 1.99	74.27 ± 6.54	89.68 ± 1.66	52.37 ± 3.03	68.60 ± 22.24
FBNN 1×500	**4.875**	**45.3**	21.90 ± 9.95	6.50 ± 2.97	47.45 ± 1.86	$\mathbf{7.83} \pm \mathbf{0.77}$	74.81 ± 5.57	89.03 ± 1.78	50.73 ± 1.53	63.77 ± 25.80
FBNN 2×500	**5.0**	**44.2**	23.93 ± 11.59	7.98 ± 3.08	46.00 ± 2.01	10.67 ± 3.52	$\mathbf{68.88} \pm \mathbf{7.09}$	89.70 ± 2.01	51.87 ± 2.38	54.57 ± 32.92
FBNN 3×500	**4.75**	**44.6**	19.07 ± 4.97	10.04 ± 5.09	$\mathbf{45.24} \pm \mathbf{2.11}$	11.48 ± 2.20	69.42 ± 7.56	90.01 ± 1.70	$\mathbf{49.73} \pm \mathbf{1.35}$	61.57 ± 21.73
MultitaskGP	5.875	46.5	20.75 ± 2.08	7.25 ± 1.80	48.37 ± 3.50	8.07 ± 1.13	76.99 ± 6.01	88.64 ± 3.20	57.86 ± 8.19	64.15 ± 27.08
BBB 1×50	11.5	56.6	24.41 ± 6.70	25.67 ± 3.46	58.25 ± 5.00	37.69 ± 15.34	75.39 ± 6.32	95.07 ± 1.57	63.96 ± 3.95	72.37 ± 16.87
BBB 1×500	13.375	68.1	26.41 ± 8.71	51.29 ± 11.27	83.91 ± 4.62	57.20 ± 7.19	78.94 ± 4.98	99.21 ± 0.79	92.73 ± 9.13	55.09 ± 13.82
BBAlphaDiv	16.0	87.4	61.00 ± 6.47	70.91 ± 10.22	97.63 ± 3.21	85.94 ± 4.88	87.80 ± 5.08	99.60 ± 1.06	100.41 ± 1.54	95.75 ± 12.31
ParamNoise	10.125	53.0	20.33 ± 13.12	13.27 ± 2.85	65.07 ± 3.47	17.63 ± 4.27	74.94 ± 7.24	95.90 ± 2.20	82.67 ± 3.86	54.38 ± 16.20
NeuralLinear	10.375	52.3	16.56 ± 11.60	13.96 ± 1.51	64.96 ± 2.54	18.57 ± 2.02	82.14 ± 3.64	96.87 ± 0.92	78.94 ± 1.87	46.26 ± 8.40
LinFullPost	9.25	nan	14.71 ± 0.67	19.24 ± 0.77	58.69 ± 1.17	10.69 ± 0.92	77.76 ± 5.67	95.00 ± 1.26	Nan $\pm$ Nan	$\mathbf{33.88} \pm \mathbf{15.15}$
Dropout	7.625	48.3	$\mathbf{12.53} \pm \mathbf{1.82}$	12.01 ± 6.11	48.95 ± 2.19	14.64 ± 3.95	71.38 ± 7.11	90.62 ± 2.21	58.53 ± 2.35	77.46 ± 27.58
RMS	8.875	53.0	15.29 ± 3.06	11.38 ± 5.63	58.96 ± 4.97	10.46 ± 1.61	72.09 ± 6.98	95.29 ± 1.50	85.29 ± 5.85	75.62 ± 30.43
BootRMS	7.5	51.9	18.05 ± 11.20	$\mathbf{6.13} \pm \mathbf{1.03}$	53.63 ± 2.15	8.69 ± 1.30	74.71 ± 6.00	94.18 ± 1.94	82.27 ± 1.84	77.80 ± 29.55
Uniform	16.75	100	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0

E.4 Time-series Extrapolation

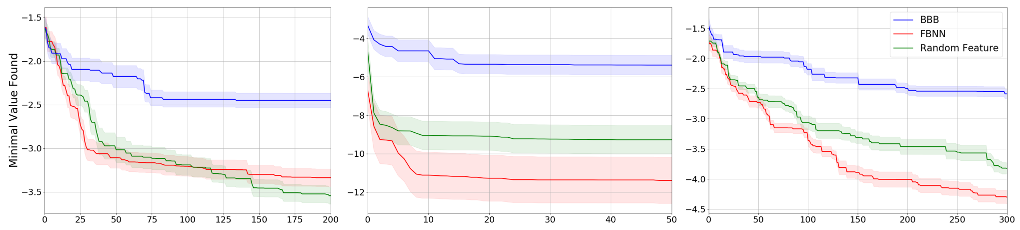
Besides the toy experiments, we would like to examine the extrapolation behavior of our method on real-world datasets. Here we consider a classic time-series prediction problem concerning the concentration of CO_2 in the atmosphere at the Mauna Loa Observatory, Hawaii [24]. The training data is given from 1958 to 2003 (with some missing values). Our goal is to model the prediction for an equally long period after 2003 (2004-2048). In Figure 4 we draw the prediction results given by BBB, fBNN, and GP. We used the same BNN architecture for BBB and fBNN: an ReLU network with 2 hidden layers, each with 100 units, and the input is a normalized year number augmented by its $\sin$ transformation, whose period is set to be one year. This special design allows both BBB and fBNN to fit the periodic structure more easily. Both models are trained for 30k iterations by the Adam optimizer, with learning rate 0.01 and batch size 20. For fBNN the prior is the same as the GP experiment, whose kernel is a combination of RBF, RBF×PER (period set to one year), and RQ kernels, as suggested in [24]. Measurement points include 20 training samples and 10 points sampled from $\mathbb{U}[1958, 2048]$, and we jointly train the prior GP hyperparameters with fBNN.

In Figure 4 we could see that the performance of fBNN closely match the exact prediction by GP. Both of them give visually good extrapolation results that successfully model the long-term trend, local variations, and periodic structures. In constrast, weight-space prior and inference (BBB) neither captures the right periodic structure, nor does it give meaningful uncertainty estimates.

E.5 Bayesian Optimization

In this section, we adopt Bayesian Optimization to explore the advantage of coherent posteriors. Specifically, we use Max Value Entropy Search (MES) [33], which trys to maximize the information gain about the minimum value $y^\star$,

$$\alpha_t(\mathbf{x}) = \mathbb{H}(p(y|D_t, \mathbf{x})) - \mathbb{H}(p(y|D_t, \mathbf{x}, y^\star)) \approx \frac{1}{K}\sum_{y^\star}[\frac{\gamma_{y^\star}(\mathbf{x})\phi(\gamma_{y^\star}(\mathbf{x}))}{\Psi(\gamma_{y^\star}(\mathbf{x}))} - \log(\Psi(\gamma_{y^\star}(\mathbf{x})))]$$



(a) RBF (b) ArcCosine (c) Matern12

Figure 5: Bayesian Optimization. We plot the minimal value found along iterations. We compare fBNN, BBB and Random Feature methods for three kinds of functions corresponding to RBF, ArcCosine and Matern12 GP kernels. We plot mean and 0.2 standard derivation as shaded areas.

Where ϕ and Ψ are probability density function and cumulative density function of a standard normal distribution, respectively. The $y^\star$ is randomly sampled from the posterior of function minimas and $\gamma_{y^\star}(\mathbf{x}) = \frac{\mu_t(\mathbf{x}) - y^\star}{\sigma_t(\mathbf{x})}$.

With a probabilistic model, we can compute or estimate the mean $\mu_t(\mathbf{x})$ and standard deviation $\sigma_t(\mathbf{x})$. However, to compute the MES acquisition function, samples $y^\star$ of function minimas are required as well, which leads to bigger difficulties. Typically when we model the data with a GP, we can get the posterior on a specific set of points but we don't have access to the extremes of the underlying function. In comparison, if the function posterior is represented parametrically, we can perform gradient decent easily and optimize for approximating function minimas.

We use 3-dim functions sampled from some Gaussian Process prior for bayesian optimization. Concretely, we experiment with samples from RBF, ArcCosine and Matern12 kernels. We compare three parametric approaches: fBNN, BBB and Random Feature [21]. For fBNN, we use the true kernel as functional priors. In contrast, ArcCosine and Matern12 kernels do not have simple explicit random feature expressions, therefore we use RBF random features for all three kernels. When looking for minimas, we sample 10 optimals $y^\star$. For each minima, we perform gradient descent along the parametric function posterior with 30 different starting points. We use 500 dimensions for random feature. We use network with 5×100 for fBNN. For BBB, we select the network within $1 \times 100, 3 \times 100$. Because of the similar issue in Figure 2, using larger networks won't help for BBB. We use batch size 30 for both fBNN and BBB. The measurement points contain 30 training points and 30 points uniformly sampled from the known input domain of functions. We train fBNN and BBB for 20000 iterations and anneal the coefficient of log likelihood term linearly from 0 to 1 for the first 10000 iterations. The results with 10 runs are shown in Figure 5.

As seen from Figure 5, fBNN and Random feature outperform BBB by a large margin on all three functions. We also observe fBNN performs slightly worse than random feature in terms of RBF priors. Because random feature method is exactly a GP with RBF kernel asymptotically, it sets a high standard for the parametric approaches. In contrast, fBNN outperforms random feature for both ArcCosine and Matern12 functions. This is because of the big discrepancy between such kernels and RBF random features. Because fBNN uses true kernels, it models the function structures better. This experiment highlights a key advantage of fBNN, that fBNN can learn parametric function posteriors for various priors.

E.6 Texture Extrapolation

In this section we perform a texture extrapolation experiment.

With a texture image, texture extrapolation tries to recover the missing region based on the observed region in some image. We use a image of 224×224, and the central 60×80 region is removed from the image. Treating each pixel as a data point, this forms 45376 training examples and 4800 test examples. Because the training size is too big to be afforded by a standard Gaussian Process. Previous GP methods [35, 32] investigate the additive structure of such patterns and take advantage of the Kronecker structure of the kernel matrix. However, such kronecker structure is only applicable to limited structures. This experiment adopts fBNN with a GAN-like generator to perform the recovery.

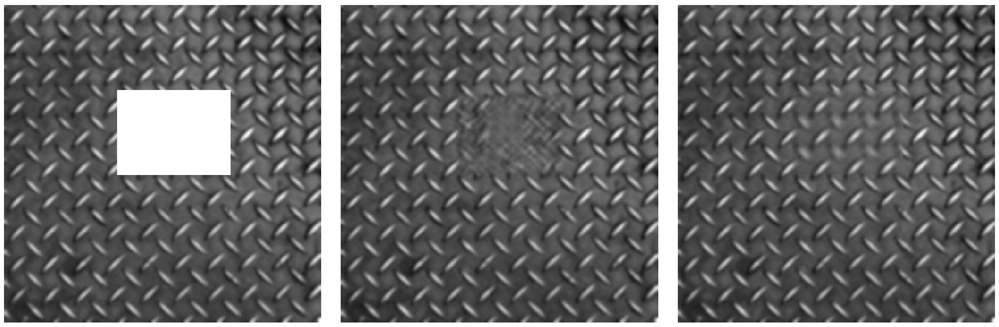

(a) Training (b) FBNN w.o. missing (c) FBNN w. missing

Figure 6: Texture Extrapolation for FBNN. For FBNN w.o. missing, the measurement points does not contain the missing region; for FBNN w. missing, the measurement points contain the missing region.

We use a GP prior with the kernel used in [32]. We firstly train the prior hyperparameters with GP regression with batch size 300. For the variational network, we use a GAN-like generator to generate the whole image starting from a 40 dimensional $\mathbb{U}[0, 1]$ noise. The network arranges as 40-(64*7*7)-(32*28*28)-(16*112*112)-(8*224*224)-(1*224*224). We use filter size 5 and leaky ReLU activation. Measurement points include 800 training points and 200 points sampled from the missing region (FBNN w. missing). We use learning rate 0.003 to train fBNN for 40000 iterations and anneal the kl coefficient from 0 to 1 linearly for the first 20000 iterations. We also evalute a setting where measurement points only contain training regions (FBNN w.o. missing). The results are shown in Figure 6.

As shown in Figure 6, fBNN correctly recovers the underlying structure and extrapolates to the missing region. However, if measurement points do not contain missing regions, fBNN fits training set perfectly but fails to extrapolate. This highlights the extrapolation effect of measurement points.