Meta-Learning to Perform Bayesian Inference in a single Forward Propagation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Currently, it is hard to reap the benefits of deep learning for Bayesian methods. We present *Prior-Data Fitted Networks (PFNs)*, a method that allows to employ large-scale machine learning techniques to approximate a large set of posteriors. The only requirement for PFNs is the ability to sample from a prior distribution over supervised learning tasks (or functions). The method repeatedly draws a task (or function) from this prior, draws a set of data points and their labels from it, masks one of the labels and learns to make probabilistic predictions for it based on the set-valued input of the rest of the data points. Presented with samples from a new supervised learning task as input, it can then make probabilistic predictions for arbitrary other data points in a single forward propagation, effectively having learned to perform Bayesian inference. We demonstrate that PFNs can near-perfectly mimic Gaussian processes and also enable efficient Bayesian inference for intractable problems, with over 200-fold speedups in multiple setups compared to current methods. We obtain strong results in such diverse areas as Gaussian process regression and Bayesian neural networks, demonstrating the generality of PFNs.

1 Introduction

In the last decade, supervised machine learning (ML) methods using deep learning architectures have made substantial progress on machine learning tasks where a large amount of training data is available (Vaswani et al., 2017; He et al., 2016; Krizhevsky et al., 2012). A very important problem in ML is thus to transfer these successes to smaller-scale setups with less data available. In this paper, we propose a way to build models that approximate posteriors with flexible and replaceable priors using deep learning models.

While the success of deep learning on large datasets can be attributed to the capacity of neural networks to approximate any function, there is still a need for encoding prior knowledge through model architecture (e.g. Convolutional Neural Networks (LeCun et al., 1989)) or regularizers (e.g. data augmentations (Hendrycks et al., 2019; Cubuk et al., 2020)). Otherwise, the no free lunch theorems show that there are no good methods to solve the class of prediction problems (Wolpert & Macready, 1997). In particular, this is true for ML applications to small datasets; thus, a large number of specialized algorithms have been developed for different tasks (LeCun et al., 1989; Kadra et al., 2021). Encoding prior information into such a model can, however, be challenging.

A well-defined way to bias a model is to use Bayesian inference. The foundation of Bayesian inference is an assumption on the distribution of the data to appear in a real world application. This assumption gives rise to a prior belief over the probability for the data to follow a particular model. One might, e.g., implement a prior, encoding the data as created by a neural network (Bayesian Neural Networks, (MacKay, 1992)), by a polynomial or the likelihood of a Gaussian mixture. In practice,

Submitted to the 5th Workshop on Meta-Learning at NeurIPS 2021, held virtually. Do not distribute.

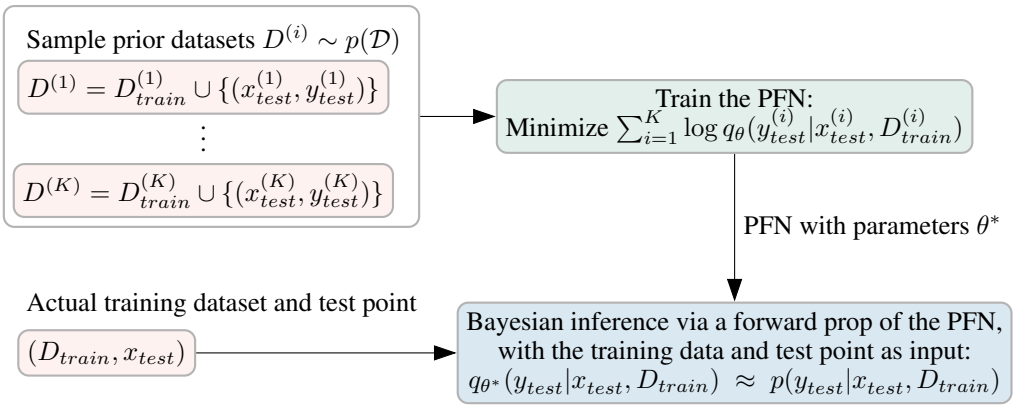

Figure 1: A visualization of Prior-Data Fitted Networks (PFNs). We sample (meta-train) datasets from a prior and fit a PFN on hold-out examples of these datasets. Given an actual (meta-test) dataset, we feed it and a test point to the PFN and obtain an approximation to Bayesian inference in a single forward propagation.

however, it is often very difficult or intractable to retrieve the Posterior Predictive Distribution (PPD) for a given prior exactly.

In this paper, we use large-scale neural networks to approximate Bayesian models. We outline our method of Prior-data Fitted Networks (PFNs) in Figure 1. We assume a given representative prior distribution over supervised learning tasks (or functions), which provides our inductive bias. To train a PFN, we repeatedly sample a meta-train task (or function) from this given prior, draw a set of data points and their labels from it, mask one of the labels and learn to make probabilistic predictions for it based on the set-valued input of the rest of the data points. Given an actual (meta-test) dataset, we feed it and a test point as inputs to the PFN and the network outputs its prediction for the test point, conditioned on the dataset. As we will demonstrate, this probabilistic prediction approximates exact Bayesian posterior prediction.

Our PFNs thus allow us to approximate the posterior predictive distribution for *any* prior from which we are able to sample data. This is a very weak requirement compared to the standard assumption that unnormalized posterior probabilities can be computed, which is made by currently common methods, such as Markov Chain Monte Carlo (MCMC) Neal (1996); Andrieu et al. (2003); Welling & Teh (2011); Hoffman et al. (2014) and Variational Inference (VI) (Jordan et al., 1999; Wainwright & Jordan, 2008; Hoffman et al., 2013). This weak requirement of being able to sample allows a simple approximation of a large set of priors, including priors that are very hard to approximate with currently available tools. We make the following contributions:

- We introduce the technique of Prior-data Fitted Networks (PFNs);
- We show that via meta learning on data sets sampled from a dataset prior, PFNs yield an approximation technique for the posterior predictive distribution (PPD);
- We present architectural changes to successfully use Transformers for PPD approximation, including a novel predictive distribution for regression tasks;
- We demonstrate the successful application of PFNs of in approximating the PPD for Gaussian processes and Bayesian neural networks.

2 Background on Bayesian Inference

Bayesian inference relies on priors for real world applications. By combining the knowledge from a general prior distribution with the data from a particular task, one obtains a posterior distribution, which can be used to predict outputs for new data points from the task at hand.

In this work, we are interested in supervised learning problems, i.e., we model the relationship between inputs x and outputs y. Generally, we consider predictions based on datasets of arbitrary size n, $\mathcal{D} = \{(x_i, y_i)|i = 1, \ldots, n\}$, where y_i is the *output* for x_i.

There are two kinds of Bayesian models commonly used:

i) In *parametric models*, e.g. Bayesian Neural Networks (BNNs), we model a distribution $p(t)$ over functions or tasks explicitly, where t is the random variable representing the task. In parametric models, t can be represented explicitly, e.g., as a vector, and either represents a conditional distribution $p(y|x,t)$ or a joint distribution $p(x,y|t)$ over the data. Thus, we can generate a prior over datasets $p(\mathcal{D}) = p(t)\prod_{i=1}^{n} p(x_i, y_i|t)$, where in the case of the conditional definition $p(x_i, y_i|t)$ factors into $p(y_i|x_i, t)$ and a a simple task-independent distribution $p(x_i)$.

ii) *Non-parametric models*, e.g., Gaussian Processes (GPs), can be viewed similarly to the above, but here we cannot explicitly instantiate t. Thus, we directly model, that means without the indirection over the task t, both the prior on data sets $p(\mathcal{D})$, as well as the PPD $p(y|x,\mathcal{D})$.

For both models, the prior distribution (PD) $p(\mathcal{D})$ is easily accessible. For all cases we consider, it is simple and cheap to sample from p(D), fulfilling the only requirement our method has. In practice, using Bayesian inference has the advantages that (i) it has a theoretical foundation that makes it valid in setups which the prior $p(t)$ fits, (ii) it can thus better account for the actual likelihood of different events; (iii) it is well calibrated and (iv) it is very interpretable as the prior describes the expectation of the model.

3 PPD Approximation with Prior-Data Fitting

Let us consider a parameterized model q_θ that can accept a set $\mathcal{D} = \{(x_i, y_i)|i = 1, \ldots, n\}$, as well as a query x as input, and which predicts a distribution over possible values of y for the query x. Many current neural network models can be used as such a model; in this paper, we use a variant of Transformers (Vaswani et al., 2017) as they are a reliable and powerful architecture.

We train this model by cross-entropy over samples drawn from the prior. Our proposed loss, the *Data Prior Negative Log-Likelihood (Prior-Data NLL)* ℓ is defined as

$$\ell_\theta = -\,\mathbb{E}_{D\cup\{x,y\}\sim p(\mathcal{D})}[\log q_\theta(y|x,\mathcal{D})], \tag{1}$$

where $D \cup \{x, y\}$ is simply a dataset of size $|D| + 1$ sampled from $p(\mathcal{D})$.

We minimize the Prior-Data NLL using a stochastic approximation of ℓ_θ, as we show in Algorithm 1. That means, we draw many samples of datasets from our prior and fit our model to predict a hold-out example correctly for these.

Algorithm 1: Training the model by Fitting Prior-Data

Result: A model q_θ that will approximate the PPD
Define data prior $p(\mathcal{D})$; initialize neural network q_θ;
while *not converged* **do**
  Sample $D \cup \{(x_i, y_i)\}_{i=1}^{m} \sim p(\mathcal{D})$;
  Compute stochastic loss approximation $\bar{\ell}_\theta = \sum_{i=1}^{m}(-\log q_\theta(y_i|x_i, D))$;
  Update model θ with gradient $\nabla_\theta \bar{\ell}_\theta$;
end

We can show that minimizing ℓ yields an approximation to the true PPD.

Theorem 1. *The proposed objective ℓ_θ is equal to the expectation of the cross-entropy $\mathbb{E}_{x,D}[H(p(\cdot|x,\mathcal{D}), q_\theta(\cdot|x,\mathcal{D}))]$ between the PPD and its approximation.*

Proof. The above can be shown with a simple derivation. We mark transformations for easy reading.

$$\ell_\theta = -\int_{D,x,y} p(x,y,D)\log q_\theta(y|x,\mathcal{D}) \tag{2}$$

$$= -\int_{D,x} p(x,D)\int_y p(y|x,D)\log q_\theta(y|x,D) \tag{3}$$

$$= \int_{\mathcal{D},x} p(x,D)\mathrm{H}(p(\cdot|x,D), q_\theta(\cdot|x,D)) \tag{4}$$

$$= \mathbb{E}_{x,\mathcal{D}}[\mathrm{H}(p(\cdot|x,D), q_\theta(\cdot|x,D))] \tag{5}$$

□

We see that by optimizing ℓ_θ, we optimize for similarity in terms of cross-entropy between the model and the PPD across the support of $p(x, \mathcal{D})$.

The proof above implies a minimization in terms of *KL*-Divergence, too.

Corollary 1.1. *The loss ℓ_θ is up to a constant, equal to the expected KL-Divergence of $p(\cdot|x, \mathcal{D})$ with $q_\theta(\cdot|x, \mathcal{D})$ over prior data $x, \mathcal{D}$.*

For the proof, please refer to Appendix A.

In the following, we consider the optimum of the given optimization problem $\theta^* = \arg\min_\theta \ell_\theta$. We consider the optimal result of our process for the case where q_θ describes a distribution family that can exactly approximate the posterior, that means there is a θ such that $q_\theta = p$.

Corollary 1.2. *If q_θ is a distribution family that can exactly approximate the posterior, we have $q_{\theta^*}(\cdot|x, \mathcal{D}) = p(\cdot|x, \mathcal{D})$ for all $x, \mathcal{D}$ with $p(x, \mathcal{D}) > 0$.*

Proof. We assume a θ with $q_\theta = p$ exists and we know that the cross-entropy is minimized for equal distributions; thus the optimum θ^* fulfills $q_{\theta^*}(\cdot|x, \mathcal{D}) = p(\cdot|x, \mathcal{D})$ for all $x, \mathcal{D}$ with $p(x, \mathcal{D}) > 0$. □

Interestingly, from our analysis, we can see that most meta learning setups that minimize the cross-entropy directly, like MAML (Finn et al., 2017a), are a form of PPD approximation as well. The prior is just created by humans, by drawing letters for the Omniglot dataset (Lake et al., 2015) for example, and they work only with a small sample from the prior, unlike in PF, where we continuously sample completely novel datasets $\mathcal{D}$ that are generated automatically.

4 Adapting the Transformer for Bayesian Inference

We use a standard transformer (Vaswani et al., 2017) with only a few changes to its architecture to fit our problem better. Transformers were developed with sequence modelling and prediction in mind, but in our application we want to map a set $D = \{(x_i, y_i)\}_{i=1}^K$ of data points (x_i, y_i) and a query x to a distribution $q_\theta(y|x, D)$ over y, which in many cases should be continuous. We propose an efficient architecture that allows batch-wise training and prediction, as well as a novel regression head for continuous y.

4.1 An Efficient Architecture

The first step in the architecture is to encode x and y; in all our experiments, we use simple linear layers for this, to project from the given dimension to the dimension of the transformer, but one could also use other application-specific encoding layers.Our setup should be equivariant to the ordering of the input dataset's examples. To achieve this, we remove positional encodings and feed (x, y) pairs together to the transformer as a sum of their encodings; see Appendix E for an ablation of the impact of the equivariance to the ordering.

In practice, we feed multiple query points to the transformer for efficiency reasons; these are the only inputs for which position matters, and we thus use the output at their positions as prediction for their corresponding PPD over y. Since different query points should not depend on each other, we use an attention mask that allows attention from every (x, y) pair to every other (x, y) pair and allows attention from all query points to the (x, y) pairs. The mask M we use can be characterized by the number of input examples m and the number of query points n, we write $M_{m,n}$. An example for such a matrix is

$$M_{3,2} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{6}$$

We predict the distribution of y from the output of the last time step. During training, we fix the total size $M = m + n$ and sample m from a distribution, in order to allow the Transformer to learn to

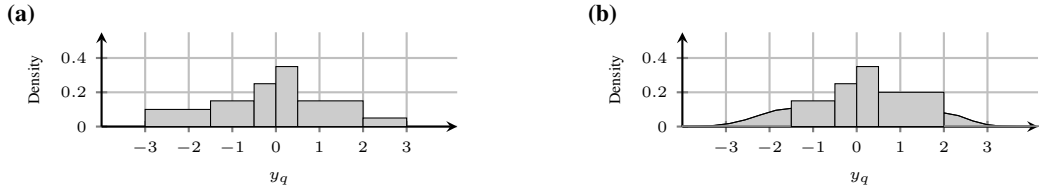

Figure 2: A visualisation of the Riemann probability distribution.

work with different dataset sizes. Since for smaller m we have more query points, we over-sample larger m, s.t. the number of query points seen during training for each m is equal during training. That is, we sample each $m \in \{0, \ldots, M-1\}$ with a weight $1/n = 1/(M-m)$.

4.2 Riemann Distribution

The modelling of continuous distributions is hard with neural networks. To yield strong performance in modelling PPDs, we developed a distribution that works particularly well with neural networks. Based on the knowledge that neural networks work well for classification and inspired by discretizations in distributional reinforcement learning (Bellemare et al., 2017), we created a discretized continuous distribution that we call *Riemann Distribution*, as it is based on a similar idea to approximation as Riemann integrals. PDFs of our distribution are bar plots, see Figure 2a.

The problem remaining is how to select the boundaries of the buckets. To make the problem balanced for the Transformer, we simply select buckets such that the likelihood of each bucket b has the same prior predictive probability $p(y \in b) = 1/|\mathbf{B}|, \forall b \in \mathbf{B}$. We approximate this by taking a large sample of y values from our prior and adjusting the buckets such that an equal number of these values fall in each bucket. The distributions described above are very powerful; e.g., in contrast to most parametric distributions they can also represent multimodal distributions. Their downside, however, is that they have finite support: the probability for y to be greater than the highest border is 0. This is a problem whenever one tries to model distributions that have unbounded support. A simple trick that worked well for us is to replace the last bar on each side with an appropriately scaled half-normal distribution. We visualize a resulting PDF in Figure 2b. For an exact mathematical definition of the distributions, we refer the reader to Appendix D. For an ablation of the Riemann distribution, see Appendix E.

We conclude this section by showing that, in terms of *KL*-Divergence, the finite Riemann distribution can approximate a large class of distributions, including distributions with infinite support like the normal distribution, up to arbitrary precision.

Theorem 2. *A Riemann distribution $q_{\mathbf{B}}$ with finite support, where $\mathbf{B}$ defines the buckets, can model any finite and almost everywhere continuous (Riemann-integrable) distribution up to arbitrary precision. That is, $\forall p \forall \epsilon \exists n, l, u. KL(q_{\mathbf{B}}, p) \leq \epsilon$.*

We refer the reader to Appendix B for the proof.

5 Posterior Approximation Studies

In our first set of experiments, we study the capability of PFNs to perform Bayesian inference for the tractable case of Gaussian Processes (GPs) with fixed hyperparameters (where we can compare to ground truth data; Section 5.1) and the intractable cases of GPs with unknown hyperparameters (Section 5.2) and Bayesian Neural Networks (BNNs; Section 5.3).

Based on Theorem 1, we present a novel way to evaluate methods approximating PPDs and Posteriors, the Prior-Data NLL itself. We simply take the negative log-likelihood of a method on data generated from the prior over datasets. We can do this, of course, for our models, but we can do this as easily with other well-known methods. We simply treat samples from the prior as training and test sets for these methods and evaluate the negative log-likelihood of the approximated distributions on the predicted outputs for the test sets. As we show in Corollary 1.1, the metric we evaluate is directly representative of the *KL* divergence between the true PPD and the approximation. There only is a constant difference. If a method only approximates the posterior and not the PPD directly, we approximate PPD with a large MC integration.

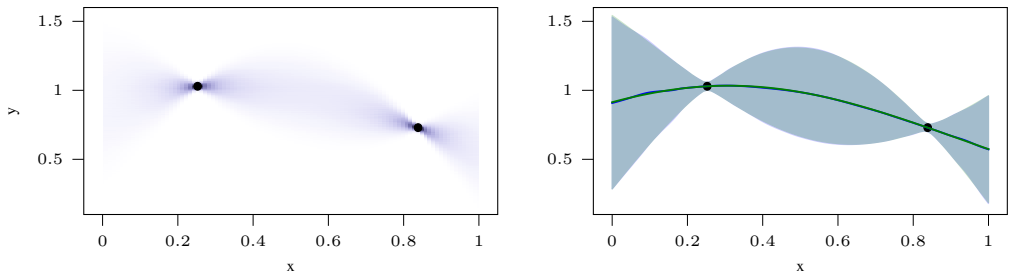

Figure 3: Left: the PFN's probability for query points x given two evaluations of the function, highlighting the binning it its predictions. Right: The PFN's mean (solid blue line) and 95% confidence interval (blue shaded region), alongside the exact GP mean posterior (solid green line) and 95% confidence interval (green shaded region). The two are near identical, with tiny differences. For additional comparisons, see Figure 6 in the appendix.

5.1 Gaussian Process approximation

We begin by approximating GPs as a study as to what the transformers are capable of; GPs with fixed hyperparameters are convenient for this purpose since the tractability of their PPD allows us to assess how close our approximation is. In this case, our meta-train datasets are functions f sampled from a GP with the given fixed hyperparameters. Since GPs describe conditional distributions given an input x, we sample $n+1$ inputs x_i uniformly at random from the unit-cube and evaluate $y_i = f(x_i)$. As usual, we then train our PFNs to predict y_{n+1} from the dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and the query point x_{n+1}. The trained PFN can then be used to compute the PPD for new datasets and arbitrary query points. Figure 3 visualizes the predictions of such a PFN for a small dataset with two data points and query points x ranging from 0 to 1. In Figure 3 (left), one can still see the very tiny boxes in which the Riemann distribution is discretized. In Figure 3 (right), we compare these predictions to those of a ground-truth GP with the given hyperparameters; both the mean function and the plotted 95% confidence interval are virtually indistinguishable from this ground truth. The model also learns on its own, completely automatically, to generate smooth distributions without any explicit knowledge of the positions of the bars in $\mathbb{R}$. In Figure 6 in the appendix we show more qualitative examples of the behavior of our approximation together with the exact posterior. Figure 4a shows that the qualitative results above generalize to much large datasets with multiple inputs. The GP Posterior achieves optimal performance by definition, and we can approximate it very closely, and better so with longer training. Additionally, there does not seem to be a trend towards worse approximation with larger datasets.

5.2 Approximating A GP with Hyper-Priors

A common practice in fields applying GPs in the real world is to define hyper-priors over the hyperparameters of GPs Rasmussen & Williams (2006); Snoek et al. (2012). Thus, the prior of these models considers a more diverse set of functions. The correlation between data points, the smoothness of the function and the scale of outputs can be variable. This allows the user to apply GPs to a larger set of tasks. The downside is that it is not possible to compute the PPD of the GP with hyper-priors exactly. There are two common practices to approximate it anyways and we evaluate against both. (i) Firstly, MLE-II is the most common, which is a simple special case of variational inference. Here, suitable hyperparameters are found by maximum a posteriori (MAP) estimation (e.g., Rasmussen & Williams (2006)). We use the fitting setup of BoTorch (Balandat et al., 2020) from which our Hyper-Priors for all models are inspired as well. (ii) Secondly, Markov Chain Monte Carlo (MCMC), a method to sample from any distribution for which one can calculate non-normalized probabilities, is also frequently applied to sample hyperparameters (e.g., Snoek et al. (2012)). Here we compare against doing so by the NUTS method (Hoffman et al., 2014), a state-of-the-art method that uses gradients to facilitate the Hamiltonian Monte Carlo algorithm. We plot the performance of the different methods in Figure 4b. Our model can approximate the PPD on the prior much closer than both of these methods, and its inference is more than $200\times$ faster than MLE-II and $500\times$ to $10\,000\times$ faster than NUTS.

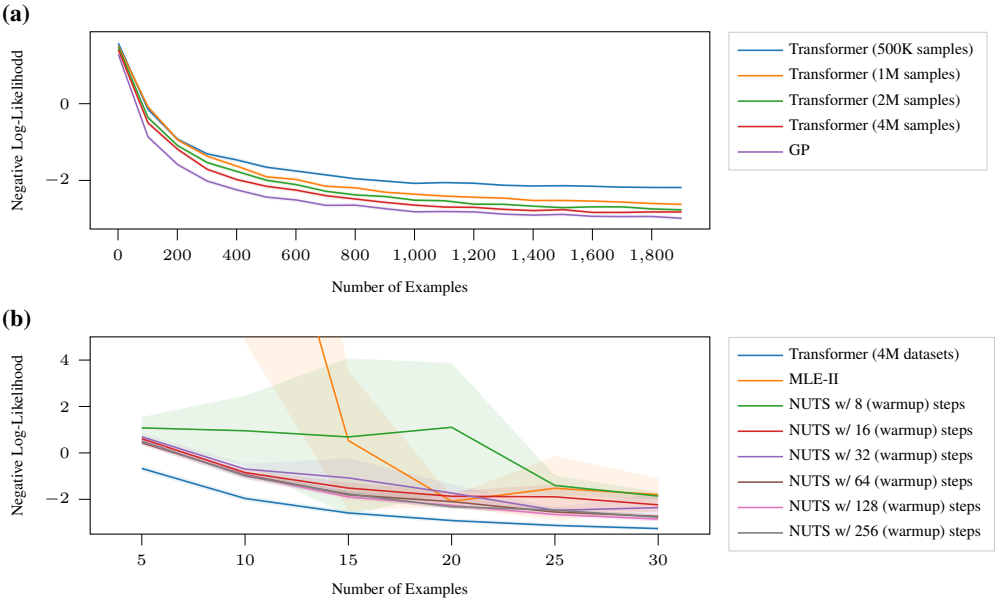

Figure 4: (a) NLL with an RBF-Kernel on different sizes of sets. Trained with Transformer. Hyperparameters: length-scale .6, noise 1e-4, out scalce 1., 5 inputs. (b) Plot with standard SingleTaskGP (with a different noise distribution: Gamma(.0001,1.0))

5.3 Approximating the BNN PPD

Bayesian Neural Networks (BNNs) provide an inherent ability to model uncertainty and strong regularization by inherent ensembling. Recently, approximation methods such as stochastic variational inference or stochastic gradient Markov Chain Monte Carlo methods allow for much faster convergence, which has sparked a great deal of interest in the area (Izmailov et al., 2021; Fortuin et al., 2021). However, these methods still converge too slowly for many practical applications.

In Figure 5, we show that our method can outperform today's default methods using a much smaller compute budget. The figure shows the approximation accuracy on synthetic datasets generated using the BNN prior that is used during inference by all solvers. To generate these synthetic datasets, we sampled random weights for a BNN BNN and used normally-distributed i.i.d. features X to obtain $y := BNN(X)$ with $\mathcal{D} := (X_{1:n}, y_{1:n})$.

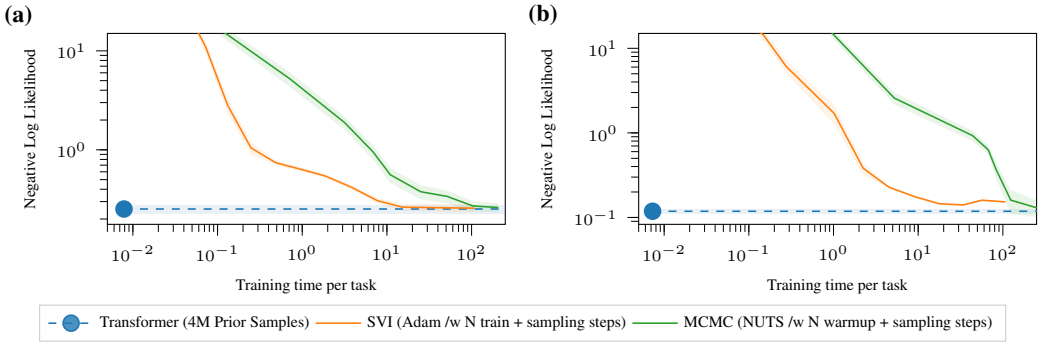

Figure 5: Training time spent per dataset at a given approximation accuracy. Negative Log Likelihood (NLL) is calculated as the mean over 100 tasks and 95% confidence intervals over tasks is indicated. Transformer and SVI are evaluated on a GPU while MCMC is evaluated on a CPU since it's speed surpasses GPU speed. (a) Bayesian Neural Network prior with 3 features, 2 layers and a hidden dimensionality of 5. (b) Bayesian Neural Network prior with 8 features, 2 layers and a hidden dimensionality of 64.

6 Conclusion & Future Work

We present a novel way to approximate the posterior predictive distribution very efficiently using deep neural networks. We show it's capability on a set of diverse tasks and priors. This opens the possibility for a multitude of ground breaking future work. We expect especially the following to be fruitful: (i) Work on finding novel priors that are cheap to sample for particular application areas. (ii) Work on architectures that are well-fit for this task, as we simply used a slight adaption of a current model.

References

Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended pac-bayes theory. In *International Conference on Machine Learning*, pp. 205–214. PMLR, 2018.

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

Yashas Annadani, Jonas Rothfuss, Alexandre Lacoste, Nino Scherrer, Anirudh Goyal, Yoshua Bengio, and Stefan Bauer. Variational causal networks: Approximate bayesian inference over causal structures. *arXiv preprint arXiv:2106.07635*, 2021.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL `http://arxiv.org/abs/1910.06403`.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145/2939672.2939785`.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7: 1–30, December 2006. ISSN 1532-4435.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pp. 1126–1135. Proceedings of Machine Learning Research, 2017a.

C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. In *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, volume 78, pp. 357–368. PMLR, 2017b.

Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Rätsch, Richard Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited, 2021.

P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909 [cs.LG]*, 2019. URL `https://arxiv.org/abs/1907.00909`. Accepted at AutoML Workshop at ICML 2019.

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

Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *International Conference on Learning Representations*, 2019.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(4):1303–1347, 2013. URL `http://jmlr.org/papers/v14/hoffman13a.html`.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. What are bayesian neural network posteriors really like? *CoRR*, abs/2104.14421, 2021. URL `https://arxiv.org/abs/2104.14421`.

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

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *arXiv preprint arXiv:1608.04471*, 2016.

Lars Lorch, Jonas Rothfuss, Bernhard Schölkopf, and Andreas Krause. Dibs: Differentiable bayesian structure learning. *arXiv preprint arXiv:2105.11839*, 2021.

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

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 2901–2907. AAAI Press, 2015. ISBN 0262511290.

R. Neal. *Bayesian Learning for Neural Networks*. 1996.

Judea Pearl. *Causality*. Cambridge University Press, 2 edition, 2009. doi: 10.1017/CBO9780511803161.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. Pacoh: Bayes-optimal meta-learning with pac-guarantees. In *International Conference on Machine Learning*, pp. 9116–9126. PMLR, 2021.

Bernhard Schölkopf, David W. Hogg, Dun Wang, Daniel Foreman-Mackey, Dominik Janzing, Carl-Johann Simon-Gabriel, and Jonas Peters. Modeling confounding by half-sibling regression. *Proceedings of the National Academy of Sciences*, 113(27):7391–7398, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1511656113. URL `https://www.pnas.org/content/113/27/7391`.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger (eds.), *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pp. 2960–2968, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Bjarni J. Vilhjálmsson, Jian Yang, Hilary Finucane, Alexander Gusev, Sara Lindström, Stephan Ripke, Giulio Genovese, Po-Ru Loh, Gaurav Bhatia, Ron Do, Tristan Hayeck, Hong-Hee Won, Biology Schizophrenia Working Group of the Psychiatric Genomics Consortium, the Discovery, Risk of Inherited Variants in Breast Cancer (DRIVE) study, Sekar Kathiresan, Michele Pato, Carlos Pato, Rulla Tamimi, Eli Stahl, Noah Zaitlen, Bogdan Pasaniuc, Mikkel H. Schierup, Philip De Jager, Nikolaos A. Patsopoulos, Steve McCarroll, Mark Daly, Shaun Purcell, Daniel Chasman, Benjamin Neale, Michael Goddard, Peter Visscher, Peter Kraft, Nick Patterson, and Alkes L. Price. Modeling linkage disequilibrium increases accuracy of polygenic risk scores. *bioRxiv*, 2015. doi: 10.1101/015859. URL `https://www.biorxiv.org/content/early/2015/03/04/015859`.

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

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)*, pp. 681–688. Citeseer, 2011.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7343–7353, 2018.

A Proof of Corollary 1.1

Proof. This can easily be seen by considering the result of Theorem 1

$$\mathbb{E}_{x,\mathcal{D}}[KL(p(\cdot|x,\mathcal{D},q_\theta(\cdot|x,\mathcal{D})] \tag{7}$$

$$= -\mathbb{E}_{x,\mathcal{D}}\left[\int_y p(y|x,\mathcal{D}\log\frac{q_\theta(y|x,\mathcal{D}}{p(y|x,\mathcal{D}}\right] \tag{8}$$

$$= -\mathbb{E}_{x,\mathcal{D}}\left[\int_y p(y|x,\mathcal{D}\log q_\theta(y|x,\mathcal{D}\right] + \mathbb{E}_{x,\mathcal{D}}\left[\int_y p(y|x,\mathcal{D}\log p(y|x,\mathcal{D}\right] \tag{9}$$

$$= \mathbb{E}_{x,\mathcal{D}}[\mathrm{H}(p(\cdot|x,\mathcal{D},q_\theta(\cdot|x,\mathcal{D}] - \mathbb{E}_{x,\mathcal{D}}[\mathrm{H}(p(\cdot|x,\mathcal{D})]] \tag{10}$$

$$= \ell_\theta + C, \tag{11}$$

where $C = -\mathbb{E}_{x,\mathcal{D}}[\mathrm{H}(\cdot|x,\mathcal{D}]]$. □

B Proof Theorem 2

Proof. Given an arbitrary distribution p, an arbitrary bound ϵ. For arbitrary minimum (maximum) bucket borders a (b), we consider the function $b_{\mathbf{B}}$ that maps each position y to its bucket $b \in \mathbf{B}$. We

define the upper bound of p for each bucket

$$u(y) = \sup_{y' \in b_s(y)} p(y). \tag{12}$$

Similarly, we define the lower bound

$$l(y) = \inf_{y' \in b_s(y)} p(y). \tag{13}$$

For each of these we consider their $\log$ form, too. That is $\hat{u}(y) = \log u(y) = \sup_{y' \in b_s(y)} \log p(y)$ and $\hat{l}(y) = \log l(y) = \inf_{y' \in b_s(y)} \log p(y)$. We can move the $\log$ inside the $\sup$ ($\inf$), since $\log$ is steadily increasing.

Integrals over $\hat{u}$ and $\hat{l}$ handily represent Darboux's upper and lower sums. We assumed that p is Riemann integrable. $\log p$ is also integrable, since the composition of continous functions with an almost everywhere continous function is almost everywhere continous and any finite value x has a finite value $\log x$. Thus, we can use $\forall \epsilon' \exists \mathbf{B}. \int_a^b \hat{u}(y) - \hat{l}(y) dy < \epsilon'$. We define $t = \sup_{y \in [a,b]} p(y)$, $T = \sup_y p(y)$ and the total probability mass of the interval $P = p(y \in [a, b])$. We use $\epsilon' = \log 1/P$ and get $\mathbf{B}$ s.t.

$$\int_a^b \hat{u}(y) - \hat{l}(y) dy < \log \frac{1}{P} \tag{14}$$

$$\Rightarrow \int_a^b t(\hat{u}(y) - \hat{l}(y)) dy < t \log 1/P \tag{15}$$

$$\Rightarrow \int_a^b t \log \frac{u(y)}{l(y)} dy < t \log 1/P \tag{16}$$

$$\Rightarrow \int_a^b u(y) \log \frac{u(y)}{l(y)} dy < t \log 1/P \tag{17}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{u(y)}{l(y)} dy < ct \log 1/P \tag{18}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < ct \log 1/P + \int_a^b cu(y) \log c \tag{19}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < ct \log 1/P + \log c \tag{20}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < ct \log 1/P + \log 1/P \tag{21}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < (ct + 1) \log 1/P \tag{22}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < -t \frac{\log P}{P} - \log P \tag{23}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < -T \frac{\log P}{P} - \log P. \tag{24}$$

Now we perform a case distinction. (i) If $t/P \leq 1$, we have

$$\int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < -T \frac{\log P}{P} - \log P \tag{25}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < 2 \log P. \tag{26}$$

Thus, by choosing a and b such that $p([a, b]) > e^{\frac{\epsilon}{2}}$ we have the wanted condition.

(ii) In the other case, namely $t/P > 1$, we have

$$\int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < -T \frac{\log P}{P} - \log P \tag{27}$$

$$\Rightarrow \int_a^b cu(y) \log \frac{cu(y)}{l(y)} dy < -2T \frac{\log P}{P}. \tag{28}$$

Here, we can use the Lambert W function to arrive at the final result, that $p([a,b]) > e^{-W_0(\frac{\epsilon}{2T})}$.

Thus, with bounds such that $p([a,b]) > \max(e^{\frac{\epsilon}{2}}, e^{-W_0(\frac{\epsilon}{2T})})$ there are buckets $\mathbf{B}$ such that $KL(q_{\mathbf{B}}, p) \leq \epsilon$).

□

C Qualitative Analysis

[h]

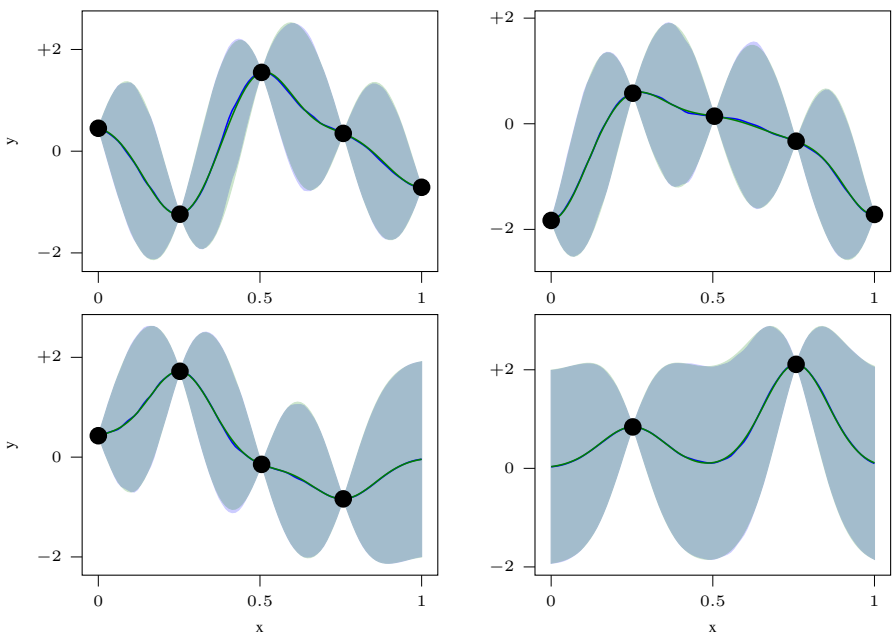

Figure 6: Green are the 95% intervals of the original GP posterior and blue are the corresponding transformer approximations. Lengthscale .1, Outscale 1., noise: 1e-4

D Riemann Distribution Definition

Given a set of of buckets $\mathbf{B}$, where each bucket $b \in \mathbf{B}$ describes an interval such that $\bigcup_{b \in \mathbf{B}} b = [a,b]$ and for any $b, b' \in \mathbf{B}$ we have $b \cap b' = \varnothing$. Additionally, we define the upper-bound (lower-bound) of the lowest (highest) bucket to be l (f). Additionally, we use a mapping $B(\cdot)$ that maps a $y \in \mathbb{R}$ to the unique bucket b with $y \in b$ and the width of each bucket to be defined as $w(b) = \max_{y \in b} y - \min_{y \in b} y$. The model gives us a probability for each bucket p_b.

In the finite case the distribution is defined as

$$p(y) = p_{b(y)}/w(b(y)). \tag{29}$$

We normalize with the width of the bucket since p_b describes the probability $p(y' \in b)$, but we are interested in the probability $p(y' = y)$.

For the case of infinite support given probabilities p_b for each bucket b, we have

$$f(y_q) = \begin{cases} 2 \cdot f_{\mathcal{N}}(y_q - l), & \text{if } y_q \geq l \\ 2 \cdot f_{\mathcal{N}}(f - y_q), & \text{if } y_q \geq f \\ p_{b(y_q)}/b(y_q), & \text{otherwise.} \end{cases} \tag{30}$$

Here, the Half-Normal distributions are scaled such that half the probability weight is in the first (last) bucket.

E Ablation

Above, we introduce a novel way of doing regression with neural networks that departs from the tradition of using normal distributions. Additionally, we remove the default positional embeddings from the Transformer. We shot the impact of both on the Transformer in Figure 7. We can see that, while the effect of the Riemann Distribution is very pronounced, the effect of making our architecture permutation invariant only slowly increases with sequence length.

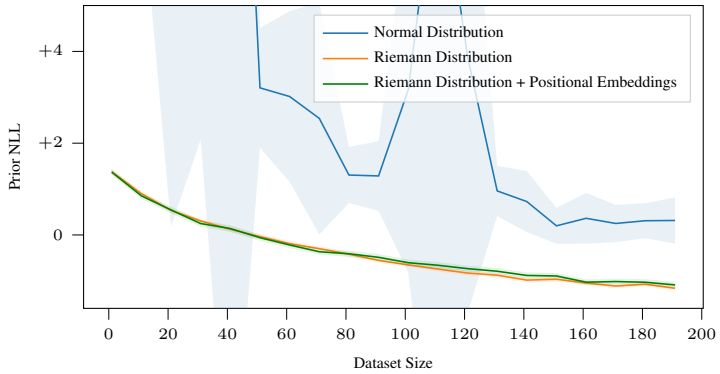

Figure 7: In this figure we show the impact of different architectural decisions on the performance of our method in fitting a Gaussian Process. We use the same Gaussian Process as in Section 5.1.