Efficient Bayesian Learning Curve Extrapolation using Prior-Data Fitted Networks

Steven Adriaensen
University of Freiburg
adriaens@cs.uni-freiburg.de

Herilalaina Rakotoarison
University of Freiburg
rakotoah@cs.uni-freiburg.de

Samuel Müller
University of Freiburg
muellesa@cs.uni-freiburg.de

Frank Hutter
University of Freiburg
Bosch Center for Artificial Intelligence
fh@cs.uni-freiburg.de

Abstract

Learning curve extrapolation aims to predict model performance in later epochs of a machine learning training, based on the performance in the first k epochs. In this work, we argue that, while the varying difficulty of extrapolating learning curves warrants a Bayesian approach, existing methods are (i) overly restrictive, and/or (ii) computationally expensive. We describe the first application of prior-data fitted neural networks (PFNs) in this context. PFNs use a transformer, pre-trained on data generated from a prior, to perform approximate Bayesian inference in a single forward pass. We present preliminary results, demonstrating that PFNs can more accurately approximate the posterior predictive distribution multiple orders of magnitude faster than MCMC, as well as obtain a lower average error predicting final accuracy obtained by real learning curve data from LCBench.

1 Introduction

Learning curve extrapolation [Domhan et al., 2015, Gargiani et al., 2019] aims to predict how much a machine learning model will improve with more training data, e.g., to determine how much more training data to collect [Cortes et al., 1993, Frey and Fisher, 1999, Kolachina et al., 2012], or to define an early-stopping criterion in online learning [Yao et al., 2007], and has recently been widely studied as a speed up technique of automated machine learning and hyperparameter optimization of deep neural networks [Swersky et al., 2014, Domhan et al., 2015, Klein et al., 2017, Wistuba et al., 2022]. Despite these efforts, learning curve extrapolation is yet to be widely adopted in practice, e.g., state-of-the-art multi-fidelity hyperparameter optimization techniques, like BOHB [Falkner et al., 2018], still rely on successive halving [Li et al., 2017], i.e., the crude heuristic that learning curves mostly do not cross. One reason is that, while many learning curves are well-behaved, some exhibit chaotic behavior and are intrinsically difficult to accurately predict [Choi et al., 2018]. In this setting, Bayesian approaches [Mockus et al., 1978], predicting the reliability of the extrapolation, show great potential. However, existing methods for Bayesian inference either (i) put strong restrictions on the prior, and are incapable of modeling the variable nature of learning curves, or (ii) are too computationally expensive, limiting their practical applicability.

In this work, we investigate the potential of learning curve extrapolation using prior-data fitted networks (PFNs), a meta-learned approximate Bayesian inference method proposed by Müller et al. [2022]. PFNs combine great flexibility with efficient and accurate approximation of the posterior predictive distribution (PPD) in a single forward pass of a transformer [Vaswani et al., 2017] trained

6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans.

on artificial data from the prior only. In particular, we compare PFNs to the MCMC approach of Domhan et al. [2015] in terms of the quality and cost of PPD approximation.

2 Methods

2.1 Bayesian inference using prior-fitted networks (PFNs)

Prior-data fitted networks (PFNs, Müller et al., 2022) are neural networks trained to perform approximate Bayesian prediction for supervised learning settings. That is, PFNs are trained to predict some output $y \in \mathbb{R}$ (in our case algorithm performance), conditioned on an input (in our case the epoch $t \in \{1, \dots, m\}$) and a training set of given input-output examples (in our case the learning curve up to some cutoff point T', $D = \{(t', y_{t'})\}_{t'=1}^{T'}$). The PFN is trained for this task with samples obtained from a pre-defined prior distribution p, i.e., $D \sim p(\mathcal{D})$, described in the next section. The loss function for training a PFN q_θ with parameters θ is the cross entropy $\ell_\theta = \mathbb{E}_{(t,y)\cup D\sim p(\mathcal{D})}[-\log q_\theta(y|t, D)]$ for predicting the hold-out example's label y. Müller et al. [2022] proved that minimizing this loss over many sampled tasks (D, t, y) directly coincides with minimizing the KL divergence between the PFN's predictions and the true PPD. In essence, the PFN meta-learns to perform approximate posterior inference on (meta-train) synthetic tasks sampled from the prior, and at inference time also does so for a (meta-test) real task.

2.2 Defining a prior over learning curves

Let $y_t \in [0, 1]$ represent the model accuracy at training step $t \in \{1, \dots, m\}$. Following Domhan et al. [2015], we model $\boldsymbol{y}$ as a linear combination of K basis growth curves f_k, each parameterized by $\boldsymbol{\theta}_k$, and i.i.d. additive Gaussian noise with variance σ^2, i.e.,

$$y_t \sim \mathcal{N}(f_{\text{comb}}(t|\boldsymbol{\xi}), \sigma^2) \quad \text{with} \quad f_{\text{comb}}(t|\xi) = \sum_{k=1}^{K} w_k \cdot f_k(t|\boldsymbol{\theta}_k),$$

where we assume our model parameters $\boldsymbol{\xi} = (w_1, \dots, w_K, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K, \sigma^2)$ to be random variables with prior $p(\boldsymbol{\xi})$. Here, Domhan et al. [2015] assumed an uninformative prior (i.e., $p(\boldsymbol{\xi}) \propto 1$), with exception of some hard constraints. We adopt a strictly more informative prior, because (i) the original prior puts almost all probability mass on parameterizations yielding invalid learning curves, e.g., $y_t \notin [0, 1]$; and (ii) we cannot practically sample from this prior having unbounded support, a requirement for PFNs.[1] Concretely, our prior distribution takes the form:

$$p(\boldsymbol{\xi}) \propto \left(\prod_{k=1}^{K} p(w_k) \cdot p(\boldsymbol{\theta}_k)\right) \cdot p(\sigma^2) \cdot \mathbb{1}(f_{\text{comb}}(1|\boldsymbol{\xi}) < f_{\text{comb}}(m|\boldsymbol{\xi})) \cdot \left(\prod_{t=1}^{m} \mathbb{1}(f_{\text{comb}}(t|\boldsymbol{\xi}) \in [0, 1])\right),$$

with $\log(\sigma^2) \sim \mathcal{N}(-4, 1)$ and $\log(w_k) \sim \mathcal{U}(\frac{\log(R)}{(1-R)K}, \frac{R\log(R)}{(1-R)K})$ to satisfy $\mathbb{E}[w_k] = \frac{1}{K}$ and impose $\frac{w_k}{w_{k'}} \leq R = 100$ as upper bound for the weight ratio. Finally, we limit ourselves to three basis curves ($K = 3$, see Table 1). These basis curves were chosen to capture a variety of growth trends and convergence behavior. See Figure 3 in Appendix A.2 for examples of curves sampled from this prior.

Table 1: Formulas of the three parametric basis curves and priors over their parameters

Reference name	Formula $f_k(x)$	Prior $p(\boldsymbol{\theta}_k)$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^\delta}$	$\alpha, \beta \sim \mathcal{U}(0, 1.5) \quad \kappa \sim \mathcal{U}(0, 1)$ $\log(\kappa) \sim \mathcal{N}(-2, 1) \quad \log(\delta) \sim \mathcal{N}(0, 0.5)$
ilog2	$c - \frac{a}{\log(x+1)}$	$c \sim \mathcal{U}(0, 1.5) \quad a \sim \mathcal{U}(0, 1)$
pow3	$c - ax^{-\alpha}$	$c \sim \mathcal{U}(0, 1.5) \quad a \sim \mathcal{U}(0, 1) \quad \log(\alpha) \sim \mathcal{N}(0, 2)$

[1]Note that the probability density of $p(y_t)$ is well-defined, a requirement for MCMC, but not for PFNs.

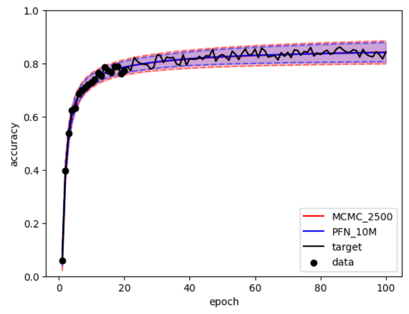

Figure 1: Extrapolation from 20 to 100 epochs

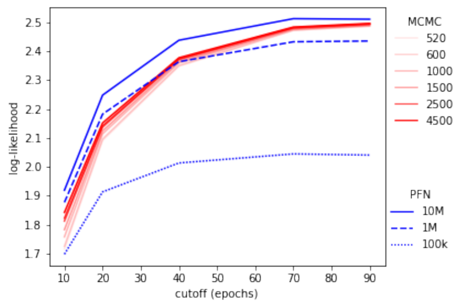

Figure 2: Quality of PPD approximations

3 Experiments

In our experiments, we aim to test the hypothesis that PFNs present a practical Bayesian approach to learning curve extrapolation. To this end, we compare against the MCMC approach of Domhan et al. [2015], using the same prior.[2] All experiments use the same PFN, a medium sized transformer (see Appendix A.1) pre-trained on 10M samples from the prior (`PFN_10M`). To show the effect of PFN training, we also include PFN checkpoints in our comparison that were trained on only 100K and 1M samples (`PFN_100k`, `PFN_1M`). For MCMC, we reuse the original code[3] with modification of the prior described in Section 2.2. We use the original hyperparameters (e.g., 500 burn-in, 100 walkers), but analyse the effect of varying chain length, originally 2 500 (`MCMC_2500`).

Quality of predictions: We first test how well PFNs and MCMC are able to approximate the posterior predictive distribution (PPD). Note that since both are (approximate) Bayesian inference methods, using the same prior, they should approximate the same target PPD, given an initial learning curve. To avoid artifacts due to out-of-distribution data, we use curves sampled from the prior in this experiment. We further vary the cutoff to analyse how both methods perform given more/less data and shorter/longer horizons. Figure 1, shows one of these curves, and inferences using `PFN_10M` and `MCMC_2500` given the data of the first 20 epochs (cutoff). We observe that both predicted mean and uncertainties (two-sided 90% ci) are indeed similar. More inference examples, with different cutoffs can be found in Figure 4, in Appendix A.2.

We can measure the quality of inference as the likelihood of hold-out prior data under the inferred PPD. This metric measures the cross-entropy to the true posterior, up to a constant, as $\mathbb{E}_{(t,y)\cup D\sim p(\mathcal{D})}[-\log q(y|t,D)] = \mathbb{E}_{t,D}[\mathrm{KL}(p(\cdot|t,D), q_\theta(\cdot|t,D))] + \mathbb{E}_{t,D}[\mathrm{H}(\cdot|t,D)]$ [Müller et al., 2022]. Figure 2 shows the average log-likelihood across PFN/MCMC inferences for different cutoffs of the same 1 000 curves taken from the prior. We observe that `PFN_10M` dominates all other approaches, for all cutoffs. We clearly see the benefit of training the PFN longer, since `PFN_100K` performed worst overall. The MCMC method is more competitive on larger cutoffs, and MCMC performance on shorter cutoffs can be somewhat improved by increasing the chain length.

Cost of predictions: Table 2 shows the log-likelihood, as well as the average computational cost (measured as wall-clock time on a single Intel(R) Xeon(R) Gold 6242 CPU) of the same 1 000 inferences, for each method and cutoff. We find that `PFN_10M` achieves a superior PPD approximation roughly 15 000 times faster than the most accurate (`MCMC_4500`) and 2 500 times faster than the fastest MCMC method (`MCMC_520`) in our comparison. For MCMC, the cost and quality of inference increases with growing chain length and cutoff. For PFNs, the inference time is the same for all checkpoints. Note that training the PFN on 10M samples from the prior took less than four hours (single CPU, single RTX2080 GPU) and is a one time cost across all of our experiments.

Predicting real learning curves: While evaluation on data from the prior gives us a controlled setting to analyse quality/cost of PPD approximation, performance on real-world learning curves is obviously essential for practical usefulness. Please note that we are interested in relative MCMC/PFN

[2]`https://github.com/automl/lc-pfns` (data and code generating figures in this section)
[3]`https://github.com/automl/pylearningcurvepredictor`

Table 2: Comparison of PFN_ and MCMC_ variants on prior dataset. See Table 4 for all results.

	Cutoff=10		Cutoff=20		Cutoff=40	
	LL	Time	LL	Time	LL	Time
MCMC_520	1.7 ± 0.85	100 ± 22	2.1 ± 0.99	106 ± 22	2.3 ± 0.91	107 ± 27
MCMC_1000	1.8 ± 0.86	162 ± 37	2.1 ± 0.98	172 ± 37	2.4 ± 0.92	174 ± 45
MCMC_2500	1.8 ± 0.91	350 ± 83	2.1 ± 1.0	369 ± 83	2.4 ± 0.93	374 ± 99
MCMC_4500	1.8 ± 0.94	598 ± 144	2.2 ± 1.0	626 ± 141	2.4 ± 0.93	633 ± 168
PFN_100k	1.7 ± 0.87	0.0393 ± 0.0007	1.9 ± 0.71	0.0393 ± 0.0008	2.0 ± 0.61	0.0393 ± 0.0007
PFN_1M	1.9 ± 0.95	0.0390 ± 0.0008	2.2 ± 0.92	0.0389 ± 0.0006	2.4 ± 0.86	0.0390 ± 0.0008
PFN_10M	1.9 ± 0.94	0.0400 ± 0.0008	2.2 ± 0.92	0.0401 ± 0.0009	2.4 ± 0.92	0.0400 ± 0.0008

Table 3: Comparison of PFN_10M and MCMC_2500 on LCBench. See Table 5 for all results.

	Cutoff=5		Cutoff=10		Cutoff=20	
	MSE	Time	MSE	Time	MSE	Time
MCMC_2500	2.18e-03 $\pm$ 0.01	85 ± 19	2.95e-04 $\pm$ 0.00	88 ± 27	8.01e-05 $\pm$ 0.00	95 ± 30
PFN_10M	1.26e-03 $\pm$ 0.00	0.0091 ± 0.0009	2.52e-04 $\pm$ 0.00	0.0091 ± 0.0008	5.24e-05 $\pm$ 0.00	0.0091 ± 0.0007

performance here, since absolute performance is strongly affected by the choice of prior. To this end, we applied PFN_10M and MCMC_2500 on validation accuracy curves from the LCBench[4] benchmark, which provides learning curve data for 2000 configurations of Auto-Pytorch [Zimmer et al., 2021] trained for 50 epochs ($m = 50$) on 35 OpenML [Vanschoren et al., 2014] datasets. Table 3 shows the mean squared error (MSE) on the expected *a posteriori* estimate and the log likelihood of the final accuracy across 350 curves (10 per dataset) for different cutoffs. We observe that for all cutoffs in $[5, 10, 20]$ (see Table 5 for all results), PFN_10M obtains a lower MSE than MCMC_2500 and a higher likelihood, while again being a factor of $10\,000\times$ faster. However, MCMC_2500 has lower MSE than PFN_10M for cutoff equal to 40.

4 Limitations and future research

While our preliminary results provide evidence of the potential of PFNs in the context of learning curve prediction, this initial study also has many limitations, to be addressed in future research.

Empirical evaluation: We focused on approximating the PPD, and compare against a specific MCMC approach. Future work should more extensively compare PFNs against prior-art, on more real-world datasets, considering additional performance metrics.

Definition of the prior: For a fair comparison, reusing the original code, our prior (Section 2.2) closely resembled the one from Domhan et al. [2015], inheriting its limitations. Future work should further refine this prior, e.g., to model divergence, correlated non-Gaussian noise, etc.

Meta-learning: In an AutoML or HPO context, we generally have more data available than a single partial learning curve. We may, e.g., have access to other curves on the same dataset, their hyperparameters, and/or curves of the same method on a different dataset, and meta-features. Previous work [E.g., Klein et al., 2017, Chandrashekaran and Lane, 2017] has explored exploiting such information for learning curve prediction and future work should investigate the potential of extending PFNs to this setting.

PFN limitations: PFNs are memory bound, limiting the size of the input sequence. While we are unlikely to hit this limit in our single curve setup, this may become an issue when incorporating meta-learning as described above. Furthermore, in this study we discovered another limitation of PFNs. PFNs, unlike other Bayesian methods, must learn the prior from data. This implies that the prior must be generative. Also, it suggests that high entropy priors may present challenges. Future research should investigate these limitations and how to overcome them.

[4]https://github.com/automl/LCBench

Acknowledgments and Disclosure of Funding

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. Robert Bosch GmbH is also acknowledged for financial support. We acknowledge funding by European Research Council (ERC) Consolidator Grant “Deep Learning 2.0” (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

Funded by
the European Union

References

Akshay Chandrashekaran and Ian R Lane. Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 477–492. Springer, 2017.

Daeyoung Choi, Hyunghun Cho, and Wonjong Rhee. On the difficulty of dnn hyperparameter optimization using learning curve prediction. In *TENCON 2018-2018 IEEE Region 10 Conference*, pages 0651–0656. IEEE, 2018.

Corinna Cortes, Lawrence D Jackel, Sara Solla, Vladimir Vapnik, and John Denker. Learning curves: Asymptotic values and rate of convergence. *Advances in neural information processing systems*, 6, 1993.

T. Domhan, J. Springenberg, and F. Hutter. Speeding up automatic Hyperparameter Optimization of deep neural networks by extrapolation of learning curves. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3460–3468, 2015.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient Hyperparameter Optimization at scale. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1437–1446. Proceedings of Machine Learning Research, 2018.

Lewis J Frey and Douglas H Fisher. Modeling decision tree performance with the power law. In *Seventh International Workshop on Artificial Intelligence and Statistics*. PMLR, 1999.

M Gargiani, A Klein, S Falkner, and F Hutter. Probabilistic rollouts for learning curve extrapolation across hyperparameter settings. In *6th ICML Workshop on Automated Machine Learning*, 2019.

A. Klein, S. Falkner, J. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

Prasanth Kolachina, Nicola Cancedda, Marc Dymetman, and Sriram Venkatapathy. Prediction of learning curves in machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22–30, 2012.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for Hyperparameter Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129), 1978.

S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, 2022. URL `https://openreview.net/forum?id=KSugKcbNf9`. Published online: `iclr.cc`.

K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. 15(2): 49–60, 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, Inc., 2017.

Martin Wistuba, Arlind Kadra, and Josif Grabocka. Dynamic and efficient gray-box hyperparameter optimization for deep learning. *arXiv preprint arXiv:2202.09774*, 2022.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:3079 – 3090, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We do not anticipate any negative societal impacts resulting directly from this work. Note that the transformers we use are very small ($\pm$ 50Mb storage) and relatively cheap to train (a couple of hours on a single CPU/GPU). In fact, if any, we expect a positive environmental impact, reducing the computational resources required to do (automated) machine / deep learning adopting efficient learning curve extrapolation methods.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We link a repository containing the data and code used to generate the plots in Section 3. This does not include the models or code used for training, which we will release upon publishing the full paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specified the most crucial aspects in Sections 2.2, 3, A.1. Note: We specify deviations from Domhan et al. [2015], for more details we refer to the original work and code.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report mean and standard deviations (sample sizes are given in text).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We used an existing benchmark LCBench and code of an existing MCMC method. We cite the scientific works that introduced these assets and provide an URL to the code/data.
   (b) Did you mention the license of the assets? [No] These are all public and open-source
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] No new assets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No person data was used
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Table 4: Detailed comparison of `PFN_` and `MCMC_` variants on prior dataset.

	Cutoff=10		Cutoff=20		Cutoff=40		Cutoff=70		Cutoff=90	
	LL	Time	LL	Time	LL	Time	LL	Time	LL	Time
`MCMC_520`	1.7 ± 0.85	100 ± 22	2.1 ± 0.99	106 ± 22	2.3 ± 0.91	107 ± 27	2.5 ± 0.93	108 ± 28	2.5 ± 0.95	108 ± 28
`MCMC_600`	1.7 ± 0.85	111 ± 25	2.1 ± 0.99	117 ± 25	2.4 ± 0.91	119 ± 30	2.5 ± 0.93	120 ± 31	2.5 ± 0.96	120 ± 32
`MCMC_1000`	1.8 ± 0.86	162 ± 37	2.1 ± 0.98	172 ± 37	2.4 ± 0.92	174 ± 45	2.5 ± 0.93	177 ± 48	2.5 ± 0.96	178 ± 48
`MCMC_1500`	1.8 ± 0.89	225 ± 53	2.1 ± 1.0	238 ± 53	2.4 ± 0.92	242 ± 63	2.5 ± 0.93	246 ± 67	2.5 ± 0.96	247 ± 68
`MCMC_2500`	1.8 ± 0.91	350 ± 83	2.1 ± 1.0	369 ± 83	2.4 ± 0.93	374 ± 99	2.5 ± 0.94	380 ± 106	2.5 ± 0.96	383 ± 107
`MCMC_3000`	1.8 ± 0.92	412 ± 98	2.1 ± 1.0	434 ± 97	2.4 ± 0.93	439 ± 116	2.5 ± 0.94	446 ± 124	2.5 ± 0.96	449 ± 126
`MCMC_4500`	1.8 ± 0.94	598 ± 144	2.2 ± 1.0	626 ± 141	2.4 ± 0.93	633 ± 168	2.5 ± 0.94	643 ± 180	2.5 ± 0.97	648 ± 182
`PFN_100k`	1.7 ± 0.87	0.0393 ± 0.0007	1.9 ± 0.71	0.0393 ± 0.0008	2.0 ± 0.61	0.0393 ± 0.0007	2.0 ± 0.6	0.0393 ± 0.0008	2.0 ± 0.61	0.0393 ± 0.0009
`PFN_1M`	1.9 ± 0.95	0.0390 ± 0.0008	2.2 ± 0.92	0.0389 ± 0.0006	2.4 ± 0.86	0.0390 ± 0.0008	2.4 ± 0.86	0.0389 ± 0.0006	2.4 ± 0.88	0.0390 ± 0.0007
`PFN_10M`	1.9 ± 0.94	0.0400 ± 0.0008	2.2 ± 0.92	0.0401 ± 0.0009	2.4 ± 0.92	0.0400 ± 0.0008	2.5 ± 0.95	0.0401 ± 0.0008	2.5 ± 0.97	0.0400 ± 0.0009

Table 5: Detailed comparison of `PFN_10M` and `MCMC_2500` on LCBench.

	Cutoff=5			Cutoff=10			Cutoff=20			Cutoff=40		
	MSE	LL	Time	MSE	LL	Time	MSE	LL	Time	MSE	LL	Time
`MCMC_2500`	2.18e-03 $\pm$ 0.01	1.8 ± 0.69	85 ± 19	2.95e-04 $\pm$ 0.00	2.7 ± 0.62	88 ± 27	8.01e-05 $\pm$ 0.00	3.4 ± 0.7	95 ± 30	1.88e-05 $\pm$ 0.00	3.9 ± 0.65	102 ± 32
`PFN_10M`	1.26e-03 $\pm$ 0.00	2.2 ± 0.59	0.0091 ± 0.0009	2.52e-04 $\pm$ 0.00	3.0 ± 0.6	0.0091 ± 0.0008	5.24e-05 $\pm$ 0.00	3.6 ± 0.69	0.0091 ± 0.0007	1.19e-04 $\pm$ 0.00	4.1 ± 0.41	0.0091 ± 0.0007

A Appendix

A.1 PFN architecture and hyperparameters

We stayed very close to the original PFN architecture and setup. We used a six-layer transformer with 4 heads, embedding size 512 and hidden size 1024. We used linear encoders for both the input and the output encoding. We trained on 10 million learning curves with 100 steps/epochs each. We used Adam (learning rate 0.0001, batch size 100) with cosine annealing [Loshchilov and Hutter, 2017] and a linear warmup of 2.5 million learning curves. We sampled the cutoff epoch, after which we predict, uniformly at random between 1 and 100. The training setup can be found in `https://github.com/automl/TransformersCanDoBayesianInference`.

A.2 Additional figures

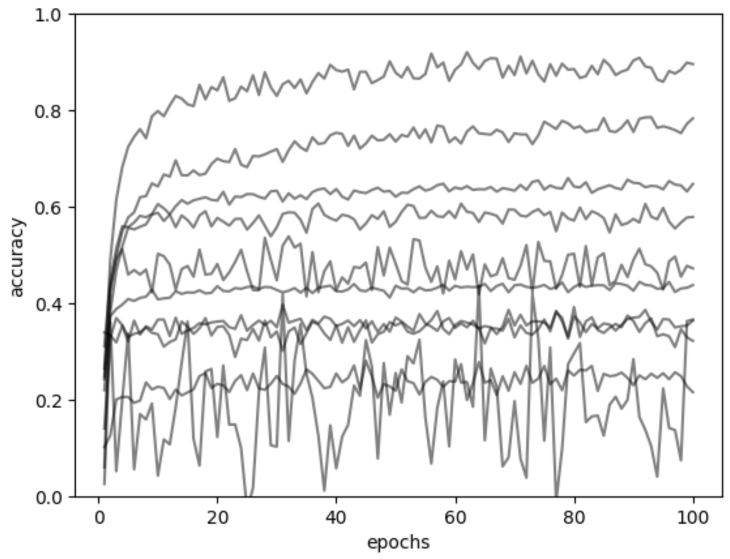

Figure 3: Sample of 10 curves taken i.i.d. from the prior (Section 2.2)

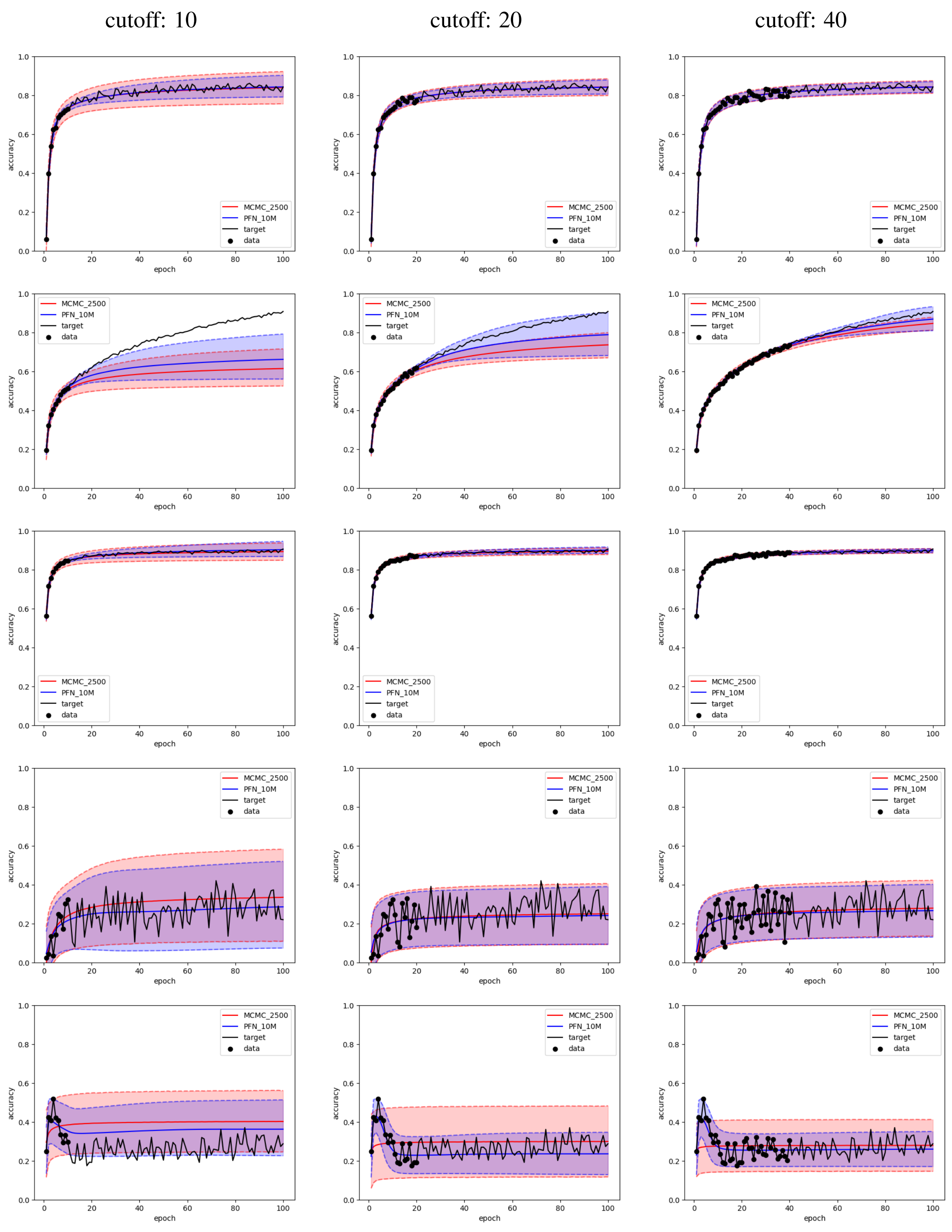

Figure 4: Extrapolations of different curves from the prior at 10, 20 and 40 cutoff