LEARNING CURVE PREDICTION WITH BAYESIAN NEURAL NETWORKS

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg & Frank Hutter
Department of Computer Science
University of Freiburg
{kleinaa,sfalkner,springj,fh}@cs.uni-freiburg.de

ABSTRACT

Different neural network architectures, hyperparameters and training protocols lead to different performances as a function of time. Human experts routinely inspect the resulting *learning curves* to quickly terminate runs with poor hyperparameter settings and thereby considerably speed up manual hyperparameter optimization. The same information can be exploited in automatic hyperparameter optimization by means of a probabilistic model of learning curves across hyperparameter settings. Here, we study the use of Bayesian neural networks for this purpose and improve their performance by a specialized learning curve layer.

1 INTRODUCTION

Deep learning has celebrated many successes, but its performance relies crucially on good hyperparameter settings. Bayesian optimization (e.g, Brochu et al. (2010); Snoek et al. (2012); Shahriari et al. (2016)) is a powerful method for optimizing the hyperparameters of deep neural networks (DNNs). However, its traditional treatment of DNN performance as a black box poses fundamental limitations for large and computationally expensive data sets, for which training a single model can take weeks. Human experts go beyond this blackbox notion in their manual tuning and exploit cheaper signals about which hyperparameter settings work well: they estimate overall performance based on runs using subsets of the data or initial short runs to weed out bad parameter settings; armed with these tricks, human experts can often outperform Bayesian optimization.

Recent extensions of Bayesian optimization and multi-armed bandits therefore also drop the limiting blackbox assumption and exploit the performance of short runs (Swersky et al., 2014; Domhan et al., 2015; Li et al., 2017), performance on small subsets of the data (Klein et al., 2017), and performance on other, related data sets (Swersky et al., 2013; Feurer et al., 2015).

While traditional solutions for scalable Bayesian optimization include approximate Gaussian process models (e.g., Hutter et al.; Swersky et al. (2014)) and random forests (Hutter et al., 2011), a recent trend is to exploit the flexible model class of neural networks for this purpose (Snoek et al., 2015; Springenberg et al., 2016). In this paper, we study this model class for the prediction of learning curves. Our contributions in this paper are:

1. We study how well Bayesian neural networks can fit learning curves for various architectures and hyperparameter settings, and how reliable their uncertainty estimates are.
2. Building on the parametric learning curve models of Domhan et al. (2015), we develop a specialized neural network architecture with a learning curve layer that improves learning curve predictions.
3. We compare different ways to generate Bayesian neural networks: probabilistic back propagation (Hernández-Lobato and Adams, 2015) and two different stochastic gradient based Markov Chain Monte Carlo (MCMC) methods – stochastic gradient Langevin dynamics (SGLD (Welling and Teh, 2011)) and stochastic gradient Hamiltonian MCMC (SGHMC (Chen et al., 2014)) – for standard Bayesian neural networks and our specialized architecture and show that SGHMC yields better uncertainty estimates.

4. We evaluate the predictive quality for both completely new learning curves and for extrapolating partially-observed curves, showing better performance than the parametric function approach by Domhan et al. (2015) at stages were learning curves have not yet converged.
5. We extend the recent multi-armed bandit strategy Hyperband (Li et al., 2017) by sampling using our model rather than uniformly at random, thereby enabling it to approach near-optimal configurations faster than traditional Bayesian optimization.

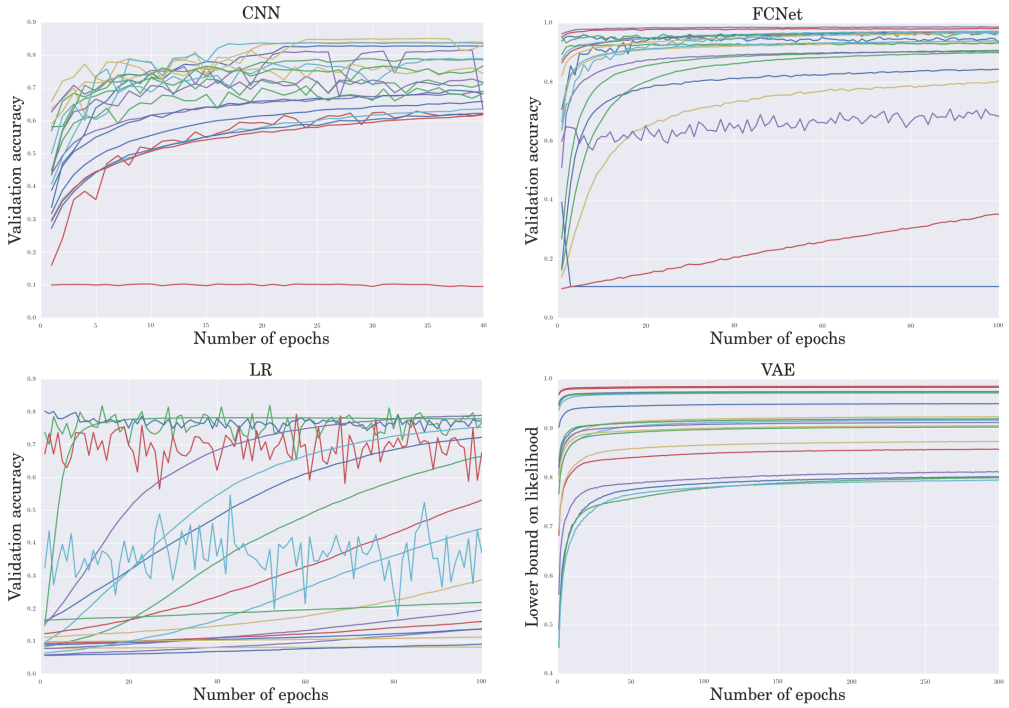

Figure 1: Example learning curves of random hyperparameter configurations of 4 different iterative machine learning methods: convolutional neural network (CNN), fully connected neural network (FCNet), logistic regression (LR), and variational auto-encoder (VAE). Although different configurations lead to different learning curves, they usually share some characteristics for a certain algorithm and dataset (but vary across these).

2 Probabilistic Prediction of Learning Curves

In this section, we describe a general framework to model learning curves of iterative machine learning methods. We first describe the approach by Domhan et al. (2015) which we will dub LC-Extrapolation from here on. Afterwards, we discuss a more general joint model across time steps and hyperparameter values that can exploit similarities between hyperparameter configurations and predict for unobserved learning curves. We also study the observation noise of different hyperparameter configurations and show how we can adapt our model to capture this noise.

2.1 Learning Curve Prediction with Basis Function

An intuitive model for learning curves proposed by Domhan et al. (2015) uses a set of k different parametric functions $\phi_i(\boldsymbol{\theta}_i, t) \in \{\phi_1(\boldsymbol{\theta}_1, t), ..., \phi_k(\boldsymbol{\theta}_k, t)\}$ to extrapolate learning curves $(y_1, \dots, y_n)$ from the first n time steps. Each parametric function ϕ_i depends on a time step $t \in [1, T]$ and on a parameter vector $\boldsymbol{\theta}_i$. The individual functions are combined into a single model by a weighted linear combination

$$\hat{f}(t|\boldsymbol{\Theta}, \boldsymbol{w}) = \sum_{i=1}^{k} w_i \phi_i(t, \boldsymbol{\theta}_i)\,, \tag{1}$$

where $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k)$ denotes the combined vector of all parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k$, and $\boldsymbol{w} = (w_1, \ldots, w_k)$ is the concatenated vector of the respective weights of each function. Assuming observation noise around the true but unknown value $f(t)$, i.e., assuming $y_t \sim \mathcal{N}(\hat{f}(t|\boldsymbol{\Theta}, \boldsymbol{w}), \sigma^2)$, Domhan et al. (2015) define a prior for all parameters $P(\boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2)$ and use a gradient-free MCMC method (Foreman-Mackey et al., 2013) to obtain S samples, $(\boldsymbol{\Theta}_1, \boldsymbol{w}_1, \sigma_1^2), \ldots, (\boldsymbol{\Theta}_S, \boldsymbol{w}_S, \sigma_S^2)$, from the posterior

$$P(\boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2 | y_1 \ldots, y_n) \propto P(y_1, \ldots, y_n | \boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2) P(\boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2) \tag{2}$$

using the likelihood

$$P(y_1, \ldots, y_n | \boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2) = \prod_{t=1}^{n} \mathcal{N}(y_t; \hat{f}(t|\boldsymbol{\Theta}, \boldsymbol{w}), \sigma^2)\,. \tag{3}$$

These samples then yield probabilistic extrapolations of the learning curve for future time steps m, with mean and variance predictions

$$\begin{aligned} \hat{y}_m &= \mathbb{E}[y_m | y_1, \ldots y_n] \approx \frac{1}{S} \sum_{s=1}^{S} \hat{f}(m|\boldsymbol{\Theta}_s, \boldsymbol{w}_s)\,, \text{ and} \\ \mathrm{var}(\hat{y}_m) &\approx \frac{1}{S} \sum_{s=1}^{S} (\hat{f}(m|\boldsymbol{\Theta}_s, \boldsymbol{w}_s) - \hat{y}_m)^2 + \sum_{s=1}^{S} \sigma_s^2\,. \end{aligned} \tag{4}$$

For our experiments, we use the original implementation by Domhan et al. (2015) with one modification: the original code included a term in the likelihood that enforced the prediction at $t = T$ to be strictly greater than the last value of that particular curve. This biases the estimation to never underestimate the accuracy at the asymptote. We found that in some of our benchmarks, this led to instabilities, especially with very noisy learning curves. Removing it cured that problem, and we did not observe any performance degradation on any of the other benchmarks.

The ability to include arbitrary parametric functions makes this model very flexible, and Domhan et al. (2015) used it successfully to terminate evaluations of poorly-performing hyperparameters early for various different architectures of neural networks (thereby speeding up Bayesian optimization by a factor of two). However, the model's major disadvantage is that it does not use previously evaluated hyperparameters at all and therefore can only make useful predictions after observing a substantial initial fraction of the learning curve.

2.2 Learning Curve Prediction with Bayesian Neural Networks

In practice, similar hyperparameter configurations often lead to similar learning curves, and modelling this dependence would allow predicting learning curves for new configurations without the need to observe their initial performance. Swersky et al. (2014) followed this approach based on an approximate Gaussian process model. Their *Freeze-Thaw* method showed promising results for finding good hyperparameters of iterative machine learning algorithms using learning curve prediction to allocate most resources for well-performing configurations during the optimization. The method introduces a special covariance function corresponding to exponentially decaying functions to model the learning curves. This results in an analytically tractable model, but using different functions to account for cases where the learning curves do not converge exponentially is not trivial.

Here, we formulate the problem using Bayesian neural networks. We aim to model the validation accuracy $g(\boldsymbol{x}, t)$ of a configuration $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ at time step $t \in (0, 1]$ based on noisy observations $y(\boldsymbol{x}, t) \sim \mathcal{N}(g(\boldsymbol{x}, t), \sigma^2)$. For each configuration $\boldsymbol{x}$ trained for $T_{\boldsymbol{x}}$ time steps, we obtain $T_{\boldsymbol{x}}$ data points for our model; denoting the combined data by $\mathcal{D} = \{(\boldsymbol{x}_1, t_1, y_{11})), (\boldsymbol{x}_1, t_2, y_{12}), \ldots, (\boldsymbol{x}_n, T_{x_n}, y_{nT_{x_n}})\}$ we can then write the joint probability of the data $\mathcal{D}$ and the network weights W as

$$P(\mathcal{D}, W) = P(W) P(\sigma^2) \prod_{i=1}^{|D|} \mathcal{N}(y_i; \hat{g}(\boldsymbol{x}_i, t_i | W), \sigma^2)\,. \tag{5}$$

where $\hat{g}(\boldsymbol{x}_i, t_i | W)$ is the prediction of a neural network. It is intractable to compute the posterior weight distribution $p(W|\mathcal{D})$, but we can use MCMC to sample it, in particular stochastic gradient

MCMC methods, such as SGLD (Welling and Teh, 2011) or SGHMC (Chen et al., 2014). Given M samples $W^1, \ldots, W^M$, we can then obtain the mean and variance of the predictive distribution $p(g|\boldsymbol{x}, t, \mathcal{D})$ as

$$\begin{aligned} \hat{\mu}(\boldsymbol{x}, t|\mathcal{D}) &= \frac{1}{M} \sum_{i=1}^{M} \hat{g}(\boldsymbol{x}, t|W^i) \,, \text{ and} \\ \widehat{\sigma^2}(\boldsymbol{x}, t|\mathcal{D}) &= \frac{1}{M} \sum_{i=1}^{M} \left(\hat{g}(\boldsymbol{x}, t|W^i) - \hat{\mu}(\boldsymbol{x}, t|\mathcal{D})\right)^2 , \end{aligned} \tag{6}$$

respectively. We will write these shorthand as $\hat{\mu}(\boldsymbol{x}, t)$ and $\widehat{\sigma^2}(\boldsymbol{x}, t)$. This is similar to Eqs. 4 and exactly the model that Springenberg et al. (2016) used for (blackbox) Bayesian optimization with Bayesian neural networks; the only difference is in the input to the model: here, there is a data point for every time step of the curve, whereas Springenberg et al. (2016) only used a single data point per curve (for its final time step).

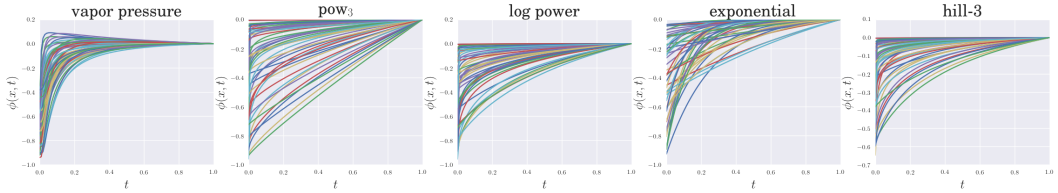

Figure 2: Example functions generated with our $k = 5$ basis functions (formulas for which are given in Appendix B). For each function, we drew 50 different parameters $\boldsymbol{\theta}_i$ uniformly at random in the output domain of the hidden layer(s) of our model. This illustrates the type of functions used to model the learning curves.

2.3 Heteroscedastic Noise of Hyperparameter Configuration

In the model described above, we assume homoscedastic noise across hyperparameter configurations. To evaluate how realistic this assumption is, we sampled 40 configurations of a fully connected network (see Section 3.1 for a more detailed description of how the data was generated and Table 2 for the list of hyperparameters) and evaluated each configuration $R = 10$ times with different pseudorandom number seeds (see the right panel of Figure 3 for some examples). Figure 3 (left) shows on the vertical axis the noise $\bar{\sigma}^2(\boldsymbol{x}, t) = \frac{1}{R} \sum_{r=1}^{R} (y(\boldsymbol{x}, t) - \bar{\mu}(\boldsymbol{x}, t))^2$ and on the horizontal axis the rank of each configuration based on their asymptotic sample mean performance $\bar{\mu}(\boldsymbol{x}, t = 1) = \frac{1}{R} \sum_{r=1}^{R} y(\boldsymbol{x}, t = 1)$.

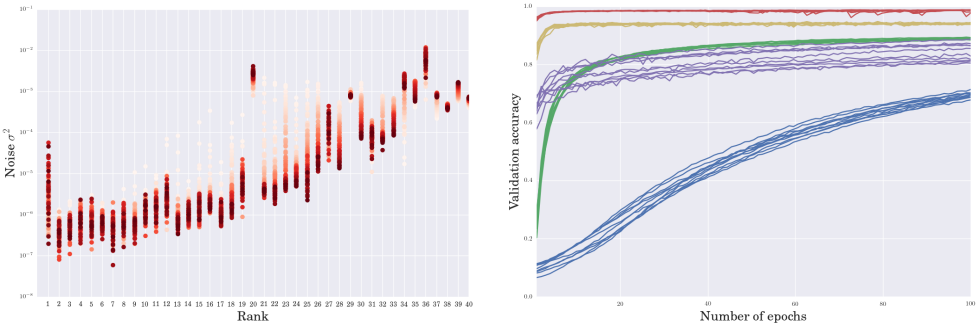

Figure 3: On the left, we plot the noise estimate of 40 different configurations on the FCNet benchmark sorted by their asymptotic mean performance at $t = 1$. The color indicates the time step t, darker meaning a larger value. On the right, we show the learning curves of 5 different configurations each evaluated 10 times (indicated with the same color) with a different seed.

Maybe not surprisingly, the noise seems to correlate with the asymptotic performance of a configuration. The fact that the noise between different configurations varies on different orders of magnitudes

suggests a heteroscedastic noise model to best describe this behavior. We incorporate this observation by making the noise dependent on the input data to allow to predict different noise levels for different hyperparameters. In principle, one could also model a t dependent noise, but we could not find the same trend across different datasets.

2.4 New Basis Function Layer for Learning Curve Prediction with Bayesian neural networks

We now combine Bayesian neural networks with the parametric functions to incorporate more knowledge about learning curves into the network itself. Instead of obtaining the parameters $\boldsymbol{\Theta}$ and $\boldsymbol{w}$ by sampling from the posterior, we use a Bayesian neural network to learn several mappings simultaneously:

1. $\hat{\mu}_\infty$: $\mathcal{X} \to \mathbb{R}$, the asymptotic value of the learning curve
2. $\boldsymbol{\Theta}$: $\mathcal{X} \to \mathbb{R}^K$, the parameters of a parametric function model (see Figure 2 for some example curves from our basis functions)
3. $\boldsymbol{w}$: $\mathcal{X} \to \mathbb{R}^k$, the corresponding weights for each function in the model
4. $\widehat{\sigma^2}$: $\mathcal{X} \to \mathbb{R}^+$, the observational noise for this hyperparameter configuration

With these quantities, we can compute the likelihood in (3) which allows training the network.

Phrased differently, we use a neural network to predict the model parameters $\boldsymbol{\Theta}$ and weights $\boldsymbol{w}$ of our parametric functions, yielding the following form for our network's mean predictions:

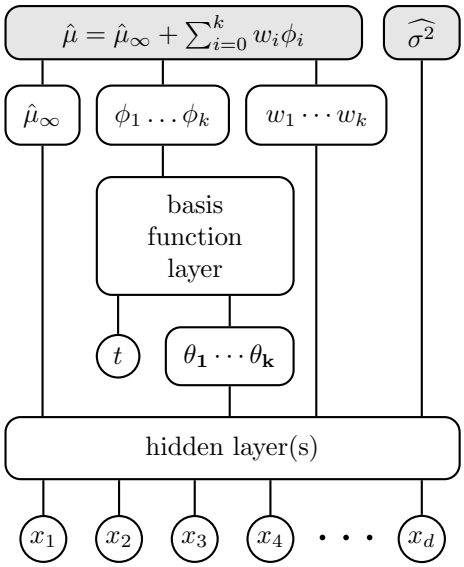

Figure 4: Our neural network architecture to model learning curves. A common hidden layer is used to simultaneously model $\hat{\mu}_\infty$, the parameters $\boldsymbol{\Theta}$ of the basis functions, their respective weights $\boldsymbol{w}$, and the noise $\widehat{\sigma^2}$.

$$\hat{g}(\boldsymbol{x}_i, t_i | W) = \hat{f}(t_i | \boldsymbol{\Theta}(\boldsymbol{x}_i, W), \boldsymbol{w}(\boldsymbol{x}_i, W)). \tag{7}$$

A schematic of this is shown in Fig. 4. For training, we will use the same MCMC methods, namely SGLD and SGHMC mentioned above.

3 Experiments

We now empirically evaluate the predictive performance of Bayesian neural networks, with and without our special learning curve layer. For both networks, we used a 3-layer architecture with tanh activations and 64 units per layer. We also evaluate two different sampling methods for both types of networks: stochastic gradient Langevin dynamics (SGLD) and stochastic gradient Hamiltonian MCMC (SGHMC), following the approach of Springenberg et al. (2016) to automatically adapt the noise estimate and the preconditioning of the gradients.

As baselines, we compare to other approaches suitable for this task. Besides the aforementioned work by Domhan et al. (2015), we also compare against random forests Breimann (2001) with empirical variance estimates (Hutter et al., 2014), a Gaussian process (GP) using the learning curve kernel from Swersky et al. (2014), another Bayesian neural network technique called probabilistic back propagation (PBP) (Hernández-Lobato and Adams, 2015), and the simple heuristic of using the last seen value (LastSeenValue) of each learning curve for extrapolation. The last model has been successfully used by Li et al. (2017) despite its simplicity.

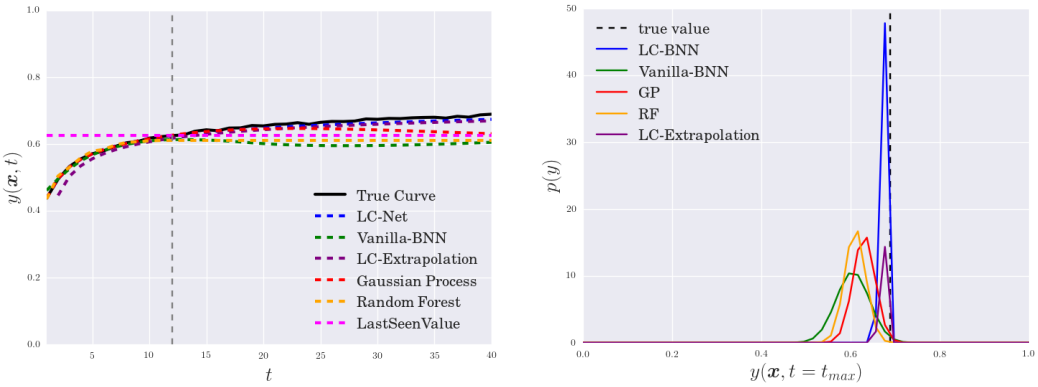

Figure 5: Qualitative comparison of the different models. The left panel shows the mean predictions of different methods on the CNN benchmark. All models observed the validation error of the first 12 epochs of the true learning curve (black). On the right, the posterior distributions over the value at 40 epochs is plotted.

3.1 DATASETS

For our empirical evaluation, we generated the following four datasets of learning curves, in each case sampling hyperparameter configurations at random from the hyperparameter spaces detailed in Table 2 in the appendix (see also Section D for some characteristic of these datasets):

- **CNN**: We sampled 256 configurations of 5 different hyperparameters of a 3-layer convolutional neural network (CNN) and trained each of them for 40 epochs on the CIFAR10 (Krizhevsky, 2009) benchmark.
- **FCNet**: We sampled 4096 configurations of 10 hyperparameters of a 2-layer feed forward neural network (FCNet) on MNIST (LeCun et al., 2001), with batch normalization, dropout and ReLU activation functions, annealing the learning rate over time according to a power function. We trained the neural network for 100 epochs.
- **LR**: We sampled 1024 configurations of the 4 hyperparameters of *logistic regression* (LR) and also trained it for 100 epochs on MNIST.
- **VAE**: We sampled 1024 configuration of the 4 hyperparameters of a *variational auto-encoder* (VAE) (Kingma and Welling, 2014). We trained the VAE on MNIST, optimizing the approximation of the lower bound for 300 epochs.

3.2 PREDICTING ASYMPTOTIC VALUES OF PARTIALLY OBSERVED CURVES

We first study the problem of predicting the asymptotic values of partially-observed learning curves tackled by Domhan et al. (2015). The LC-Extrapolation method by Domhan et al. (2015), the GP, and the last seen value work on individual learning curves and do not allow to model performance across hyperparameter configurations. Thus, we trained them separately on individual partial learning curves. The other models, including our Bayesian neural networks, on the other hand, can use training data from different hyperparameter configurations. Here, we used training data with the same number of epochs for every partial learning curve.[1]

The left panel of Figure 5 visualizes the extrapolation task, showing a learning curve from the CNN dataset and the prediction of the various models trained only using the first 12 of 40 epochs of the learning curve. The right panel shows the corresponding predictive distributions obtained with these models. LastSeenValue does not yield a distribution and uncertainties are not defined.

For a more quantitative evaluation, we used all models to predict the asymptotic value of all learning curves, evaluating predictions based on observing between 10% and 90% of the learning curves.

[1]We note that when used inside Bayesian optimization, we would have access to a mix of fully-converged and partially-converged learning curves as training data, and could therefore expect better extrapolation performance.

Method	CNN		FCNet		LR		VAE	
	MSE $\cdot 10^2$	ALL	MSE$\cdot 10^2$	ALL	MSE$\cdot 10^2$	ALL	MSE $\cdot 10^4$	ALL
SGLD	1.8 ± 0.9	0.60 ± 0.25	4.5 ± 0.6	0.13 ± 0.05	1.1 ± 0.3	0.77 ± 0.08	4.7 ± 1.7	2.16 ± 0.07
SGLD-LC	1.4 ± 0.9	1.09 ± 0.22	3.3 ± 0.7	0.54 ± 0.06	1.3 ± 0.7	0.94 ± 0.09	3.1 ± 1.1	1.55 ± 0.01
SGHMC	0.8 ± 0.6	0.96 ± 0.15	1.9 ± 0.3	0.50 ± 0.04	0.5 ± 0.2	1.08 ± 0.05	3.6 ± 1.5	2.34 ± 0.07
SGHMC-LC	0.7 ± 0.6	1.15 ± 0.34	2.0 ± 0.3	0.80 ± 0.05	0.5 ± 0.2	1.17 ± 0.08	1.9 ± 1.1	1.39 ± 0.65

Table 1: In each column we report the mean squared error (MSE) and the average log-likelihood (ALL) of the 16 fold CV learning curve prediction for a neural network without learning curve prediction layer (SGLD and SGHMC) and with our new layer (SGLD-LC and SGHMC-LC).

Figure 6 shows the mean squared error between the true asymptotic value and the models' predictions (left) and the average log-likelihood of the true value given each model as a function of how much of the learning curves has been observed. Note that we removed some methods from the plots to avoid clutter; the same plots with all methods can be found in the supplementary material. We notice several patterns:

1. Throughout, our specialized network architecture performs better than the standard Bayesian neural networks, and SGHMC outperformed SGLD (see Appendix C).
2. PBP shows mixed results for the extrapolated mean, and does not provide reliable uncertainties (also see Appendix C). We hypothesize that this may be due to its linear activation functions.
3. If no uncertainties are required, LastSeenValue is hard to beat. This is because many configurations approach their final performance quite quickly.
4. The GP mean predictions are very competitive, but the average log-likelihood indicates overconfidence, especially for short learning curves. We assume that the prior assumption of an exponential function is not flexible enough in practice and that after observing some data points the Gaussian process becomes too certain of its predictions.
5. The random forest's mean predictions almost match the quality of the GP. The uncertainty estimates are better, but still too aggressive when only a small fraction of the learning curve was observed. Random forests do not extrapolate and similar to LastSeenValue also achieve a very good mean prediction if the learning curve has almost converged.
6. Local models for single curves clearly outperform global models for almost complete learning curves. Global models, like our BNN approach, on the other hand, are trained on many configurations and need to generalize across these, yielding somewhat worse performance for this (arguably easy) task.[2]

3.3 Predicting unobserved learning curves

As mentioned before, training a joint model across hyperparameters and time steps allows us to make predictions for completely unobserved learning curves of new configurations. To estimate how well Bayesian neural networks perform in this task, we used the datasets from Section 3.1 and split all of them into 16 folds, allowing us to perform cross-validation of the predictive performance. For each fold, we trained all models on the full learning curves in the training set and let them predict the held-out learning curves.

Table 1 shows the mean squared error and the average log-likelihood (both computed for all points in each learning curve) across the 16 folds. We make two observations: firstly, both neural network architectures lead to a reasonable mean squared error and average log-likelihood, for both SGLD and SGHMC. Except on the VAE dataset our learning curve layer seems to improve mean squared error and average log likelihood. Secondly, SGHMC performed better than SGLD, with the latter resulting in predictions with too small variances. Figure 7 visualizes the results for our CNN dataset in more detail, showing that true and predicted accuracies correlate quite strongly.

[2]In the future, we aim to improve our BNN architecture for this case of partially-observed learning curves by also giving the network access to the partial learning curve it should extrapolate.

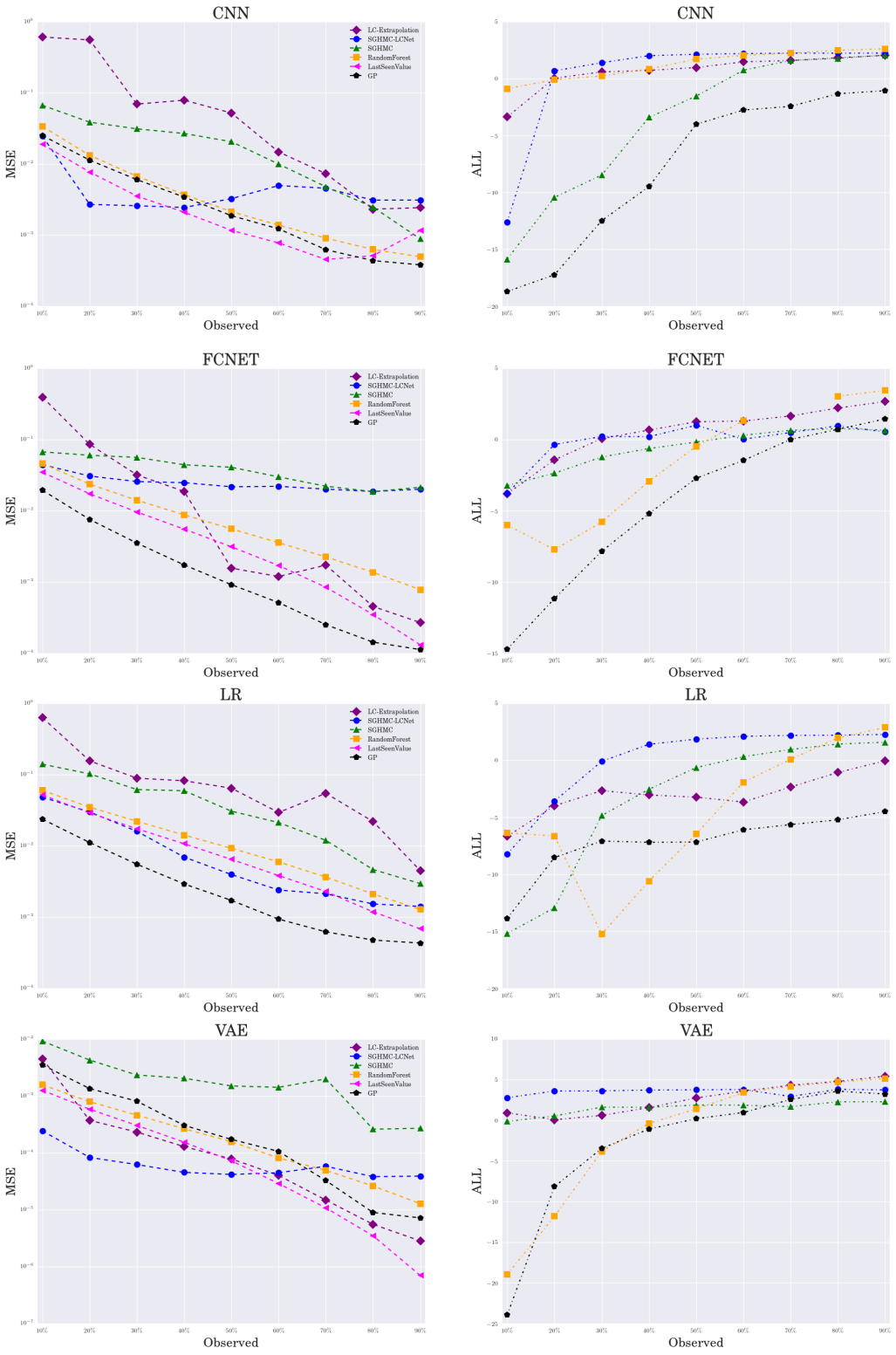

Figure 6: Assessment of the predictive quality based on partially observed learning curves on all 4 benchmarks. The panels on the left show the mean squared error of the predicted asymptotic value (y-axis) after observing all learning curves up to a given fraction of maximum number of epochs (x-axis). The panels on the right show the average log-likelihood based on the predictive mean and variance of the asymptotic value. Note that LastSeenValue does not provide a predictive variance.

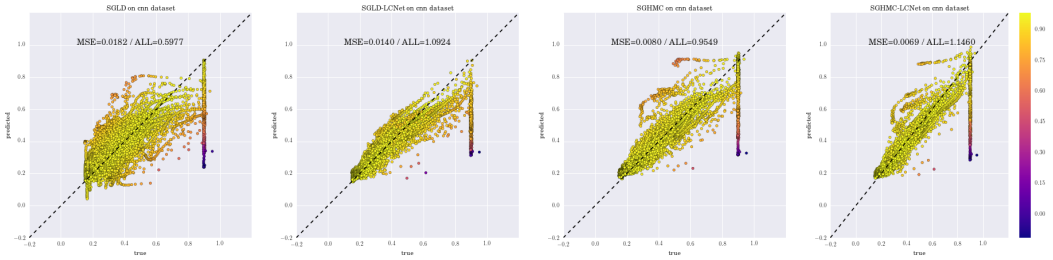

Figure 7: On the horizontal axis, we plot the true value and on the vertical axis the predicted values. Each point is colored by its log-likelihood (the brighter the higher). Our learning curve BNN trained with SGHMC leads to the best mean predictions and assigns the highest likelihood to the test points.

3.4 Model-based Hyperband

In this section, we show how our model can be used to improve hyperparameter optimization of iterative machine learning algorithms. For this, we extended the multi-armed bandit strategy Hyperband (Li et al., 2017), which in each iteration i first samples N_i hyperparameter configurations $C = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_i}\}$ and then uses successive halving (Jamieson and Talwalkar, 2016) to iteratively discard poorly-performing configurations from C. While the original Hyperband method samples configurations C from a uniform distribution $\mathcal{U}$ over hyperparameter configurations, our extension instead samples them based on our model, with all other parts remaining unchanged. More precisely, we sample $M_i >> N_i$ configurations uniformly, $\hat{C} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M_i}\} \sim \mathcal{U}$, and pick the N_i configurations with the highest predicted asymptotic mean $a(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}, t = 1)$ or upper confidence value $a(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}, t = 1) + \hat{\sigma}(\boldsymbol{x}, t = 1)$.

For a thorough empirical evaluation and to reduce computational requirements we reused the data from Section 3.1 to construct a surrogate benchmark based on a random forest (see Appendix E for more details). After each iteration we report the final performance of the best observed configuration so far, along with the wall clock time that would have been needed for optimizing the true objective function. Reporting the (predicted) wall clock time (rather than the number of iterations) also takes our optimizer's overhead into account. This is important since one may worry that this overhead may be substantial due to our use of a model; however, as we show it does not harm performance.

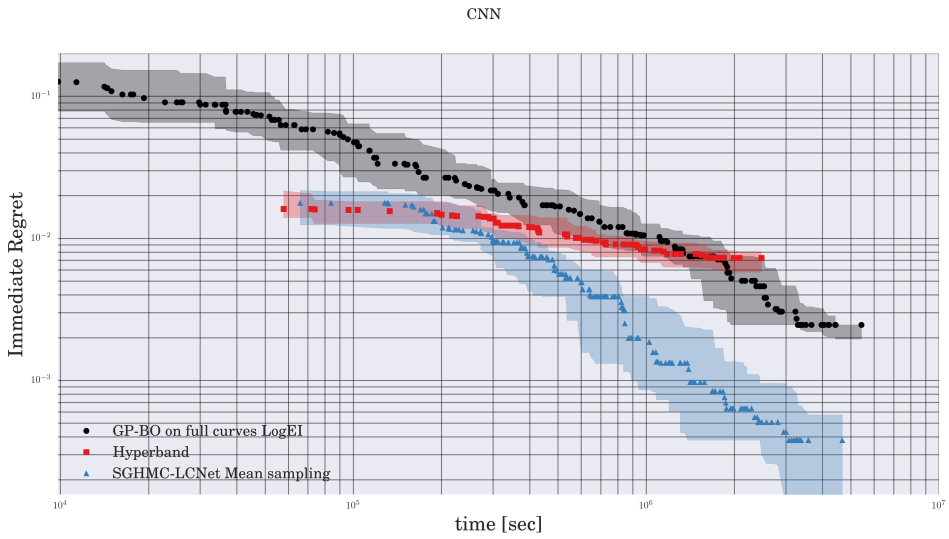

Figure 8: Comparison of Hyperband, Hyperband with our model, and standard Bayesian optimization on the CNN benchmark. Hyperband finds a good configuration faster than standard Bayesian optimization, but it only approaches the global optimum quickly when extended with our model.

Figure 8 shows the immediate regret on the CNN benchmark as a function of wallclock time, for three optimizers: Hyperband, our model-based extension of Hyperband, as well as standard Bayesian optimization with a Gaussian process (a comparison to additional baselines can be found in Appendix E). Standard Bayesian optimization does not make use of learning curves and thus needs to evaluate each configuration for the full amount of epochs. Nevertheless, since training one configuration to the end takes less time than one round of successive halving, Bayesian optimization produces its first results earlier than Hyperband. In accordance with results by Li et al. (2017), in this experiment Hyperband found a configuration with good performance faster than standard Bayesian optimization, but its random sampling did not suffice to quickly approach the best configuration; given enough time Bayesian optimization performed better. However, extended by our model, Hyperband both found a good configuration fast and approached the global optimum fastest.

We would like to emphasize that our model-based extension of Hyperband is not limited to our particular Bayesian neural networks as the underlying model. Since Hyperband stops the majority of the configurations very early, the setting is quite similar to that of very short partially observed learning curves in Section 3.2, with the differences that some configurations have been evaluated for longer and that the model now has to generalize to unseen configurations. For example, for the CNN and LR benchmarks, the experiments in Section 3.2 showed that random forests achieve strong average log-likelihoods for very short partial learning curves, and we would therefore also expect them to work well when combined with Hyperband. However, given our model's more robust likelihoods and mean squared error values we believe it to be less sensitive to the underlying data. Additionally, we can incorporate more prior knowledge about the general shape of the curves into our model, something that is not easily done for many other model classes.

4 Conclusion

We studied Bayesian neural networks for modelling the learning curves of iterative machine learning methods, such as stochastic gradient descent for convolutional neural networks. Based on the parametric learning curve models of Domhan et al. (2015), we also developed a specialized neural network architecture with a learning curve layer that improves learning curve predictions. In future work, we aim to study recurrent neural networks for predicting learning curves and will extend Bayesian optimization methods with Bayesian neural networks (Springenberg et al., 2016) based on our learning curve models.

Acknowledgment

This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721, by the European Commission under grant no. H2020-ICT-645403-ROBDREAM, and by the German Research Foundation (DFG) under Priority Programme Autonomous Learning (SPP 1527, grant HU 1900/3-1).

References

E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, 2010.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, 2012.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A Review of Bayesian Optimization. *Proc. of the IEEE*, (1), 12/2015 2016.

K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *CoRR*, 2014.

T. Domhan, J. T. 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, *Proc. of IJCAI'15*, 2015.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proc. of ICLR'17*, 2017.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proc. of AISTATS'17*, 2017.

K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In *Proc. of NIPS'13*, 2013.

M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proc. of AAAI'15*, 2015.

F. Hutter, H. Hoos, K. Leyton-Brown, and K. Murphy. Time-bounded sequential parameter optimization.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION'11*, 2011.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proc. of ICML'15*, 2015.

J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In *Proc. of NIPS'16*, 2016.

J. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proc. of ICML'15*, 2015.

M. Welling and Y. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proc. of ICML'11*, 2011.

T. Chen, E.B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proc. of ICML'14*, 2014.

D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee : The MCMC Hammer. *PASP*, 2013.

L. Breimann. Random forests. *MLJ*, 2001.

F. Hutter, L. Xu, H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods and evaluation. *AIJ*, 2014.

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

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In S. Haykin and B. Kosko, editors, *Intelligent Signal Processing*. IEEE Press, 2001. URL `http://www.iro.umontreal.ca/~lisa/pointeurs/lecun-01a.pdf`.

D. Kingma and M. Welling. Auto-encoding variational bayes. In *Proc. of ICLR'14*, 2014.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proc. of AISTATS'16*, 2016.

K. Eggensperger, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proc. of AAAI'15*, 2015.

A EXPERIMENTAL SETUP – DETAILS

Table 2 shows the hyperparameters of our 4 benchmarks described in Section 3.1 and their ranges.

	Name	Range	log scale
CNN	batch size	$[32, 512]$	-
	number of units layer 1	$[2^4, 2^{10}]$	✓
	number of units layer 2	$[2^4, 2^{10}]$	✓
	number of units layer 3	$[2^4, 2^{10}]$	✓
	learning rate	$[10^{-6}, 10^{-0}]$	✓
FCNet	initial learning rate	$[10^{-6}, 10^{0}]$	✓
	L_2 regularization	$[10^{-8}, 10^{-1}]$	✓
	batch size	$[32, 512]$	-
	γ	$[-3, -1]$	-
	κ	$[0, 1]$	-
	momentum	$[0.3, 0.999]$	-
	number units 1	$[2^5, 12^2]$	✓
	number units 2	$[2^5, 2^{12}]$	✓
	dropout rate layer 1	$[0.0, 0.99]$	-
	dropout rate layer 2	$[0.0, 0.99]$	-
LR	learning rate	$[10^{-6}, 10^{0}]$	✓
	L_2	$[0.0, 1.0]$	-
	batch size	$[20, 2000]$	-
	dropout rate on inputs	$[0.0, 0.75]$	-
VAE	L	$[1, 3]$	-
	number of hidden units	$[32, 2048]$	-
	batch size	$[16, 512]$	-
	z dimension	$[2, 200]$	-

Table 2: Hyperparameter configuration space of the four different iterative methods. For the FCNet we decayed the learning rate by a $\alpha_{decay} = (1 + \gamma * t)^{-\kappa}$ and also sampled different values for γ and κ.

B DESCRIPTION OF THE BASIS FUNCTIONS

To reduce complexity, we used a subset of the basis function from Domhan et al. (2015) which we found to be sufficient for learning curve prediction. We adapted these functions to model the residual between the asymptotic value y_∞ and we scaled the parameters Θ to be in $[0, 1]$. Table 3 shows the exact equations we used.

C PREDICTING ASYMPTOTIC VALUES OF PARTIALLY OBSERVED CURVES

Figure 9 shows the mean squared error and the average log-likelihood of predicting the asymptotic value after observing different amounts of the learning curve for all methods. See Section 3.2 in the main text for more details.

D DATASET CHARACTERISTICS

Figure 10 shows the distributions over runtimes for all random configurations of different benchmarks. As it can be seen there is a high variance of the runtime between different configurations for all benchmarks and some configurations need order of magnitudes longer than others.

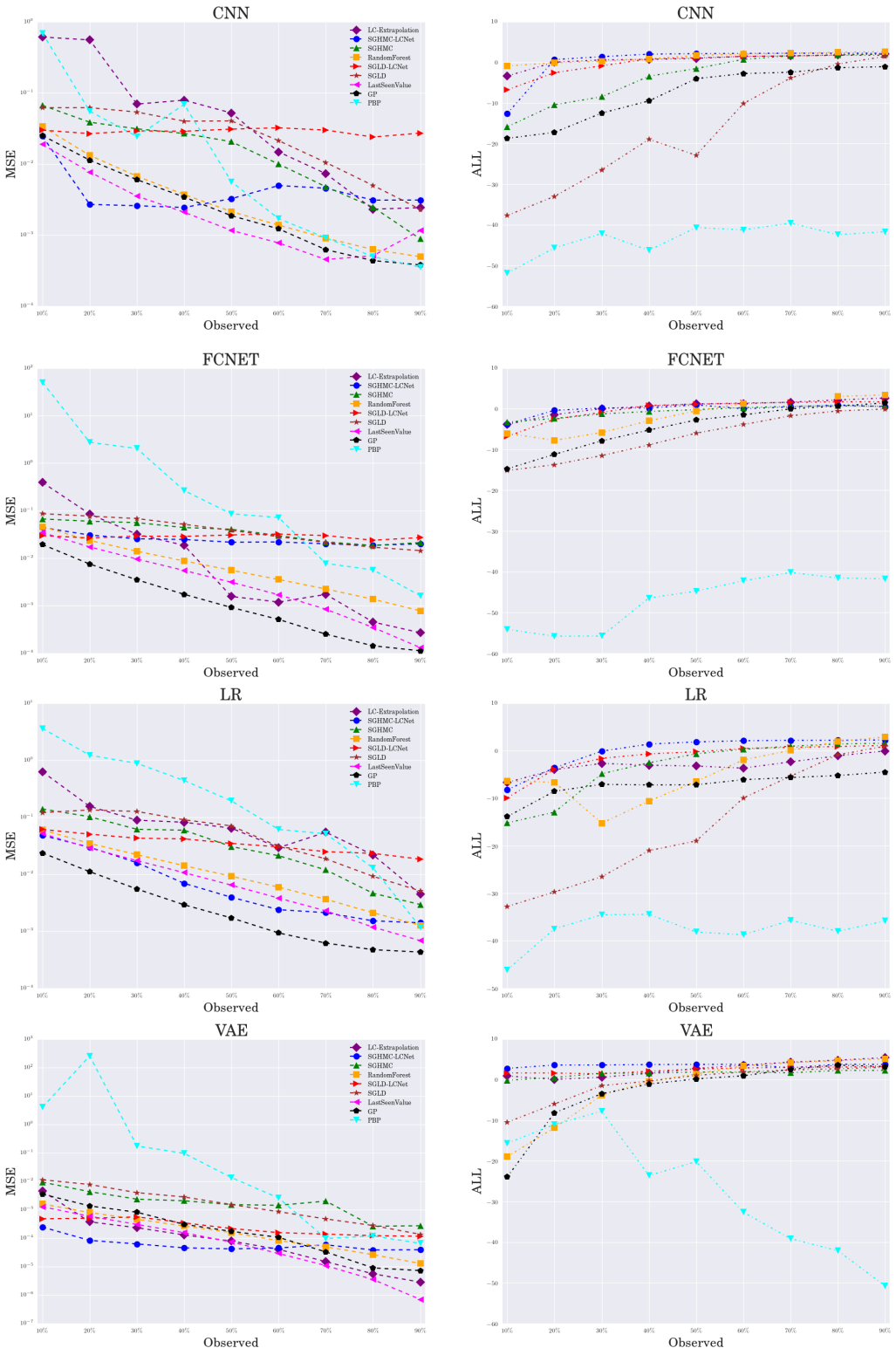

Figure 9: Assessment of the predictive quality based on partially observed learning curves on all 4 benchmarks. The panels on the left show the mean squared error of the predicted asymptotic value (y-axis) after observing all learning curves up to a given fraction of maximum number of epochs (x-axis). The panels on the right show the average log-likelihood based on the predictive mean and variance of the asymptotic value.

Name	Formula
vapor pressure	$\phi(t, a, b, c) = \mathrm{e}^{-a-0.1 \cdot b \cdot t^{-1}-0.1 \cdot c \cdot \log(t)} - \mathrm{e}^{-a-0.1 \cdot b}$
pow	$\phi(t, a, b) = a \cdot (t^b - 1)$
log power	$\phi(t, a, b, c) = 2a \cdot \left[\left(1 + \mathrm{e}^{-0.1 \cdot b \cdot c}\right)^{-1} - \left(1 + \mathrm{e}^{c \cdot (\log(t) - 0.1 \cdot b)}\right)^{-1}\right]$
exponential	$\phi(t, a, b) = a \cdot \left[\mathrm{e}^{-10 \cdot b} - \mathrm{e}^{-10 \cdot b \cdot t}\right]$
hill-3	$\phi(t, a, b, c) = a \cdot \left[\left(c^b \cdot t^{-b} + 1\right)^{-1} - \left(c^b + 1\right)^{-1}\right]$

Table 3: The formulas of our 5 basis functions.

Figure 11 shows the empirical cumulative distribution of the asymptotic performance of random configurations. These plots give an intuition about the difficulty of a benchmark as they show how many random configurations one has to sample in order to achieve a good performance.

E Optimization on Surrogate benchmarks

We follow the approach by Eggensperger et al. (2015) and used the generated datasets from Section 3.1 to build surrogates of the objective functions. This allows us to compare different optimizers on this benchmark very efficiently while (approximately) preserving the characteristics of the underlying benchmark. Using surrogates instead of optimizing the real benchmarks allows us to carry out a more thorough empirical evaluation (since single function evaluations are cheap and we therefore can afford more and longer optimization runs). We used random forests as surrogates following Eggensperger et al. (2015) since they do not introduce a bias for our approach (in contrast to, e.g., surrogates based on standard feed forward neural networks trained with stochastic gradient descent). For each benchmark, we used all configurations to train a random forest predicting the validation accuracy of a configuration at a certain time step as well as the wall clock time for its evaluation.

After each round of successive halving, we return the current best observed configuration and its asymptotic performance. For each function evaluation, we predict the accuracy, and the runtime. By adding the optimization overhead, we can predict the wallclock time necessary to optimize the real objective function. Reporting the wallclock time rather than the number of iterations also takes the additional overhead from running our method into account. Furthermore, we argue that the wallclock time is more interesting in practice.

As baselines, we compare to Gaussian process based Bayesian optimization with the upper confidence bound acquisition function (GP-BO-UCB) and the (log) expected improvement acquisition function (GP-BO-Log-EI), as well as Bayesian optimization with Bayesian neural networks (BOHAMIANN) (Springenberg et al., 2016). As an additional baseline, we use standard Bayesian neural networks (SGHMC-BNN) inside Hyperband instead of our specialized neural network architecture (SGHMC-LCNet). We compare our model-based version of Hyperband against all these baselines with sampling from the asymptotic mean (SGHMC-LCNet-mean-sampling) and the asymptotic upper confidence bound (SGHMC-LCNet-ucb-sampling). Figure 12 shows that our specialized neural network architecture helps to speed up the convergence of Hyperband on the CNN and the FCNet benchmark. For the LR benchmarks our model still improves over Hyperband but does not perform better than Bayesian optimization. Hyperband as well as our method find the optimum already after the first round of successive halving for the VAE benchmark, which leads to an regret of 0.

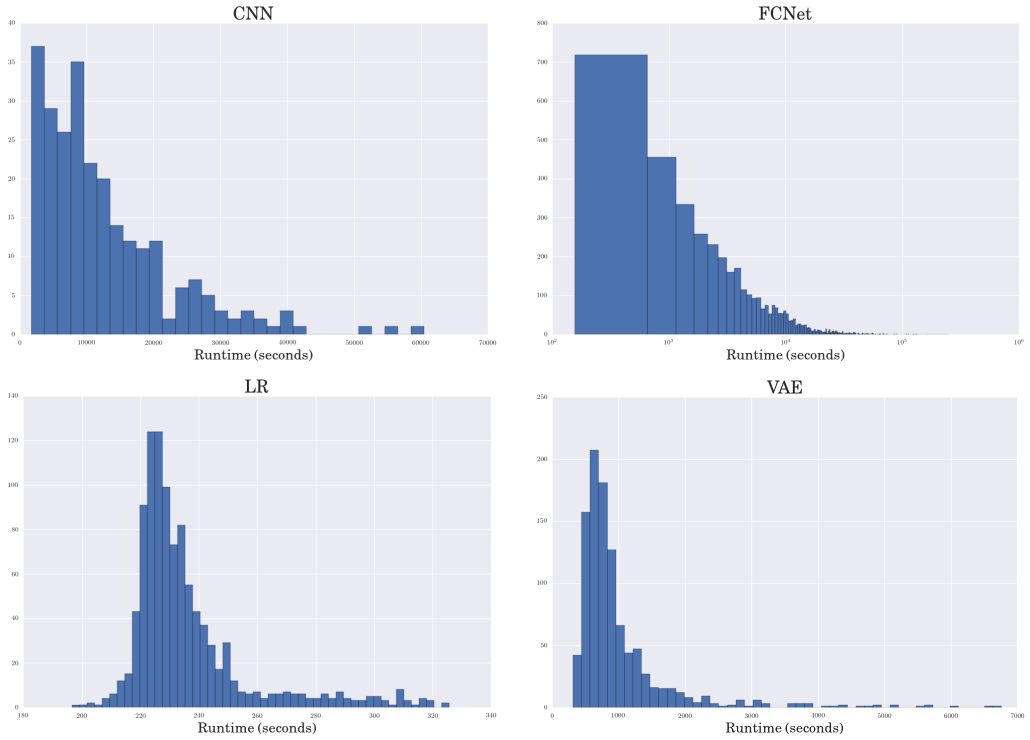

Figure 10: Distributions over runtimes for different random configurations for the CNN, FCNet, LR and VAE benchmark described in Section 3.One can see that all distributions are long tailed and that especially on the FCNet benchmark some configuration need order of magnitudes longer than others.

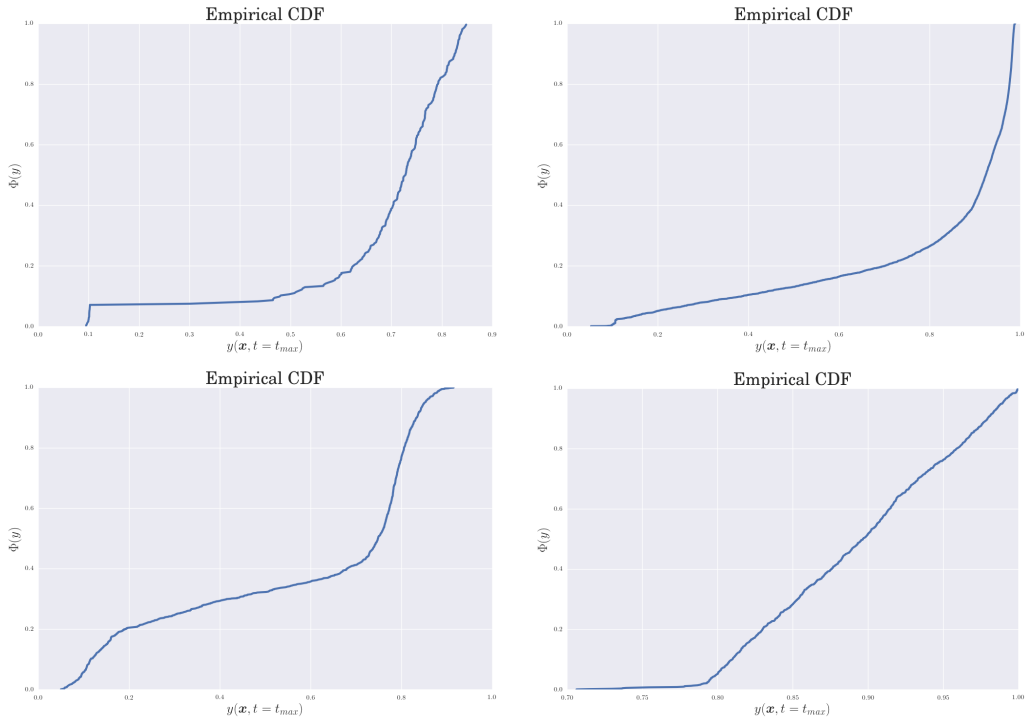

Figure 11: Empirical cumulative distributions for the CNN, FCNet, LR and VAE benchmark.

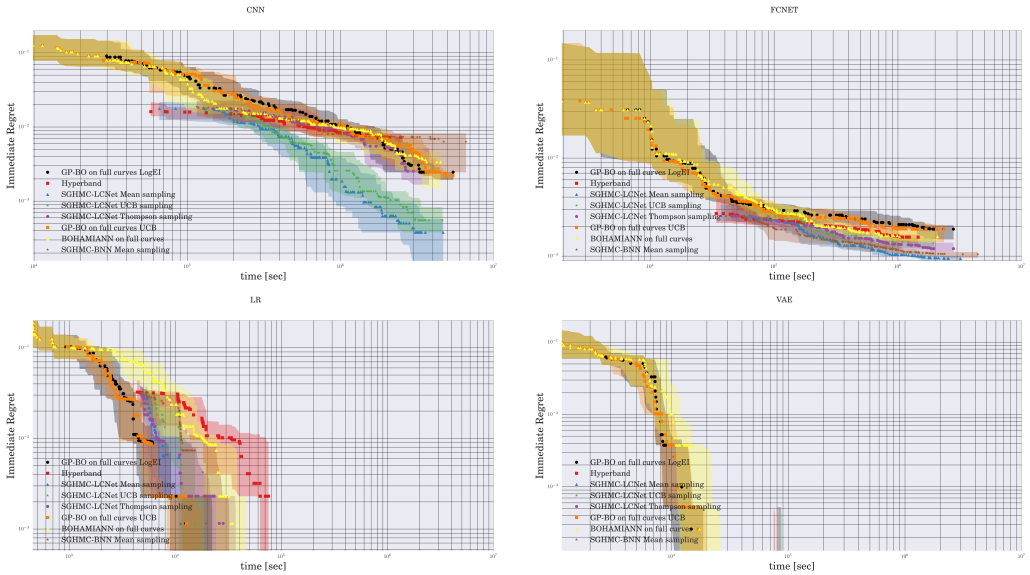

Figure 12: Different optimizers on a random forests based suggorate of all collected learning curves on different benchmarks. Note how sampling from the model improves Hyperband's performance by converging to the optimum faster.