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. Exploiting the same information in automatic Bayesian hyperparameter optimization requires 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 such 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 and based on 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 mulit-armed bandits therefore also drop the limiting blackbox assumption and exploit the performance of short runs Swersky et al. (2014a); Domhan et al. (2015); Li et al. (2016), performance on small subsets of the data Klein et al. (2016), 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. (2014a)) 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 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 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) if learning curves at stages were learning curves have not fully converged.

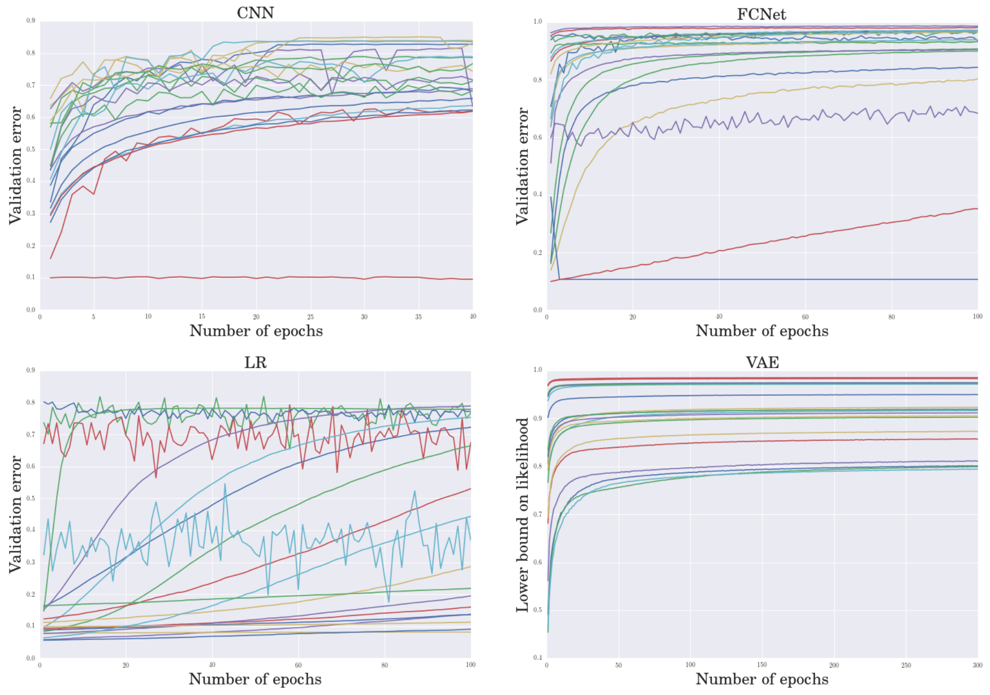

Figure 1: Example learning curves of random hyperparameter configurations of 4 different iterative machine learning methods: convolutional neural network, fully connected neural network, variational auto-encoder, and logistic regression. Although different configurations can lead to different learning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

2 Probabilistic Prediction of Learning Curve

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 between time steps and hyperparameter values that can exploit similarities between hyperparameter configurations and predict for unobserved learning curves. We also give some insights about the observation noise of the evaluation of hyperparameter configuration and 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 $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k)$ denotes the combined vector of all parameters $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k$, and $\boldsymbol{w} = (w_1, \dots, w_k)$ is the concateneted vector of the respective weights of each function. Assuming observational noise around the true but unknown value $f(t|\boldsymbol{\theta}, \boldsymbol{w})$, i.e. $y_t \sim \mathcal{N}(f(t|\boldsymbol{\theta}, \boldsymbol{w}), \sigma^2)$, Domhan et al. (2015) define a prior for all parameters $P(\boldsymbol{\Theta}, \boldsymbol{w}, \sigma^2)$ and use MCMC 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

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

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

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

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 smaller than the last value of that particular curve. This biases the estimation to never overestimated the error 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 make 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. (2014a) 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 a analytically tractable model, but using different function accounting 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 loss $f(\boldsymbol{x}, t)$ of a configuration $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ at time step $t \in [1, T]$ based on noisy observations $y(\boldsymbol{x}, t) \sim \mathcal{N}(f(\boldsymbol{x}, t), \sigma^2)$. 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 $\mathcal{D} = \{(\boldsymbol{x}_1, t_1, y_{11})), (\boldsymbol{x}_1, t_2, y_{12}), \ldots, (\boldsymbol{x}_n, t_T, y_{nT})\}$ we can then write the joint probability of the data and the model parameters as

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

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 as

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

, respectively. 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).

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 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. Figure 4 (left) shows on the vertical axis the noise $\sigma^2(\boldsymbol{x},t) = \frac{1}{R}\sum_{r=1} R(y(\boldsymbol{x},t)-\mu(\boldsymbol{x},t))^2$ and on the horizontal axis the rank of each configuration based on their mean performance $\mu(\boldsymbol{x},t) = \frac{1}{R}\sum_{r=1}^{R} y(\boldsymbol{x},t)$. Figure 4 (right) shows that the signal to noise ratio $SNR = \frac{\mu}{\sigma}$ is almost constant across configurations.

Maybe not surprisingly, the noise seems to correlate with the asymptotic performance of a configuration. More interesting is that the noise between different configurations varies on different orders of magnitudes and is thus heteroscedastic. We can incorporate this observation by making the noise dependent on the input data to allow to predict different noise levels for different hyperparameters.

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. $\hat{y}_\infty$: $\mathcal{X} \to \mathbb{R}$, 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 3 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. σ^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. A schematic of this is shown in Fig. 2. For training, we will use the same MCMC methods, namely SGLD and SGHMC described above.

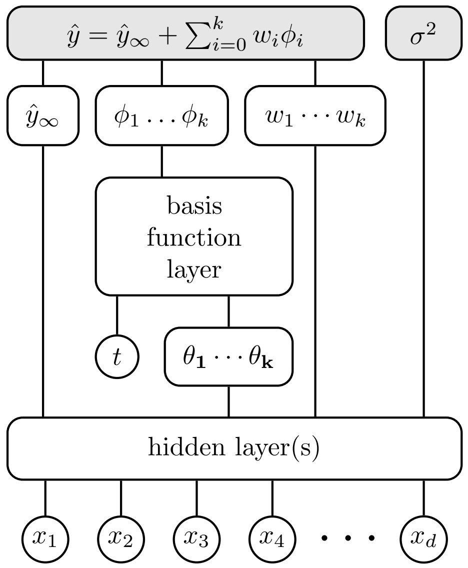

Figure 2: Our neural network architecture to model learning curves. A common hidden layer is used to simultaneously model $\hat{y}(\boldsymbol{x},T)$, the parameters of the basis functions, their respective weights, and the noise.

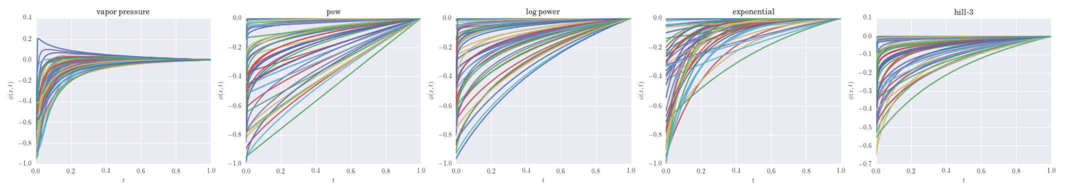

Figure 3: Example functions generated with our $k = 5$ basis functions (formulas for which are given in Appendix B). For each function, we drew 50 different parameters $\boldsymbol{\theta}_i$ uniformly at random in the output domain of the hidden layer(s) of our model. This illustrates the type of functions used to model the learning curves.

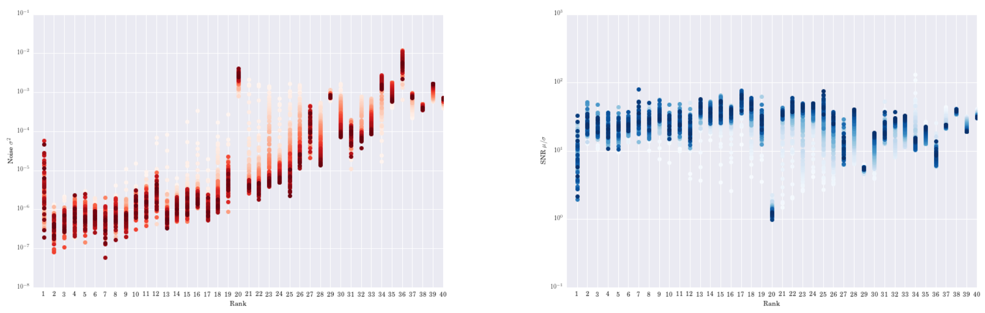

Figure 4: On the left we plot the noise estimated of 40 different configurations on the FCNet benchmark sorted by their mean performance at $t = T$. The color indicates the time step t, darker meaning a larger value. On the right we show the signal to noise ratio for the same configurations.

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 50 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 preconditoning of the gradients.

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:

- **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 autoencoder* (VAE) (Kingma and Welling, 2014). We trained the VAE on MNIST, optimizing the approximation of the lower bound for 300 epochs.

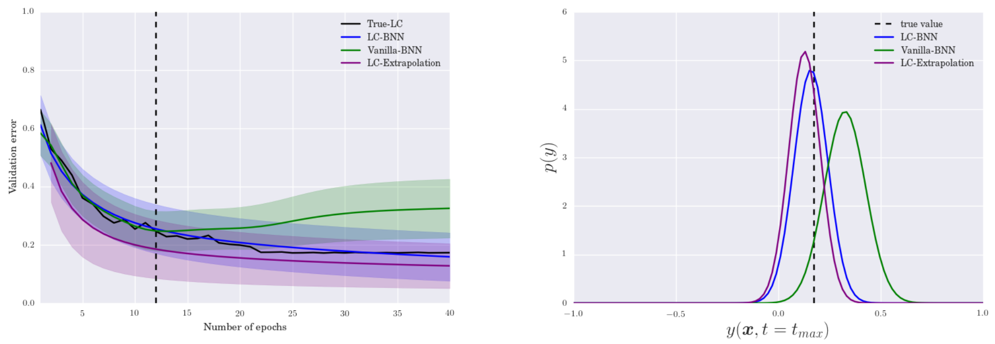

Figure 5: Qualitative comparison of the different models. The left panel shows learning on the CNN benchmark. All models observed the validation error of the first 4 epochs of the true learning curve (black). We plot the mean predictions and a one σ confidence interval around it. On the right, the posterior distributions over the value at 40 epochs is plotted.

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 method by Domhan et al. (2015) (which we dub LC-Extrapolation) works on one individual learning curve at a time and does not allow to model performance across hyperparameter configurations. Thus, we trained it separately on single partial learning curves. Our Bayesian neural network models, on the other hand, can use training data from different hyperparameter configurations. Here, for a simple comparison, we used training data with the same number of epochs for every partial learning curve.[1]

Figure 5 (left) 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(s). Figure 5 (right) shows the corresponding predictive distributions obtained with all models.

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. Figure 6 shows the mean squared error between true and predicted asymptotic value as a function of how much of the learning curves has been observed. We notice several patterns. Firstly, throughout, our specialized network architecture performed better than the standard Bayesian neural networks. Secondly, throughout, SGHMC outperformed SGLD. Finally, comparing LC-Extrapolation and the Bayesian neural network (BNN) approach, we notice that BNNs work better when only short parts of the learning curve have been observed, but that LC-Extrapolation in some cases works better for almost completely-observed learning curves. Specifically, this is the case for the datasets FCNet and VAE, for which most learning curves had already converged after about a third of the maximum number of epochs (compare Figure 1). In those cases, LC-Extrapolation can basically just predict the last observed performance value, which yields very strong performance. Our BNN approach, on the other hand, was trained on many configurations and needs 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

[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.

[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.

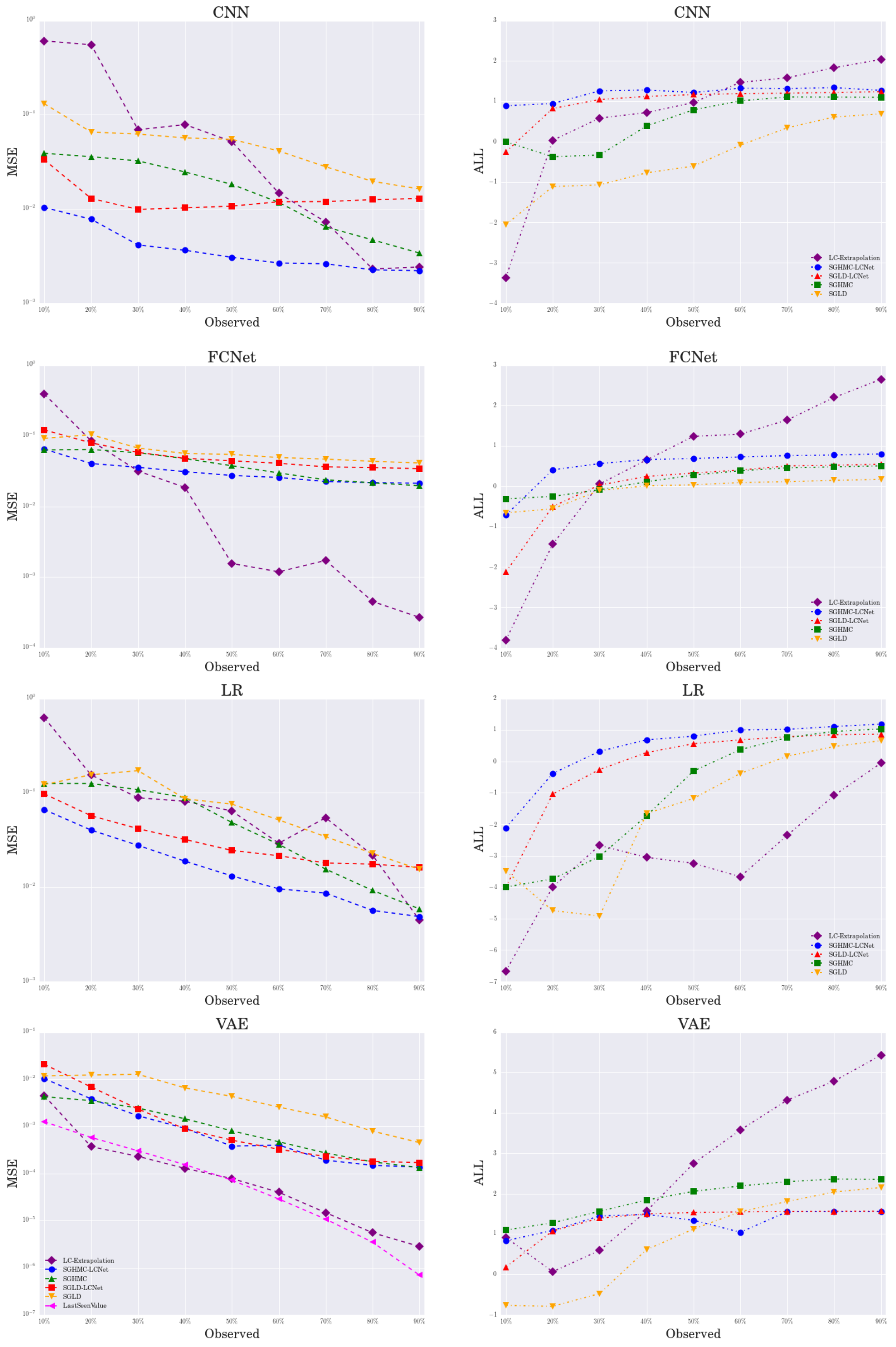

Figure 6: Assessment of the predictive quality based on partially observed learning curves. The panels on the left show the mean squared error of the prediction after the final epoch (y-axis) after observing a fraction of all learning curves (x-axis).

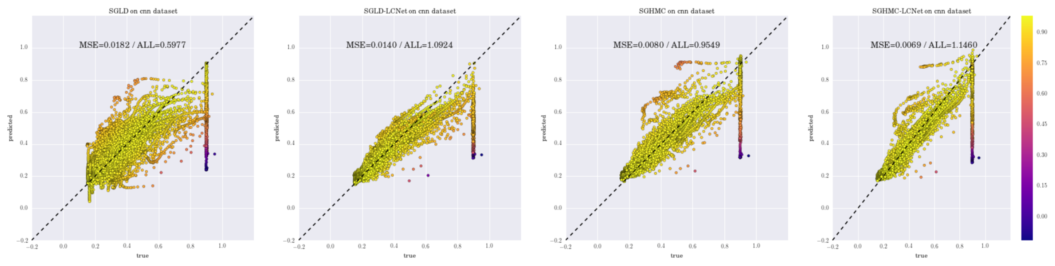

Figure 7: On the horizontal axis we plot the true value and on the vertical axis the predicted values. Each point is colored by its log-likelihood (the brighter the higher). Our learning curve BNN trained with SGHMC leads to the best mean predictions and assigns the highest likelihood to the test points.

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).

fold we trained all models on the full learning curves in the training set and let them predict for 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 errors correlate quite closely.

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 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) and under the BrainLinks-BrainTools Cluster of Excellence (grant number EXC 1086).

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*, 2014a.

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. Efficient hyperparameter optimization and infinitely many armed bandits. 2016.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *CoRR*, 2016.

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.

M. Welling and Y. Teh. Bayesian learning via stochastic gradient Langevin dynamics. 2011.

T. Chen, E.B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proc. of ICML'14*, 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. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *CoRR*, 2014b.

A Experimental Setup – Details

	Name	Range	log scale
CNN	batch size	$[32, 512]$	-
	number of units layer 1	$[4, 10]$	✓
	number of units layer 2	$[4, 10]$	✓
	number of units layer 3	$[4, 10]$	✓
	learning rate	$[10^{-6}, 10^{-0}]$	✓
FCNet	inital 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	$[5, 12]$	✓
	number units 2	$[5, 12]$	✓
	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 just used a subset of basis function from Domhan et al. (2015) which we found to be sufficient for learning curve prediction. We slightly changed these function such that their parameters are between $[0, 1]$.

Name	**Formula**
vapor pressure	$\theta(t, a, b, c) = \exp(-a - {}^{b}/_{10} - {}^{c}/_{10} \log(t)) - \exp(-a - {}^{b}/_{10})$
pow	$\theta(t, a, b) = a(t^b - 1)$
log power	$\theta(t, a, b, c) = 2a\frac{1}{1+\exp(-c^{b}/10)} - \frac{1}{1+(\frac{t}{\exp(b/10)})^c}$
exponential	$\theta(t, a, b) = b(-\exp(-10at) + \exp(-10a))$
hill-3	$\theta(t, a, b, c) = a(\frac{1}{(c/t)^b+1} - \frac{1}{c^b+1})$

Table 3: The formulas of our 5 basis functions.