IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 3 October 2023, accepted 30 October 2023, date of publication 2 November 2023, date of current version 8 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329685

RESEARCH ARTICLE

Improved Uncertainty Quantification for Neural Networks With Bayesian Last Layer

FELIX FIEDLER AND SERGIO LUCIA, (Member, IEEE)
Chair of Process Automation Systems, TU Dortmund University, 44227 Dortmund, Germany
Corresponding author: Felix Fiedler (felix.fiedler@tu-dortmund.de)

ABSTRACT Uncertainty quantification is an important task in machine learning - a task in which standard neural networks (NNs) have traditionally not excelled. This can be a limitation for safety-critical applications, where uncertainty-aware methods like Gaussian processes or Bayesian linear regression are often preferred. Bayesian neural networks are an approach to address this limitation. They assume probability distributions for all parameters and yield distributed predictions. However, training and inference are typically intractable and approximations must be employed. A promising approximation is NNs with Bayesian last layer (BLL). They assume distributed weights only in the linear output layer and yield a normally distributed prediction. To approximate the intractable Bayesian neural network, point estimates of the distributed weights in all but the last layer should be obtained by maximizing the marginal likelihood. This has previously been challenging, as the marginal likelihood is expensive to evaluate in this setting. We present a reformulation of the log-marginal likelihood of a NN with BLL which allows for efficient training using backpropagation. Furthermore, we address the challenge of uncertainty quantification for extrapolation points. We provide a metric to quantify the degree of extrapolation and derive a method to improve the uncertainty quantification for these points. Our methods are derived for the multivariate case and demonstrated in a simulation study. In comparison to Bayesian linear regression with fixed features, and a Bayesian neural network trained with variational inference, our proposed method achieves the highest log-predictive density on test data.

INDEX TERMS Bayesian last layer, Bayesian neural network, uncertainty quantification.

I. INTRODUCTION

Machine learning tries to capture patterns and trends through data. Both, the data and the identified patterns are subject to uncertainty [1], [2]. For many applications, especially those where machine learning is applied to safety critical tasks, it is imperative to quantify the uncertainty of the predictions. An important example is learning-based control, where probabilistic system models are identified from data and used for safe control decisions [3], [4], [5], [6], [7], [8].

Bayesian linear regression (BLR) [3], [4], [5], [6], [7] and Gaussian processes (GPs) [7], [8] are prominent methods for probabilistic system identification. Both assume that a nonlinear feature space can be mapped linearly to the outputs. As a main difference, BLR requires the features to be explicitly defined, while for GPs the features are implicitly defined through a kernel function [9]. This is an advantage of GPs, as the most suitable features for BLR are often challenging to determine. On the other hand, GPs scale poorly with the number of data samples [10]. For big data problems they are typically approximated with sparse GPs [11], and can be further improved with deep kernel learning [12], [13].

Especially in recent years, neural networks (NNs) and deep learning have gained significant popularity for a vast variety of machine learning tasks [14]. NNs have also been successfully applied for control applications [15], [16], often to infer the system model from data [17], [18], [19]. A challenge with NNs is their tendency to overfit and their inability to express uncertainty [1]. Bayesian neural networks (BNNs), in which the weights and predictions are probability distributions, are a concept to tackle this shortcoming. In practice, BNNs can be intractable to train and query and are often approximated [1], [2]. Most approximate BNN approaches fall into one of two categories: Markov chain

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

Monte Carlo (MCMC) and variational inference (VI) [1]. MCMC methods do not require a classical training phase and instead sample directly the posterior weight distribution of the Bayesian neural network [20]. Unfortunately, MCMC scales poorly to large models which limits their applicability [1]. Variational inference is a popular alternative to MCMC and based on the idea of learning the parameters of a surrogate posterior distribution of the weights [1], [2]. For neural networks, variational inference is often implemented with the *Bayes by Backprop* algorithm [21].

Unfortunately, even a BNN trained with VI requires sampling to approximate the predictive distribution [1]. This can be a significant disadvantage in comparison to GPs and BLR, which yield analytical results. A promising compromise between tractability and expressiveness are NNs with Bayesian last layer (BLL) [10], [22]. NNs with BLL can be seen as a simplified BNN, where only the weights of the output layer follow a Gaussian distribution, and the remaining layers contain deterministic weights. At the same time, they can be interpreted as a deep kernel learning approach with linear kernel. NNs with BLL are also strongly related to BLR in that they consider a nonlinear feature space which is mapped linearly onto the outputs. Similarly to GPs, BLR and other probabilistic models [9], [23], NNs with BLL are trained by maximizing the marginal likelihood with respect to the parameters of the probabilistic model. These parameters include prior and noise variance and, importantly, the weights of the deterministic layers. While the log-marginal likelihood (LML) can be expressed analytically for NNs with BLL, it contains expressions such as the inverse of the precision matrix, making it unsuitable for direct gradient-based optimization. Previous works, especially for control applications, have therefore either assumed knowledge of all parameters [3], [5], [19], including prior and noise variance, or have maximized the LML after training a NN with fixed features [4], [24]. In previous works that did include the deterministic weights as parameters, maximizing the LML required sampling the surrogate posterior during training [22], or using an approximate precision matrix [12] to enable gradient-based optimization.

As a main contribution of this work, we propose an approach to maximize the exact LML of a NN with BLL that does not require sampling and is suitable for gradient-based optimization. Most importantly, we avoid the matrix inverse in the LML by reintroducing the weights of the last layer, which were marginalized, as optimization variables. We show that our reformulation of the LML satisfies the conditions of optimality for the same solution as the original formulation. In this way, we provide a simpler training procedure, in comparison to training with variational inference, and can outperform BLR with fixed features obtained from a NN. Our second main contribution is an algorithm to improve the uncertainty quantification for extrapolation points. To this end, we relate the computation of the BLL covariance to an intuitive metric for extrapolation, which is inspired by the definition of extrapolation in [25]. Based on this relationship, the proposed algorithm adjusts a scalar parameter to improve the log-predictive density on additional validation data. Our proposed methods are derived for the multivariate-case, estimating individual noise-variances for each output. This is in contrast to GP and BLR applications, where the multivariate case is typically tackled by fitting an individual model to each output [4], [7], [8] or by assuming i.i.d. noise for all outputs [5]. The advantage of a single model for the multivariate case is apparent for the application of system identification for optimization-based control, as multiple models may significantly increase computation times.

This work is structured as follows. In Section II, we introduce NNs with BLL. The marginal likelihood, which is maximized during training, is discussed in Section III. In Section IV, we discuss interpolation and extrapolation for NNs with BLL and present an algorithm to improve the predictive distribution in the extrapolation regime. We discuss the special considerations required for the multivariate case in Section V. The Bayes by Backprop method is introduced in Section VI. A simulation example to compare the proposed method with BLR and Bayes by Backprop is presented in Section VII. The paper is concluded in Section VIII.

II. BAYESIAN LAST LAYER

We investigate a dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{t}\}$ consisting of m data pairs of inputs $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and targets $t \in \mathbb{R}$ from which the matrices $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_m]^\top \in \mathbb{R}^{m \times n_x}$ and $\boldsymbol{t} = [t_1, \dots, t_m]^\top \in \mathbb{R}^{m \times 1}$ are formed. We assume a scalar output for ease of notation and address the multivariate case in Section V. Regarding the notation, lower case bold symbols denote vectors, upper case bold symbols denote matrices, and regular lower case symbols are scalars. For the regression task, we introduce a feed-forward NN model with L hidden layers as

$$y = NN(\boldsymbol{x}; \mathbb{W}_{L+1}) = g_{L+1} \circ h_{L+1} \circ \cdots \circ g_1 \circ h_1(\boldsymbol{x}), \quad (1)$$

where $\circ$ denotes function composition. At each layer, we have a linear mapping $h_l(\cdot)$ followed by a nonlinear activation function $g_l(\cdot)$, that is:

$$\boldsymbol{h}_l^\top = h_l(\boldsymbol{a}_{l-1}) = \begin{bmatrix} \boldsymbol{a}_{l-1}^\top, & 1 \end{bmatrix} \boldsymbol{W}_l \quad l \in \mathbb{I}_{[1,L+1]}, \quad (2)$$

$$\boldsymbol{a}_l = g_l(\boldsymbol{h}_l) \quad l \in \mathbb{I}_{[1,L+1]}. \quad (3)$$

The set $\mathbb{I}_{[1,L+1]}$ denotes the set of integers from 1 to $L+1$. We have $\boldsymbol{a}_0 = \boldsymbol{x}$, $\boldsymbol{a}_{L+1} = \boldsymbol{y}$, and $\boldsymbol{a}_l \in \mathbb{R}^{n_{a,l}}$ for all $l \in \mathbb{I}_{[1,L]}$. The number of neurons in layer l is denoted as $n_{a,l}$, and we have the weights $\boldsymbol{W} \in \mathbb{R}^{n_{a,l-1}+1 \times n_{a,l}}$, which include the bias term. The set of weight matrices with cardinality $L+1$ is denoted $\mathbb{W}_{L+1} = \{\boldsymbol{W}_1, \dots, \boldsymbol{W}_{L+1}\}$. As a requirement for NNs with BLL, we state the following assumption.

Assumption 1: The NN (1) has a linear activation function in the output layer, i.e. $g_{L+1}(h_{L+1}) = h_{L+1}$.

With the linear mapping, the output of the last internal layer is of particular importance and we introduce the notation:

$$\tilde{\boldsymbol{\phi}} = \boldsymbol{a}_L, \text{ and } \boldsymbol{\phi} = [\boldsymbol{a}_L^\top, \ 1]^\top, \quad (4)$$

where $\tilde{\boldsymbol{\phi}} \in \mathbb{R}^{n_{\tilde{\phi}}}$ are referred to as *linear features* and $\boldsymbol{\phi} \in \mathbb{R}^{n_\phi}$ are the corresponding *affine features* with $n_\phi = n_{\tilde{\phi}} + 1$. With slight abuse of notation, we use $\tilde{\boldsymbol{\phi}} = \tilde{\boldsymbol{\phi}}(\boldsymbol{x}; \mathbb{W}_L)$ and $\boldsymbol{\phi} = \boldsymbol{\phi}(\boldsymbol{x}; \mathbb{W}_L)$ as both, the values of the features, and the function, parameterized with $\mathbb{W}_L$. We require two additional assumptions to state Lemma 1 which formally describes a NN with BLL.

Assumption 2: The NN (1) provides a feature space $\boldsymbol{\phi}(\boldsymbol{x}; \mathbb{W}_L) \in \mathbb{R}^{n_\phi}$ from which the targets are obtained through a linear mapping, according to Assumption 1:

$$\boldsymbol{t} = \boldsymbol{\phi}\left(\boldsymbol{X}; \mathbb{W}_L\right)^\top \boldsymbol{w} + \boldsymbol{\epsilon} = \boldsymbol{y} + \boldsymbol{\epsilon}, \tag{5}$$

where we introduce the set $\mathbb{W}_L = \{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L\}$ for all weights until layer L and have $\boldsymbol{w} = \boldsymbol{W}_{L+1}$ the weights of the output layer. The additive noise $\boldsymbol{\epsilon} \in \mathbb{R}^m$ is zero-mean normally distributed, that is, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_E)$.

Assumption 3: We have a prior belief for the weights of the output layer $\mathbf{w} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_W)$.

We introduce the parameters of the probabilistic model (5) as:

$$\boldsymbol{\Theta} = \{\mathbb{W}_L, \boldsymbol{\Sigma}_E, \boldsymbol{\Sigma}_W\}, \tag{6}$$

and state the posterior distribution of the weights $\mathbf{w}$ of the last layer using Bayes' law:

$$p(\mathbf{w}|\mathcal{D}, \boldsymbol{\Theta}) = \frac{p(\mathcal{D}|\mathbf{w}, \boldsymbol{\Theta}) p(\mathbf{w}|\boldsymbol{\Theta})}{p(\mathcal{D}|\boldsymbol{\Theta})}. \tag{7}$$

A neural network with Bayesian last layer has deterministic weights in all hidden layers and distributed weights in the output layer. We obtain an analytical expression for the distribution of the the predicted outputs as shown in the following lemma [22].

Lemma 1: Assumptions 1-3 hold. The predicted outputs are normally distributed with:

$$p(\mathbf{y}|\mathcal{D}, \boldsymbol{\Theta}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_y^{NN}(\boldsymbol{x}), \boldsymbol{\Sigma}_y^{NN}(\boldsymbol{x})), \tag{8a}$$

$$\boldsymbol{\mu}_y^{NN}(\boldsymbol{x}) = NN(\boldsymbol{x}; \mathbb{W}_{L+1}), \tag{8b}$$

$$\boldsymbol{\Sigma}_y^{NN}(\boldsymbol{x}) = \boldsymbol{\phi}^\top \boldsymbol{\Lambda}_p^{-1} \boldsymbol{\phi}, \tag{8c}$$

where $\boldsymbol{\Phi} = \boldsymbol{\phi}(\boldsymbol{X}; \mathbb{W}_L)$ is the feature matrix for the training data, $\boldsymbol{\phi} = \boldsymbol{\phi}(\boldsymbol{x}; \mathbb{W}_L)$ is the feature matrix for the test data, and with the precision matrix

$$\boldsymbol{\Lambda}_p = \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Phi} + \boldsymbol{\Sigma}_W^{-1}. \tag{9}$$

Proof: The posterior $p(\mathbf{w}|\mathcal{D}, \boldsymbol{\Theta})$ in (7) yields a normal distribution in the weights:

$$p(\mathbf{w}|\mathcal{D}, \boldsymbol{\Theta}) = \mathcal{N}(\bar{\boldsymbol{w}}, \boldsymbol{\Lambda}_p^{-1}), \tag{10a}$$

$$\text{with:} \quad \bar{\boldsymbol{w}} = \boldsymbol{\Lambda}_p^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_E^{-1} \boldsymbol{t}, \tag{10b}$$

as shown for the case of arbitrary features in [23]. Finally, the predicted output $\mathbf{y}$ is a linear transformation of the random variable $\mathbf{w}$, yielding the distribution in (8). □

We can also obtain a posterior distribution for the targets, for which we consider (5) and obtain

$$p(\mathbf{t}|\mathcal{D}, \boldsymbol{\Theta}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_y^{\text{NN}}(\boldsymbol{x}), \boldsymbol{\Sigma}_t^{\text{NN}}(\boldsymbol{x})), \tag{11a}$$

$$\boldsymbol{\Sigma}_t^{\text{NN}}(\boldsymbol{x}) = \boldsymbol{\Sigma}_y^{\text{NN}}(\boldsymbol{x}) + \boldsymbol{\Sigma}_E, \tag{11b}$$

where $\boldsymbol{\mu}_y^{\text{NN}}(\boldsymbol{x})$ and $\boldsymbol{\Sigma}_y^{\text{NN}}(\boldsymbol{x})$ stem from (8). We will revisit (11) again for the definition of performance metrics.

III. MARGINAL LIKELIHOOD MAXIMIZATION

The posterior distribution of the weights of the last layer can be obtained with (10) for given values of the parameters $\boldsymbol{\Theta}$. Determining suitable values of the parameters from prior knowledge is often challenging. Instead, the parameters can be inferred by maximizing the marginal likelihood in (7), which is also known as empirical Bayes or type-2 maximum likelihood [26]. In BLR, that is, for a fixed feature space, the log-marginal likelihood (LML) can be maximized as shown in [23]. Following this idea, the authors in [24] propose an approach where a fixed feature space is obtained through classical NN regression. After NN training, the LML is then maximized with these fixed features. However, the LML is also influenced by the weights of the deterministic layers and the resulting feature space. It is therefore reasonable to include the set of weights $\mathbb{W}_L$ until the last layer as parameters and train them by maximizing the LML. Unfortunately, this poses significant challenges.

We proceed by introducing the LML and discuss the challenge of maximizing this expression in the next subsection.

Lemma 2 (Log-Marginal Likelihood): If Assumptions 1-3 hold, the negative LML of (7), denoted as $J(\boldsymbol{\Theta}; \mathcal{D}) = -\log(\mathcal{D}|\boldsymbol{\Theta})$, results in:

$$\begin{aligned} J(\boldsymbol{\Theta}; \mathcal{D}) = &\frac{m}{2}\log(2\pi) + \frac{1}{2}\log\det(\boldsymbol{\Sigma}_W) + \frac{1}{2}\log\det(\boldsymbol{\Sigma}_E) \\ &+ \frac{1}{2}\log\det(\boldsymbol{\Lambda}_p) + \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{y}\|^2_{\boldsymbol{\Sigma}_E^{-1}} + \frac{1}{2}\|\bar{\boldsymbol{w}}\|^2_{\boldsymbol{\Sigma}_W^{-1}}. \end{aligned} \tag{12}$$

In this formulation, we have $\boldsymbol{\Lambda}_p$ from (9), $\bar{\boldsymbol{w}}$ from (10b), $\boldsymbol{y} = \boldsymbol{\Phi}\bar{\boldsymbol{w}}$ and $\boldsymbol{\Theta}$ from (6).

Proof: The proof for fixed features is shown in [23]. □

For practical applications, the formulation in (12) is further simplified by considering the following assumption, where $\text{diag}(\boldsymbol{A}, \boldsymbol{B})$ denotes a block-diagonal matrix with $\boldsymbol{A}$ and $\boldsymbol{B}$ on the diagonal.

Assumption 4: The additive noise introduced in Assumption 2 is i.i.d. for all samples m, yielding $\boldsymbol{\Sigma}_E = \sigma_e^2 \boldsymbol{I}_m$. The weight prior introduced in Assumption 3 is i.i.d. with a flat prior for the bias, that is, $\boldsymbol{\Sigma}_w^{-1} = \sigma_w^{-2}\text{diag}(\boldsymbol{I}_{n_{\tilde{\phi}}}, 0)$.

The assumption of a flat prior for the bias term will have a negligible effect in practice but is important for the theoretical results presented in Section IV. For ease of notation, we will reuse $J(\boldsymbol{\Theta}; \mathcal{D})$ and $\boldsymbol{\Theta}$ in following results.

Result 1: Applying Assumption 4 to Lemma 2, we can write the negative LML in (12) as:

$$\begin{aligned} J(\boldsymbol{\Theta}; \mathcal{D}) = &\frac{m}{2}\log(2\pi) + n_\phi \log(\sigma_w) + m\log(\sigma_e) \\ &+ \frac{1}{2}\log\det(\boldsymbol{\Lambda}_p) + \frac{1}{2\sigma_e^2}\|\boldsymbol{t} - \boldsymbol{y}\|_2^2 + \frac{1}{2\sigma_w^2}\|\bar{\boldsymbol{w}}\|_2^2. \end{aligned} \tag{13}$$

We introduce $\tilde{\boldsymbol{I}}_{n_\phi} = \operatorname{diag}(\boldsymbol{I}_{n_{\tilde{\phi}}}, 0)$ *and obtain simplified expressions for (9) and (10b):*

$$\boldsymbol{\Lambda}_p = \sigma_e^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{\sigma_w^2}\tilde{\boldsymbol{I}}_{n_\phi}, \tag{14a}$$

$$\bar{\boldsymbol{w}} = \sigma_e^{-2}\boldsymbol{\Lambda}_p^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}. \tag{14b}$$

In this setting, we denote $\boldsymbol{\Theta} = \{\mathbb{W}_L, \sigma_e, \sigma_w\}$.

A. AUGMENTED LOG-MARGINAL LIKELIHOOD MAXIMIZATION

To train a NN with BLL we seek to minimize the negative LML:

$$\min_{\boldsymbol{\Theta}} \quad J(\boldsymbol{\Theta}; \mathcal{D}), \tag{15}$$

with $J(\boldsymbol{\Theta}; \mathcal{D})$ and $\boldsymbol{\Theta}$ according to Result 1. This problem excludes the weights in the last layer of the NN as they have been marginalized. A result of this marginalization is the expression (14b) which computes these weights explicitly. Unfortunately, this creates a major challenge when iteratively solving problem (15) for the optimal values $\boldsymbol{\Theta}^*$, as the computational graph contains unfavorable expressions such as the inverse of $\boldsymbol{\Lambda}_p$, which are both numerically challenging and computationally expensive. Furthermore, (14b) requires leveraging the entire training dataset for the computation of the gradient $\nabla_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}; \mathcal{D})$.

The authors in [22] circumvent these issues by variational inference, replacing the LML objective by the evidence lower bound objective (ELBO). As the variational posterior they choose a Gaussian distribution, parameterized with mean and covariance. This is the obvious choice in the BLL setting where the true posterior is Gaussian as shown in (8) and, consequently, the ELBO objective is equivalent to the LML [22]. Variational inference comes at the cost, however, of introducing as optimization variables the parameters to describe the variational posterior. Furthermore, the variational inference training loop requires sampling this variational posterior.

In this work, we present an alternative approach, which simultaneously avoids parameterizing the variational posterior and yields a computational graph with lower complexity than (13). The resulting formulation is suitable for fast gradient-based optimization. To obtain this result we reformulate (15) as

$$\begin{aligned} \min_{\boldsymbol{\Theta}, \bar{\boldsymbol{w}}} \quad & J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}) \\ \text{s.t.} \quad & \bar{\boldsymbol{w}} = \frac{1}{\sigma_e^2}\boldsymbol{\Lambda}_p^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}, \end{aligned} \tag{16}$$

with $\boldsymbol{\Theta} = \{\mathbb{W}_L, \sigma_e, \sigma_w\}$. Importantly, we introduce $\bar{\boldsymbol{w}}$ as an optimization variable and add an equality constraint corresponding to (14b). The optimal solution $\boldsymbol{\Theta}^*$ of (16) is thus identical to the optimal solution of (15). As a main contribution of this work, we state the following theorem.

Theorem 1 (Augmented Log-Marginal Likelihood Maximization): The optimal solution $(\boldsymbol{\Theta}^*, \bar{\boldsymbol{w}}^*)$ of problem (16) is identical to the optimal solution obtained from the unconstrained problem:

$$\min_{\boldsymbol{\Theta}, \bar{\boldsymbol{w}}} \quad J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}), \tag{17}$$

where, as the only difference to (15), $\bar{\boldsymbol{w}}$ is now an optimization variable and, in comparison to (16), the equality constraint has been dropped.

Proof: The Lagrangian of problem (16) can be written as:

$$\mathcal{L}(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}, \lambda; \mathcal{D}) = J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}) + \lambda^\top\left(\boldsymbol{\Lambda}_p\bar{\boldsymbol{w}} - \frac{1}{\sigma_e^2}\boldsymbol{\Phi}^\top\boldsymbol{t}\right).$$

We then state the first-order condition of optimality for the optimization variable $\bar{\boldsymbol{w}}$:

$$\nabla_{\bar{\boldsymbol{w}}}\mathcal{L}(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}, \lambda; \mathcal{D}) = \nabla_{\bar{\boldsymbol{w}}} J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}) + \lambda^\top\boldsymbol{\Lambda}_p \overset{!}{=} 0. \tag{18}$$

Considering (13) and (5), that is, $\boldsymbol{y} = \boldsymbol{\Phi}\bar{\boldsymbol{w}}$, we obtain:

$$\begin{aligned} &\nabla_{\bar{\boldsymbol{w}}} J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}) \\ &\quad= \nabla_{\bar{\boldsymbol{w}}}\left(\frac{1}{2\sigma_e^2}\|\boldsymbol{t} - \boldsymbol{\Phi}\bar{\boldsymbol{w}}\|_2^2 + \frac{1}{2\sigma_w^2}\|\bar{\boldsymbol{w}}\|_2^2\right) && (19)\\ &\quad= \frac{2}{2\sigma_e^2}\left(\bar{\boldsymbol{w}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{t}^\top\boldsymbol{\Phi}\right) + \frac{2}{2\sigma_w^2}\bar{\boldsymbol{w}}^\top && (20)\\ &\quad= \bar{\boldsymbol{w}}^\top\left(\frac{1}{\sigma_e^2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{\sigma_w^2}\tilde{\boldsymbol{I}}_{n_\phi}\right) - \frac{1}{\sigma_e^2}\boldsymbol{t}^\top\boldsymbol{\Phi}. && (21) \end{aligned}$$

Using (14a), we obtain:

$$\nabla_{\bar{\boldsymbol{w}}} J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}) = \bar{\boldsymbol{w}}^\top\boldsymbol{\Lambda}_p - \frac{1}{\sigma_e^2}\boldsymbol{t}^\top\boldsymbol{\Phi}. \tag{22}$$

We substitute (22) into (18) and have:

$$\begin{aligned} &\lambda^\top\boldsymbol{\Lambda}_p = -\bar{\boldsymbol{w}}^\top\boldsymbol{\Lambda}_p + \frac{1}{\sigma_e^2}\boldsymbol{t}^\top\boldsymbol{\Phi}, && (23)\\ &\Leftrightarrow \lambda = -\boldsymbol{\Lambda}_p^{-1}\boldsymbol{\Lambda}_p\bar{\boldsymbol{w}} + \frac{1}{\sigma_e^2}\boldsymbol{\Lambda}_p^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}, && (24)\\ &\Leftrightarrow \lambda = -\bar{\boldsymbol{w}} + \bar{\boldsymbol{w}} = 0, && (25) \end{aligned}$$

where in (24) we have substituted (14b). From (25), we obtain that $\lambda = 0$ and the Lagrangian of problem (16) thus simplifies to:

$$\mathcal{L}(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}, \lambda; \mathcal{D}) = J(\boldsymbol{\Theta}, \bar{\boldsymbol{w}}; \mathcal{D}). \tag{26}$$

This is exactly the Lagrangian of problem (17) and therefore problem (16) and (17) yield the same optimal solution. □

Theorem 1 enables us to maximize the LML (13) without having to explicitly compute $\bar{\boldsymbol{w}}$ according to Equation (14b). This means, in particular, that we are not required to compute the inverse of $\boldsymbol{\Lambda}_p$ to express the LML. Consequentially, we can use back-propagation and gradient-based optimization to maximize the LML, which significantly simplifies training NNs with BLL.

B. CHANGE OF VARIABLES AND SCALING

With Theorem 1 we can state the LML as a function of $\bar{\boldsymbol{w}}$ which is now included in the set of parameters $\boldsymbol{\Theta} = \{\mathbb{W}_L, \bar{\boldsymbol{w}}, a, b\} = \{\mathbb{W}_{L+1}, a, b\}$. Furthermore, we propose a change of variables and scale the objective function to improve numerical stability. In particular, we introduce:

$$\alpha = \frac{\sigma_w^2}{\sigma_e^2}, \tag{27}$$

which can be interpreted as a signal-to-noise ratio and optimize over $\log(\alpha)$ and $\log(\sigma_e)$ to ensure that $\sigma_e > 0$ and $\sigma_w > 0$ without constraining the problem. These changes are formalized in the following result.

Result 2: Considering the definition of α in (27), we reformulate (14a):

$$\begin{aligned}\boldsymbol{\Lambda}_p &= \sigma_e^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \sigma_w^{-2}\tilde{\boldsymbol{I}}_{n_\phi} \\ &= \sigma_e^{-2}\left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \alpha^{-1}\tilde{\boldsymbol{I}}_{n_\phi}\right) = \sigma_e^{-2}\bar{\boldsymbol{\Lambda}}_p,\end{aligned} \tag{28}$$

where

$$\bar{\boldsymbol{\Lambda}}_p = \boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \alpha^{-1}\tilde{\boldsymbol{I}}_{n_\phi}. \tag{29}$$

The scaled negative LML from (13) can then be written as:

$$\begin{aligned}&J(\boldsymbol{\Theta}; \mathcal{D}) \\ &= \frac{1}{2}\log(2\pi) + \frac{n_\phi}{2m}\log\alpha + \log\sigma_e \\ &\quad + \frac{1}{2m}\left(\log\det(\bar{\boldsymbol{\Lambda}}_p) + \sigma_e^{-2}\|\boldsymbol{t}-\boldsymbol{y}\|_2^2 + \alpha^{-1}\sigma_e^{-2}\|\bar{\boldsymbol{w}}\|_2^2\right),\end{aligned} \tag{30}$$

with $\boldsymbol{\Theta} = \{\mathbb{W}_{L+1}, \alpha, \sigma_e\}$.

To train a NN with BLL and univariate output, we consider in the following the LML and parameters in the form of Result 2. Additionally, the newly introduced parameter α in (27) will play an important role in the following discussion on interpolation and extrapolation and ultimately helps to improve the extrapolative uncertainty.

IV. IMPROVING THE EXTRAPOLATIVE UNCERTAINTY

One of the main challenges of the BLL predictive distribution (8) is that for arbitrary extrapolation points the required assumptions for Lemma 1 will not hold. In this section, we discuss the behavior of the predictive distribution in the interpolation and extrapolation regime and propose a method to improve the performance of NNs with BLL for extrapolation.

To formalize the notion of interpolation and extrapolation, we follow the definition of interpolation described in [25], for which we also need to define the convex hull.

Definition 1 (Convex Hull): The convex hull of a set of samples $\boldsymbol{X} \in \mathbb{R}^{m\times n_x}$ is defined as the set:

$$\mathbb{C}_{\boldsymbol{X}} = \left\{\boldsymbol{X}^\top\boldsymbol{v} | \boldsymbol{v} \in \mathbb{R}^m, \sum\boldsymbol{v} = 1, \boldsymbol{v} \geq 0\right\}.$$

Definition 2 (Interpolation): A sample $\boldsymbol{x}$ is considered to be an interpolation point of a set of samples $\boldsymbol{X}$, given a feature space $\tilde{\boldsymbol{\Phi}}(\boldsymbol{X}; \mathbb{W}_L)$ which satisfies Assumption 1 and 2, if: $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) \in \mathbb{C}_{\tilde{\boldsymbol{\Phi}}(\boldsymbol{X};\mathbb{W}_L)}$.

Interpolation is an attribute of the input space but its definition considers the learned feature space of the neural network. Considering the feature space in Definition 2 may seem counter-intuitive but applies well to reality, where nonlinear features for regression can show arbitrary behavior between data points, even for a univariate input. In this case, interpolation points in the input domain are rightly classified as extrapolation points by considering the feature domain.

Classifying a point as interpolation or extrapolation is a binary decision. In reality this is a shortcoming, as different degrees of extrapolation are possible. That is, a point "close" to the convex hull might still lead to a trustworthy prediction. The distance to a set, e.g. the convex hull, is defined below.

Definition 3 (Distance): For a set $\mathbb{X} \subset \mathbb{R}^{n_x}$ and a point $\boldsymbol{x} \in \mathbb{R}^{n_x}$ we define the distance $d(\boldsymbol{x}, \mathbb{X})$ as:

$$d(\boldsymbol{x}, \mathbb{X}) = \inf_{a\in\mathbb{X}} \|x - a\|_2^2. \tag{31}$$

A. QUANTIFICATION OF INTERPOLATION AND EXTRAPOLATION

In the following, we seek to define a metric to quantify the degree of extrapolation for a nonlinear regression model with feature space. Intuitively, such a metric could be based on the distance, according to Definition 3, to the convex hull of the features. Unfortunately, the distance to the convex hull results in an optimization problem that scales with the number of samples and the feature dimension, which can be a limitation for practical applications. Instead, we propose an approximate metric for which we introduce the *affine cost*. As related concepts of the affine cost, we define the well known *span* and *affine hull*.

Definition 4 (Span): The span of a set of samples $\boldsymbol{X} \in \mathbb{R}^{m\times n_x}$ is defined as the set:

$$\mathbb{S}_{\boldsymbol{X}} = \left\{\boldsymbol{X}^\top\boldsymbol{v} | \boldsymbol{v} \in \mathbb{R}^m\right\}.$$

Definition 5 (Affine Hull): The affine hull of a set of samples $\boldsymbol{X} \in \mathbb{R}^{m\times n_x}$ is defined as the set:

$$\mathbb{A}_{\boldsymbol{X}} = \left\{\boldsymbol{X}^\top\boldsymbol{v} | \boldsymbol{v} \in \mathbb{R}^m, \sum\boldsymbol{v} = 1\right\}.$$

Definition 6 (Affine Cost): The affine cost of a test point $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and given the data $\boldsymbol{X} \in \mathbb{R}^{m\times n_x}$ is defined as the optimal cost:

$$c_{\mathbb{A}_{\boldsymbol{X}}}(\boldsymbol{x}) = \underset{\boldsymbol{v},\boldsymbol{e}}{\text{minimize}} \quad \|\boldsymbol{v}\|_2^2 + \gamma\|\boldsymbol{e}\|_2^2 \tag{32a}$$

$$\text{subject to :} \quad \boldsymbol{X}^\top\boldsymbol{v} + \boldsymbol{e} = \boldsymbol{x}, \tag{32b}$$

$$\sum\boldsymbol{v} = 1, \tag{32c}$$

where the spanning coefficient $\boldsymbol{v} \in \mathbb{R}^m$ and the residual variable $\boldsymbol{e} \in \mathbb{R}^{n_x}$ are optimization variables and $\gamma \in \mathbb{R}$ is a weighting factor for the residuals.

The affine cost naturally complements the definition of the affine hull by computing the norm of the respective spanning coefficients $\boldsymbol{v}$ from Definition 5. Importantly, test values

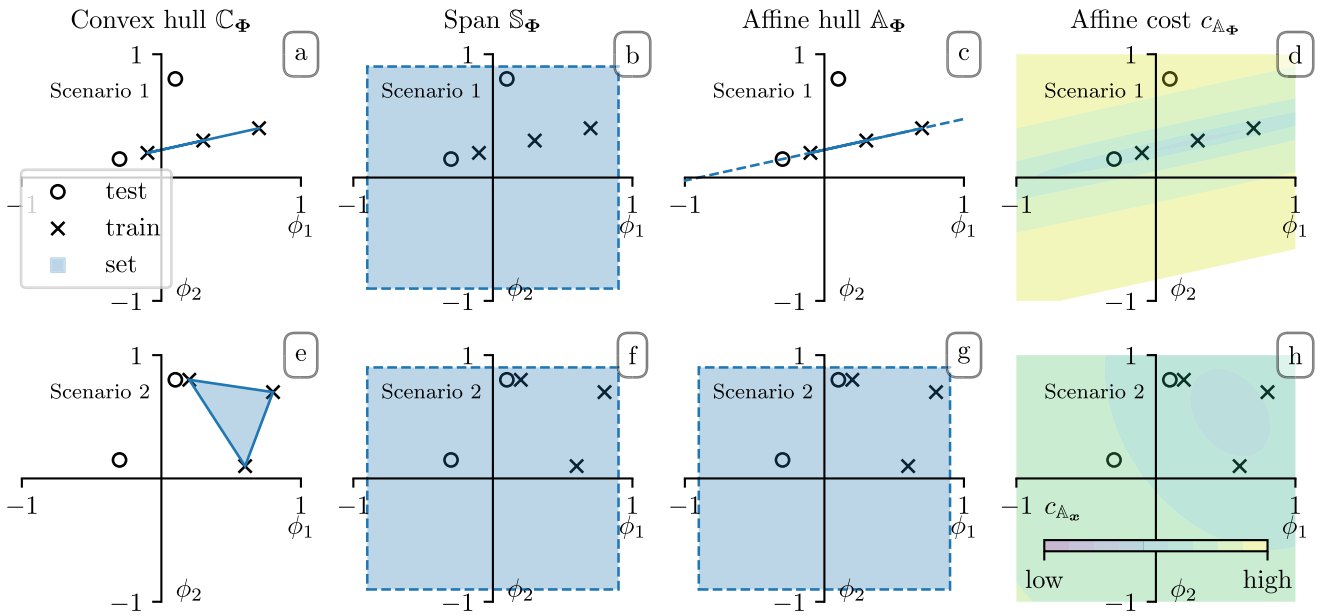

FIGURE 1. Comparison of convex hull (Definition 1), span (Definition 4), affine hull (Definition 5) and affine cost (Definition 6) for two exemplary sets of features $\tilde{\Phi} \in \mathbb{R}^{m \times n_{\tilde{\phi}}}$, both with $m = 3$ and $n_{\tilde{\phi}} = 2$.

$\boldsymbol{x} \notin \mathbb{A}_X$ also have a value assigned, for which the γ-weighted norm of the residuals $\boldsymbol{e}$ is considered.

The relationship of convex hull, span and affine hull as well as the affine cost can be inspected in Figure 1. In this figure, two exemplary sets of features $\tilde{\boldsymbol{\Phi}} \in \mathbb{R}^{m \times n_{\tilde{\phi}}}$, both with $m = 3$ and $n_{\tilde{\phi}} = 2$ are presented and we compare the relationship of test to training samples.

By comparing, for example Figure 1a) and d), we can see that the affine cost has a strong relationship with the convex hull on which we have based the notion of interpolation in Definition 2. In particular, we consider the level set

$$\mathbb{L}_{\mathbb{A}_{\tilde{\Phi}}} = \{\tilde{\boldsymbol{\phi}} \in \mathbb{R}^{n_{\tilde{\phi}}} | c_{\mathbb{A}_{\tilde{\Phi}}}(\tilde{\boldsymbol{\phi}}) \leq l\},$$

obtained with the affine cost and suitable level l as a soft approximation of the convex hull. Such level sets can be seen in Figure 1 d) and h) as the contour lines of $c_{\mathbb{A}_{\tilde{\Phi}}}$. It holds that for a test point $\tilde{\boldsymbol{\phi}}$, the affine cost grows with the distance to the level set $\mathbb{L}_{\mathbb{A}_{\tilde{\Phi}}}$. In this sense, we consider the distance $d(\tilde{\boldsymbol{\phi}}, \mathbb{L}_{\mathbb{A}_{\tilde{\Phi}}})$ and, more directly, the affine cost $c_{\mathbb{A}_{\tilde{\Phi}}}(\tilde{\boldsymbol{\phi}})$ itself as the desired metric for the degree of extrapolation.

For the behavior of this metric we distinguish two important cases. In the first case, small values of the affine cost are achieved for test points that are within the affine hull, i.e. $\tilde{\boldsymbol{\phi}} \in \mathbb{A}_{\tilde{\Phi}}$, and for small distances to the convex hull. According to Definition 2, these include all interpolation points and what we consider mild extrapolation. Both test points in Figure 1 h) are examples for this case.

In the second case, a test point is not within the affine hull $\tilde{\boldsymbol{\phi}} \notin \mathbb{A}_{\tilde{\Phi}}$. The affine cost is now influenced primarily through the parameter γ. This can be seen by inspection of (32), where the residuals $\boldsymbol{e}$ must be used if $\tilde{\boldsymbol{\phi}} \notin \mathbb{A}_{\tilde{\Phi}}$. The cost may then be dominated by the term $\|\boldsymbol{e}\|_2^2$ which is weighted with γ. We argue that in the second case the test point can be considered an extrapolation point and γ can be interpreted as a penalty for extrapolation. This case can be observed in Figure 1 d) for the test point with higher affine cost.

B. RELATIONSHIP OF AFFINE COST AND COVARIANCE

As another main contribution of this work, we introduce Theorem 2 to establish the relationship of the BLL covariance and the previously presented affine cost.

Theorem 2 (BLL Affine Cost): If Assumption 4 holds, and with $\gamma = \alpha$, the affine cost $c_{\mathbb{A}_{\tilde{\Phi}}}(\tilde{\boldsymbol{\phi}}(\boldsymbol{x}))$, according to Definition 6, is equivalent to the scaled BLL covariance (8c):

$$c_{\mathbb{A}_{\tilde{\Phi}}}(\tilde{\boldsymbol{\phi}}(\boldsymbol{x})) = \sigma_e^{-2} \boldsymbol{\Sigma}_{\hat{y}}^{NN}(\boldsymbol{x}), \tag{33}$$

where $\boldsymbol{\phi}$ and $\tilde{\boldsymbol{\phi}}$ describe the features obtained at the last internal layer of the neural network as defined in (4).

Proof: We reformulate the equality constraints of $c_{\mathbb{A}_{\tilde{\Phi}}}(\tilde{\boldsymbol{\phi}})$ shown in (32):

$$\begin{bmatrix} \tilde{\boldsymbol{\Phi}}^\top & \boldsymbol{I}_{n_{\tilde{\phi}}} \\ \boldsymbol{1}_{1,n_m} & \boldsymbol{0}_{1,n_{\tilde{\phi}}} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{e} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{\phi}} \\ 1 \end{bmatrix}, \tag{34}$$

where $\boldsymbol{0}$ and $\boldsymbol{1}$ denote matrices filled with zeros or ones and their respective dimensions are given in the subscript. Considering also (4), we then introduce:

$$\boldsymbol{\Phi}^\top = \begin{bmatrix} \tilde{\boldsymbol{\Phi}}^\top \\ \boldsymbol{1}_{1,n_m} \end{bmatrix}, \ \boldsymbol{M} = \begin{bmatrix} \boldsymbol{I}_{n_{\tilde{\phi}}} \\ \boldsymbol{0}_{1,n_{\tilde{\phi}}} \end{bmatrix}, \ \boldsymbol{p} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{e} \end{bmatrix} \text{ and } \boldsymbol{\phi} = \begin{bmatrix} \tilde{\boldsymbol{\phi}} \\ 1 \end{bmatrix},$$

which allows to reformulate Equation (34) as:

$$\begin{bmatrix} \boldsymbol{\Phi}^\top & \boldsymbol{M} \end{bmatrix} \boldsymbol{p} = \boldsymbol{\phi}. \tag{35}$$

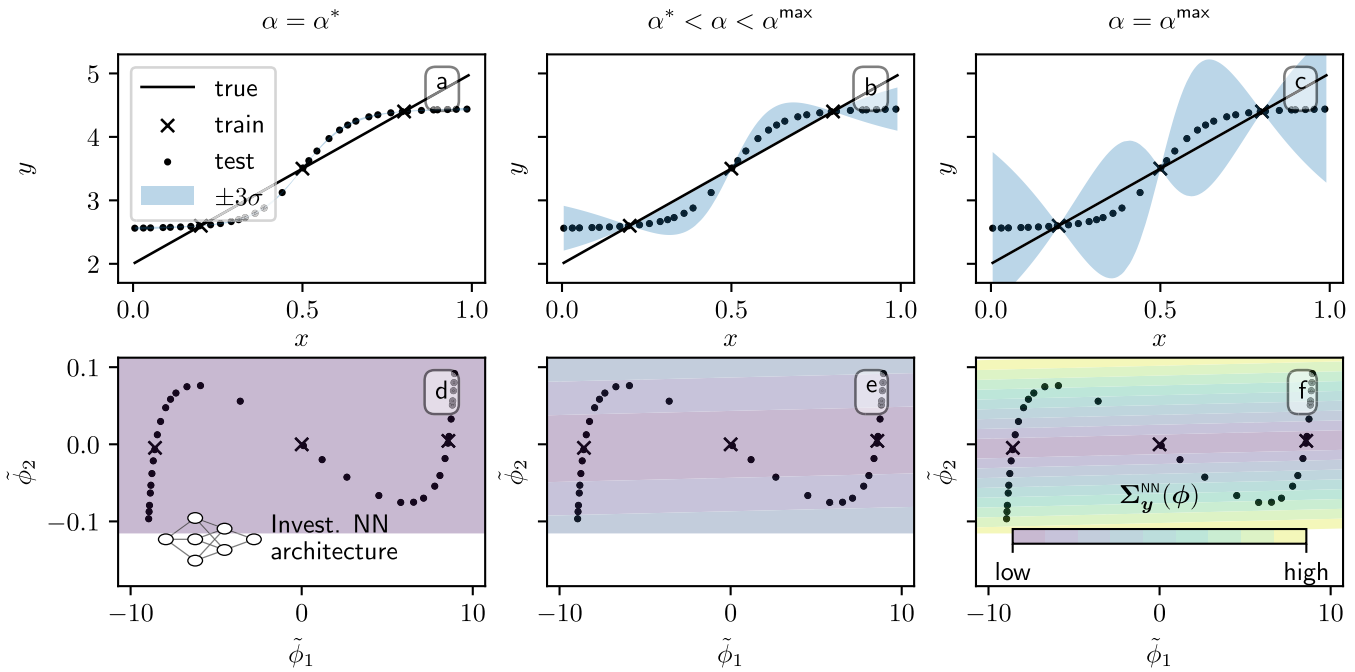

FIGURE 2. NN with BLL: Predicted mean and standard deviation (8) and feature space with $n_{\tilde{\phi}} = 2$ for $m = 3$ training samples. The effect of parameter α on the extrapolation uncertainty is shown by comparing the optimal α^* (maximization of LML (30)) with suggested improved $\alpha^{\max}$.

By further introducing $\boldsymbol{D} = \begin{bmatrix}\boldsymbol{\Phi}^\top & \boldsymbol{M}\end{bmatrix}$ and $\boldsymbol{W} = \text{diag}(\boldsymbol{I}_m,\ \gamma \boldsymbol{I}_{n_{\tilde{\phi}}})$, Problem (32) can be stated as:

$$c_{\mathbb{A}_X}(\tilde{\boldsymbol{\phi}}) = \underset{\boldsymbol{p}}{\text{minimize}} \quad \|\boldsymbol{p}\|_{\boldsymbol{W}}^2 \qquad \text{subject to :} \quad \boldsymbol{D}\boldsymbol{p} = \boldsymbol{\phi}. \tag{36}$$

We have a weighted least-squares problem in (36) for which the solution can be obtained as:

$$\boldsymbol{p}^* = \boldsymbol{\phi}^\top (\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{D}^\top)^{-1}\boldsymbol{D}\boldsymbol{W}^{-1}. \tag{37}$$

Substituting $\boldsymbol{p}^*$ from (37) into the cost function of (36) yields the affine cost:

$$c_{\mathbb{A}_X}(\tilde{\boldsymbol{\phi}}) = \boldsymbol{\phi}^\top (\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{D}^\top)^{-1}\boldsymbol{\phi}, \tag{38}$$

where

$$\begin{aligned}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{D}^\top &= \begin{bmatrix}\boldsymbol{\Phi}^\top & \boldsymbol{M}\end{bmatrix}\begin{bmatrix}\boldsymbol{I}_m & \\ & \gamma^{-1}\boldsymbol{I}_{n_{\tilde{\phi}}}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Phi} \\ \boldsymbol{M}^\top\end{bmatrix} \\ &= \boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \gamma^{-1}\tilde{\boldsymbol{I}}_{n_\phi}.\end{aligned} \tag{39}$$

Considering that $\gamma = \alpha$, as stated in the theorem, and (28)-(29), we have:

$$\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{D}^\top = \boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \alpha^{-1}\tilde{\boldsymbol{I}}_{n_\phi} = \bar{\boldsymbol{\Lambda}}_p = \sigma_e^2\boldsymbol{\Lambda}_p, \tag{40}$$

where $\tilde{\boldsymbol{I}}_{n_\phi} = \text{diag}(\boldsymbol{I}_{n_{\tilde{\phi}}},\ 0)$. Substituting (40) in (38) yields:

$$c_{\mathbb{A}_X}(\tilde{\boldsymbol{\phi}}(\boldsymbol{x})) = \sigma_e^{-2}\boldsymbol{\phi}^\top\boldsymbol{\Lambda}_p^{-1}\boldsymbol{\phi} = \sigma_e^{-2}\boldsymbol{\Sigma}_{\hat{\boldsymbol{y}}}^{\text{NN}}(\boldsymbol{x}), \tag{41}$$

which concludes the proof. □

Theorem 2 establishes the theoretical relationship between affine cost and BLL covariance. In the next subsection, we discuss how this relationship can be used to improve the extrapolative uncertainty of NNs with BLL.

C. IMPROVING THE UNCERTAINTY QUANTIFICATION WITH THE AFFINE COST INTERPRETATION

The first practical implication of Theorem 2 is that a NN with BLL should indeed be trained by maximizing the marginal likelihood as discussed in Section III. Using the features of a trained NN and then applying BLR, as previously shown in [4] and [24] might lead to suboptimal performance. The reason for this is the log det-regularization of the precision matrix $\boldsymbol{\Lambda}_p$ which arises only in the marginal likelihood cost formulation. This regularization encourages a low rank of the feature matrix $\tilde{\boldsymbol{\Phi}}$, resulting in a proper subspace for the affine hull, that is, $\mathbb{A}_{\tilde{\boldsymbol{\Phi}}} \subset \mathbb{R}^{n_{\tilde{\phi}}}$. Only in this setting can we potentially obtain test points with $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) \notin \mathbb{A}_{\tilde{\boldsymbol{\Phi}}}$, clearly indicating extrapolation through high values of the affine cost. The effect can be seen in Figure 1 by comparing subplot d) and h). The application of log det-regularization to obtain matrices with low rank is well known [27] and also applied in other fields such as compressed sensing [28].

The second important implication of Theorem 2 is the interpretation of the parameter α (or γ respectively) in the context of the affine cost from Definition 6. The parameter directly controls the affine cost and thus the variance for test points $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) \notin \mathbb{A}_{\tilde{\boldsymbol{\Phi}}}$. This causes a dilemma: Naturally, we have the situation that all training samples $\tilde{\boldsymbol{\Phi}}$ are within the affine hull of themselves. Therefore, extrapolation in the sense of $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) \notin \mathbb{A}_{\tilde{\boldsymbol{\Phi}}}$ does not occur during training and α^*, which maximizes the LML, might not yield desirable results.

To illustrate the issue we present a simple regression problem with $n_x = n_y = 1$ and $m = 3$ samples. We investigate a NN with $n_{\tilde{\phi}} = 2$ which allows for a

graphical representation of the feature space. The NN is trained by maximizing the LML (30), yielding the optimal parameters $\boldsymbol{\Theta}^*$, which includes α^*. The predicted mean and standard deviation for the trained model using α^* can be seen in Figure 2 a). The predicted mean of the NN, in light of the sparse training data, is suitable. However, the variance shows that the prediction is overconfident. We show in Figure 2 b) and c) that this overconfidence can be tackled simply by increasing α relative to the optimal value α^*.

The reason for this effect of α on the extrapolation uncertainty can be seen by inspecting Figure 2 d)-f), where the features (recall $n_{\tilde{\phi}} = 2$) for the test ($\tilde{\boldsymbol{\phi}}$) and the training points ($\tilde{\boldsymbol{\Phi}}$) are displayed.

As desired, we have that $\mathbb{A}_{\tilde{\boldsymbol{\Phi}}} \subset \mathbb{R}^{n_{\tilde{\phi}}}$, that is, the training features lie in a subspace of $\mathbb{R}^2$, which can be seen in Figure 2 where the training samples in the feature space could be connected by a straight line. This effect can be attributed to the log det-regularization in the LML. Extrapolation thus occurs for $\tilde{\boldsymbol{\phi}} \notin \mathbb{A}_{\tilde{\boldsymbol{\Phi}}}$ and for these points the extrapolative uncertainty grows with increasing α. Importantly, by considering Definition 2, we also have extrapolation for test points that are within the convex hull of the input space, i.e. $\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{X}}$. In this example, the only true interpolation points are the training samples for which increasing α has no significant effect on the predicted variance.

In Figure 3, we further investigate the effect of increasing α for the same regression problem and NN architecture as displayed in Figure 2. To quantify the quality of the predictive distribution, we use the log-predictive density (LPD) [29]:

$$\log \bar{p}(\boldsymbol{t}^{\text{test}}) = \frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} \log p(\mathrm{t} = t_i^{\text{test}} | \mathcal{D}, \boldsymbol{\Theta}, \boldsymbol{x}^{\text{test}}), \tag{42}$$

which evaluates the logarithm of the posterior distribution of the targets (11) for all test values and computes the average thereof.

In Figure 3, we display log-predictive density (42) and the negative LML (30) for the optimal parameters $\boldsymbol{\Theta}^*$ and as a function of α. As expected, α^* minimizes the negative LML. We see, however, that the optimal value is almost at a plateau and that increasing α has only a minor effect on the LML. We also see in the top diagram of Figure 3 that the log-predictive density for the training points remains independent of α. On the other hand, we see that the log-predictive density for test points can be increased significantly by setting $\alpha = \alpha^{\max}$. Further increasing α has a negative effect as the predicted posterior distribution becomes increasingly flat.

As a consequence of this investigation we propose Algorithm 1 to obtain a NN with BLL, trained with LML maximization, and enhanced extrapolation uncertainty.

The optimal value for the scalar parameter α in Algorithm 1 can be obtained with a simple bijection approach.

Algorithm 1 LML Optimization With Adapted Extrapolation Penalty α for Enhanced BLL

Require: $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{val}}$
Require: NN structure $(L, n_{a_l}, g_l(\cdot)\ \forall l \in \mathbb{I}_{[1,L]})$
$\boldsymbol{\Theta}^* \leftarrow \arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}; \mathcal{D}^{\text{train}})$ ▷ solve (30)
$\alpha^{\max} \leftarrow \arg\max_{\alpha} \log \bar{p}(\boldsymbol{t}^{\text{val}})$ ▷ solve (42)

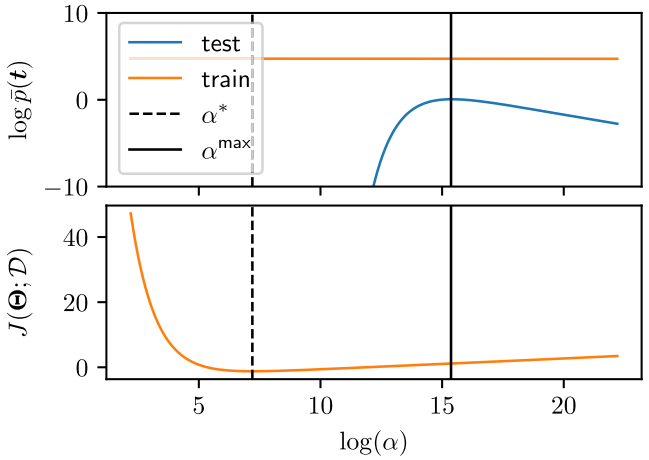

FIGURE 3. Effect of log(α) on the LML for a trained NN and the mean log-predictive density (42). The same regression problem and NN as in Figure 2 is considered.

V. THE MULTIVARIATE CASE

In general, we seek to investigate multivariate problems where, in contrast to Section II, we now have that $\boldsymbol{t} \in \mathbb{R}^{n_y}$. As before, we consider a dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{T}\}$ consisting of m samples and introduce $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m]^\top \in \mathbb{R}^{m \times n_y}$.

To obtain the setting in which Lemma 1 holds, we augment (5) for the multivariate case:

$$\boldsymbol{T} = \boldsymbol{\Phi}\boldsymbol{W} + \boldsymbol{E}, \tag{43}$$

where $\boldsymbol{E} \in \mathbb{R}^{m \times n_y}$ is the matrix of residuals of the regression model. We use the vec($\cdot$) operation as defined in [30, Defn. 11.5] and vectorize (43):

$$\text{vec}(\boldsymbol{T}) = \text{vec}(\boldsymbol{\Phi}\boldsymbol{W}) + \text{vec}(\boldsymbol{E}). \tag{44}$$

Introducing $\otimes$ as the Kronecker product, this equation can be reformulated as [30, Prop. 11.16(b)]:

$$\text{vec}(\boldsymbol{T}) = (\boldsymbol{I}_{n_y} \otimes \boldsymbol{\Phi})\text{vec}(\boldsymbol{W}) + \text{vec}(\boldsymbol{E}), \tag{45}$$

and expressed as:

$$\hat{\boldsymbol{t}} = \hat{\boldsymbol{\Phi}}\hat{\boldsymbol{w}} + \hat{\boldsymbol{\epsilon}}. \tag{46}$$

In the form of (46), Lemma 1 directly applies to the multivariate case where the prior and noise covariances from Assumption 2 and 3 now refer to:

$$\hat{\boldsymbol{\epsilon}} \sim \mathcal{N}(0, \hat{\boldsymbol{\Sigma}}_E),$$
$$\hat{\boldsymbol{w}} \sim \mathcal{N}(0, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}}),$$

and for which we need to consider the multivariate feature matrix $\hat{\boldsymbol{\Phi}}$.

A. SIMPLIFIED TRAINING

The multivariate settings adds significant complexity to the marginal likelihood maximization introduced in Section III. To reduce the computational complexity, we present Result 3 and two required assumptions in the following.

Assumption 5: Noise and prior, as introduced in Assumption 2 and 3, are uncorrelated for all n_y outputs and identical for all m samples and all n_ϕ features, respectively. We denote $\boldsymbol{\sigma}_e = \left[\sigma_{e,1}, \ \ldots, \ \sigma_{e,n_y}\right]$ and $\boldsymbol{\sigma}_w = \left[\sigma_{w,1}, \ \ldots, \ \sigma_{w,n_y}\right]$ and obtain:

$$\hat{\boldsymbol{\Sigma}}_E = \operatorname{diag}(\boldsymbol{\sigma}_e) \otimes \boldsymbol{I}_m, \tag{47}$$

$$\hat{\boldsymbol{\Sigma}}_W = \operatorname{diag}(\boldsymbol{\sigma}_w) \otimes \boldsymbol{I}_{n_\phi}. \tag{48}$$

We introduce $\boldsymbol{\alpha} = \frac{\boldsymbol{\sigma}_w^2}{\boldsymbol{\sigma}_e^2}$, similarly to (27), and assume the following.

Assumption 6: For all predicted outputs y_i, we have the same parameter α, i.e.:

$$\alpha_1 = \cdots = \alpha_{n_y} \overset{!}{=} \alpha. \tag{49}$$

Result 3: Let Assumption 5 and 6 hold. The scaled negative LML in (12) for the vectorized multivariate case (46) is:

$$\begin{aligned} J(\boldsymbol{\Theta}; \mathcal{D}) = \frac{n_y}{2m} \left(m \log(2\pi) + n_\phi \log(\alpha) + \log\det(\bar{\boldsymbol{\Lambda}}_p)\right) \\ + \sum_{i=1}^{n_y} \left(\log(\sigma_{e,i}) + \frac{1}{2m} \sigma_{e,i}^{-2} \|\boldsymbol{t}_i - \boldsymbol{y}_i\|_2^2 \right) \\ + \sum_{i=1}^{n_y} \left(\frac{1}{2m} \alpha^{-1} \sigma_{e,i}^{-2} \|\bar{\boldsymbol{w}}_i\|_2^2 \right), \end{aligned} \tag{50}$$

with $\boldsymbol{\Theta} = \left\{\mathbb{W}_{L+1}, \alpha, \sigma_{e,1}, \ldots, \sigma_{e,n_y}\right\}$ and $\bar{\boldsymbol{\Lambda}}_p$ according to (29). The full precision matrix $\hat{\boldsymbol{\Lambda}}_p$ for the multivariate case can be obtained as:

$$\hat{\boldsymbol{\Lambda}}_p = \operatorname{diag}(\boldsymbol{\sigma}_e^{-2}) \otimes \bar{\boldsymbol{\Lambda}}_p. \tag{51}$$

Proof: The result follows directly from the properties of the Kronecker product [30] applied to the log-marginal likelihood in (12) with (46), as well as Assumption 5 and 6. □

Result 3 shows that the LML can be easily expressed for the multivariate case, allowing for fast and efficient NN training. This is only possible due to Assumption 5 and 6. While Assumption 5 is a natural extension of Assumption 4 for the multivariate case, Assumption 6 might be questioned. We argue again with the interpretation of α as an extrapolation penalty weight, as discussed in Section IV. Importantly, extrapolation, as defined in Definition 2, is a property of the feature space and occurs regardless of the number of outputs.

Apart from simplifying the LML in (50), Assumption 5 and 6 also yield a simplified computation of the predictive distribution. The outputs are uncorrelated due to Assumption 5 and we can obtain independent covariance matrices:

$$\boldsymbol{\Sigma}_{\hat{y}_i}^{\mathrm{NN}}(\boldsymbol{x}) = \boldsymbol{\phi}^\top \boldsymbol{\Lambda}_{p,i}^{-1} \boldsymbol{\phi}, \quad \text{with: } \boldsymbol{\Lambda}_{p,i} = \sigma_{e,i}^{-2} \bar{\boldsymbol{\Lambda}}_p, \tag{52}$$

where we have the same $\bar{\boldsymbol{\Lambda}}_p$ for all outputs due to Assumption 6. Therefore, the complexity of evaluating the predictive distribution scales negligibly with the number of outputs. This is a major advantage of NNs with BLL, in comparison to GPs, where it is common to fit an independent model for each output.

VI. BAYES BY BACKPROP

As a comparative baseline to our proposed method, we also employ variational inference with the *Bayes by Backprop* [1], [21] method to train a full BNN, that is, a neural network in which all weights follow a probability distribution.

A. BACKGROUND

For the formulation of the NN with BLL we require the posterior distribution of the weights in the last layer in (7). As a main difference, we now state the posterior distribution for all weights of the NN, that is:

$$p(\mathbb{W}_{L+1}|\mathcal{D}, \boldsymbol{\Theta}) = \frac{p(\mathcal{D}|\mathbb{W}_{L+1}, \boldsymbol{\Theta}) p(\mathbb{W}_{L+1}|\boldsymbol{\Theta})}{p(\mathcal{D}|\boldsymbol{\Theta})}. \tag{53}$$

In contrast to NNs with BLL, the exact posterior distribution is intractable and we resort to variational inference. To this end, we introduce a surrogate distribution $q(\mathbb{W}_{L+1})$ and minimize the Kullback-Leibler (KL) divergence between the surrogate and the true posterior:

$$\begin{aligned} &D_{\mathrm{KL}}(q(\mathbb{W}_{L+1}) \parallel p(\mathbb{W}_{L+1}|\mathcal{D}, \boldsymbol{\Theta})) \\ &\quad = \mathbb{E}_{q(\mathbb{W}_{L+1})}\left[\log \frac{q(\mathbb{W}_{L+1})}{p(\mathbb{W}_{L+1}|\mathcal{D}, \boldsymbol{\Theta})}\right]. \end{aligned} \tag{54}$$

Minimizing the KL-divergence yields the surrogate distribution of the weights that approximates the true posterior distribution. As proposed in [21], we choose the surrogate distribution as:

$$q(\operatorname{vec}(\boldsymbol{W}_l)) = \mathcal{N}(\boldsymbol{\mu}_{w,l}, \operatorname{diag}(\boldsymbol{\sigma}_{w,l}^2)) \quad \forall l \in \mathbb{I}_{[1,L+1]}, \tag{55}$$

with trainable parameters $\boldsymbol{\mu}_{w,l}$ and $\boldsymbol{\sigma}_{w,l}$. Consequently, each weight of the neural network is now represented with two parameters, which makes Bayes by Backprop a tractable method even for larger models.

For the concrete implementation of training the BNN with Bayes by Backprop, we refer to [1]. In particular, we use Bayes by Backprop in combination with empirical Bayes to obtain the optimal parameters $\boldsymbol{\Theta} = \{\boldsymbol{\sigma}_w, \boldsymbol{\sigma}_e\}$ which, similarly to the NN with BLL, include the prior variances of the weights in the last layer, that is, $\boldsymbol{\sigma}_w$ and the noise covariances $\boldsymbol{\sigma}_e$. Following the discussion in [21], we assume a fixed prior for the weights in the previous layers.

B. EVALUATION

The predictive distribution of the trained BNN can be evaluated as:

$$p(\mathbf{t}|\mathcal{D}, \boldsymbol{\Theta}, \boldsymbol{x}) = \mathbb{E}_{q(\mathbb{W}_{L+1})}\left[p(\mathbf{t}|\boldsymbol{x}, \mathbb{W}_{L+1})\right]. \tag{56}$$

In contrast to predictive distribution of the NN with BLL in (11), expression (56) has no analytical solution. Instead,

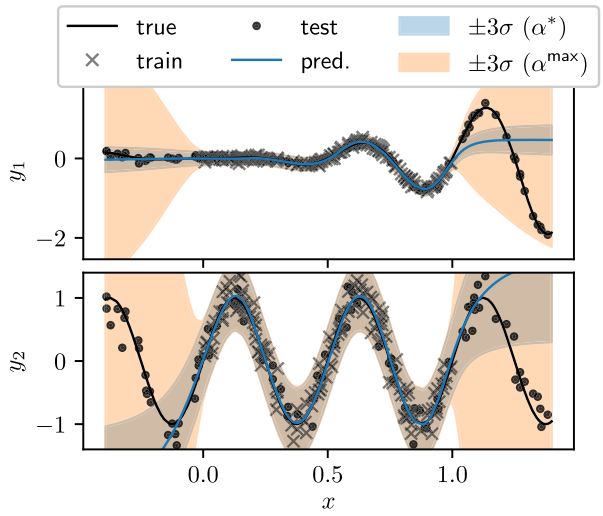

FIGURE 4. Neural network with Bayesian last layer: Mean and standard deviation for two outputs with unknown noise levels. Comparison of the distributions with optimal α^* and $\alpha^{\max}$ from Algorithm 1.

TABLE 1. Simulation study in Figure 4 and 5: Comparison of log-predictive density (LPD) for a NN with BLL (minimized LML), BLR with NN features (minimized MSE) and a BNN (trained with variational inference).

		LPD (↑ is better)	
		train	test
NN w. BLL	α^*	0.42	-18.14
NN w. BLL	$\alpha^{\max}$	0.41	**-1.83**
BLR w. NN features	α^*	0.29	-19.20
BLR w. NN features	$\alpha^{\max}$	0.29	-1.95
BNN w. VI [21]		0.00	-2.30

we resort to Monte Carlo sampling of the weights from the surrogate distribution and obtain the predictive distribution by averaging over the samples, that is:

$$p(\mathbf{t}|\mathcal{D}, \boldsymbol{\Theta}, \boldsymbol{x}) \approx \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{t}|\boldsymbol{x}, \mathbb{W}_{L+1}^{(i)}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{N}(\bar{\boldsymbol{y}}^{(i)}, \boldsymbol{\Sigma}_E), \tag{57}$$

with $\mathbb{W}_{L+1}^{(i)} \sim q(\mathbb{W}_{L+1})$. Equation (57) yields a Gaussian mixture model and also allows for the computation of the log-predictive density, similarly to (42).

VII. SIMULATION STUDY

In this section, we investigate our proposed method in comparison to BLR with features from a trained NN, and a full BNN trained with Bayes by Backprop. The code to reproduce the presented results is available online[1].

[1]https://github.com/4flixt/2023_Paper_BLL_LML

We investigate data from a function with $n_x = 1$ input and $n_y = 2$ outputs, which is shown in Figure 4. Both outputs exhibit non-linear behavior with different noise-levels, in particular $\sigma_1 = 0.05$ and $\sigma_2 = 0.2$. Both noise variances are assumed to be unknown.

We train a NN with BLL using our proposed Algorithm 1. For the study, a suitable structure with $L = 2$, $n_{\tilde{\phi}_1} = n_{\tilde{\phi}_2} = n_{\tilde{\phi}_3} = 20$ and $g_1(\cdot) = g_2(\cdot) = g_3(\cdot) = \tanh(\cdot)$ is determined using trial and error. To maximize the LML, we use the Adam [31] optimizer and employ early-stopping to avoid overfitting.

The results of the simulation study are shown in Figure 4. We visualize mean and standard deviation of the predicted distribution obtained from the NN with BLL. A focus of the investigation is the comparison of the distributions obtained with optimal value α^* and $\alpha^{\max}$. Both distributions suitably describe the training data and capture the behavior of the unknown function in the interpolation regime. The estimated noise variance $\sigma_1 = 0.051$ and $\sigma_2 = 0.17$ is close to the true value. However, only the distribution with tuned extrapolative uncertainty, that is, with $\alpha^{\max}$, is suitable to describe the extrapolation regime. To quantify the improvement, we compute the log-predictive density of the NN with BLL for α^* and $\alpha^{\max}$, and present the results in Table 1. For comparison, we train a NN with the same structure by minimizing the mean-squared-error and then use the learned features for BLR as in previous works [4], [24]. The second step of Algorithm 1, that is, updating α subsequently with validation data, can also be applied in this setting and the resulting performance metric is also shown in Table 1.

The results in Table 1 show that updating α, as proposed in Algorithm 1, significantly improves the log-predictive density (42) of the test data, both for BLL and BLR. We see that the NN with BLL, which is trained according to Algorithm 1, achieves overall the best results with respect to the log-predictive density.

A. COMPARISON TO BAYES BY BACKPROP

As a final step of our evaluation, we compare the NN with BLL to a full BNN trained with Bayes by Backprop, as described in Section VI. We investigate the same neural network architecture as described in the previous subsection and, as a main difference, now have distributed weights in all layers. We determine a fixed prior distribution for the weights in all but the last layer as:

$$p(\mathrm{vec}(\boldsymbol{W}_{l,0})) = \mathcal{N}(0, 0.5\boldsymbol{I}) \quad \forall l \in \mathbb{I}_{[1,L]}. \tag{58}$$

Setting the mean of the prior distribution to zero is justified by employing batch normalization [32] after each variational layer. As for the NN with BLL, the prior variance of the weights in the last layer is also learned.

Training the BNN with Bayes by Backprop is performed as outlined in Section VI and using the Adam optimizer. The resulting predictive distribution is obtained by sampling

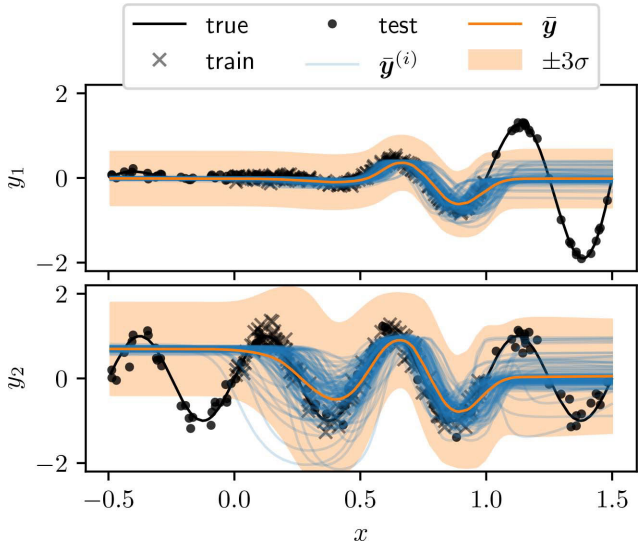

FIGURE 5. Variational inference for a BNN with Bayes by Backprop. Sampled ($N = 100$) predictive distribution with (57).

$N = 100$ weights from the surrogate distribution and displayed in Figure 5. We show the mean values $\bar{y}^{(i)}$ that are obtained with the sampled weights $\mathbb{W}_{L+1}^{(i)} \sim q(\mathbb{W}_{L+1})$. Additionally, we compute mean and variance of the resulting Gaussian mixture model in (57) and display them in the same figure. Qualitatively, the resulting distribution suitably describes the training data. However, the predicted variance does not increase significantly in the extrapolation regime. For a quantitative comparison, we compute the log-predictive density of the Gaussian mixture model and report the results in Table 1. For test data, the BNN trained with Bayes by Backprop achieves a higher LPD than the NN with BLL and the BLR with NN features, both with optimal value α^*. However, the LPD of these methods significantly exceeds the results of the BNN after tuning α with additional validation data, as proposed in Algorithm 1. With the resulting $\alpha^{\max}$, the overall highest LPD on test data is achieved with the NN with BLL. For a BNN trained with Bayes by Backprop it is not possible to modify α to calibrate the extrapolation behavior of the model. Furthermore, it is necessary to sample weights from the surrogate posterior to obtain the predictive distribution, adding significant complexity to this method. This is in contrast to the NN with BLL, where the predictive distribution can be computed analytically.

VIII. CONCLUSION

Neural networks with Bayesian last layer are an attractive compromise between tractability and expressiveness in the field of Bayesian neural networks. As a main contribution of this work, we propose an efficient algorithm for training neural networks with Bayesian last layer, based on a reformulation of the marginal likelihood. Importantly, we show that our reformulation satisfies the conditions of optimality for the same argument as the original formulation. The reformulated marginal likelihood can be efficiently minimized with gradient descent. In future work, we seek to extend our method to also enable stochastic gradient descent.

In comparison to Bayesian linear regression with features from a previously trained neural network, we find that training on the log-marginal likelihood shows advantages in a presented simulation study. Our second contribution, an algorithm to tune the extrapolative uncertainty also shows excellent results in the simulation study. To derive this algorithm, we contribute a discussion on the relationship of the Bayesian last layer covariance and a proposed metric to quantify extrapolation. This metric intuitively relates to the definition of extrapolation which is based on the convex hull. Our proposed method also compares favorably to a full Bayesian neural network trained with Bayes by Backprop. Important advantages are the overall ease of implementation, the possibility to tune the extrapolative uncertainty, and an analytical expression for the predictive distribution.

Especially the last aspect is pivotal for our future work, the identification of probabilistic dynamic system models that can be used for safe control decisions.

REFERENCES

[1] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on Bayesian neural networks—A tutorial for deep learning users," *IEEE Comput. Intell. Mag.*, vol. 17, no. 2, pp. 29–48, May 2022.

[2] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Inf. Fusion*, vol. 76, pp. 243–297, Dec. 2021.

[3] J. Harrison, A. Sharma, R. Calandra, and M. Pavone, "Control adaptation via meta-learning dynamics," in *Proc. Workshop Meta-Learn. (NeurIPS)*, 2018, pp. 1–6.

[4] C. D. McKinnon and A. P. Schoellig, "Meta learning with paired forward and inverse models for efficient receding horizon control," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3240–3247, Apr. 2021.

[5] K. P. Wabersich and M. N. Zeilinger, "Nonlinear learning-based model predictive control supporting state and input dependent model uncertainty estimates," *Int. J. Robust Nonlinear Control*, vol. 31, no. 18, pp. 8897–8915, Dec. 2021.

[6] J. Dong, "Robust data-driven iterative learning control for linear-time-invariant and Hammerstein–Wiener systems," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 1144–1157, Feb. 2023.

[7] C. D. McKinnon and A. P. Schoellig, "Learning probabilistic models for safe predictive control in unknown environments," in *Proc. 18th Eur. Control Conf. (ECC)*, Jun. 2019, pp. 2472–2479.

[8] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using Gaussian process regression," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 6, pp. 2736–2743, Nov. 2020.

[9] C. E. Rasmussen, *Gaussian Processes in Machine Learning*. Cham, Switzerland: Springer, 2003.

[10] M. Lazaro-Gredilla and A. R. Figueiras-Vidal, "Marginalized neural network mixtures for large-scale regression," *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1345–1351, Aug. 2010.

[11] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When Gaussian process meets big data: A review of scalable GPs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4405–4423, Nov. 2020.

[12] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, "Deep Kernel Learning," in *Proc. 19th Int. Conf. Artif. Intell. Statist.*, 2016, pp. 370–378.

[13] H. Liu, Y.-S. Ong, X. Jiang, and X. Wang, "Deep latent-variable kernel learning," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10276–10289, Oct. 2022.
[14] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[15] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3866–3878, Sep. 2020.
[16] L. Dong, J. Yan, X. Yuan, H. He, and C. Sun, "Functional nonlinear model predictive control based on adaptive dynamic programming," *IEEE Trans. Cybern.*, vol. 49, no. 12, pp. 4206–4218, Dec. 2019.
[17] J. Sarangapani, "System identification using discrete-time neural networks," in *Neural Network Control of Nonlinear Discrete-Time Systems*, 1st ed. Boca Raton, FL, USA: CRC Press, 2006, pp. 443–466.
[18] F. Fiedler, A. Cominola, and S. Lucia, "Economic nonlinear predictive control of water distribution networks based on surrogate modeling and automatic clustering," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 16636–16643, 2020.
[19] F. Fiedler and S. Lucia, "Model predictive control with neural network system model and Bayesian last layer trust regions," in *Proc. IEEE 17th Int. Conf. Control Autom. (ICCA)*, Jun. 2022, pp. 141–147.
[20] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using Markov chain Monte Carlo," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 880–887.
[21] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1613–1622.
[22] J. Watson, J. A. Lin, P. Klink, J. Pajarinen, and J. Peters, "Latent derivative Bayesian last layer networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 1198–1206.
[23] C. M. Bishop, *Pattern Recognition and Machine Learning*. Cham, Switzerland: Springer, 2006.
[24] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable Bayesian optimization using deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2171–2180.
[25] R. Balestriero, J. Pesenti, and Y. LeCun, "Learning in high dimension always amounts to extrapolation," 2021, *arXiv:2110.09485*.
[26] J. M. Bernardo and A. F. Smith, *Bayesian Theory*, vol. 405. Hoboken, NJ, USA: Wiley, 1994.
[27] M. Fazel, H. Hindi, and S. P. Boyd, "Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices," in *Proc. Amer. Control Conf.*, Jun. 2003, pp. 2156–2162.
[28] W. Dong, G. Shi, X. Li, Y. Ma, and F. Huang, "Compressive sensing via nonlocal low-rank regularization," *IEEE Trans. Image Process.*, vol. 23, no. 8, pp. 3618–3632, Aug. 2014.
[29] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proc. 24th Int. Conf. Mach. Learn.*, Jun. 2007, pp. 393–400.
[30] George A. F. Seber," *A Matrix Handbook for Statisticians*. Hoboken, NJ, USA: Wiley, 2008.
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

FELIX FIEDLER received the M.Sc. degree in chemical engineering from TU Berlin University, Germany, in 2018. He was Ph.D. candidate at TU Berlin University, HEIBRiDS Graduate School program, until 2020. In the same year, he joined TU Dortmund University, Germany, where he is currently pursuing the Ph.D with the Chair of Process Automation Systems. He is also the current lead developer of the open-source do-mpc toolbox which enables rapid non-linear MPC developments in Python (www.do-mpc.com). His research interests include model predictive control under uncertainty, system identification, and machine learning.

SERGIO LUCIA (Member, IEEE) received the M.Sc. degree in electrical engineering from the University of Zaragoza, Spain, in 2010, and the Dr.-Ing. degree in optimization and automatic control from TU Dortmund University, Germany, in 2014. He joined Otto-von-Guericke Universität Magdeburg and visited the Massachusetts Institute of Technology as a Postdoctoral Fellow. From 2017 to 2020, he was an Assistant Professor with TU Berlin. Since 2020, he has been a Professor with TU Dortmund University and the Head of the Chair of Process Automation Systems. His research interests include decision-making under uncertainty, distributed control, and the interplay between machine learning techniques and control theory. He is currently an Associate Editor of the *Journal of Process Control*.