BNN Training as State Estimation: A Progressive Filtering Approach

Leon Winheim[1] and Uwe D. Hanebeck[2]

***Abstract*— Standard deterministic neural networks produce point estimates as prediction results. Although these results are sufficient in many fields, critical applications require computational models to make an uncertainty statement in combination with their prediction. Bayesian neural networks (BNNs) provide this combination by modeling uncertainty and yielding a predictive distribution instead of a point estimate. The parameters of a BNN are random variables and therefore standard training methods cannot be used directly. The training of a BNN can be understood as a problem of Bayesian inference, where filtering methods are a common approximate solution. This paper reviews existing filter-based training approaches and introduces implementation of and results from training a BNN with a Progressive Gaussian Filter.**

I. INTRODUCTION

Context: Traditional system modeling relies on explicit mathematical equations derived from domain knowledge. Machine learning (ML) offers an alternative by learning surrogate models, such as neural networks (NNs), directly from observed data [1]. However, standard NNs provide only point estimates and lack uncertainty quantification. In critical applications like, e.g., medical imaging, a point estimate is not sufficient [2]. Bayesian neural networks (BNNs) address this by modeling weights as random variables, resulting in probabilistic outputs that quantify prediction uncertainty. This distinction is illustrated in Figure 1.

The probabilistic nature of BNNs requires specialized training methods, as conventional optimization techniques are not directly applicable to learning distributions over weights. In particular, the training of a BNN constitutes a problem of *Bayesian inference* [2]. The goal can be formulated as inferring the *posterior distribution* over the weights that incorporates the training data into a *prior distribution*.

Several classes of methods exist for solving Bayesian inference problems; one such class for sequential problems is filtering [3]. The *Kalman filter* is a well-known filtering method for linear systems; however, numerous other filtering techniques exist, some of which have been applied to the training of BNNs. This paper aims to advance filter-based BNN training by using a *Progressive Gaussian Filter*.

Contributions: This work presents a review of filter-based training methods for BNNs, with focus on their underlying assumptions. We propose and implement a novel BNN training algorithm based on *Progressive Gaussian Filtering*, and evaluate its performance, particularly in low-data regimes. Furthermore, we investigate the impact of different sample generation strategies—namely, i) random sampling and ii) deterministic sampling based on *localized cumulative distributions*.

All authors are with the Intelligent Sensor-Actuator-Systems Laboratory (ISAS), Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology (KIT), Germany leon.winheim@kit.edu, uwe.hanebeck@kit.edu.

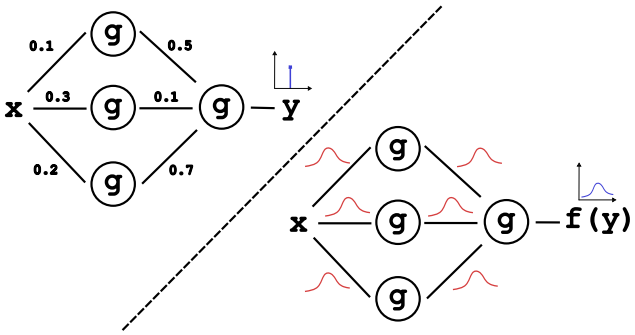

Fig. 1. Conventional neural network on the left vs. Bayesian neural network on the right. "g" denotes the activation function.

After outlining the problem statement, the paper presents a survey of related work on filter-based BNN training. This is followed by a discussion of the progressive filtering approach and its application in a BNN. The paper concludes with experimental results and a discussion.

II. PROBLEM FORMULATION

A. Training as Bayesian Inference

This work addresses the problem of training BNNs, which can be formulated as a problem of *Bayesian inference*. In this framework, a hidden, non-observable state must be estimated from observations of a related quantity [4]. The mathematical foundation for Bayesian inference is *Bayes' Law*

$$f(\underline{\theta}|\mathcal{D}) = \frac{\mathcal{L}(\mathcal{D}|\underline{\theta}) f(\underline{\theta})}{\int \mathcal{L}(\mathcal{D}|\underline{\theta}) f(\underline{\theta}) d\underline{\theta}} \ ,$$

where $\underline{\boldsymbol{\theta}}$ is the random vector of the state, $\mathcal{D}$ denotes the observed data and $f(\cdot)$ denotes the probability density function of the respective random variable. The *prior* distribution over the state $f(\underline{\theta})$ encodes any prior knowledge about the state before seeing any data. The term $\mathcal{L}(\mathcal{D}|\underline{\theta})$ is called *likelihood* of the data given a specific state parametrization and the integral term in the denominator is the *evidence*, which is a measure of explanatory power of the model regarding all its possible parameterizations. Bayes' law relates these values to the *posterior distribution* $f(\underline{\theta}|\mathcal{D})$, which is the desired inference result [3]. In the case of BNN training, the distribution over the network weights constitutes the state,

the training data is the observed data, and the likelihood expresses the probability of the observed outputs given the network's predictions based on those weights.

Bayes' law provides a closed-form expression for the posterior distribution. In practical scenarios, the integral term in the denominator is often analytically intractable. Therefore, approximate methods must be employed.

Two of the most widely used classes of approximate methods are *Markov Chain Monte Carlo* (MCMC) and *Variational Inference* (VI). MCMC methods allow obtaining samples from the posterior under certain conditions. One example is the *Metropolis*-Algorithm and its descendants like *Hybrid Monte Carlo*, that are especially suited for situations that arise from Bayes' Law, where the main problem is the computation of the evidence [5]. Methods of *Variational Inference* reformulate the task into an optimization problem. The parameters of a *variational distribution* are chosen in a way that maximizes the *evidence lower bound* (ELBO) to the true posterior. Maximizing the ELBO is a tractable version of minimizing the *Kullback-Leibler-divergence* (KL), which is a measure of divergence between two probability distributions [4]. These methods are in general popular for BNN training, but will play a subordinate role in this publication.

B. Filtering Methods

Filtering methods represent another class of approaches for problems of Bayesian inference [3]. To clarify their relevance to the present topic, it is useful to frame BNN training as a state estimation problem for a (quasi-) dynamic system defined by its generative system and measurement equations

$$\begin{aligned} \underline{y}_k &= \underline{g}(\underline{\boldsymbol{\theta}}) + \underline{\boldsymbol{\varepsilon}}, \quad \text{Measurement Equation} \\ \underline{\boldsymbol{\theta}}_{k+1} &= \underline{\boldsymbol{\theta}}_k + \underline{\boldsymbol{\rho}}, \quad \text{System Equation} \\ \underline{\boldsymbol{\varepsilon}} &\sim \mathcal{N}(0, v), \quad \underline{\boldsymbol{\rho}} \sim \mathcal{N}(0, r) \ . \end{aligned}$$

This formulation entails several assumptions. The system state, corresponding to the network weights, is assumed to remain static throughout the training process, apart from additive Gaussian process noise $\underline{\boldsymbol{\rho}}$, which can be used to take time variance of the weights into account. Note that the static nature of the state refers to the classic setup of training on a fixed, closed dataset. In an online-learning scenario, the state may evolve in time as new data becomes available. Additionally, the measurements are assumed to be corrupted by additive Gaussian observation noise $\underline{\boldsymbol{\varepsilon}}$. In the BNN case, this noise resembles the uncertainty in the training data. The measurement equation is nonlinear due to the function g, which represents the neural network model.

In summary, the goal is to estimate a BNNs state $\underline{\boldsymbol{\theta}}$ given the training data $\mathcal{D} = \left\{\underline{y}_k\right\}$. Naturally, each output $\underline{y}_k$ corresponds to a deterministic input, which is omitted from the notation for readability.

In practice, filtering methods are used in a recursive manner, where new data arrives sequentially and the state estimate is updated with each new observation [3]. Filtering algorithms can be categorized based on their assumptions about the system and measurement equations, the underlying distributions, and the representation of probability densities.

A *Particle Filter* (PF) does not assume a specific shape of distributions or linearity of model equations, but instead is a stochastic approximation of Bayes' Law that uses a set of *particles* (samples) to represent the distributions and perform computations. The filter step in a PF consists of weighting every particle with its corresponding likelihood and then obtaining equally weighted samples from the weighted sample set. The distributions can be arbitrary, as they are nonparametric [3]. Common problems in particle filtering are sample impoverishment and degeneration, which are especially prominent in systems with little dynamic behavior. Besides that, PFs are computationally intensive.

An important class of filtering methods are *Gaussian Filters*. They assume the state distribution to be Gaussian and thus enable simplifications [3]. The *Kalman Filter* is a well-known Gaussian Filter. Importantly, the Kalman Filter additionally makes a second Gaussian assumption on the joint density of state and measurement distributions [3]. This enables fast, closed-form computation, but imposes a strong assumption. Methods such as the *Extended Kalman Filter* and the *Unscented Kalman Filter* make use of different linearization strategies to enable the use of Kalman filters for nonlinear system and measurement equations [3]. An extensive overview of different nonlinear Kalman filter approaches is given in [6]. In the case of a BNN, the joint density may be strongly non-Gaussian and thus the second Gaussian assumption imposed by Kalman filters could be too strong.

The Kalman and Particle Filters are edge cases regarding their underlying assumptions on the system. Several methods exist that impose assumptions on specific components of the problem. For example, *Gaussian Particle Filters* perform the resampling step based on a Gaussian re-approximation of the weighted particles, but no second Gaussian assumption is required [7]. *Progressive filters* such as the *Progressive Gaussian Filter* (PGF) extend this idea by gradually incorporating the likelihood to prevent the mentioned problems in particle filtering. They only make a Gaussian assumption about the state distribution, but no second Gaussian assumption like the Kalman Filter [8]. The filter step is still based on weighting samples with their likelihood.

As the PGF mitigates typical problems of a Particle Filter, but does not make the second Gaussian assumption, it appears suitable for application to the BNN case. After reviewing related work, a more detailed explanation of the algorithm will be presented.

III. RELATED WORK

Several comprehensive surveys on the history of Bayesian neural networks and their training methods are available, e.g., [2] and [4]. For detailed discussions of classical approaches such as Markov Chain Monte Carlo (MCMC) and Variational Inference (VI), the reader is referred to these works.

A. Filter-based Training Methods

Training conventional neural networks with filter-based methods is beyond the scope of this work, examples can be found in [9] and [10]. Instead, we focus on publications that specifically address the BNN case. As previously mentioned, filtering methods can be categorized based on assumptions they make about underlying probability densities and the ways these densities are represented. We emphasize on how the methods differ in this regard.

Probabilistic backpropagation (PBP), introduced in [11], can be understood as an instance of *Assumed Density Filtering*. In PBP, the distribution over the weights is modeled as multiple independent, one-dimensional Gaussians. During the forward pass, moment matching is used to approximate the pre-activation and the post-activation values as Gaussian distributions. To perform the backward pass, the authors use a closed-form update rule based on the partial derivative of the model evidence with respect to the network parameters. The update minimizes the *Kullback-Leibler* (KL) divergence between the collection of Gaussians and the true, intractable posterior [11]. To summarize, PBP relies on Gaussian approximations of all intermediate quantities and updates these distributions by using a rule that minimizes the KL divergence, with all computations performed in closed form. An interesting detail in the original publication is the repeated processing of the input data, which is relatively uncommon in traditional filtering approaches as it can lead to covariance degeneration [11].

Two closely related methods are the *Kalman Bayesian Neural Network* (KBNN) [12] and *Tractable Approximate Gaussian Inference* (TAGI) [13]. Like PBP, both rely heavily on moment matching of Gaussian distributions during the forward pass. While TAGI restricts covariance structure to full independence, KBNN allows correlation between weights associated with a single neuron. Another distinction is how the moments are matched. In the KBNN case, the transformed moments are obtained by applying the *law of the unconscious statistician*. In contrast, TAGI utilizes the moment generating function. The backward pass is based on the Kalman update formula for conditioning on the joint distribution of weights and observations, which is assumed to be *jointly Gaussian*. Especially this second Gaussian assumption enables closed-form processing, but poses a strong limitation on the approximation quality. In summary, KBNN and TAGI rely on Gaussian approximations of every intermediate quantity and use Kalman-based updates for the the weights, similar to PBP performed in closed form. Again, multiple training epochs are a possibility in both methods.

In [14], the use of an *Ensemble Kalman Filter* (EnKF) is suggested. The authors note that the mathematical situation does not fit directly as the system equation of the neural network regarding time is linear while the measurement equation is nonlinear. The EnKF algorithm tolerates a nonlinear system equation, but requires a linear measurement equation. As a solution, a state augmentation is proposed. The forward pass in the EnKF case starts with drawing N samples from the prior distribution of the state. For every sample in the distribution, the combined predicted measurement of a mini-batch is computed to obtain the predicted measurement distribution as samples. From these samples, the empirical mean and covariance of joint measurements and states are estimated. These are processed in a Kalman filter step, applied individually to each state sample. The resulting transformed state samples form the prior for the next mini-batch. The results in [14] are generated based on sequential processing of minibatches with a batchsize of 32 data points. In the aforementioned methods PBP, TAGI, and KBNN, sequential processing refers to the isolated processing of every data point, which is an important difference. In summary, the EnKF-BNN relies on a sample approximation of the state distribution without assuming a specific parametric form, but the update of each state sample is a Kalman filter step based on the sampled covariances. Again, multiple training epochs are conducted on the same data.

The first application of a *Particle Filter* to BNNs can be found in [15]. The authors employ a black-box approach to modeling the BNN, where no sequential structure of the network is exploited. Instead, one particle (one sample) represents a parametrization of a deterministic network. The BNN can thus be seen as an ensemble of deterministic networks. The publication discusses bootstrap filtering, a basic form of a Particle Filter. In dynamic systems, the particles are propagated through the system equation to obtain the next predicted state. Common issues in particle filtering include impoverishment and weight degradation. For weakly (or non)-dynamic systems like the training scenario of a BNN, this is especially a problem. As a solution, the publication offers a method called *HySIR*, which uses gradient information of the error surface in combination with an *Extended Kalman Filter* to move the particle set in an optimal direction in the parameter space.

In [16] and [17], two more particle filter-based algorithms are presented. Instead of viewing the BNN as a black-box, multiple particle filtering steps are applied on layer-wise subproblems. This reduces the dimensionality of the problem. As in [15], movement of the samples is introduced by incorporating gradient information. At prediction time, after every layer, only the empirical mean is passed on to be processed in the next layer. This is a limitation of the method, but eases the handling and algorithmic implementation.

All presented particle filter-based methods utilize mechanisms for movement of the particle set in parameter space to avoid typical challenges associated with resampling-based algorithms. An alternative strategy is progressive filtering, which we introduce and motivate in the next section.

B. Progressive Gaussian Filtering

In [8], a Progressive Gaussian Filter (PGF) using explicit likelihoods is proposed. It can be categorized as a *Gaussian Assumed Density Filter*. Similar to particle filters, measurement updates are performed by weighting a sample-based representation of the state density with the corresponding likelihood. However, unlike particle filters, the PGF introduces

the likelihood gradually. Instead of weighting the particle set with the full likelihood values $\mathcal{L}(\mathcal{D}|\underline{\theta})$, a progression parameter $\gamma \in [0, 1]$ is introduced. It describes the partial likelihood $\mathcal{L}(\mathcal{D}|\underline{\theta})^\gamma$ that is already applied. For small values of γ, only little information is incorporated. If $\gamma = 1$, the progression is finished. After every progression step $\Delta\gamma$ (how many are made can be adaptive), a Gaussian re-approximation of the weighted samples occurs, which is the second key difference to standard particle filters. The progression procedure mitigates the mentioned degeneration and impoverishment problems.

The ability to gradually incorporate the likelihood via progression offers an alternative to designing suitable importance distributions, which often requires nested filtering or gradient-based optimization. However, it is important to note that the PGF imposes assumptions about the density.

IV. PROGRESSIVE NET

As the considered model is a Bayesian neural network trained with a progressive filter, we call it *Progressive Net*.[1] In our experiments, the model was used as an ensemble. This means that the forward pass for one input value consists of as many deterministic network instances as there are ensemble members. Each ensemble member is one sample from the distribution over the weights. The collection of predictions from all ensemble members constitutes the predictive distribution. We assume the distribution over the weights is a fully correlated, multivariate Gaussian distribution, with parameters to be inferred during training. We do not exploit the sequential, layer-wise structure of the network but instead view it as a black box. Obtaining the posterior distribution is the objective of the training procedure shown in Algorithm 1.

Algorithm 1 PROGRESSIVE NET

Generate $j = 1...N$ samples from prior distribution of $\underline{\boldsymbol{\theta}}$ with importance weights $w_0^{(j)} = \frac{1}{N}$
Iterate through $i = 1...M$ training data points
for $i = 1$ to M **do**
  Set $\gamma = 0$
  while $\gamma < 1$ **do**
    Compute (log)-Likelihood $\mathcal{L}(\mathcal{D}_i|\underline{\theta}^{(j)})$
    $\Delta\gamma = -\frac{\log N}{min(\log \mathcal{L}) - max(\log \mathcal{L})}$
    $\gamma = \gamma + \Delta\gamma$
    Partial Likelihood $\mathcal{L}_p = \mathcal{L}(\mathcal{D}_i|\underline{\theta}^{(j)})^{\Delta\gamma}$
    $w^{(j)} = w^{(j)} \cdot \mathcal{L}_p \cdot \frac{1}{\sum w^{(j)} \cdot \mathcal{L}_p}$
    $\hat{\underline{\theta}} = \sum_{j=1}^{N} w^{(j)} \cdot \underline{\theta}^{(j)}$
    $\mathbf{C}_{\underline{\theta}} = \sum_{j=1}^{N} w^{(j)} \cdot (\underline{\theta}^{(j)} - \hat{\underline{\theta}}) \cdot (\underline{\theta}^{(j)} - \hat{\underline{\theta}})^T$
    Obtain new samples $\underline{\theta}^{(j)}$ from $\mathcal{N}(\hat{\underline{\theta}}, \mathbf{C}_{\underline{\theta}})$
    Set $w^{(j)} = \frac{1}{N}$
  end while
end for
return $\underline{\theta}^{(j)}$

[1]The implementation of Progressive Net and the presented examples can be found at `https://github.com/leonwinheim/PROGRESSIVE_NET_MFI25_PUB`

In summary, we process every training point sequentially and once. The assumed Gaussian weight distribution is represented through samples. In the filter step, the samples are weighted with a partial likelihood and no assumption on the joint density is required. For every processed data point we perform multiple intermediate Gaussian re-approximations progressively.

Although in our case the algorithmic realization of the update step resembles a particle filter, the motivation behind a PGF is not a direct statistical approximation of Bayes' law. As mentioned earlier, it is better understood as a Gaussian assumed density filter, in our special case using a sample representation to perform computations. The Gaussian assumption is introduced by intermediate re-approximations in every filter step. There also exist realizations of true particle filters that use a progressive approach, see, e.g., [18], [19].

As every Gaussian re-approximation introduces an error, we aim to reduce their number by using the adaptive step size strategy proposed in [8]. When the likelihood of a training point assigns significant probability mass far from the current distribution over the weights, more intermediate steps are taken to slowly shift the distribution in that direction.

Samples used for density representation can be categorized into two classes: *random* samples and *deterministic* samples. The simplest approach to Gaussian resampling is using a random sample generator. As an alternative, in [8], the use of deterministic samples based on the *Localized Cumulative Distribution* (LCD) is proposed. Deterministic LCD samples exhibit superior space coverage and homogeneity when compared to random samples. Unfortunately, computing them is costly as they are the result of an optimization procedure. To enable their use, a sample set from a standard normal distribution with desired number and dimensionality is generated offline and the adaptation to arbitrary Gaussian distributions is performed by a matrix decomposition method. This preserves optimality and reduces computational cost. In the result section, we compare the performance of both sampling methods on an example.

V. EXPERIMENTS

For all experiments, a prior with diagonal covariance matrix is chosen. Progressive Net is implemented in PyTorch and run on an NVIDIA RTX 500 Ada laptop GPU. The PyTorch-based KBNN implementation from [12] is used and executed on CPU. The MCMC model uses the JAX-based NumPyro library [20] and also runs on CPU.

A. Synthetic Regression

A classic benchmark is the synthetic regression problem defined by the cubic function $y = x^3 + \boldsymbol{v}, \quad \boldsymbol{v} \sim \mathcal{N}(0, 9)$, as it is also used in [12]. The networks consists of a single hidden layer with 100 neurons and ReLU activation. We used 800 training points with an 80/20 train/test split and applied input normalization. All data is processed sequentially and only once. Progressive Net used $25\,000$ LCD samples. MCMC was run with 10 chains and 500 samples per chain. As shown in Figure 2, Progressive Net is very similar in

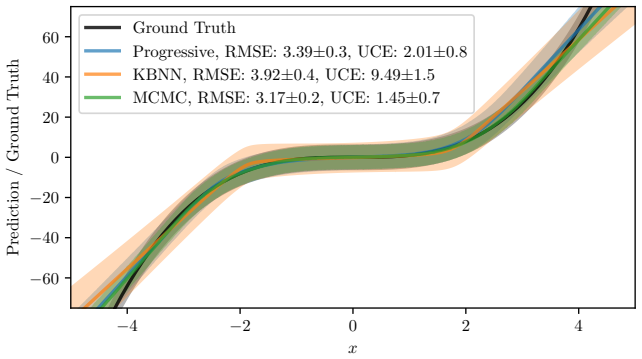

Fig. 2. Comparison of regression results on cubic function.

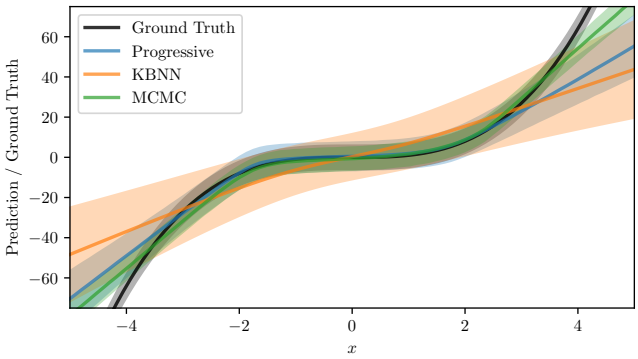

Fig. 4. Cubic regression with 100 training points.

performance to the quality gold standard method MCMC and outperforms KBNN — particularly in terms of uncertainty calibration error (UCE). The mean training times were 1.2 s for KBNN, 27.4 s for Progressive Net and 99.9 s for MCMC.

B. Synthetic Classification

We tested binary classification and especially the online learning capabilities as proposed in [12], using the moon dataset [21]. A 2x10x10x1 ReLU Network with Sigmoid output activation was trained on 1500 samples, achieving $(98.80 \pm 0.01)\,\%$ accuracy. Results are shown in Figure 3. After initial training, the dataset was rotated by 20° and the network was further trained with 100 samples from the rotated set. The model successfully adapted to each rotation step, maintaining accuracy of above 90 % throughout the ten applied rotations.

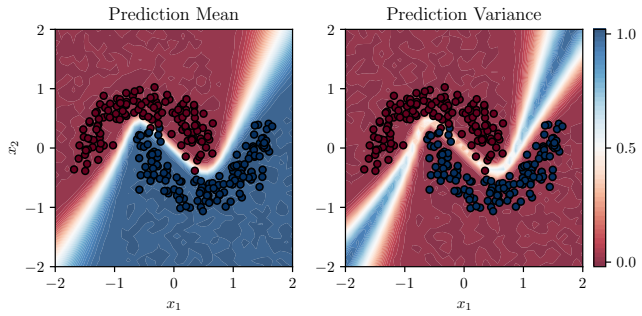

Fig. 3. Moon dataset classification results with Progressive Net.

C. Learning with Limited Data

One notable advantage of Progressive Net is its ability to learn meaningful representations from limited data. As an illustrative example, we present a comparison of the cubic regression problem, but only 100 samples of the training set were used. We perform 10 experimental runs for statistical averaging. In Figure 4, the clear advantage of the Progressive Net over the KBNN is visible.

For the moon classification problem, we conducted similar experiments and the performance of the KBNN and Progressive Net over varying sizes of the training data set is compared in 10 independent experiments for statistical averaging. In Figure 5, the accuracy advantage is observable. Progressive Net reaches superior accuracy with smaller training sets compared to the KBNN.

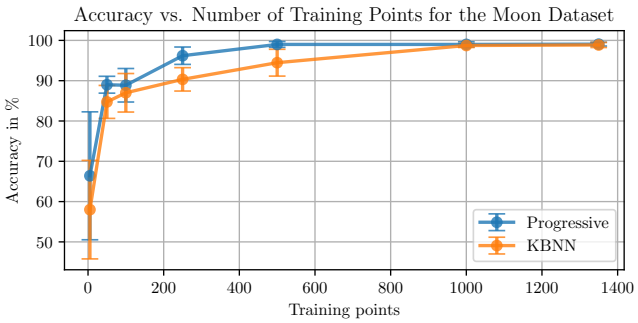

Fig. 5. Evolution of accuracy over training set size, mean and standard deviation.

D. Random vs. Deterministic Samples

We compare the performance of deterministic LCD samples against the performance of random samples in the Progressive Net. For that, we again turn to the cubic toy example, but slightly modify the network structure to have only 20 hidden units instead of 100.

Sample-based methods often face challenges in high-dimensional domains due to the number of samples required for adequate space coverage. Especially in situations like the *Progressive Net* where we apply a potentially complex and narrow likelihood, computation of the empirical, weighted covariance matrix can lead to unstable estimates.

For the original network structure, training with random samples failed entirely for low sample counts. Therefore, a smaller network was selected for this demonstration. Figure 6 shows the performance of training with different sample counts, depending on the type of samples. Training was performed on the same dataset, but shuffled differently 30 times. This results in the variance visible in the charts. For low sample counts, the high uncertainty for random sample results can be explained by sensitivity with respect to the shuffling. The deterministic samples do not exhibit that sensitivity,

TABLE I

PERFORMANCE ON UCI REGRESSION DATASETS.

Dataset	RMSE ↓			NLL ↓			Training Time (s) ↓		
	MCMC	KBNN	Progressive	MCMC	KBNN	Progressive	MCMC	KBNN	Progressive
Concrete	5.81 ± 0.31	8.24 ± 0.69	7.02 ± 0.33	3.17 ± 0.05	3.52 ± 0.09	3.38 ± 0.06	17.04 ± 1.41	1.34 ± 0.28	64.48 ± 3.99
Yacht	0.47 ± 0.15	4.23 ± 0.91	2.35 ± 0.72	-0.32 ± 0.27	3.06 ± 0.05	2.34 ± 0.12	5.86 ± 0.71	0.23 ± 0.02	21.76 ± 2.22
Energy	0.62 ± 0.08	3.42 ± 0.22	3.31 ± 1.04	0.81 ± 0.10	3.36 ± 0.02	2.61 ± 0.19	14.60 ± 1.58	0.87 ± 0.27	60.66 ± 2.61
Power	370.2 ± 23.3	4.15 ± 0.09	4.19 ± 0.09	9.16 ± 1.15	3.44 ± 0.16	2.98 ± 0.04	236.2 ± 7.6	7.18 ± 0.92	454.8 ± 6.7
Kin8nm	0.07 ± 0.00	0.17 ± 0.01	0.10 ± 0.01	-1.17 ± 0.01	0.04 ± 0.26	-0.83 ± 0.08	198.9 ± 3.4	6.61 ± 0.69	514.9 ± 65.9
Naval	2.5e-3 ± 5e-4	0.04 ± 0.01	0.01 ± 2e-4	-3.56 ± 0.10	1.64 ± 0.22	-0.23 ± 1e-4	198.6 ± 20.7	10.42 ± 0.94	641.4 ± 9.4

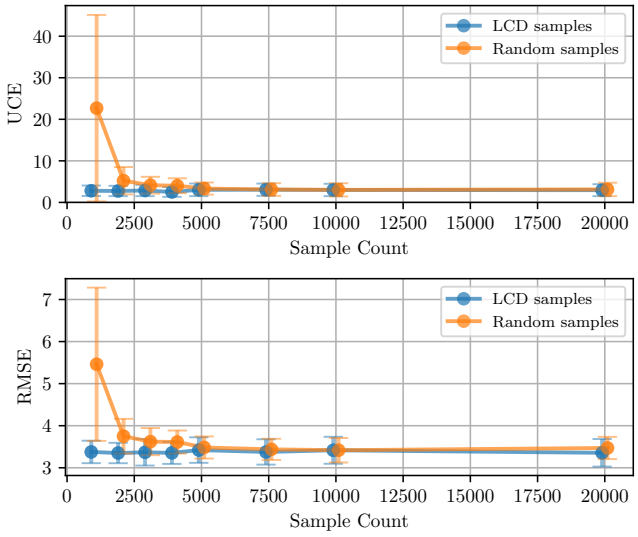

Fig. 6. Performance of the Progressive Net with random and LCD samples, mean and standard deviation.

which is an advantage. Additionally, different random sample sets were used for each run, which also increases the variance. For high sample counts, both methods perform equally well, but significantly fewer samples are required to obtain reliable performance when using LCD-based, deterministic samples. We suggest that this is due to the optimal space coverage of the LCD samples. The apparent convergence of both methods for higher sample counts is to be expected because the accuracy is eventually limited by the expressivity of the Gaussian density itself.

E. UCI Regression Datasets

We evaluated the proposed method on UCI regression datasets [22] and compared it to the performance of KBNN and MCMC. A subset of datasets from [11] was selected, and we adopted their proposed architecture: a single-hidden-layer neural network with 50 ReLU-activated neurons in the hidden layer and linear activation in the output layer. The number of inputs was determined by the respective dataset. Results are shown in Table I. Due to runtime constraints, we limited the MCMC network training to 100 samples, which is in general considered a small number. Despite this, its performance is dominant. The performance of Progressive Net is in between the KBNN and MCMC method, but the training time exceeded that of both methods. Using a more realistic sample count for MCMC would change that scenario, and Progressive Net would likely fall between. Both MCMC and Progressive Net require specification of the likelihood variance as a hyperparameter. Both have the ability to learn it from the data alongside the weights. For the Naval Propulsion dataset, however, Progressive Net failed to learn this parameter effectively, and a grid search had to be performed to identify an optimal value. For the KBNN, a grid search was performed for its hyperparameter for each dataset, as it currently lacks the ability to learn it.

VI. DISCUSSION

We implemented a novel training algorithm for BNNs based on a Progressive Gaussian Filter (PGF) and evaluated its performance across various tasks. The method demonstrated strong results, particularly in low-data regimes. The progressive update approach showed sufficient dynamic capability in the update step. Compared to closed-form methods such as KBNN, Progressive Net has a higher computational cost. However, this cost enables higher-quality results and may be acceptable up to a reasonable scale in scenarios where training on bigger datasets can be done offline. As the cubic regression example demonstrates, Progressive Net can have a benefit in computational efficiency compared to MCMC sampling, particularly when a high number of samples is desired. Importantly, Progressive Net is a highly parallelizable method, allowing for efficient scaling with increased sample counts — especially when larger GPU resources are available. Beyond its performance, the proposed method is a useful application of employing deterministic samples in high-dimensional spaces, where they have a significant advantage. Future work could consider a layer-wise approach to improve scalability or employing the mean-field assumption to simplify the resampling stage.

REFERENCES

[1] Mingi Cho, Jaepil Ban, Minseok Seo, and Sang Woo Kim. “Neural Network MPC for Heating Section of Annealing Furnace”. In: *Expert Systems with Applications* 223 (Aug. 2023), p. 119869.

[2] Martin Magris and Alexandros Iosifidis. “Bayesian Learning for Neural Networks: An Algorithmic Survey”. In: *Artificial Intelligence Review* 56.10 (Oct. 2023), pp. 11773–11823.

[3] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*. 2nd ed. Cambridge University Press, May 2023.

[4] Ethan Goan and Clinton Fookes. “Bayesian neural networks: An introduction and survey”. In: *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018* (2020), pp. 45–87.

[5] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. “An Introduction to MCMC for Machine Learning”. In: *Machine learning* 50 (2003), pp. 5–43.

[6] Jannik Steinbring and Uwe D. Hanebeck. “LRKF Revisited: The Smart Sampling Kalman Filter (S2KF)”. In: *Journal of Advances in Information Fusion* 9.2 (Dec. 2014), pp. 106–123.

[7] J.H. Kotecha and P.M. Djuric. “Gaussian Particle Filtering”. In: *IEEE Transactions on Signal Processing* 51.10 (Oct. 2003), pp. 2592–2601.

[8] Jannik Steinbring and Uwe D. Hanebeck. “Progressive Gaussian Filtering Using Explicit Likelihoods”. In: *Proceedings of the 17th International Conference on Information Fusion (Fusion 2014)*. Salamanca, Spain, July 2014.

[9] Isabelle Rivals and Léon Personnaz. “A Recursive Algorithm Based on the Extended Kalman Filter for the Training of Feedforward Neural Models”. In: *Neurocomputing* 20.1-3 (Aug. 1998), pp. 279–294.

[10] Simon S. Haykin, ed. *Kalman Filtering and Neural Networks*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. New York: Wiley, 2001.

[11] José Miguel Hernández-Lobato and Ryan Adams. “Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks”. In: *International Conference on Machine Learning*. PMLR, 2015, pp. 1861–1869.

[12] Philipp Wagner, Xinyang Wu, and Marco F. Huber. “Kalman Bayesian Neural Networks for Closed-Form Online Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.8 (June 2023), pp. 10069–10077.

[13] James-A Goulet, Luong Ha Nguyen, and Saeid Amiri. “Tractable Approximate Gaussian Inference for Bayesian Neural Networks”. In: *Journal of machine learning research* 22 (2021), pp. 1–23.

[14] Chao Chen, Xiao Lin, Yuan Huang, and Gabriel Terejanu. “Approximate Bayesian Neural Network Trained with Ensemble Kalman Filter”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, July 2019, pp. 1–8.

[15] João Ferdinando Gomes de Freitas. “Bayesian Methods for Neural Networks”. PhD thesis. University of Cambridge, 2003.

[16] Giuseppina Carannante, Nidhal C. Bouaynaya, and Lyudmila Mihaylova. “An Enhanced Particle Filter for Uncertainty Quantification in Neural Networks”. In: *2021 IEEE 24th International Conference on Information Fusion (Fusion)*. Sun City, South Africa: IEEE, Nov. 2021, pp. 1–7.

[17] Giuseppina Carannante, Nidhal Bouaynaya, Lyudmila Mihaylova, and Ghulam Rasool. “BaSIS-Net: From point estimate to predictive distribution in neural networks-a Bayesian sequential importance sampling framework”. In: *Transactions on Machine Learning Research* (2024).

[18] Dominik Prossel and Uwe D. Hanebeck. “Progressive Particle Filtering Using Projected Cumulative Distributions”. In: *2023 IEEE Symposium Sensor Data Fusion and International Conference on Multisensor Fusion and Integration (SDF-MFI)*. Bonn, Germany: IEEE, Nov. 2023, pp. 1–8.

[19] Uwe D. Hanebeck. “Progressive Bayesian Particle Flows Based on Optimal Transport Map Sequences”. In: *Proceedings of the 26th International Conference on Information Fusion (Fusion 2023)*. Charleston, USA, June 2023.

[20] Du Phan, Neeraj Pradhan, and Martin Jankowiak. *Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro*. Dec. 2019.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[22] Arthur Asuncion, David Newman, et al. *UCI machine learning repository*. 2007.