Layer-wise Training of a BNN Based on Progressive Gaussian Filtering

Leon Winheim and Uwe D. Hanebeck

***Abstract*— We investigate different approaches to layer-wise training for a Bayesian neural network (BNN) based on progressive Gaussian filtering. The goal of the layer-wise approach is to reduce the sample count needed to represent the distribution over the weights, as sample-based computation methods suffer from the curse of dimensionality. This is a typical problem in BNNs, where the distribution over the weights easily reaches dimensions that require an impractical amount of samples. We discuss two approaches taken from related methods and introduce a simple version that proves efficient in our experiments. On standard benchmarks, we analyze the relationship of model performance, dimensionality reduction, and sample count to demonstrate the effectiveness of the proposed method.**

I. INTRODUCTION

Although deterministic neural networks (NNs) are a powerful and scalable class of machine learning methods, their lack of inherent uncertainty quantification is a well-known disadvantage [1]. One of the probabilistic extensions of standard deterministic NNs are Bayesian neural networks (BNNs). They model network parameters (weights) as random variables. Predicting with such a model inherently propagates the uncertainty over the weights to the output, allowing for uncertainty quantification. However, standard training methods have to be adapted to be able to account for the distributional character of the parameters. Generally, BNNs are trained by methods of Bayesian inference, which aim to infer the posterior distribution over parameters (in our case network weights) based on a prior distribution and given observed data [2].

Apart from popular approaches of Variational Inference (VI) and Markov Chain Monte Carlo (MCMC), filter-based training methods have been proposed in literature, see, e.g., [3], [4]. One direction of further research in this field is the employment of advanced filtering methods that enable nonlinear updates while still being fast. Progressive Net (PN) was proposed as one such direction, employing a Progressive Gaussian Filter (PGF) for the training [5]. A PGF is a filtering method that assumes a Gaussian state distribution, but performs nonlinear updates while mitigating typical sample-based computation problems of impoverishment and degeneration [6]. At the core, the PGF works by applying a potentially narrow likelihood gradually by performing a progression, which is displayed in Fig. 1 and will be described in detail later. PN showed promising initial results and has potential for further improvements, especially regarding scalability.

The 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.

The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

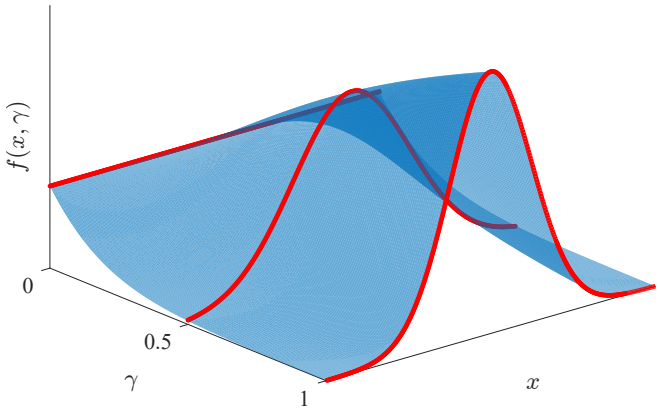

Fig. 1: Progression of a Gaussian likelihood function $\mathcal{L}(x)^{\gamma}$ over γ.

As a method with sample-based computation strategy, it suffers from the curse of dimensionality and it is therefore problematic to scale it up directly. The distribution over the network parameters (the network weights) is a Gaussian distribution that is fully correlated, which means that every weight has a relationship to every other weight. For this Gaussian case, this results in a fully populated covariance matrix. Representing this high-dimensional distribution requires a large number of samples, growing rapidly with the network size due to the curse of dimensionality. To mitigate this issue, we introduce a layer-wise separation of the weight distribution through independence assumptions, reducing the effective dimensionality by decomposing the joint distribution into independent blocks. This allows us to use significantly fewer samples while maintaining comparable predictive performance.

Contributions: We investigate different computational approaches to achieve this separation of the weight distribution in PN and discuss implementation details. Several benchmark tasks are evaluated to demonstrate the effects and relationship between decomposed structure, sample count and performance.

II. PROBLEM FORMULATION

A. BNN Training by Filtering

Following the definition in [2], we require a NN to have two attributes to be called a BNN:

- The weights of the network are modeled as random variables.
- The training of the network is performed by methods of Bayesian inference.

We adopt this definition for this publication, but want to mention that alternative, broader definitions exist in the literature, see [7]. Additionally, we focus on the architectural subset of feed-forward multi layer perceptrons (MLPs).

In deterministic MLPs, training means finding the optimal set of weights that minimizes some loss function on a training dataset by means of optimization methods, often based on gradient descent. In the case of BNNs, training means inferring the posterior distribution $f(\underline{\theta}|\mathcal{D})$ over the random weights $\underline{\boldsymbol{\theta}}$ given a training dataset $\mathcal{D} = \{(\underline{x}_i, \underline{y}_i)\}_{i=1}^{M}$ and a prior distribution over the parameters $f(\underline{\theta})$ that represents prior knowledge about the parameters. Using Bayes' theorem, the posterior can be expressed as

$$f(\underline{\theta}|\mathcal{D}) = \frac{\mathcal{L}(\mathcal{D}|\underline{\theta}) f(\underline{\theta})}{f(\mathcal{D})} \quad , \tag{1}$$

where $\mathcal{L}(\mathcal{D}|\underline{\theta})$ is the *likelihood* of the data given the weights, $f(\underline{\theta})$ is the *prior* distribution over the weights, and $f(\mathcal{D})$ is the *evidence*, which acts as a normalization constant [1]. The likelihood is usually defined based on an assumed noise model of the data generating process. In regression tasks, a Gaussian noise model is often chosen for representing uncertainty in the data, leading to a Gaussian likelihood function. At prediction time, the model is used by marginalizing over the posterior distribution to perform predictions for new inputs, which yields a predictive distribution over the outputs [1].

In practice, approximate computational methods need to be employed. For the training, a variety of approximate Bayesian inference methods are available, with the most prominent members being VI and MCMC [1]. Apart from them, *filtering methods* were successfully applied to this problem [5]. They are usually employed to estimate a probabilistic state of a dynamic system by processing noisy measurements sequentially at each timestep, recursively updating the state distribution with every measurement [8]. To make use of filtering methods for BNNs, the learning task is reframed as a state estimation problem. The weights of the network are treated as the hidden state to be estimated, and the training data points are treated as measurements.

We re-frame the BNN as a state space model defined by

$$\underline{y}_k = \underline{g}(\underline{\boldsymbol{\theta}}, \underline{x}_k) + \underline{\boldsymbol{\varepsilon}}, \quad \text{Measurement Equation} \tag{2}$$

$$\underline{\boldsymbol{\theta}}_{k+1} = \underline{\boldsymbol{\theta}}_k + \underline{\boldsymbol{\rho}}, \quad \text{System Equation} \tag{3}$$

$$\underline{\boldsymbol{\varepsilon}} \sim \mathcal{N}(0, v), \quad \underline{\boldsymbol{\rho}} \sim \mathcal{N}(0, r) \ , \tag{4}$$

where $\underline{g}(\cdot, \cdot)$ describes the forward pass of the BNN, the measurement noise $\underline{\boldsymbol{\varepsilon}}$ captures the uncertainty in the data generation process, while the process noise $\underline{\boldsymbol{\rho}}$ represents a random walk that allows for some flexibility in the weight evolution over time, enabling, e.g., a forgetting factor that is controlled by the character and intensity of the noise. With this formulation, starting from a prior over the weights, various filtering algorithms can be applied to estimate the posterior distribution over the weights $\underline{\boldsymbol{\theta}}$ given the observed data points $\underline{y}_k$ sequentially. For more details we refer to [5].

B. Progressive Net (PN)

PN is a BNN trained by using a PGF and was introduced by the authors of this paper in [5]. The PGF was chosen to enable nonlinear updates while mitigating typical problems that other nonlinear methods like particle filtering have, namely sample impoverishment and degeneration. The use of the PGF brings in some assumptions, the most important being that the distribution over the state vector (i.e., the network weights) is Gaussian. Unlike methods like Kalman Bayesian Neural Networks (KBNN) [3] or probabilistic backpropagation (PBP) [9], the original version in [5] allowed for a fully correlated posterior (we also call this the *black-box* version). As mentioned before, this means that the covariance matrix of the Gaussian distribution over the weights is fully populated, allowing for correlations between all weights in the network. The key characteristic of the PGF framework is the concept of a gradual introduction of the likelihood. The likelihood of a single measurement $\mathcal{L}(\mathcal{D}_i|\underline{\theta})$ is written as $\mathcal{L}(\mathcal{D}_i|\underline{\theta})^{\gamma}$, where γ is an artificial time parameter going from 0 (no information) to 1 (full information) in steps $\Delta\gamma$, which is also displayed in Fig. 1. Samples are drawn from the Gaussian state distribution, and each sample is weighted by the likelihood increment. Then, the set of weighted samples is reapproximated by a Gaussian distribution and the next increment is introduced until $\gamma = 1$. By choosing a suitable adaptive increment size, degeneration is mitigated as the particles are moved slowly towards the relevant domain of the state space and the likelihood fraction remains broad enough to prevent degeneration.

C. Dimensionality in BNNs

Even for what would be considered small models, the dimensionality of the posterior distribution over the weights can become impossible to handle in BNNs, and this is not exclusive to PN. As an example, any two hidden layers with 50 neurons each (plus bias) produce $D = 2550$ dimensions through their connection. Representing such a high-dimensional distribution is hard in practice. When represented as a Gaussian distribution, the covariance matrix over the weights grows quadratically with the number of dimensions and if it is approximated as a fully correlated Gaussian like in PN, the number of parameters in it grow quadratically as well. The internally required matrix decomposition methods even scale with $\mathcal{O}(D^3)$ in the case of Cholesky decomposition. On the other hand, when representing the distribution as samples (which also happens in PN), the number of samples required grows rapidly with dimensionality due to the curse of dimensionality. An important step to improve scalability for PN is therefore to reduce the effective dimensionality through a layer-wise separation of the weight distribution, which decomposes the joint distribution into smaller, independent blocks. We aim to reduce the required sample count significantly while maintaining predictive performance, and reduce the size of the covariance matrices as a byproduct.

III. RELATED WORK

Decomposing the weight distribution into independent blocks has been a common strategy for improving scalability in NN models. The first application of filtering to the training of NNs in literature was presented by Singhal and Wu in the late 1980s [10]. They found the Extended Kalman Filter (EKF) to be an effective alternative to backpropagation, even including second order information by tracking the covariance matrix over the weights. As a consequence, they faced the same issue regarding the full covariance matrix as we do with PN. To improve on scalability, versions with independence assumptions were presented in [11] shortly after.

In McKay's major early application of Bayesian theory to NNs, the importance of regarding the correlations between the weights is also discussed [12]. Shortly after, laying the groundwork for later VI-methods, Hinton and van Camp introduced independent weights for their experiments [13] and therefore employed the mean field approximation to enable larger models.

The sequential Monte Carlo approach from [14] is a non-parametric approach similar to the initial version of PN, which does not force any assumptions on the state distribution. The ensemble Kalman filter-approaches (not to be confused with the EKF) considered in [15], [16], [17] are also non-parametric and do not assume a fixed distribution for the state. Instead, they represent the state by an ensemble of samples, each of which is updated linearly using a common Kalman gain computed from the ensemble covariance.

Early assumed density filtering methods, as employed in PBP, use the mean field approximation with the goal to make the situation computationally tractable [9].

In KBNN [3] and Tractable Approximate Gaussian Inference (TAGI) [18], the authors explicitly designed the procedures to work in a layer-wise fashion. This already limits inter-layer correlations, however, to keep the derivation and computation manageable, they even reduced it to a mean field approximation in TAGI and a pairwise independent assumption in KBNN.

In [4], a particle filtering scheme is employed, but in contrast to the work in [14], a layer-wise approach is taken from the beginning. Based on a multiple particle filtering scheme from [19], the effective dimensionality is kept low and they are able to reduce the number of samples for the state representation.

The literature shows that dimensionality reduction is a common strategy for improving the scalability of NN methods. For each employed filtering method, the actual computation has to be adapted individually, mainly regarding the computations based on samples vs. based on moment propagation. For the PN, the situation presents unique challenges as it combines different aspects of representation by samples *and* moments simultaneously, and additionally handles an internal progression that spans multiple forward passes. Direct employment of the approaches above is not straightforward, which will be investigated in the following section.

IV. LAYER-WISE APPROACHES FOR PROGRESSIVE NET

We present three different approaches to achieve this separation of the weight distribution in PN and discuss the implications specific to each. All methods result in the same separated structure: only weights within each layer are correlated, while weights of different layers are assumed to be independent.

Algorithm 1 PROGRESSIVE NET (FULL layer-wise)

Generate $j = 1...N$ samples from prior distribution of $\underline{\boldsymbol{\theta}}$ with importance weights $w_0^{(j)} = \frac{1}{N}$
Iterate through M training data points and L layers
for $i = 1$ to M **do**
  Perform forward pass to save all $\underline{z}_l^{(j)}$
  for $l = L$ to 1 **do**
    Set $\gamma = 0$
    Joint state $\underline{\boldsymbol{\zeta}} = [\underline{\boldsymbol{\theta}}_l, \underline{\boldsymbol{z}}_l]$
    Consider measurement $\mathcal{D}_i = \underline{z}_{l+1}^{+}$ (except for last layer, where $\mathcal{D}_i = y_i$)
    while $\gamma < 1$ **do**
      Compute (log)-Likelihood $\mathcal{L}(\mathcal{D}_i|\underline{\zeta}^{(j)})$
      $\gamma = \gamma + \Delta\gamma$
      Partial Likelihood $\mathcal{L}_p = \mathcal{L}(\mathcal{D}_i|\underline{\zeta}^{(j)})^{\Delta\gamma}$
      $w^{(j)} = w^{(j)} \cdot \mathcal{L}_p \cdot \frac{1}{\sum w^{(j)} \cdot \mathcal{L}_p}$
      $\hat{\underline{\zeta}} = \sum_{j=1}^{N} w^{(j)} \cdot \underline{\zeta}^{(j)}$
      $\mathbf{C}_{\underline{\zeta}} = \sum_{j=1}^{N} w^{(j)} \cdot (\underline{\zeta}^{(j)} - \hat{\underline{\zeta}}) \cdot (\underline{\zeta}^{(j)} - \hat{\underline{\zeta}})^T$
      Obtain new samples $\underline{\theta}_l^{(j)}$ from partial $\mathcal{N}(\hat{\underline{\zeta}}, \mathbf{C}_{\underline{\zeta}})$ with $\hat{\underline{\boldsymbol{\zeta}}} = [\underline{\boldsymbol{\theta}}_l, \underline{\boldsymbol{z}}_l^{+}]$
      Set $w^{(j)} = \frac{1}{N}$
      Save updated $\underline{z}_l^{(j)}$ for next computation
    end while
  end for
end for
return $\underline{\theta}^{(j)}$

A. Full Layer-Wise

In this first approach, layer-wise partial states $\underline{\boldsymbol{\theta}}_l$ are estimated jointly with intermediate layer results $\underline{\boldsymbol{z}}_l$, which is an adoption of the closed-form approach used in [3]. Basically, the network layers are treated as separate sub-models. A visualization is given in Fig. 2. The black arrows symbolize a forward pass, and the red arrows symbolize the backward pass. It can be seen that from one observed quantity (e.g., $\underline{y}$), two unobserved quantities have to be estimated (here $\underline{\boldsymbol{\theta}}_L$ and $\underline{\boldsymbol{z}}_L$), and this continues for all layers backwards. The updated distribution of $\underline{\boldsymbol{z}}_L$ from the filter step acts as measurement distribution for the preceding layer. The formal procedure is described in Algorithm 1.

Note that $\underline{\boldsymbol{\theta}}_l$ describes the subset of weights that belong to layer l and $\mathbf{z}_l$ describes the input to layer l, and the superscript $()^+$ represents an updated quantity.

The processing of parameters of a single layer is now isolated, but the approach introduces a critical additional

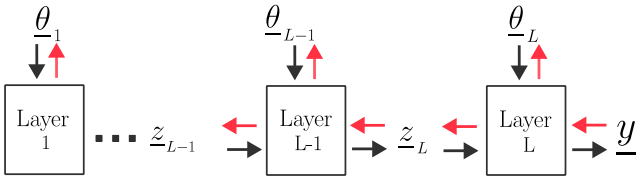

Fig. 2: Representation of the full layer-wise version. The model is separated into single layers, and filtering is performed backwards for each.

assumption: a joint Gaussian distribution is imposed on both the layer weights $\underline{\boldsymbol{\theta}}_l$ and the intermediate activation values $\underline{\boldsymbol{z}}_l$. A key advantage of the original PGF framework is that the Gaussian assumption is placed only on the weight distribution, leaving the activations unconstrained. Forcing Gaussianity on $\underline{\boldsymbol{z}}_l$ introduces a compounding approximation error across the progression steps, as the activation distributions are generally non-Gaussian after nonlinear layers. This proved detrimental in practice: initial experiments showed slow convergence and degraded predictive performance. We therefore do not consider this approach in further evaluations.

B. Naive Layer-wise

In this layer-wise version of PN, instead of estimating the intermediate input values, we perform separate resampling for weights in each layer and treat samples from different layers as independent. Each layer's sub-state is resampled using the same importance weights as in the black-box case. No additional forward passes are needed compared to the original version. The computation is described in Algorithm 2. Again, $\underline{\boldsymbol{\theta}}_l$ describes the subset of weights that belong to layer l.

Algorithm 2 PROGRESSIVE NET (NAIVE layer-wise)

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

This approach effectively reduces the maximum dimensionality of the involved distributions to that of the widest layer in the network. We require L resampling steps per progression step, but each one has reduced dimensionality and they could be parallelized. Inter-layer weight correlations are neglected. Starting from the black-box case, the implementation of this version is straightforward. A sample array is kept in memory in the full dimensionality, meaning it is $(N \times D)$. In the black-box version, the resampling happens for the full array and each sample has D dimensions. In the naive layer-wise version, subsets of the columns that belong to a layer are resampled independently. To keep the computation simple, a limitation here is that the same number of samples has to be used for every layer. Otherwise, the forward pass computation would become significantly more complex as each layer's output would need to have a different number of samples to fit the next layer, which would mean resampling procedures after every forward pass and would in the worst case require a distributional assumption on the post-activation values of each layer. Note that this approach could also be extended to accommodate any desired grouping of weights.

This method is a good trade-off between implementation complexity, computational load, and performance. Therefore, it will be used as the representative of layer-wise methods in the later evaluations.

C. MPF Layer-wise

This version is inspired by the BNN implementation of [4], where a multiple particle filtering (MPF) approach was used that was first introduced in [19]. Unlike [4], our version propagates not only the means but the complete distribution of intermediate outputs forward through each layer. The main idea is to perform a forward pass, update the first layer's weights based on the output observations, then repeat the forward pass and update the next layer until all layers are updated. This results in more forward passes and is therefore computationally more expensive. Each layer's weights are independently resampled and the correlation structure and dimensionality corresponds to the naive version. The formal procedure can be found in Algorithm 3.

While this method is also easy to implement, the increased computational load from multiple forward passes makes it less attractive.

An additional concern is that updating the first layers first and then proceeding to the next ones leads to suboptimal performance as the last layer's updates do not influence the weights of the first ones anymore. This is due to the sequential nature of the NN, which is not the standard case of a multiple particle filtering procedure.

D. Adaptive Step Size Heuristic

In general, the degeneration of samples (having a large variance in sample weights) should be avoided because it results in information loss. The risk of excessive downweighting could be mitigated by using a very small step size in the progression procedure. However, this is problematic because every reapproximation introduces an error. Therefore, heuristic approaches for determining an optimal, adaptive

Algorithm 3 PROGRESSIVE NET (MPF layer-wise)

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

progression step size $\Delta\gamma$ are used in PN. A general approach is presented in [6], where the step size is chosen to be

$$\Delta\gamma = -\frac{\log N}{lb - \max(\log \mathcal{L})} \quad , \tag{5}$$

where originally $lb = \min(\log \mathcal{L})$. As an alternative, we modify this heuristic to be based on quantiles of the likelihoods (PN is based on a Gaussian likelihood assumption) instead of the extrema. This increases robustness against outliers. We experimented with the approach $lb = (\log \mathcal{L})_{0.05}$ which is the $5\,\%$ quantile of the log likelihoods. In preliminary observations, this approach accelerated computation and did not lead to measurable loss in quality. This modification was applied to all versions of PN.

V. EXPERIMENTS

We evaluate the discussed methods on different examples and analyze the influence of reduced dimensionality.[1]

A. Scalar Regression

The goal of the layer-wise extensions of PN is to be able to use fewer samples while keeping prediction quality at the same level. We demonstrate general performance and the achievement of that goal by evaluating the classic scalar cubic regression task that is also described in [5]. The results displayed in Fig. 3 are obtained with a network structure of $[1, 20, 20, 1]$ with hidden ReLU activations. We trained on 800 data points and performed 10 runs with random shuffling and used the root mean squared error (RMSE) as evaluation criterion.

We interpret the observed behavior as the result of two separate mechanisms. On the one hand, for low sample counts,

[1]Code for experiments: `https://github.com/KIT-ISAS/layer_wise_progressive_net`

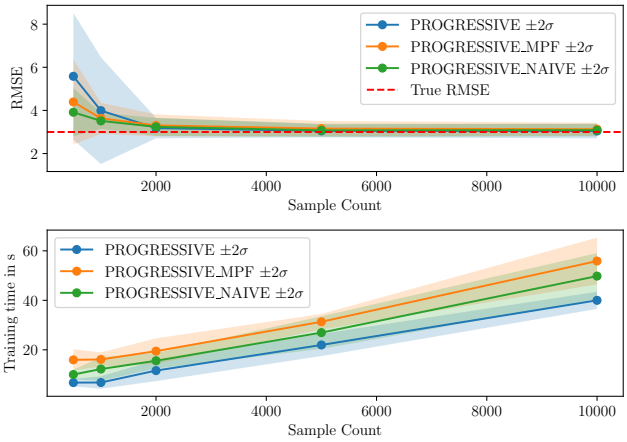

Fig. 3: Comparison of the different approaches on the cubic regression task. "PROGRESSIVE" corresponds to the full black-box version.

the layer-wise methods (except for the full layer-wise method, which is not considered) show lower error metrics than the black-box version. This validates our approach: the separation through layer-wise independence demonstrates that fewer samples suffice when decomposing the joint distribution. On the other hand, the limited expressivity of this separated structure explains the slightly lower error for the black-box method for higher sample counts, although it is not clearly visible in this simple example.

The training time differences also have two driving explanations. First, the layer-wise methods require a higher number of resampling steps and/or forward passes by algorithm design. Second, the full version requires fewer progression steps because the unseparated joint distribution allows incorporating more information per step, which will be discussed in the next subsection.

For real-world problems, one has to consider these trade-offs. For a high-dimensional model, increasing sample count will increase computation time. Using layer-wise methods can decrease the required sample count, but can be slower because of more required progression steps. The nature of the constraints at hand (time or memory capacity) has to be considered in each case.

B. UCI Regression Benchmark

We evaluated different versions of PN on 6 UCI regression datasets [20] regarding RMSE, negative log-likelihood (NLL) metrics, and training times on a cluster using 32-core Intel Xeon Platinum 8358 hardware while PN computations were accelerated by a NVIDIA A100. As a typical performance baseline, we evaluated a BNN trained with MCMC using the complete dataset at once and a posterior sample count of 100 with 100 burn-in steps and the No U-Turn sampler. The evaluated PN variants include the black-box version with random samples, the black-box version with deterministic samples based on the localized cumulative distribution (LCD) (see [5] for an explanation), and the naive layer-wise version with random samples. We use an 80/20 train/test split and perform 10 runs with random shuffling. The likelihood

TABLE I: RMSE and NLL Performance on UCI Regression Datasets (1-Sequential).

			RMSE				NLL			
Dataset	**N**	**d**	**MCMC**	**Prog.**	**Prog. LCD**	**Prog. Naive**	**MCMC**	**Prog.**	**Prog. LCD**	**Prog. Naive**
Yacht	308	6	0.60 ± 0.11	1.90 ± 0.31	1.94 ± 0.37	2.51 ± 0.46	4.78 ± 3.21	1.99 ± 0.12	1.98 ± 0.14	2.18 ± 0.12
Concrete	1030	8	6.31 ± 0.23	9.00 ± 0.42	8.60 ± 0.67	8.45 ± 0.38	3.27 ± 0.04	3.76 ± 0.10	3.72 ± 0.12	3.67 ± 0.07
Energy	768	8	0.69 ± 0.09	2.71 ± 0.13	2.73 ± 0.13	2.86 ± 0.20	1.01 ± 0.18	2.44 ± 0.06	2.45 ± 0.06	2.44 ± 0.07
Kin8nm	8192	4	0.08 ± 0.00	0.16 ± 0.01	0.11 ± 0.01	0.16 ± 0.01	-1.14 ± 0.05	-0.38 ± 0.04	-0.69 ± 0.13	-0.42 ± 0.05
Naval	11934	16	0.00 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	-4.00 ± 0.20	-3.13 ± 0.32	-3.22 ± 0.34	-3.04 ± 0.24
Power	9568	4	3.99 ± 0.11	4.23 ± 0.12	4.15 ± 0.12	4.21 ± 0.11	2.81 ± 0.03	2.86 ± 0.03	2.85 ± 0.04	2.86 ± 0.03

TABLE II: RMSE and NLL Performance on UCI Regression Datasets (100-Sequential).

			RMSE				NLL			
Dataset	**N**	**d**	**MCMC**	**Prog.**	**Prog. LCD**	**Prog. Naive**	**MCMC**	**Prog.**	**Prog. LCD**	**Prog. Naive**
Yacht	308	6	0.60 ± 0.11	1.34 ± 0.31	1.33 ± 0.27	1.69 ± 0.37	4.78 ± 3.21	1.82 ± 0.08	1.81 ± 0.07	1.98 ± 0.11
Concrete	1030	8	6.31 ± 0.23	6.65 ± 0.31	6.83 ± 0.28	7.43 ± 0.30	3.27 ± 0.04	3.55 ± 0.12	3.62 ± 0.11	3.73 ± 0.10
Energy	768	8	0.69 ± 0.09	2.57 ± 0.15	2.61 ± 0.19	2.70 ± 0.19	1.01 ± 0.18	2.36 ± 0.05	2.38 ± 0.07	2.41 ± 0.07
Kin8nm	8192	4	0.08 ± 0.00	0.13 ± 0.01	0.14 ± 0.01	0.13 ± 0.01	-1.14 ± 0.05	-0.56 ± 0.04	-0.49 ± 0.07	-0.55 ± 0.04
Naval	11934	16	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	-4.00 ± 0.20	-3.70 ± 0.25	-3.29 ± 0.44	-3.45 ± 0.23
Power	9568	4	3.99 ± 0.11	4.13 ± 0.11	4.13 ± 0.12	4.11 ± 0.11	2.81 ± 0.03	2.84 ± 0.03	2.85 ± 0.04	2.84 ± 0.03

TABLE III: Training Time on UCI Regression Datasets (1-Sequential).

Dataset	**N**	**d**	**MCMC**	**Progressive**	**Progressive LCD**	**Progressive Naive**
Yacht	308	6	9.46 ± 2.89	7.87 ± 3.12	11.09 ± 0.51	6.56 ± 0.31
Concrete	1030	8	16.86 ± 0.53	21.80 ± 0.49	41.52 ± 1.22	22.01 ± 0.41
Energy	768	8	15.93 ± 1.80	16.03 ± 0.31	29.58 ± 0.90	16.27 ± 0.34
Kin8nm	8192	4	96.31 ± 19.31	118.50 ± 0.64	176.56 ± 5.77	111.41 ± 0.93
Naval	11934	16	152.14 ± 28.69	554.14 ± 3.92	186.71 ± 1.39	466.20 ± 3.88
Power	9568	4	96.10 ± 21.58	66.38 ± 0.66	129.87 ± 2.85	67.70 ± 1.30

TABLE IV: Training Time on UCI Regression Datasets (100-Sequential).

Dataset	**N**	**d**	**MCMC**	**Progressive**	**Progressive LCD**	**Progressive Naive**
Yacht	308	6	9.46 ± 2.89	6.43 ± 3.52	4.85 ± 0.48	4.35 ± 0.21
Concrete	1030	8	16.86 ± 0.53	8.95 ± 1.14	7.99 ± 1.10	8.11 ± 0.83
Energy	768	8	15.93 ± 1.80	8.33 ± 1.69	7.31 ± 0.67	7.04 ± 1.19
Kin8nm	8192	4	96.31 ± 19.31	13.42 ± 2.32	11.01 ± 1.53	13.01 ± 2.00
Naval	11934	16	152.14 ± 28.69	41.50 ± 8.67	17.91 ± 1.49	36.69 ± 10.52
Power	9568	4	96.10 ± 21.58	16.26 ± 2.40	13.81 ± 2.56	11.73 ± 0.62

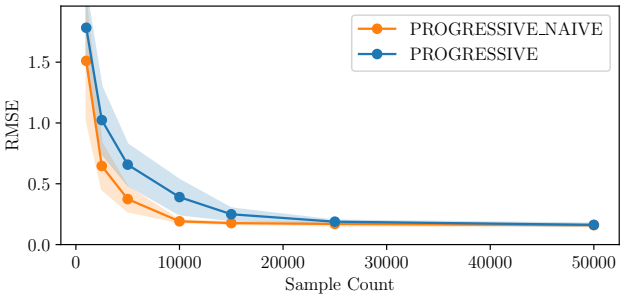

Fig. 4: Performance comparison of black-box and naive version on the UCI "Kin8nm" dataset.

variance is estimated alongside the weights. A network architecture with a single hidden layer with 50 neurons and ReLU activations is used, which means that the fully correlated versions have 151 correlated dimensions, and the layer-wise version has correlated blocks of 100 and 51 dimensions.

Fig. 4 exemplifies the general behavior of PN methods across most datasets (shown here for *Kin8nm*), where the naive layer-wise method reaches stationary performance with fewer samples than the black-box version. This indicates success in achieving our goal of reducing the required sample count. In cases where the complexity of the full rank distribution is required, we expect to observe an intersection point of the two lines because the layer-wise method's performance stagnates at a higher level.

For the full evaluation shown in the tables, randomly sampled versions of PN, both black-box and layer-wise, used 50000 random samples, while Prog. LCD used 25000 deterministic samples. There are two sets of tables for performance and execution time, with Tables I including RMSE and NLL results and III showing results of fully sequential processing (we call it 1-sequential), meaning one filter step is done with one data point. Tables II and IV, however, show results with a batch size of 100 data points for each executed filter step (we call it 100-sequential).

Prog. LCD reaches either best or near best performance of the PN-methods despite using half the number of samples. This shows the importance of sample quality to get good approximations and shows the potential to use layer-wise versions with high fidelity samples in the future as well.

Apart from that, for the 1-sequential case, the layer-wise method is either better or similar in performance compared to the black-box method for most datasets. This is an indicator that in this case, the reduced complexity of the posterior approximation is not limiting and the information per filter step can be effectively incorporated in both cases.

For the 100-sequential version, the situation changes and

the black-box PN method outperforms the layer-wise naive method, reaching or surpassing Prog. LCD performance and in some cases even nearing MCMC-level performance. We explain this behavior by the increased amount of information incorporated per filter step through batch processing. The separated structure limits the model's ability to incorporate all information meaningfully, making the expressivity of the full joint distribution advantageous. While there is a performance gain in using the 100-sequential version, there is also a reduction in execution time because the resampling step happens less frequently.

When we compare the execution times between 1-sequential and 100-sequential versions, we see different speedups for different datasets. The speedup is more pronounced for larger datasets, which can be explained by thinking about the training progress. In the beginning, many progression steps have to be made to incorporate the information of a batch, and then later on, the progression steps can be larger and fewer are needed. For smaller datasets, the progression steps are already large in the beginning when only a batch of one is processed, and the speedup is therefore less pronounced.

VI. CONCLUSIONS

We investigated three approaches to layer-wise training in PN that achieve dimensionality reduction through separation of the weight distribution and found that the naive layer-wise version provides an effective trade-off between separation fidelity and implementation cost. On the UCI regression benchmark, it matches or exceeds the black-box version in predictive performance at lower sample counts for the 1-sequential case, confirming that the separated structure lowers sample requirements. For batch processing, the full joint distribution of the black-box version becomes advantageous, suggesting that the optimal level of independence depends on the amount of information incorporated per filter step. The full layer-wise approach was found to be impractical for PN due to the compounding Gaussian approximation error on activation distributions.

The results establish layer-wise separation as a practical and effective strategy for reducing sample requirements in PN, enabling more tractable training at larger scales. Future work will target principled hyperparameter selection and higher-dimensional problem settings, where the benefits of this separation are most pronounced.

REFERENCES

[1] 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.

[2] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. "Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users". In: *IEEE Computational Intelligence Magazine* 17.2 (May 2022), pp. 29–48.

[3] 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*. Vol. 37. 8. 2023, pp. 10069–10077.

[4] 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).

[5] Leon Winheim and Uwe D. Hanebeck. "BNN Training as State Estimation: A Progressive Filtering Approach". In: *Proceedings of the 2025 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2025)*. College Station, Texas, Sept. 2025.

[6] 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.

[7] David J. Schodt, Ryan Brown, Michael Merritt, Samuel Park, Delsin Menolascino, and Mark A. Peot. *A Framework for Variational Inference of Lightweight Bayesian Neural Networks with Heteroscedastic Uncertainties*. Feb. 22, 2024. Pre-published.

[8] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*. 2nd ed. Cambridge University Press, May 31, 2023.

[9] 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.

[10] Sharad Singhal and Lance Wu. "Training Multilayer Perceptrons with the Extended Kalman Algorithm". In: *Advances in neural information processing systems* 1 (1988).

[11] G.V. Puskorius and L.A. Feldkamp. "Decoupled Extended Kalman Filter Training of Feedforward Layered Networks". In: *IJCNN-91-Seattle International Joint Conference on Neural Networks*. Vol. i. July 1991, 771–777 vol.1.

[12] David MacKay. "A Practical Bayesian Framework for Backprop Networks". In: *Neural Comput* 4.3 (1992), pp. 448–472.

[13] Geoffrey E Hinton and Drew Van Camp. "Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights". In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*. 1993, pp. 5–13.

[14] João de Freitas, Mahesan Niranjan, Arnaud Doucet, and Andrew Gee. "Global Optimisation of Neural Network Models via Sequential Sampling". In: *Advances in Neural Information Processing Systems*. Ed. by M. Kearns, S. Solla, and D. Cohn. Vol. 11. MIT Press, 1998.

[15] 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.

[16] Shushuai Xie, Wei Cheng, Zelin Nie, Qian Huang, Ji Xing, Xuefeng Chen, Rongyong Zhang, and Yunjie Yang. "Bayesian Physics-Informed Neural Networks with Iterative Ensemble Kalman Inversion for RUL Prediction and Uncertainty Quantification". In: *Advanced Engineering Informatics* 69 (Jan. 1, 2026), p. 103907.

[17] Andrew Pensoneault and Xueyu Zhu. "Efficient Bayesian Physics Informed Neural Networks for Inverse Problems via Ensemble Kalman Inversion". In: *Journal of Computational Physics* 508 (July 1, 2024), p. 113006.

[18] 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.

[19] Petar M. Djurić and Mónica F. Bugallo. "Multiple Particle Filtering with Improved Efficiency and Performance". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2015, pp. 4110–4114.

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