Bridging Bayesian Inference and Neural Network Training: Equivalence of KBNN and Statistical Linearization

Hayk Amirkhanian[*†], **Markus Walker**[*‡], **Uwe D. Hanebeck**[‡], and **Marco F. Huber**[†§]

[†]Research Unit Artificial Intelligence and Machine Vision
Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany
hayk.amirkhanian.namagerdi@ipa.fraunhofer.de, marco.huber@ieee.org

[‡]Intelligent Sensor-Actuator-Systems Laboratory (ISAS)
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT), Germany
markus.walker@kit.edu, uwe.hanebeck@kit.edu

[§]Institute of Industrial Manufacturing and Management IFF
University of Stuttgart, Germany

***Abstract*—Accurate uncertainty quantification is critical for robust and trustworthy predictions in many real-world applications. Bayesian Neural Networks (BNNs) provide a principled approach for modeling uncertainty but are often limited by the computational complexity of Bayesian inference. In this paper, we introduce a statistical linearization approach for multi-layer feedforward BNNs. We demonstrate that this statistical linearization is equivalent to the Kalman Bayesian Neural Networks (KBNN) framework. This equivalence unifies these methodologies, providing a theoretical foundation for understanding the relationship between different BNN training approaches.**

***Index Terms*—Bayesian Neural Networks, uncertainty quantification, statistical linearization.**

I. Introduction

In many practical scenarios, accurately capturing uncertainty is crucial for robust and trustworthy predictions. Classical neural networks are powerful universal approximation methods, e.g., in areas such as time series forecasting [1], [2], yet they typically generate only point estimates, overlooking inherent uncertainties in real-world data. For instance, point estimates fail to account for the full range of possible outcomes, highlighting the importance of uncertainty quantification. However, to capture uncertainties in real-world data, Bayesian Neural Networks (BNNs) extend classical neural networks by quantifying the uncertainty of predictions as *predictive distributions*. For example, univariate Gaussian distributed predictions not only provide a mean value (equivalent to a point estimate) but also a variance, serving as a confidence measure. Providing predictive distributions using BNNs, leads to well-calibrated predictions that match the uncertainty of the real-world data-generating process well when sufficient training data is available [3], which can be tested using methods and calibration measures such as in [4], [5], [6].

However, leveraging BNNs for uncertainty quantification introduces its own set of challenges. Despite their potential for effective uncertainty quantification, training BNNs involves learning the weight posterior distribution, which represents the probability distribution of the network's weights given the data. Exact learning is intractable for BNNs, necessitating the use of approximation methods. These approximation methods are based on well-known principles of Bayesian inference, such as Markov Chain Monte Carlo (MCMC) [7], Variational Inference (VI) [8], Expectation Propagation (EP) [9], Laplace approximation (LA) [10], or Bayesian filtering techniques [11].

Especially interesting are Bayesian filtering-based BNN learning techniques such as the Kalman Bayesian Neural Networks (KBNN), which avoid explicit gradient computation during training, allowing them to be used for fast sequential learning. The Bayesian perceptron, proposed in [12], forms the core building block of the KBNN and provides a closed-form solution for the forward pass, which computes the output of the BNN given an input; and the weight update, which adjusts the weights based on the observed data. Furthermore, [13] formally proves that training a Bayesian perceptron is equivalent to first applying statistical linearization to a single perceptron and then estimating its parameters using a Kalman filter.

However, the proof presented in [13] is limited to the Bayesian perceptron, and extending it to feedforward BNNs with multiple layers remains an open question. Addressing this gap, the present paper explores the extension of the proof to multi-layer feedforward BNNs.

The main motivation here is that by reformulating BNN training as an linear state-space model (LSSM), we can exploit the extensive Bayesian state-estimation toolbox such as the Kalman filter, the extended- or unscented Kalman filter, and related variants.

Contribution: This paper briefly reviews the principle methods for training BNNs and recent training procedures.

This work was supported by the Baden-Württemberg Ministry of Economic Affairs, Labor, and Tourism within the KI-Fortschrittszentrum "Lernende Systeme und Kognitive Robotik" (036-140100), and the Fraunhofer-Gesellschaft within the project ML4Safety. This work is also part of the German Research Foundation (DFG) AI Research Unit 5339 regarding the combination of physics-based simulation with AI-based methodologies for the fast maturation of manufacturing processes.

[*]These authors contributed equally.

Furthermore, we review the KBNN and prove that the KBNN is equivalent to statistical linearization of feedforward BNNs with multiple layers.

Notation: In this paper, underlined letters, e.g., $\underline{x}$, denote vectors, boldface letters, such as $\underline{\boldsymbol{x}}$, represent random variables, while boldface capital letters, e.g., $\mathbf{A}$, indicate matrices and underlined boldface capital letters such as $\underline{\mathbf{W}}$ denote matrices consisting of random variables.

II. Related Work

Unlike classical neural networks that employ deterministic weights, BNNs represent weights as posterior distributions, which cannot be learned using standard backpropagation. Learning weight posterior distributions relies on fundamental approaches of approximate Bayesian inference, which are detailed in this section.

A. Markov Chain Monte Carlo

MCMC-based approaches are widely recognized for training BNNs, as they approximate the posterior distribution through sampling techniques. However, the original form of MCMC, specifically the Metropolis–Hastings algorithm [7], is computationally intensive due to the necessity of generating a large number of samples. To address this limitation, several extensions have been developed, including Gibbs sampling [14], Hybrid Monte Carlo [15], Hamiltonian Monte Carlo [16], and the no-U-turn sampler [17].

To further mitigate the computational costs associated with these methods, [18] introduced a scalable subsampling-based MCMC scheme by combining stochastic gradient descent with Langevin dynamics. This approach was subsequently extended in [19], [20], providing efficient sampling techniques for training BNNs.

B. Variational Inference

VI [8] provides an optimization-based method for training BNNs by employing a surrogate function, i.e., the variational distribution, to approximate the true weight posterior. This approach converts the intractable Bayesian inference problem into a manageable optimization task by maximizing the evidence lower bound, thereby minimizing the reverse Kullback–Leibler divergence between the variational distribution and the true posterior.

Several implementations of VI have been developed for BNNs, including Stochastic Variational Inference [21], Deterministic Variational Inference [22], and the widely recognized Bayes by Backprop [23].

C. Expectation Propagation

EP [9] minimizes the forward Kullback–Leibler divergence, which, however, differs from the reverse Kullback–Leibler divergence optimization employed by VI, resulting in no guaranteed convergence for EP methods. Nevertheless, convergence can be achieved by implementing damped versions of EP, such as double-loop EP [24] and damped EP [25].

A notable variant of EP tailored for BNNs, known as probabilistic backpropagation [26], has attracted significant attention in the field. This method incorporates hyperprior distributions for the weights and relies on a moment-matching approach during the forward pass. During the backward pass, probabilistic backpropagation calculates gradients of the marginal likelihood with respect to the parameters of the posterior approximation.

D. Laplace Approximation

LA [10] is a foundational method for approximate Bayesian inference. This approach approximates the posterior distribution around the maximum a posteriori (MAP) estimate using a Gaussian distribution. Specifically, standard deep learning techniques are first employed to identify the MAP point estimate. Around this estimate, LA assumes that the shape of the posterior distribution can be accurately represented by a Gaussian distribution. This is achieved through a Taylor-series expansion of the log-posterior around the MAP estimate up to the second order. The curvature of the log-posterior distribution at the MAP point, characterized by the Hessian matrix of the log-posterior, is utilized to construct a Gaussian distributed approximation of the posterior. Consequently, LA can be applied post-hoc to a pre-trained network.

Direct computation of the Hessian is often infeasible for large neural networks due to its size and computational cost. Therefore, recent curvature approximation methods [27], such as the Kronecker-Factored Approximate Curvature, have been developed to construct a scalable LA for neural networks. Other approaches efficiently compute a posterior over a subnetwork by treating only a subset of weights as random variables [28], exemplified by the last-layer LA [29], which applies Bayesian learing exclusively to the network's final layer.

E. Bayesian Filtering

In addition to MCMC and gradient-based optimization methods, Bayesian filtering approaches offer a means to train BNNs without requiring explicit gradient computations of any specific loss function. Typically, Bayesian filtering-based methods assume that both weights and predictions are Gaussian random variables. Bayesian filtering approaches vary in their handling of nonlinearities. E.g., the extended Kalman filter [30], [31] necessitates analytical linearization of the network's nonlinearities, whereas Kalman filtering techniques such as the unscented Kalman filter [32] and the ensemble Kalman filter [33] utilize sampling-based methods.

Recent Bayesian filtering-based training methods for BNNs include Tractable Approximate Gaussian Inference [34], which assumes independent weights (mean-field assumption), and KBNN [11], which allows for full covariance matrices per neuron, representing a less restrictive assumption compared to Tractable Approximate Gaussian Inference. In the KBNN framework, analytical formulations are provided for forecasting each individual perceptron's output and for weight learning, accommodating widely used activation functions such as *sigmoid* and *rectified linear unit*.

F. Alternative Training Techniques

While the aforementioned training methods are grounded in the fundamental principles of Bayesian inference, they are often computationally intensive and more complex to implement compared to conventional neural network training. Consequently, some approaches aim to offer simpler

implementations and computationally efficient approximations of the weight posterior. Examples include the dropout technique proposed by [35], which serves as an approximation of VI [36], [37]; ensemble methods consisting of different models trained for the same purpose [38], [39], [40], also referred to as deep ensembles; and the Stochastic Weight Averaging Gaussian [41].

III. Learning Framework

We consider a supervised learning framework where the training data set $\mathcal{D} = \{(\underline{x}_n, \underline{y}_n)\}_{n=1}^N$ comprises N input-output pairs. Each pair consists of an input vector $\underline{x}_n \in \mathbb{R}^{d_x}$ and its corresponding output realization $\underline{y}_n \in \mathbb{R}^{d_y}$ generated by the underlying noisy data-generating process. Typically, the true data-generating process and the precise mapping between inputs and noisy outputs are unknown, making the primary objective of the supervised learning framework to accurately infer this mapping using the available training data.

To achieve this, we utilize a feedforward BNN to model the relationship between inputs and outputs, defined by $\underline{\boldsymbol{y}} = f(\underline{x}, \underline{\boldsymbol{w}}) + \underline{\boldsymbol{\varepsilon}}$ with $\underline{\boldsymbol{\varepsilon}} \sim \mathcal{N}(0, \mathbf{C}_{\underline{\boldsymbol{\varepsilon}}})$. Here, $\underline{x}$ is the deterministic d_x-dimensional input, $\underline{\boldsymbol{w}}$ denotes the random weight vector encompassing all network weights, and $\underline{\boldsymbol{y}}$ represents the d_y-dimensional output as a random vector. The weight vector $\underline{\boldsymbol{w}}$ is constructed by flattening the random weight matrices $\mathbf{W}_l$ from each layer $l = 1, \ldots, L$ into vectors $\underline{\boldsymbol{w}}_l$, and then concatenating these vectors into a single d_w-dimensional column vector $\underline{\boldsymbol{w}}$. The weight posterior $p(\underline{w} \mid \mathcal{D})$ as well as the predictive distribution $p(\underline{y} \mid \underline{x}, \mathcal{D})$ are given by

$$p(\underline{w} \mid \mathcal{D}) = \frac{p(\mathcal{Y} \mid \mathcal{X}, \underline{w})\, p(\underline{w})}{p(\mathcal{Y} \mid \mathcal{X})} \quad , \text{ and}$$
$$p(\underline{y} \mid \underline{x}, \mathcal{D}) = \int_{\Omega_{\underline{\boldsymbol{w}}}} p(\underline{y} \mid \underline{x}, \underline{w})\, p(\underline{w} \mid \mathcal{D})\, \mathrm{d}\underline{w} \quad ,$$

where $\Omega_{\underline{\boldsymbol{w}}} \subseteq \mathbb{R}^{d_w}$ is the sample space of the weights, $p(\underline{w})$ is the prior, $p(\mathcal{Y} \mid \mathcal{X}) = \int_{\Omega_{\underline{\boldsymbol{w}}}} p(\mathcal{Y} \mid \mathcal{X}, \underline{w})\, p(\underline{w})\, \mathrm{d}\underline{w}$ is the normalization constant, and $\mathcal{X} = \{\underline{x}_1, \ldots, \underline{x}_N\}$ and $\mathcal{Y} = \{\underline{y}_1, \ldots, \underline{y}_N\}$ are the sets of input and output data of the training data set $\mathcal{D}$.

Typically, output data $\underline{y}_n$ are assumed independent given deterministic $\underline{x}_n$ and therefore the likelihood is given by $p(\mathcal{Y} \mid \mathcal{X}, \underline{w}) = \prod_{n=1}^N p(\underline{y}_n \mid \underline{x}_n, \underline{w})$. For the remainder of this paper, we assume that the likelihood is Gaussian, i.e., $p(\underline{y}_n \mid \underline{x}_n, \underline{w}) = \mathcal{N}(\underline{y}_n | f(\underline{x}_n, \underline{w}), \mathbf{C}_{\underline{\boldsymbol{\varepsilon}}})$.

It should be noted that there is no tractable solution available for training or prediction. Therefore, in practice, approximation methods must be applied, as described in Sec. II. Throughout the remainder of this paper, we will examine the KBNN and its relationship with the statistical linearization of feedforward BNNs.

IV. KBNN

The KBNN assumes that the Markov property holds in BNNs allowing each layer to be treated individually. Both the prediction of the output, i.e., the forward pass, and the weight update are thus performed sequentially, as shown in Fig. 1.

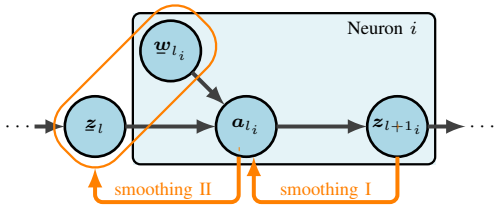

Fig. 1. Probabilistic graphical model for neuron i in the l-th layer in a feedforward BNN, adapted from [11]. The neuron's input $\underline{\boldsymbol{z}}_l$ is multiplied by the i-th column of $\mathbf{W}_l = (\underline{\boldsymbol{w}}_{l_1} \ldots \underline{\boldsymbol{w}}_{l_i} \ldots \underline{\boldsymbol{w}}_{l_{M_l}})$ to form the activation $\underline{\boldsymbol{a}}_{l_i}$, which is subsequently passed through the activation function to yield the neuron's output $\underline{\boldsymbol{z}}_{l+1_i}$. In the backward pass, this procedure is reversed, starting with updating the activation and then the neuron's weights and the input.

A. Forward Pass

Given an input $\underline{\boldsymbol{z}}_l$ and a weight matrix $\mathbf{W}_l$, KBNN assumes that both the pre-activation $\underline{\boldsymbol{a}}_l$ as well as the post-activation $\underline{\boldsymbol{z}}_{l+1}$ of the l-th layer are jointly Gaussian distributed. It is also assumed, that both the pre- and post-activations are all pairwise independent, meaning their covariance matrices are fully diagonal. This in turn means, that one only has to calculate the elements of the main diagonal. The moments of the pre-activation $\underline{\boldsymbol{a}}$ can be calculated as

$$\begin{aligned}
\mu_{\underline{\boldsymbol{a}}_{l_i}} &= \underline{\mu}_{\underline{\boldsymbol{w}}_{l_i}}^\top \underline{\mu}_{\underline{\boldsymbol{z}}_l} \quad , \\
\sigma^2_{\underline{\boldsymbol{a}}_{l_i}} &= \underline{\mu}_{\underline{\boldsymbol{w}}_{l_i}}^\top \mathbf{C}_{z_l} \underline{\mu}_{\underline{\boldsymbol{w}}_{l_i}} + \underline{\mu}_{z_l}^\top \mathbf{C}_{\underline{\boldsymbol{w}}_{l_i}} \underline{\mu}_{z_l} \\
&\quad + \mathrm{Tr}\Big(\mathbf{C}_{\underline{\boldsymbol{w}}_{l_i}} \mathbf{C}_{\underline{\boldsymbol{z}}_l}\Big) \quad , \quad \forall i = 1, ..., M_l \quad ,
\end{aligned}$$

where M_l denotes the depth of the l-th layer, $\mathrm{Tr}(\cdot)$ the trace-function of a matrix and $\underline{\boldsymbol{w}}_{l_i}$ is the i-th column of the weight matrix $\mathbf{W}_l$.

If the activation function is the rectified linear unit function, then the moments of the post-activation $\underline{\boldsymbol{z}}_l$ are matched according to

$$\begin{aligned}
\underline{\mu}_{\underline{\boldsymbol{z}}_{l+1}} &= \underline{\mu}_{\underline{\boldsymbol{a}}_l}\, \phi\left(\frac{\underline{\mu}_{\underline{\boldsymbol{a}}_l}}{\underline{\sigma}_{\underline{\boldsymbol{a}}_l}}\right) + \underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}\, \mathcal{N}\Big(\underline{0}\,\Big|\, \underline{\mu}_{\underline{\boldsymbol{a}}_l}, \underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}\Big) \quad , \\
\underline{\sigma}^2_{\underline{\boldsymbol{z}}_{l+1}} &= \Big(\underline{\mu}^2_{\underline{\boldsymbol{a}}_l} + \underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}\Big)\, \phi\left(\frac{\underline{\mu}_{\underline{\boldsymbol{a}}_l}}{\underline{\sigma}_{\underline{\boldsymbol{a}}_l}}\right) \\
&\quad + \underline{\mu}_{\underline{\boldsymbol{a}}_l}\, \underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}\, \mathcal{N}\Big(\underline{0}\,\Big|\, \underline{\mu}_{\underline{\boldsymbol{a}}_l}, \underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}\Big) - \underline{\mu}^2_{\underline{\boldsymbol{z}}_{l+1}} \quad ,
\end{aligned}$$

where $\mathcal{N}(\underline{0}|\cdot,\cdot)$ denotes the Gaussian probability density function evaluated at $\underline{0}$, $\underline{\mu}^2_{\underline{\boldsymbol{a}}_l}$ and $\underline{\sigma}^2_{\underline{\boldsymbol{a}}_l}$ are interpreted element-wise and $\phi(\cdot)$ represents the cumulative distribution function of the standard Gaussian distribution. It is defined as $\phi(x) = \frac{1}{2}(1 + \mathrm{erf}(x/\sqrt{2}))$ with $\mathrm{erf}(\cdot)$ being the Gaussian error function. The moments of other activation functions can be found in [12].

B. Backward Pass

The backward pass of KBNN is again performed sequentially for all layers. First, the pre-activation $\underline{\boldsymbol{a}}_l$ is updated, followed by the weight matrix $\mathbf{W}_l$ and the input of that layer $\underline{\boldsymbol{z}}_l$, which

corresponds to the post-activation of the preceding layer. The pre-activation $\boldsymbol{a}_{l_i}$ can then be updated according to

$$\begin{aligned}
\mu^+_{\boldsymbol{a}_{l_i}} &= \mu_{\boldsymbol{a}_{l_i}} + \underline{k}_{l_i}^\top \left(\underline{\mu}^+_{\underline{\boldsymbol{z}}_{l+1}} - \underline{\mu}_{\underline{\boldsymbol{z}}_{l+1}} \right) \ , \\
\left(\sigma^+_{\boldsymbol{a}_{l_i}} \right)^2 &= \sigma^2_{\boldsymbol{a}_{l_i}} + \underline{k}_{l_i}^\top \left(\mathbf{C}^+_{\underline{\boldsymbol{z}}_{l+1}} - \mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}} \right) \underline{k}_{l_i} \ , \\
\underline{k}_{l_i} &= \left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}} \right)^{-1} \sigma^2_{\boldsymbol{a}_{l_i} \boldsymbol{z}_{l+1_i}} \ , \\
\sigma^2_{\boldsymbol{a}_{l_i} \boldsymbol{z}_{l+1_i}} &= \sigma^2_{\boldsymbol{z}_{l_i}} + \mu^2_{\boldsymbol{z}_{l+1_i}} - \mu_{\boldsymbol{a}_{l_i}} \mu_{\boldsymbol{z}_{l+1_i}} \ ,
\end{aligned} \tag{1}$$

with $\forall i = 1, \ldots, M_l$. The weight vector $\underline{\boldsymbol{w}}_l$ and the post-acitvation $\underline{\boldsymbol{z}}_l$ can then be updated according to

$$\begin{aligned}
\begin{bmatrix} \underline{\mu}^+_{\underline{\boldsymbol{w}}_l} \\ \underline{\mu}^+_{\underline{\boldsymbol{z}}_l} \end{bmatrix} &= \begin{bmatrix} \underline{\mu}_{\underline{\boldsymbol{w}}_l} \\ \underline{\mu}_{\underline{\boldsymbol{z}}_l} \end{bmatrix} + \mathbf{L}_l \left(\underline{\mu}^+_{\underline{\boldsymbol{a}}_l} - \underline{\mu}_{\underline{\boldsymbol{a}}_l} \right) \ , \\
\begin{bmatrix} \mathbf{C}^+_{\underline{\boldsymbol{w}}_l} & \mathbf{C}_{\underline{\boldsymbol{w}}_l \underline{\boldsymbol{z}}_l} \\ \mathbf{C}^\top_{\underline{\boldsymbol{w}}_l \underline{\boldsymbol{z}}_l} & \mathbf{C}^+_{\underline{\boldsymbol{z}}_l} \end{bmatrix} &= \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_l} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{z}}_l} \end{bmatrix} \\
&\quad + \mathbf{L}_l \left(\mathbf{C}^+_{\underline{\boldsymbol{a}}_l} - \mathbf{C}_{\underline{\boldsymbol{a}}_l} \right) \mathbf{L}_l^\top \ , \\
\mathbf{L}_l &= \mathbf{C}_{\underline{\boldsymbol{w}}_l \underline{\boldsymbol{z}}_l \underline{\boldsymbol{a}}_l} \cdot \left(\mathbf{C}_{\underline{\boldsymbol{a}}_l} \right)^{-1} \ , \\
\mathbf{C}_{\underline{\boldsymbol{w}}_l \underline{\boldsymbol{z}}_l \underline{\boldsymbol{a}}_l} &= \begin{bmatrix} \operatorname{diag}\left(\mathbf{C}_{\underline{\boldsymbol{w}}_1 \underline{\mu}_{\underline{\boldsymbol{z}}_l}}, \ldots, \mathbf{C}_{\underline{\boldsymbol{w}}_{M_l} \underline{\mu}_{\underline{\boldsymbol{z}}_l}} \right) \\ \mathbf{C}_{\underline{\boldsymbol{z}}_l \underline{\mu}_{\underline{\boldsymbol{w}}_1}}, \ldots, \mathbf{C}_{\underline{\boldsymbol{z}}_l \underline{\mu}_{\underline{\boldsymbol{w}}_{M_l}}} \end{bmatrix} \ .
\end{aligned} \tag{2}$$

Here, the cross-covariance matrix $\mathbf{C}_{\underline{\boldsymbol{w}}_l \underline{\boldsymbol{z}}_l \underline{\boldsymbol{a}}_l}$ denotes the covariance between the vector $\begin{bmatrix} \underline{\boldsymbol{w}}_l & \underline{\boldsymbol{z}}_l \end{bmatrix}^\top$ and $\underline{\boldsymbol{a}}_l$. The operator $\operatorname{diag}(\cdot)$ outputs a diagonal matrix whose main diagonal elements are the elements of its input vector.

V. Concept of Statistical Linearization

Before we introduce the statistical linearization of feedforward BNNs, we first introduce the basic concept of statistical linearization based on the general formulation in [42].

Assume the following situation: $\underline{\boldsymbol{y}} = f(\underline{\boldsymbol{x}})$ with $\underline{\boldsymbol{x}} \sim \mathcal{N}(\underline{\mu}_{\underline{\boldsymbol{x}}}, \mathbf{C}_{\underline{\boldsymbol{x}}})$ and $\underline{\mu}_{\underline{\boldsymbol{y}}}, \mathbf{C}_{\underline{\boldsymbol{y}}}$ being the mean and covariance matrix of $\underline{\boldsymbol{y}}$, respectively. We are now trying to approximate

$$\underline{\boldsymbol{y}} \approx \mathbf{A} \underline{\boldsymbol{x}} + \underline{\boldsymbol{b}} \ ,$$

with $\mathbf{A}$ being a matrix and $\underline{\boldsymbol{b}}$ being a vector. We wish to preserve the first two moments of $\underline{\boldsymbol{y}}$ and also to minimize the mean-squared error of the linearization. By matching the moments, it immediately follows that $\underline{\boldsymbol{b}} \sim \mathcal{N}(\underline{\mu}_{\underline{\boldsymbol{y}}} - \mathbf{A} \underline{\mu}_{\underline{\boldsymbol{x}}}, \mathbf{C}_{\underline{\boldsymbol{y}}} - \mathbf{A} \mathbf{C}_{\underline{\boldsymbol{x}}} \mathbf{A}^\top)$ must hold. The mean-squared error is given as

$$\operatorname{MSE}(\mathbf{A}, \underline{\boldsymbol{b}}) = \mathrm{E} \left\{ \left(\underline{\boldsymbol{y}} - \mathbf{A} \underline{\boldsymbol{x}} - \underline{\boldsymbol{b}} \right)^\top \left(\underline{\boldsymbol{y}} - \mathbf{A} \underline{\boldsymbol{x}} - \underline{\boldsymbol{b}} \right) \right\} \ . \tag{3}$$

Setting derivatives of (3) with respect to $\mathbf{A}$ to zero gives

$$\mathbf{A} = \mathbf{C}_{\underline{\boldsymbol{x}} \underline{\boldsymbol{y}}}^\top \left(\mathbf{C}_{\underline{\boldsymbol{x}}} \right)^{-1} \ ,$$

where $\mathbf{C}_{\underline{\boldsymbol{x}} \underline{\boldsymbol{y}}}$ is the cross-covariance matrix between $\underline{\boldsymbol{x}}$ and $\underline{\boldsymbol{y}}$ and hence the parameters of the linearization are given by

$$\mathbf{A} = \mathbf{C}_{\underline{\boldsymbol{x}} \underline{\boldsymbol{y}}}^\top \left(\mathbf{C}_{\underline{\boldsymbol{x}}} \right)^{-1} \ , \tag{4}$$

$$\underline{\mu}_{\underline{\boldsymbol{b}}} = \underline{\mu}_{\underline{\boldsymbol{y}}} - \mathbf{A} \underline{\mu}_{\underline{\boldsymbol{x}}} \ , \tag{5}$$

$$\mathbf{C}_{\underline{\boldsymbol{b}}} = \mathbf{C}_{\underline{\boldsymbol{y}}} - \mathbf{A} \mathbf{C}_{\underline{\boldsymbol{x}}} \mathbf{A}^\top \ . \tag{6}$$

Detailed derivations of this can be found in [42, pp. 179–180].

VI. Statistical Linearization of BNNs

We begin by iteratively applying statistical linearization to all layers of the BNN. To improve readability, we will omit the noise term $\underline{\boldsymbol{\varepsilon}}$, leading to

$$\begin{aligned}
\underline{\boldsymbol{y}} &= f_L(\underline{\boldsymbol{a}}_L) = f_L(\underline{\mathbf{W}}_L \cdot \underline{\boldsymbol{z}}_L) \ , \\
&\approx \mathbf{F}_L \mathbf{G}_L \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{z}}_L \end{bmatrix} + \underbrace{\mathbf{F}_L \underline{\boldsymbol{b}}_{\mathbf{G}_L} + \underline{\boldsymbol{b}}_{\mathbf{F}_L}}_{=: \underline{\boldsymbol{b}}_L} \ ,
\end{aligned} \tag{7}$$

where $\mathbf{F}_L$ represents the statistical linearization of the activation function of the last layer f_L, and $\mathbf{G}_L$ corresponds to the linearization of the matrix-vector multiplication. If either the input or the output of a given layer are one-dimensional, then both $\mathbf{F}$ and $\mathbf{G}$ are vectors, otherwise they are matrices. Since in most cases, neither the input nor the output of a layer is one-dimensional, we will denote them as matrices. Based on the statistical linearization introduced in Sec. V, the linearized activation function is given by

$$\mathbf{F}_L := \mathbf{C}^\top_{\underline{\boldsymbol{a}}_L \underline{\boldsymbol{z}}_{L+1}} \left(\mathbf{C}_{\underline{\boldsymbol{a}}_L} \right)^{-1} \ ,$$

assuming that $\underline{\boldsymbol{a}}_L$ and $\underline{\boldsymbol{z}}_{L+1}$ are jointly Gaussian distributed, where $\mathbf{C}_{\underline{\boldsymbol{a}}_L \underline{\boldsymbol{z}}_{L+1}}$ is the cross-covariance matrix between $\underline{\boldsymbol{a}}_L$ and $\underline{\boldsymbol{z}}_{L+1}$, and $\mathbf{C}_{\underline{\boldsymbol{a}}_L}$ is the covariance matrix of $\underline{\boldsymbol{a}}_L$.

Note that $\underline{\boldsymbol{z}}_{L+1} = \underline{y}_n$ is the realization from the training data set and thus $\underline{\mu}_{\underline{\boldsymbol{z}}_{L+1}} = \underline{y}_n$ for some $n \in \{1, \ldots, N\}$ and $\mathbf{C}_{\underline{\boldsymbol{z}}_{L+1}} = \mathbf{0}$ holds. The linearized matrix-vector product is given by

$$\mathbf{G}_L := \mathbf{C}^\top_{\underline{\boldsymbol{w}}_L \underline{\boldsymbol{z}}_L \underline{\boldsymbol{a}}_L} \left(\mathbf{C}_{\underline{\boldsymbol{w}}_L \underline{\boldsymbol{z}}_L} \right)^{-1} \ ,$$

where $\mathbf{C}_{\underline{\boldsymbol{w}}_L \underline{\boldsymbol{z}}_L \underline{\boldsymbol{a}}_L}$ is the cross-covariance matrix between the concatenated quantities $(\underline{\boldsymbol{w}}_L \ \ \underline{\boldsymbol{z}}_L)^\top$ and the activation $\underline{\boldsymbol{a}}_L$. Again it is assumed that these quantities are jointly Gaussian distributed. The bias of the statistical linearization is then given by

$$\begin{aligned}
\underline{\boldsymbol{b}}_L &\sim \mathcal{N}\left(\underline{\mu}_{\underline{\boldsymbol{b}}}, \mathbf{C}_{\underline{\boldsymbol{b}}} \right) \ , \\
\underline{\mu}_{\underline{\boldsymbol{b}}} &= \underbrace{\underline{\mu}_y - \underbrace{\mathbf{F}_L \mathbf{G}_L}_{\text{(a)}} \begin{bmatrix} \underline{\boldsymbol{\mu}}_{\underline{\boldsymbol{w}}_L} \\ \underline{\boldsymbol{\mu}}_{z_L} \end{bmatrix}}_{\text{(b)}} \ , \\
\mathbf{C}_{\underline{\boldsymbol{b}}} &= \underbrace{\mathbf{C}_{\underline{\boldsymbol{y}}} - \mathbf{F}_L \mathbf{G}_L \mathbf{C}_{\underline{\boldsymbol{w}}_L \underline{\boldsymbol{z}}_L} \mathbf{G}_L^\top \mathbf{F}_L^\top}_{\text{(c)}} \ ,
\end{aligned}$$

where $\mathbf{C}_{\underline{\boldsymbol{w}}_L \underline{\boldsymbol{z}}_L}$ is the covariance matrix of $(\underline{\boldsymbol{w}}_L \ \ \underline{\boldsymbol{z}}_L)^\top$.

The relation to the statistical linearization can be seen by comparing (a), (b), and (c) with (4) to (6), respectively. Repeating the statistical linearization step from (7) for the second last layer results in

$$\begin{aligned}
\underline{\boldsymbol{y}} &\approx \mathbf{F}_L \mathbf{G}_L \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{z}}_L \end{bmatrix} + \underline{\boldsymbol{b}}_L \\
&\approx \mathbf{F}_L \mathbf{G}_L \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \mathbf{F}_{L-1} \mathbf{G}_{L-1} \begin{bmatrix} \underline{\boldsymbol{w}}_{L-1} \\ \underline{\boldsymbol{z}}_{L-1} \end{bmatrix} + \underline{\boldsymbol{b}}_{L-1} \end{bmatrix} + \underline{\boldsymbol{b}}_L \\
&= \mathbf{F}_L \mathbf{G}_L \begin{bmatrix} \mathbf{1}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-1} \mathbf{G}_{L-1} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{w}}_{L-1} \\ \underline{\boldsymbol{z}}_{L-1} \end{bmatrix} + \tilde{\underline{\boldsymbol{b}}}_{L-1} \ ,
\end{aligned}$$

where $\tilde{\underline{b}}_{L-1} = \mathbf{F}_L\ \mathbf{G}_L\ \underline{b}_{L-1} + \underline{b}_L$ combines all linearization biases into a single term.

Extending this approach further until the input layer is reached gives

$$\begin{aligned}
\underline{y} &\approx \mathbf{F}_L\mathbf{G}_L \begin{bmatrix} \mathbf{1}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-1}\mathbf{G}_{L-1} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{w}}_{L-1} \\ \underline{\boldsymbol{z}}_{L-1} \end{bmatrix} + \tilde{\underline{b}}_{L-1} \\
&= \mathbf{F}_L\mathbf{G}_L \begin{bmatrix} \mathbf{1}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-1}\mathbf{G}_{L-1} \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} \mathbf{1}_L & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{L-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F}_{L-2}\mathbf{G}_{L-2} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{w}}_{L-1} \\ \underline{\boldsymbol{w}}_{L-2} \\ \underline{\boldsymbol{z}}_{L-2} \end{bmatrix} + \tilde{\underline{b}}_{L-2} \\
&= \mathbf{F}_L\mathbf{G}_L \prod_{l=1}^{2} \begin{bmatrix} \mathbf{1}_{L-l+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-l}\mathbf{G}_{L-l} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \underline{\boldsymbol{w}}_{L-1} \\ \underline{\boldsymbol{w}}_{L-2} \\ \underline{\boldsymbol{z}}_{L-2} \end{bmatrix} + \tilde{\underline{b}}_{L-2} \\
&= \dots \\
&= \underbrace{\mathbf{F}_L\mathbf{G}_L \prod_{l=1}^{L-1} \begin{bmatrix} \mathbf{1}_{L-l+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-l}\mathbf{G}_{L-l} \end{bmatrix}}_{=:\mathbf{H}} \begin{bmatrix} \underline{\boldsymbol{w}}_L \\ \vdots \\ \underline{\boldsymbol{w}}_1 \\ \underline{x} \end{bmatrix} + \tilde{\underline{b}}_1 \ .
\end{aligned}$$

Note that the dimensions of the identity matrix, $\mathbf{1}_l$, vary between different terms in the products, depending on the number of weights per layers involved. Expanding $\mathbf{H}$ while keeping track of the identity matrix dimensions leads to

$$\begin{aligned}
\mathbf{H}^\top &= \left(\prod_{l=1}^{L-1} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{L-l}\mathbf{G}_{L-l} \end{bmatrix} \right)^\top \cdot \mathbf{G}_L^\top \mathbf{F}_L^\top \\
&= \begin{bmatrix} \mathbf{1} & & \mathbf{0} \\ \mathbf{0} & & \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \mathbf{0} & \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_{L-2}^\top \mathbf{F}_{L-2}^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \vdots & & \vdots \\ \mathbf{0} & \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} \prod_{l=2}^{L-2} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_l^\top \mathbf{F}_l^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \mathbf{0} & \prod_{l=1}^{L-2} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_l^\top \mathbf{F}_l^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \end{bmatrix} \cdot \mathbf{G}_L^\top \mathbf{F}_L^\top \ .
\end{aligned}$$

After linearization, the BNN is transformed into a LSSM. Consequently, the update equations are derived from the Kalman filter [43] and are given as

$$\underline{\mu}_{\underline{\boldsymbol{w}}}^+ = \underline{\mu}_{\underline{\boldsymbol{w}}} + \mathbf{K}\left(\underline{y}_n - \underline{\mu}_{\underline{\boldsymbol{y}}_L} \right), \tag{8}$$

$$\mathbf{C}_{\underline{\boldsymbol{w}}}^+ = \mathbf{C}_{\underline{\boldsymbol{w}}} - \mathbf{K}\,\mathbf{H}\,\mathbf{C}_{\underline{\boldsymbol{w}}}, \tag{9}$$

$$\mathbf{K} := \mathbf{C}_{\underline{\boldsymbol{w}}}\ \mathbf{H}^\top \underbrace{\left(\mathbf{H}\ \mathbf{C}_{\underline{\boldsymbol{w}}}\ \mathbf{H}^\top + \mathbf{C}_{\underline{\boldsymbol{\varepsilon}}} \right)}_{\mathbf{C}_{\underline{\boldsymbol{y}}}}{}^{-1} \ ,$$

where $\mathbf{C}_{\underline{\boldsymbol{w}}}$ is given as

$$\mathbf{C}_{\underline{\boldsymbol{w}}} = \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_L} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{w}}_{L-1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{w}}_2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{x}} \end{bmatrix} \end{bmatrix} ,$$

and $\mathbf{C}_{\underline{\boldsymbol{x}}} = \mathbf{0}$ holds since $\underline{x}$ is deterministic. The Kalman gain $\mathbf{K}$ is given as

$$\begin{aligned}
&\begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_L} & & \mathbf{0} \\ \mathbf{0} & & \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_{L-1}} & \mathbf{0} \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \mathbf{0} & \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_{L-2}} & \mathbf{0} \end{bmatrix} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_{L-2}^\top \mathbf{F}_{L-2}^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \vdots & & \vdots \\ \mathbf{0} & \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_2} & \mathbf{0} \end{bmatrix} \prod_{l=2}^{L-2} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_l^\top \mathbf{F}_l^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \\ \mathbf{0} & \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \prod_{l=1}^{L-2} & \begin{bmatrix} \mathbf{0} & \mathbf{G}_l^\top \mathbf{F}_l^\top \end{bmatrix} \mathbf{G}_{L-1}^\top \mathbf{F}_{L-1}^\top \end{bmatrix} \\
&\quad \cdot \mathbf{G}_L^\top \mathbf{F}_L^\top \left(\mathbf{C}_{\underline{\boldsymbol{y}}} \right)^{-1} \ .
\end{aligned}$$

Since the statistical linearization is unbiased in the first two moments, it is obvious that the forward pass of the LSSM is unbiased too. In the following section we will thus prove the equivalence of the backward pass as well.

VII. Proof of Equivalence Between KBNN and Statistical Linearization

Theorem: Both BNN formulations, as presented in Sec. IV-B and Sec. VI, are equivalent in that they yield identical update steps for both the mean and covariance matrix of the weights. Specifically, the update procedures of KBNN, detailed in (1) and (2), and those of the LSSM, outlined in (8) and (9), produce the same results.

Proof: In the LSSM the update for the weight matrix $\underline{\mathbf{W}}_j$ of the j-th layer corresponds to the j-th row of the Kalman gain $\mathbf{K}$. To further analyze the update formula, we now focus on this specific entry in $\mathbf{K}$. For clarity and to simplify the proof, we introduce the following notation:

$$\mathbf{C}_j := \mathbf{C}_{\underline{\boldsymbol{w}}_j \underline{\boldsymbol{z}}_j \underline{\boldsymbol{a}}_j} \left(\mathbf{C}_{\underline{\boldsymbol{a}}_j} \right)^{-1} \mathbf{C}_{\underline{\boldsymbol{a}}_j \underline{\boldsymbol{z}}_{j+1}} \ .$$

Let us consider the first elements of the product of the j-th entry in the Kalman filter gain $\mathbf{K}$, which is given by

$$\begin{aligned}
&\begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_j} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{G}_j^\top \mathbf{F}_j^\top \end{bmatrix} \\
&\overset{(a)}{=} \begin{bmatrix} \mathbf{C}_{\underline{\boldsymbol{w}}_j} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{0} & \begin{bmatrix} \left(\mathbf{C}_{\underline{\boldsymbol{w}}_j} \right)^{-1} & \mathbf{0} \\ \mathbf{0} & \left(\mathbf{C}_{\underline{\boldsymbol{z}}_j} \right)^{-1} \end{bmatrix} \mathbf{C}_j \end{bmatrix} \\
&\overset{(b)}{=} \begin{bmatrix} \mathbf{0} & \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} \mathbf{C}_j \end{bmatrix} \\
&\overset{(c)}{=} \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{C}_j \end{bmatrix} \ .
\end{aligned}$$

Here, (a) substitutes the definitions of $\mathbf{G}_j^\top$ and $\mathbf{F}_j^\top$, (b) simplifies the multiplication, and (c) rewrites the matrix multiplication in block-wise form.

We now focus on the next element in the product of the j-th entry in the Kalman filter gain $\mathbf{K}$:

$$\begin{aligned}
&\begin{bmatrix}\mathbf{0} & \mathbf{C}_j\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{G}_{j+1}^\top \mathbf{F}_{j+1}^\top\end{bmatrix} \\
&= \underbrace{\begin{bmatrix}\mathbf{0} & \mathbf{C}_j\end{bmatrix}}_{\mathbf{I}} \begin{bmatrix}\mathbf{0} & \begin{bmatrix}\left(\mathbf{C}_{\underline{\boldsymbol{w}}_{j+1}}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\end{bmatrix}\mathbf{C}_{j+1}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{0} & \mathbf{C}_j\end{bmatrix}\begin{bmatrix}\left(\mathbf{C}_{\underline{\boldsymbol{w}}_{j+1}}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{C}_{j+1}\end{bmatrix} \\
&= \mathbf{C}_j\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\underbrace{\begin{bmatrix}\mathbf{0} & \mathbf{C}_{j+1}\end{bmatrix}}_{\mathbf{II}},
\end{aligned}$$

where $\mathbf{I}$ and $\mathbf{II}$ represent the same term with different indices, allowing the process to be repeated across all layers. Repeating these steps and combining them yields the final entry of the j-th entry in the Kalman filter gain $\mathbf{K}$. Remember that this result must be multiplied by $(\underline{y}_n - \underline{\mu}_{\underline{\boldsymbol{y}}})$ for some $\underline{y}_n$ from the training data set $\mathcal{D}$ as in (8) and the initial mean $\underline{\mu}_{\underline{\boldsymbol{w}}_j}$ must be added as well. Therefore, the final update formula for the mean of the weights $\underline{\boldsymbol{w}}$ of the statistically linearized BNN is given by

$$\begin{aligned}
\underline{\mu}^+_{\underline{\boldsymbol{w}}_j} &= \underline{\mu}_{\underline{\boldsymbol{w}}_j} + \\
&\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\prod_{l=j}^{L-1}\mathbf{C}_l\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}}\right)^{-1}\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_L\left(\mathbf{C}_{\underline{\boldsymbol{y}}}\right)^{-1}(\underline{y}_n - \underline{\mu}_{\underline{\boldsymbol{y}}}) .
\end{aligned} \tag{10}$$

This completes the calculations for this approach. Next, we examine the update formula for KBNN and show that it matches the derived expression. Since KBNN updates both the weight vector $\underline{\boldsymbol{w}}_j$ and the post-activation $\underline{\boldsymbol{z}}_j$, we extract the relevant parts using projections such as $\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}$ or $\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}$, depending on whether the update is for $\underline{\boldsymbol{w}}_j$ or $\underline{\boldsymbol{z}}_j$. The update for the weight vector $\underline{\boldsymbol{w}}_j$ is thus given by

$$\underline{\mu}^+_{\underline{\boldsymbol{w}}_j} = \underline{\mu}_{\underline{\boldsymbol{w}}_j} + \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\mathbf{C}_j\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}(\underline{\mu}^+_{\underline{\boldsymbol{z}}_{j+1}} - \underline{\mu}_{\underline{\boldsymbol{z}}_{j+1}}) .$$

The update for the post-activation $\underline{\boldsymbol{z}}$ can be extracted using the projection $\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}$. Applying this recursively yields the following update formula:

$$\begin{aligned}
&\underline{\mu}^+_{\underline{\boldsymbol{w}}_j} = \underline{\mu}_{\underline{\boldsymbol{w}}_j} + \\
&\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\mathbf{C}_j\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\prod_{l=j+1}^{L}\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_l\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}}\right)^{-1}(\underline{y}_n - \underline{\mu}_{\underline{\boldsymbol{y}}}) \\
&= \underline{\mu}_{\underline{\boldsymbol{w}}_j} + \\
&\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\prod_{l=j}^{L-1}\mathbf{C}_l\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}}\right)^{-1}\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_L\left(\mathbf{C}_{\underline{\boldsymbol{y}}}\right)^{-1}(\underline{y}_n - \underline{\mu}_{\underline{\boldsymbol{y}}})
\end{aligned}$$

Since this matches the formula in (10), the update for the mean is complete.

We now shift our focus to the covariance matrix update. The KBNN updates the covariance matrix according to:

$$\begin{aligned}
&\begin{bmatrix}\mathbf{C}^+_{\underline{\boldsymbol{w}}_j} & \mathbf{C}_{\underline{\boldsymbol{w}}_j\underline{\boldsymbol{z}}_j} \\ \mathbf{C}^\top_{\underline{\boldsymbol{w}}_j\underline{\boldsymbol{z}}_j} & \mathbf{C}^+_{\underline{\boldsymbol{z}}_j}\end{bmatrix} = \begin{bmatrix}\mathbf{C}_{\underline{\boldsymbol{w}}_j} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{z}}_j}\end{bmatrix} + \\
&\mathbf{C}_j\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\left(\mathbf{C}^+_{\underline{\boldsymbol{z}}_{j+1}} - \mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\mathbf{C}_j^\top
\end{aligned}$$

By again using the projections $\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}, \begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}, \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}^\top$ and $\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}^\top$ one can now extract the update step for $\mathbf{C}^+_{\underline{\boldsymbol{w}}_j}$. It is given as

$$\begin{aligned}
\mathbf{C}^+_{\underline{\boldsymbol{w}}_j} &= \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{C}_{\underline{\boldsymbol{w}}_j} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{z}}_j}\end{bmatrix}\begin{bmatrix}\mathbf{1} \\ \mathbf{0}\end{bmatrix} \\
&\quad + \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\mathbf{C}_j\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\left(\mathbf{C}^+_{\underline{\boldsymbol{z}}_{j+1}} - \mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right) \\
&\quad \cdot \left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\right)^{-1}\mathbf{C}_j^\top\begin{bmatrix}\mathbf{1} \\ \mathbf{0}\end{bmatrix}
\end{aligned}$$

and for $\mathbf{C}^+_{\underline{\boldsymbol{z}}_{j+1}}$ as

$$\begin{aligned}
\mathbf{C}^+_{\underline{\boldsymbol{z}}_{j+1}} &= \begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\begin{bmatrix}\mathbf{C}_{\underline{\boldsymbol{w}}_{j+1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{z}}_{j+1}}\end{bmatrix}\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} + \\
&\quad \begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_{j+1}\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+2}}\right)^{-1}\left(\mathbf{C}^+_{\underline{\boldsymbol{z}}_{j+2}} - \mathbf{C}_{\underline{\boldsymbol{z}}_{j+2}}\right) \\
&\quad \left(\mathbf{C}_{\underline{\boldsymbol{z}}_{j+2}}\right)^{-1}\mathbf{C}_{j+1}^\top\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} .
\end{aligned}$$

Repeating the procedure recursively from layer j to the L-th layer leads to the following update formula for $\mathbf{C}_{\underline{\boldsymbol{w}}_j}$

$$\begin{aligned}
&\mathbf{C}^+_{\underline{\boldsymbol{w}}_j} = \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{C}_{\underline{\boldsymbol{w}}_j} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\underline{\boldsymbol{z}}_j}\end{bmatrix}\begin{bmatrix}\mathbf{1} \\ \mathbf{0}\end{bmatrix} + \\
&\begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\prod_{l=j}^{L-1}\mathbf{C}_l\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}}\right)^{-1}\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_L\left(\mathbf{C}_{\underline{\boldsymbol{y}}}\right)^{-1}\left(\mathbf{0} - \mathbf{C}_{\underline{\boldsymbol{y}}}\right) \\
&\cdot\left(\mathbf{C}_{\underline{\boldsymbol{y}}}\right)^{-1}\mathbf{C}_L^\top\prod_{l=j}^{L-1}\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix}\left(\mathbf{C}_{\underline{\boldsymbol{z}}_{l+1}}\right)^{-1}\mathbf{C}_l^\top\begin{bmatrix}\mathbf{1} \\ \mathbf{0}\end{bmatrix} .
\end{aligned} \tag{11}$$

In the proof for the mean we already showed that the Kalman filter update for the j-th layer is given by

$$\mathbf{K}_j = \begin{bmatrix}\mathbf{1} & \mathbf{0}\end{bmatrix}\prod_{l=j}^{L-1}\mathbf{C}_l\begin{bmatrix}\mathbf{0} & \mathbf{1}\end{bmatrix}\mathbf{C}_L\left(\mathbf{C}_{\underline{\boldsymbol{y}}}\right)^{-1} . \tag{12}$$

Substituting (12) into (11) gives

$$\mathbf{C}^+_{\underline{\boldsymbol{w}}_j} = \mathbf{C}_{\underline{\boldsymbol{w}}_j} - \mathbf{K}_j\,\mathbf{C}_{\underline{\boldsymbol{y}}}\,\mathbf{K}_j^\top ,$$

and by vectorizing this equation, we obtain

$$\begin{aligned}
\mathbf{C}^+_{\underline{\boldsymbol{w}}_j} &= \mathbf{C}_{\underline{\boldsymbol{w}}_j} - \mathbf{K}\,\mathbf{C}_{\underline{\boldsymbol{y}}}\,\mathbf{K}^\top , \\
&= \mathbf{C}_{\underline{\boldsymbol{w}}_j} - \mathbf{K}\,\mathbf{H}\,\mathbf{C}_{\underline{\boldsymbol{w}}} ,
\end{aligned}$$

which matches (9). ■

VIII. Discussion

Recasting BNN training as an LSSM offers several compelling benefits over alternative closed-form approaches such as KBNN, while retaining their non-gradient-based nature.

First, although both LSSM and KBNN perform exact posterior updates via state-space estimation (e.g., Kalman filtering and smoothing), LSSM circumvents the need to explicitly integrate a high-dimensional joint distribution over latent states and observations. Instead, it leverages classical Bayesian filters—Kalman, extended Kalman, unscented Kalman, and related square-root variants [44]—to yield exact, closed-form updates with minimal computational overhead.

Second, the state-space structure of LSSM imposes an inherent regularization that mitigates overfitting, reduces sensitivity to measurement noise, and enhances generalization performance.

Third, LSSM naturally supports *dual estimation* of latent states and model parameters via state augmentation. For instance, consider the system

$$\underline{\boldsymbol{x}}_{k+1} = \underline{a}(\underline{\boldsymbol{x}}_k, \underline{\boldsymbol{\theta}}_k) \ , \quad \underline{\boldsymbol{y}}_k = \underline{h}(\underline{\boldsymbol{x}}_k) + \underline{\boldsymbol{v}}_k \ ,$$

where $a(\cdot)$ is modeled by a BNN. By augmenting the state vector with parameters $\underline{\boldsymbol{\theta}}_k$, one can jointly estimate both $\underline{\boldsymbol{x}}_k$ and $\underline{\boldsymbol{\theta}}_k$ from observations $\underline{\boldsymbol{y}}_k$. This approach is particularly advantageous when the system is partially observable or subject to external disturbances.

Fourth, the incremental-update capability of LSSM makes it ideally suited for online and streaming applications. New data can be assimilated without retraining from scratch, enabling real-time inference and adaptation.

Fifth, the explicit state-space formulation enhances interpretability. By viewing parameter evolution as a latent dynamical process, one can trace the transformation of priors into posteriors over time. This transparency stands in stark contrast to the opaque weight updates of conventional deep learning, facilitating both theoretical analysis and practical diagnostics.

Finally, embedding BNN training within the rich ecosystem of Bayesian state-estimation techniques opens the door to a vast array of methodological innovations—ranging from robust filtering to adaptive noise modeling—that can be directly applied to improve training stability and performance.

Taken together, these advantages establish the LSSM-based paradigm as a robust, interpretable, and computationally efficient alternative for BNN training, especially in contexts where stability and real-time adaptability are paramount.

IX. Conclusion

In summary, reformulating Bayesian Neural Network (BNN) training within an linear state-space model (LSSM) framework provides an efficient, stable, and interpretable approach to probabilistic modeling. By leveraging state-space estimation techniques, it reduces computational complexity, enhances numerical stability, and enables efficient online learning. The structured nature of the formulation further mitigates overfitting while offering clearer insights into parameter dynamics. These advantages collectively position LSSM-based training as a powerful alternative to conventional approaches.

Future work will focus on refining key aspects of this framework. Specifically, we aim to investigate alternative linearization methods and their impact on weight updates, as well as strategies for optimizing the linearization matrix $\mathbf{H}$ to enhance stability and convergence. Through these refinements, we seek to extend the applicability of this approach to a broader range of learning scenarios.

Moreover, because both recurrent- and convolutional neural networks rely primarily on linear transformations (aside from operations such as max-pooling), this framework can be straightforwardly extended to both architectures.

References

[1] A. Tealab, "Time series forecasting using artificial neural networks methodologies: A systematic review," *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 334–340, Dec. 2018.

[2] L. Su, X. Zuo, R. Li, X. Wang, H. Zhao, and B. Huang, "A systematic review for transformer-based long-term series forecasting," *Artificial Intelligence Review*, vol. 58, no. 3, p. 80, Jan. 2025.

[3] M. Walker and U. D. Hanebeck, "Multi-scale uncertainty calibration testing for Bayesian neural networks using ball trees," in *Proceedings of the 2024 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2024)*, Plzeň, Czech Republic, Sep. 2024, pp. 1–7.

[4] M. Walker, M. Reith-Braun, P. Schichtel, M. Knaak, and U. D. Hanebeck, "Identifying trust regions of Bayesian neural networks," in *Proceedings of the 2023 IEEE Symposium Sensor Data Fusion and International Conference on Multisensor Fusion and Integration (SDF-MFI)*, Bonn, Germany, Nov. 2023, pp. 1–8.

[5] F. Küppers, J. Schneider, and A. Haselhoff, "Parametric and multivariate uncertainty calibration for regression and object detection," in *Computer Vision – ECCV 2022 Workshops*, 2023, vol. 13805, pp. 426–442.

[6] M. Walker, P. S. Bien, and U. D. Hanebeck, "Voronoi trust regions for local calibration testing in supervised machine learning models," in *Proceedings of the 2024 Sensor Data Fusion: Trends, solutions, applications (SDF 2024)*, Bonn, Germany, Nov. 2024, pp. 1–8.

[7] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[8] A. Graves, "Practical variational inference for neural networks," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, 2011, pp. 2348–2356.

[9] T. P. Minka, "A family of algorithms for approximate Bayesian inference," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.

[10] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[11] P. Wagner, X. Wu, and M. F. Huber, "Kalman Bayesian neural networks for closed-form online learning," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023.

[12] M. F. Huber, "Bayesian Perceptron: Towards fully Bayesian neural networks," in *Proceedings of the 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 3179–3186.

[13] M. Walker, H. Amirkhanian, M. F. Huber, and U. D. Hanebeck, "Trustworthy Bayesian perceptrons," in *Proceedings of the 2024 27th International Conference on Information Fusion (FUSION 2024)*, Venice, Italy, Jul. 2024, pp. 1–8.

[14] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 721–741, Nov. 1984.

[15] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Physics Letters B*, vol. 195, no. 2, pp. 216–222, 1987.

[16] R. M. Neal, *Bayesian Learning for Neural Networks*, ser. Lecture Notes in Statistics. New York, NY: Springer, 1996, vol. 118.

[17] M. D. Homan and A. Gelman, "The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, Jan. 2014.

[18] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of the 28th International Conference on Machine Learning*, Madison, WI, USA, 2011, pp. 681–688.

[19] S. Ahn, B. Shahbaba, and M. Welling, "Distributed stochastic gradient MCMC," in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., vol. 32, Bejing, China, Jun. 2014, pp. 1044–1052.

[20] R. Zhang, A. G. Wilson, and C. De Sa, "Low-precision stochastic gradient Langevin dynamics," in *Proceedings of the 39th international conference on machine learning*, vol. 162, 2022, pp. 26 624–26 644.

[21] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, May 2013.

[22] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt, "Deterministic variational inference for robust Bayesian neural networks," in *International Conference on Learning Representations*, 2019.

[23] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., vol. 37, Lille, France, Jul. 2015, pp. 1613–1622.

[24] T. Heskes and O. Zoeter, "Extended version: Expectation Propagation for approximate inference in dynamic Bayesian networks," University of Nijmegen, Tech. Rep., 2003.

[25] T. P. Minka, "Power EP," Microsoft Research, Tech. Rep. MSR-TR-2004-149, 2004.

[26] J. M. Hernández-Lobato and R. P. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, Jul. 2015, pp. 1861–1869.

[27] H. Ritter, A. Botev, and D. Barber, "A scalable Laplace approximation for neural networks," in *International Conference on Learning Representations*, 2018.

[28] E. Daxberger, E. Nalisnick, J. U. Allingham, J. Antoran, and J. M. Hernandez-Lobato, "Bayesian deep learning via subnetwork inference," in *Proceedings of the 38th International Conference on Machine Learning*, vol. 139, Jul. 2021, pp. 2510–2521.

[29] A. Kristiadi, M. Hein, and P. Hennig, "Being Bayesian, even just a bit, fixes overconfidence in ReLU networks," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, Jul. 2020, pp. 5436–5446.

[30] K. Watanabe and S. G. Tzafestas, "Learning algorithms for neural networks with the Kalman filters," *Journal of Intelligent and Robotic Systems*, vol. 3, no. 4, pp. 305–319, 1990.

[31] G. V. Puskorius and L. A. Feldkamp, "Parameter-based Kalman filter training: Theory and Implementation," in *Kalman Filtering and Neural Networks*, 2001, pp. 23–67.

[32] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000, pp. 153–158.

[33] C. Chen, X. Lin, Y. Huang, and G. Terejanu, "Approximate Bayesian neural network trained with ensemble Kalman filter," in *Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8.

[34] J.-A. Goulet, L. H. Nguyen, and S. Amiri, "Tractable approximate Gaussian inference for Bayesian neural networks," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 11 374–11 396, Jan. 2021.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[36] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, New York, NY, USA, 2016, pp. 1050–1059.

[37] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[38] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6405–6416.

[39] I. Harris, "Predictive fit for natural exponential functions," *Biometrika*, vol. 76, pp. 675–684, Dec. 1989.

[40] T. Fushiki, F. Komaki, and K. Aihara, "Nonparametric bootstrap prediction," *Bernoulli*, vol. 11, no. 2, pp. 293–307, 2005.

[41] W. J. Maddox, T. Garipov, P. Izmailov, D. Vetrov, and A. G. Wilson, "A simple baseline for Bayesian uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019.

[42] S. Särkkä and L. Svensson, *Bayesian filtering and smoothing*, 2nd ed., ser. Institute of Mathematical Statistics Textbooks. Cambridge: Cambridge University Press, 2023.

[43] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[44] P. Kaminski, A. Bryson, and S. Schmidt, "Discrete square root filtering: A survey of current techniques," *IEEE Transactions on automatic control*, vol. 16, no. 6, pp. 727–736, 1971.