Stein Variational Newton Neural Network Ensembles

Klemens Flöge [1 2] **Muhammad Abdul Moeed** [1] **Vincent Fortuin** [1 2]

Abstract

Deep neural network ensembles are powerful tools for uncertainty quantification, which have recently been re-interpreted from a Bayesian perspective. However, current methods inadequately leverage second-order information of the loss landscape, despite the recent availability of efficient Hessian approximations. We propose a novel approximate Bayesian inference method that modifies deep ensembles to incorporate Stein Variational Newton updates. Our approach uniquely integrates scalable modern Hessian approximations, achieving faster convergence and more accurate posterior distribution approximations. We validate the effectiveness of our method on diverse regression and classification tasks, demonstrating superior performance with a significantly reduced number of training epochs compared to existing ensemble-based methods, while enhancing uncertainty quantification and robustness against overfitting.

1. Introduction

Effectively approximating intractable probability distributions with finite samples is crucial for evaluating expectations and characterizing uncertainties in machine learning and statistics (Pearson, 1895; Silverman, 1986; Givens & Hoeting, 2012). This challenge is particularly pronounced in high-dimensional and multimodal distributions, such as those in approximate Bayesian inference for deep neural networks. Traditional methods to address these challenges include parametric variational inference (VI; Blei et al., 2017) and Markov chain Monte Carlo (MCMC; Andrieu et al., 2003), each with tradeoffs between computational efficiency and estimation accuracy.

Stein Variational Gradient Descent (SVGD; Liu & Wang, 2016) is a potent non-parametric VI technique that balances efficiency and performance by coupling a tractable reference distribution with a complex target distribution via transport maps. Enhancing SVGD with curvature information results in Stein Variational Newton (SVN; Detommaso et al., 2018), which, despite the increased cost of Hessian computations, achieves significantly faster convergence. However, until now, SVN's application has been limited to low-dimensional inference problems.

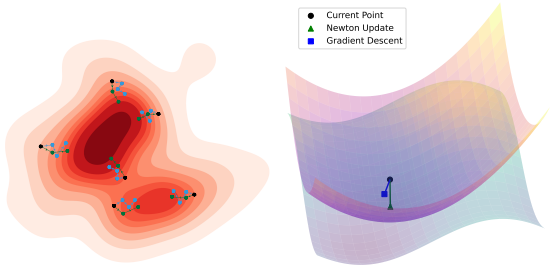


Figure 1. Conceptual overview of the SVN method. The green curvature-informed SVN updates are much higher quality and require fewer steps than the corresponding blue SVGD ones.

Since the advent of neural networks, researchers have applied Bayesian principles to them (BNNs; Neal, 1995). SVGD has also been extended to neural networks (D'Angelo et al., 2021), however, for SVN, this still remains elusive. Li et al. (2018) showed that deep neural network loss landscapes are complex yet structured (Garipov et al., 2018), suggesting that curvature information can improve learning dynamics (Lin et al., 2023; 2024; Eschenhagen et al., 2024). Therefore, we aim to extend SVN to high-dimensional neural network posteriors.

The recent resurgence of interest in second-order methods in machine learning stems from more efficient Hessian approximations (Heskes, 2000; Botev et al., 2017; Martens & Grosse, 2015) and the availability of plug-and-play software libraries (Dangel et al., 2020; Osawa et al., 2023). These advances have made Hessians more practical, particularly in the Laplace approximation for Bayesian deep learning (MacKay, 1992a; Laplace, 1774), now available as a PyTorch package (Daxberger et al., 2021a). This progress has motivated us to re-evaluate the use of Hessians in ensemble-based Bayesian inference methods.

[1]Department of Computer Science, Technical University of Munich, Munich, Germany [2]Helmholtz AI, Munich, Germany. Correspondence to: Klemens Flöge <klemens.floege@helmholtz-munich.de>.

Accepted by the Structured Probabilistic Inference & Generative Modeling workshop of ICML 2024, Vienna, Austria. Copyright 2024 by the author(s).

The primary contribution of this work is demonstrating SVN's capabilities in BNNs through extensive empirical studies and providing a user-friendly guide and modular codebase for practical deployment. Our adaptation, Stein Variational Newton Neural Network Ensembles, incorporates local curvature information of the posterior landscape using modern, scalable Hessian approximations, achieving faster convergence than traditional gradient-based methods. Additionally, we use geometry-aware kernels within the SVN framework to enhance convergence and performance. Figure 1 illustrates how SVN ensembles make more informed gradient updates by incorporating second-order information.

Our main contributions are to

- Incorporate modern, scalable Hessian approximations into particle-based inference
- Extend SVN from simple low-dimensional problems to deep neural network posteriors, including with a scalable last-layer version
- Demonstrate effective BNN inference on various synthetic and real-world datasets

2. Stein Variational Newton Neural Network Ensembles

The SVN algorithm (Detommaso et al., 2018) enhances SVGD (Liu & Wang, 2016) by incorporating curvature or Hessian information. While Hessian approximations are more computationally involved, they significantly improve convergence rates and stability by leveraging second-order optimization principles. We will first discuss these techniques in deep learning, focusing on Newton's method. Next, we will explore various approaches for modern Hessian approximations. Finally, we will introduce the Stein Variational Newton (SVN) algorithm. Consider a dataset $\mathcal{D} = \{(x_j \in \mathbb{R}^m, y_j \in \mathbb{R}^k)\}_{j=1}^n$ and a neural network model $g_\phi : \mathbb{R}^m \to \mathbb{R}^k$, parameterized by ϕ. Let $\ell : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ be the loss function, e.g., mean squared error, or negative log-likelihood. The empirical risk for a batch of data $\mathcal{B} = \{(x_i \in \mathbb{R}^m, y_i \in \mathbb{R}^k)\}_{i=1}^b \subset \mathcal{D}$, with $|\mathcal{B}| = b$, is defined as:

$$\mathcal{L}(\mathcal{B};\phi) = \mathbb{E}_{\tilde{y}\sim p(y|g_\phi(x))}\left[\ell(g_\phi(x), y)\right] \tag{1}$$

$$\approx \frac{1}{b}\sum_{i=1}^{b} \ell(g_\phi(x_i), y_i). \tag{2}$$

Newton's method optimizes the empirical risk using a second-order Taylor approximation:

$$\mathcal{L}(\mathcal{B};\phi) \approx \mathcal{L}(\mathcal{B};\phi_t) + (\phi-\phi_t)^\top \nabla_\phi \mathcal{L}(\mathcal{B};\phi_t) \tag{3}$$

$$+ \frac{1}{2}(\phi-\phi_t)^\top \mathbf{H}_{\phi_t}(\phi-\phi_t), \tag{4}$$

at timestep t, where

$$\mathbf{H}_{\phi_t} = \nabla^2_\phi \mathcal{L}(\mathcal{B};\phi_t) \approx \frac{1}{b}\sum_{i=1}^{b} \nabla^2_\phi \ell(g_\phi(x_i), y_i), \tag{5}$$

is the Hessian matrix of $\mathcal{L}$ with respect to ϕ, evaluated at ϕ_t for a given batch $\mathcal{B}$. Following Goodfellow et al. (2016), the Newton parameter update would then be defined as:

$$\phi_{t+1} = \phi_t - \mathbf{H}^{-1}_{\phi_t} \nabla_\phi \mathcal{L}(\mathcal{B};\phi_t). \tag{6}$$

This update step adjusts parameters by considering both the gradient and curvature of the empirical risk function. For locally quadratic functions with a positive definite Hessian, rescaling the gradient by $\mathbf{H}^{-1}$ allows Newton's method to move directly to the minimum. For convex functions with higher-order terms, iterative updates lead to faster and more stable convergence than first-order methods, a feature that we visualize in Figure 1 and demonstrate empirically for SVN Ensembles in Section 3.

2.1. Hessian Approximations in Deep Learning

As previously noted, a crucial component in implementing the SVN algorithm for neural networks is the Hessian matrix $\mathbf{H}_\phi$ (Detommaso et al., 2018). However, directly computing the Hessian, as defined in Equation (5), is often impractical due to its quadratic scaling with the number of network parameters. Therefore, a good approximation is necessary. A positive-definite approximation is particularly beneficial for optimization purposes, as mentioned above. In this context, we will consider the Fisher Information Matrix (FIM) (Amari, 1998) defined for batch $\mathcal{B}$ as follows:

$$\mathbf{F}_\phi = \sum_{i=1}^{b} \mathbb{E}_{\tilde{y}\sim p(y|g_\phi(x_i))}\left[J_\phi(\tilde{y}|x_i)\, J_\phi(\tilde{y}|x_i)^\top\right], \tag{7}$$

with $J_\phi(\tilde{y}|x_i) = \nabla_\phi \log p(\tilde{y}|g_\phi(x_i))$. It is also possible to use the Generalized-Gauss-Netwon (GGN) approximation (Schraudolph, 2002):

$$\mathbf{G}_\phi = \sum_{i=1}^{b} \nabla_\phi g_\phi(x_i)\left(\nabla^2_\phi \log p(y_i|g_\phi(x_i))\right)\nabla_\phi g_\phi(x_i)^\top. \tag{8}$$

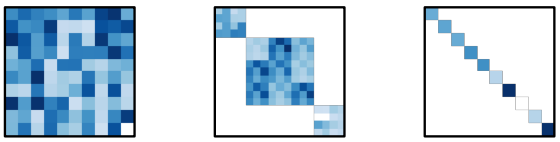


Figure 2. Overview of the Hessian approximations used in our SVN algorithm.

For most common likelihoods, and the ones we consider in this paper, the FIM and GGN are equivalent (Martens, 2020). For the deep Laplace approximation, these have emerged as the default choices (Ritter et al., 2018b;a; Kristiadi et al., 2020; Lee et al., 2020; Daxberger et al., 2021b; Immer et al., 2021b) and are thus the ones we implement in this work. While $\mathbf{F}$ and $\mathbf{G}$ can be estimated efficiently, their entries are still quadratic in the number of parameters. We thus consider three Hessian approximations in our work. **Full** represents the matrices as defined in Equation (7) and Equation (8). The **Diagonal** (Denker & LeCun, 1990; LeCun et al., 1989) approximation considers only diagonal entries of these matrices. Lastly, the Kronecker-Factored Approximate Curvature (**KFAC** or **Kron**; Heskes, 2000; Martens & Grosse, 2015; Botev et al., 2017) considers block-diagonal entries that correspond to the layers of the neural network. These are represented as lightweight Kronecker factors. The KFAC can be improved by using low-rank approximation factors (Lee et al., 2020) and leveraging the eigendecomposition (George et al., 2018). An overview of these methods is shown in Figure 2. They are efficiently implemented for NNs by Daxberger et al. (2021a).

2.2. Particle-based BNN inference

The goal of BNN inference is to approximate the intractable Bayesian posterior distribution $p(\phi \mid \mathcal{D})$ on $\mathbb{R}^d$, abbreviated as π, where d is the dimensionality of ϕ and number of parameters in the neural network g_ϕ. In particle-based BNN inference, π is approximated using an ensemble of N models $\{g_{\phi_1}, \cdots, g_{\phi_N}\}$, which can be written as:

$$\rho(\phi) = \frac{1}{N} \sum_{i=1}^{N} \delta_{\phi_i}(\phi), \tag{9}$$

where $\delta_\phi(x)$ denotes the Dirac delta function centered at ϕ. In **Deep Ensembles** (Lakshminarayanan et al., 2017), each neural network is trained independently with MAP training. Due to the stochasticity of neural network training, $g_{\phi_i} \neq g_{\phi_j}$, for $i \neq j$. A major breakthrough in approximate Bayesian inference was **Stein Variational Gradient Descent** (Liu & Wang, 2016), which defines a sequence of transport maps $T = T_1 \circ \cdots \circ T_k \circ \cdots$ to minimize $\mathcal{D}_{\mathrm{KL}}(T(\rho) \| \pi)$. This method updates the particles at each iteration to minimize the Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) between the true posterior π and our approximation $T(\rho)$. The update rule T_l is chosen to be a perturbation of the identity mapping in the vector-valued RKHS $\mathcal{H}^d = \mathcal{H} \times \cdots \times \mathcal{H}$. The sequence of maps $\{T_1, T_2, \cdots, T_l, \cdots\}$ then corresponds to gradient descent with respect to the KL divergence. Following Liu & Wang (2016), T_l, with $\phi_i^{l+1} = T_l \phi_i^l$, can be expressed explicitly as: $T_l = I - \epsilon\, v_l^{\mathrm{SVGD}}(\cdot)$, where

$$v_l^{\mathrm{SVGD}}(\phi) = \frac{1}{N} \sum_{j=1}^{N} \underbrace{k(\phi_j^l, \phi)\, \nabla_{\phi_j^l} \log \pi(\phi_j^l)}_{\text{weighted average steepest descent direction}} \tag{10}$$

$$+ \underbrace{\nabla_{\phi_j^l} k(\phi_j^l, \phi)}_{\text{repulsive force}} \tag{11}$$

Since I is the identity mapping, $v_l^{\mathrm{SVGD}}(\cdot)$ serves as a gradient update. The first term moves particles towards high-probability regions in π, while the second term acts as a 'repulsive force,' dispersing particles to prevent clustering around a single mode. SVGD was extended to neural network ensembles by D'Angelo et al. (2021), and we will compare our results to this method in Section 3.

2.3. Stein Variational Newton

The SVN algorithm, as proposed by Detommaso et al. (2018), enhances SVGD by performing steepest gradient descent in the functional Newton direction. This is represented by the transport map $T_l = I - \epsilon\, v_l^{\mathrm{SVN}}(\cdot)$ at time step l. The SVN update can be expressed as a Galerkin approximation in the finite-dimensional linear space $\mathcal{H}_N^d = \mathrm{span}\{k(\phi_1, \cdot), \cdots, k(\phi_N, \cdot)\}$. The j-th component of v^{SVN} then corresponds to:

$$v_j^{\mathrm{SVN}}(\phi) = \sum_{k=1}^{N} \alpha_j^k\, k(\phi_k, \phi) \tag{12}$$

We use the Hessians $\mathbf{H}_{\phi_1}, \cdots, \mathbf{H}_{\phi_N}$ to solve for the alpha vector $\vec{\alpha}_j = (\alpha_j^1, \cdots, \alpha_j^N)^\top$ for each particle ϕ_j. Following the framework in Gallego & Insua (2020) and Leviyev et al. (2022)'s SVN formulation, we define the kernel matrix $K \in \mathbb{R}^{Nd \times Nd}$ as:

$$K = \frac{1}{N} \begin{pmatrix} k(\phi_1, \phi_1)\, I_{d\times d} & \cdots & k(\phi_1, \phi_N)\, I_{d\times d} \\ \vdots & \ddots & \vdots \\ k(\phi_N, \phi_1)\, I_{d\times d} & \cdots & k(\phi_N, \phi_N)\, I_{d\times d} \end{pmatrix}, \tag{13}$$

where N refers to the number of particles as before, and d is the number of parameters in each ensemble member. Therefore, a vector of length Nd can represent the whole ensemble. Furthermore, for $1 \leq m, n \leq N$, we define the matrix blocks $h^{m,n} \in \mathbb{R}^{d \times d}$,

$$h^{m,n} := \frac{1}{N} \sum_{p=1}^{N} \left[-k\left(\phi_p, \phi_m\right) k\left(\phi_p, \phi_n\right) \mathbf{H}_{\phi_p} \right. \tag{14}$$

$$\left. + \nabla_{\phi_p}\left(k\left(\phi_p, \phi_n\right)\right) \nabla_{\phi_p}\left(k\left(\phi_p, \phi_m\right)\right)^{\top} \right]. \tag{15}$$

The first term in the equation above can be regarded as a weighted kernel average of the Hessians in Equation (6). The second term corresponds to a repulsive force between particles in the RKHS similar to Equation (11). Denoting the SVN-Hessian as

$$H^{\text{SVN}} = \begin{bmatrix} h^{11} & \cdots & h^{1N} \\ \vdots & \ddots & \vdots \\ h^{N1} & \cdots & h^{NN} \end{bmatrix}, \tag{16}$$

computing the SVN update amounts to solving the following linear system of Nd equations:

$$H^{\text{SVN}} \alpha = v^{\text{SVGD}}. \tag{17}$$

Finally, solving for $\alpha \in \mathbb{R}^{Nd}$, the SVN update vector corresponding to the functional Newton direction of steepest descent of the KL divergence can be calculated as:

$$v^{\text{SVN}} = NK\alpha. \tag{18}$$

2.4. SVN Ensembles Algorithm

Algorithm 1 summarizes SVN neural network ensembles. While we demonstrate in Section 3 that the method above is computationally feasible and yields better results for moderately-sized neural networks with the full Hessian approximation, the linear system in Equation (17) has Nd equations and needs to be computed for every gradient step. Moreover, a naïve implementation of K scales with $\mathcal{O}(N^2 d^2)$ in memory terms, which is extremely prohibitive for larger deep learning models.

To this end, Detommaso et al. (2018) already proposed two modifications to the above procedure to improve its scalability. The first approach is using the Newton-conjugate-gradient (NCG) (Wright & Nocedal, 1999, Chapters 5 and 7), which approximates Equation (17). Furthermore, this approach only requires evaluating the matrix-vector product $H^{\text{SVN}}\alpha$, without explicitly constructing the matrix. This

Algorithm 1 One iteration of our SVN neural network ensemble algorithm

Input: Particles $\{\phi_i^l\}_{i=1}^N$ at stage l; step size ε, kernel $k(\cdot,\cdot) \in \{k^{M_{\text{SVN}}}, k^I\}$, Hessian approximation $\mathbf{H} \in \{\mathbf{H}_{\text{Full}}, \mathbf{H}_{\text{KFAC}}, \mathbf{H}_{\text{Diag}}\}$
Output: Particles $\{\phi_i^{l+1}\}_{i=1}^N$ at stage $l+1$
Compute v_l^{SVGD}
for $i = 1, 2, \ldots, n$ **do**
  if Block Diagonal Approximation **then**
    Solve the linear system from Equation (19) for $\alpha^1, \ldots, \alpha^n$
  else
    Solve the linear system from Equation (17) for $\alpha^1, \ldots, \alpha^n$
  end if
  Set $\phi_i^{k+1} \leftarrow \phi_i^k + \varepsilon\, v_l^{\text{SVN}}\left(\phi_i^k\right)$ given $\alpha^1, \ldots, \alpha^n$
end for

allows us to efficiently use the KFAC and Diagonal approximations detailed in Section 2.1. Additionally, one can consider a block diagonal approximation to Equation (17), where the off-diagonal matrix blocks in Equation (16) are disregarded, and we solve the following linear system for each particle:

$$h^{m,m}\alpha_m = v_m^{\text{SVGD}} \quad \text{for} \quad m = 1, \cdots, N. \tag{19}$$

This computes α_m for each particle independently and involves only d equations. Applying the inverse of the SVN Hessian matrix block $h^{m,m}$ to both sides of Equation (19) makes the computation of the α vector in kernel space similar to the Newtonian parameter update vector discussed in Equation (6) at the beginning of this section.

Hessian kernel. Our Hessian approximation $\mathbf{H}_\phi \approx \nabla^2 \log \pi$ characterizes the local curvature of the posterior. Following Detommaso et al. (2018), we consider the average curvature metric of the posterior π with respect to our ensemble:

$$M_\pi = \mathbb{E}_{\phi\sim\pi}[\nabla_\phi^2 \log \pi(\phi)] \approx \frac{1}{N}\sum_{j=1}^{N} \mathbf{H}_{\phi_j} = M_{SVN}, \tag{20}$$

and use it to construct an anisotropic Gaussian Kernel (Liu & Wang, 2016):

$$k_l^M(\phi, \phi') := \exp\left(-\frac{1}{2d}\|\phi - \phi'\|_{M_{\text{SVN}}}\right), \tag{21}$$

with $\|\phi\|_{M_{\text{SVN}}} = \phi^\top M_{\text{SVN}}\, \phi$. In Appendix D.2, we perform

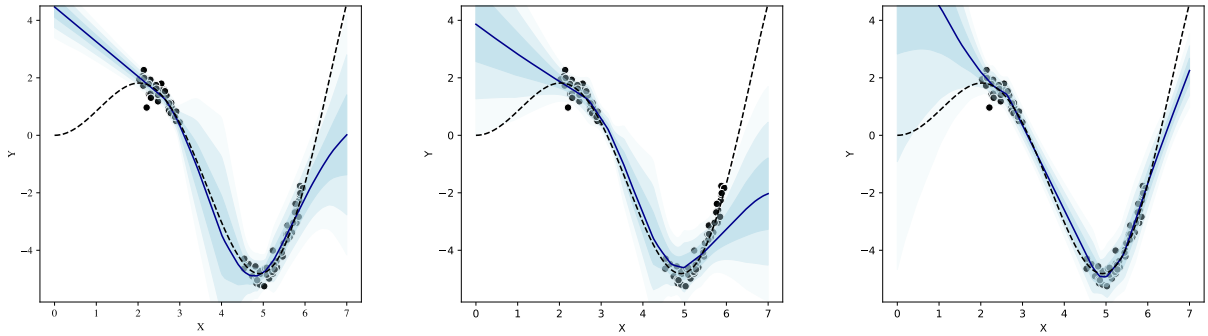


Figure 3. Synthetic regression example for Ensemble, SVGD, and SVN methods. The training data is marked with black dots, and the true function is represented with a dashed line. The predictive mean of the neural network ensemble is shown in dark blue, with the standard deviations highlighted in light blue. SVN best captures the underlying data distribution.

ablation studies between using $M = M_{\text{SVN}}$ and simply using the scaled Gaussian kernel with $M = I$.

LL-SVN. Inspired by recent work on subnetwork inference (Daxberger et al., 2021b) and competitive performance of the last-layer Laplace approximation (Daxberger et al., 2021a), we propose a novel and computationally efficient modification to Algorithm 1, called **LL-SVN**, where the SVN update v^{SVN} is performed only for the last layer of the network, while v^{SVGD} is used for the other layers. This approach leverages high-quality curvature information in the most expressive part of the neural network, better utilizing the learned representation. We compute v_l^{SVGD} for the entire network, then use the last d_{ll} elements, corresponding to the last layer, to compute the SVN vector v_l^{SVGD}. Algorithm 2 in Appendix B summarizes this procedure.

3. Experiments

We evaluate the proposed Stein Variational Newton neural network ensemble (**SVN**) on a diverse range of synthetic and real-world datasets, demonstrating its competitive or superior performance for BNN inference. Our approach is benchmarked against other particle-based VI methods, including deep ensembles (**Ensemble**) (Lakshminarayanan et al., 2017), repulsive ensembles (D'Angelo & Fortuin, 2021b) (specifically, the weight-space version (**WGD**)), and Stein Variational Gradient Descent (**SVGD**) (Liu & Wang, 2016) for neural network ensembles (D'Angelo et al., 2021). All these methods involve different training schemes for a collection of neural networks. Thus, the resulting ensemble model, denoted as $\{g_{\phi_1}, \cdots, g_{\phi_N}\}$, can be uniformly evaluated across all methods. Note, however, that deep ensembles do not asymptotically converge to the true posterior as $N \to \infty$.

3.1. Toy Regression

As a sanity check, we start with a synthetic one-dimensional regression problem using a self-generated dataset. Details on data generation are in Appendix C.1. In Figure 3, we plot the predictive mean of the ensemble, $\bar{g} = \frac{1}{N}\sum_{i=1}^{N} g_{\phi_i}$, along with the predictive mean plus or minus three standard deviations in increasingly lighter shades of blue. Training data points are in black, and the true function is represented with a dashed line. The plots show that while the ensemble and SVGD yield better uncertainties between the two clusters, they are underconfident within the clusters and overconfident outside the data region. In contrast, SVN's predictions and uncertainties best match the underlying function and given data, capturing the data distribution most effectively.

3.2. UCI Benchmark

To demonstrate the effectiveness of SVN on real-world datasets, we follow Hernández-Lobato & Adams (2015) and consider several regression and binary classification datasets from the UCI Machine Learning Repository (Dua & Graff, 2017). We use 5-fold cross-validation, combining the training and test data. Additionally, we split off 20% of the training data for validation. We report the mean performance metrics along with the standard error. The primary metric considered is the Gaussian negative log-likelihood (NLL) (Nix & Weigend, 1994) on the test dataset. In the regression tasks of Figure 4, SVN generally outperforms other methods, except for the naval dataset. The corresponding MSE and NLL results are listed in Table 5 and Table 4 in the Appendix. Next, we evaluate SVN on the UCI binary classification tasks using standard metrics: accuracy, NLL, and AUROC. We also include calibration metrics such as the Expected Calibration Error (ECE) (Guo et al., 2017) and the Brier score (Brier, 1950), which has been argued to offer

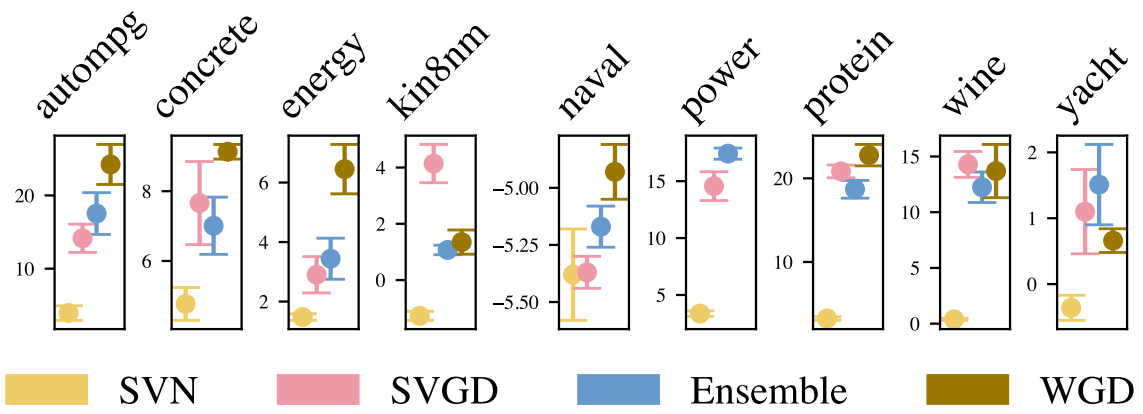


Figure 4. Test negative log-likelihood on UCI regression datasets. We truncated the power plot as a result of WGD's inferior performance. Our proposed SVN method outperforms the ensemble, WGD, and SVGD on all datasets except for naval.

many benefits over the ECE (Gruber & Buettner, 2024). As shown in Table 1, SVN performs competitively across all five tasks, outperforming the baselines on three datasets with respect to NLL. Further details on this experiment can be found in Appendix C.2.

Table 1. Classification metrics on test datasets of the UCI Binary Classification tasks. Our method is competitive on all datasets and outperforms the baselines on three datasets in terms of NLL.

Dataset	Methods	ACC ↑	NLL ↓	ECE ↓	Brier ↓	AUROC ↑
australian	Ensemble	$\mathbf{0.84_{0.01}}$	$1.04_{0.12}$	$\mathbf{0.17_{0.01}}$	$\mathbf{0.24_{0.01}}$	$\mathbf{0.91_{0.01}}$
	WGD	$\mathbf{0.84_{0.01}}$	$1.05_{0.10}$	$\mathbf{0.16_{0.01}}$	$\mathbf{0.24_{0.02}}$	$\mathbf{0.92_{0.01}}$
	SVGD	$\mathbf{0.84_{0.02}}$	$1.03_{0.10}$	$\mathbf{0.15_{0.01}}$	$\mathbf{0.24_{0.02}}$	$\mathbf{0.92_{0.01}}$
	SVN	$\mathbf{0.84_{0.01}}$	$\mathbf{0.62_{0.06}}$	$\mathbf{0.15_{0.01}}$	$\mathbf{0.26_{0.01}}$	$\mathbf{0.90_{0.01}}$
breast	Ensemble	$\mathbf{0.97_{0.00}}$	$0.23_{0.05}$	$\mathbf{0.03_{0.00}}$	$\mathbf{0.05_{0.00}}$	$\mathbf{0.99_{0.00}}$
	WGD	$0.96_{0.00}$	$0.29_{0.08}$	$0.04_{0.00}$	$\mathbf{0.05_{0.00}}$	$0.96_{0.02}$
	SVGD	$\mathbf{0.97_{0.00}}$	$0.38_{0.13}$	$\mathbf{0.03_{0.00}}$	$\mathbf{0.05_{0.00}}$	$0.98_{0.00}$
	SVN	$\mathbf{0.97_{0.00}}$	$\mathbf{0.13_{0.02}}$	$0.04_{0.00}$	$\mathbf{0.05_{0.01}}$	$0.97_{0.00}$
ionosphere	Ensemble	$\mathbf{0.86_{0.02}}$	$\mathbf{0.40_{0.04}}$	$\mathbf{0.14_{0.01}}$	$\mathbf{0.21_{0.02}}$	$\mathbf{0.94_{0.01}}$
	WGD	$0.82_{0.01}$	$1.31_{0.11}$	$0.18_{0.01}$	$0.28_{0.01}$	$\mathbf{0.92_{0.01}}$
	SVGD	$\mathbf{0.86_{0.01}}$	$\mathbf{0.42_{0.04}}$	$\mathbf{0.15_{0.01}}$	$\mathbf{0.21_{0.02}}$	$\mathbf{0.92_{0.01}}$
	SVN	$\mathbf{0.86_{0.01}}$	$\mathbf{0.40_{0.04}}$	$\mathbf{0.16_{0.01}}$	$\mathbf{0.21_{0.02}}$	$\mathbf{0.93_{0.01}}$
parkinsons	Ensemble	$0.85_{0.03}$	$\mathbf{0.43_{0.04}}$	$0.18_{0.01}$	$0.26_{0.03}$	$0.85_{0.03}$
	WGD	$\mathbf{0.91_{0.03}}$	$\mathbf{0.56_{0.27}}$	$\mathbf{0.09_{0.03}}$	$\mathbf{0.14_{0.05}}$	$\mathbf{0.92_{0.03}}$
	SVGD	$\mathbf{0.92_{0.03}}$	$\mathbf{0.47_{0.21}}$	$\mathbf{0.09_{0.03}}$	$\mathbf{0.14_{0.05}}$	$\mathbf{0.97_{0.02}}$
	SVN	$\mathbf{0.91_{0.03}}$	$\mathbf{0.29_{0.10}}$	$\mathbf{0.11_{0.02}}$	$\mathbf{0.15_{0.05}}$	$\mathbf{0.94_{0.03}}$
heart	Ensemble	$\mathbf{0.78_{0.02}}$	$1.22_{0.09}$	$\mathbf{0.21_{0.01}}$	$\mathbf{0.37_{0.02}}$	$\mathbf{0.86_{0.01}}$
	WGD	$0.76_{0.01}$	$2.02_{0.19}$	$\mathbf{0.22_{0.01}}$	$\mathbf{0.38_{0.02}}$	$\mathbf{0.87_{0.01}}$
	SVGD	$\mathbf{0.79_{0.01}}$	$2.14_{0.22}$	$\mathbf{0.21_{0.01}}$	$\mathbf{0.36_{0.01}}$	$\mathbf{0.86_{0.01}}$
	SVN	$\mathbf{0.77_{0.02}}$	$\mathbf{0.75_{0.03}}$	$\mathbf{0.21_{0.01}}$	$\mathbf{0.36_{0.02}}$	$\mathbf{0.86_{0.01}}$

3.3. Computer Vision Benchmark

In order to further investigate our method, we now assess the capabilities of SVN on the computer vision benchmarks MNIST (LeCun, 1998) and FashionMNIST (Xiao et al., 2017). Since these are multi-class classification problems, we use the evaluation metrics from the previous section. We use the LeNet (LeCun et al., 1998) architecture for our experiments. Since this network is considerably larger, we also evaluate our proposed **LL-SVN** algorithm. Recent work suggested that for sufficiently large neural networks (Farquhar et al., 2021), the diagonal approximation works well. As such, our SVN experiments are conducted with the diagonal approximation to the Hessian, while our LL-SVN uses the KFAC approximation to the last layer. Our results are detailed in Table 2 and further experiment details can be found in Appendix C.3. We see that our methods are competitive with the baselines, and LL-SVN outperforms the baselines on FashionMNIST in terms of the NLL.

Table 2. Computer vision image datasets. Our LL-SVN algorithm is competitive in almost all metrics and outperforms the ensemble, WGD, and SVGD on the more complicated FashionMNIST dataset in terms of the NLL.

Dataset	Methods	ACC ↑	NLL ↓	ECE ↓	Brier ↓	AUROC ↑
MNIST	Ensemble	$\mathbf{0.991_{0.000}}$	$\mathbf{0.039_{0.001}}$	$\mathbf{0.011_{0.000}}$	$\mathbf{0.014_{0.000}}$	$\mathbf{1.000_{0.00}}$
	WGD	$\mathbf{0.991_{0.000}}$	$\mathbf{0.041_{0.003}}$	$\mathbf{0.012_{0.001}}$	$\mathbf{0.014_{0.001}}$	$\mathbf{1.000_{0.000}}$
	SVGD	$\mathbf{0.991_{0.000}}$	$\mathbf{0.038_{0.000}}$	$\mathbf{0.012_{0.001}}$	$\mathbf{0.014_{0.000}}$	$\mathbf{1.000_{0.000}}$
	SVN	$\mathbf{0.991_{0.001}}$	$0.041_{0.001}$	$0.015_{0.001}$	$0.016_{0.000}$	$\mathbf{1.000_{0.000}}$
	LL-SVN	$\mathbf{0.991_{0.000}}$	$\mathbf{0.038_{0.000}}$	$\mathbf{0.011_{0.000}}$	$\mathbf{0.014_{0.000}}$	$\mathbf{1.000_{0.000}}$
FashionMNIST	Ensemble	$\mathbf{0.916_{0.001}}$	$0.259_{0.005}$	$\mathbf{0.051_{0.000}}$	$\mathbf{0.122_{0.001}}$	$\mathbf{0.995_{0.000}}$
	WGD	$0.912_{0.004}$	$0.257_{0.009}$	$\mathbf{0.056_{0.004}}$	$\mathbf{0.128_{0.007}}$	$\mathbf{0.994_{0.001}}$
	SVGD	$\mathbf{0.916_{0.002}}$	$0.253_{0.006}$	$\mathbf{0.051_{0.001}}$	$\mathbf{0.123_{0.002}}$	$\mathbf{0.995_{0.000}}$
	SVN	$0.907_{0.005}$	$0.261_{0.009}$	$0.0561_{0.001}$	$0.135_{0.007}$	$\mathbf{0.994_{0.001}}$
	LL-SVN	$\mathbf{0.918_{0.001}}$	$\mathbf{0.241_{0.002}}$	$\mathbf{0.051_{0.001}}$	$\mathbf{0.120_{0.001}}$	$\mathbf{0.995_{0.000}}$

3.4. Convergence Speed

As discussed in Section 2, incorporating curvature information in the optimization of deep neural networks can yield significantly faster convergence. We investigate whether this intuition holds for SVN ensembles. We visualize the validation NLL on three UCI datasets in Figure 5 at the end of every epoch for the first 20 epochs. Interestingly, SVN's initial updates are worse than SVGD, resulting in significantly higher validation NLL scores. However, within a few epochs, SVN surpasses the ensemble and SVGD quickly. We hypothesize that SVN requires a few iterations to fully leverage the curvature information in the posterior landscape to position its particles for more efficient Newtonian updates compared to the 'simple' SVGD updates. More-

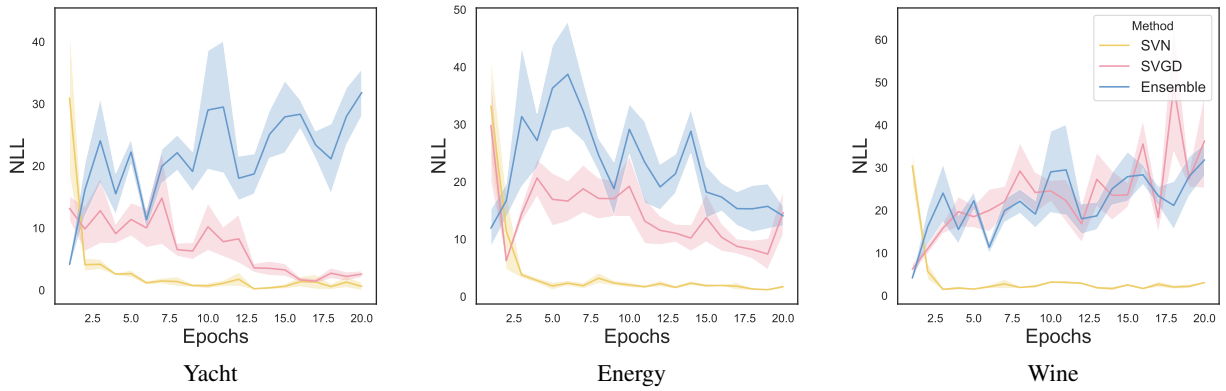


Figure 5. Comparison of validation negative log-likelihood computed at the end of every epoch for the first 20 epochs of training on Yacht, Energy, and Wine datasets. While SVN's initial performance is considerably worse than the other methods, it outperforms both within a few epochs.

over, we visualize some Hessians on a downscaled example in Appendix D.3, showing that initially the Hessian is not very informative. As such, the SVN updates are likely not effective initially, and we only benefit from second-order optimization once our particles are better positioned in the posterior.

4. Related Work

BNN inference. Approximate BNN inference methods fall on a spectrum from cheap Laplace approximations (Laplace, 1774; MacKay, 1992a; Daxberger et al., 2021a; Immer et al., 2021b;a) and variational inference (VI; Blei et al., 2017) to expensive Markov chain Monte Carlo (MCMC; Andrieu et al., 2003; Welling & Teh, 2011) techniques, with a notable tradeoff in computational efficiency and estimation accuracy. Arguably, approximations to the posterior can yield even better results than the actual posterior (Wenzel et al., 2020a). While there has been some recent interest in MCMC for deep neural networks (Izmailov et al., 2021; Huang et al., 2023; Garriga-Alonso & Fortuin, 2021), the computational requirements of these techniques remain a challenge (Papamarkou et al., 2021; Izmailov et al., 2021; Papamarkou et al., 2024). However, mean-field VI, especially for small neural nets, has also recently been criticized (Farquhar et al., 2021; Foong et al., 2020).

Particle-based inference. Situated between mean-field VI and MCMC, particle-based VI approaches offer a good tradeoff between efficiency and performance (Liu et al., 2019). Neural network ensembles have a rich history, beginning with the work of (Hansen & Salamon, 1990). Deep ensembles (Lakshminarayanan et al., 2017) and variations (Wenzel et al., 2020b; Kobayashi et al., 2022; Wild et al., 2023) are relatively simple scalable methods. Including repulsive forces in the particle updates even results in asymptotic convergence to the true posterior (D'Angelo & Fortuin, 2021b). Variants of SVGD (Liu & Wang, 2016) include projected SVGD (Chen & Ghattas, 2020), Riemannian SVGD (Liu & Zhu, 2017), message-passing SVGD (Zhuo et al., 2018), annealed SVGD (D'Angelo & Fortuin, 2021a), as well as their extensions to NN Ensembles (D'Angelo et al., 2021). Variations of the SVN (Detommaso et al., 2018) include stochastic SVN (Leviyev et al., 2022), which turns the method into an MCMC sampler by adding Gaussian noise, and projected SVN (Chen et al., 2020). However, SVN has so far only been used in low-dimensional problems and not for high-dimensional BNN posteriors, as in our work.

Second-order methods. Second-order methods in VI have been pivotal in improving optimization efficiency, and parametric variational inference leveraging second-order information has shown accelerated convergence (Khan et al., 2017a;b; 2018). Furthermore, curvature information can significantly improve learning dynamics in deep neural networks (Lin et al., 2023; 2024; Eschenhagen et al., 2024), which has spawned recent work on software packages for scalable Hessian computations (Daxberger et al., 2021a; Immer et al., 2021a). We make use of this development in our work to scale up SVN to BNN posteriors.

5. Conclusion

In this paper, we have introduced the SVN algorithm for BNN inference, which enhances traditional first-order methods by incorporating second-order curvature information through Hessian approximations. Our extensive evaluations across synthetic and real-world datasets, including UCI benchmarks and computer vision tasks, demonstrate that SVN consistently matches or outperforms

existing particle-based methods like deep ensembles, repulsive ensembles, and SVGD, in terms of predictive performance, uncertainty estimation, and convergence speed. Additionally, our novel Last-Layer SVN variant proves computationally efficient for larger neural networks, maintaining competitive performance. These findings position SVN as a promising advancement in particle-based variational inference, offering more accurate and reliable deep learning models.

Limitations. We have demonstrated the competitive or superior performance of our SVN algorithm on a range of benchmark tasks, focusing on moderately-sized neural networks. While the Hessian approximations discussed in this work can be scaled to more complex models like GNNs, ViTs (Eschenhagen et al., 2024), or LLMs (Yang et al., 2024; Daxberger et al., 2021a; Onal et al., 2024), it remains unclear how well our method will perform in these scenarios. However, given the recent success of second-order methods in large NN training (Lin et al., 2023), we remain optimistic. Additionally, while some Hessian approximations can be numerically unstable, we can mitigate this by selecting the appropriate approximation for each specific task.

Acknowledgments We want to thank Alexander Immer, Richard Paul, Emma Caldwell, Andrea Rubbi and Arsen Sheverdin for helpful discussions. Moreover, we want to thank the Jülich Supercumputing Centre's support team and in particular Rene Halver. VF was supported by a Branco Weiss Fellowship. This work was supported by Helmholtz AI computing resources (HAICORE) of the Helmholtz Association's Initiative and Networking Fund through Helmholtz AI. This work was also supported by the Helmholtz Association Initiative and Networking Fund on the HAICORE@KIT partition.

References

Amari, S. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10:251–276, 1998.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1-2):5–43, January 2003. ISSN 1573-0565. doi: 10.1023/A:1020281327116.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational Inference: A Review for Statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Botev, A., Ritter, H., and Barber, D. Practical Gauss-Newton Optimisation for Deep Learning. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 557–565. PMLR, 06–11 Aug 2017.

Brier, G. W. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1):1 – 3, 1950.

Chen, P. and Ghattas, O. Projected Stein Variational Gradient Descent. *arXiv preprint arXiv:2002.03469*, 2020.

Chen, P., Wu, K., Chen, J., O'Leary-Roseberry, T., and Ghattas, O. Projected Stein Variational Newton: A Fast and Scalable Bayesian Inference Method in High Dimensions, 2020.

Dangel, F., Kunstner, F., and Hennig, P. Backpack: Packing more into backprop. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

D'Angelo, F. and Fortuin, V. Annealed stein variational gradient descent. *arXiv preprint arXiv:2101.09815*, 2021a.

D'Angelo, F. and Fortuin, V. Repulsive Deep Ensembles are Bayesian. In *Neural Information Processing Systems*, 2021b.

D'Angelo, F., Fortuin, V., and Wenzel, F. On Stein Variational Neural Network Ensembles. *arXiv preprint arXiv:2106.10760*, 2021.

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace Redux - Effortless Bayesian Deep Learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20089–20103. Curran Associates, Inc., 2021a.

Daxberger, E., Nalisnick, E., Allingham, J. U., Antorán, J., and Hernández-Lobato, J. M. Bayesian Deep Learning via Subnetwork Inference. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 2510–2521. PMLR, 2021b.

Denker, J. and LeCun, Y. Transforming Neural-Net Output Levels to Probability Distributions. In Lippmann, R., Moody, J., and Touretzky, D. (eds.), *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann, 1990.

Detommaso, G., Cui, T., Marzouk, Y., Spantini, A., and Scheichl, R. A Stein variational Newton method. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, pp. 9169–9179. Curran Associates, Inc., 2018.

Dua, D. and Graff, C. UCI Machine Learning Repository, 2017.

Eschenhagen, R., Immer, A., Turner, R. E., Schneider, F., and Hennig, P. Kronecker-Factored Approximate Curvature for Modern Neural Network Architectures, 2024.

Farquhar, S., Smith, L., and Gal, Y. Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior approximations, 2021.

Foong, A. Y. K., Burt, D. R., Li, Y., and Turner, R. E. On the Expressiveness of Approximate Inference in Bayesian Neural Networks, 2020.

Gallego, V. and Insua, D. R. Stochastic Gradient MCMC with Repulsive Forces, 2020.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D., and Wilson, A. G. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. *arXiv preprint arXiv:1802.10026*, 2018.

Garriga-Alonso, A. and Fortuin, V. Exact langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*, 2021.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast Approximate Natural Gradient Descent in a Kronecker Factored Eigenbasis. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Givens, G. and Hoeting, J. *Computational Statistics*. Wiley Series in Computational Statistics. Wiley, 2012. ISBN 9780470533314.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

Gruber, S. G. and Buettner, F. Better Uncertainty Calibration via Proper Scores for Classification and Beyond, 2024.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On Calibration of Modern Neural Networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 06–11 Aug 2017.

Hansen, L. K. and Salamon, P. Neural Network Ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12 (10):993–1001, October 1990. ISSN 0162-8828. doi: 10.1109/34.58871.

Hernández-Lobato, J. M. and Adams, R. P. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks, 2015.

Heskes, T. On "Natural" Learning and Pruning in Multilayered Perceptrons. *Neural Computation*, 12(4):881–901, 2000.

Huang, Y., Chouzenoux, E., Elvira, V., and Pesquet, J.-C. Efficient Bayes Inference in Neural Networks through Adaptive Importance Sampling, 2023.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., and Khan, M. E. Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning. *arXiv preprint arXiv:2104.04975*, 2021a.

Immer, A., Korzepa, M., and Bauer, M. Improving predictions of Bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics*, pp. 703–711. PMLR, 2021b.

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. G. What Are Bayesian Neural Network Posteriors Really Like? In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4629–4640. PMLR, 18–24 Jul 2021.

Jain, A. *Fundamentals of Digital Image Processing*. Prentice-Hall information and system sciences series. Prentice Hall, 1989. ISBN 9780133361650.

Khan, M., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable Bayesian Deep Learning by Weight-Perturbation in Adam. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2611–2620. PMLR, 10–15 Jul 2018.

Khan, M. E., Lin, W., Tangkaratt, V., Liu, Z., and Nielsen, D. Variational Adaptive-Newton Method for Explorative Learning, 2017a.

Khan, M. E., Liu, Z., Tangkaratt, V., and Gal, Y. Vprop: Variational Inference using RMSprop, 2017b.

Kobayashi, S., Aceituno, P. V., and von Oswald, J. Disentangling the Predictive Variance of Deep Ensembles through the Neural Tangent Kernel, 2022.

Kristiadi, A., Hein, M., and Hennig, P. Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5436–5446. PMLR, 13–18 Jul 2020.

Kullback, S. and Leibler, R. A. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in neural information processing systems*, pp. 6402–6413, 2017.

Laplace, P.-S. *Mémoires de Mathématique et de Physique, Tome Sixieme*. Imprimerie Royale, 1774.

LeCun, Y. The MNIST Database of Handwritten Digits. 1998.

LeCun, Y., Denker, J., and Solla, S. Optimal Brain Damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Lee, J., Humt, M., Feng, J., and Triebel, R. Estimating Model Uncertainty of Neural Networks in Sparse Information Form. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5702–5713. PMLR, 13–18 Jul 2020.

Leviyev, A., Chen, J., Wang, Y., Ghattas, O., and Zimmerman, A. A stochastic Stein Variational Newton method, 2022.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the Loss Landscape of Neural Nets, 2018.

Lin, W., Dangel, F., Eschenhagen, R., Neklyudov, K., Kristiadi, A., Turner, R. E., and Makhzani, A. Structured Inverse-Free Natural Gradient: Memory-Efficient & Numerically-Stable KFAC for Large Neural Nets, 2023.

Lin, W., Dangel, F., Eschenhagen, R., Bae, J., Turner, R. E., and Makhzani, A. Can We Remove the Square-Root in Adaptive Gradient Methods? A Second-Order Perspective, 2024.

Liu, C. and Zhu, J. Riemannian Stein Variational Gradient Descent for Bayesian Inference, 2017.

Liu, C., Zhuo, J., Cheng, P., Zhang, R., and Zhu, J. Understanding and Accelerating Particle-Based Variational Inference. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4082–4092. PMLR, 09–15 Jun 2019.

Liu, Q. and Wang, D. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *Advances in neural information processing systems*, pp. 2378–2386, 2016.

MacKay, D. J. Bayesian Interpolation. *Neural Computation*, 4(3):415–447, 1992a.

MacKay, D. J. A Practical Bayesian Framework for Backpropagation Networks. *Neural computation*, 4(3):448–472, 1992b.

Martens, J. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Martens, J. and Grosse, R. Optimizing Neural Networks with Kronecker-Factored Approximate Curvature. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 2408–2417. PMLR, 2015.

Neal, R. M. Bayesian Learning for Neural Networks. 1995.

Nix, D. and Weigend, A. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pp. 55–60 vol.1, 1994. doi: 10.1109/ICNN.1994.374138.

Njieutcheu Tassi, C. R., Gawlikowski, J., Fitri, A. U., and Triebel, R. The impact of averaging logits over probabilities on ensembles of neural networks. In *2022 Workshop on Artificial Intelligence Safety, AISafety 2022*, CEUR Workshop Proceedings, 2022.

Ober, S. W. and Rasmussen, C. E. Benchmarking the Neural Linear Model for Regression, 2019.

Onal, E., Flöge, K., Caldwell, E., Sheverdin, A., and Fortuin, V. Gaussian Stochastic Weight Averaging for Bayesian Low-Rank Adaptation of Large Language Models, 2024.

Osawa, K., Ishikawa, S., Yokota, R., Li, S., and Hoefler, T. ASDL: A Unified Interface for Gradient Preconditioning in PyTorch, 2023.

Papamarkou, T., Hinkle, J., Young, M. T., and Womble, D. Challenges in Markov chain Monte Carlo for Bayesian neural networks, 2021.

Papamarkou, T., Skoularidou, M., Palla, K., Aitchison, L., Arbel, J., Dunson, D., Filippone, M., Fortuin, V., Hennig, P., Hubin, A., Immer, A., Karaletsos, T., Khan, M. E., Kristiadi, A., Li, Y., Mandt, S., Nemeth, C., Osborne, M. A., Rudner, T. G., Rügamer, D., Teh, Y. W., Welling, M., Wilson, A. G., and Zhang, R. Position Paper: Bayesian Deep Learning in the Age of Large-Scale AI. *arXiv preprint arXiv:2402.00809*, 2024.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019.

Pearson, K. Contributions to the Mathematical Theory of Evolution, II: Skew Variation in Homogeneous Material. *Philosophical Transactions of the Royal Society*, 186: 343–414, 1895. doi: 10.1098/rsta.1895.0010.

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

Pop, R. and Fulop, P. Deep Ensemble Bayesian Active Learning : Addressing the Mode Collapse issue in Monte Carlo dropout via Ensembles, 2018.

Ritter, H., Botev, A., and Barber, D. Online Structured Laplace Approximations For Overcoming Catastrophic Forgetting. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018a.

Ritter, H., Botev, A., and Barber, D. A Scalable Laplace Approximation for Neural Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018b.

Rogozhnikov, A. Einops: Clear and Reliable Tensor Manipulations with Einstein-like Notation. In *International Conference on Learning Representations*, 2022.

Schraudolph, N. Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent. *Neural computation*, 14: 1723–38, 08 2002.

Silverman, B. W. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Welling, M. and Teh, Y. W. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th international conference on machine learning*, pp. 681–688, 2011.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. How Good is the Bayes Posterior in Deep Neural Networks Really? *arXiv preprint arXiv:2002.02405*, 2020a.

Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. Hyperparameter Ensembles for Robustness and Uncertainty Quantification. In *Advances in Neural Information Processing Systems*, 2020b.

Wild, V. D., Ghalebikesabi, S., Sejdinovic, D., and Knoblauch, J. A Rigorous Link between Deep Ensembles and (Variational) Bayesian Methods. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Wright, S. and Nocedal, J. *Numerical Optimization*. Springer Science, 1999.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yang, A. X., Robeyns, M., Wang, X., and Aitchison, L. Bayesian Low-rank Adaptation for Large Language Models, 2024.

Zhuo, J., Liu, C., Shi, J., Zhu, J., Chen, N., and Zhang, B. Message Passing Stein Variational Gradient Descent. In *International Conference on Machine Learning*, pp. 6018–6027. PMLR, 2018.

A. Hessian Approximation Details

Diagonal The Diagonal approximation disregards off diagonal elements in the Fisher Information Matrix $\mathbf{F}_\phi$. As such we can rewrite Equation (7) as:

$$\mathbf{F}_\phi = \sum_{i=1}^{b} \mathbb{E}_{\tilde{y}\sim p(y|g_\phi(x_i))} \left[\nabla_\phi(\log p(\tilde{y}|g_\phi(x_i))) \odot \nabla_\phi(\log p(\tilde{y}|g_\phi(x_i))) \right], \tag{22}$$

where $\odot$ represents the element-wise or Hadamard product. While MacKay has suggested to refrain from this approximation due to its oversimplification (MacKay, 1992b) the computational benefits are immense. Inverting a diagonal Fisher $\mathbf{F}_\phi$, $\phi \in \mathbb{R}^d$, can be done in $\mathcal{O}(d)$, which is considerably lower than for the full hessian $\mathcal{O}(d^3)$. Moreover, recent works suggests that for sufficiently large neural networks the approximation can yield good results (Farquhar et al., 2021).

KFAC The KFAC factorization serves as a balance between two extremes: diagonal factorization, which can be overly restrictive, and the full Fisher matrix, which is computationally impractical. The main idea is to capture the correlation between weights within the same layer while assuming that weights from different layers are independent. This assumption is more refined than the diagonal factorization, where all weights are considered independent. As such are approximating the GGN $\mathbf{H}_\phi$ in the l-th layer with parameters ϕ_l

$$\mathbf{H}_{\phi_l} = \mathbf{Q}_{\phi_l} \otimes \mathbf{K}_{\phi_l}.$$

Here the factors are computed with the outer products of pre-activations and Jacobians with respect to the output of a layer (Martens & Grosse, 2015; Botev et al., 2017). Following (George et al., 2018) the kronecker factors are represented in their eigendecompositions in (Daxberger et al., 2021a).

B. SVN Algorithm details

We implemented all algorithms using the popular machine learning PyTorch (Paszke et al., 2019) framework. We also make extensive use of the EinOps (Rogozhnikov, 2022) library to handle some of the more involved tensor operations. The Hessian computations are done using the Laplace-Redux library (Daxberger et al., 2021a), which itself uses the Backpack (Dangel et al., 2020) and ASDL (Osawa et al., 2023) libraries in its backend. In order to solve the linear systems presented in Section 2.3 we utilise solvers implemented in the SciPy (Virtanen et al., 2020).

Block Diagonal Approximation In the block-diagonal approximation, we are just considering the diagonal entries of the SVN Hessian, which uses the following block-diagonal SVN Hessian:

$$H^{\text{SVN}} := \begin{bmatrix} h^{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & h^{NN} \end{bmatrix}$$

in Equation (17). This results in solving Equation (19) for each particle independently. This approach has significant computational advantages over the full SVN Hessian H^{SVN}.

Linear System Solver In order to solve the full linear system in Equation (17) or the block diagonal version Equation (19) we utilize Scipy's Conjugate Newton method. The conjugate newton method has strong theoretical grounding (Wright & Nocedal, 1999) for sufficient numbers of iterations and was used by Detommaso et al. (2018) in their original implementation as well. We use 50 iterations for all our computations. Leviyev et al. (2022) adjusted the Hessian with a Levenberg-like damping term and then employ a Cholesky Decomposition to solve the resulting linear system, however we will leave this for future work.

Matrix vector products for different Hessian Approximations In order to use the numerical techniques discussed above to solve the linear systems in Equation (17), Equation (19) one needs to define a Hessian vector product. For $\mathbf{H}_\phi \in \mathbb{R}^{d\times d}$ and

$x \in \mathbb{R}^d$. For the full Hessian approximation this is just the standard matrix-vector product. In the diagonal approximation we just consider the diagonal elements, which means that $\mathbf{H}_\phi^{\text{Diag}} \in \mathbb{R}^d$ is a vector. As such the matrix vector product collapses to a simple Hadamard product of these two vectors:

$$\mathbf{H}_\phi^{\text{Diag}} x = \mathbf{H}_\phi^{\text{Diag}} \odot x \tag{23}$$

The KFAC approximation involves a layer-wise block diagonal approximation. For the parameter vector ϕ_l representing the l-th layer, we aim to calculate $\mathbf{H}_{\phi_l}\phi_l$. The Hessian of the l-th layer is decomposed into its Kronecker factors: $\mathbf{H}_{\phi_l}^{\text{KFAC}} = \mathbf{Q}_{\phi_l} \otimes \mathbf{K}_{\phi_l}$, which are represented in their eigendecomposition as $\mathbf{Q}_{\phi_l} = V_Q \Lambda_Q V_Q^\top$ and $\mathbf{K}_{\phi_l} = V_K \Lambda_K V_K^\top$. For notational convenience, we omit the dependency on ϕ_l.

A straightforward approach to $\mathbf{H}_{\phi_l}^{\text{KFAC}} x$ would involve using the eigenvectors (V_Q, V_K) and eigenvalues (Λ_Q, Λ_K) to construct $\mathbf{Q}_{\phi_l}$ and $\mathbf{K}_{\phi_l}$, then explicitly calculate the Kronecker product $\mathbf{Q}_{\phi_l} \otimes \mathbf{K}_{\phi_l}$ and perform a standard matrix-vector product. However, this approach is highly inefficient in PyTorch. Instead, we utilize the "vec trick" (Jain, 1989) for Kronecker products. With $Q \in \mathbb{R}^a$ and $K \in \mathbb{R}^b$, we have $ab = |\phi_l| = |x|$ and write $X \in \mathbb{R}^{a \times b}$ as a reshaped version of x. The "vec trick" computes our desired product as follows:

$$\begin{aligned} v &= (Q \otimes K)x && (24)\\ &= (Q \otimes K^\top)x && (25)\\ &= \text{vec}((QXK)^\top) && (26)\\ &= \text{vec}(Q^\top)X^\top)K^\top)). && (27) \end{aligned}$$

This approach allows us to use the Kronecker factors exactly without explicitly constructing $\mathbf{H}_{\phi_l}^{\text{KFAC}}$ using the Kronecker product.

LL-SVN Algorithm Algorithm 2 details our novel last-layer SVN algorithm. Conceptually, it is very similar to the SVN Ensembles algorithm described in Section 2. The main difference is that it restricts the computation of the SVN update to only the parameters in the last layer, while using v_l^{SVGD} for the rest of the parameters. Since our SVN algorithm also needs to compute v_l^{SVGD} for the entire network, this part remains identical in both algorithms. The only difference is that we approximate the Hessian only for the parameters in the last layer, ϕ^{LL}, using the KFAC approximation. We then solve the full linear system Equation (17) or block diagonal linear systems Equation (19) only with the ϕ^{LL} parameters.

Algorithm 2 One iteration of our LL-SVN neural network ensemble algorithm

Input: Particles $\{\phi_i^l\}_{i=1}^n$ at stage l; step size ε, Kernel $k(\cdot,\cdot) \in \{k^{M_{\text{SVN}}}, k^I\}$, Hessian Approximation $\mathbf{H} \in \{\mathbf{H}_{\text{Full}}, \mathbf{H}_{\text{KFAC}}, \mathbf{H}_{\text{Diag}}\}$, number of parameters in last layer d_{ll}
Output: Particles $\{\phi_i^{l+1}\}_{i=1}^n$ at stage $l+1$
Compute v_l^{SVGD}
for $i = 1, 2, \ldots, n$ **do**
  Obtain last-layer weights: $\phi_i = \phi_i^{FL} \cup \phi_i^{LL}$
  if Block Diagonal Approximation **then**
    Solve linear system Equation (19) for $\alpha^1, \ldots, \alpha^n$ with ϕ_i^{LL}
  else
    Solve linear system Equation (17) for $\alpha^1, \ldots, \alpha^n$ with ϕ_i^{LL}
  end if
  Concatenate: $v_l^{\text{LL-SVN}}\left(\phi_i^k\right) = v^{\text{SVGD}}(\phi_i^{FL}) \cup v_l^{\text{SVN}}(\phi_i^{LL})$
  Set $\phi_i^{k+1} \leftarrow \phi_i^k + \varepsilon\, v_l^{\text{LL-SVN}}\left(\phi_i^k\right)$ given $\alpha^1, \ldots, \alpha^n$
end for

C. Experiment Details

C.1. Toy Regression Details

The dataset for this study is a 1D regression dataset generated using a specific mapping function and domain. We use `task_set=4` of the Toy Regression presented in D'Angelo & Fortuin (2021b). The dataset characteristics are as follows:

The mapping function used to generate the dataset is given by:

$$f(x) = (x - 3)^3 \tag{28}$$

The dataset is created within the following domains:

- **Training Domain:** $[2, 3] \cup [4.5, 6]$
- **Test Domain:** $[0, 7]$

To simulate real-world data variations, Gaussian noise with a standard deviation of 0.25 is added to the function values. The dataset includes:

- **Number of Training Samples:** 150
- **Number of Test Samples:** 200

Additionally, specific clusters of data points, referred to as blobs, are included within the training domain at the intervals $[1.5, 2.5]$ and $[4.5, 6.0]$. These blobs introduce additional complexity to the dataset, providing a more challenging environment for evaluating regression models.

The dataset generation process is controlled by a fixed random seed of 42 to ensure reproducibility. The training, testing, and validation sets are drawn from their respective domains, incorporating noise and blobs to enhance the dataset's realism and complexity.

We utilized the M2 Apple Chip CPU on our local machine to run these experiments, which last about 30 minutes for SVN with a full SVN-Hessian and using the anisotropic Gaussian Curvature Kernel.

C.2. UCI Experiments

For all the UCI experiments, we use the following hyperparameters originally based on (Ober & Rasmussen, 2019) for all methods:

- number of particles: $N = 5$
- learning rate: lr $\in [1e^{-4}, 3e^{-4}, 5e^{-4}, 1e^{-3}, 3e^{-3}, 5e^{-3}, 1e^{-2}, 3e^{-2}, 5e^{-2}]$
- Batch size: $B \in [8, 16, 32, 64]$
- Hidden layer sizes: $[50, 50]$
- Number of Epochs: 50

For all experiments we perform $K = 5$ fold cross validation and then split off an additional 20% off the training set as validation data. This yields the following splits:

To normalize the training and test datasets using the `StandardScaler` from (Pedregosa et al., 2011), we start by fitting the scaler to the training data, which calculates and stores the mean and standard deviation for each feature. After the scaler is fitted and the training data is transformed, we use the same scaler to transform the test data. The transformation of the test data uses the mean and standard deviation calculated from the training data, ensuring that both datasets are standardized based on the same criteria. This consistency is crucial for the performance of many machine learning algorithms, which

Split	Percentage (%)
Training Set	64
Validation Set	16
Test Set	20

Table 3. Data Split Percentages

require features to be on a similar scale to function properly. After this we then split the validation data from the training data.

For the SVN algorithm we used the full Hessian approximation, with the full SVN Hessian matrix H^{SVN} and the anistropic Gaussian Kernel detailed in Equation (21) on all datasets except for protein. On the protein dataset we used the KFAC Hessian approximation without the anisotropic Gaussian Kernel.

We have included the full numerical results for the UCI regression tasks in the test MSE Table 5 and test NLL Table 4. We train all the models for 50 epochs and then load the model with the lowest validation loss for final testing. For the binary classification experiments detailed in Table 1 we load the model from the epoch with the lowest negative log-likelihood on the validation dataset.

The **Gaussian Negative Log-Likelihood** (NLL) Loss (Nix & Weigend, 1994) is used in regression tasks to evaluate the performance of a neural network that predicts both the mean and variance of a target variable, assuming the target follows a Gaussian distribution. The loss function is designed to handle both heteroscedastic and homoscedastic cases. For a target y with predicted mean μ and predicted variance σ^2, the Gaussian NLL loss is given by:

$$\text{loss} = \frac{1}{2}\left(\log(\max(\sigma^2, \epsilon)) + \frac{(y-\mu)^2}{\max(\sigma^2, \epsilon)}\right) + \text{const.} \tag{29}$$

Here, ϵ is a small positive value (default 10^{-6}) used to ensure numerical stability. The constant term is omitted in our experiments. While Gaussian NLL loss is primarily used for regression, it can also be adapted for certain classification contexts, such as probabilistic regression classification or estimating continuous class scores with uncertainty, where the network predicts mean and variance for the continuous outputs.

For the **Negative Log-Likelihood** in all our classification experiments, so the UCI binary classification as well as the computer vision experiments we follow (Njieutcheu Tassi et al., 2022) and average the predicted logits over our ensemble. We then use PyTorch (Paszke et al., 2019) to compute the corresponding cross-entropy/NLL based on the unnormalized averaged logits and targets.

We utilized a supercomputing environment for our experiments, featuring NVIDIA A100 GPUs in its compute nodes. Depending on the dataset size, the SVN algorithm takes around 2 hours for smaller datasets and up to 18 hours to perform the entire cross-validation for $K = 5$ splits. These times can vary between sweeps, as they depend on the learning rate and batch size.

C.3. Computer Vision Experiments

For our computer vision experiments on the MNIST (LeCun, 1998) and FashionMNIST datasets (Xiao et al., 2017), we performed cross-validation with $K = 5$. We then split the validation data as described in Appendix C.2 to obtain the final splits, as detailed in Table 3.

- number of particles: $N = 5$
- learning rate: lr $\in [1e^{-3}, 3e^{-3}, 5e^{-3}, 1e^{-2}, 3e^{-2}]$
- learning rate schedule: constant
- Batch size: $B \in [64, 128, 256]$
- Optimizer: Adam

Table 4. Test negative log-likelihood on UCI regression datasets. Our proposed SVN method generally outperforms the baselines.

	Test Negative Log-Likelihood ↓			
Dataset	Ensembles	WGD	SVGD	SVN
autompg (n = 392)	$17.53_{2.85}$	$24.23_{2.73}$	$14.15_{1.93}$	$\mathbf{3.94_{0.99}}$
concrete (n = 1030)	$7.01_{0.82}$	$9.13_{0.21}$	$7.66_{1.19}$	$\mathbf{4.77_{0.47}}$
energy (n = 768)	$3.44_{0.69}$	$6.45_{0.83}$	$2.90_{0.61}$	$\mathbf{1.48_{0.11}}$
kin8nm (n = 8192)	$1.07_{0.17}$	$1.35_{0.43}$	$4.14_{0.68}$	$\mathbf{-1.27_{0.16}}$
naval (n = 11934)	$\mathbf{-5.17_{0.09}}$	$-4.93_{0.12}$	$\mathbf{-5.37_{0.07}}$	$\mathbf{-5.38_{0.20}}$
power (n = 9568)	$17.42_{0.49}$	$66.57_{13.63}$	$14.56_{1.27}$	$\mathbf{3.36_{0.24}}$
protein (n = 45730)	$18.69_{1.06}$	$22.77_{1.28}$	$20.83_{0.76}$	$\mathbf{3.24_{0.21}}$
wine (n = 1599)	$12.24_{1.37}$	$13.69_{2.39}$	$14.29_{1.16}$	$\mathbf{0.39_{0.09}}$
yacht (n = 308)	$1.51_{0.61}$	$0.66_{0.18}$	$1.10_{0.64}$	$\mathbf{-0.36_{0.19}}$

Table 5. Test MSE on UCI Regression Tasks

	Test MSE ↓			
Dataset	Ensembles	WGD	SVGD	SVN
autompg (n = 392)	$2.53_{0.04}$	$\mathbf{0.77_{0.28}}$	$3.22_{0.15}$	$1.61_{0.28}$
concrete (n = 1030)	$\mathbf{2.68_{0.66}}$	$\mathbf{2.61_{0.31}}$	$\mathbf{2.42_{0.81}}$	$\mathbf{2.45_{0.72}}$
energy (n = 768)	$0.19_{0.04}$	$0.19_{0.05}$	$0.14_{0.02}$	$\mathbf{0.07_{0.01}}$
kin8nm (n = 8192)	$\mathbf{7.4e^{-6} \pm 2.36e^{-6}}$	$4.5e^{-5} \pm 1.3e^{-5}$	$1.9e^{-5} \pm 0.3e^{-5}$	$7.06e^{-5} \pm 1.42e^{-5}$
naval (n = 11934)	$\mathbf{6.2e^{-8} \pm 1.4e^{-8}}$	$5.4e^{-7} \pm 1.2e^{-7}$	$2.8e^{-7} \pm 0.7e^{-7}$	$1.2e^{-7} \pm 0.3e^{-7}$
power (n = 9568)	$0.16_{0.02}$	$0.49_{0.09}$	$0.77_{0.06}$	$\mathbf{0.08_{0.02}}$
protein (n = 45730)	$\mathbf{0.11_{0.11}}$	$\mathbf{0.06_{0.01}}$	$\mathbf{0.11_{0.11}}$	$0.36_{0.10}$
wine (n = 1599)	$0.05_{0.01}$	$\mathbf{0.02_{0.01}}$	$\mathbf{0.04_{0.01}}$	$\mathbf{0.02_{0.01}}$
yacht (n = 308)	$\mathbf{0.51_{0.20}}$	$\mathbf{0.26_{0.08}}$	$0.63_{0.10}$	$\mathbf{0.61_{0.47}}$

- Number of Epochs SVN: 20
- Number of Epochs LL-SVN: 100
- Number of Epochs SVGD: 100
- Number of Epochs Ensemble: 100

For all our experiments, we used the LenNet architecture (LeCun et al., 1998). We also experimented with weight decay and gradient clipping but found no improvement. For our SVN algorithm, we used the Diagonal Hessian Approximation as well as the Block Diagonal Approximation in conjunction with the anisotropic curvature kernel to ensure the method scales well to larger models. We limited the number of epochs to evaluate its ability to converge faster than the other methods and to decrease the computational requirements of the experiments. For LL-SVN, we used the KFAC Hessian approximation for the parameters of the last layer.

e utilized a supercomputing environment for our experiments, with NVIDIA A100 GPUs in its compute nodes. The LL-SVN algorithm takes about $2-3$ hours to finish training. SVN requires around 16 hours for the MNIST dataset and around 25 hours for running 20 epochs on the MNIST and FashionMNIST datasets, respectively. These numbers can vary depending on the batch size used and the learning rate.

D. Further Experiments

D.1. SVN in low-data regime

To further investigate the performance of SVN ensembles in low-data scenarios, we conducted an additional evaluation using subsamples of the MNIST dataset (LeCun, 1998). By employing a random seed, we created subsampled datasets with

sizes $n \in \{500, 2000, 5000, 10000\}$. The results of this evaluation are depicted in Figure 6, where we present the accuracy, negative log-likelihood, and expected calibration error for three methods: Ensemble, Stein Variational Gradient Descent (SVGD), and SVN.

Our analysis revealed that while SVGD consistently achieved higher accuracy across all subsample sizes, SVN demonstrated superior calibration performance, particularly in the low-data regime. Despite these differences, the overall performance of the three methods remained comparable under the conditions tested. This suggests that both SVN offers a competitive alternative to the Ensemble and SVGD methods in scenarios with limited data availability.

D.2. Ablation Studies:

An overview of SVN's main hyperparameters are:

- Choice of Hessian Approximation
- Anistropic Gaussian Curvature Kernel
- Use of the Block Diagonal Approximation
- Number of particles N

Block Diagonal Approximation and Curvature Kernel Figure 7 demonstrates several key findings regarding the performance and stability of different kernel choices in the SVN algorithm for the Yacht dataset. Firstly, using the anisotropic Gaussian kernel or the curvature kernel appears to improve performance in terms of mean squared error (MSE) and stabilize training, as indicated by smaller standard errors. Secondly, with regard to the negative log-likelihood (NLL) metric, the curvature kernel interestingly makes a more significant impact on improving NLL when using the full SVN Hessian. In contrast, when the block-diagonal approximation is employed, the anisotropic Gaussian kernel slightly improves performance.

Number of Particles Since an important hyperparameter in particle-based variational inference (VI) is the number of particles N, we investigated the behavior of the methods with different numbers of particles. As depicted in Figure 8, the results for the Yacht dataset provide significant insights into the impact of particle quantity on performance.

It can be observed that all methods generally improve with an increasing number of particles, as one might expect. However, while the test NLL of Ensembles and SVGD appears to plateau with larger numbers of particles, SVN demonstrates a more consistent improvement trend. This suggests that SVN's performance continues to benefit from additional particles without encountering the diminishing returns seen in the other methods.

We hypothesize that the curvature information incorporated in SVN updates allows it to more effectively allocate particles within the posterior landscape. This capability likely enables SVN to scale better with a larger number of particles, maintaining its performance improvement. In contrast, Ensembles may suffer from mode collapse, where particles fail to adequately explore the posterior distribution beyond a certain point. SVGD, while more robust than Ensembles, also appears to reach a saturation point where additional particles do not translate into significant performance gains.

These findings underscore the potential advantages of SVN in scenarios requiring the deployment of numerous particles, highlighting its robustness and scalability. Further research could investigate the underlying mechanisms in more detail and explore ways to mitigate the mode collapse observed in Ensemble methods (Pop & Fulop, 2018).

D.3. Hessian Plots during SVN training

In order to gain deeper insights into the behavior of our method, particularly in light of the observation in Section 3 that SVN exhibits considerably faster convergence of the validation NLL scores during training compared to SVGD or Ensembles. However, during the first one to two epochs its performance is considerably worse. As such we conducted additional experiments on a scaled-down architecture. The aim is to visualize the Hessian matrix throughout the training process. For this purpose, we trained the model on the Yacht dataset from the UCI regression benchmark datasets. The following hyperparameters are used for the experiment: number of particles: $N = 5$, learning rate: $\text{lr} = 1e^{-2}$, batch size: $B = 16$, single hidden fully connected layer of size: 10. This results in a total of 81 trainable parameters of the model.

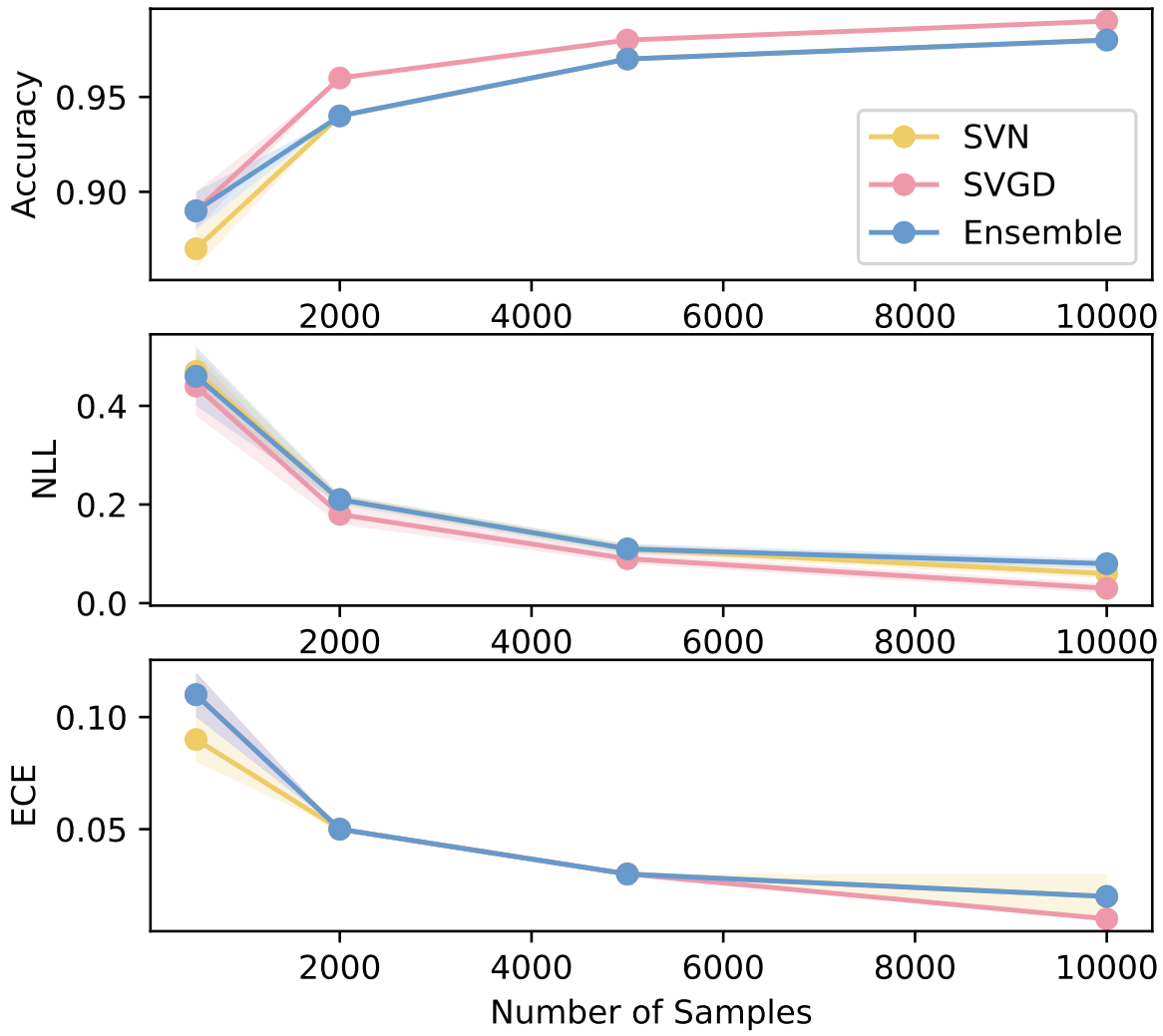


Figure 6. Performance metrics for Ensemble, SVGD, and SVN on subsampled MNIST data with varying sizes ($n \in \{500, 2000, 5000, 10000\}$). The plots display accuracy, negative log-likelihood, and expected calibration error.

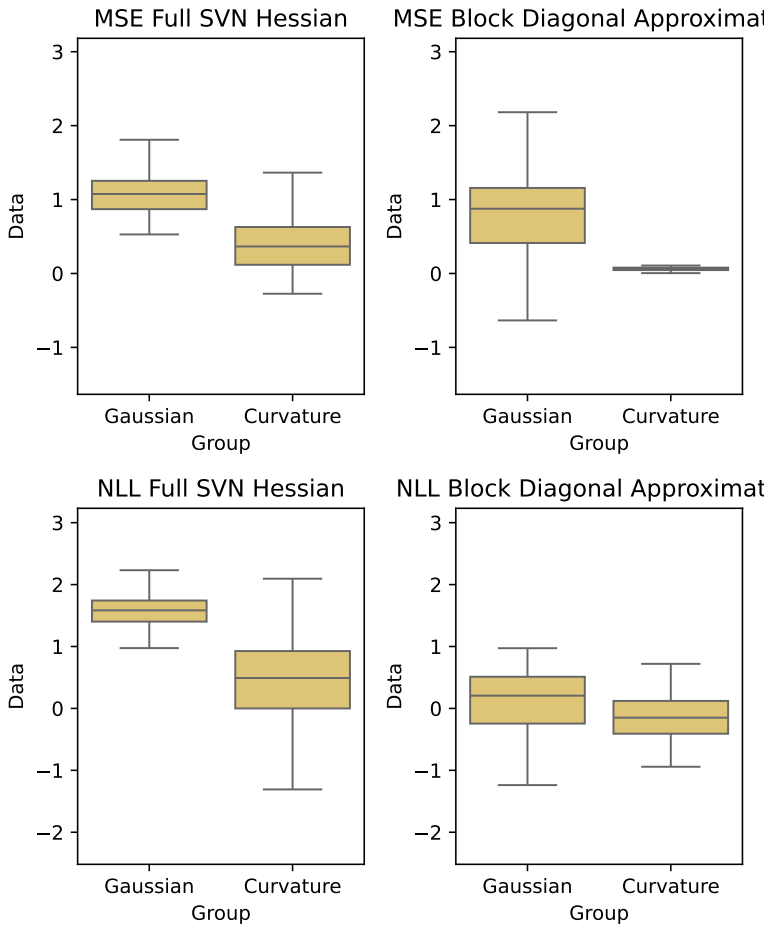


Figure 7. Ablation study on the use of the Block Diagonal Approximation and Anistropic Curvature Kernel. The first and third plot correpsond to the use of the Full SVN in computing the SVN update, while the second and fourth plot show the Block Diagonal approximation.

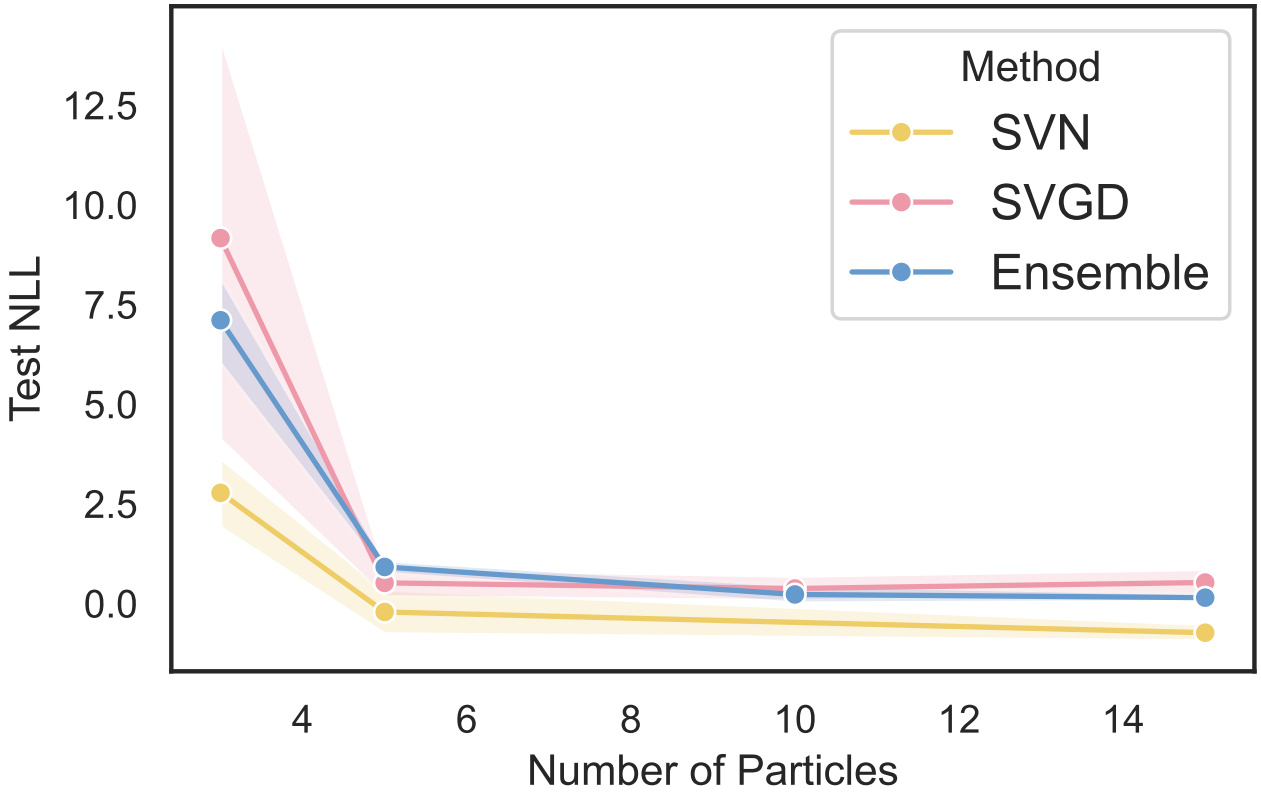


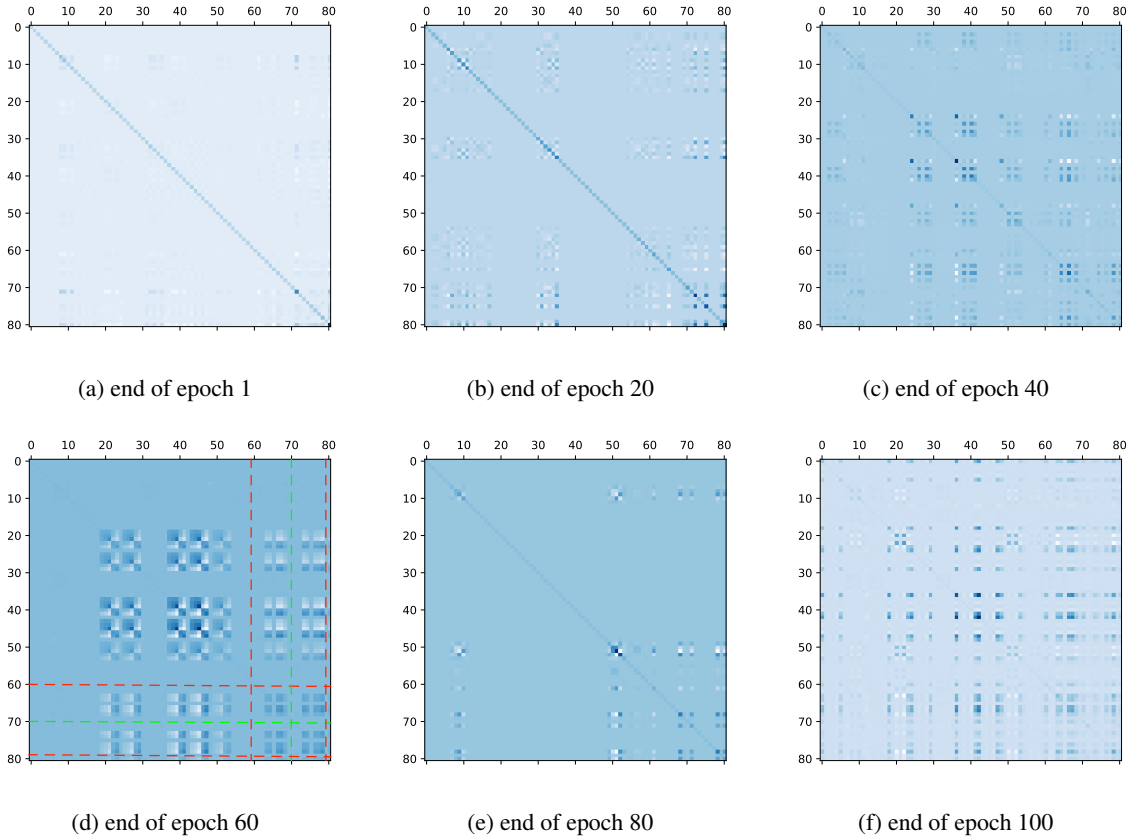
Figure 8. Test NLL for methods with varying numbers of particles for the Yacht dataset. While SVN exhibits superior performance for all considered numbers of particles, it is the only method with a consistent downwards trend, in contrast to SVGD and Ensembles, which appear to flatten for larger numbers of particles.

We utilized full Hessian approximations for the gradient update using the Laplace library (Daxberger et al., 2021a). The results were evaluated on a 20% holdout of unseen data. The training was conducted over 100 epochs, and snapshots of the Hessian were taken at six different points during the training process. The experiment converged to a test MSE score of 0.164 and a test NLL score of 0.382.

Figures Figures 9 to 13 visualize the Hessian matrices throughout the training process for the five different particles. Figure Figure 14 shows the state of the curvature matrix at the same timestamps. These visualizations were crucial for understanding the dynamic behavior of the Hessian and its impact on the training process.

The primary motivation for this analysis was to investigate why our method, despite converging faster in the initial gradient steps, exhibited the worst performance among all methods during the first two epochs. Empirically, we observed that our method needed some time to stabilize before achieving its rapid convergence speed. By plotting the Hessians for this downscaled example, we aimed to obtain more detailed insights and a better understanding of our method's behavior. These visualizations help in diagnosing potential issues such as instability in the Hessian matrix or improper allocation of particles, which could contribute to the observed performance drop.

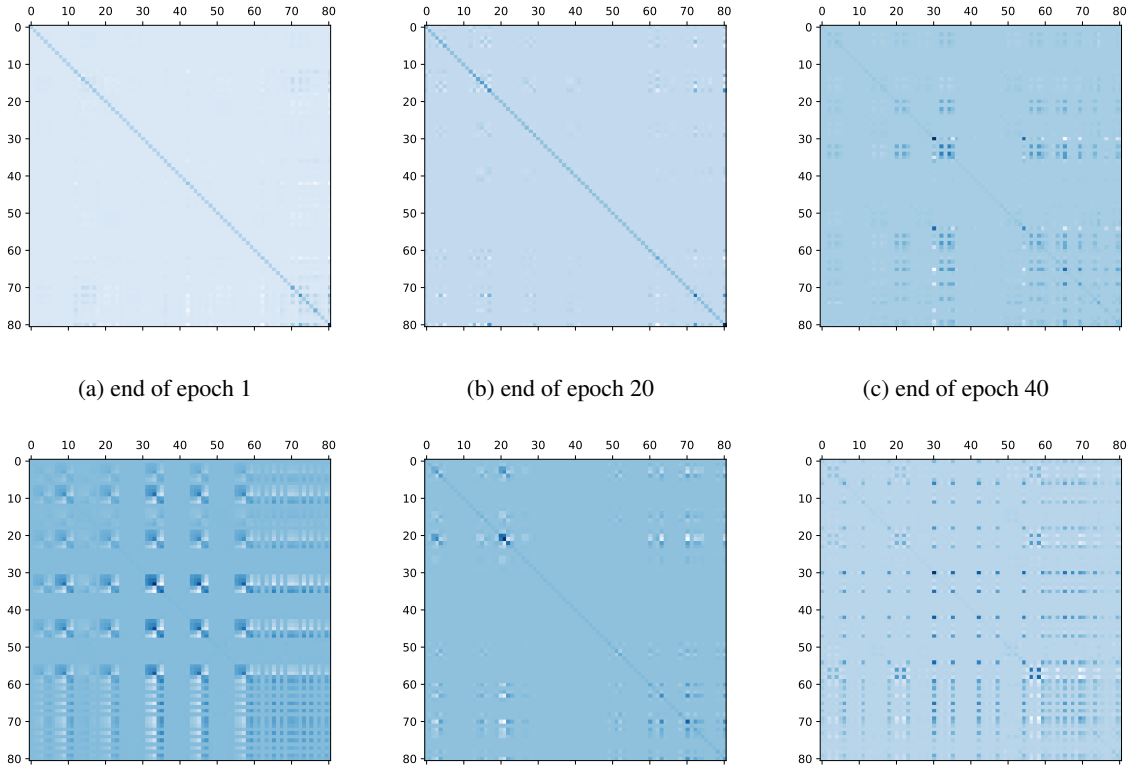
Figures Figures 9 to 13 illustrate the Hessian matrices for different particles throughout the training process, while Figure 14 depicts the average curvature matrix used in the anisotropic Gaussian kernel. In all these plots, it can be observed that the Hessian matrix is not particularly expressive at the beginning of the training process. Consequently, we do not expect our SVN method to benefit significantly from more informed gradient steps through curvature information initially. This lack of initial expressiveness means that our method requires a few gradient steps to start effectively utilizing the curvature information and improving its performance. Thus, the SVN method needs some time to settle in before achieving its characteristic convergence speed. While this is a scaled-down example, we are confident that the same principles hold for larger neural networks as well, as seen in Section 3.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

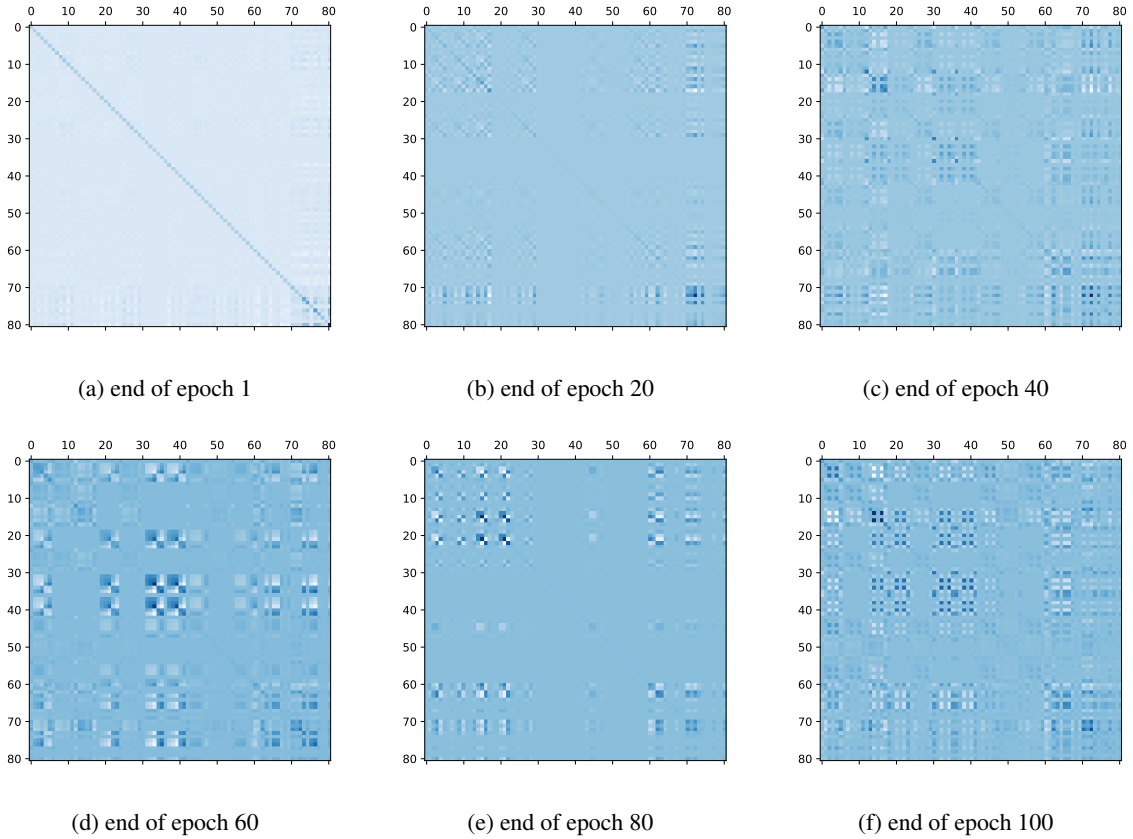
Figure 9. Visualization of snapshots of Hessian approximations for **Particle 1** at different times during the training process. Furthermore, the drawn lines in (d) illustrate the separation of weights and biases of different layers. The top left area for axes [0,59] represent the weight connections of input layer with the hidden layer, the next range [60,69] represent bias terms of the first layer, the next range [70,79] represent weight connections from hidden layer to output layer and the axes at 80 represent the bias term for output neuron.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

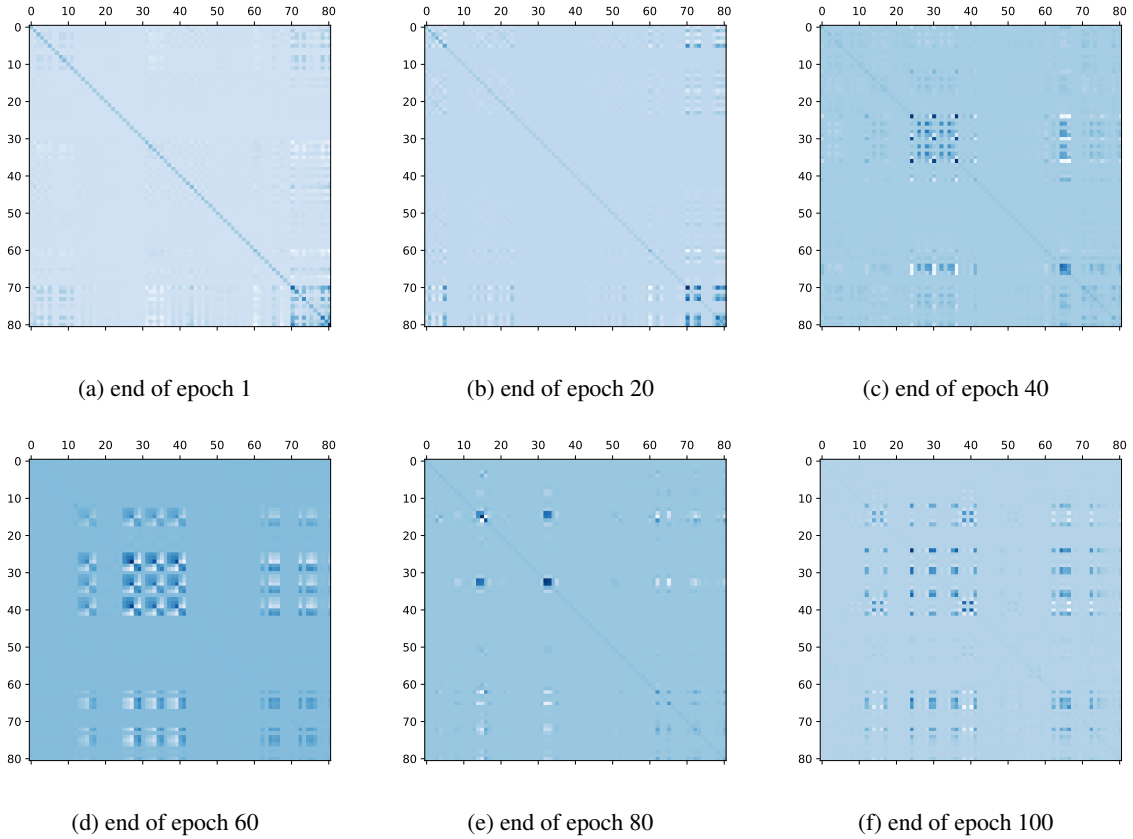
Figure 10. Visualization of snapshots of Hessian approximations for **Particle 2** at different times during the training process.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

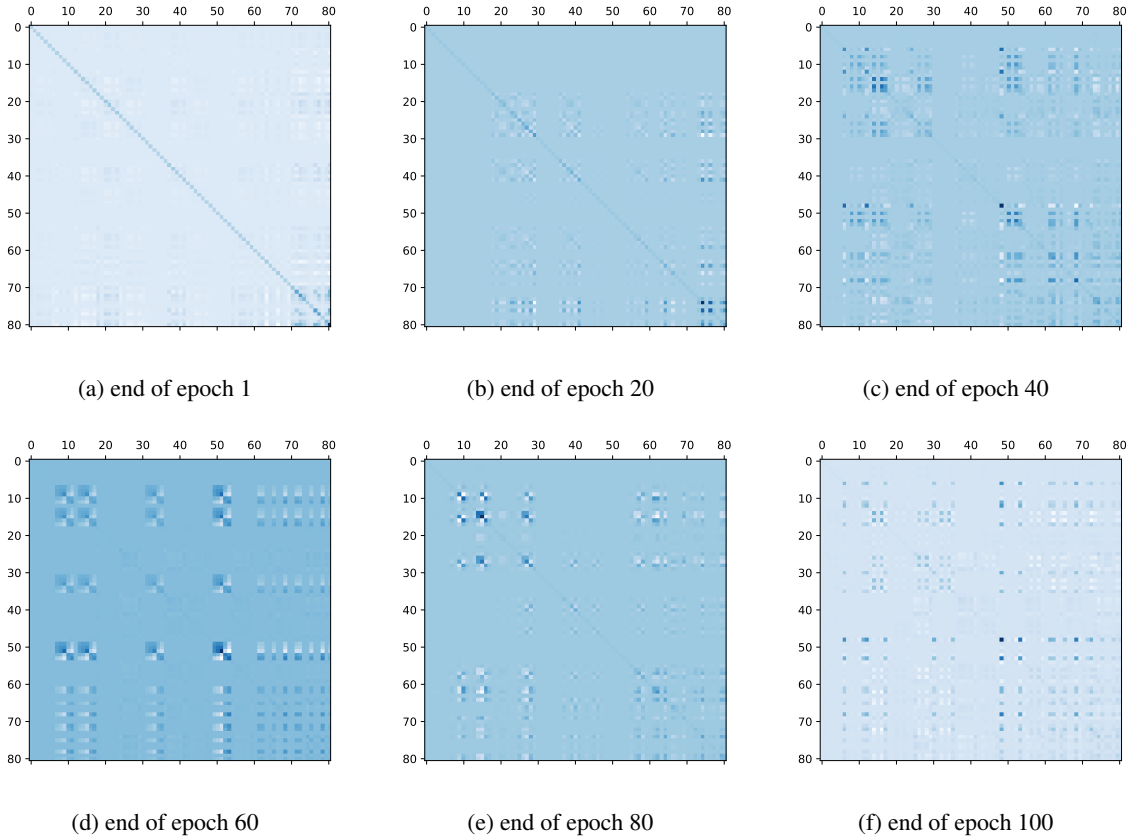
Figure 11. Visualization of snapshots of Hessian approximations for **Particle 3** at different times during the training process.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

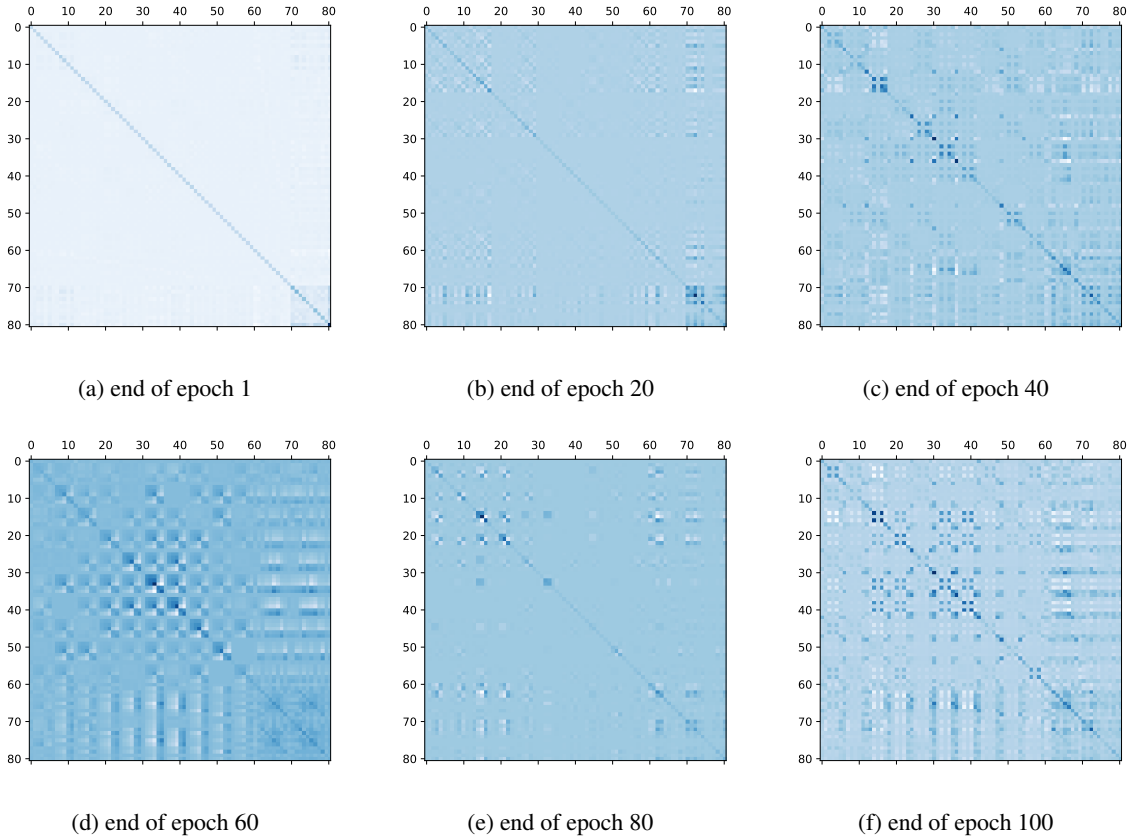
Figure 12. Visualization of snapshots of Hessian approximations for **Particle 4** at different times during the training process.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

Figure 13. Visualization of snapshots of Hessian approximations for **Particle 5** at different times during the training process.



(a) end of epoch 1 (b) end of epoch 20 (c) end of epoch 40

(d) end of epoch 60 (e) end of epoch 80 (f) end of epoch 100

Figure 14. Visualization of snapshots for the **mean curvature matrix** M_{SVN}, which is the average of the Hessian matrices of all 5 particles at different times during the training process.