
Uncertainty Quantification with the Empirical Neural Tangent Kernel

Joseph Wilson

School of Mathematics And Physics
University of Queensland
joseph.wilson1@uqconnect.edu.au

Chris van der Heide

Dept. of Electrical and Electronic Engineering
University of Melbourne
chris.vdh@gmail.com

Liam Hodgkinson

School of Mathematics and Statistics
University of Melbourne
lhodgkinson@unimelb.edu.au

Fred Roosta

CIRES and School of Mathematics And Physics
University of Queensland
fred.roosta@uq.edu.au

Abstract

While neural networks have demonstrated impressive performance across various tasks, accurately quantifying uncertainty in their predictions is essential to ensure their trustworthiness and enable widespread adoption in critical systems. Several Bayesian uncertainty quantification (UQ) methods exist that are either cheap or reliable, but not both. We propose a post-hoc, sampling-based UQ method for overparameterized networks at the end of training. Our approach constructs efficient and meaningful deep ensembles by employing a (stochastic) gradient-descent sampling process on appropriately linearized networks. We demonstrate that our method effectively approximates the posterior of a Gaussian Process using the empirical Neural Tangent Kernel. Through a series of numerical experiments, we show that our method not only outperforms competing approaches in computational efficiency—often reducing costs by multiple factors—but also maintains state-of-the-art performance across a variety of UQ metrics for both regression and classification tasks.

1 Introduction

Neural networks (NN) achieve impressive performance on a wide array of tasks, in areas such as speech recognition (Nassif et al., 2019; Abdel-Hamid et al., 2014; Deng et al., 2013), image classification (LeCun et al., 1998; Krizhevsky et al., 2012; He et al., 2016), computer vision (Redmon et al., 2016; Redmon & Farhadi, 2017), and language processing (Vaswani et al., 2017; Ray, 2023; Devlin et al., 2019), often significantly exceeding human performance. While the promising predictive and generative performance of modern NNs is evident, accurately quantifying uncertainty in their predictions remains an important and active research frontier (Abdar et al., 2021). Models are often over-confident in predictions on out-of-distribution (OoD) inputs (Guo et al., 2017), and sensitive to distribution-shift (Ford et al., 2019). By quantifying a model’s uncertainty, we can determine when it fails to provide well-calibrated predictions, indicating the need for additional training (possibly on more diverse data) or even human intervention. This is vital for deploying NNs in critical applications like diagnostic medicine and autonomous machine control (Nemani et al., 2023b).

An array of uncertainty quantification methods for NNs exist, each with benefits and drawbacks. Frequentist statistical methods, such as conformal prediction (Vovk et al., 2005; Papadopoulos et al., 2002; Lei & Wasserman, 2014), create prediction intervals through parameter estimation and probability distributions on the data. While currently considered state-of-the-art, the main drawback

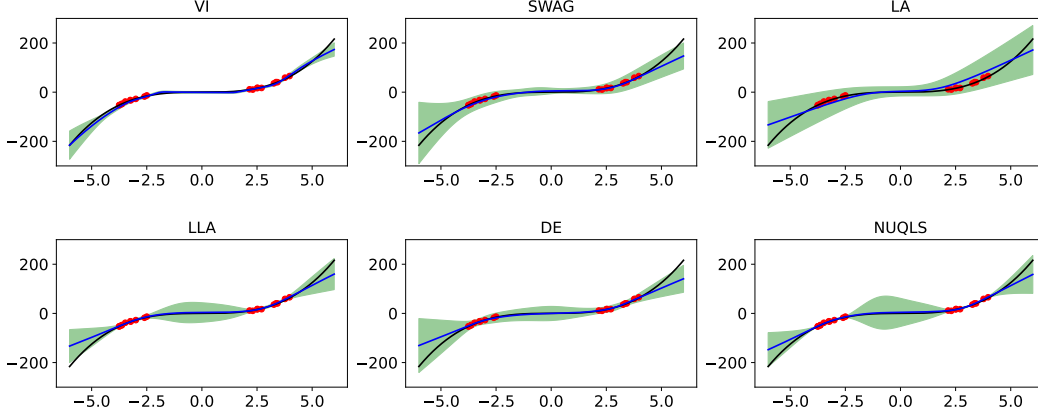


Figure 1: Comparison of various Bayesian UQ methods (see Section 2) on a 1-layer MLP, trained on the data (red) lying on $y = x^3$ (black), with Gaussian noise added. The methods’ mean predictors (blue) $\pm 3\sigma$ (green) are shown, where σ^2 is the variance estimated via each method. We see that NUQLS performs well on this task.

is that conformal prediction is data-hungry, requiring a large hold-out set. Several works Altieri et al. (2024); Dadalto et al. (2023); Granese et al. (2021) employ the feature-space representations of the network to quantify the uncertainty of test predictions, through kernel densities (Kotelevskii et al., 2022), Gaussian Discriminant analysis (Mukhoti et al., 2023), non-constant mapping functions (Tagasovska & Lopez-Paz, 2019), etc. The interpretation of uncertainty in these works is related to risk of misclassification; these methods generally perform very well at detecting OoD points. While both conformal predictions and feature-space works are important, we limit our scope to Bayesian methods in this paper. Though Bayesian methods can suffer from prior misspecification (Masegosa, 2020) and computational burdens, the predictive and posterior distributions of a model are very natural frameworks for quantifying the uncertainty and spread of possible values of the model.

Unfortunately, largely due to the curse of dimensionality, existing Bayesian methods are very expensive to compute, and provide poor approximations to the predictive distribution (Folgc et al., 2021). This results in a suite of methods that require excessive approximations to scale to large problems (Daxberger et al., 2021), often at the cost of theoretical underpinnings, or necessitating modifications to the network itself (He et al., 2020).

Gaussian Processes (GPs) (Rasmussen & Williams, 2005) are important tools in Bayesian machine learning (ML) that can directly capture the epistemic (model) uncertainty of predictions, and arise as large-width limits of NNs (Neal, 1996). However, naïve GP training scales cubically in the number of training datapoints, necessitating approximations for modern applications. Neural Tangent Kernels (NTKs) (Jacot et al., 2018) describe the functional evolution of a NN under gradient flow, and naturally arise in the analysis of model quality (Hodgkinson et al., 2023). Due to their deep connection to NNs, these covariance functions are enticing as potential tools for UQ of their corresponding NNs.

Motivated by this, we present a UQ method for a trained, over-parameterized NN model, wherein we approximate the predictive distribution through an ensemble of linearized models, trained using (stochastic) gradient-descent. For certain loss functions, this ensemble samples from the posterior of a GP with an empirical NTK.

Contributions. Our contributions are as follows:

1. In Section 3.1, we present a Monte-Carlo sampling based UQ method to approximate the predictive distribution of NNs, called Neural Uncertainty Quantification by Linearized Sampling (NUQLS). Our method is lightweight, post-hoc, numerically stable, and embarrassingly parallel.
2. Under certain assumptions, Section 3.2 details the convergence of NUQLS to the predictive distribution of a GP with an empirical NTK kernel, providing a novel perspective on the connection between NNs, GPs, and the NTK.
3. On various ML problems in Section 4, we show that NUQLS performs as well as or better than leading UQ methods, is less computationally expensive than deep ensemble, and scales to large image classification tasks.

4. In Section 4.4 we introduce a novel metric for evaluating the quality of UQ methods for classification tasks. This metric more directly measures the quality of UQ methods than existing UQ metrics.

Remark 1.1 (Necessity of Contribution 4.). Evaluating the quality of UQ methods is non-trivial. There exists in the literature a lack of a suitable framework for evaluating the quality of UQ methods in classification settings. Common metrics such as Negative Log-Likelihood (NLL), Expected Calibration Error (ECE) (Naeini et al., 2015) and Area Under the Curve of the Receiver Operating Characteristic (AUCROC) are actually metrics for prediction quality, or are based on flawed surrogates for uncertainty. Further, it was shown in Abe et al. (2022) that the goal of Bayesian methods should not be to improve predictive ability; performance gains will be attained more economically by choosing a larger model class. Instead, Bayesian methods should seek to accurately quantify the uncertainty of a model, by computing the predictive variance. As uncertainty is not a measurement that can easily be shown to be well-calibrated, one requires a more qualitative approach to evaluating the performance of a UQ model. This is the motivation for the graphical technique we introduce in Section 4.4 for comparing the quality of UQ estimates for multi-class classification. For an in-depth discussion of these points, please see Appendix C.

Notation. Throughout the paper, we denote scalars, vectors, and matrices as lower-case, bold lower-case, and bold upper-case letters, e.g., c , θ and $\mathbf{K}$, respectively. For two vectors $\mathbf{v} \in \mathbb{R}^p$ and $\mathbf{w} \in \mathbb{R}^p$, their Euclidean inner product is denoted as $\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w}$. We primarily consider supervised learning, which involves a function $\mathbf{f} : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}^c$, assumed sufficiently smooth in its parameters $\theta \in \mathbb{R}^p$, a training dataset $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}^c$, and a loss function $\ell : \mathbb{R}^c \times \mathbb{R}^c \rightarrow [0, \infty)$.

The process of training amounts to finding a solution, $\hat{\theta}$, to the optimization problem $\min_{\theta} \sum_{i=1}^n \ell(\mathbf{f}(\mathbf{x}_i, \theta), \mathbf{y}_i) + \mathcal{R}(\theta)$, where $\mathcal{R}(\theta)$ is a regulariser. For a kernel function $\mathbf{K} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^{c \times c}$, we define $\mathbf{K}_{\mathcal{X}, \mathcal{X}} \in \mathbb{R}^{nc \times nc}$ where the $(i, j)^{\text{th}}$ (block) element is $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^{c \times c}$. Additionally, we define $\mathbf{K}_{\mathcal{X}, \mathbf{x}} \triangleq [\mathbf{K}(\mathbf{x}_1, \mathbf{x}) \ \dots \ \mathbf{K}(\mathbf{x}_n, \mathbf{x})]^\top \in \mathbb{R}^{nc \times c}$ with $\mathbf{K}_{\mathcal{X}, \mathbf{x}}^\top = \mathbf{K}_{\mathbf{x}, \mathcal{X}}$. For a matrix $\mathbf{J}$, its Moore-Penrose pseudo-inverse is denoted by $\mathbf{J}^\dagger$.

2 Background

Bayesian Framework. Parametric Bayesian methods admit access to a distribution over predictions $\mathbf{f}(\theta, \mathbf{x}^*)$ for unseen test points $\mathbf{x}^*$, through the posterior distribution $p(\theta|\mathcal{D}) \propto p(\theta)p(\mathcal{D}|\theta)$, and the predictive distribution $p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^*|\mathbf{f}(\theta, \mathbf{x}^*))p(\theta|\mathcal{D})d\theta$, where $p(\theta)$ is the prior, and $p(\mathcal{D}|\theta)$ is the likelihood function evaluated on the training data. Both the posterior and the predictive distributions are computationally intractable in all but the simplest cases. In our setting, their calculation involves very high-dimensional integrals, which we can approximate through a Monte Carlo (MC) approximation $p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx 1/S \sum_s p(\mathbf{y}^*|\mathbf{f}(\theta_s, \mathbf{x}^*))$ for $\theta_s \sim q(\theta)$, where $q(\theta)$ is an approximation to the posterior distribution. The effectiveness of classical Markov Chain Monte Carlo (MCMC) methods in posterior sampling diminishes in this setting due to the curse of dimensionality, limiting the tractable techniques available with theoretical guarantees. This limitation necessitates coarser approximations for estimating the posterior $q(\theta)$, leading to the emergence of the following Bayesian methods for posterior approximation.

Proposed in (Gal & Ghahramani, 2016), *Monte Carlo Dropout* (MC-Dropout) takes a trained NN, and uses the *dropout* regularization technique at test time to sample S sub-networks, $\{\mathbf{f}(\theta_s, \mathbf{x})\}_{s=1:S}$ as an MC approximation of the predictive distribution. MC-Dropout is an inexpensive method, yet it is unlikely to converge to the true posterior, and is erroneously multi-modal (Folgot et al., 2021).

In *Variational Inference* (VI) (Hinton & Van Camp, 1993; Graves, 2011), a tractable family of approximating distributions for $p(\theta|\mathcal{D})$ is chosen, denoted by $q_\psi(\theta)$, and parameterized by ψ . The optimal distribution in this family is obtained by finding ψ that minimizes the Kullback-Leibler divergence between $q_\psi(\theta)$ and $p(\theta|\mathcal{D})$. To be computationally viable, mean field and low-rank covariance structures are often required for $q_\psi(\theta)$.

The *Laplace Approximation* (LA) (MacKay, 1992; Ritter et al., 2018) is a tool from classical statistics which approximates the posterior distribution by a Gaussian centered at the maximum a posteriori

solution (MAP) with normalized inverse Fisher information covariance. This is justified by the Bernstein-von Mises Theorem (Van der Vaart, 2000, pp. 140–146), which guarantees that the posterior converges to this distribution in the large-data limit, for well-specified regular models. However, NNs are often over-parameterized, and the regime where $n \rightarrow \infty$ with fixed p is no longer valid or a reasonable reflection of modern deep learning models (De Bortoli & Desolneux, 2022).

These limitations are acknowledged but seldom discussed by the Bayesian Deep Learning (BDL) community, which tends to view this as an additional layer of approximation rather than a modelling error, leading to the development of synonymous LA-inspired methods. However, we will show that such methods typically perform poorly compared to deep ensembles, which are often excluded from comparisons.

We can evaluate the posterior and predictive distribution in the LA using the linearization of $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ around the MAP solution. This approach is known as the *Linearized Laplace Approximation* (LLA) and typically delivers better performance than LA (Immer et al., 2021). LLA generally requires reduction to a subset of parameters or approximations of the covariance structure (Martens & Grosse, 2015) to scale, at the cost of performance. Recent work (Antorán et al., 2022; Ortega et al., 2023) has enabled LLA to become more scalable with better performance for larger models and datasets.

Deep Ensembles (DE) (Lakshminarayanan et al., 2017) are comprised of S networks that are independently trained on the same training data, with different initializations, leading to a collection of parameters $\{\boldsymbol{\theta}_s; s = 1 \dots, S\}$. At test time, our predictive distribution becomes $p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx 1/S \sum_{s=1}^S p(\mathbf{y}|\mathbf{f}(\boldsymbol{\theta}_s, \mathbf{x}))$. Despite their simple construction, DEs are able to obtain samples from different modes of the posterior, and are often considered state-of-the-art for BDL (Hoffmann & Elster, 2021). However, due to the often large cost of training performant neural networks, deep ensembles of reasonable size can be undesirably expensive to obtain.

Stochastic Weight Averaging Gaussian (SWAG) (Maddox et al., 2019) takes a trained network and undergoes further epochs of SGD training to generate a collection of parameter samples. A Gaussian distribution with sample mean and a low-rank approximation of the sample covariance is then used to approximate a posterior mode.

Gaussian Processes. A GP is a stochastic process that is defined by a mean and a kernel function. A GP models the output of a random function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^c$, at a finite collection of points $\mathbf{x}$, as being jointly Gaussian distributed. Conditioning on training data $\mathcal{D}$, it generates a posterior predictive distribution $p(\mathbf{f}(\mathbf{x}_*)|\mathcal{D})$ at a test point $\mathbf{x}_*$. For example, in regression settings where $c = 1$, with the mean and kernel functions $\mu : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\kappa : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, as well as the observations $y \sim \mathcal{N}(f(\mathbf{x}), \sigma^2)$, there is a closed form expression for the predictive distribution, $p(f(\mathbf{x}_*)|\mathcal{D}) \sim \mathcal{N}(\mu(\mathbf{x}_*), \sigma(\mathbf{x}_*))$, where

$$\begin{aligned}\mu(\mathbf{x}_*) &= \mathbf{k}_{\mathbf{x}_*, \mathcal{X}} [\mathbf{K}_{\mathcal{X}, \mathcal{X}} + \sigma^2 \mathbf{I}]^{-1} (\mathbf{y} - \mu(\mathcal{X})) + \mu(\mathbf{x}_*) \\ \sigma(\mathbf{x}_*) &= \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_{\mathbf{x}_*, \mathcal{X}} [\mathbf{K}_{\mathcal{X}, \mathcal{X}} + \sigma^2 \mathbf{I}]^{-1} \mathbf{k}_{\mathcal{X}, \mathbf{x}_*}\end{aligned}$$

for $\mathbf{y} \triangleq [y_1, \dots, y_n]^\top$ and $\mu(\mathcal{X}) \triangleq [\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_n)]^\top$. GPs can yield impressive predictive results when a suitable kernel is chosen (Rasmussen, 1997). However, forming the kernel and solving linear systems makes GP computations intractable for large datasets. Approximations such as sparse variational inference (Titsias, 2009), Nyström methods (Martinsson & Tropp, 2020), and other subspace approximations (Gardner et al., 2018) can alleviate the computational burden; however, these approximations often result in a significant decline in predictive performance.

Neural Tangent Kernel. Under continuous time gradient flow, it can be shown that a NN output $\mathbf{f}(\cdot, \boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}^c$ undergoes kernel gradient descent, namely $\partial_t \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}_t) = - \sum_{i=1}^n \mathbf{K}_{\boldsymbol{\theta}_t}(\mathbf{x}, \mathbf{x}_i) \nabla_{\mathbf{f}} \ell(\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}_t), \mathbf{y}_i)$, where

$$\mathbf{K}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) \triangleq \left\langle \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta}), \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}(\mathbf{y}, \boldsymbol{\theta}) \right\rangle \in \mathbb{R}^{c \times c}, \quad (1)$$

is the *empirical NTK* (Jacot et al., 2018). As the width of a network increases, the empirical NTK converges (in probability) to a deterministic limit, sometimes referred to as the analytic NTK, that is independent of the network’s parameters. Lee et al. (2019) showed that in this limit, the network acts according to its NTK linearization during gradient descent (GD) training. This parameter

independence results in a loss of feature learning in the limiting regime (Yang & Hu, 2021). However, for finite-width NNs, Fort et al. (2020) empirically showed that the empirical NTK becomes “data-dependent” during training. Since we focus exclusively on the finite-width regime, we refer to the empirical NTK simply as the NTK.

Related Works Our method NUQLS shares notable similarities with, and exhibits distinct differences from, several prior works, such as Sampling-LLA, Bayesian Deep Ensembles and *local ensembles*. Due to the breadth and depth of this discussion, we relegate the related works discussion to Appendix B.

3 NUQLS

We now present Neural Uncertainty Quantification by Linearized Sampling (NUQLS), our post-hoc sampling method for quantifying the uncertainty of a trained NN. We begin by presenting the motivation and a high-level overview of our method. Subsequently, we provide theoretical justification, demonstrating that, under specific conditions, the NUQLS samples represent draws from the approximate posterior of the neural network, which is equivalent to a GP defined by the NTK.

Algorithm 1 NUQLS

Input: number of realizations S , weights $\hat{\theta}$.
for $s = 1$ **to** S **do**
 $\theta_{0,s} \leftarrow \hat{\theta} + \mathbf{z}_0$, where $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \gamma^2 \mathbf{I})$
 $\theta_s^* \leftarrow$ Run (stochastic) GD from $\theta_{0,s}$ to
 (approximately) solve (3) and obtain θ_s^*
end for
return $\{\tilde{\mathbf{f}}(\theta_s^*, \cdot)\}_{s=1}^S$

3.1 Motivation and High-level Overview

NNs are often over-parameterized, resulting in non-uniqueness of interpolating solutions, with sub-manifolds of parameter space able to perfectly predict the training data (Hodgkinson et al., 2023). To generate a distribution over predictions, we adopt a Bayesian framework, where the uncertainty in a neural network’s prediction can be interpreted as the spread of possible values the network might produce for a new test point, conditioned on the training data. To quantify this uncertainty, we can evaluate the test point on other “nearby” models with high posterior probability and analyze their range of predictions. To identify such models, we propose using the linearized approximation of the original network around its trained parameters as a simpler yet expressive surrogate. This approach can retain, to a great degree, the rich feature representation of the original network while enabling tractable exploration of the posterior distribution. In the same spirit as DE, in the overparameterized setting, we can fit this linear model to the original training data, using (stochastic) gradient descent with different initializations, resulting in an ensemble of linear predictors. Not only does this ensemble explain the training data well, but it also provides a practical way to estimate predictive uncertainty.

More precisely, let $\hat{\theta}$ be a set of parameters obtained after training the original NN. Linearizing $\mathbf{f}$ around $\hat{\theta}$ gives

$$\mathbf{f}(\theta, \mathbf{x}) \approx \tilde{\mathbf{f}}(\theta, \mathbf{x}) \triangleq \mathbf{f}(\hat{\theta}, \mathbf{x}) + \mathbf{J}(\hat{\theta}, \mathbf{x})(\theta - \hat{\theta}), \quad (2)$$

where $\mathbf{J}(\cdot, \mathbf{x}) = [\partial \mathbf{f}(\cdot, \mathbf{x}) / \partial \theta]^\top \in \mathbb{R}^{c \times p}$ is the Jacobian of $\mathbf{f}$. Using the linear approximation (2), we consider

$$\min_{\theta} \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\theta, \mathbf{x}_i), \mathbf{y}_i). \quad (3)$$

In overparameterized settings, (3) may have infinitely many solutions. To identify these solutions and create our ensemble of linear predictors, we employ (stochastic) gradient descent, initialized at zero-mean isotropic Gaussian perturbations of the trained parameter, $\hat{\theta}$. The pseudo-code for this algorithm is provided in Algorithm 1. For a given test point $\mathbf{x}^*$, the mean prediction and uncertainty can be computed using $\{\tilde{\mathbf{f}}(\theta_s^*, \mathbf{x}^*)\}_{s=1}^S$.

Note that while the training cost for a linearised network is only slightly higher per epoch compared to standard NN training, each network in the NUQLS ensemble is initialized in a neighborhood of a local minimum of the original NN. As a result, NUQLS often requires significantly fewer epochs to converge, leading to an order-of-magnitude computational speedup relative to DE (see Tables 1, 2 and 9 for wall-clock time comparisons, and Appendix E for a more in-depth analysis of the computation costs.).

3.2 Theoretical Analysis

We now establish the key property of Algorithm 1: *under mild conditions, NUQLS generates samples from the approximate posterior of the neural network, which in many cases corresponds to a Gaussian process defined by the NTK*. Proofs are provided in Appendix A.

Suppose $\theta^\dagger$ is any solution to (3). Using $\theta^\dagger$, one can construct a family of solutions to (3) as

$$\theta_z^* = \theta^\dagger + \left(\mathbf{I} - \mathbf{J}_{\mathcal{X}}^\dagger \mathbf{J}_{\mathcal{X}} \right) \mathbf{z}, \quad \forall \mathbf{z} \in \mathbb{R}^p, \quad (4)$$

where $\hat{\theta}$ is the parameters of the trained NN and $\mathbf{J}_{\mathcal{X}} = [\mathbf{J}^\top(\hat{\theta}, \mathbf{x}_1) \dots \mathbf{J}^\top(\hat{\theta}, \mathbf{x}_n)]^\top \in \mathbb{R}^{nc \times p}$. Note that the second term in (4) consists of all vectors in the null space of $\mathbf{J}_{\mathcal{X}}$. Since any such $\theta^\dagger$ can be decomposed as the direct sum of components in the null space of $\mathbf{J}_{\mathcal{X}}$ and its orthogonal complement, the family of solutions in (4) depends on the choice of $\theta^\dagger$. However, under certain assumptions, we can ensure that the representation (4) is uniquely determined, i.e., $\theta^\dagger$ can be taken as the unique solution to (3) that is orthogonal to the null space of $\mathbf{J}_{\mathcal{X}}$. More precisely, we can show that, under these assumptions on the loss, $\theta^\dagger$ in (4) can be taken as the unique solution to

$$\min_{\theta} \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\theta, \mathbf{x}_i), \mathbf{y}_i), \quad \text{s.t. } \theta \in \text{Range}(\mathbf{J}_{\mathcal{X}}^\top). \quad (5)$$

Lemma 3.1. *Suppose the loss, $\ell(\cdot, \mathbf{y})$, is either:*

- *strongly convex in its first argument, or*
- *strictly convex in its first argument, and a solution to (3) exists.*

The problem (5) admits a unique solution.

As it turns out, any solution of the form (4) can be efficiently obtained using (stochastic) gradient descent.

Theorem 3.2. *Consider the optimization problem (3) and assume $\mathbf{J}_{\mathcal{X}}$ is full row-rank.*

- **(Gradient Descent)** *Suppose $\ell(\mathbf{f}, \mathbf{y})$ is strictly convex with locally Lipschitz continuous gradient, both with respect to $\mathbf{f}$, and the problem (3) admits a solution. Gradient descent, initialized at $\mathbf{z}$ and with appropriate learning rate, converges to (4).*
- **(Stochastic Gradient Descent)** *Suppose $\ell(\mathbf{f}, \mathbf{y})$ is strongly convex with Lipschitz continuous gradient, both with respect to $\mathbf{f}$, and any solution to the problem (3) is interpolating. Stochastic gradient descent, initialized at $\mathbf{z}$ and with small enough learning rate, converges to (4) with probability one.*

We note that the local smoothness requirement in the first part of Theorem 3.2 is a relatively mild assumption; for example, it holds if we simply assume that ℓ is twice continuously differentiable. Also, the full-row rank assumption on $\mathbf{J}_{\mathcal{X}}$ in the second part of Theorem 3.2 is reasonable for highly over-parameterized networks; e.g., see Liu et al. (2022).

Now, suppose $\mathbf{J}_{\mathcal{X}}$ is full row-rank and the assumption of Lemma 3.1 holds, ensuring the existence of the unique solution $\theta^\dagger$ to (5). Noting $\text{Range}(\mathbf{J}_{\mathcal{X}}^\top) = \text{Range}(\mathbf{J}_{\mathcal{X}}^\dagger)$, we can write (4) as

$$\theta_z^* = \mathbf{J}_{\mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{w} + \left(\mathbf{I} - \mathbf{J}_{\mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{J}_{\mathcal{X}} \right) \mathbf{z}, \quad \forall \mathbf{z} \in \mathbb{R}^p$$

where $\mathbf{w}$ is a vector for which $\theta^\dagger = \mathbf{J}_{\mathcal{X}}^\top \mathbf{w}$, and $\mathbf{K}_{\mathcal{X}, \mathcal{X}} \triangleq \mathbf{J}_{\mathcal{X}} \mathbf{J}_{\mathcal{X}}^\top = \mathbf{K}_{\hat{\theta}}(\mathcal{X}, \mathcal{X}) \in \mathbb{R}^{nc \times nc}$ is the Gram matrix of the empirical NTK (1) on the training data $\mathcal{X}$. Setting $\mathbf{z} = \hat{\theta} - \mathbf{z}_0$ for some $\mathbf{z}_0$, we get

$$\begin{aligned} \theta^* &= \mathbf{J}_{\mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{w} + \left(\mathbf{I} - \mathbf{J}_{\mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{J}_{\mathcal{X}} \right) (\hat{\theta} - \mathbf{z}_0), \\ \tilde{\mathbf{f}}(\theta^*, \mathbf{x}) &= \mathbf{f}(\hat{\theta}, \mathbf{x}) + \mathbf{K}_{\mathbf{x}, \mathcal{X}} \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} (\mathbf{w} - \mathbf{J}_{\mathcal{X}} \hat{\theta}) + \left(\mathbf{K}_{\mathbf{x}, \mathcal{X}} \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{J}_{\mathcal{X}} - \mathbf{J}(\hat{\theta}, \mathbf{x}) \right) \mathbf{z}_0, \end{aligned} \quad (6)$$

where $\mathbf{K}_{\mathbf{x}, \mathcal{X}} \triangleq \mathbf{K}_{\hat{\theta}}(\mathbf{x}, \mathcal{X}) \in \mathbb{R}^{c \times nc}$. Taking $\mathbf{z}_0$ to be a random variable, we form an ensemble of predictors $\{\tilde{\mathbf{f}}(\theta^*, \mathbf{x})\}_{\mathbf{z}}$, where each θ^* is formed from the projection of a random $\mathbf{z}$ onto $\text{Null}(\mathbf{J}_{\mathcal{X}})$. We require $\mathbf{z}$'s distribution to be symmetric, isotropic, and centered at $\hat{\theta}$, as we do not know *a priori*

which directions contain more information. We take the maximum entropy distribution for a given mean and variance, which is Gaussian¹. Hence, we let $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \gamma^2 \mathbf{I})$, for some hyper-parameter $\gamma \in \mathbb{R}$. The expectation and variance of (6), and the distribution of the predictor $\mathbf{f}(\boldsymbol{\theta}^*, \mathbf{x})$, are then

$$\mathbb{E}(\tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathbf{x})) = \boldsymbol{\mu}(\hat{\boldsymbol{\theta}}, \mathbf{x}) = \mathbf{K}_{\mathbf{x}, \mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{J}_{\mathcal{X}}(\boldsymbol{\theta}^\dagger - \hat{\boldsymbol{\theta}}) + \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}), \quad (7)$$

$$\text{Var}(\tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathbf{x})) = \boldsymbol{\sigma}^2(\hat{\boldsymbol{\theta}}, \mathbf{x}) = (\mathbf{K}_{\mathbf{x}, \mathbf{x}} - \mathbf{K}_{\mathbf{x}, \mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{K}_{\mathcal{X}, \mathbf{x}}) \gamma^2. \quad (8)$$

$$\mathbf{f}(\boldsymbol{\theta}^*, \mathbf{x}) \stackrel{\text{approx}}{\sim} \mathcal{N}(\boldsymbol{\mu}(\hat{\boldsymbol{\theta}}, \mathbf{x}), \boldsymbol{\sigma}^2(\hat{\boldsymbol{\theta}}, \mathbf{x})). \quad (9)$$

Remark 3.3 (Connections to GP: Regression). For scalar-valued $f : \mathbb{R}^p \times \mathbb{R}^d \rightarrow \mathbb{R}$ with quadratic loss $\ell(f(\boldsymbol{\theta}, \mathbf{x}), y) = (f(\boldsymbol{\theta}, \mathbf{x}) - y)^2$, we can explicitly write $\boldsymbol{\theta}^\dagger = \mathbf{J}_{\mathcal{X}}^\dagger(\mathbf{y} - \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathcal{X}) + \mathbf{J}_{\mathcal{X}} \hat{\boldsymbol{\theta}})$, where $\mathbf{f}(\boldsymbol{\theta}, \mathcal{X}) \triangleq [f(\boldsymbol{\theta}, \mathbf{x}_1), f(\boldsymbol{\theta}, \mathbf{x}_2), \dots, f(\boldsymbol{\theta}, \mathbf{x}_n)]^\top$ and $\mathbf{y} \triangleq [y_1, \dots, y_n]^\top$. For $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \gamma^2 \mathbf{I})$, we thus have $f(\boldsymbol{\theta}^*, \mathbf{x}) \stackrel{\text{approx}}{\sim} \mathcal{N}(\boldsymbol{\mu}(\hat{\boldsymbol{\theta}}, \mathbf{x}), \sigma^2(\hat{\boldsymbol{\theta}}, \mathbf{x}))$ with

$$\begin{aligned} \boldsymbol{\mu}(\hat{\boldsymbol{\theta}}, \mathbf{x}) &= \mathbf{k}_{\mathbf{x}, \mathcal{X}} \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} (\mathbf{y} - \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathcal{X})) + f(\hat{\boldsymbol{\theta}}, \mathbf{x}), \\ \sigma^2(\hat{\boldsymbol{\theta}}, \mathbf{x}) &= (\kappa(\mathbf{x}, \mathbf{x}) - \mathbf{k}_{\mathbf{x}, \mathcal{X}} \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{k}_{\mathcal{X}, \mathbf{x}}) \gamma^2. \end{aligned}$$

By the full-rank assumption on the Jacobian, this amounts to the conditional distribution of the following normal distribution, conditioned on interpolation $\mathbf{f}(\boldsymbol{\theta}^*, \mathcal{X}) = \mathbf{y}$,

$$\begin{bmatrix} \mathbf{f}(\boldsymbol{\theta}^*, \mathcal{X}) \\ f(\boldsymbol{\theta}^*, \mathbf{x}) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \\ f(\hat{\boldsymbol{\theta}}, \mathbf{x}) \end{bmatrix}, \gamma^2 \begin{bmatrix} \mathbf{K}_{\mathcal{X}, \mathcal{X}} & \mathbf{k}_{\mathbf{x}, \mathcal{X}} \\ \mathbf{k}_{\mathcal{X}, \mathbf{x}}^\top & \kappa_{\mathbf{x}, \mathbf{x}} \end{bmatrix} \right). \quad (10)$$

Therefore, $f(\boldsymbol{\theta}^*, \mathbf{x})$ follows a GP with an NTK kernel. Conditioning on $\mathbf{f}(\boldsymbol{\theta}^*, \mathcal{X}) = \mathbf{y}$ is reasonable, since by construction $\mathbf{f}(\boldsymbol{\theta}^*, \mathcal{X}) \approx \tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathcal{X})$, and under the full-rank assumption of $\mathbf{J}_{\mathcal{X}}$, we have $\tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathcal{X}) = \mathbf{y}$.

Remark 3.4 (Connections to GP: General Loss). Beyond quadratic loss, for a general loss function satisfying the assumptions of Lemma 3.1, a clear GP posterior interpretation like that in (10) may not exist. Nevertheless, we can still derive related insights. If $\hat{\boldsymbol{\theta}}$ is an interpolating solution from initial training, which is common for modern NNs, then as long as $\mathbf{J}_{\mathcal{X}}$ is full row-rank, solving (3) is equivalent to finding $\boldsymbol{\theta} \in \mathbb{R}^p$ such that $\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathcal{X}) = \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathcal{X})$, i.e., find $\boldsymbol{\theta} \in \mathbb{R}^p$ such that $\mathbf{J}_{\mathcal{X}}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) = \mathbf{0}$. So we obtain (9) with $\boldsymbol{\mu}(\hat{\boldsymbol{\theta}}, \mathbf{x}) = \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x})$ and $\boldsymbol{\sigma}^2(\hat{\boldsymbol{\theta}}, \mathbf{x}) = (\mathbf{K}_{\mathbf{x}, \mathbf{x}} - \mathbf{K}_{\mathbf{x}, \mathcal{X}}^\top \mathbf{K}_{\mathcal{X}, \mathcal{X}}^{-1} \mathbf{K}_{\mathcal{X}, \mathbf{x}}) \gamma^2$ as the conditional distribution of (10), conditioned on the event $\mathbf{f}(\boldsymbol{\theta}^*, \mathcal{X}) = \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathcal{X})$, i.e., interpolation with $\ell(\mathbf{f}(\boldsymbol{\theta}^*, \mathbf{x}_i), \mathbf{y}_i) = 0$ for $i = 1, \dots, n$.

The Punchline. Drawing samples from the posterior (9) by explicitly calculating (7) and (8) can be intractable in large-scale settings. Moreover, it can be numerically unstable due to the highly ill-conditioned nature of the NTK matrix². However, by combining the above derivations with Theorem 3.2, we arrive at the key property of Algorithm 1: it enables efficient sampling from the posterior (9).

Corollary 3.5 (Key Property of NUQLS). *With the assumptions of Theorem 3.2, the samples generated by Algorithm 1 represent draws from the predictive distribution in (9).*

Hence, we approximate (7) and (8) by computing the sample mean and covariance of $\{\tilde{\mathbf{f}}(\boldsymbol{\theta}_s^*, \mathbf{x})\}_{s=1}^S$ obtained from Algorithm 1. By the law of large numbers, the quality of these approximations improves as $S \rightarrow \infty$.

Remark 3.6. Loss functions that do not satisfy the assumption of Lemma 3.1, such as the cross-entropy loss, may fail to yield a unique representation of (4), so the above posterior analysis does not apply. However, our experiments demonstrate that Algorithm 1 can still generate samples that effectively capture the posterior variance (see Section 4.4) and posterior mean (see Appendix G.1). In cases where (3) lacks a solution, Algorithm 1 can still be executed by terminating the iterations of (stochastic) GD early. Investigating the distribution of the resulting ensemble and its connection to an explicit posterior remains a potential direction for future research.

¹Heavier-tailed distributions matching the above criteria, e.g. logistic distributions, may improve results.

²Recall that the condition number of $\mathbf{K}_{\mathcal{X}, \mathcal{X}}$ is the square of that of $\mathbf{J}_{\mathcal{X}}$.

4 Experiments

We now empirically demonstrate the result of Theorem 3.2, as well as compare the performance of our method with alternatives on various regression and classification tasks. Implementation details are given in Appendix I. The PyTorch implementation of our experiments is available here. We have also released our method as a package. For additional experimental results, please see Appendix H.

4.1 Empirical Convergence

We demonstrate empirically the convergence of the predictive distribution of NUQLS to that of an NTK-GP. We take a series of MLPs with NTK scaling (Jacot et al., 2018), and train these NNs on normalized Gaussian data (a regression task). Under this setting, the condition number of the resultant NTK is reasonable ($\approx 10^3 - 10^8$, compared to $\approx 10^{17}$ on some UCI regression datasets). This allows us to explicitly compute the predictive variance for the NTK-GP for some normalized Gaussian test set. We then also compute the predictive variance using NUQLS, and take the l_2 norm of the difference between the two variance sets, which we term Squared Error of the Variance (SEV). We plot this against number of epochs of training for NUQLS, and the number of NUQLS realizations s . We also plot the average training loss of the linearised networks in the NUQLS ensemble. The values are averaged over 10 random realizations. The results are displayed in Figure 2. Note that the SEV cannot be zero, due to the condition number of the NTK; this affects the solution of the linear system $K_{X \times Y} = K_{X \times x}$ in the computation of the variance for the NTK-GP. However, we see clear convergence of our method to the distribution of an NTK-GP as the ensemble members of NUQLS approach their minima, and as the number of ensemble members increases.

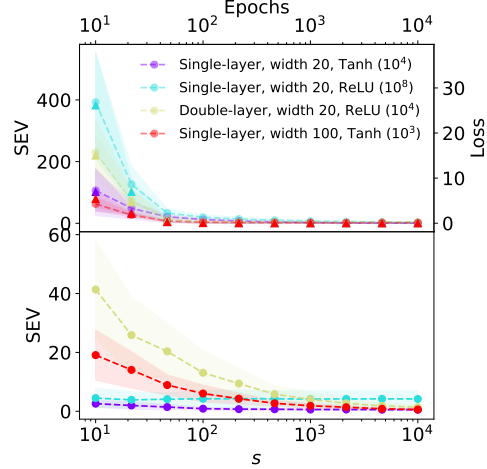


Figure 2: Plot of SEV (●) and NUQLS train loss (▲) against (top) number of epochs of training for NUQLS and (bottom) number of NUQLS realizations. Bracketed number is condition number of NTK Gram matrix. Mean and 95% confidence intervals are shown from 10 random realizations.

4.2 Toy Regression

We compare the performance of our method on a toy regression problem, taken from (Hernández-Lobato & Adams, 2015) and extended in (Park & Blei, 2024). In Figure 1, we take 20 uniformly sampled points in the domain $x \in [-4, -2] \cup [2, 4]$, and let $y = x^3 + \epsilon$, $\epsilon \sim \mathcal{N}(0, 3^2)$. A small MLP was trained on these data and used for prediction. We apply VI, SWAG, LA, LLA, DE and NUQLS to the network to find a predictive mean and uncertainty. Close to the training data, i.e. in $[-4, -2] \cup [2, 4]$ we expect low uncertainty; outside of this region, the uncertainty should grow with distance from the training points. VI underestimates, while SWAG and LA overestimate the uncertainty. DE grows more uncertain with distance from the training points, however both NUQLS and the LLA contain the underlying target curve within their confidence intervals. Note that deep ensembles output a heteroskedastic variance term, and were trained on a Gaussian likelihood; in comparison, the variances for LLA and NUQLS were computed post-hoc.

4.3 UCI Regression

In Tables 1 and 9, we compare NUQLS with DE, LLA and SWAG on a series of UCI regression problems. Mean squared error (MSE) and expected calibration error (ECE) respectively evaluate the predictive and UQ performance, with Gaussian negative log likelihood (NLL) evaluating both. See (Nemani et al., 2023a, §4.11) for an explanation of ECE. We see that NUQLS consistently has the (equal) best ECE (except for the Song dataset, where it falls short of the LLA ECE by 0.1%). It has comparable or better NLL than other methods on all datasets, and often gives an improvement on RMSE. Finally, it is the quickest method, often by a very significant margin, and it does not fail

Table 1: Comparing performance of NUQLS, DE, LLA and SWAG on UCI regression tasks. NUQLS performs as well as or better than all other methods, while showing a speed up over other methods; this speed up increases with the size of the datasets. LLA-K denotes LLA with a KFAC covariance structure. Reported time for NUQLS, LLA and SWAG includes the training time for the original NN, with the run-time of the post-hoc method given in brackets.

Dataset	Method	RMSE ↓	NLL ↓	ECE ↓	Time(s)
Energy	NUQLS	0.047 ±0.006	-2.400 ±0.209	0.002 ±0.002	8.374 (0.151)
	DE	0.218±0.032	-1.651 ±0.783	0.004 ±0.002	102.244
	LLA	0.048 ±0.006	-2.475 ±0.128	0.004 ±0.004	8.491 (0.269)
	SWAG	0.058 ±0.015	-1.950±0.158	0.080±0.011	45.306 (37.084)
Kin8nm	NUQLS	0.252 ±0.005	-0.796±0.025	0.000 ±0.000	26.570 (0.264)
	DE	0.252 ±0.006	-0.914 ±0.028	0.002±0.001	73.967
	LLA	0.260 ±0.010	-0.783±0.054	0.001 ±0.001	38.272 (11.966)
	SWAG	0.457±0.149	-0.006±0.295	0.054±0.012	176.569 (150.263)
Protein	NUQLS	0.623 ±0.005	0.209 ±0.047	0.002 ±0.000	81.264 (1.356)
	DE	0.741±0.052	0.203 ±0.203	0.011 ±0.020	1014.827
	LLA-K	0.640±0.007	0.458±0.071	0.002 ±0.000	89.414 (9.506)
	SWAG	0.730±0.044	0.187 ±0.080	0.002 ±0.002	548.88 (468.972)

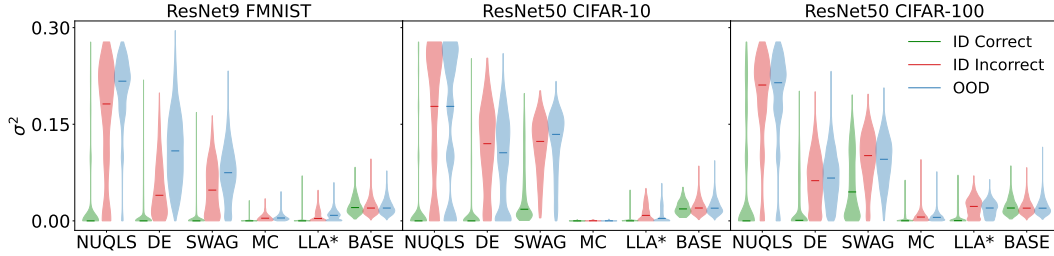


Figure 3: Violin plot of VMSP, for correctly predicted ID test points, incorrectly predicted ID test points, and OoD test points. Median is shown, with violin width depicting density. Low variance is expected for ID correct points, and large variance for ID incorrect and OoD points.

on any datasets, like the other methods do. Note that for the two largest datasets, Protein and Song, we required approximations on the covariance structure of LLA (see (Daxberger et al., 2021)). A detailed explanation of the hyper-parameter tuning method for NUQLS is given in Appendix F.1.

4.4 Image Classification - Uncertainty

We now compare the UQ performance of NUQLS, DE, SWAG, LLA* (LLA with a last-layer and KFAC approximation), and MC-Dropout (MC), on larger image classification tasks. We take variance of the maximum predicted softmax probability (VMSP), for a given test point, as the correct quantifier of uncertainty in this setting (see Appendix C for justification).

Figure 3 presents a violin plot of the VMSP for three test-groups: correctly predicted in-distribution (FashionMNIST, CIFAR-10) test points, incorrectly predicted in-distribution test points, and out-of-distribution (MNIST, CIFAR-100) test points. We would expect that there should be, on average, much smaller uncertainty for ID test points that have been correctly predicted, and larger uncertainty for incorrectly predicted ID and OoD test points. We compare against a completely randomized baseline method (BASE), where we sample 10 standard normal realizations of logits, passed through a softmax. In Table 10 in Appendix H.8, we display the corresponding median and sample skew values for each method in each test group, to quantify the distribution of VMSP for each test set. Ideally, a method should far outperform the baseline, so we can use the median and sample skew difference between a method and the baseline as a way to compare different methods. We see that NUQLS outperforms all other methods, including the SOTA method DE. In Appendix H.7 we provide additional experimental evaluation of NUQLS. In Figure 6 we evaluate NUQLS on a ResNet50 trained on both SVHN and ImageNet, displaying the scalability of our method, as well as providing comparison with other competing methods, including Bayesian Deep Ensembles (BDE), Spectral-Normalized Neural Gaussian Process (SNGP), BatchEnsemble (BE), and Stochastic

Gradient Langevin Dynamics (SGLD). Against these competing methods, NUQLS performs the strongest.

5 Conclusion

We have presented NUQLS, a Bayesian, post-hoc UQ method that approximates the predictive distribution of an over-parameterized NN through GD/SGD training of linear networks, allowing scalability without sacrificing performance. Under assumptions on the loss function, this predictive distribution reduces to a GP using the NTK. We find that our method is competitive with, and often far outperforms, existing UQ methods on regression and classification tasks, whilst providing a novel connection between NNs, GPs and the NTK.

Limitation. A theoretical limitation of this work is that its connection to the NTK-GP does not extend to loss functions that violate the assumptions of Theorem 3.2, such as the cross-entropy loss. The strong empirical performance of NUQLS on classification tasks motivates future research to extend Theorem 3.2 to broader classes of loss functions and alternative optimization algorithms. A further limitation of the method is the dependence of NUQLS on the linearization approximation. As can be seen in Appendix H.1, when the target neural network is poorly trained, and hence the loss-landscape is far from flat around the ‘trained’ parameters, the performance of NUQLS suffers. However, when the network is well-trained, as is common in practical settings, the linear approximation holds and hence NUQLS performs well.

Acknowledgments and Disclosure of Funding

Liam Hodgkinson is supported by the Australian Research Council through a Discovery Early Career Researcher Award (DE240100144). Fred Roosta was partially supported by the Australian Research Council through an Industrial Transformation Training Centre for Information Resilience (IC200100022).

References

- Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76: 243–297, 2021. ISSN 1566-2535. doi: <https://doi.org/10.1016/j.inffus.2021.05.008>.
- Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., and Yu, D. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.
- Abe, T., Buchanan, E. K., Pleiss, G., Zemel, R., and Cunningham, J. P. Deep ensembles work, but are they necessary? *Advances in Neural Information Processing Systems*, 35:33646–33660, 2022.
- Altieri, A., Romanelli, M., Pichler, G., Alberge, F., and Piantanida, P. Beyond the norms: Detecting prediction errors in regression models. *Forty-first International Conference on Machine Learning*, 2024.
- Antorán, J., Padhy, S., Barbano, R., Nalisnick, E., Janz, D., and Hernández-Lobato, J. M. Sampling-based inference for large linear models, with application to linearised laplace. *arXiv preprint arXiv:2210.04994*, 2022.
- Bassily, R., Belkin, M., and Ma, S. On exponential convergence of SGD in non-convex over-parametrized learning. *arXiv preprint arXiv:1811.02564*, 2018.
- Bitterwolf, J., Mueller, M., and Hein, M. In or out? fixing imagenet out-of-distribution detection evaluation. *arXiv preprint arXiv:2306.00826*, 2023.
- Blondel, M. and Roulet, V. The elements of differentiable programming. *arXiv preprint arXiv:2403.14606*, 2024.

- Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural network. *International Conference on Machine Learning*, pp. 1613–1622, 2015.
- Bubeck, S. et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.
- Chan, M., Molina, M., and Metzler, C. Estimating epistemic and aleatoric uncertainty with a single model. *Advances in Neural Information Processing Systems*, 37:109845–109870, 2024.
- Dadalto, E., Romanelli, M., Pichler, G., and Piantanida, P. A data-driven measure of relative uncertainty for misclassification detection. *arXiv preprint arXiv:2306.01710*, 2023.
- Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace redux – effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34: 20089–20103, 2021.
- De Bortoli, V. and Desolneux, A. On quantitative Laplace-type convergence results for some exponential probability measures with two applications. To appear in the *Journal of Machine Learning Research*, 2022.
- Deng, L., Hinton, G., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: An overview. *International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603, 2013.
- Deng, Z., Zhou, F., and Zhu, J. Accelerated linearized laplace approximation for bayesian deep learning. *Advances in Neural Information Processing Systems*, 35:2695–2708, 2022.
- Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423.
- Eschenhagen, R., Daxberger, E., Hennig, P., and Kristiadi, A. Mixtures of laplace approximations for improved post-hoc uncertainty in deep learning. *arXiv preprint arXiv:2111.03577*, 2021.
- Folgoc, L. L., Baltatzis, V., Desai, S., Devaraj, A., Ellis, S., Manzanera, O. E. M., Nair, A., Qiu, H., Schnabel, J., and Glocker, B. Is mc dropout bayesian? *arXiv preprint arXiv:2110.04286*, 2021.
- Foong, A. Y., Li, Y., Hernández-Lobato, J. M., and Turner, R. E. ‘in-between’ uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.
- Ford, N., Gilmer, J., Carlini, N., and Cubuk, D. Adversarial examples are a natural consequence of test error in noise. *International Conference on Machine Learning*, 97, 2019.
- Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., and Ganguli, S. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 34, 2020.
- Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., and Bloch, I. Encoding the latent posterior of bayesian neural networks for uncertainty quantification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(4):2027–2040, 2023.
- Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/gal16.html>.
- Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.
- Garrigos, G. and Gower, R. M. Handbook of convergence theorems for (stochastic) gradient methods. *arXiv preprint arXiv:2301.11235*, 2023.

- Granese, F., Romanelli, M., Gorla, D., Palamidessi, C., and Piantanida, P. Doctor: A simple method for detecting misclassification errors. *Advances in Neural Information Processing Systems*, 34: 5669–5681, 2021.
- Graves, A. Practical variational inference for neural networks. *Advances in Neural Information Processing Systems*, 24, 2011.
- Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. *International Conference on Machine Learning*, 70:1321–1330, 2017.
- Havasi, M., Jenatton, R., Fort, S., Liu, J. Z., Snoek, J., Lakshminarayanan, B., Dai, A. M., and Tran, D. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.
- He, B., Lakshminarayanan, B., and Teh, Y. W. Bayesian deep ensembles via the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:1010–1022, 2020.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
- Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., and Song, D. Natural adversarial examples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15262–15271, 2021.
- Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of bayesian neural networks. *International Conference on Machine Learning*, pp. 1861–1869, 2015.
- Hinton, G. E. and Van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, 1993.
- Hodgkinson, L., van der Heide, C., Salomone, R., Roosta, F., and Mahoney, M. W. The interpolating information criterion for overparameterized models. *arXiv preprint arXiv:2307.07785v1*, 2023.
- Hoffmann, L. and Elster, C. Deep ensembles from a bayesian perspective. *arXiv preprint arXiv:2105.13283*, 2021.
- Huang, Z., Lam, H., and Zhang, H. Efficient uncertainty quantification and reduction for over-parameterized neural networks. *Advances in neural information processing systems*, 36: 64428–64467, 2023.
- Huseljic, D., Sick, B., Herde, M., and Kottke, D. Separation of aleatoric and epistemic uncertainty in deterministic deep neural networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 9172–9179. IEEE, 2021.
- 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, 2021.
- Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.
- Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.
- Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*, pp. 795–811. Springer, 2016.
- Khan, M. E. E., Immer, A., Abedi, E., and Korzepa, M. Approximate inference turns deep networks into gaussian processes. *Advances in Neural Information Processing Systems*, 33, 2019.

- Kotelevskii, N., Artemenkov, A., Fedyanin, K., Noskov, F., Fishkov, A., Shelmanov, A., Vazhentsev, A., Petiushko, A., and Panov, M. Nonparametric uncertainty quantification for single deterministic neural network. *Advances in Neural Information Processing Systems*, 35:36308–36323, 2022.
- Krishnan, R., Esposito, P., and Subedar, M. Bayesian-torch: Bayesian neural network layers for uncertainty estimation, January 2022. URL <https://github.com/IntelLabs/bayesian-torch>.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.
- Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems*, 30, 2017.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in Neural Information Processing Systems*, 32, 2019.
- Lehmann, N., Gottschling, N. M., Gawlikowski, J., Stewart, A. J., Depeweg, S., and Nalisnick, E. Lightning uq box: Uncertainty quantification for neural networks. *Journal of Machine Learning Research*, 26(54):1–7, 2025. URL <http://jmlr.org/papers/v26/24-2110.html>.
- Lei, J. and Wasserman, L. Distribution-free prediction bands for non-parametric regression. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 76(1):71–96, 2014.
- Liu, C., Zhu, L., and Belkin, M. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 59:85–116, 2022.
- MacKay, D. J. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.
- Maddox, W., Garipov, T., Izmailov, P., Vetrov, D., and Wilson, A. G. A simple baseline for bayesian uncertainty in deep learning, 2019.
- Madras, D., Atwood, J., and D’Amour, A. Detecting underspecification with local ensembles. *arXiv preprint arXiv:1910.09573*, 2019.
- Malitsky, Y. and Mishchenko, K. Adaptive gradient descent without descent. In *International Conference on Machine Learning*, pp. 6702–6712. PMLR, 2020.
- Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417. PMLR, 2015.
- Martinsson, P.-G. and Tropp, J. A. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.
- Masegosa, A. Learning under model misspecification: Applications to variational and ensemble methods. *Advances in Neural Information Processing Systems*, 33:5479–5491, 2020.
- Matthews, A. G. d. G., Hron, J., Turner, R. E., and Ghahramani, Z. Sample-then-optimize posterior sampling for bayesian linear models. *NeurIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.
- Meurant, G. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*. SIAM, 2006.
- Miani, M., Beretta, L., and Hauberg, S. Sketched lanczos uncertainty score: a low-memory summary of the fisher information. *arXiv preprint arXiv:2409.15008*, 2024a.
- Miani, M., Roy, H., and Hauberg, S. Bayes without underfitting: Fully correlated deep learning posteriors via alternating projections. *arXiv preprint arXiv:2410.16901*, 2024b.
- Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P., and Gal, Y. Deterministic neural networks with inductive biases capture epistemic and aleatoric uncertainty. *arXiv preprint arXiv:2102.11582*, 2, 2021.

- Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P. H., and Gal, Y. Deep deterministic uncertainty: A new simple baseline. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24384–24394, 2023.
- Naeini, M. P., Cooper, G., and Hauskrecht, M. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.
- Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. Speech recognition using deep neural networks. *IEEE access*, 7:19143–19165, 2019.
- Neal, R. M. *Bayesian Learning for Neural Networks, Vol. 118 of Lecture Notes in Statistics*. Springer-Verlag, 1996.
- Nemani, V., Biggio, L., Huan, X., Hu, Z., Fink, O., Tran, A., Wang, Y., Zhang, X., and Hu, C. Uncertainty quantification in machine learning for engineering design and health prognostics: A tutorial. *Mechanical Systems and Signal Processing*, 205:110796, 2023a.
- Nemani, V., Biggio, L., Huan, X., Hu, Z., Fink, O., Tran, A., Wang, Y., Zhang, X., and Hu, C. Uncertainty quantification in machine learning for engineering design and health prognostics: A tutorial. *Mechanical Systems and Signal Processing*, 205:110796, 2023b.
- Nixon, J., Dusenberry, M. W., Zhang, L., Jerfel, G., and Tran, D. Measuring calibration in deep learning. In *CVPR workshops*, volume 2, 2019.
- Ortega, L. A., Santana, S. R., and Hernández-Lobato, D. Variational linearized laplace approximation for bayesian deep learning. *arXiv preprint arXiv:2302.12565*, 2023.
- Papadopoulos, H., Proedrou, K., Vovk, V., and Gammernan, A. Inductive confidence machines for regression. In *13th European Conference on Machine Learning*, pp. 345–356. Springer, 2002.
- Park, Y. and Blei, D. Density uncertainty layers for reliable uncertainty estimation. *International Conference on Artificial Intelligence and Statistics*, 238:163–171, 2024.
- Rasmussen, C. E. *Evaluation of Gaussian processes and other methods for non-linear regression*. PhD thesis, University of Toronto Toronto, Canada, 1997.
- Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005. ISBN 9780262256834. doi: 10.7551/mitpress/3206.001.0001. URL <https://doi.org/10.7551/mitpress/3206.001.0001>.
- Ray, P. P. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 2023.
- Redmon, J. and Farhadi, A. Yolo9000: Better, faster, stronger. *Conference on Computer Vision and Pattern Recognition*, pp. 7263–7271, 2017.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. *Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.
- Ritter, H., Botev, A., and Barber, D. A scalable Laplace approximation for neural networks. *International Conference on Learning Representations*, 6, 2018.
- Roux, N., Schmidt, M., and Bach, F. A stochastic gradient method with an exponential convergence rate for finite training sets. *Advances in neural information processing systems*, 25, 2012.
- Shalev-Shwartz, S. and Zhang, T. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(1), 2013.
- Tagasovska, N. and Lopez-Paz, D. Single-model uncertainties for deep learning. *Advances in neural information processing systems*, 32, 2019.
- Titsias, M. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pp. 567–574. PMLR, 2009.
- Van der Vaart, A. W. *Asymptotic Statistics*, volume 3. Cambridge University Press, 2000.

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- Vovk, V., Gammerman, A., and Shafer, G. *Algorithmic Learning in a Random World*, volume 29. Springer, 2005.
- Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. pp. 681–688, 2011.
- Wen, Y., Tran, D., and Ba, J. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.
- Xie, Z., Tang, Q.-Y., Cai, Y., Sun, M., and Li, P. On the power-law hessian spectrums in deep learning. *arXiv preprint arXiv:2201.13011*, 2022.
- Yang, G. and Hu, E. J. Feature learning in infinite-width neural networks. *International Conference on Machine Learning*, 139, 2021.

A Proofs

Proof of Lemma 3.1. First, we note that by the assumption on ℓ , a solution to (5) always exists. Suppose to the contrary that (5) has two distinct solutions $\tilde{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}$ such that $\tilde{\boldsymbol{\theta}} \neq \hat{\boldsymbol{\theta}}$. Since $\tilde{\boldsymbol{\theta}} \in \text{Range}([\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})]^\top)$ and $\hat{\boldsymbol{\theta}} \in \text{Range}([\mathbf{J}(\tilde{\boldsymbol{\theta}}, \mathcal{X})]^\top)$, i.e., $(\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}) \perp \text{Null}([\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})])$, it follows that $\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\tilde{\boldsymbol{\theta}} \neq \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\hat{\boldsymbol{\theta}}$, which in particular implies $\langle \nabla \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), \tilde{\boldsymbol{\theta}} \rangle \neq \langle \nabla \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), \hat{\boldsymbol{\theta}} \rangle$ for all $i = 1, \dots, n$. Consider $\bar{\boldsymbol{\theta}} = (\tilde{\boldsymbol{\theta}} + \hat{\boldsymbol{\theta}})/2$. By strict convexity on ℓ , we have

$$\begin{aligned} \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\bar{\boldsymbol{\theta}}, \mathbf{x}_i), y_i) &= \sum_{i=1}^n \ell\left(\mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i) + \left\langle \nabla \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), \frac{\tilde{\boldsymbol{\theta}} + \hat{\boldsymbol{\theta}}}{2} - \hat{\boldsymbol{\theta}} \right\rangle, y_i\right) \\ &= \sum_{i=1}^n \ell\left(\frac{\mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i) + \langle \nabla \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), \tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}} \rangle}{2} + \frac{\mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i) + \langle \nabla \mathbf{f}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), \hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}} \rangle}{2}, y_i\right) \\ &< \frac{1}{2} \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\tilde{\boldsymbol{\theta}}, \mathbf{x}_i), y_i) + \frac{1}{2} \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i), y_i) \\ &= \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\bar{\boldsymbol{\theta}}, \mathbf{x}_i), y_i), \end{aligned}$$

which is a contradiction. $\square$

Proof of Theorem 3.2.

• (Gradient Descent) Denoting

$$\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \triangleq [\mathbf{J}^\top(\hat{\boldsymbol{\theta}}, \mathbf{x}_1) \ \dots \ \mathbf{J}^\top(\hat{\boldsymbol{\theta}}, \mathbf{x}_n)]^\top \in \mathbb{R}^{nc \times p},$$

we can write

$$\mathbf{z} = \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{z} + \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z},$$

where $\hat{\boldsymbol{\theta}}$ represents the parameters around which the linear model $\tilde{\mathbf{f}}$ is defined in (2). The first iteration of gradient descent, initialized at $\boldsymbol{\theta}^{(0)} = \mathbf{z}$, is given by

$$\begin{aligned} \boldsymbol{\theta}^{(1)} &= \mathbf{z} - \alpha \sum_{i=1}^n \frac{\partial \tilde{\mathbf{f}}}{\partial \boldsymbol{\theta}}(\mathbf{z}, \mathbf{x}_i) \nabla \ell(\tilde{\mathbf{f}}(\mathbf{z}, \mathbf{x}_i), y_i) \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z} + \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{z} - \alpha \sum_{i=1}^n \frac{\partial \tilde{\mathbf{f}}}{\partial \boldsymbol{\theta}}(\mathbf{z}, \mathbf{x}_i) \nabla \ell(\tilde{\mathbf{f}}(\mathbf{z}, \mathbf{x}_i), y_i) \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z} + \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{z} - \alpha \sum_{i=1}^n \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i)\right]^\top \nabla \ell(\tilde{\mathbf{f}}(\mathbf{z}, \mathbf{x}_i), y_i) \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z} + \mathbf{v}^{(1)}, \end{aligned}$$

where $\mathbf{v}^{(1)} \in \text{Range}\left(\left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right]^\top\right)$. The next iteration is similarly given by

$$\begin{aligned} \boldsymbol{\theta}^{(2)} &= \boldsymbol{\theta}^{(1)} - \alpha \sum_{i=1}^n \frac{\partial \tilde{\mathbf{f}}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(1)}, \mathbf{x}_i) \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}^{(1)}, \mathbf{x}_i), y_i) \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z} + \mathbf{v}^{(1)} - \alpha \sum_{i=1}^n \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i)\right]^\top \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}^{(1)}, \mathbf{x}_i), y_i) \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X})\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})\right)\mathbf{z} + \mathbf{v}^{(1)} + \mathbf{v}^{(2)}, \end{aligned}$$

where again $\mathbf{v}^{(2)} \in \text{Range} \left(\left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right]^\top \right)$. Generalizing to the n th iteration,

$$\boldsymbol{\theta}^{(n)} = \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X}) \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right) \mathbf{z} + \sum_{i=1}^n \mathbf{v}^{(i)}.$$

Hence, by the assumption on ℓ , as long as an adaptive learning rate is chosen appropriately according to Malitsky & Mishchenko (2020), GD must converge to a solution of the form

$$\begin{aligned} \boldsymbol{\theta}^* &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X}) \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right) \mathbf{z} + \sum_{i=1}^{\infty} \mathbf{v}^{(i)} \\ &= \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X}) \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right) \mathbf{z} + \mathbf{v}, \end{aligned}$$

where $\mathbf{v} \in \text{Range} \left(\left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right]^\top \right)$. In particular, for $\mathbf{z} = \mathbf{0}$, by Lemma 3.1, we must have that $\mathbf{v} = \boldsymbol{\theta}^\ddagger$, where $\boldsymbol{\theta}^\ddagger$ is the solution to (5). Therefore,

$$\boldsymbol{\theta}^* = \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X}) \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right) \mathbf{z} + \boldsymbol{\theta}^\ddagger.$$

- (Stochastic Gradient Descent) Using a similar argument as above, it is easy to show that each iteration of the mini-batch SGD is of the form

$$\boldsymbol{\theta}^{(n)} \in \left(\mathbf{I} - \mathbf{J}^\dagger(\hat{\boldsymbol{\theta}}, \mathcal{X}) \mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right) \mathbf{z} + \text{Range} \left(\left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right]^\top \right).$$

Hence, it suffices to show that SGD converges almost surely. Defining

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \sum_{i=1}^n \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_i), \mathbf{y}_i), \quad \text{and} \quad \mathbf{g}(\boldsymbol{\theta}, \mathcal{X}) \triangleq \begin{bmatrix} \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_1), \mathbf{y}_1) \\ \vdots \\ \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_n), \mathbf{y}_n) \end{bmatrix} \in \mathbb{R}^{nc},$$

we write

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^n \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i) \right]^\top \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_i), \mathbf{y}_i) = \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right]^\top \mathbf{g}(\boldsymbol{\theta}, \mathcal{X}).$$

Let $\boldsymbol{\theta}^*$ be any solution to (3). The full row-rank assumption on $\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X})$ implies that we must have $\mathbf{g}(\boldsymbol{\theta}^*, \mathcal{X}) = \mathbf{0}$, i.e., $\nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathbf{x}_i), \mathbf{y}_i) = \mathbf{0}, i = 1, \dots, n$. By the μ -strong convexity assumption on $\ell(\cdot, \mathbf{y})$ with respect to its first argument, it is easy to see that $\mathcal{L}(\boldsymbol{\theta})$ satisfies the Polyak-Łojasiewicz inequality (Karimi et al., 2016) with constant $2\mu\lambda$ where

$$\lambda \triangleq \min_{i=1, \dots, n} \sigma_{\min}^2(\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i)),$$

and $\sigma_{\min}(\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i))$ is the smallest non-zero singular value of $\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i)$. Indeed, let $\boldsymbol{\theta}^*$ be any solution to (3). From μ -strong convexity of $\ell(\cdot, \mathbf{y})$ with respect to its first argument, for any $\boldsymbol{\theta}$, we have

$$\begin{aligned} \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_i), \mathbf{y}_i) - \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}^*, \mathbf{x}_i), \mathbf{y}_i) &\leq \frac{1}{2\mu} \left\| \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_i), \mathbf{y}_i) \right\|^2 \\ &\leq \frac{1}{2\mu\sigma_{\min}(\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i))} \left\| \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathbf{x}_i) \right]^\top \nabla \ell(\tilde{\mathbf{f}}(\boldsymbol{\theta}, \mathbf{x}_i), \mathbf{y}_i) \right\|^2, \end{aligned}$$

which implies

$$\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^*) \leq \frac{1}{2\mu\lambda} \left\| \left[\mathbf{J}(\hat{\boldsymbol{\theta}}, \mathcal{X}) \right]^\top \mathbf{g}(\boldsymbol{\theta}, \mathcal{X}) \right\|^2 = \frac{1}{2\mu\lambda} \left\| \nabla \mathcal{L}(\boldsymbol{\theta}) \right\|^2.$$

By the smoothness assumption on each $\ell(\tilde{\mathbf{f}}(\cdot, \mathbf{x}_i), \mathbf{y}_i)$ as well as the interpolating property of $\boldsymbol{\theta}^*$, Bassily et al. (2018, Theorem 1) implies that the mini-batch SGD with small enough step size η has an exponential convergence rate as

$$\mathbb{E}\mathcal{L}(\boldsymbol{\theta}^{(k)}) \leq (1 - \rho)^k \mathcal{L}(\boldsymbol{\theta}^{(0)}),$$

for some constant $0 < \rho < 1$. This in particular implies that for any $\epsilon > 0$,

$$\sum_{k=1}^{\infty} \mathbb{P}(\mathcal{L}(\boldsymbol{\theta}^{(k)}) > \epsilon) \leq \sum_{k=1}^{\infty} \frac{\mathbb{E}\mathcal{L}(\boldsymbol{\theta}^{(k)})}{\epsilon} \leq \frac{\mathcal{L}(\boldsymbol{\theta}^{(0)})}{\epsilon} \sum_{k=1}^{\infty} (1 - \rho)^k = \frac{\mathcal{L}(\boldsymbol{\theta}^{(0)})}{\epsilon \rho} < \infty.$$

Now, the Borel-Cantelli lemma gives $\mathcal{L}(\boldsymbol{\theta}^{(k)}) \rightarrow 0$, almost surely.

□

B Related Works: Further Details and Discussions

NUQLS shares notable similarities with, and exhibits distinct differences from, several prior works, which are discussed in-depth below.

B.1 Linearized Laplace Approximation (LLA) Framework.

The popular LLA framework (Khan et al., 2019; Foong et al., 2019; Immer et al., 2021; Daxberger et al., 2021) shares a close connection with NUQLS, as both methods fundamentally rely on linearizing the network. However, a subtle yet significant distinction lies in their constructions: LLA begins by obtaining a proper distribution over the parameters and then draws parameter samples from it, while NUQLS bypasses this step and directly targets an approximation of the posterior distribution of the neural network.

As a direct consequence of this, in overparameterized settings, where the Hessian (or its Generalized Gauss-Newton approximation) is not positive definite, the LLA framework necessitates imposing an appropriate prior over the parameters to avoid degeneracy. In sharp contrast, NUQLS directly generates samples from the predictive distribution without introducing any artificial prior, thereby avoiding potential biases that such priors might impose on the covariance structure of the outputs and eliminating the need for additional hyperparameters. Consequently, as the LLA framework corresponds to a Bayesian generalized linear model (GLM), the weight-space vs. function-space duality in GLMs implies that its predictive distribution corresponds to a noisy GP with an NTK kernel. On the other hand, NUQLS leads to a noise-free GP. In interpolating regimes, where the model perfectly fits the data, the noise-free setting of NUQLS appears to be more suitable (Hodgkinson et al., 2023).

Another important consequence of this distinction arises in classification tasks. Due to the Laplace approximation, the LLA framework produces an independent GP for each output of the linearized model. In contrast, NUQLS captures the covariance between outputs, offering a more comprehensive representation of the predictive distribution.

Finally, for regression tasks, NUQLS offers additional flexibility by allowing the variance to be scaled post-hoc by a factor γ . This enables efficient hyperparameter tuning on a validation set without the need for retraining the model or optimizing a marginal likelihood—a level of flexibility not available in the LLA framework.

NUQLS can be seen as an extension of the “sample-then-optimize” framework for posterior sampling of large, finite-width NNs (Matthews et al., 2017). In this context, the work of Antorán et al. (2022), henceforth referred to as Sampling-LLA, enables drawing samples from the posterior distribution of the LLA in a manner analogous to Algorithm 1. In this approach, a series of regularized least-squares regression problems are constructed, and the collection of their solutions is shown to be distributed according to the LLA posterior. An EM algorithm is then employed for hyperparameter tuning. In addition to the fundamental differences between NUQLS and LLA-inspired methods mentioned earlier, Sampling-LLA has notable distinctions from Algorithm 1. First, the objective functions in Sampling-LLA have non-trivial minimum values. As a result, the convergence of SGD for such problems necessitates either a diminishing learning rate (Bubeck et al., 2015), which slows down convergence, or the adoption of variance reduction techniques (Roux et al., 2012; Shalev-Shwartz &

Zhang, 2013; Johnson & Zhang, 2013), which can introduce additional computational and memory overhead. By contrast, in overparameterized settings and under the assumptions of Theorem 3.2, the optimization problem in (3) allows interpolation. Consequently, SGD can employ a constant step size for convergence, improving optimization efficiency (Garrigos & Gower, 2023). Second, the inherent properties of the LLA framework, which require a positive definite Hessian or its approximation, necessitate regularizing the least-squares term in the subproblem of Sampling-LLA. This results in a strongly convex problem with a unique solution. Consequently, to generate a collection of solutions, Sampling-LLA constructs a random set of such subproblems, each involving fitting the linearized network to random outputs. These random outputs are sampled from a zero-mean Gaussian, with covariance given by the Hessian of the loss function, evaluated on the data. In contrast, the subproblem of NUQLS, i.e., (3), involves directly fitting the training data, and the ensemble of solutions is constructed as a result of random initialization of the optimization algorithm. Hence, the uncertainty captured by NUQLS arises naturally from the variance of solutions in the overparameterized regime, without the need for additional regularization or artificially constructed subproblems.

While the Sampling-LLA method enhances the scalability of LLA, the competing method Variational LLA (VaLLA) (Ortega et al., 2023) offers comparable or superior UQ performance while significantly reducing computation time. VaLLA achieves this by computing the LLA predictive distribution using a variational sparse GP with an NTK. Another competing LLA extension is Accelerated LLA (ELLA), which uses a Nyström approximation of the functional LLA covariance matrix (Deng et al., 2022), and seems to attain similar performance to VaLLA, again at a reduced cost compared to Sampling-LLA. We compare the performance of NUQLS to Sampling-LLA, VaLLA and ELLA in Appendix H.3.

Note that work by (Miani et al., 2024b) approximates an LLA by taking the covariance as the projection onto the null space of the GGN matrix, in order to compute a posterior that retains the same performance as the original network on training data. To compute samples, this work uses *alternating projections*.

B.2 SNGP

We briefly describe the Spectral-Normalized Neural Gaussian Process (SNGP). SNGP lies at the intersection of feature-space methods and Bayesian methods, and similarly to NUQLS combines a GP with a NN. Specifically, SNGP adds a weight normalization step during training, and then replaces the output layer of NN with a GP that takes the feature extractor of the network as an input. While this method is also not post-hoc, and is not strictly a Bayesian method, we still compare SNGP with NUQLS in Figure 6 and Table 11, where we observe superior performance of NUQLS.

B.3 Ensemble Framework.

In Lee et al. (2019), infinitely wide neural networks were shown to follow a GP distribution; however, this GP did not correspond to a true predictive distribution. Building on this, He et al. (2020) introduced a random, untrainable function to an infinitely wide neural network, deriving a GP predictive distribution using the infinite-width NTK. An ensemble of these modified NNs was then interpreted as samples from the GP’s predictive posterior. In contrast, our method demonstrates that trained, finite-width, unmodified linearized networks are inherently samples from a GP with an NTK kernel. While their method, Bayesian Deep Ensembles (BDE), shares some conceptual similarities with ours, we omit it from our main experiments for several reasons. Firstly, the posterior analysis of BDE is valid only in the limit of large model sizes. For smaller datasets and models, the infinite-width NTK differs greatly from the empirical NTK (see Fort et al. (2020)). We can see that in this regime, in both Figure 4 and Table 8, that NUQLS outperforms BDE. Secondly, the method is computationally more expensive than DE, due to the need to compute the untrainable function ($\delta(\cdot)$) and tune the scaling hyperparameter for classification. This contradicts our goal to provide a UQ method that is computationally more efficient than the state-of-the-art (SOTA) method DE while maintaining competitive performance. Finally, for larger model sizes, we provide comparison of NUQLS to BDE in Figure 6 and Table 11, where we observe that NUQLS outperforms BDE.

In Madras et al. (2019), the authors propose a method called “local ensembles”, which perturbs the parameters of a trained network along directions of small loss curvature to create an ensemble of

nearly loss-invariant networks. The uncertainty of the original network is then quantified as the standard deviation across predictions in the ensemble. Building on this approach, Miani et al. (2024a), in a method called Sketched Lanczos Uncertainty (SLU), use the GGN approximation of the Hessian of the loss to identify these directions and introduce a sketched Lanczos algorithm (Meurant, 2006) to efficiently compute them. We compare the performance of NUQLS to SLU in Appendix H.2.

While similar, there are key differences between local ensembles and NUQLS. Local ensembles form a subspace of networks that attain *similar* loss values, allowing for directions with small but non-zero curvature, potentially encompassing more directions than those with exactly zero curvature. In contrast, when a solution to the linear optimization problem exists, NUQLS creates an ensemble of networks that all attain *exactly* the same loss. Additionally, NUQLS relies on first-order information to construct this ensemble, whereas local ensembles depend on second-order information.

While the zero-curvature directions of the GGN approximation correspond to the Jacobian’s null space, the local ensembles method also includes directions with small but non-zero curvature. This inclusion introduces a notable distinction between the ensembles generated by local ensembles and those formed by NUQLS. Finally, while local ensembles employ low-rank approximations to compute directions efficiently, such approximations may inadequately represent the Hessian and fail to accurately capture the true curvature structure, as highlighted in Xie et al. (2022).

We also give mention to methods that seek to make DE more efficient, such as BatchEnsemble (BE) (Wen et al., 2020) and SnapshotEnsemble (Havasi et al., 2020). We note that these methods are yet to surpass DE as the SOTA in the literature. However, we compare NUQLS against BE in Figure 6 and Table 11, where we see that NUQLS shows superior performance, whilst remaining as scalable as BE. Further, BE requires modification to the structure of the network, and hence is not post-hoc.

B.4 Neural Tangent Kernel Methods

The Procedural-Noise-Correcting (PNC) method of Huang et al. (2023) employs the limiting NTK for infinite-width networks to characterize the noise in the optimization process of training a neural network. It uses this to provide a frequentist confidence interval around test predictions of the neural network, and as it arises from the frequentist framework, PNC recovers statistical guarantees. In comparison, NUQLS does not rely on the limiting regime of the NTK, but rather employs the feature extraction of the empirical NTK to quantify uncertainty using a Bayesian framework. In Appendix H.1 we compare the performance of NUQLS vs. PNC. Unsurprisingly, given their constructions, we see that PNC outperforms NUQLS when the width of a network is extreme, while for finite-width networks that are fully trained, NUQLS outperforms PNC. This experiment illustrates that the methods are actually complementary, as they both excel in different settings.

C Uncertainty vs. Prediction

Here we discuss the use of common metrics to measure UQ ability in the BDL community. Generally, NLL, ECE and AUCROC (or the entropic version OODAUC) are used to measure the ability of a method to correctly quantify the uncertainty in a neural network. However, we argue that the use of these metrics for UQ is ill-advised, due to either a misalignment between the metric and the goal of UQ, or due to flawed measurements of uncertainty implicit in the metric. Specifically, we argue that variance over the softmax predictions *is* the uncertainty in the network, in the same way that the variance over probabilities *is* the uncertainty in a Dirichlet model. In contrast, NLL and AUCROC (OODAUC) are metrics for *prediction*, while ECE is a metric based on a flawed surrogate for uncertainty, calibration.

Prediction While we will argue that uncertainty quantification *must* involve a computed variance term, and that common UQ metrics are either actually prediction metrics, or are based on a *prediction* of uncertainty, this discussion also begs the question: what about the purported benefits of BDL to prediction ability? As has been argued in Abe et al. (2022), it is often more economical to simply train a larger capacity model, than to use BDL to improve prediction quality. Hence, we restrict our main focus of NUQLS to its UQ ability. However, for completeness, we also evaluate the novel predictor of NUQLS; these results can be found in the Appendix G.

Uncertainty Uncertainty can be divided into two categories: *aleatoric* uncertainty, or data uncertainty, and *epistemic* uncertainty, or model uncertainty. *Aleatoric* uncertainty arises from intrinsic randomness or noise in the data, and can often not be controlled. In contrast, *epistemic* uncertainty arises from having many potential models that may fit the data, and the uncertainty in not knowing which model is ‘correct’. As this arises from a lack of knowledge of the original data-mapping, uncertainty in this manner can often be reduced through increased training data, using a larger model, training for longer etc. There are numerous benefits to untangling the aleatoric and epistemic uncertainty (Mukhoti et al., 2021; Huseljic et al., 2021; Chan et al., 2024). We can use the framework of a Dirichlet model, the conjugate prior of a Categorical distribution, to understand how to evaluate these uncertainties in the context of K -class classification. Assume there is some subspace $\mathcal{M}$ of the parameter space $\Theta \in \mathbb{R}^p$, such that our network $\mathbf{f}(\cdot; \theta^*)$ attains ‘good’ test performance for any $\theta^* \in \mathcal{M}$. If it is common across parameter values in $\mathcal{M}$ that $\mathbf{f}(\mathbf{x}; \theta^*)$ has high entropy for some $\mathbf{x}$, i.e. $\mathbf{f}(\mathbf{x}; \theta^*)$ outputs a prediction close to the uniform distribution with high probability across $\theta^* \in \mathcal{M}$, then we are sure that the outcome is unpredictable in $\mathbf{x}$, i.e. that there is high aleatoric uncertainty. This equates to a Dirichlet model with parameters $\alpha = (\alpha_1, \dots, \alpha_K)$ where α is uniform and of large magnitude. In contrast, assume that over all parameter values in $\mathcal{M}$, the softmax outputs of $\mathbf{f}(\mathbf{x}; \theta^*)$ for a given $\mathbf{x}$ have very large variance. This means that no model can agree on the ‘correct’ data-mapping, and hence there is very large epistemic uncertainty. This is analogous to when the parameters α of a Dirichlet model are small in magnitude. Our method, NUQLS, measures the variance over softmax outputs for high-performing parameters $\theta^* \in \mathcal{M}$. Hence, we are measuring the epistemic uncertainty of the neural network.

NLL Due to its probabilistic framework, NLL is often used as a metric for UQ ability. For K -class classification, up to a constant the NLL is just the cross-entropy loss

$$\text{NLL}(\theta) = - \sum_{i=1}^n \sum_{k=1}^K y_{i,k} \log \mu_{i,k},$$

where $y_{i,k}$ corresponds to the k -th element of the one-hot encoding of the i -th label, and $\mu_{i,k} = \text{softmax}(\mathbf{f}(\mathbf{x}_i; \theta))_k$. This loss is minimised as $\text{softmax}(\mathbf{f}(\mathbf{x}_i; \theta))_k \rightarrow \mathbb{1}_{c_i}$, where c_i is the correct label for input $\mathbf{x}_i$. While a lower NLL is generally considered to show better UQ ability, we can see that this metric is instead a smooth, probabilistic measurement of accuracy, where proximity of the predictor to the identity function is rewarded with lower NLL. *This metric does not measure uncertainty quantification ability.*

ECE A common surrogate for uncertainty in neural networks is the predicted softmax probabilities. While neural networks generally output over-confident probabilities, many methods, such as temperature scaling and many Bayesian methods, attempt to create well-calibrated predictors. A well-calibrated output is one where the predicted softmax probabilities accurately convey the forecasted probability of seeing each of the K -classes. For example, of all test points where a network has a maximum predicted probability of 0.9, we would hope that the network is 90% accurate on these test points. Once a predictor is well-calibrated, the softmax probabilities are then seen to quantify uncertainty. However, there are issues with this:

1. Firstly, a point prediction from a network cannot be interpreted as the uncertainty in the network; uncertainty quantities must accompany a network prediction, for example with a variance term. At best, network confidence can be seen to *predict* the uncertainty.
2. Secondly, if softmax probability does in fact accurately predict the uncertainty, is this uncertainty due to data uncertainty, or model uncertainty? If a model accurately predicts that there is a 90% chance that class c is correct, does that uncertainty arise due to noise in the data, or some misspecification in the labeling process? Or is it due to the fact that the input is OoD, and thus the model is unsure of itself? Hence, we still require a variance term to ascertain what the epistemic uncertainty is.
3. Finally, calibration is only well-defined for ID test points. To see this, take a network trained on MNIST, and then tested on FashionMNIST. Any network prediction is meaningless on this test set, as the model has been trained for integer labels. The prediction that will minimize the calibration error will be the uniform prediction i.e. the prediction is $[0.1, \dots, 0.1] \in \mathbb{R}^{10}$ for all points in the OoD test set. However, the accuracy on OoD test points *should be* 0. Instead the predicted accuracy is 0.1, which is what arises from randomly picking one class for each OoD test

point. Therefore, the predicted probabilities cannot be directly interpreted as forecasted probability of accuracy for OoD points.

In practice, calibration is tested using ECE. This metric bins predictions, and then computes the average confidence in each bin. This is compared with the average predicted accuracy, and an l_2 norm is taken over the differences. As was noted in Nixon et al. (2019), there are several issues with ECE. For examples, it is reductive in a multi-class setting, where the bottom $K - 1$ probabilities do not contribute meaningfully to the error, and it suffers from the inherent sharpness of neural networks, where the majority of maximum probabilities tend to be very close to 1, and thus this confidence range is highly over-represented. So we struggle to even tell when a method is well-calibrated.

This issue with ECE can be summarised as follows: confidence is a flawed *estimate* of the total uncertainty; further, it is hard to tell *when* a method is well-calibrated. It is thus evident that it is much better to *directly* model the uncertainty in the model, by calculating the variance over the softmax predictions. Once equipped with this model uncertainty, entropy (or network confidence) can be better trusted to inform of the aleatoric uncertainty.

AUCROC The final metric commonly used for UQ is AUCROC (OODAUC). This metric uses the top softmax prediction (entropy) as a predictor, to attempt to detect whether a point is ID or OoD. While this is a very useful task, it is still a metric of *prediction*, as only the mean predictor is used to compute this value. As we have argued, UQ requires a variance term. Miani et al. (2024a) in fact use the variance over *logits* to compute the AUCROC score, and report this score for their method SLU in comparison to other leading UQ methods. In Appendix H.2 we show that in this metric, we outperform SLU.

VMSP To combat the issues of the previous metrics, we compare the performance of Bayesian methods using the variance of the maximum softmax predictor, or VMSP. We first provide an explicit definition of VMSP: for a Bayesian method, we generally have a mean predictor $\mu : \mathbb{R}^d \rightarrow \mathbb{R}^c$ and a covariance function $\Sigma : \mathbb{R}^d \rightarrow \mathbb{R}^{c \times c}$, for example the mean and covariance of the linearized ensemble in the case of NUQLS, that output in the probit space. To compute VMSP for a given test point $\mathbf{x}^*$, we first find $\hat{c} = \arg\max_k \mu(\mathbf{x}^*)_k$, where $\mu(\mathbf{x}^*)_k$ denotes the k -th output of $\mu(\mathbf{x}^*)$. That is, we find the class that the Bayesian method predicts, given $\mathbf{x}^*$. We then define $\text{VMSP} := \Sigma(\mathbf{x}^*)_{\hat{c}, \hat{c}} = \sigma^2(\mathbf{x}^*)_{\hat{c}}$, that is, the variance of this prediction.

We also introduce a pictorial method for evaluating the performance of Bayesian methods, using VMSP as the correct uncertainty measure. For a given dataset, we want a UQ method to provide low uncertainty for correctly predicted test points, and high uncertainty for incorrectly predicted or OOD test points. We then compare these distributions pictorially using a violin plot, and quantitatively using the median and skew values for the respective distributions. With the addition of a poorly-performing baseline model, we are able to easily compare the ability of UQ models to quantify uncertainty.

D Guarantees Against Mode Collapse

Given that all linearized models are initialized locally around the trained parameters, one may consider whether it is possible that all linear networks will converge to the same parameter solution, or mode of the linear loss. Fortunately, we are able to show that this will not occur, almost surely. To see this, we note that the solution to (3) for each linearized model is given by a unique row-space component (given our assumptions), plus a projection of the initialization $\mathbf{z}_i$ onto the null-space of the Jacobian. For mode-collapse, we would require the projection for two i.i.d initializations $\mathbf{z}_1, \mathbf{z}_2 \sim N(\mathbf{0}, \gamma^2 \mathbf{I})$ to be equal, or $(\mathbf{I} - \mathbf{J}_{\mathcal{X}}^\dagger \mathbf{J}_{\mathcal{X}})(\mathbf{z}_1 - \mathbf{z}_2) = \mathbf{0}$, where $\mathbf{z}_1 - \mathbf{z}_2 \neq \mathbf{0}$ with probability one. Hence, $\mathbf{z}_1 - \mathbf{z}_2 = \mathbf{J}_{\mathcal{X}}^\dagger \mathbf{J}_{\mathcal{X}}(\mathbf{z}_1 - \mathbf{z}_2)$, and thus $\mathbf{z}_1 - \mathbf{z}_2 \in \text{Range}(\mathbf{J}_{\mathcal{X}}^T)$, i.e. $\mathbf{z}_1, \mathbf{z}_2 \in \text{Range}(\mathbf{J}_{\mathcal{X}}^T)$. However, $\text{Range}(\mathbf{J}_{\mathcal{X}}^T)$ is a low-dimensional subspace of $\mathbb{R}^p$, and as we are drawing i.i.d. from a p -dimensional normal distribution (i.e. not degenerate), the probability of this event occurring is 0. Hence, we do not need to be concerned with all models converging to the same behaviour.

E Computational Cost

We now provide examination of the computational cost of NUQLS. We compare the computational complexity for an epoch of training for the neural network $f_\theta(x)$, for batch $x \in \mathbb{R}^{d \times n}$, and an

epoch of training for a single linear network $\hat{f}_\theta(x)$. We take approximations on the computational complexity of both forward-mode AD and backward-mode AD from Blondel & Roulet (2024, Chapter 8). Specifically, we take $[fp]$ as the computational complexity for evaluating $f_\theta(x)$. We note from Blondel & Roulet (2024, Chapter 8) that both a JVP and a VJP cost roughly $2 \times$ a forward pass in computational complexity and memory. Further, both a JVP and a VJP return a function evaluation. Now, a standard epoch of training for the neural network involves a forward-pass to compute the error, and then a backward-pass (VJP), hence the complexity is approx. $3[fp]$. The linearised network involves a JVP (which includes a function evaluation) to form the linear network, and a VJP to compute the gradient. Hence, the complexity for an epoch of training for the linearized network is approx. $4[fp]$. So we observe that each epoch for a linearized network is only $4/3 \times$ as expensive as for a neural network. In regards memory, we see that the memory requirement for both the linearized network and the neural network are similar. However, we generally train all linearized networks in parallel. For computational complexity, this will incur some additional cost, though it will not be linear in number of networks, that is dependent upon the specific software and hardware. However, memory cost will scale linearly by number of networks, i.e. $2S \times M([fp])$, where $M([fp])$ is the memory cost for a forward-pass, and S is the number of linear networks. As an example, we employ a batch size of 56 for ImageNet on ResNet50 with 10 ensemble members when using an 80GB H100 GPU, due to the large parameter count and large number of classes. Note that in the case where the training set is small, it is beneficial to compute and save in memory the Jacobian of the NN evaluated on the entire training set. We can then train the linear networks very quickly. This contributes to the impressive run-times seen in Tables 1 and 9.

F Hyper-parameter Tuning

NUQLS contains several hyper-parameters: the number of linear networks to be trained, the number of epochs and learning rate of training, and the variance of initialisation, γ . In this section, we discuss strategies to select optimal hyper-parameters.

F.1 Regression

For regression, and with a sufficiently small learning rate, NUQLS samples from the distribution given in Remark 3.3. We can see that the variance of the predictions scales linearly with γ^2 . Hence, we use the following framework to tune γ :

1. To obtain θ_s in Algorithm 1, we initialise our parameters with a very small gamma, e.g. $\gamma = 0.01$. This enables (stochastic) gradient descent to converge quickly with a small learning rate.
2. We compute $\{\tilde{f}(\theta_s^*, \mathcal{X}_{\text{val}})\}_{s=1}^S$, where $\mathcal{X}_{\text{val}}$ is the inputs from a validation set. As per Remark 3.3, for each point $\mathbf{x} \in \mathcal{X}_{\text{val}}$, we compute the mean prediction as $\mu(x) = \text{SampleMean}\left(\{\tilde{f}(\theta_s^*, \mathbf{x})\}_{s=1}^S\right)$ and the variance as $\sigma_\gamma^2(x) = \text{SampleVariance}\left(\{\gamma \tilde{f}(\theta_s^*, \mathcal{X}_{\text{val}})\}_{s=1}^S\right)$ (note the scaling by γ for only the variance). We then use these values to compute the ECE across the validation dataset $\mathcal{D}_{\text{val}}$.
3. As coverage of a confidence interval scales linearly with the size of the given standard deviation, and our computed standard deviation scales linearly with γ , we find that the ECE is convex in γ . Hence, we employ the Ternary search method (see Algorithm 2) to find the value $\hat{\gamma}$ that minimizes this validation ECE.
4. For a test point $\mathbf{x}^*$, our mean prediction and variance is then $\mu(\mathbf{x}^*)$ and $\sigma_{\hat{\gamma}}^2(\mathbf{x}^*)$.

As can be seen in Table 1 and Table 9, this framework means that for regression our method is incredibly fast and computes well-calibrated variance values.

F.2 Classification

For uncertainty quantification performance, as measured by VMSP in Figure 3, we find that as long as γ is small, training SGD for a small amount of epochs generally gives small training loss, and hence provides good performance. This means that our method can be computed quite quickly in larger data/model settings.

Algorithm 2 Ternary Search

f : function to minimize, l : left boundary of search space, r : right boundary of search space, δ : tolerance, iter: iterations.

```
 $i = 0$   
while  $|l - r| \geq \delta$  and  $i < \text{iter}$  do  
   $l_{1/3} = l + (r - l)/3$   
   $r_{1/3} = r - (r - l)/3$   
  if  $f(l_{1/3}) < f(r_{1/3})$  then  
     $l = l_{1/3}$   
  else  
     $r = r_{1/3}$   
  end if  
end while  
return  $(l + r)/2$ 
```

Table 2: Image classification predictive performance, using LeNet5 on MNIST and FashionMNIST (FMNIST). Experiment was run 5 times with different random MAP initialisations to get standard deviation on metrics.

Datasets	Method	NLL ↓	ACC ↑	ECE ↓	OOD-AUC ↑	AUC-ROC ↑	Time (s)
MNIST	MAP	0.034±0.002	0.990±0.001	0.008±0.001	0.888±0.008	0.886±0.008	257
	NUQLS	0.035±0.002	0.989±0.001	0.003±0.000	0.930±0.026	0.928±0.026	106
	DE	0.034±0.004	0.991±0.000	0.011±0.004	0.932±0.009	0.928±0.009	2845
	MC-Dropout	0.044±0.002	0.989±0.000	0.017±0.01	0.873±0.032	0.871±0.031	533
	SWAG	0.029±0.003	0.991±0.000	0.004±0.002	0.902±0.008	0.900±0.008	489
	LLA*	0.034±0.002	0.990±0.001	0.008±0.001	0.888±0.008	0.886±0.008	45
	VaLLA	0.034±0.002	0.990±0.001	0.008±0.001	0.889±0.008	0.886±0.008	1583
FMNIST	MAP	0.298±0.007	0.891±0.3	0.006±0.001	0.840±0.022	0.804±0.021	158
	NUQLS	0.302±0.006	0.891±0.002	0.005±0.002	0.904±0.007	0.870±0.006	89
	DE	0.288±0.002	0.896±0.001	0.013±0.001	0.876±0.003	0.836±0.003	1587
	MC-Dropout	0.306±0.007	0.892±0.003	0.026±0.002	0.856±0.021	0.813±0.019	291
	SWAG	0.283±0.005	0.899±0.003	0.018±0.002	0.817±0.023	0.783±0.022	264
	LLA*	0.298±0.007	0.891±0.003	0.006±0.001	0.841±0.022	0.805±0.021	26
	VaLLA	0.298±0.007	0.891±0.003	0.007±0.001	0.841±0.022	0.805±0.021	1583

G Evaluation of Predictive Mean

While the main focus of NUQLS is the variance term, to compute the epistemic uncertainty of a NN, we would also like to demonstrate the predictive ability of our novel predictive mean, for completeness.

G.1 Image Classification - Predictive

We compare NUQLS against the MAP NN, DE, MC-Dropout, SWAG, LLA* and VaLLA, on the MNIST and FashionMNIST datasets, using the LeNet5 network. We compare test cross-entropy (NLL), test accuracy (ACC), ECE, and the AUC-ROC measurement for maximum softmax probability (AUC-ROC) and entropy (OOD-AUC) as the detector. The last two metrics evaluate a methods ability to detect out-of distribution points. We display the results in Table 2. We see that NUQLS performs the best in ECE, AUC-ROC and OOD-AUC, and is competitive in the other metrics, while having the second fastest wall-time.

H Further Experimental Results

H.1 Comparison to PNC using Confidence Intervals

We use this section to compare the difference in performance of NUQLS vs PNC. To test the difference between the two methods, we tested NUQLS on the confidence intervals problem from Huang et al. (2023). In this experiment, an MLP with a single hidden-layer is trained on n datapoints

Table 3: Evalaution of coverage (CR) and width (IW) of computed confidence intervals from toy problem, for a neural network with **extreme** width and **partial** training. A computed coverage exceeding the expected coverage is bolded. The mean prediction (MP) is also provided.

	PNC		MP	NUQLS		MP
	95%CI (CR/IW)	90%CI (CR/IW)		95%CI (CR/IW)	90%CI (CR/IW)	
$d = 2, n = 128$	0.98 /0.0437	0.95 /0.0323	0.1998	0.93/0.0357	0.92 /0.0299	0.2047
$d = 4, n = 256$	0.98 /0.0411	0.95 /0.0304	0.3991	0.92/0.0596	0.86/0.0500	0.4084

Table 4: Evalaution of coverage (CR) and width (IW) of computed confidence intervals from toy problem, for a neural network with **finite** width and **full** training. A computed coverage exceeding the expected coverage is bolded. The mean prediction (MP) is also provided.

	PNC		MP	NUQLS		MP
	95%CI (CR/IW)	90%CI (CR/IW)		95%CI (CR/IW)	90%CI (CR/IW)	
$d = 2, n = 128$	0.8/0.136	0.72/0.0105	0.2022	0.99 /0.0135	0.96 /0.0114	0.2012
$d = 4, n = 256$	0.96 /0.0437	0.90 /0.0336	0.4045	0.97 /0.0313	0.97 /0.0313	0.4030
$d = 8, n = 512$	0.88/0.0740	0.88/0.0568	0.8078	1.00 /0.0667	0.98 /0.0559	0.8050
$d = 16, n = 128$	0.8/0.1443	0.8/0.1108	1.6265	1.00 /0.1350	0.98 /0.1133	1.6121

from $U([0, 0.2]^d)$, where the target is $y = \sum_{i=1}^d \sin(x_i) + \epsilon$, and ϵ is a small noise term. The method is then asked to form a confidence interval around the prediction for $\mathbf{x} = (0.1, 0.1, \dots, 0.1)$. This setup is repeated several times, i.e. the network and training data are randomly initialized, and the coverage and average width of the confidence intervals are recorded, as well as the mean prediction. For more details, we refer the reader to Huang et al. (2023). When the width of the network is $32 \times n$, and the network is only partially trained with 80 epochs and a learning rate of 0.01, NUQLS performs poorly compared to PNC, as can be seen in Table 3. Note that bolded numbers indicate intervals that have reached or exceeded the expected coverage. Further, smaller width intervals are preferred.

We note that scaling the width of a network by $32 \times n$ is rare in practice, and that such a wide network is difficult to train. If we instead form an MLP with width equal to the number of training points, and increase epochs to 100 and learning rate to 0.5, so that the network is properly trained, NUQLS far outperforms PNC, as can be seen in Table 4. We note that for these experiments we tuned the γ hyper-parameter for NUQLS on a small validation set. We conclude that NUQLS and PNC are in fact complementary methods. For infinite-width networks near initialization, PNC performs well while NUQLS struggles. Conversely, for finite-width networks trained to a minimum, NUQLS excels while PNC performs poorly. We believe the latter regime is more representative of practical scenarios, where NUQLS offers significantly better performance.

H.2 Comparison to SLU

Here we compare the performance of our method against the competing SLU method. See Appendix B for a discussion of the differences between NUQLS and SLU. Due to the extensive experimental details given in (Miani et al., 2024a), we run certain experiments from this work and report the performance of NUQLS against the SLU results found in (Miani et al., 2024a). In Table 5, we compare NUQLS against SLU on OoD detection using the AUC-ROC metric, on a smaller single-layer MLP and a larger LeNet model. The MLP is trained on the MNIST dataset, while the LeNet model is trained on the FashionMNIST dataset. For MNIST, the OoD datasets are FashionMNIST, KMNIST, and a Rotated MNIST dataset, where for each experiment run, we compute the average AUC-ROC score over a range of rotation angles (15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180). For FashionMNIST, the OoD datasets are MNIST, and the average Rotated FashionMNIST dataset. The AUC-ROC metric was calculated using the **variance of the logits**, summed over the classes for each test point, as a score. We observe that NUQLS has a better AUC-ROC value over all OoD datasets, often by a significant margin.

H.3 Comparison to Sampling-LLA, VaLLA and ELLA

We now compare NUQLS against Sampling-LLA, VaLLA and ELLA. Due to issues with convergence, performance, memory usage and package compatibility when either running source

Table 5: Comparing performance of NUQLS against SLU method. Metric given is the AUC-ROC, computed using the variance of the logits, summed over the classes for each test point, as a score. AUC-ROC measures ability of a method to differentiate between ID and OoD points. We see that NUQLS out-performs SLU in these experiments.

Model ID Data OoD Data	MLP $p = 15k$ MNIST vs			LeNet $p = 40k$ FashionMNIST	
	FashionMNIST	KMNIST	Rotation (avg)	MNIST	Rotation (avg)
SLU	0.26 ± 0.02	0.42 ± 0.04	0.59 ± 0.02	0.94 ± 0.01	0.74 ± 0.03
NUQLS	0.67 ± 0.07	0.79 ± 0.02	0.74 ± 0.01	0.95 ± 0.02	0.91 ± 0.01

Table 6: Comparing performance of NUQLS against Sampling-LLA, VaLLA and ELLA on MNIST, trained used a 2-layer MLP with 200 hidden units in each layer, and tanh activation. Original results taken from (Ortega et al., 2023).

Model	ACC	NLL	ECE	BRIER	OOD-AUC
ELLA	97.6	0.076	0.008	0.036	0.905
Sampled LLA	97.6	0.087	0.026	0.040	0.954
VaLLA 100	97.7	0.076	0.010	0.036	0.916
VaLLA 200	97.7	0.075	0.010	0.035	0.921
NUQLS	98.0 ± 0.1	0.065 ± 0.003	0.005 ± 0.001	0.031 ± 0.001	0.953 ± 0.006

code or implementing methods from instructions given in (Antorán et al., 2022) and (Ortega et al., 2023), we instead compare NUQLS against the results for Sampling-LLA, VaLLA and ELLA taken verbatim from (Ortega et al., 2023, Figure 3). We train a 2-layer MLP, with 200 hidden-units in each layer, on the MNIST dataset, according to the experimental details given in (Ortega et al., 2023). We then compare the accuracy, NLL, ECE, Brier score and the OOD-AUC metric of NUQLS against those reported for Sampling-LLA, VaLLA and ELLA. The results are shown in Table 6. We display the mean and standard deviation for NUQLS; as is quoted in (Ortega et al., 2023), the standard deviation for the other methods was below 10^{-4} in magnitude, and was thus omitted. We see that NUQLS outperforms Sampling-LLA, VaLLA and ELLA in accuracy, NLL, ECE, and Brier score, and is within one-sixth of a standard deviation of the leading OOD-AUC value, attained by Sampling-LLA. While we cannot directly comment on differences in computation time between our method and these LLA extensions, we were able to run the source code for VaLLA for the experiments in Table 2, where NUQLS was an order-of-magnitude faster than VaLLA in wall-time. In (Ortega et al., 2023, Figure 3 (right)), we also observe that VaLLA is an order-of-magnitude faster than both Sampling-LLA and ELLA. We also compare with (Ortega et al., 2023, Table 1), where a ResNet20 and ResNet34 model is trained on CIFAR-10. The results are displayed in Table 7. We see that NUQLS is competitive with all other methods in this setting.

Table 7: Comparing performance of NUQLS against Sampling-LLA, VaLLA and ELLA on CIFAR-10, with ResNet20 and ResNet32. Original results taken from (Ortega et al., 2023). Purple figures correspond to the top result, while blue figures are the second-best result.

Model	ResNet20			ResNet32		
	ACC	NLL	ECE	ACC	NLL	ECE
ELLA	92.5	0.233	0.009	93.5	0.215	0.008
Sampled LLA	92.5	0.231	0.006	93.5	0.217	0.008
VaLLA	92.6	0.228	0.007	93.5	0.211	0.007
NUQLS	92.5	0.228	0.006	93.4	0.215	0.007

Table 8: Comparing performance of NUQLS and BDE on UCI regression tasks. We see that NUQLS outperforms BDE on all tasks.

Dataset	Method	RMSE ↓	NLL ↓	ECE ↓
Energy	BDE	0.416 ± 0.039	-0.125 ± 0.212	0.008 ± 0.005
	NUQLS	0.047 ± 0.006	-2.400 ± 0.209	0.002 ± 0.002
Concrete	BDE	0.714 ± 0.054	1.563 ± 0.449	0.063 ± 0.012
	NUQLS	0.330 ± 0.047	-0.316 ± 0.501	0.003 ± 0.001
Kin8nm	BDE	0.851 ± 0.037	1.383 ± 0.582	0.042 ± 0.012
	NUQLS	0.252 ± 0.005	-0.796 ± 0.025	0.000 ± 0.000

H.4 Comparison to BDE

Figure 4 displays the performance of BDE on the toy regression problem from Figure 1. We also compare BDE against NUQLS on several UCI regression tasks in Table 8.

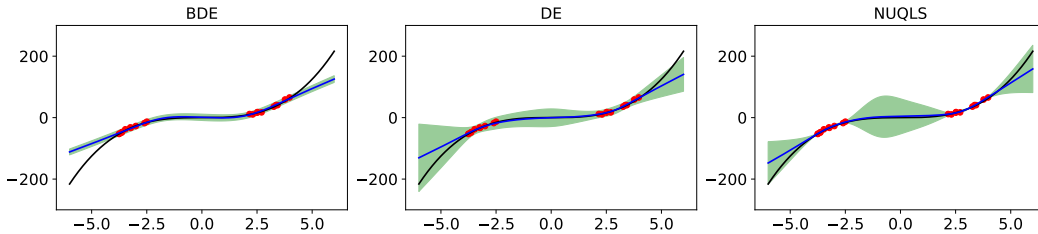


Figure 4: Comparison of BDE, DE and NUQLS on the toy regression problem from Figure 1. We can see that the uncertainty of the BDE method is quite small.

H.5 UCI Regression

We present the results for select UCI regression datasets in Table 1 in the main body; we present results for several more datasets in Table 9.

H.6 eNUQLS

In Figure 5 we demonstrate the performance of an ensembled version of NUQLS, eNUQLS. This method is similar to a Mixture of Laplace Approximations (Eschenhagen et al., 2021). We observe excellent separation of the variances between correct and incorrect/OoD test groups for eNUQLS, especially for CIFAR-10 on ResNet9. Note that there is significant cost to ensembling our method, and we provide this figure simply to illustrate performance capacity.

H.7 Additional VMSP Results

We use this section to expand the empirical evaluation of NUQLS. In the top left of Figure 6, we observe that on a ResNet50 model trained on SVHN, with CIFAR-10 as the OoD test data, NUQLS outperforms all other methods. In the top right of Figure 6, we evaluated NUQLS on the ImageNet dataset in order to display the scalability of our method to larger datasets. We employed the pre-trained weights for a ResNet50, as found on *torch.hub*, and used the ImageNet-o dataset as our OOD test set (Hendrycks et al., 2021). Due to resource budget constraints, we were only able to compare NUQLS against a baseline method, though future research could include a comparison of other methods against NUQLS, using VMSP, on ImageNet. As can be seen, we see excellent separation between correct and incorrect predictions, while only adequate separation between correct and OOD points. We note from Bitterwolf et al. (2023) that approximately 21% of the images in ImageNet-o are actually in-distribution; hence, we see that NUQLS correctly quantifies the variance of these test sets. We also provide comparison of NUQLS with additional competing methods. In the bottom left of Figure 6, we compare NUQLS against the BE method. Note that BE requires modification to the structure of a neural network; hence, we compare BE on a modified WideResNet-34-1, of which an implementation of the correct modifications existed (Franchi et al., 2023), trained on FashionMNIST.

Table 9: Comparing performance of NUQLS, DE, LLA and SWAG on UCI regression tasks. NUQLS performs as well as or better than all other methods, while showing a speed up over other methods; this speed up increases with the size of the datasets. Note that for the Song dataset, the LLA method uses a diagonal covariance structure due to memory constraints: this is denoted as LLA-D.

Dataset	Method	RMSE ↓	NLL ↓	ECE ↓	Time(s)
Concrete	NUQLS	0.330 ±0.047	-0.316 ±0.501	0.003 ±0.001	7.339 (0.185)
	DE	0.379±0.019	-0.574 ±0.098	0.002 ±0.002	29.047
	LLA	0.333 ±0.050	-0.294 ±0.479	0.003 ±0.002	7.451(0.297)
	SWAG	0.334 ±0.050	-0.562 ±0.224	0.009±0.006	43.416 (36.262)
Naval	NUQLS	0.049 ±0.012	-2.546 ±0.134	0.002 ±0.002	11.360 (0.295)
	DE	0.076±0.006	-1.761±0.250	0.093±0.040	96.570
	LLA	0.070±0.022	25.292±17.570	0.192±0.029	140.724 (129.659)
	SWAG	1.130±1.500	0.303±1.091	0.084±0.022	103.727 (92.662)
CCPP	NUQLS	0.244±0.008	-0.885±0.020	0.000 ±0.000	6.698 (0.174)
	DE	0.227 ±0.006	-1.009 ±0.041	0.002 ±0.003	79.791
	LLA	0.243±0.007	29.420±4.565	0.163±0.008	38.572 (32.048)
	SWAG	0.252±0.012	-0.849±0.038	0.001 ±0.002	73.357 (66.833)
Wine	NUQLS	0.789 ±0.042	0.284 ±0.066	0.001 ±0.000	1.164 (0.115)
	DE	0.789 ±0.041	0.320 ±0.109	0.001 ±0.001	13.241
	LLA	0.792 ±0.041	1.012±0.182	0.009±0.004	1.389 (0.340)
	SWAG	0.798 ±0.038	0.367 ±0.103	0.005±0.003	12.856 (11.807)
Yacht	NUQLS	0.042 ±0.013	-1.561 ±2.319	0.012 ±0.010	3.390 (0.164)
	DE	0.647±0.121	-2.032±0.349	0.016 ±0.008	40.132
	LLA	0.043 ±0.014	-2.733 ±0.468	0.011 ±0.006	3.403 (0.177)
	SWAG	0.044 ±0.014	-2.565±0.118	0.067±0.025	19.408 (15.822)
Song	NUQLS	0.839 ±0.014	0.646±0.056	0.001±0.000	295.058 (91.673)
	DE	0.846 ±0.006	0.180 ±0.013	0.005±0.000	2562.789
	LLA-D	0.851 ±0.029	0.456±0.093	0.000 ±0.000	413.814 (210.429)
	SWAG	0.845±0.002	0.680±0.062	0.003±0.001	3477.825 (3274.440)

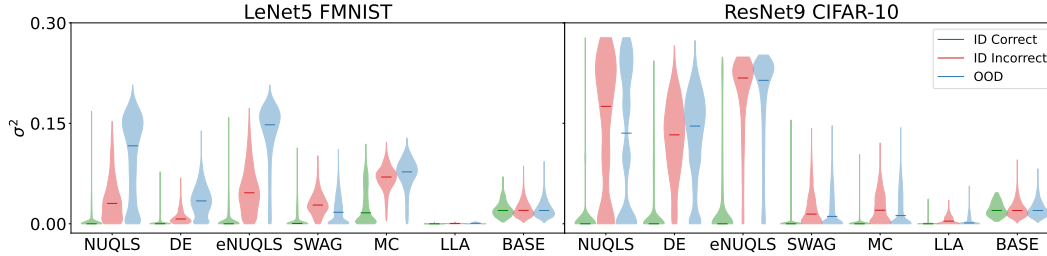


Figure 5: Violin plot of VMSP, with an ensemble version of NUQLS, eNUQLS, included.

We again observe that NUQLS shows greater separation than BE for VMSP across correct predictions versus incorrect or OOD predictions. Finally, in the bottom right of Figure 6, we compare NUQLS against further competing methods, specifically BDE, SGLD (Welling & Teh, 2011) (which we include to demonstrate the inferior performance of MCMC methods in this regime) and SNGP on ResNet9 trained on FMNIST (note SNGP requires modifications to the training procedure). We see that BDE performs similarly to DE, and SGLD performs similarly to SWAG, which is unsurprising considering their respective constructions. We also observe that NUQLS still performs the strongest across all methods using the VMSP violin plots. The respective median and skew values for Figure 6 are displayed in Table 10 and Table 11; these values also evidence the strong performance of NUQLS.

H.8 Image Classification Skew and Median

The median and sample skew for the VMSP in Figure 3 and Figure 6 is found in Table 10 and Table 11.

Table 10: Sample median and sample skew of variance from Figure 3 and Figure 6. IDC = ID correct, IDIC = ID incorrect, FMNIST = FashionMNIST. The median and skew for a method is compared against a baseline method. We expect positive skew for the ID correct (IDC) variances, and a negative skew for the ID incorrect (IDIC) and OoD variances. If the median or skew for a method is worse than the corresponding baseline, then it is written in gray.

Median		NUQLS	DE	SWAG	MC	LLA	BASE
ResNet9	IDC ↓	4.76×10^{-15}	1.31×10^{-5}	4.64×10^{-7}	5.05×10^{-7}	9.02×10^{-10}	0.020
FMNIST	IDIC ↑	0.182	0.040	0.048	0.004	0.004	0.020
	OoD ↑	0.217	0.109	0.075	0.004	0.009	0.020
ResNet50	IDC ↓	0.00	8.29×10^{-8}	0.018	1.49×10^{-9}	1.70×10^{-7}	0.019
CIFAR-10	IDIC ↑	0.178	0.120	0.123	1.32×10^{-4}	0.008	0.020
	OoD ↑	0.178	0.106	0.134	6.74×10^{-5}	0.004	0.020
ResNet50	IDC ↓	1.00×10^{-8}	8.73×10^{-4}	0.0449	1.12×10^{-4}	3.25×10^{-4}	0.020
CIFAR-100	IDIC ↑	0.211	0.0624	0.101	5.93×10^{-3}	0.0225	0.020
	OoD ↑	0.214	0.0665	0.0956	5.46×10^{-3}	0.0199	0.020
ResNet50	IDC ↓	8.61×10^{-15}	3.59×10^{-6}	1.71×10^{-6}	2.18×10^{-8}	1.61×10^{-6}	0.020
SVHN	IDIC ↑	0.217	0.0569	0.0302	3.96×10^{-4}	0.0241	0.020
	OoD ↑	0.233	0.127	0.0789	7.51×10^{-4}	0.0252	0.020
ResNet50	IDC ↓	0.00	-	-	-	-	0.020
ImageNet	IDIC ↑	0.232	-	-	-	-	0.020
	OoD ↑	0.100	-	-	-	-	0.020
Sample Skew							
ResNet9	IDC ↑	2.4	3.51	3.72	4.92	6.46	1.01
FMNIST	IDIC ↓	-0.615	0.928	0.378	1.60	1.18	1.11
	OoD ↓	-1.7	0.321	0.076	1.69	0.926	1.11
ResNet50	IDC ↑	2.97	2.68	4.02	4.55	5.14	1.14
CIFAR-10	IDIC ↓	-0.15	-0.05	0.66	2.45	0.816	1.09
	OoD ↓	0.02	-0.01	0.77	2.72	0.96	1.05
ResNet50	IDC ↑	1.08	1.48	0.508	2.87	1.8	1.07
CIFAR-100	IDIC ↓	-1.15	0.241	-0.47	2.05	0.228	1.09
	OoD ↓	-1.37	0.269	-0.233	1.85	0.217	1.13
ResNet50	IDC ↑	2.2	2.42	3.52	6.91	4.15	0.989
SVHN	IDIC ↓	-1.31	0.52	0.878	1.64	0.601	1.12
	OoD ↓	-1.78	-0.47	-0.0119	1.18	0.296	1.06
ResNet50	IDC ↑	1.52	-	-	-	-	1.21
ImageNet	IDIC ↓	-0.88	-	-	-	-	1.10
	OoD ↓	0.179	-	-	-	-	0.985

Table 11: Sample median and sample skew of variance from Figure 6. IDC = ID correct, IDIC = ID incorrect, FMNIST = FashionMNIST. The median and skew for a method is compared against a baseline method. We expect positive skew for the ID correct (IDC) variances, and a negative skew for the ID incorrect (IDIC) and OoD variances. If the median or skew for a method is worse than the corresponding baseline, then it is written in gray.

Median		NUQLS	BE	BDE	SGLD	SNGP	BASE
WRN-34-1	IDC ↓	0.00	8.31×10^{-5}	-	-	-	0.020
FMNIST	IDIC ↑	0.178	0.045	-	-	-	0.020
	OoD ↑	0.173	0.107	-	-	-	0.020
ResNet9	IDC ↓	4.76×10^{-15}	-	3.35×10^{-20}	6.79×10^{-8}	9.19×10^{-7}	0.020
FMNIST	IDIC ↑	0.182	-	0.00915	0.023	0.000322	0.020
	OoD ↑	0.217	-	0.0504	0.0352	0.000342	0.020
Sample Skew							
WRN-34-1	IDC ↑	3.1	2.8	-	-	-	1.01
FMNIST	IDIC ↓	-0.342	0.944	-	-	-	1.11
	OoD ↓	-0.0875	-0.272	-	-	-	1.11
ResNet9	IDC ↑	2.4	-	6.46	5.49	29.8	0.93
FMNIST	IDIC ↓	-0.615	-	1.43	1.23	24.4	1.15
	OoD ↓	-1.7	-	0.515	1.1	1.21	1.05

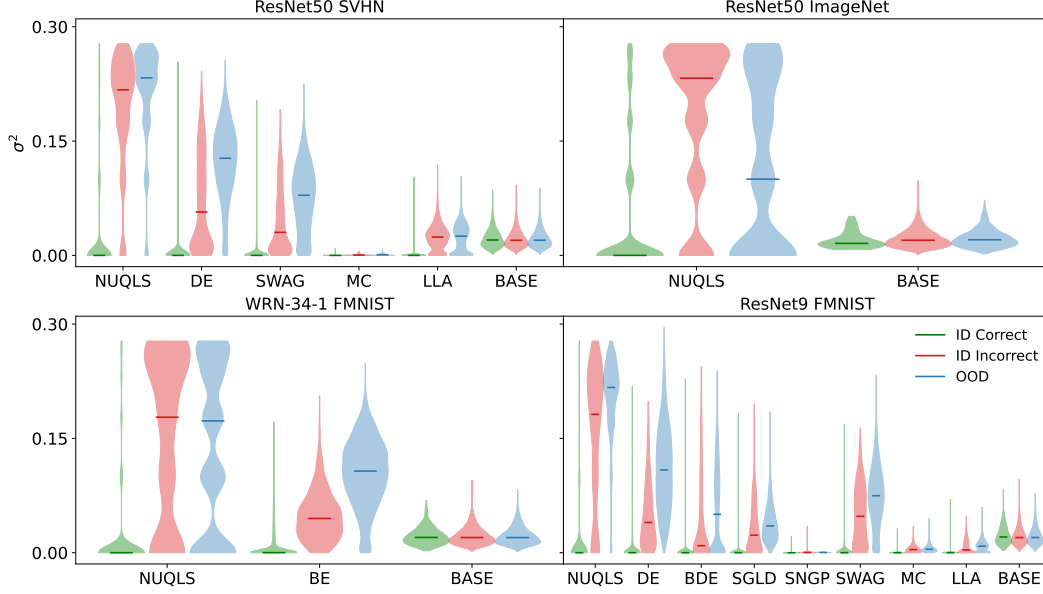


Figure 6: Violin plot of VMSP, for (top left) ResNet50 SVHN, (top right) ResNet50 ImageNet, (bottom left) WRN-34-1 FMNIST, and (bottom right) ResNet9 FMNIST.

I Experiment Details

All experiments were run either on an Intel i7-12700 CPU (toy regression), or on an H100 80GB GPU (UCI regression and image classification). Where multiple experiments were run, mean and standard deviation were presented.

I.1 NTK Convergence

For this experiment, 2 randomly sampled sets of 100 Gaussian inputs of dimension 5, and 100 scalar Gaussian targets, were used as training points and test points respectively. The weights of the MLP were initialised to $\mathcal{N}(0, 1)$; no bias terms were used. To train, GD was employed with a Nesterov momentum parameter of 0.9, a learning rate of 0.1, and 5000 epochs. For NUQLS, a learning rate of 0.1 was used, and γ was set to 1. Error bars on Figure 2 are the 95% sample confidence interval.

I.2 Toy Regression

We use a 1-layer MLP, with a width of 50 and SiLU activation. For the maximum a posteriori (MAP) network, we train for 10000 epochs, with a learning rate of 0.001, using the Adam optimizer and the PyTorch polynomial learning rate scheduler, with parameters `total_iters = epochs`, `power = 0.5`. For DE, each network in the ensemble outputs a heteroskedastic variance, and is trained using a Gaussian NLL, with 2000 epochs and a learning rate of 0.05. We combine the predictions of the ensembles as per (Lakshminarayanan et al., 2017). Both DE and NUQLS use 10 realizations. The γ hyper-parameter in NUQLS is set to 5, and each linear realization is trained for 1000 epochs with a learning rate of 0.001, using SGD with a momentum parameter of 0.9. In SWAG, the MAP network is trained for a further 10000 epochs, using the same learning rate, and the covariance is formed with a rank-10 approximation. A prior precision of 0.1 and 1 is used for LLA and LA respectively, as well as the full covariance matrix. The variational inference method used is Bayes By Backprop (Blundell et al., 2015), as deployed in the Bayesian Torch package (Krishnan et al., 2022). The prior parameters are ($\mu = 0, \sigma = 1$), and the posterior is initialized at ($\mu = 0, \rho = -3$). For SWAG, LLA, LA and VI, 1000 MC sample were taken at test time. These design choices gave the best performance for this problem.

Table 12: Training procedure for UCI regression results in Table 1 and Table 9.

NN	Energy	Concrete	Kin8nm	Naval	CCPP	Wine	Yacht	Protein	Song
Learning Rate	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
Epochs	1500	1000	500	150	100	100	1000	250	50
Weight Decay	10^{-5}	10^{-5}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	0
Optimizer	Adam	Adam	SGD	SGD	Adam	SGD	Adam	SGD	SGD
Scheduler	PolyLR	PolyLR	None	None	PolyLR	None	PolyLR	None	None
MLP Size	[150]	[150]	[100, 100]	[150, 150]	[100, 100]	[100]	[100]	[150, 200, 150]	[1000, 1000, 500, 50]
NUQLS									
Learning Rate	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-3}
Epochs	150	100	50	15	10	10	100	25	10
DE									
Learning Rate	10^{-3}	10^{-3}	10^{-2}	10^{-3}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
Epochs	1500	300	100	100	100	100	1000	250	50
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Scheduler	None	None	Cosine	None	None	None	Cosine	None	None
Experiment									
No. experiments	10	10	10	10	10	10	10	10	3

I.3 UCI Regression

We now provide the experimental details for the UCI regression experiments (as seen in Table 1 and Table 9). For each dataset, we ran a number of experiments to get a mean and standard deviation for performance metrics. In each experiment, we took a random 70%/15%/15% split of the dataset for training, testing, and validation. The training hyper-parameters for the MAP, DE and NUQLS networks, size of the MLP used, and the number of experiments conducted for each dataset can be found in Table 12. Reported time for NUQLS, LLA and SWAG includes the training time for the original NN, with the time to run the method given in brackets.

- **NN:** For the PolyLR learning rate scheduler, a PyTorch polynomial learning rate scheduler was used, with parameters $\text{total_iters}=10 \times \text{epochs}$, $\text{power}=0.5$. The MLP used a tanh activation, so as to have smooth gradients. MLP weights were initialized as Xavier normal, and bias as standard normal. The dataset was normalized, so that the inputs and the outputs each had zero mean and unit standard deviation.
- **NUQLS:** The linear networks were trained using (S)GD with Nesterov momentum parameter 0.9. For all datasets except for Song, the full training Jacobian could be stored in memory; this made training extremely fast. For the Song dataset, we trained all linear networks in parallel, by explicitly computing the gradient using JVPs and VJP. The number of linear networks used was 10 across all datasets, and the γ hyper-parameter was kept at 0.01.
- **DE:** Each member of the ensemble output a separate heteroskedastic variance, and was trained to minimise the Gaussian negative log likelihood. The ensemble weights were also initialized as Xavier normal, and bias as standard normal. The number of ensemble members was kept at 10.
- **LLA:** LLA requires two parameters for regression: a dataset noise parameter, and a prior variance on the parameters (Foong et al., 2019). To find the noise parameter, a grid search over 10 values on a log-scale between $1e-2$ and $1e2$ was used to find the noise that minimized the Gaussian likelihood of the validation set, with the LLA mean predictor as the Gaussian mean. The same grid search was used to find the prior variance, in order to minimize the expected calibration error (ECE) of LLA on the validation set. For the Protein dataset, a Kronecker-Factored Curvature (KFAC) covariance structure was used (Immer et al., 2021), and for the Song dataset a diagonal covariance structure was used. For all other datasets, LLA used the full covariance structure. The predictive distribution was computed with 1000 MC samples.
- **SWAG:** SGD was used, and the learning rate and number of epochs was kept the same as the NN. We used a grid-search for weight decay, over the values $[0, 0.005, 0.00005]$, to minimize the ECE on the validation set. The rank of the covariance matrix was 10 for all datasets, and the predictive distribution used 1000 MC samples.

Table 13: Training procedure for image classification results in Table 2, Figure 3, and Figure 6.

MAP	LeNet5 MNIST	Lenet5 FMNIST	ResNet9 FMNIST	ResNet50 CIFAR10	ResNet50 CIFAR100	ResNet50 SVHN	ResNet50 ImageNet
Learning Rate	5×10^{-3}	5×10^{-3}	10^{-3}	10^{-2}	10^{-1}	10^{-2}	-
Epochs	35	35	10	200	200	50	-
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}	5×10^{-4}	5×10^{-4}	-
Batch Size	152	152	100	128	128	128	-
Optimizer	Adam	Adam	Adam	SGD	SGD	SGD	-
Scheduler	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	-
Accuracy	99%	90%	92.5%	92.5%	75%	87%	76%
NUQLS							
Learning Rate	10^{-2}	10^{-2}	10^{-1}	10^{-1}	10^{-2}	10^{-2}	10^{-2}
Epochs	10	10	2	50	10	2	2
Batch Size	152	152	50	128	128	256	56
γ	1	0.7	0.1	0.01	0.05	0.05	0.7

I.4 Image Classification

We display the training procedure in Table 13 for both Figure 3 and Table 2. For MNIST and FashionMNIST, we took a 5 : 1 training/validation split of the training data. For CIFAR-10, we simply used the test data as a validation set. For CIFAR-10, random horizontal crop and flip on the training images was used as regularization.

- **NN:** We chose the training procedures to provide the best MAP performance. All networks have weights initialized as Xavier uniform. For SGD, a momentum parameter of 0.9 was used. For the Cosine Annealing learning rate scheduler, the maximum epochs was set to the training epochs.
- **NUQLS:** The number of samples was kept to 10 for all datasets.
- **DE:** Similarly, 10 ensemble members were used for all datasets.
- **MC-Dropout:** Dropout was applied to the network before the last fully connected layer. The dropout probability was set to 0.1 (a larger probability of 0.25 was also used, but it did not change the result). At test time, 100 MC-samples were taken.
- **SWAG:** The network was trained for a further $1 \times$ training epochs. For CIFAR-100 and SVHN, learning rate was kept the same as the original NN; for other datasets, a larger learning rate of $1e2 \times$ NN learning rate was used. The covariance rank was set at 10. At test time, 100 MC-samples were taken.
- **LLA*:** We used a last-layer KFAC approximation to the covariance. The prior precision was found through a grid search over 20 values on a log scale from $1e-2$ to $1e2$, using the probit approximation to the predictive, and a validation set. This configuration for LLA is what is recommended in Daxberger et al. (2021). We used 1000 samples, to remedy the large amount of approximations used.
- **VaLLA:** We used the implimentation found in (Ortega et al., 2023). We kept nearly all hyper-parameters the same as in the MNIST and FashionMNIST experiments in (Ortega et al., 2023); however, we reduced the number of iterations to 5000, due to time constraints. We were unable to run this implimentation for ResNet9 or ResNet50, due to an out-of-memory error.

I.5 Confidence Intervals

For the confidence intervals experiments in Table 4 and Table 3, we used 100 repeats for $d \leq 4$ and 50 repeats for $d \geq 8$. We used a multiplier of 1.5 on all NUQLS variance values, as we note that the search strategy for γ for regression only employs ECE; this does not reward confidence intervals that exceed the expected confidence, and hence computed confidence intervals may be conservative in width.

I.6 Comparison Experiments

I.6.1 BDE

For Figure 4 and Table 8, BDE was implimented following details given in (He et al., 2020). To make comparison with NUQLS fair, each ensemble member of BDE did *not* output a separate heteroskedastic variance term. Instead, the variance over predictions was taken as the variance. Further, the large computational cost of BDE relative to NUQLS meant that it is unrealistic for comparison to allow BDE to tune hyper-parameters through a grid-search, as this would take the run time of BDE

(which is slightly greater than that of DE) severely out-of-proportion to NUQLS. For Figure 4, the noise-parameter was given as the noise in the data, 9. For Table 8, standard Gaussian noise was taken a priori. The ensemble members were trained with the same training scheme as the original NN in both Figure 4 and Table 8. For both experiments, 10 ensemble members were used. For classification, we again followed the implementation details given in He et al. (2020) to the best of our ability. We employed the same hyperparameters as DE/MAP, and used 10 ensemble members.

I.6.2 SLU

For the results in Table 5, we copied the training details for (Miani et al., 2024a, Table 3.); we point the reader to (Miani et al., 2024a, Appendix D.1) for the exact details. For NUQLS, we used the following hyper-parameters for both MNIST and FashionMNIST: **Epochs** 10, **samples** 10, **learning rate** 0.01, **batch size** 152, γ 1.

I.6.3 Sampling-LLA, VaLLA and ELLA

For the results in Table 6, we used the training details for (Antorán et al., 2022, Figure 3. (left)); again, we point the reader to the details given in (Antorán et al., 2022, Appendix F.1). For NUQLS, we used the following hyper-parameters for MNIST: **Epochs** 10, **samples** 10, **learning rate** 0.01, **batch size** 152, γ 0.25. For ResNet20 the following hyper-parameters were used: **Epochs** 1, **samples** 100, **learning rate** 0.0001, **batch size** 152, γ 0.01. For ResNet32 the following hyper-parameters were used: **Epochs** 1, **samples** 150, **learning rate** 0.0001, **batch size** 152, γ 0.01.

I.6.4 BE

We employed the WideResNet architecture from Franchi et al. (2023). We used a depth of 10 and a widening factor of 1. The MAP network was trained with: **Epochs** 20, **learning rate** 0.005, **weight decay** 0.0001, **batch size** 152, the **Adam** optimizer and the **Cosine** learning rate scheduler. BE employed the same hyperparameters, yet with $1.5 \times$ **Epochs**, to aid with training. BE used 5 ensemble members, in order to match the computational budget of NUQLS. The NUQLS hyperparameters were: **Epochs** 10, **samples** 10, **learning rate** 0.01, **batch size** 152, γ 1.

I.6.5 SNGP

We employed the *Lightning UQ Box* (Lehmann et al., 2025) implementation of SNGP for our experiments. The same hyperparameters as the MAP network were used. Note that we also tested SNGP on ResNet50 SVHN, however SNGP was unable to train successfully, even after trying 3 learning rates and 2 optimizers. Hence, we did not include the results for ResNet50 SVHN.

I.6.6 SGLD

We have implemented the SGLD method from Welling & Teh (2011) using the Lightning UQ Box package (Lehmann et al., 2025) as the basis of the code. We then amended this code to include the learning rate scheduler from Welling & Teh (2011). We followed Maddox et al. (2019), and initialized the SGLD trajectory from the weights of the trained network. We also copied the learning rate from Maddox et al. (2019), and used the same weight-decay as the original network for the SGLD prior. However, in contrast to Maddox et al. (2019), we took the noise-factor scaling to be 10^{-3} instead of 5×10^{-4} , as we found that this gave better performance. We sampled 100 epochs from the posterior, using a batch size of 100.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: We claim that our method approximates the posterior of an NTK-GP, and provide theoretical and empirical proof in Section 3 and Section 4.1 respectively. We also claim that we outperform competing methods in performance and efficiency, which can be seen in Section 4 and Appendix H.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: Discussion of our limitations can be found in the "Limitations" subsection of the Conclusion, Section 5.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: We contain two provable results, Lemma 3.1 and Theorem 3.2. The proofs can be found in Appendix A.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: All code for the experiments has been included with the submission. All experimental details have been described in Appendix I.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 1. If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 2. If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 3. If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 4. We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [\[Yes\]](#)

Justification: Link to code, as well as a downloadable package of our method, can be found at the top of Section 4.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so No is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All training and test details are found in Appendix I.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: UCI Regression experiments (Tables 1, 8 and 9 and the empirical convergence figure (Figure 2) have error bars included, and are defined in Figure 2 and in Appendix I.3. Violin plots do not have error bars, as they are generally not reported with error bars.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).

- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Experimental resources are listed at the top of Appendix I.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: After reading through the NeurIPS Code of Ethics, we believe that we have conformed to this code.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss the benefits of improved UQ methods in our Introduction and Abstract. We are not aware of any direct negative impacts of our work (though secondary negative impacts may exist).

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We did not use any high risk models or datasets, and instead only use highly common models and datasets that are commonly used in the UQ literature.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Any prior code for related methods that has been used has been referenced in the linked GitHub repository for our code. Any related methods for comparison has been properly cited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Our method has been released as package, with examples given for how to correctly use our method.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.

- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The development of our method did not involve the use of LLMs.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.