GALERKIN MEETS LAPLACE: FAST UNCERTAINTY ESTIMATION IN NEURAL PDEs

Christian Jimenez Beltran[1], **Antonio Vergari**[2]*, **Aretha L. Teckentrup**[1]*, **Konstantinos C. Zygalakis**[1]* *
[1]School of Mathematics and Maxwell Institute for Mathematical Sciences, [2] School of Informatics
University of Edinburgh
{s2113174,avergari,a.teckentrup,k.zygalakis}@ed.ac.uk

ABSTRACT

The solution of partial differential equations (PDEs) by deep neural networks trained to satisfy the differential operator has become increasingly popular. While these approaches can lead to very accurate approximations, they tend to be overconfident and fail to capture the uncertainty around the approximation. In this work, we propose a Bayesian treatment to the deep Galerkin method (Sirignano & Spiliopoulos, 2018), a popular neural approach for solving parametric PDEs. In particular, we reinterpret the deep Galerkin method as the maximum a posteriori estimator corresponding to a likelihood term over a fictitious dataset, leading thus to a natural definition of a posterior. Then, we propose to model such posterior via the Laplace approximation, a fast approximation that allows us to capture meaningful uncertainty in out of domain interpolation of the PDE solution and in low data regimes with little overhead, as shown in our preliminary experiments.

1 INTRODUCTION

Partial differential equation (PDE) models appear in numerous areas of science and engineering, including geophysics, climate modelling and medical imaging to name a few (De Marsily, 1986; Isaacson et al., 2004; Schneider et al., 2017). In the last few years, deep learning approaches that use neural networks to approximate the solution of PDEs have shown remarkable success in terms of approximating certain classes of high-dimensional PDEs (Raissi et al., 2019; Kelshaw et al., 2022), thus providing fast alternatives to classical numerical methods such as finite differences, elements, volumes and spectral methods (Larsson & Thomée, 2003).

One of the first deep learning approaches designed for solving PDEs is the physics-informed neural networks (PINNs) (Raissi et al., 2019). PINNs define a suitable loss function that takes explicitly into account the PDE in question over specific collocation points, as well as over specific initial and boundary data. To do so, PINNs require fixed collocation points at training time, an aspect that can become problematic for high-dimensional PDEs. In addition, in many applications, one is interested in solving the PDE for a range of parameters, which is something that cannot be addressed by PINNs. The deep Galerkin method (DGM) (Sirignano & Spiliopoulos, 2018) can deal with these two issues by averaging oversampled collocation points and initial and boundary data during training and adding the PDE parameters as additional inputs to the neural network.

While these neural approaches can scale PDE solving to high dimensions, they still require large amounts of data to be trained and tend to be highly overconfident even when their approximations are far from the ground truth. This is especially problematic when they are used as surrogate solutions on collocations unseen during training (out-of-sample) and when trying to perform statistical analysis of real-world differential equation models, as it is important to have calibrated estimates of uncertainty with regards to our numerical approximations (Hennig et al., 2015; Conrad et al., 2017).

In this work, we address this issue for DGMs. We start by reinterpreting the DGM as the maximum a posteriori estimator corresponding to a likelihood term over a fictitious data set, leading thus to a natural Bayesian treatment. Then, we propose a simple and effective solution: approximating the

*Shared supervision.

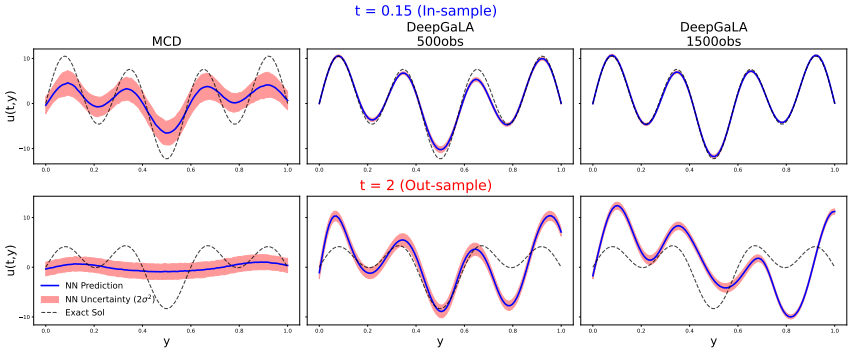

Figure 1: **Our DEEPGALA extends the DGM to provide meaningful uncertainty on out-of-sample** input locations (bottom) for the PDE solution of (PDE1) for $\theta = 0.001$ and when trained over 1000 (second column) and 3000 (third column) samples respectively. MCD method failes to approximate the solution even in-sample.

corresponding posterior via Laplace's method. This allows us to deliver fast uncertainty estimation over the PDEs that can be meaningful when training data is scarce or we evaluate our surrogate solution on out-of-sample collocations (see Figure 1 and Figure 2).

2 THE DEEP GALERKIN METHOD

We consider a general parametric PDE (see e.g. Deveney et al. (2023)), given by

$$\mathcal{A}(x, u(x;\theta);\theta_A) = h(x;\theta_h), \quad u(\partial x;\theta) = b(\partial x;\theta_b), \quad x \in \Omega, \quad \partial x \in \partial\Omega, \quad \theta \in \Theta \tag{1}$$

where the domain $\Omega \subset \mathbb{R}^d$ is the input space with boundary $\partial\Omega$, and $\Theta \subset \mathbb{R}^p$ is the parameter space for $\theta = (\theta_A, \theta_h, \theta_b)$. The differential operator $\mathcal{A}$ is parameterized by θ_A, whereas the functions h and b are parameterized by θ_h and θ_b, respectively. We denote by $u(x;\theta)$ the solution of the parametric PDE. Note that $\mathcal{A}$ could be non-linear, and that time could be a component of the input x. We want to use a neural network (NN) to approximate $u(x;\theta)$. To this end, consider a NN f_{W} with L layers and parameterized by $\mathrm{W} = \{\mathrm{W}_1, \ldots, \mathrm{W}_L\}$.[1] The NN takes as inputs a point $x \in \overline{\Omega}$ in the closure of the input domain and a parameter value $\theta \in \Theta$, and it is trained to approximate the solution of the PDE, $f_{\mathrm{W}}(x;\theta) \approx u(x;\theta)$.

The deep Galerkin method (DGM) (Sirignano & Spiliopoulos, 2018) uses a mesh-free strategy: At each iteration of the training step, K collocations for $x_i, \partial x_i$, and θ_i are sampled from densities π^p, π^b, and π^θ, respectively. Then, the NN f_{W} is trained to minimize the following loss:

$$\frac{1}{K}\sum_{i=1}^{K} \ell(D_i;\mathrm{W}) = \frac{1}{K}\sum_{i=1}^{K} \left(\ell_h(D_{h,i};\mathrm{W}) + \ell_b(D_{b,i};\mathrm{W})\right), \tag{2}$$

where $\mathcal{D} = \{D_i\}_{i=1}^K = \{D_{h,i}\}_{i=1}^K \cup \{D_{b,i}\}_{i=1}^K$, and $D_{h,i} = \{x_i, \theta_i\}$ and $D_{b,i} = \{\partial x_i, \theta_i\}$. Here, $\ell_h(D_{h,i};\mathrm{W}) = \left(\mathcal{A}(x_i, f_{\mathrm{W}}(x_i,\theta_i);\theta_{A,i}) - h(x_i;\theta_{h,i})\right)^2$ measures the error in the approximation of the differential operator and $\ell_b(D_{b,i};\mathrm{W}) = \left(f_{\mathrm{W}}(\partial x_i,\theta_i) - b(\partial x_i;\theta_{b,i})\right)^2$ measures the error in the boundary conditions. Optimization is carried out by gradient-based approaches, e.g., SGD, possibly increasing the number of samples K and the number of iterations. In this work, we followed a two-step training as suggested by He et al. (2020). Although conceptually simple, there are no convergence guarantees for DGM, and the error in predictions made by the NN f_W could be significant. In applications where the NN is then used in a computational pipeline, such as using NNs to approximate the likelihood in an inverse problem (Deveney et al., 2023), it becomes crucial to quantify the uncertainty in the predictions of the NN to avoid overconfident and biased inference.

[1] We consider the biases to be included in each W_l. Note that other non-sequential architectures are possible.

3 DEEPGALA: LAPLACE APPROXIMATION FOR BAYESIAN DGM

In this section, we explain how to extend the DGM to allow for uncertainty quantification in its predictions, using Bayesian theory and the Laplace approximation. (Psaros et al., 2023) provides a complete guide for uncertainty quantification for NN for function approximations, including Neural PDE solvers such as PINNS. They provide LA as an alternative to quantify uncertainty in their work, but they don't specifically evaluate this method using neural solvers. Furthermore, our method places greater emphasis on measuring the uncertainty brought on by small sample sizes, NN hyperparameters and overparametrization, and training methods.

3.1 BAYESIAN RE-INTERPRETATION OF DGM

In a Bayesian version of DGM, we are interested not in a single parameter configuration W, but in its posterior distribution $p(\mathrm{W} \mid \mathcal{D})$. To this end, we first interpret the data $\mathcal{D}$ to be the fictious dataset built by K collocations points sampled from π^p, π^b, and π^θ. Then, we interpret the loss in equation 2 as a negative log-likelihood by considering Gaussian noise around the fictitious data observed at the collocation points. By doing this, the likelihood $p(\mathcal{D}|\mathrm{W})$ characterises how well a choice of weights W approximates the PDE solution $u(x;\theta)$, and is given by

$$p(\mathcal{D}|\mathrm{W}) = p(D_h|\mathrm{W})p(D_b|\mathrm{W}) = \frac{1}{(2\pi\sigma^2)^{2K}} \prod_{i=1}^{K} \exp\left(-\frac{\ell_h(D_{h,i};\mathrm{W}) + \ell_b(D_{b,i};\mathrm{W})}{2\sigma^2}\right), \tag{3}$$

where we assume that the likelihoods of the collocations D_h and D_b are conditionally independent given W. We further introduce a Gaussian prior distribution $p(W)$. By Bayes' rule, we then have the posterior distribution $p(W|\mathcal{D}) \propto p(\mathcal{D}|\mathrm{W})p(W)$ on the weights W given the fictitious training data $\mathcal{D}$. Note that although not explicitly indicated, the posterior $p(W|\mathcal{D})$ is also conditioned on $\mathcal{A}, h$ and b from equation 1, since these are used to define the likelihood. While intractable in general, $p(W|\mathcal{D})$ is amenable to a fast and and effective approximation, as discussed next.

3.2 A LAPLACE APPROXIMATION OF BAYESIAN DGM

We propose a Laplace Approximation (LA) of the posterior of our Bayesian DGM, which we call DEEPGALA– Deep Galerkin via Laplace. The LA is a well-known method for approximating intractable posterior distributions as a Gaussian distribution centred on the maximum a posteriori (MAP) solution W_{MAP}, i.e.,

$$\arg\min_{W} \frac{1}{2\sigma^2} \sum_{i=1}^{K} \ell(D_i;\mathrm{W}) + r(W), \tag{4}$$

where the negative log-prior $r(W) = -\log p(W)$ is an L2 regularizer (weight decay) in our experiments. Despite its simplicity (assuming unimodality of the posterior), the LA has been demonstrated to be a solid alternative to more complex approximations such as MCMC and VI on a number of uncertainty quantification tasks for Bayesian deep learning (Daxberger* et al., 2021). To compute the LA, the first step is to find the minimizer W_{MAP} of equation 4, which can be achieved by training f_{W} via gradient-based optimization. The second step is to fit the local Gaussian distribution, namely $\mathcal{N}(W; W_{\text{MAP}}, \Lambda)$ where Λ^{-1} is the Hessian of the negative log-posterior evaluated at W_{MAP}. In our case, $r(W)$ is a weight decay regularizer, which corresponds to a Gaussian prior distribution $p(W) = \mathcal{N}(W; 0, \gamma^2 I)$ (Daxberger* et al., 2021), Λ^{-1} taking the form

$$\Lambda^{-1} = -\frac{1}{2\sigma^2} \sum_{i=1}^{K} \nabla^2_W \log p(D_i|W)|_{W_{\text{MAP}}} - \gamma^{-2} I. \tag{5}$$

Exactly computing Λ^{-1} above can be computationally demanding, as W in modern NNs can be very large (Nilsen et al., 2019). To this end, we employ a fast approximation of the covariance in DEEPGALA, by leveraging a number of heuristics from the modern Bayesian deep learning literature. First, we consider only the weights W_L of the last layer of f, which is generally enough to deliver good uncertainty estimates as noted in Sharma et al. (2023). [2] Second, we subsample a subset K' of all the training points seen by f during training. Third, we either compute the full Hessian or approximate the Hessian by its diagonal (Pearlmutter, 1994). Appendix A details the whole process.

[2] This also allows us to compute the posterior predictive in closed-form, see Appendix A.1 for details.

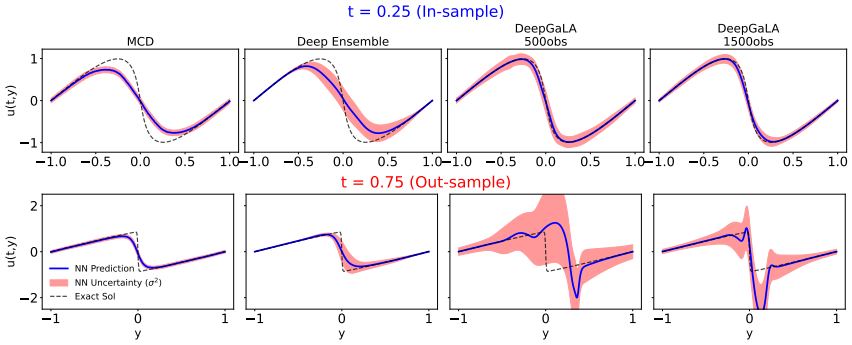

Figure 2: **DEEPGALA provides meaningful uncertainty on out-of-sample** input locations (bottom) for the Burger's solution of equation PDE2 for $\theta = 0.01/\pi$ and when trained over 500 (third column) and 1500 (fourth) samples. MCD and Deep failed to approximate the solution of equation PDE2.

4 EXPERIMENTS AND DISCUSSION

We aim to answer the following research questions: **RQ1)** does DEEPGALA capture meaningful uncertainty over out-of-sample inputs x? **RQ2)** does increasing training size properly reduce the epistemic uncertainty highlighted by DEEPGALA?

PDEs. To this end, we consider two PDEs defined over $x = (y, t)$: A heat equation with external heat source:

$$\frac{\partial u}{\partial t} = \theta \frac{\partial^2 u}{\partial y^2} + \sin(5\pi y), \quad t > 0, \quad \theta > 0 \quad \text{and} \quad y \in [0, 1], \tag{PDE1}$$

with conditions $u(y, 0) = 4\sin(3\pi y) + 9\sin(7\pi y)$, and $u(0, t) = u(1, t) = 0$; and Burgers Equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial y} - \theta \frac{\partial^2 u}{\partial y^2} = 0, \quad t > 0, \quad \theta > 0 \quad \text{and} \quad y \in [-1, 1], \tag{PDE2}$$

with conditions $u(y, 0) = -\sin(\pi y)$, and $u(-1, t) = u(1, t) = 0$. PDE1 admits analytic solutions, allowing us to directly compare the performance of DEEPGALA both in terms of accuracy and uncertainty. For PDE2, we use the results obtained by the work of Raissi et al. (2019) for $\theta = 0.01/\pi$ as a baseline.

The experiments were carried out in a machine with 12th Gen Intel Core i7-1265Ux12, 32 GB of RAM and Ubuntu 22.04 operating system.

NNs models. For both PDEs we use a feed-forward NN with three layers and 40 neurons using hyperbolic tangent non-linearities, trained with Adam for 1,200 epochs, a learning rate of 0.01 and regularizer, γ, of 0.0015 for PDE1 and 0.0001 for PDE2, followed by the L-BFGS optimizer in a two-step optimisation, as recommended by He et al. (2020). We show results for the diagonal approximation in Appendix B. The mean time to fit and call the full Hessian LA for PDE1 and PDE2 was 0.081 and 0.003, and 0.077 and 0.0044 seconds respectively.

Alternative UQ baselines. We test our approach against Deep Ensemble (DE) (Lakshminarayanan et al., 2017b) and Monte Carlo Dropout (MCD) (Lakshminarayanan et al., 2017a). In order to calculate the mean and variance for the MCD, 1,000 samples were used, with the dropout layer set to $p = 0.05$ for PDE1 and $p = 0.1$ for PDE2, respectively. For the PDE2, the NN architecture stays the same, however for the PDE1, five hidden layers comprising forty neurons were employed. The reason for the size modification was that smaller NN were not producing satisfactory results for this specific PDE. Five NN's of three layers and forty neurons were trained to approximate the PDE2 using the DE method in accordance with Lakshminarayanan et al. (2017a). Nevertheless, obtaining satisfactory results for the PDE1 was surprisingly challenging; this difficulty may have arisen from the PDE's initial condition. That's the reason why we don't present the results utilizing this baselines for the PDE1.

RQ1) In- vs out-of-sample. For both PDEs, we provide DEEPGALA only a portion of the input at training time (in-sample), expecting it to extrapolate poorly – but delivering good uncertainty – on out-of-sample inputs. In particular, we draw our in-sample collocations from the following intervals: $y_i \sim \mathcal{U}[0, 1]$, $t_i \sim \mathcal{U}[0, 1]$, $\theta_i \sim \mathcal{U}[0.0001, 0.05]$ for PDE1 and $y_i \sim \mathcal{U}[-1, 1]$, $t_i \sim \mathcal{U}[0, 0.5]$, $\theta_i \sim \mathcal{U}[0.0001, 0.05]$ for PDE2, where $\mathcal{U}$ denotes the uniform distribution. The top (in-sample) and bottom (out-of-sample) sides of Figure 1 and Figure 2 show two uncertainty surfaces representing 95% and 68% confidence intervals around the MAP estimate for DEEPGALA for PDE1 and PDE2 respectively. Reassuringly, these uncertainties increase when the model makes more mistakes, highlighting how DEEPGALA can be used to indicate that the NN are "aware" that they may be forecasting incorrectly. In contrast, the uncertainty exhibits no specific behaviour when examined both in- and out-of-sample for the methods MCD and DE, nor do they provide good approximations at the mean.

RQ2) training size effect. We train DEEPGALA over sample sizes of 1,000, and 3,000 for PDE1 and 500, and 1,500 for PDE2, plotting the results in the top and bottom halves of Figure 1 and Figure 2. All models trained with more samples outperform the other (both for in-sample and out-sample values of t). Nonetheless, the exact solution is within the uncertainty range of the models for this particular value of t. Again, DEEPGALA is shown to be an effective way to estimate the epistemic uncertainty of NNs for PDE solving, which can be useful in quantifying how much data can improve performance. As such, we plan to use the uncertainty DEEPGALA provides in an active learning loop to select what are the most promising collocation to improve accuracy of the PDE solution.

5 CONCLUSION AND DISCUSSION

We proposed DEEPGALA as a fast and effective way to estimate uncertainty in deep neural PDE solvers, and tested it on preliminary benchmarks with encouraging results. We plan to evaluate DEEPGALA to more complex PDEs, extending it with recent advancements in Bayesian deep learning (Wilson, 2020) and comparing to other Bayesian PDE solvers such as the ones presented in Psaros et al. (2023), which however are limited to non-parametric PDEs. We hope that scaling uncertainty estimation can lead to a renewed interest for efficient probabilistic numerics for complex neural PDEs in the real-world.

ACKNOWLEDGEMENTS

The Maxwell Institute of Mathematical Sciences and the University of Edinburgh's School of Mathematics provided funding for this research.

REFERENCES

Patrick R. Conrad, Mark Girolami, Simo Särkkä, Andrew Stuart, and Konstantinos Zygalakis. Statistical analysis of differential equations: introducing probability measures on numerical solutions. *Statistics and Computing*, 27(4):1065–1082, 2017.

E. Daxberger*, A. Kristiadi*, A. Immer*, R. Eschenhagen*, M. Bauer, and P. Hennig. Laplace Redux — Effortless Bayesian Deep Learning. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, pp. 20089–20103. Curran Associates, Inc., December 2021. URL https://proceedings.neurips.cc/paper/2021/file/a7c9585703d275249f30a088cebba0ad-Paper.pdf. *equal contribution.

Ghislain De Marsily. *Quantitative hydrogeology*. Academic Press, London, 1986.

Teo Deveney, Eike H. Mueller, and Tony Shardlow. Deep Surrogate Accelerated Delayed-Acceptance Hamiltonian Monte Carlo for Bayesian Inference of Spatio-Temporal Heat Fluxes in Rotating Disc Systems. *SIAM/ASA Journal on Uncertainty Quantification*, 11(3):970–995, 2023.

QiZhi He, David Barajas-Solano, Guzel Tartakovsky, and Alexandre M. Tartakovsky. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources*, 141:103610, 2020. ISSN 0309-1708. doi: https://doi.org/10.1016/j.advwatres.2020.103610. URL https://www.sciencedirect.com/science/article/pii/S0309170819311649.

Philipp Hennig, Michael A. Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, 2015. doi: 10.1098/rspa.2015.0142. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2015.0142.

David Isaacson, Jennifer L Mueller, Jonathan C Newell, and Samuli Siltanen. Reconstructions of chest phantoms by the D-bar method for electrical impedance tomography. *IEEE Transactions on medical imaging*, 23(7):821–828, 2004.

Daniel Kelshaw, Georgios Rigas, and Luca Magri. Physics-informed CNNs for super-resolution of sparse observations on dynamical systems. *arXiv preprint arXiv:2210.17319*, 2022.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017a.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017b.

Stig Larsson and Vidar Thomée. *Partial differential equations with numerical methods*. Springer, 2003.

Geir K Nilsen, Antonella Z Munthe-Kaas, Hans J Skaug, and Morten Brun. Efficient computation of hessian matrices in tensorflow. *arXiv preprint arXiv:1905.05559*, 2019.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Apostolos F. Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics*, 477:111902, 2023. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2022.111902. URL https://www.sciencedirect.com/science/article/pii/S0021999122009652.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Tapio Schneider, Shiwei Lan, Andrew Stuart, and João Teixeira. Earth system modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, 44(24):12–396, 2017.

Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. Do Bayesian Neural Networks Need To Be Fully Stochastic? In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent (eds.), *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pp. 7694–7722. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/sharma23a.html.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.029. URL https://www.sciencedirect.com/science/article/pii/S0021999118305527.

Andrew Gordon Wilson. The case for Bayesian deep learning. *arXiv preprint arXiv:2001.10995*, 2020.

A EFFICIENT COMPUTATION OF THE LAPLACE APPROXIMATION

Following Sharma et al. (2023), we consider partially stochastic networks and introduce uncertainty only in W_L, the parameters in the last layer of f_{W}. The last layer, involving only an affine transformation, can be represented as $f_W(x,\theta) = W_L \cdot [1, \phi(y_{L-1}(x,\theta))]$, where W_L is the vector of weights and biases, and $\phi(y_{L-1}(x,\theta))$ is the vector of outputs of the penultimate layer for input (x,θ). Let's just focus on calculating the Hessian for one input $z_i = (x_i, \theta_i)$ and compute $\nabla^2_{W_L}\ell(z_i;\mathrm{W})$, where we will use the abbreviation $l_i$and $f_{i,W}$ to denote $\ell(z_i;W)$ and $f_W(z_i)$. Therefore,

$$\nabla^2_{W_L} l_i = \nabla_W \begin{bmatrix} \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^0} \\ \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^1} \\ \vdots \\ \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^K} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 l_i}{\partial f^2_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^0} \frac{\partial f_{i,W}}{\partial w_L^0} & \cdots & \frac{\partial^2 l_i}{\partial f^2_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^0} \frac{\partial f_{i,W}}{\partial w_L^K} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 l_i}{\partial f^2_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^K} \frac{\partial f_{i,W}}{\partial w_L^0} & \cdots & \frac{\partial^2 l_i}{\partial f^2_{i,W}} \frac{\partial f_{i,W}}{\partial w_L^K} \frac{\partial f_{i,W}}{\partial w_L^K} \end{bmatrix} + \begin{bmatrix} \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial^2 f_{i,W}}{\partial w_L^{02}} & \cdots & \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial^2 f_{i,W}}{\partial w_L^0 \partial w_L^K} \\ \vdots & \ddots & \vdots \\ \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial^2 f_{i,W}}{\partial w_L^K \partial w_L^0} & \cdots & \frac{\partial l_i}{\partial f_{i,W}} \frac{\partial^2 f_{i,W}}{\partial w_L^{K2}} \end{bmatrix}. \quad (6)$$

In the special case of a last layer with only an affine transformation, the second term in the previous equation becomes zero. Additionally, the terms $\frac{\partial f_{i,W}}{\partial w_L^k} = \phi_k(y_{L-1}(z_i))$ and $\frac{\partial f_{i,W}}{\partial w_L^0} = 1$. Consequently, we can reformulate the previous equation as:

$$\nabla^2_{W_L} l_i = \mathbf{J}_{W_L}(f_{i,W}) \frac{\partial^2 l_i}{\partial f^2_{i,W}} \mathbf{J}_{W_L}(f_{i,W})^T = \\ = \frac{\partial^2 l_i}{\partial f^2_{i,W}} \begin{bmatrix} 1 & \phi_1(y_{L-1}(z_i)) & \cdots & \phi_K(y_{L-1}(z_i)) \\ \phi_1(y_{L-1}(z_i)) & \phi_1(y_{L-1}(z_i))^2 & \cdots & \phi_K(y_{L-1}(z_i))\phi_1(y_{L-1}(z_i)) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_K(y_{L-1}(z_i)) & \phi_1(y_{L-1}(z_i))\phi_K(y_{L-1}(z_i)) & \cdots & \phi_K(y_{L-1}(z_i))^2 \end{bmatrix}. \quad (7)$$

Another observation is that the last equation is akin to the Gauss-Newton Matrix (GNN), up to a constant $G(W) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{J}_W(f_W(z_n))^T \nabla^2_f l(f_W(z_n)) \mathbf{J}_W(f_W(z_n))$. The GNN is a first-order Taylor approximation of the loss function. We can compute the equation 5 by using the next equation:

$$\Lambda^{-1} = -\gamma^{-2} I - \frac{1}{2\sigma^2} \sum_{i=1}^{K} (\mathbf{J}_{W_L}(f_W(z_i)) \nabla^2_f \ell(D_{h,i};\mathrm{W}) \mathbf{J}_{W_L}(f_W(z_i))^T + \\ \mathbf{J}_{W_L}(f_W(\partial z_i)) \nabla^2_f \ell(D_{b,i};\mathrm{W}) \mathbf{J}_{W_L}(f_W(\partial z_i))^T), \quad (8)$$

where $\mathbf{J}_{W_L}(f_W(z_i)) = [\frac{\partial f_W(z_i)}{\partial w_L^0}, ..., \frac{\partial f_W(z_i)}{\partial w_L^c}]^T$. Note that equation 8 represents a full Hessian; for a big enough NN, we can approximate even further to obtain a diagonal approximation by using the Hadamard product. Thus, the Diagonal approximation of the Hessian is computed in the following way:

$$\Lambda^{-1} = -\gamma^{-2} I - \frac{1}{2\sigma^2} \sum_{i=1}^{K} (\mathbf{J}_{W_L}(f_W(z_i)) \odot \nabla_f^2 \ell(D_{h,i}; \mathrm{W}) \odot \mathbf{J}_{W_L}(f_W(z_i))^T + \mathbf{J}_{W_L}(f_W(\partial z_i)) \odot \nabla_f^2 \ell(D_{b,i}; \mathrm{W}) \odot \mathbf{J}_{W_L}(f_W(\partial z_i))^T). \quad (9)$$

A.1 Posterior predictive

Finally, in order to perform predictions, one needs to compute the posterior predictive distribution. This is done in the following way. Given our focus on the last layer of f_W, linearity in the weights W is observed. Therefore, for a test input $\hat{z}_i$ of the neural network, we obtain $p(f_*|f_W(\hat{z}_i), \mathcal{D}) = \mathcal{N}(f_*; f_{W_{MAP}}(\hat{z}_i), \phi(y_{z_i,L-1})^T \Lambda \phi(y_{z_i,L-1}))$, where $f_{W_{\text{MAP}}}$ is the output of the neural network at the MAP. In the case of a diagonal Hessian, Λ is easily obtained by computing the inverse of each value on the diagonal. For a full Hessian, the process involves first computing the lower triangular decomposition, where $\Lambda^{-1} = L^T L$, then finding the inverse of L, and finally obtaining $\Lambda = L^{-1}(L^{-1})^T$.

B Diagonal Hessian uncertainty

Figures 3 and 4 depict the results of DeepGaLa when applying a diagonal approximation to the Hessian to quantify uncertainty for the solutions of PDE1 and PDE2. The mean time to fit and call the diagonal LA for the PDE1 and PDE was 0.07 and 0.004, and 0.067 and 0.06 seconds respectively. As we can see, the uncertainty estimation has been reduced; yet, our method continues to be a good tool to predict how much performance may be improved. However, we still need to understand and quantify how much uncertainty estimation changes when utilising full or diagonal Hessian.

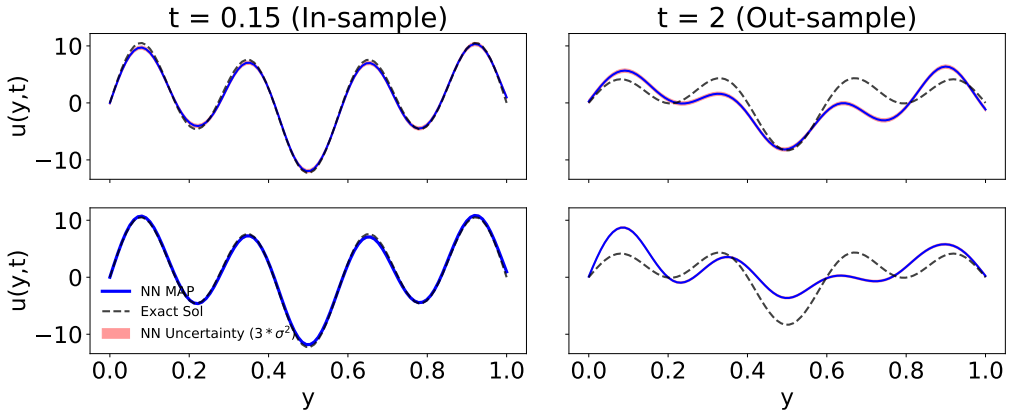

Figure 3: **DeepGaLa Diagonal Hessian Approximation for the PDE1 with** $\theta = 0.001$: Uncertainty results for in-sample (*left column*) and out-of-sample (*right column*) inputs when DeepGaLa is trained over 1000 (top) and 3000 (bottom) data points respectively.

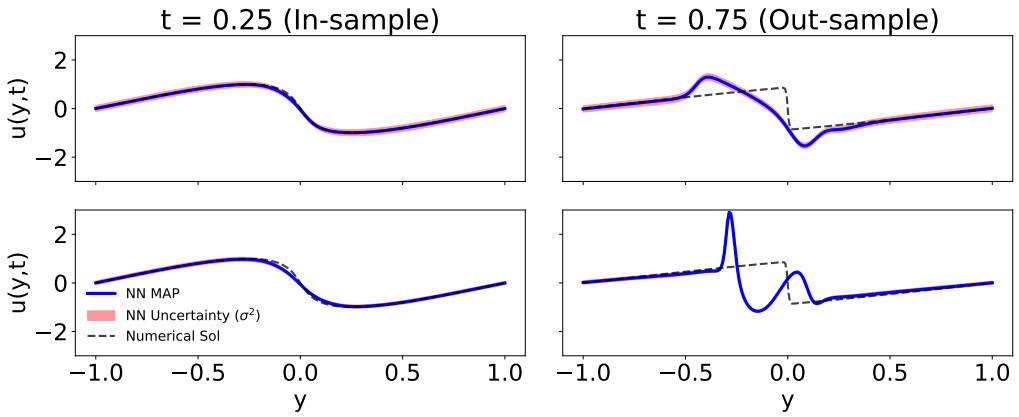

Figure 4: **DEEPGALA Diagonal Hessian Approximation for the PDE2 with** $\theta = 0.01/\pi$: Uncertainty results for in-sample (*left column*) and out-of-sample (*right column*) inputs when DEEPGALA is trained over 500 (top) and 1500 (bottom) data points respectively.