Intrinsic uncertainties and where to find them

Francesco Farina [1] **Lawrence Phillips** [1] **Nicola J Richmond** [1]

Abstract

We introduce a framework for uncertainty estimation that both describes and extends many existing methods. We consider typical hyperparameters involved in classical training as random variables and marginalise them out to capture various sources of uncertainty in the parameter space. We investigate which forms and combinations of marginalisation are most useful from a practical point of view on standard benchmarking data sets. Moreover, we discuss how some marginalisations may produce reliable estimates of uncertainty without the need for extensive hyperparameter tuning and/or large-scale ensembling.

1. Introduction

While modern neural network-based machine learning architectures have yielded a step change in predicition accuracy over traditional methods, especially in application domains such as computer vision, speech recognition and natural language processing, accurately estimating uncertainty remains an issue. Providing accurate estimations of the uncertainty associated to each prediction is of critical importance, particularly in the healthcare domain. Classical Bayesian inference methods (Wang & Yeung, 2020; MacKay, 1992; Neal, 2012; Murphy, 2012) address the uncertainty question but they typically do not scale with architecture and data set size and so resort to leveraging computationally-tractable approximations via parametrisation.

This paper introduces a framework for uncertainty estimation that provides a novel and simple way to describe and extend many uncertainty estimation methods. The main idea relies on considering hyperparameters involved in classical training procedures as random variables. We show that through marginalising out different combinations of hyperparameters, different sources of uncertainty can be estimated in the (model) parameter space. Within this framework, methods such as SWAG (Maddox et al., 2019), deep ensembles (Lakshminarayanan et al., 2017), MultiSWAG (Wilson & Izmailov, 2020), MC-dropout (Gal & Ghahramani, 2016) and hyperparameter ensembles (Lévesque et al., 2016; Wenzel et al., 2020) are shown to be approximations to particular marginalisations.

We apply methods resulting from this framework on benchmarking problems to investigate which forms and combinations of marginalisation are most useful from a practical point of view. As one may expect, results suggest that increasing the number of random variables that are marginalised out tends to increase the quality of the uncertainty estimates, with certain random variables playing a more evident role. In particular, some combinations of marginalisations can produce reliable estimates of uncertainty without the need for extensive hyperparameter tuning. However, this is not the case with all combinations.

2. Finding uncertainties

Given a set of data points $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$, one usually wishes to determine the predictive distribution $p(y \mid x, \mathcal{D})$ for some new value x. As this problem is generally intractable, it is common practice to assume the predictive distribution takes on a certain functional form described by a parametric model $\mathcal{F}_\Theta = \{f_\theta : \mathbb{R}^n \to \mathbb{R}^m, \theta \in \Theta\}$ for some parameter space $\Theta \subset \mathbb{R}^d$, and to determine

$$p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta) = \int_\theta p(y \mid x, \theta) p(\theta \mid \mathcal{D}) \mathrm{d}\theta. \qquad (1)$$

where the value of y, given x and θ (and consequently f_θ) is deterministic and $p(\theta \mid \mathcal{D})$ is the distribution over θ that likely gave rise to $\mathcal{D}$.

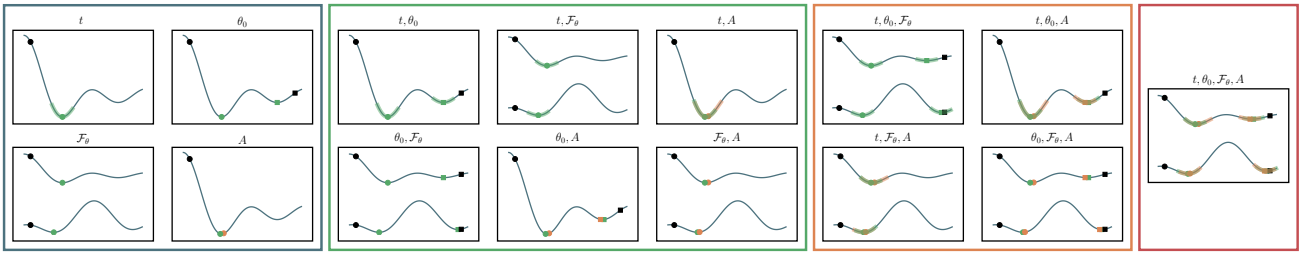
For modern architectures and data sets, computing $p(\theta \mid \mathcal{D})$ is computationally intractable. This has given rise to a number of proposed approximate methods. In the next sections, we show that many of these methods can be realised within our framework as approximating the conditional probability distribution of θ given the data set $\mathcal{D}$ and additional random variables that capture different aspects of the distribution, a graphical representation of which is depicted in Figure 1.

2.1. A starting point: classical training

In the classical (deterministic) modern neural network training paradigm, the training goal is to find the optimal model

[1]GSK.ai, GlaxoSmithKline, London, United Kingdom. Correspondence to: Francesco Farina <francesco.x.farina@gsk.com>.

Presented at the ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning.,



(a) single marginalisation (b) two marginalisations (c) three marginalisations (d) four marginalisations

Figure 1. Pictorial view of the uncertainty induced by the different hyperparameters. The blue line represent the loss function $\mathcal{L}$, black dots are the initial conditions θ_0 and green and orange dots are (sub)optimal solutions after t iterations of the algorithm. Shaded regions represent the possible position of θ_t for $t \geq \bar{\tau}$.

parameter $\theta_\star \in \Theta$ that minimises some notion of difference between y_i and $f_\theta(x_i)$, usually referred to as loss $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$, by *solving*

$$\underset{\theta}{\text{minimise}} \quad \sum_{(x_i, y_i) \in \mathcal{D}} \mathcal{L}(y_i, f_\theta(x_i)). \tag{2}$$

Given $\theta_\star$, classical training methods can be seen as approximating $p(\theta \mid \mathcal{D}) \approx \delta(\theta - \theta_\star \mid \mathcal{D})$. Determining an exact solution to (2) is computationally intractable for most problems arising in machine learning due to factors such as data dimensionality and size, non-convexity of the problem, etc. The aim then is to find a suboptimal solution via an iterative algorithm A_h from the countably infinite set of optimisation algorithms $\mathcal{A}$, with chosen hyperparameter point h in an algorithm-appropriate hyperparameter space H[1].

Let θ_τ be the solution estimate at iteration τ of A_h. The sequence $\{\theta_\tau\}_{\tau=1}^t$ produced by t iterations of A_h is determined by the initial condition $\theta_0 \in \Theta_0 \subset \mathbb{R}^d$ and the hyperparameters h, *e.g.*, step-size, batch-size and batch process order. With a slight abuse of notation, we consider A_h as a function $A_h : \mathbb{R}^d \to \mathbb{R}^d$ and write θ_t as $A_h^t(\theta_0)$ to represent t compositions (iterations) of A_h.

It should now be clear that since problem (2) is generally intractable, rather than simply making the approximation $p(\theta \mid \mathcal{D}) \approx \delta(\theta - \theta_\star \mid \mathcal{D})$, classical training computes the value of $p(\theta \mid \mathcal{D})$ further conditioned on θ_0, h and t, *i.e.*,

$$p(\theta \mid \mathcal{D}, t, h, \theta_0) = \delta(\theta - A_h^t(\theta_0) \mid \mathcal{D}). \tag{3}$$

Then the output of a trained model f_{θ_t} is characterised as

$$p(y \mid x, \mathcal{D}, t, h, \theta_0) = \delta(y - f_{A_h^t(\theta_0)}(x) \mid \mathcal{D}). \tag{4}$$

2.2. Intrinsic uncertainties in classical training

By marginalising out the different variables θ_0, t, h and possibly the model family $\mathcal{F}$, we should obtain estimates for $p(y \mid x, \mathcal{D})$ that are better than the point mass distribution (4). In the next sections, we will show that this is indeed the case and that many of the current state-of-the-art algorithms for uncertainty estimation can be considered as estimating $p(\theta \mid \mathcal{D})$ (or $p(y \mid x, \mathcal{D})$) by marginalising out different subsets of the conditioning variables in (3) (or (4)).

2.2.1. Number of iterations t

Under suitable assumptions, algorithms in $\mathcal{A}$ usually generate a sequence of solution estimates that, in the limit, converges to a neighbourhood of a stationary point $\hat{\theta}_\star$. In practice, convergence to this neighbourhood usually occurs after a certain number of iterations $\bar{\tau}$. While in general there are no guarantees regarding the position of θ_t in relation to $\hat{\theta}_\star$ within the neighbourhood, the trajectory of the solution estimates $\{\theta_\tau\}$ generated by the algorithm provides information on the geometry of parameters space near $\hat{\theta}_\star$ which in turn induces a probability distribution over Θ (see Figure 1, plot t in box (a)). Formally, we have

$$p(\theta \mid \mathcal{D}, h, \theta_0) = \lim_{t \to \infty} \frac{1}{t - \bar{\tau}} \sum_{\tau=\bar{\tau}}^{t} \delta(\theta - A_h^t(\theta_0) \mid \mathcal{D}),$$

which for stochastic gradient descent algorithms can be shown to be Gaussian in the limit. Note that this formalism nicely describes the SWAG algorithm (Maddox et al., 2019), which assumes $p(\theta \mid \mathcal{D}, h, \theta_0)$ is a Gaussian distribution and computes the first and second order momenta by taking finite samples from $\{\tau, \tau + 1, \ldots\}$, and the TRADI algorithm (Franchi et al., 2020) which again makes the Gaussian assumption and tracks the momenta of Θ along the algorithm evolution through a Kalman filter.

2.2.2. Initial condition θ_0

Due to the likely non-convexity of the loss function, the initial condition θ_0 may determine the stationary point around which $\{\theta_\tau\}$ stabilises. Thus, the initial condition induces a probability distribution over Θ that can be computed by

[1] Hereafter, when talking about hyperparameters we will refer only to those characterising the algorithm A.

marginalising (3) with respect to θ_0, as

$$p(\theta \mid \mathcal{D}, t, h) = \int_{\theta_0} \delta(\theta - A_h^t(\theta_0) \mid \mathcal{D}) p(\theta_0) \mathrm{d}\theta_0, \quad (5)$$

where $p(\theta_0)$ is chosen according to an appropriate parameter initialisation strategy for the model architecture such as the Glorot uniform distribution (Glorot & Bengio, 2010). See Figure 1, plot θ_0 in box (a) for a graphical representation.

Approximating (5) is clearly the goal of classical ensemble methods for uncertainty estimation (see, *e.g.*, Lakshminarayanan et al. (2017)).

2.2.3. Algorithm A

As mentioned, in general, different algorithm procedures and/or different points in appropriate hyperparameter space produce different approximate solutions to problem (2). For example, in the case of the simple Stochastic Gradient Descent (SGD), the step size, batch size and the order in which batches are processed by the algorithm all influence the trajectory $\{\theta_\tau\}$. This is also true for more complex algorithms where other hyperparameters may play a role. To capture the uncertainty associated to the choice of algorithm, we can marginalise out h from $p(\theta \mid \mathcal{D}, t, h, \theta_0)$ to give

$$p(\theta \mid \mathcal{D}, t, \theta_0) = \int_h \delta(\theta - A_h^t(\theta_0) \mid \mathcal{D}) p(h) \mathrm{d}h.$$

In Appendix B we characterise for the SGD and for the general class of stochastic algorithms. A graphical representation is reported in Figure 1, plot A in box (a), where different choices of h lead to different values of $A_h^t(\theta_0)$.

Ensembling over hyperparameter space to improve model performance and uncertainty estimation has been proposed in (Lévesque et al., 2016) and (Wenzel et al., 2020).

2.2.4. Model family $\mathcal{F}$

Recall that assumptions are made about the functional form of the predictive distribution to enable its computation. With this in mind, so far, we have presented our framework in the context of a fixed parametric model family $\mathcal{F}_\Theta$ over some parameter space Θ. The particular choice of model family typically involves architecture and hyperparameter space selection and is usually informed by the problem domain and the available data set $\mathcal{D}$. As different parametric model families yield different estimates of the predictive distribution, the choice of model family is also a source of uncertainty which one can account for by marginalising out $\mathcal{F}_\Theta$ to give

$$p(y \mid x, \mathcal{D}) = \int_{\mathcal{F}_\Theta} p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta) p(\mathcal{F}_\Theta) \mathrm{d}\mathcal{F}_\Theta, \quad (6)$$

where $p(\mathcal{F}_\Theta)$ is some unknown distribution over the space of all possible parametric model families over all possible parameter spaces. In practice, when deciding on a parametric model family, a typical approach is to focus on a finite subset of possible modelling approaches informed by *a priori* knowledge of the problem domain and then choose the best family or families via a model selection procedure (see, *e.g.*, (Murphy, 2012)). Another approach is to use Monte Carlo dropout (Gal & Ghahramani, 2016) which ensembles models obtained via edge dropout (see Appendix C). Figure 1, plot $\mathcal{F}_\theta$ in box (a), pictorially shows how selecting different models (which in turns produces different loss landscapes) may lead to different solutions.

2.3. Multiple marginalisation

Until now, we have focussed on marginalising out single conditioning random variables to obtain better approximations to $p(\theta \mid \mathcal{D})$. A natural step at this point is to combine two or more of the proposed marginalisations.

In Figure 1, boxes (b)-(d), all the possible combinations of marginalisations in our framework are depicted, some of which have recently been studied. For example, multi-SWAG (Wilson & Izmailov, 2020), an extension to SWAG that can be seen as an approximation to marginalising out t and θ_0 to give

$$p(\theta \mid \mathcal{D}, h) = \int_{\theta_0} \lim_{t \to \infty} \frac{1}{t - \bar{\tau}} \sum_{\tau=\bar{\tau}}^{t} \delta(\theta - A_h^t(\theta_0) \mid \mathcal{D}) p(\theta_0) \mathrm{d}\theta_0.$$

This is depicted in Figure 1, box (b), plot t, θ_0. Another example is hyper-ensembles (Wenzel et al., 2020) which can be seen as marginalising out both θ_0 and the hyperparameter space (Figure 1, box (b), plot θ_0, A).

In general, this framework allows a number of marginalisation combinations that have not been addressed in the literature thus far. Depending on the particular problem at hand, different combinations may lead to improved uncertainty estimates.

3. Estimating uncertainty

Until now, we have focussed on estimating uncertainty in the parameter solution space. By using (1), that uncertainty can be propagated to the predictive distribution. However, computing it explicitly is often intractable and approximations are preferred. Monte Carlo sampling is arguably the most widely used approach to approximating $p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, *)$ as

$$\hat{p}(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, *) = \frac{1}{K} \sum_{k=1}^{K} p(y \mid x, \mathcal{D}, \theta_k, *),$$

where K is the number of samples drawn from the trained model parameter space and $\theta_k \sim \hat{p}(\theta \mid \mathcal{D}, *)$. As K increases, the accuracy of $\hat{p}(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, *)$ also increases.

Marginalisation	boston housing	concrete	energy	kin8nm	naval	power plant	protein	wine	yacht
$\mathcal{M}_\theta$	13.134 ± 3.835	9.342 ± 1.388	1.487 ± 0.352	3.374 ± 0.590	-4.395 ± 3.015	32.769 ± 4.371	96.558 ± 13.491	23.328 ± 7.372	1.300 ± 0.543
t	5.958 ± 1.962	5.752 ± 1.092	1.278 ± 0.164	0.198 ± 0.551	-4.464 ± 0.049	14.668 ± 3.007	11.097 ± 5.094	10.749 ± 2.840	0.733 ± 0.183
θ_0	5.681 ± 2.039	7.162 ± 1.961	2.058 ± 1.392	2.636 ± 1.209	$\mathbf{-6.020 \pm 0.173}$	23.229 ± 6.921	35.612 ± 7.356	6.649 ± 1.791	1.414 ± 2.311
$\mathcal{M}_\theta + t$	4.096 ± 0.730	4.365 ± 0.810	1.310 ± 0.087	-0.596 ± 0.250	-4.417 ± 0.065	9.824 ± 1.965	11.526 ± 3.242	7.121 ± 1.806	0.763 ± 0.106
$\mathcal{M}_\theta + \theta_0$	4.015 ± 1.287	4.351 ± 0.417	$\mathbf{1.032 \pm 0.136}$	-0.210 ± 0.189	-5.847 ± 0.095	12.105 ± 1.754	24.517 ± 1.083	4.157 ± 0.858	$\mathbf{0.525 \pm 0.198}$
$t + \theta_0$	$\mathbf{3.511 \pm 0.816}$	3.930 ± 0.387	1.230 ± 0.086	-0.673 ± 0.204	-4.449 ± 0.024	11.539 ± 1.688	$\mathbf{8.910 \pm 2.600}$	4.022 ± 0.772	$\mathbf{0.634 \pm 0.147}$
$\mathcal{M}_\theta + t + \theta_0$	$\mathbf{2.998 \pm 0.593}$	$\mathbf{3.494 \pm 0.286}$	1.317 ± 0.055	$\mathbf{-0.977 \pm 0.123}$	-4.425 ± 0.023	$\mathbf{8.223 \pm 1.159}$	$\mathbf{7.030 \pm 2.080}$	$\mathbf{3.035 \pm 0.559}$	0.741 ± 0.133

Table 1. NLL on UCI data sets expressed and mean $\pm$ standard deviation over the 20 train-test folds (5 for the protein dataset). Blue represents the best values and bold those whose mean is within the error bound of the best. Note that marginalising out $\mathcal{M}_\theta$, t and θ_0 is approximately by MC-dropout, SWAG and ensembles respectively.

Similarly, the mean and variance of the predictive distribution can be approximated as $\hat{\mu}_{\hat{Y}} = \frac{1}{K}\sum_{k=1}^{K} f_{\theta_k}(x)$ and $\hat{\sigma}^2_{\hat{Y}} = \frac{1}{K}\sum_{k=1}^{K}(f_{\theta_k}(x) - \mu_{\hat{Y}})^2$, with $\theta_k \sim \hat{p}(\theta \mid \mathcal{D}, *)$.

Assumed Density Filtering (ADF) (Ghosh et al., 2016) can also be used to perform to compute the expected value and variance of $p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, *)$ in a single forward pass (see Appendix D for further details).

4. Experimental results

In this section, we apply methods described by the proposed framework on benchmarking problems to investigate which forms and combinations of marginalisation are most useful from a practical point of view. We consider problems on benchmark UCI datasets and CIFAR-10. Additional details and experiments are provided in Appendix A.

4.1. Regression on UCI datasets

We consider regression problems on UCI data sets, originally proposed in (Hernández-Lobato & Adams, 2015) and used for benchmarking MC-dropout (Gal & Ghahramani, 2016), SWAG (Maddox et al., 2019) and deep ensembles (Blundell et al., 2015). Each data set is split into 20 train-test folds except for *protein* where 5 folds were used. The network architecture had one hidden layer consisting of 50 units (100 for the *protein* dataset), followed by a dropout layer with dropout rate fixed at 0.01. Each model was trained for 400 epochs to minimise the Mean Squared Error with no regularisation, using Adam with a learning rate $\alpha = 0.01$ and batch-size $b = 100$.

We used SWAG, MC-dropout and 5-model ensembles to approximate the marginalisation of t, $\mathcal{F}_\theta$ (as shown in the appendix) and θ_0 respectively. Results on all marginalisation combinations are reported in Table 1 in terms of negative log-likelihood (NLL). The tendency is for NLL to decrease (and the quality of the uncertainty estimate to improve) as more random variables are marginalised out, with combined $\mathcal{F}_\theta$, t and θ_0 producing the lowest NLL in 6 of 9 data sets.

4.2. Classification on CIFAR-10

We train VGG16 neural networks for 300 epochs over CIFAR-10 using SGD with batch-size $b = 128$. We approximated the marginalisation over t and $\mathcal{M}_\theta$ using SWAG and MC-dropout respectively, while ensembles were used to marginalise over θ_0 and learning rates α (see Appendix A.3 for furhter detais).

The NLL and accuracy on the test data set with increasing ensemble size are reported in Figure 2. As we increase the ensemble size, the NLL decreases and the accuracy increases, an outcome we know from (Blundell et al., 2015). However, one must note that even with a single model, marginalising out additional random variables t, $\mathcal{M}_\theta$ and α produces a significantly lower NLL (and higher accuracy). In particular, while t have a significant impact, other marginalisations seems to be less effective. This trend continues when we add more models to the ensemble.

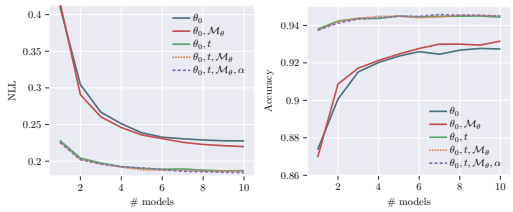


Figure 2. Evolution of NLL and accuracy over the CIFAR dataset.

5. Discussion

From the preliminary experiments we performed some trends seem to appear. First, we see that marginalising θ_0 and t always produce good results. This is probably due to the fact that these two marginalisations better capture the geometry of the parameter space for a fixed model and set of hyperparameters. Marginalising out other hyperparameters can be useful (Wenzel et al., 2020), but it probably requires some preliminary fine tuning to select a good distribution from which to draw them. A similar reasoning can be applied to the marginalisation of the model family, where a preliminary search for good candidates seems to be required. MC-dropout with a very small dropout rate can be a good option to minimally perturb a good architecture. If fine tuning is not an option, then marginalising θ_0 and t seems to be the best way to go among the options available in this framework.

References

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pp. 1613–1622. PMLR, 2015.

Brach, K., Sick, B., and Dürr, O. Single shot mc dropout approximation. *arXiv preprint arXiv:2007.03293*, 2020.

Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., and Bloch, I. Tradi: Tracking deep neural network weight distributions. In *European Conference on Computer Vision (ECCV) 2020*. Springer, 2020.

Frey, B. J. and Hinton, G. E. Variational learning in nonlinear gaussian belief networks. *Neural Computation*, 11(1): 193–213, 1999.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 3, pp. 1651–1660, 2016.

Gast, J. and Roth, S. Lightweight probabilistic deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018a.

Gast, J. and Roth, S. Lightweight probabilistic deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3369–3378, 2018b.

Ghosh, S., Delle Fave, F. M., and Yedidia, J. S. Assumed density filtering methods for learning bayesian neural networks. In *AAAI*, pp. 1589–1595, 2016.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pp. 1861–1869, 2015.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pp. 6402–6413, 2017.

Lévesque, J.-C., Gagné, C., and Sabourin, R. Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 437–446, 2016.

Loquercio, A., Segu, M., and Scaramuzza, D. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 5(2):3153–3160, 2020.

MacKay, D. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. A Simple Baseline for Bayesian Uncertainty in Deep Learning. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 13153–13164. Curran Associates, Inc., 2019.

Minka, T. P. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Neal, R. M. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Wang, H. and Yeung, D.-Y. A survey on bayesian deep learning. *ACM Computing Surveys (CSUR)*, 53(5):1–37, 2020.

Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in Neural Information Processing Systems*, 33, 2020.

Wilson, A. G. and Izmailov, P. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. *arXiv:2002.08791 [cs, stat]*, April 2020. arXiv: 2002.08791.

Wu, A., Nowozin, S., Meeds, E., Turner, R. E., Hernández-Lobato, J. M., and Gaunt, A. L. Deterministic variational inference for robust bayesian neural networks. In *International Conference on Learning Representations*, 2018.

A. Details and additional experiments

A.1. Regression on toy data

We first consider a simple 1-dimensional toy regression dataset consisting of 10 training samples generated from $y = x^3 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 9)$ and $x \in \mathcal{U}[-4, 4]$, as

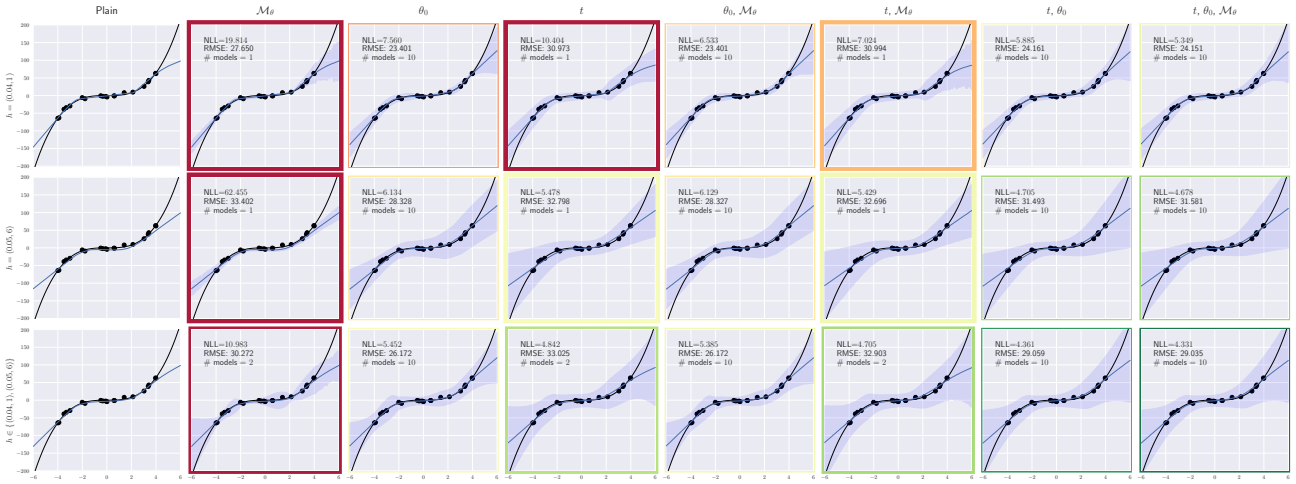


Figure 3. Regression on toy data. Training samples are depicted as blue dots. The black line denotes the true data generating function, while the blue line denotes the predictive mean of the trained model. The shadowed area denotes the 3σ confidence interval. Each plot (except for those in the first column) has a border that is coloured according to the NLL on test data, with green indicating low NLL and red high. The thickness of the border represents the number of trained models in the ensemble: the thicker the border, the smaller the ensemble. The NLL, RMSE and ensemble size are also reported in each plot.

in (Hernández-Lobato & Adams, 2015; Blundell et al., 2015; Franchi et al., 2020), to evaluate the performance of different marginalisations. A neural network with one hidden layer consisting of 100 units with ReLU activation was trained for 100 epochs to fit the data using the stochastic gradient descent algorithm. We compared the effects of approximating the marginalisation over t and θ_0 with hyperparameters $h = (\alpha, b) \in \{(0.04, 1), (0.05, 6)\}$. The marginalisation over t was approximated using SWAG, while over θ_0, we used an ensemble of 20 models. RMSE and NLL were calculated on a test set of 1000 equally spaced points (x, x^3), $x \in [-6, -6]$ that lie on the black line.

The results for the various marginalisation strategies over the different hyperparameter combinations are depicted in Figure 3. The first and second rows correspond to models trained with hyperparameter points $h = (0.04, 1)$ and $h = (0.05, 6)$ respectively, while results in the third row are obtained by ensembles combining the two hyperparameter point. For the first, third and fifth columns, we ensembled over two models (one for each hyperparameter point), whereas the remaining columns, corresponding to marginalising out θ_0, are ensembles of 5 initial conditions, trained once with $h = (0.04, 1)$ and once with $h = (0.05, 6)$.

Our results show that the quality of predictive uncertainty clearly benefits from multiple marginalisations with a decreasing NLL trend along the rows and columns. Interesting points to note are that despite the hyperparameter choice, $h = (0.05, 6)$ often produces a lower NLL than $h = (0.04, 1)$ and that the combined hyperparameters (third row) outperforms both individual hyperparameters (first and second rows). For RMSE, $h = (0.04, 1)$ outperforms $h = (0.05, 6)$, with the two combined generally lying somewhere in between, but with the additional advantage of a lower NLL. Also worthy of note is the computation time, measured in terms of number of trained models. While marginalising out θ_0 will outperform other random variable combinations given a sufficiently large ensemble, the point is that good estimates can be achieved with limited computational budget, *e.g.*, combined hyperparameters with SWAG and MD-dropout (third row, sixth column).

Dataset	N	n	m	$\bar{\alpha}$
boston housing	506	13	1	0.050
concrete	1030	8	1	0.015
energy	768	8	1 (2)	0.056
kin8nm	8192	8	1	0.017
naval	11934	16	1 (2)	0.033
power plant	9568	4	1	0.099
protein	45730	9	1	0.099
wine	1599	11	1	0.013
yacht	308	6	1	0.099

Table 2. Details on UCI data sets.

A.2. UCI datasets

A.2.1. Details on datasets

Recall that a given data set $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$, with $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}^m$, has number of samples N, (input) random variable dimension n and (output) random variable dimension m. Table 2 provides additional details on the UCI data sets used for our experiments and the tuned learning rates $\bar{\alpha}$ we will use in the next section. As in (Gal & Ghahramani, 2016) and other works using these UCI data sets, for *energy* and *naval*, we predict the first output variable so, $m = 1$.

Marginalisation	boston housing	concrete	energy	kin8nm	naval	power plant	protein	wine	yacht
$\mathcal{M}_\theta$	**3.247 ± 0.622**	6.584 ± 0.957	1.806 ± 0.541	0.118 ± 0.009	0.002 ± 0.001	4.818 ± 0.564	5.140 ± 0.129	**0.660 ± 0.043**	1.387 ± 0.448
t	**2.900 ± 0.600**	**5.535 ± 0.451**	1.570 ± 0.313	**0.100 ± 0.004**	0.003 ± 0.000	**4.289 ± 0.199**	5.072 ± 0.159	**0.633 ± 0.041**	**1.134 ± 0.254**
θ_0	**2.798 ± 0.660**	**5.467 ± 0.424**	**1.278 ± 0.115**	0.102 ± 0.004	**0.002 ± 0.000**	**4.341 ± 0.270**	**4.929 ± 0.065**	**0.635 ± 0.040**	**0.890 ± 0.278**
$\mathcal{M}_\theta + t$	**2.954 ± 0.643**	**5.597 ± 0.446**	1.568 ± 0.276	**0.100 ± 0.005**	0.003 ± 0.001	**4.276 ± 0.182**	**4.932 ± 0.054**	**0.634 ± 0.042**	**1.107 ± 0.252**
$\mathcal{M}_\theta + \theta_0$	**2.834 ± 0.617**	**5.445 ± 0.429**	**1.221 ± 0.171**	0.103 ± 0.004	**0.002 ± 0.000**	**4.331 ± 0.160**	**4.943 ± 0.060**	**0.634 ± 0.041**	**0.913 ± 0.295**
$t + \theta_0$	**2.917 ± 0.678**	**5.510 ± 0.407**	1.475 ± 0.145	**0.098 ± 0.003**	0.003 ± 0.000	**4.261 ± 0.185**	**4.940 ± 0.066**	**0.631 ± 0.039**	**1.024 ± 0.220**
$\mathcal{M}_\theta + t + \theta_0$	**2.912 ± 0.660**	**5.464 ± 0.372**	1.481 ± 0.158	**0.098 ± 0.003**	0.003 ± 0.000	**4.258 ± 0.179**	**4.944 ± 0.073**	**0.630 ± 0.039**	**1.006 ± 0.231**

Marginalisation	boston housing	concrete	energy	kin8nm	naval	power plant	protein	wine	yacht
$\mathcal{M}_\theta$	51.813 ± 42.183	$\sim 10^2$	$\sim 10^2$	21.343 ± 19.598	2.788 ± 18.344	26.959 ± 13.069	$\sim 10^6$	$\sim 10^4$	9.075 ± 13.178
t	3.360 ± 0.443	3.764 ± 0.246	**2.129 ± 0.178**	−0.615 ± 0.100	−3.812 ± 0.111	3.305 ± 0.167	7.725 ± 3.579	8.861 ± 2.216	1.462 ± 0.181
θ_0	6.471 ± 3.983	5.692 ± 2.593	2.651 ± 1.768	0.972 ± 0.747	**−4.759 ± 0.152**	6.490 ± 2.578	21.574 ± 10.052	16.994 ± 3.050	**1.219 ± 0.324**
$\mathcal{M}_\theta + t$	**3.321 ± 0.735**	3.684 ± 0.241	**2.170 ± 0.166**	−0.713 ± 0.092	−3.776 ± 0.252	3.252 ± 0.130	7.542 ± 1.405	9.034 ± 2.350	1.434 ± 0.142
$\mathcal{M}_\theta + \theta_0$	4.738 ± 1.919	4.779 ± 0.739	**2.097 ± 0.389**	0.037 ± 0.240	**−4.858 ± 0.117**	5.062 ± 1.122	16.541 ± 2.250	17.652 ± 6.301	**1.095 ± 0.336**
$t + \theta_0$	**2.847 ± 0.426**	**3.492 ± 0.134**	**2.114 ± 0.097**	**−0.794 ± 0.054**	−3.837 ± 0.044	**3.138 ± 0.127**	5.165 ± 1.392	**7.390 ± 1.010**	**1.415 ± 0.131**
$\mathcal{M}_\theta + t + \theta_0$	**2.842 ± 0.500**	**3.450 ± 0.132**	**2.146 ± 0.117**	**−0.822 ± 0.058**	−3.786 ± 0.100	**3.114 ± 0.134**	**3.677 ± 0.488**	**6.833 ± 1.452**	1.456 ± 0.122

Table 3. Training for 40 epochs with $\alpha = 0.1$. RMSE (top) and NLL (bottom) on UCI data sets expressed and mean ± standard deviation over the 20 train-test folds (5 for the protein dataset). Blue represents the best metrics and bold those whose mean is within the error bound of the best.

Marginalisation	boston housing	concrete	energy	kin8nm	naval	power plant	protein	wine	yacht
t	**3.294 ± 0.796**	**5.386 ± 0.431**	1.244 ± 0.164	0.079 ± 0.002	0.002 ± 0.000	**4.260 ± 0.184**	4.894 ± 0.041	**0.623 ± 0.044**	**1.103 ± 0.247**
θ_0	**3.125 ± 0.624**	**5.048 ± 0.367**	**0.573 ± 0.086**	**0.075 ± 0.002**	**0.001 ± 0.000**	**4.255 ± 0.158**	4.880 ± 0.052	**0.615 ± 0.038**	**0.856 ± 0.288**
$\theta_0 + \alpha$	**3.141 ± 0.740**	**5.042 ± 0.350**	**0.595 ± 0.092**	**0.075 ± 0.002**	**0.001 ± 0.000**	**4.245 ± 0.154**	**4.874 ± 0.081**	**0.614 ± 0.038**	**0.871 ± 0.280**
$t + \theta_0$	**3.231 ± 0.778**	**5.372 ± 0.412**	1.111 ± 0.114	**0.075 ± 0.002**	**0.001 ± 0.000**	**4.225 ± 0.158**	**4.864 ± 0.056**	**0.619 ± 0.042**	**0.987 ± 0.199**
$t + \theta_0 + \alpha$	**3.232 ± 0.788**	**5.372 ± 0.408**	1.120 ± 0.138	**0.075 ± 0.002**	**0.001 ± 0.000**	**4.224 ± 0.159**	**4.840 ± 0.040**	**0.618 ± 0.042**	**0.981 ± 0.190**

Marginalisation	boston housing	concrete	energy	kin8nm	naval	power plant	protein	wine	yacht
t	3.548 ± 0.847	**3.426 ± 0.208**	2.063 ± 0.122	−1.100 ± 0.040	−3.715 ± 0.101	3.245 ± 0.241	10.192 ± 1.791	8.290 ± 1.010	**1.408 ± 0.098**
θ_0	6.332 ± 2.875	7.386 ± 2.497	**1.306 ± 0.341**	0.552 ± 0.528	**−5.129 ± 0.258**	7.034 ± 2.323	19.269 ± 4.607	13.956 ± 3.587	**1.276 ± 0.603**
$\theta_0 + \alpha$	5.663 ± 2.183	7.322 ± 2.172	**1.215 ± 0.390**	0.427 ± 0.433	**−5.261 ± 0.173**	7.387 ± 3.239	21.291 ± 4.050	15.283 ± 4.751	**1.330 ± 0.739**
$t + \theta_0$	**2.836 ± 0.398**	**3.269 ± 0.159**	2.067 ± 0.055	**−1.136 ± 0.022**	−3.699 ± 0.039	**3.017 ± 0.056**	**4.847 ± 1.249**	**5.726 ± 0.795**	**1.412 ± 0.088**
$t + \theta_0 + \alpha$	**2.853 ± 0.445**	**3.274 ± 0.162**	2.068 ± 0.056	**−1.139 ± 0.022**	−3.704 ± 0.038	**3.014 ± 0.074**	**5.522 ± 1.850**	**5.728 ± 0.792**	**1.410 ± 0.089**

Table 4. Training for 40 epochs with tuned learning rates. RMSE (top) and NLL (bottom) on UCI data sets expressed and mean ± standard deviation over the 20 train-test folds (5 for the protein dataset). Blue represents the best metrics and bold those whose mean is within the error bound of the best.

A.2.2. Training for 40 epochs

Here we report the results obtained by training the networks for 40 epochs with two different set-ups: firstly using Adam with a fixed learning rate $\alpha = 0.1$ for all data sets and secondly using the tuned learning rates $\alpha = \bar{\alpha}$ reported in Table 2.

Fixed learning rate The results for the first case are reported in Table 3. Clearly, the number of epochs is too low for MC-dropout to converge (see (Gal & Ghahramani, 2016)), thus the NLL for $\mathcal{M}_\theta$ is generally poor. However, when also marginalising out t and/or θ_0 (again approximated via SWAG and an ensemble of 5 models respectively), the addition of MC-dropout generally results in lower NLL suggesting that even with few training epochs, MC-dropout may help when marginalised out in combination with other random variables.

Tuned learning rates Results with tuned learning rates are reported in Table 4. We approximate jointly marginalising out α and θ_0 by drawing 5 α samples from $\mathcal{N}(\bar{\alpha}, \bar{\alpha}/100)$, one for each θ_0 in the ensemble, and combine this marginalisation approximation with other SWAG and multiSWAG as required. While we propose that aggressive hyperparameter tuning can be avoided with multitple marginalisations, we do see an improvement in NLL when compared to Table 1 and for a number of data sets the NLL decreases further by perturbing the value of α.

Comments By comparing the results in Tables 1, 3, and 4, we make several observations.

1. The trend of NLL decreasing as the number of marginalisation combinations increases is maintained.
2. Training for 40 epochs with a (fixed) higher or tuned learning rate generally produces comparable or even better results than those reported in Table 1. However, this comes with a cost of higher RMSE for most data sets which is likely due to non-convergence after training for 40 epochs and hence higher prediction error. In general, the training set up very much depends on the desired use of a prediction. In domains such as disease diagnostics, understanding prediction certainty is more valuable than accuracy.
3. Fine-tuning the learning rates leads to an improvement in NLL across all data sets. However, the addition of MC-dropout can produce better results without the need for hyperparameter fine tuning.

A.3. Implementation details for CIFAR-10

As in (Maddox et al., 2019), the learning rate was defined as a decreasing schedule along epochs $e = 1, \ldots, n_e$ as

$$\alpha_e = \begin{cases} \alpha_u & \text{if } e < 0.5 n_e \\ \frac{\alpha_u - \alpha_l}{0.4 n_e} e & \text{if } e \in [0.5 n_e, 0.9 n_e] \\ \alpha_l & \text{if } e > 0.9 n_e \end{cases}$$

for some $\alpha_u > \alpha_l$. When approximating the marginalisation over the learning rate α, we randomly selected $\alpha_u \in$

$\mathcal{U}[0.04, 0.06]$ and $\alpha_l \in \mathcal{U}[0.008, 0.012]$ when marginalising out α, otherwise we fixed $\alpha_u = 0.05$ and $\alpha_l = 0.01$.

B. Algorithm marginalisation

We first provide a detailed description of marginalising out hyperparameters in the case of stochastic gradient descent. Then, we extend this idea to the (countably infinite) set of optimisation algorithms typically used in machine learning.

Stochastic gradient descent Over a single iteration, the SGD is governed by the step size $\alpha \in \mathbb{R}$, the batch size $b \in [N] = \{1, \dots, N\}$ and the batch number $i \in [N_b]$, where $N_b = \lceil \frac{N}{b} \rceil$, which without loss of generality, corresponds to

$$\mathcal{B}_{b,i} = \{(x_{ib+1}, y_{ib+1}), \dots, (x_{(i+1)b}, y_{(i+1)b})\} \subset \mathcal{D}.$$

In the context of our framework, the hyperparameter space H is defined as

$$H = \big\{(\alpha, b, i) \in \mathbb{R} \times [N] \times [N] \mid i \leq N_b\big\},$$

where points $p(\alpha, b, i)$ are drawn randomly with probability $p(\alpha, i \mid b) \times p(b)$, $p(\alpha, i \mid b) = p(\alpha) \times \mathcal{U}\{1, N_b\}$ and $p(\alpha)$ and $p(b)$ are assumed to be well chosen distributions, for example, through hyperparameter optimisation. Note that $p(\alpha)$ is a continuous distribution whereas $p(b)$ and $\mathcal{U}\{1, N_b\}$ are discrete. Since random variables A, B and I are independent, taking products of their respective marginal distributions is well defined.

To allow for scheduling of step size and batch size, over the course of a complete training run, the hyperparameter space H^t over which we marginalise is defined as

$$\big\{(\boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{i}) \in \mathbb{R}^t \times [N]^t \times [N]^t \mid i_\tau \leq N_b\big\},$$

where variable vectors are indexed over $\tau \in \{1, \dots, t\}$. Hyperparameter points are drawn randomly with probability

$$p(\boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{i}) = p(\boldsymbol{\alpha}) \times \Pi_{\tau=1}^t \mathcal{U}\{1, N_b\} \times p(\boldsymbol{b}).$$

Note that at each iteration, we are compelled to select the next batch for processing uniformly at random to ensure that, in the limit, the gradient estimator is unbiased. If t happens to be a multiple of N_b, we can consider t as a number of epochs, each of which processes every batch exactly once. Thus the stochastic gradient is guaranteed to be an unbiased estimator which allows us to select a more appropriate distribution for batch ordering.

We can now marginalise out each of the hyperparameter variables to express $p(\theta \mid \mathcal{D}, t, \theta_0)$ as

$$\sum_{\boldsymbol{b} \in [N]^t} \frac{1}{N_b^t} p(\boldsymbol{b}) \sum_{\boldsymbol{i} \in [N_b]^t} \int_{\boldsymbol{\alpha}} \delta(\theta - A^t_{\boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{i}}(\theta_0) \mid \mathcal{D}) p(\boldsymbol{\alpha}) \mathrm{d}\boldsymbol{\alpha},$$

where $(\boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{i}) \in H^t$.

The class of optimisation algorithms Let $\mathcal{A} = \{A_{h_j}\}_j$ be a (countably infinite) set of optimisation algorithms A_{h_j} that, given an initial condition θ_0 and a point in the appropriate hyperparameter space $h_j \in H_j$, return a solution $A^t_{h_j}(\theta_0)$ to the given problem after t iterations. Let $\gamma_j \in \{0, 1\}$ be a weight associated to the output of algorithm A_{h_j} and assume that

$$\sum_j \gamma_j = 1. \tag{7}$$

Then, if we define

$$\theta_t = \sum_j \gamma_j A^t_{h_j}(\theta_0), \tag{8}$$

the output of the j-th algorithm is recovered by imposing $\gamma_j = 1$ (since, thanks to (7), $\gamma_i = 0$ for all $i \neq j$).

Let $H = \prod_j H_j$ be the hyperparameter space associated to $\mathcal{A}$. We define $\Gamma = \left\{\boldsymbol{\gamma} \in \prod_j \mathbb{Z}_2 \mid \sum \gamma_j = 1\right\}$ and let $\boldsymbol{A_h(\theta_0)} = [A^t_{h_j}(\theta_0)]_j \in \prod_j \mathbb{R}^d$. Then, for some $\boldsymbol{\gamma} \in \Gamma$, we can rewrite (8) as

$$\theta_t = \langle (\mathbf{1}_d \otimes \boldsymbol{\gamma}), \boldsymbol{A_h(\theta_0)} \rangle.$$

Then, we marginalise out $\boldsymbol{\gamma}$ and $\boldsymbol{h}$ to express $p(\theta \mid \mathcal{D}, t, \theta_0)$ as

$$\int_{(\boldsymbol{\gamma}, \boldsymbol{h})} \delta(\theta - \langle (\mathbf{1}_d \otimes \boldsymbol{\gamma}), \boldsymbol{A_h(\theta_0)} \rangle \mid \mathcal{D}) p(\boldsymbol{\gamma}, \boldsymbol{h}) \mathrm{d}(\boldsymbol{\gamma}, \boldsymbol{h}).$$

C. MC-dropout as model marginalisation

Given a particular $\theta \in \Theta$ and parametric model family $\mathcal{F}_\Theta$, let $\mathcal{M}_\theta \subset \mathcal{F}_\Theta$ be the parametric model family obtained by masking elements of θ. This is a finite set of models given by

$$\mathcal{M}_\theta = \{f_{\theta \odot \omega_1}, f_{\theta \odot \omega_2}, \dots, f_{\theta \odot \omega_{2^d}}\},$$

where the dropout masks $\omega_j, j = 1, \dots, d$ take values from $\{0, 1\}^d$ and $\odot$ is the Hadamard product. Masks ω can be drawn randomly with probability $p(\omega) = \Pi_{j=1}^d \mathbf{Bern}(p_j)$, $0 \leq p_j \leq 1$ and we can marginalise out ω to give

$$p(y \mid x, \mathcal{D}, \mathcal{M}_\theta) = \sum_\omega p(y \mid x, \mathcal{D}, \theta, \omega) p(\omega)$$

and then θ to give

$$\begin{aligned} p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta) &= \int_\theta p(y \mid x, \mathcal{D}, \mathcal{M}_\theta) \mathrm{d}\theta \\ &= \int_\theta \sum_\omega p(y \mid x, \mathcal{D}, \theta, \omega) p(\omega) \mathrm{d}\theta \\ &= \int_\theta \sum_\omega p(y \mid x, \theta) p(\theta \mid \mathcal{D}, \omega) p(\omega) \mathrm{d}\theta. \end{aligned}$$

Similar, we can condition on all the variables we have considered thus far to express $p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, t, h, \theta_0)$ as

$$\int_\theta \sum_\omega p(y \mid x, \theta) \underbrace{p(\theta \mid \mathcal{D}, t, h, \theta_0, \omega)}_{\delta(\theta - A_h^t(\theta_0) \odot \omega \mid \mathcal{D})} p(\omega) \mathrm{d}\theta.$$

The idea of marginalising out the entire class of parametric model families can be formulated in a similar way to the algorithm marginalisation strategy described in Section B.

D. One-shot estimation of the predictive probability distribution

In the main paper we showed how to compute statistics of the predictive probability distribution via Monte Carlo sampling. The main drawback of Monte Carlo sampling is the requirement to sample multiple times from the parameter distribution in order to compute $\hat{p}(y \mid x, \mathcal{D}, \mathcal{F}_\theta, *)$, which in turn, requires multiple forward passes through the network.

To overcome this issue, Assumed Density Filtering (ADF) can be used to perform a one-shot estimation, where we compute the expected value and variance of $p(y \mid x, \mathcal{D}, \mathcal{F}_\Theta, *)$ in a single forward pass. ADF has previously been used to propagate input uncertainty through the network to provide output uncertainty (Gast & Roth, 2018b), and to learn probability distributions over parameters (Ghosh et al., 2016) (a similar approach has been used in (Wu et al., 2018)). Also in (Brach et al., 2020), a one-shot approach has been proposed to approximate uncertainty for MC-dropout.

Let $Z^i = f_\theta^i(Z^{i-1})$ represent the output of the i-th layer of a feed-forward neural network (considered as a function between two random variables). We can convert the i-th layer into an uncertainty propagation layer by simply matching first and second-order central momenta (Minka, 2001), *i.e.*,

$$\mu_{Z^i} = \mathbb{E}_{Z^{i-1}, \Theta^i}[f^i(Z^{i-1}, \Theta^i)] \tag{9}$$
$$\sigma^2_{Z^i} = \mathbb{V}\mathrm{ar}_{Z^{i-1}, \Theta^i}[f^i(Z^{i-1}, \Theta^i)] \tag{10}$$

where $Z^{i-1} \sim \mathcal{N}(\mu_{Z^{i-1}}, \sigma_{Z^{i-1}})$ and $\Theta^i \sim \mathcal{N}(\mu_{\Theta^i}, \sigma_{\Theta^i})$. By doing so, the values of $\hat{\mu}_{\hat{Y}}$ and $\hat{\sigma}_{\hat{Y}^2}$ are obtained as the output of the last layer of the modified neural network. Notice that this procedure can account for input (aleatoric) uncertainty as long as the input $Z^0 \sim \mathcal{N}(\mu_{Z^0}, \sigma_{Z^0})$.

The main drawback with ADF is the reliance on a modification to the structure of the neural network, where each layer is replaced with its probabilistic equivalent. Also, while many probabilistic layers admit a closed form solution to compute (9), (10) (see below), in some cases approximation is needed.

D.1. Probabilistic layers

Linear layer Let X, W, B be independent random variables and consider a linear layer, $f(x, w, b) = wx + b$. Then

$$\mathbb{E}[f(X)] = \mathbb{E}[W]\mathbb{E}[X] + \mathbb{E}[B] \tag{11}$$
$$\begin{aligned}\mathbb{V}\mathrm{ar}[f(X)] &= \mathbb{V}\mathrm{ar}[WX] + \mathbb{V}\mathrm{ar}[B] \\ &= \mathbb{V}\mathrm{ar}[W]\mathbb{V}\mathrm{ar}[X] + \mathbb{V}\mathrm{ar}[W]\mathbb{E}[X]^2 \\ &\quad + \mathbb{E}[W]^2\mathbb{V}\mathrm{ar}[X] + \mathbb{V}\mathrm{ar}[B]\end{aligned} \tag{12}$$

In this case, if X, W, B are assumed to be normally distributed, $f(x, W, b)$ can be approximated by gaussian with the above statistics.

ReLU layer Given a random variable $X \sim \mathcal{N}(\mu, \sigma)$ the ReLU non-linearity activation function $f(x) = \max\{0, x\}$ leads to a closed form solution of its momenta (Frey & Hinton, 1999).

$$\mathbb{E}[f(X)] = \mu\Phi\left(\frac{\mu}{\sigma}\right) + \sigma\phi\left(\frac{\mu}{\sigma}\right) \tag{13}$$
$$\mathbb{V}\mathrm{ar}[f(X)] = (\mu^2 + \sigma^2)\Phi\left(\frac{\mu}{\sigma}\right) + \mu\sigma\phi\left(\frac{\mu}{\sigma}\right) - \mathbb{E}[f(X)]^2 \tag{14}$$

where ϕ is the standard normal distribution function

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$$

and Φ is the cumulative normal distribution function

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}t^2} \mathrm{d}t.$$

Other layers Almost all types of layers have closed form solutions or can be approximated. See (Gast & Roth, 2018a; Loquercio et al., 2020) for more details and (Brach et al., 2020) for dropout layers.