# Conditionally Gaussian PAC-Bayes

**Eugenio Clerico**[*] **George Deligiannidis** **Arnaud Doucet**

Department of Statistics, University of Oxford

## Abstract

Recent studies have empirically investigated different methods to train stochastic neural networks on a classification task by optimising a PAC-Bayesian bound via stochastic gradient descent. Most of these procedures need to replace the misclassification error with a surrogate loss, leading to a mismatch between the optimisation objective and the actual generalisation bound. The present paper proposes a novel training algorithm that optimises the PAC-Bayesian bound, without relying on any surrogate loss. Empirical results show that this approach outperforms currently available PAC-Bayesian training methods.

## 1 INTRODUCTION

Understanding generalisation for neural networks is among the most challenging tasks for learning theorists (Allen-Zhu et al., 2019; Kawaguchi et al., 2017; Neyshabur et al., 2017; Poggio et al., 2020; Zhang et al., 2021). Only a few of the theoretical tools developed in the literature can produce non-vacuous bounds on the error rates of over-parametrised architectures, and PAC-Bayesian bounds have proven to be among the tightest in the context of supervised classification (Ambroladze et al., 2007; Langford and Shawe-Taylor, 2003; McAllester, 2004). Several recent works have focused on algorithms aiming to minimise a generalisation bound for stochastic classifiers by optimising a PAC-Bayesian objective via stochastic gradient descent; see e.g. Alquier et al. (2016); Biggs and Guedj (2021); Clerico et al. (2021); Dziugaite and Roy (2017); Letarte et al. (2019); Pérez-Ortiz et al. (2021a,b); Zhou et al. (2019). Most of these studies use a surrogate loss to avoid dealing with the zero-gradient of the misclassification loss. However, there are exceptions, such as Biggs and Guedj (2021) and Clerico et al. (2021), which rely on the fact that an analytically tractable output distribution allows for an estimate of the misclassification error with a non-zero gradient with respect to the trainable parameters of the classifier.

Clerico et al. (2021) treat the case of a stochastic network with a single hidden layer. They prove a Central Limit Theorem (CLT) ensuring the convergence of the output distribution to a multivariate Gaussian, whose mean and covariance can be evaluated explicitly in terms of the network deterministic hyper-parameters. However, this result cannot be straightforwardly extended to the multilayer case, as the nodes of the deeper layers are not independent and so the CLT might not apply. Moreover, even assuming that the output is Gaussian, the computational cost of this method is prohibitive for deep architectures.

In Biggs and Guedj (2021), the focus is on a stochastic binary classifier whose output is of the form $\mathrm{sign}(w \cdot a)$, where $w$ is a Gaussian vector and $a$ is the output of the last hidden layer. The explicit form of the conditional expectation of the network's output (conditioned with respect to $a$) allows for a PAC-Bayesian training method applicable to arbitrarily deep networks. Nevertheless, this approach is only suitable for binary classification and cannot be easily extended to the multiclass case.

In the present work, we conjugate the two above ideas: in order to train the network with a method inspired by the Gaussian PAC-Bayesian approach from Clerico et al. (2021), we exploit the output's Gaussianity that can be obtained by conditioning on the previous layers, as in Biggs and Guedj (2021). This training procedure can be applied to a fairly general class of stochastic classifiers, overcoming some of the main limitations of the two aforementioned works, namely the single hidden layer and the binary classification setting. The

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

[*] Correspondence to: *clerico@stats.ox.ac.uk*

main requirement for our method to be valid is that the parameters of the last linear layer are independent Gaussian random variables. Additionally, as we are not relying on any CLT result to obtain the Gaussianity, we do not need the network to be very wide for the algorithm to work. Consequently, the approach we propose can be computationally much cheaper than the one from Clerico et al. (2021).

We empirically validate our training algorithm on MNIST and CIFAR10 for a range of architectures, testing both data-dependent and data-free PAC-Bayesian priors. We compare our results to those from Pérez-Ortiz et al. (2021a), as, to our knowledge, these are currently the tightest empirical PAC-Bayesian bounds available on these datasets. Our novel approach outperforms their standard PAC-Bayesian training methods in all our experiments.

## 2 BACKGROUND

### 2.1 PAC-Bayesian framework

In a standard classification problem, to each instance $x \in \mathcal{X} \subseteq \mathbb{R}^p$ corresponds a true label $y = f(x) \in \mathcal{Y} = \{1, \dots, q\}$. A training set $S = (X_k)_{k=1,\dots,m}$ is correctly labelled: for every $X_k \in S$ we have access to $Y_k = f(X_k)$. Each $X_k$ is an independent draw from a fixed probability measure $\mathbb{P}_X$ on $\mathcal{X}$, so that $S \sim \mathbb{P}_S = \mathbb{P}_X^{\otimes m}$. We consider a neural network, namely a parameterised function $F^\theta : \mathbb{R}^p \to \mathbb{R}^q$. For each input $x$, the network returns a prediction $\hat{y}$, defined as the largest output's node index:

$$\hat{y} = \hat{f}^\theta(x) = \mathrm{argmax}_{i \in \{1,\dots,q\}} F^\theta_i(x)\,.$$

The goal is to train the net to make good predictions, exploiting the information in $S$ to tune the parameters.

Define the misclassification loss as

$$\ell(\hat{y}, y) = \begin{cases} 0 & \text{if } y = \hat{y}\,; \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

For a given configuration $\theta$ of the network parameters, we call empirical error the empirical mean of the misclassification loss on the training sample: $\mathcal{E}_S(\theta) = \frac{1}{m}\sum_{x\in S} \ell(\hat{f}^\theta(x), f(x))$. This quantity can be explicitly evaluated, as we have access to the true labels on $S$. Therefore, it can be seen as an estimate for the true error $\mathcal{E}_\mathbb{P}(\theta) = \mathbb{E}_X[\ell(\hat{f}^\theta(X), f(X))] = \mathbb{P}_X(\hat{f}^\theta(X) \neq f(X))$, which in general cannot be computed exactly.

The PAC-Bayesian bounds are upper bounds on the true error, holding with high probability on the choice of the training sample $S$; see e.g. Alquier (2021); Catoni (2007); Guedj (2019); McAllester (1998, 1999). A main feature of the PAC-Bayesian framework is that it requires the network to be stochastic, that is we are dealing with architectures whose parameters $\theta$ are random variables.

Let us fix $\mathcal{P}$, a probability measure for the parameters $\theta$. We assume that $\mathcal{P}$ is data-independent, in the sense that it has to be selected without accessing the information in the training sample $S$. In line with most PAC-Bayesian literature, we will refer to $\mathcal{P}$ as the prior distribution. For a stochastic network, the training consists in efficiently modifying the distribution of $\theta$. This leads to a new distribution $\mathcal{Q}$ on the parameters, usually referred to as the posterior distribution. The main idea behind the PAC-Bayesian theory is that if the posterior $\mathcal{Q}$ is not "too far" from the prior $\mathcal{P}$, then the network should not be prone to overfitting. The essential tool to measure this "distance" between the prior and posterior distributions is the Kullback–Leibler divergence, defined as

$$\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) = \begin{cases} \mathbb{E}_{\theta\sim\mathcal{Q}}\left[\log \frac{\mathrm{d}\mathcal{Q}}{\mathrm{d}\mathcal{P}}(\theta)\right] & \text{if } \mathcal{Q} \ll \mathcal{P}\,; \\ +\infty & \text{otherwise.} \end{cases}$$

The PAC-Bayesian bounds are upper bounds on the expected value of the true classification error $\mathcal{E}_\mathbb{P}$ with respect to the posterior $\mathcal{Q}$. Two main ingredients constitute these bounds: the expected empirical error under $\mathcal{Q}$ and a complexity term, involving the divergence $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$. For simplicity, we will introduce the notations $\mathcal{E}_\mathbb{P}(\mathcal{Q}) = \mathbb{E}_{\theta\sim\mathcal{Q}}[\mathcal{E}_\mathbb{P}(\theta)]$ and $\mathcal{E}_S(\mathcal{Q}) = \mathbb{E}_{\theta\sim\mathcal{Q}}[\mathcal{E}_S(\theta)]$. The next proposition states some frequently used PAC-Bayes bounds (Langford and Seeger, 2001; Maurer, 2004; McAllester, 1999; Pérez-Ortiz et al., 2021a; Thiemann et al., 2017).

**Proposition 1.** *Fix $\delta \in (0,1)$, a data-independent prior $\mathcal{P}$, and a training set $S = (X_k)_{k=1,\dots,m}$ drawn according to $\mathbb{P}_S$. Define*

$$\mathtt{Pen} = \tfrac{1}{m}\left(\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \log \tfrac{2\sqrt{m}}{\delta}\right)\,; \tag{2}$$
$$\mathrm{kl}^{-1}(u|c) = \sup\{v \in [0,1] : \mathrm{kl}(u\|v) \leq c\}\,, \tag{3}$$

*where $\mathrm{kl}(u\|v)$ denotes the KL divergence between two Bernoulli distributions, with means $u$ and $v$ respectively. Then, with probability at least $1-\delta$ on the random draw of the training set, for any posterior $\mathcal{Q}$ each of the following quantities upper bounds $\mathcal{E}_\mathbb{P}(\mathcal{Q})$:*[1]

$$\mathcal{B}_1 = \mathrm{kl}^{-1}\left(\mathcal{E}_S(\mathcal{Q})|\,\mathtt{Pen}\right)\,; \tag{4a}$$
$$\mathcal{B}_2 = \mathcal{E}_S(\mathcal{Q}) + \sqrt{\mathtt{Pen}\,/2}\,; \tag{4b}$$
$$\mathcal{B}_3 = \left(\sqrt{\mathcal{E}_S(\mathcal{Q}) + \mathtt{Pen}\,/2} + \sqrt{\mathtt{Pen}\,/2}\right)^2\,; \tag{4c}$$
$$\mathcal{B}_4 = \inf_{\lambda\in(0,1)} \tfrac{1}{1-\lambda/2}\left(\mathcal{E}_S(\mathcal{Q}) + \mathtt{Pen}\,/\lambda\right)\,. \tag{4d}$$

[1]For (4a) we additionally assume that $S$ has size $m \geq 8$.

In the above proposition, the bound $\mathcal{B}_1$ is always the tightest. Moreover, all the above bounds are still valid if the empirical classification error $\mathcal{E}_S$ is replaced by the empirical average of any loss function $\tilde{\ell}$ in $[0,1]$.

So far, we have assumed the prior $\mathcal{P}$ to be data-independent. However, empirical evidence shows that using a data-dependent prior can lead to much tighter generalisation bounds, see e.g. Ambroladze et al. (2007); Dziugaite and Roy (2018); Dziugaite et al. (2021); Parrado-Hernández et al. (2012); Pérez-Ortiz et al. (2021b). Indeed, the actual requirement for the bounds (4) to hold is that $\mathcal{P}$ is independent of the sample $S$ used to evaluate $\mathcal{E}_S(\mathcal{Q})$. Hence, one can split the dataset $S$ into two disjoint sets, $S^{(1)}$ and $S^{(2)}$, use $S^{(1)}$ to train the prior, and obtain the data-dependent versions of the PAC-Bayesian bounds from Proposition 1, by redefining $\texttt{Pen} = (\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_{S^{(1)}}) + \log\frac{2\sqrt{m_2}}{\delta})/m_2$ and replacing $\mathcal{E}_S(Q)$ with $\mathcal{E}_{S^{(2)}}(\mathcal{Q})$. For instance (4a) becomes

$$\mathcal{E}_{\mathbb{P}}(\mathcal{Q}) \leq \mathrm{kl}^{-1}\left(\mathcal{E}_{S^{(2)}}(\mathcal{Q})\middle|\frac{\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_{S^{(1)}}) + \log\frac{2\sqrt{m_2}}{\delta}}{m_2}\right), \tag{5}$$

where $m_2 \geq 8$ is the size of $S^{(2)}$.

### 2.2 PAC-Bayesian training

Ideally, one would like to implement the following procedure (McAllester, 1998):

- Fix the PAC parameter $\delta \in (0,1)$ and a prior $\mathcal{P}$ for the network stochastic parameters;
- Collect a sample $S$ of $m$ iid data points, according to $\mathbb{P}_S = \mathbb{P}_X^{\otimes m}$, and label it correctly;
- Compute an optimal posterior $\mathcal{Q}$ minimising a generalisation bound, such as (4a);
- Implement a stochastic network whose random parameters have distribution $\mathcal{Q}$.

Unfortunately, in most realistic non-trivial scenarios, it can be extremely hard to compute and sample from an optimal posterior $\mathcal{Q}$ (Guedj, 2019). A possible approach consists in using Markov chain Monte Carlo (Alquier and Biau, 2013; Dalalyan and Tsybakov, 2012; Guedj and Alquier, 2013), sequential Monte Carlo or variational methods (Alquier et al., 2016), in order to approximately sample from the Gibbs posterior, which can be shown to be the optimal $\mathcal{Q}$ when the PAC-Bayesian bound is linear in the empirical loss (Catoni, 2007). However, these methods can often be inefficient, especially in the case of deep architectures and large datasets.

An alternative approach relies on simplifying the problem by constraining $\mathcal{P}$ and $\mathcal{Q}$ to belong to some simple distribution class. A common choice is to focus on the case of multivariate Gaussian distributions with diagonal covariance (Dziugaite and Roy, 2017; Pérez-Ortiz et al., 2021a): all the parameters are independent normal random variables. Conveniently, in this case the law of the random parameters can be easily expressed in terms of their means and standard deviations. These are deterministic trainable quantities that we will call hyper-parameters and denote by $\mathfrak{p}$. Furthermore, with this choice of $\mathcal{P}$ and $\mathcal{Q}$, the KL divergence between prior and posterior takes a simple closed-form. Denoting as $\mathfrak{m}$ and $\mathfrak{s}$ (resp. $\widetilde{\mathfrak{m}}$ and $\widetilde{\mathfrak{s}}$) the means and standard deviations of the posterior (resp. prior), we have

$$\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) = \frac{1}{2}\sum_k \frac{\mathfrak{s}_k^2 - \widetilde{\mathfrak{s}}_k^2}{\widetilde{\mathfrak{s}}_k^2} + \frac{1}{2}\sum_k\left(\frac{\mathfrak{m}_k - \widetilde{\mathfrak{m}}_k}{\widetilde{\mathfrak{s}}_k}\right)^2 + \sum_k \log\frac{\widetilde{\mathfrak{s}}_k}{\mathfrak{s}_k},$$

where the index $k$ runs over all the stochastic parameters of the networks.

Now, the idea is to tune the hyper-parameters $\mathfrak{p} = (\mathfrak{m}, \mathfrak{s})$ to minimise a PAC-Bayesian bound, such as (4a). A natural way to proceed is to perform a numerical optimisation via stochastic gradient descent, an approach originally proposed by Germain et al. (2009) and Dziugaite and Roy (2017), and referred to as PAC-Bayes with BackProp by Pérez-Ortiz et al. (2021a). First, we fix a PAC-Bayesian bound as our optimisation objective. As previously mentioned, this will be an expression involving a complexity term and the empirical error ($\texttt{Pen}$ and $\mathcal{E}_S(\mathcal{Q})$ respectively). We will hence denote it as $\mathcal{B}(\mathcal{E}_S(\mathcal{Q}), \texttt{Pen})$. Generally, an explicit form for $\mathcal{E}_S(\mathcal{Q})$ is not available, but sampling from $\mathcal{Q}$ easily provides an unbiased estimate $\hat{\mathcal{E}}_S(\mathcal{Q})$ of this quantity. However, we cannot perform a gradient descent step on $\mathcal{B}(\hat{\mathcal{E}}_S(\mathcal{Q}), \texttt{Pen})$. Indeed, $\hat{\mathcal{E}}_S(\mathcal{Q})$ has a null gradient almost everywhere, as it is the average over a finite set of realisations of the misclassification loss, which is constant almost everywhere (Pérez-Ortiz et al., 2021a). In order to overcome this problem, it is common to use a surrogate loss function (usually a bounded version of the cross-entropy) instead of the misclassification loss; see e.g. (Dziugaite and Roy, 2017) and (Pérez-Ortiz et al., 2021a,b). However, this creates a mismatch between the optimisation objective and the actual target bound.

It is worth noting that the zero-gradient problem is due to the particular form of the estimate $\hat{\mathcal{E}}_S(\mathcal{Q})$ and in general $\mathcal{E}_S(\mathcal{Q})$ has a non-zero gradient (Clerico et al., 2021). Indeed, as it will be shown in Section 3, a different choice of estimator for $\mathcal{E}_S(\mathcal{Q})$ can allow training the network without the use of any surrogate loss.

### 2.3 Stochastic network and notations

Consider a stochastic classifier featuring several hidden layers and a final linear layer. We denote $H(x)$ the output of the last hidden layer when the input is $x$, $\phi$ the activation function (here applied component-wise), and $W$ and $B$ the weight and bias of the linear output layer. The output of the network will be

$$F(x) = W\phi(H(x)) + B\,, \tag{6}$$

where we wrote $F$ instead of $F^\theta$ to simplify the notation. Since the network is stochastic, $W$, $B$, and $H(x)$ are random quantities. We denote $\mathcal{F}^{\mathcal{L}}$ the $\sigma$-algebra generated by the last layer's stochasticity, and $\mathcal{F}^{\mathcal{H}}$ the one due to the hidden layers.

We will henceforth assume the following:

- $\mathcal{F}^{\mathcal{L}} \perp\!\!\!\perp \mathcal{F}^{\mathcal{H}}$, that is the two $\sigma$-algebras are independent;
- $W$ and $B$ have independent normal components.

We can thus express the stochastic parameters of the last layer in terms of a set of deterministic trainable hyper-parameters $\mathfrak{m}$ and $\mathfrak{s}$:

$$W_{ij} = \zeta^W_{ij}\,\mathfrak{s}^W_{ij} + \mathfrak{m}^W_{ij}\,; \qquad B_i = \zeta^B_i\,\mathfrak{s}^B_i + \mathfrak{m}^B_i\,,$$

where the $\zeta$ are all independent standard normal random variables $\mathcal{N}(0,1)$.

For the hidden layers, we do not require any strong assumption: essentially, we need to be able to sample a realisation $h(x)$ of $H(x)$, to evaluate the KL divergence between prior and posterior, and to differentiate both KL and $h(x)$ with respect to the trainable deterministic hyper-parameters. However, for the sake of simplicity, in the rest of this paper we will assume that all the parameters of the hidden layers have independent normal laws, as in Clerico et al. (2021); Dziugaite and Roy (2017); Pérez-Ortiz et al. (2021a). All the architectures used for our experiments are indeed in this form. We refer to the supplementary material (Section SM3) for the extension of our results on more general architectures.

## 3 COND-GAUSS ALGORITHM

We present here a training procedure to optimise a PAC-Bayesian generalisation bound without the need for a surrogate loss. The two main ideas are the following:

- An unbiased estimate of $\mathcal{E}_S(\mathcal{Q})$ and its gradient can be evaluated if the output of the network is Gaussian, as in Clerico et al. (2021);
- If the parameters of the last layer are Gaussian, the output of the network is Gaussian as well when conditioned on the nodes of the last hidden layer, as pointed out by Biggs and Guedj (2021).

### 3.1 Gaussian output

Fix an input $x$ and assume that the network's output $F(x)$ follows a multivariate normal distribution, with mean vector $M(x)$ and covariance matrix $Q(x)$. For our purposes, we can suppose that $Q(x)$ is diagonal, meaning that the components of the output are mutually independent (we refer to Section 4.1 in Clerico et al. (2021) for the discussion of the general case). Let us denote $V(x)$ the diagonal of $Q(x)$, consisting of the output's variances, so that

$$\mathbb{E}_{\mathcal{Q}}[F_i(x)] = M_i(x)\,; \qquad \mathbb{V}_{\mathcal{Q}}[F_i(x)] = V_i(x)\,.$$

The stochastic prediction of our classifier is $\hat{y} = \hat{f}(x) = \operatorname{argmax}_{i\in\{1,\dots,q\}} F_i(x)$. In order to compute $\mathcal{E}_S(\mathcal{Q})$, for each input $x \in S$ we shall evaluate $\mathbb{E}_{\mathcal{Q}}[\ell(\hat{f}(x), f(x))]$. As $\ell$ is the misclassification loss (1), this is simply the probability of making a mistake for the input $x$. Letting $y = f(x)$ and $\hat{y} = \hat{f}(x)$, we have

$$\mathbb{E}_{\mathcal{Q}}[\ell(\hat{y}, y)] = \mathbb{P}_{\mathcal{Q}}(\hat{y} \neq y) = \mathbb{P}_{\mathcal{Q}}\left(F_y(x) \leq \max_{i\neq y} F_i(x)\right). \tag{7}$$

In the case of binary classification, the above expression has a simple closed-form. Indeed, if we consider for instance the case $y = 1$, we have

$$\begin{aligned}\mathbb{P}_{\mathcal{Q}}(\hat{y} \neq 1) &= \mathbb{P}_{\mathcal{Q}}(F_2(x) - F_1(x) \geq 0)\\ &= \mathbb{P}\left(\zeta \leq \frac{M_2(x) - M_1(x)}{\sqrt{V_1(x) + V_2(x)}}\right),\end{aligned}$$

where $\zeta \sim \mathcal{N}(0,1)$. This can be expressed in terms of the error function erf, as the cumulative distribution function of a standard normal is given by $\psi(u) = \mathbb{P}(\zeta \leq u) = \frac{1}{2}(1 + \operatorname{erf}(u/\sqrt{2}))$. Notice that the above expression no longer suffers from vanishing gradients, as $\psi' \neq 0$.

For multiple classes ($q > 2$), (7) does not have a simple closed-form. However, we can easily find Monte Carlo estimators that also bring unbiased estimates for the gradient with respect to $M$ and $Q$.

**Proposition 2.** *Denote the cumulative distribution function of a standard normal random variable as $\psi : u \mapsto \frac{1}{2}(1 + \operatorname{erf}(u/\sqrt{2}))$. Fix $x$, let $y$ be its true label, and $\hat{y}$ the network's stochastic prediction. Define*

$$L_1 = \psi\left(\max_{i\neq y} \frac{F_i(x) - M_y(x)}{\sqrt{V_y(x)}}\right);$$

$$L_2 = 1 - \prod_{i\neq y} \psi\left(\frac{F_y(x) - M_i(x)}{\sqrt{V_i(x)}}\right),$$

*where $F(x) \sim \mathcal{N}(M(x), \mathrm{diag}(V(x)))$. Then*

$$\mathbb{E}_{\mathcal{Q}}[L_1] = \mathbb{E}_{\mathcal{Q}}[L_2] = \mathbb{P}_{\mathcal{Q}}(\hat{y} \neq y)\,,$$
$$\mathbb{E}_{\mathcal{Q}}[\nabla L_1] = \mathbb{E}_{\mathcal{Q}}[\nabla L_2] = \nabla \mathbb{P}_{\mathcal{Q}}(\hat{y} \neq y)\,,$$

*where the gradient is with respect to all the components of $M(x)$ and $V(x)$.*
*In particular, by sampling realisations of $L_1$ or $L_2$, we can get unbiased Monte Carlo estimators of the misclassification loss and its gradient.*

## 3.2 Conditional Gaussianity

In practice, the output of a stochastic network is generally not Gaussian. However, we can overcome this issue by conditioning on the hidden layers, similarly to what was done by Biggs and Guedj (2021).

Recall that the network's output is given by (6):

$$F = W\phi(H) + B\,,$$

where the explicit dependence of $H$ on $x$ is omitted to make the notations lighter. Conditioned on the stochasticity of the hidden layers $\mathcal{F}^{\mathcal{H}}$, $F$ follows a normal multivariate distribution, as

$$F = W\phi(H) + B \sim \mathcal{N}(M(H), Q(H))\,.$$

We can easily evaluate $M(H)$ and $Q(H)$ in terms of $\mathfrak{m}$ and $\mathfrak{s}$. We have

$$\begin{aligned} M_i(H) = \mathbb{E}_{\mathcal{Q}}[F_i|\mathcal{F}^{\mathcal{H}}] &= \sum_j \mathbb{E}_{\mathcal{Q}}[W_{ij}]\phi(H_j) + \mathbb{E}_{\mathcal{Q}}[B_i] \\ &= \sum_j \mathfrak{m}^W_{ij}\phi(H_j) + \mathfrak{m}^B_i \end{aligned}$$

and $Q_{ij}(H) = \delta_{ij}V_i(H)$, with

$$\begin{aligned} V_i(H) = \mathbb{V}_{\mathcal{Q}}[F_i|\mathcal{F}^{\mathcal{H}}] &= \sum_j \mathbb{V}_{\mathcal{Q}}[W_{ij}]\phi(H_j)^2 + \mathbb{V}_{\mathcal{Q}}[B_i] \\ &= \sum_j (\mathfrak{s}^W_{ij}\phi(H_j))^2 + (\mathfrak{s}^B_i)^2\,. \end{aligned}$$

Finally, we note that by iterated expectations

$$\mathbb{E}_{\mathcal{Q}}[\ell(\hat{f}(x), f(x))] = \mathbb{E}_{\mathcal{Q}}[\mathbb{E}_{\mathcal{Q}}[\ell(\hat{f}(x), f(x))|\mathcal{F}^{\mathcal{H}}]]\,.$$

In particular, if we draw the hidden parameters and get a realisation $h$ of $H$, we obtain an unbiased estimate $\frac{1}{m}\sum_{x\in S}\mathbb{E}[\ell(\hat{f}(x), f(x))|H(x) = h(x)]$ of $\mathcal{E}_S(\mathcal{Q})$, where each term $\mathbb{E}[\ell(\hat{f}(x), f(x))|H(x) = h(x)]$ can be estimated with the methods from Section 3.1, since $F(x)$ is a multivariate Gaussian when conditioned on $H(x) = h(x)$.

## 3.3 Training algorithm

We sketch here the Cond-Gauss training algorithm. First, we fix a PAC-Bayesian bound $\mathcal{B}$ as the optimisation objective. Then, we initialise the deterministic hyper-parameters of our network, and we select this configuration as the prior. Finally, we split our dataset into batches $S_1, \dots, S_K$. To train the network, we iterate over the batches and, similarly to what is done in most PAC-Bayesian training methods based on stochastic gradient descent, we sample the network's parameters at each batch iteration. However, we only perform this sampling for the hidden layers and not for the final linear layer. In this way, for each $x$ in the batch, we have a realisation $h(x)$ of the last hidden layer's output. Conditioned on $H = h$, the output is

**Algorithm 1** Cond-Gauss PAC-Bayesian training

```
Require:
    $\tilde{\mathfrak{p}} = (\tilde{\mathfrak{p}}^{\mathcal{H}}, \tilde{\mathfrak{p}}^{\mathcal{L}})$        ▷ Initial hyper-parameters (defining the prior)
    $S$        ▷ Training set of size $\#S$
    $\delta \in (0,1)$        ▷ PAC parameter
    $\eta, T$        ▷ Learning rate and number of epochs
Ensure:
    Optimal $\mathfrak{p}$ parameterizing the posterior

1:  procedure Cond-Gauss
2:      $\mathfrak{p}^{\mathcal{H}} \leftarrow \tilde{\mathfrak{p}}^{\mathcal{H}}$
3:      $\mathfrak{p}^{\mathcal{L}} = (\mathfrak{m}, \mathfrak{s}) \leftarrow \tilde{\mathfrak{p}}^{\mathcal{L}}$
4:      for $t \leftarrow 1 : T$ do
5:          Sample $\theta^{\mathcal{H}} \sim \mathcal{Q}^{\mathcal{H}}_{\mathfrak{p}^{\mathcal{H}}}$        ▷ Sample the parameters of the hidden layers
6:          $h = h(S, \theta^{\mathcal{H}})$        ▷ Evaluate the last hidden layer's output for all $x \in S$
7:          $M = M(h, \mathfrak{m}) = \mathfrak{m}^W\phi(h) + \mathfrak{m}^B$        ▷ Evaluate the conditional mean of the output
8:          $V = V(h, \mathfrak{s}) = (\mathfrak{s}^W\phi(h))^2 + (\mathfrak{s}^B)^2$        ▷ Evaluate the conditional variance of the output
9:          $\hat{\mathcal{E}}_S(\mathcal{Q}_{\mathfrak{p}}) = \mathcal{E}(M, V)$        ▷ Evaluate $\hat{\mathcal{E}}_S(\mathcal{Q}_{\mathfrak{p}})$ from $M$ and $V$ as in Section 3.1
10:         $\hat{\mathcal{B}} = \mathcal{B}(\hat{\mathcal{E}}_S(\mathcal{Q}_{\mathfrak{p}}), \mathtt{Pen})$        ▷ Evaluate the estimate $\hat{\mathcal{B}}$ of the PAC-Bayesian objective $\mathcal{B}$
11:         $\mathfrak{p} \leftarrow \mathfrak{p} - \eta\nabla_{\mathfrak{p}}\hat{\mathcal{B}}$        ▷ Perform the gradient step
12:     return $\mathfrak{p}$
```

Gaussian and we can proceed as discussed earlier to get an estimate $\hat{\mathcal{E}}_{S_k}(\mathcal{Q}, h)$ of $\mathcal{E}_S(\mathcal{Q})$. After that, we can obtain an estimate $\hat{\mathcal{B}}$ of the target bound $\mathcal{B}$, by replacing $\mathcal{E}_S(\mathcal{Q})$ with $\hat{\mathcal{E}}_{S_k}(\mathcal{Q}, h)$. Finally, we compute the gradient of $\hat{\mathcal{B}}$ with respect to the trainable hyper-parameters, and we perform the gradient step.

If we want to use a data-dependent prior, we simply split the dataset into two subsets $S^{(1)}$ and $S^{(2)}$, and then use $S^{(1)}$ to learn $\mathcal{P}$. For instance, we might train the prior using $\hat{\mathcal{E}}_{S^{(1)}}(\mathcal{Q})$ as optimisation objective or tuning only the prior's means by treating the network as if it was deterministic, similarly to what was done in Pérez-Ortiz et al. (2021a). Once the prior's training is complete, we perform the Cond-Gauss algorithm, replacing $S$ with $S^{(2)}$.

The training procedure is summarised in Algorithm 1, where, for the sake of simplifying the notation, it is assumed that the whole training set forms a single batch. For convenience, we introduce the superscripts $^{\mathcal{H}}$ and $^{\mathcal{L}}$ to refer to the hidden layers and the last layer, respectively. Thus, we denote as $\theta = (\theta^{\mathcal{H}}, \theta^{\mathcal{L}})$ the random parameters of the network, where $\theta^{\mathcal{H}}$ are the parameters in the hidden layers, while $\theta^{\mathcal{L}} = (W, B)$ are those of the last layer. Similarly, $\mathfrak{p}^{\mathcal{H}}$ are the deterministic hyper-parameters relative to the hidden layers, whilst $\mathfrak{p}^{\mathcal{L}} = (\mathfrak{m}, \mathfrak{s})$ are those of the last layer. We introduced the subscript $_{\mathfrak{p}}$ for the posterior $\mathcal{Q}$, to stress the fact that it is determined by the hyper-parameters, and we denoted by $\mathcal{Q}^{\mathcal{H}}$ the marginal posterior distribution for the hidden layers. Finally, the tilde notation represents the values at initialisation.

As a final remark, $\mathrm{kl}^{-1}$ is currently not implemented in most of the standard deep learning libraries. Yet, it can be easily computed numerically with few iterations of Newton's method, as in Dziugaite and Roy (2017). Nevertheless, most of the empirical studies on PAC-Bayesian gradient descent optimisation (see e.g. Dziugaite and Roy (2017) and Pérez-Ortiz et al. (2021a)), do not use as objective (4a), in order to avoid computing $\nabla\mathrm{kl}^{-1}$. However, since this gradient can be expressed as a function of $\mathrm{kl}^{-1}$ itself, we were able to optimise (4a) in our experiments (see Section SM4 in the supplementary material for further details).

## 3.4 Unbiasedness of the estimates

One might wonder whether the estimates of $\mathcal{B}$ and its gradient are actually unbiased. Notably, this is indeed the case if the chosen PAC-Bayesian objective $\mathcal{B}$ is an affine function of the empirical error, as (4b) and (4d).

**Proposition 3.** *Assume that $\mathcal{B}$ is locally Lipschitz in the hidden stochastic parameters $\theta^{\mathcal{H}}$, and that $\nabla_{\theta^{\mathcal{H}}}\mathcal{B}$ is polynomially bounded.*[2] *If $\mathcal{B}(\mathcal{E}_S(\mathcal{Q}), \mathtt{Pen})$ is affine in $\mathcal{E}_S(\mathcal{Q})$, then we have $\mathbb{E}[\hat{\mathcal{B}}] = \mathcal{B}$ and $\mathbb{E}[\nabla\hat{\mathcal{B}}] = \nabla\mathcal{B}$, the gradient being with respect to the trainable hyper-parameters $\mathfrak{p}$.*

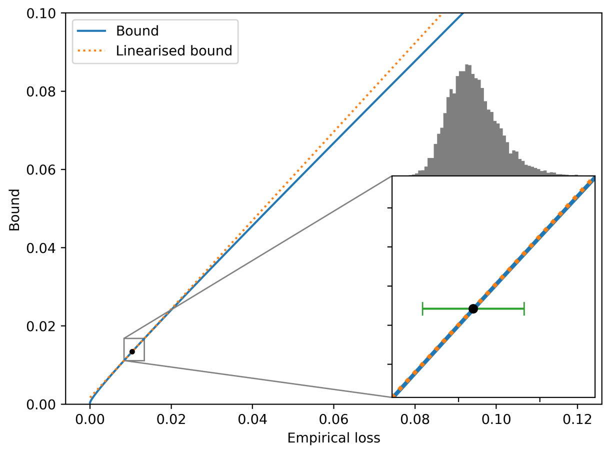

**Figure 1:** Experimental evidence that the bound (4a) is almost affine in the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates. The network used was the one achieving the best generalisation bound in our experiment on MNIST with *data-dependent* priors. 10000 realisations of $\hat{\mathcal{E}}_S(\mathcal{Q})$ were sampled. Their distribution is summarized by the histogram above the zoomed portion of the plot. The black dot is the bound for the average value found for $\hat{\mathcal{E}}_S(\mathcal{Q})$, while the green error bar has a total width of 4 empirical standard deviations. In the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates, the bound and its linearised version almost coincide. Along the green error bar, the bound's slope has a relative variation of $\pm 0.8\%$.

Although this unbiasedness property does not hold for objectives not affine in $\mathcal{E}_S(\mathcal{Q})$, if $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates enough around $\mathcal{E}_S(\mathcal{Q})$ we can linearise $\hat{\mathcal{B}}$ as

$$\hat{\mathcal{B}} \simeq \mathcal{B} + (\hat{\mathcal{E}}_S(\mathcal{Q}) - \mathcal{E}_S(\mathcal{Q}))\, \partial_{\mathcal{E}}\mathcal{B}\,.$$

Then, both $\hat{\mathcal{B}}$ and $\nabla\hat{\mathcal{B}}$ are essentially almost unbiased estimates. Considering the good performance of our method in the experiments we ran, we conjecture that this is indeed what happens in practice with (4a) and (4c). Figure 1 gives some empirical support to this hypothesis. We refer the reader to the supplementary material (Section SM2) for additional discussion and empirical evidence on this subject.

## 3.5 Final evaluation of the bound

In order to evaluate the final generalisation bound, we need the exact value of $\mathcal{E}_S(\mathcal{Q})$ once the training is complete. As this cannot be computed, we use an empirical upper bound, as done for instance in (Dziugaite and Roy, 2017).

[2]These are mild technical assumptions, verified in all the experimental settings considered in this paper.

Let $\theta_1, \dots, \theta_N$ be $N$ independent realisations of the whole set of the network stochastic parameters, drawn according to $\mathcal{Q}$. An unbiased Monte Carlo estimator of $\mathcal{E}_S(\mathcal{Q})$ is simply given by

$$\widetilde{\mathcal{E}}_S(\mathcal{Q}) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}_S(\theta_n)\,.$$

As shown by Langford and Caruana (2002), for fixed $\delta' \in (0,1)$, with probability at least $1-\delta'$ we have,

$$\mathcal{E}_S(\mathcal{Q}) \leq \mathrm{kl}^{-1}\left(\widetilde{\mathcal{E}}_S(\mathcal{Q}) \middle| \tfrac{1}{N}\log\tfrac{2}{\delta'}\right)\,,$$

where $\mathrm{kl}^{-1}$ is defined in (3). We conclude from Proposition 1 that, with probability higher than $1-(\delta+\delta')$, we have

$$\begin{aligned}&\mathcal{E}_{\mathbb{P}}(\mathcal{Q})\\ &\leq \mathrm{kl}^{-1}\left(\mathrm{kl}^{-1}\left(\widetilde{\mathcal{E}}_S(\mathcal{Q})\middle|\tfrac{1}{N}\log\tfrac{2}{\delta'}\right)\middle|\tfrac{\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\log\frac{2\sqrt{m}}{\delta}}{m}\right)\,,\end{aligned} \tag{8}$$

as $\mathrm{kl}^{-1}$ is an increasing function of its first argument.

## 4 NUMERICAL RESULTS

We tested the Cond-Gauss algorithm empirically on the MNIST and the CIFAR10 datasets (Deng, 2012; Krizhevsky, 2009). In the literature, several works benchmark various PAC-Bayesian algorithms on these and other datasets (Biggs and Guedj, 2021; Clerico et al., 2021; Dziugaite and Roy, 2017, 2018; Letarte et al., 2019; Pérez-Ortiz et al., 2021a,b). To our knowledge, in the case of over-parameterised deep neural networks, the bounds from Pérez-Ortiz et al. (2021a) are currently the tightest on both MNIST and CIFAR10. Thus, in order to assess our Cond-Gauss method by comparing their results with ours, we decided to mimic some of their multilayer convolutional architectures[3], although our training schedules, as well as the prior's training procedures and the choice of initial variances, differed from theirs. All the generalisation bounds obtained with our training algorithm were tighter than those reported by Pérez-Ortiz et al. (2021a).

We illustrate below some of our main empirical results. All the final generalisation bounds are obtained from (8), or its natural variant based on (5) for data-dependent priors. We always use $\delta = 0.025$ and $\delta' = 0.01$ as in Pérez-Ortiz et al. (2021a), so that the final generalisation bounds hold with probability greater or equal to 0.965. For all the bounds but those in Figure 2, we fixed $N = 150000$ as in Pérez-Ortiz et al. (2021a).

We refer to Section SM5 in the supplementary material for the full results and the missing experimental details. The PyTorch code developed for this paper is available at `https://github.com/eclerico/CondGauss`.

### 4.1 MNIST

For our experiments on MNIST, we only used the standard training dataset (60000 labelled examples) for the training procedure. We tested a 4-layer ReLU stochastic network, whose parameters were independent Gaussians. The architecture was composed of two convolutional layers followed by two linear layers.

We first experimented on data-free priors. We compared the performance of the standard PAC-Bayes with BackProp training algorithm (S), where the misclassification loss is replaced by a bounded version of the cross-entropy loss as in Pérez-Ortiz et al. (2021a), and the Cond-Gauss algorithm (G). We used the four training objectives from (4):

$$\begin{aligned}&\texttt{invKL}: && \mathrm{kl}^{-1}(\mathcal{E}_S(\mathcal{Q})|\texttt{Pen}_\kappa)\,;\\ &\texttt{McAll}: && \mathcal{E}_S(\mathcal{Q}) + \sqrt{\texttt{Pen}_\kappa/2}\,;\\ &\texttt{quad}: && (\sqrt{\mathcal{E}_S(\mathcal{Q}) + \texttt{Pen}_\kappa/2} + \sqrt{\texttt{Pen}_\kappa/2})^2\,;\\ &\texttt{lbd}: && \tfrac{1}{1-\lambda/2}(\mathcal{E}_S(\mathcal{Q}) + \texttt{Pen}_\kappa/\lambda)\,,\end{aligned}$$

where the KL penalty is defined as

$$\texttt{Pen}_\kappa = \frac{\kappa}{m}\left(\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \log\frac{2\sqrt{m}}{\delta}\right)\,. \tag{9}$$

The factor $\kappa$ in (9) can increase or reduce the weight of the KL term during the training. For the last objective, `lbd`, the parameter $\lambda$ takes values in $(0,1)$ and is optimised during training, similarly to what was done in Pérez-Ortiz et al. (2021a).[4]

The network was trained via SGD with momentum. During training, at the end of each epoch, we kept track of the bound (4a)'s empirical value to pick the best epoch at the end of the training.

In Figure 2 we report the values of the bounds for different training settings with data-free priors on MNIST. As evaluating the true bound via (8) can be extremely time-consuming when $N = 150000$, the values reported in the figure are obtained for $N = 10000$.

[3]The only difference between their architectures and ours is that we sometimes swapped the order between the application of the activation function and the max pooling. This fact was merely accidental, but we believe that it did not significantly affect our results.

[4]In our experiments, we initialised $\lambda$ at 0.5 and then doubled the number of epochs, alternating one epoch of $\lambda$'s optimisation with one of optimisation for $\mathfrak{m}$ and $\mathfrak{s}$.

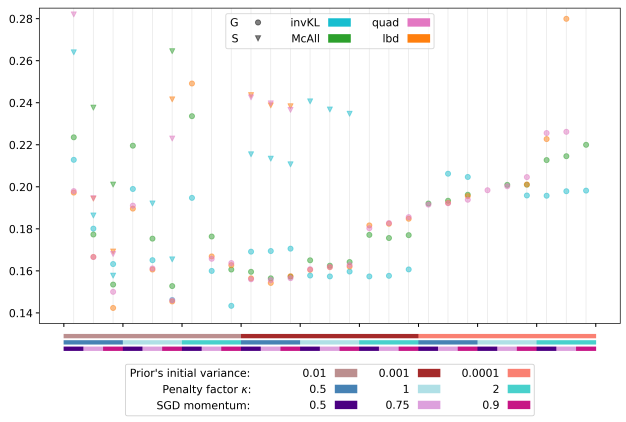

**Figure 2:** Results for MNIST with random prior. Each dot is the PAC-Bayesian bound obtained via (8) with $N = 10000$. The marker shape represents the training method (in the legend, 'G' stands for our method, 'S' for the standard one), and the colour represents the training objective. Different columns indicate different momentum values, penalty factor $\kappa$, and initial variance for the prior. The initial prior's means were the same for all the different training possibilities. The values higher than 0.285 are not reported.

The Cond-Gauss algorithm always achieved the best performance. Note that some of the bounds in the figure are substantially tighter than the best value reported in Pérez-Ortiz et al. (2021a), namely .2165. Perhaps unexpectedly, this is sometimes the case even for the standard PAC-Bayes with BackProp algorithm, although it happened for training settings that were not tried therein, namely with the `invKL` objective or $\kappa = 0.5$.

In Table 1 we report the final generalisation bounds with $N = 150000$, evaluated via (8). For each method, we selected the training achieving the best bound in Figure 2. The Cond-Gauss procedure achieved better results than the standard algorithm with all the objectives. Quite surprisingly, the tightest bound was achieved by the `lbd` objective. The column 'emp err' reports the empirical error on the training dataset, obtained when computing the final bounds. The test errors provided in the column 'test err' are evaluated on the standard held-out test dataset of MNIST, by averaging over 1000 realisations of the random network's parameter. We also report the empirical standard deviation of this estimate. Interestingly, the test error on the held-out dataset often resulted smaller than the empirical error on the training dataset. We do not have an explaination for this fact, which might be a mere coincidence and did not occur in most of the experiments with data-dependent priors.

**Table 1:** PAC-Bayesian bounds for MNIST - data-free prior

| method | emp err | test err | `Pen` (2) | bound |
|---|---|---|---|---|
| S `McAll` | .0670 | $.0900_{\pm.0047}$ | .0320 | .1916 |
| S `lbd` | .0636 | $.0623_{\pm.0013}$ | .0413 | .1606 |
| S `quad` | .0622 | $.0577_{\pm.0031}$ | .0420 | .1594 |
| S `invKL` | .0438 | $.0407_{\pm.0022}$ | .0560 | **.1495** |
| G `McAll` | .0472 | $.0435_{\pm.0024}$ | .0477 | .1446 |
| G `lbd` | .0279 | $.0272_{\pm.0016}$ | .0669 | **.1348** |
| G `quad` | .0399 | $.0374_{\pm.0021}$ | .0518 | .1380 |
| G `invKL` | .0356 | $.0340_{\pm.0019}$ | .0556 | .1355 |

For the data-dependent priors, we used 50% of the dataset to train $\mathcal{P}$ and the remaining 50% to train $\mathcal{Q}$. We always used the Cond-Gauss algorithm for both prior and posterior. All the posteriors were trained with the `invKL` objective and $\kappa = 1$, whilst for the prior, we experimented with different objectives, penalty factors $\kappa$, and dropout values. The final best generalisation bound was .0144, about 7% better than the tightest one from Pérez-Ortiz et al. (2021a) for the same architecture, .0155. However, it is interesting to note that the role of the posterior's training seems to be quite marginal, as, in our experiments, the prior already achieved a quite low empirical error on the posterior's dataset, .0108, which could be improved only to .0104 by tuning the posterior. The results of the whole experiment can be found in Table SM1 in the supplementary material.

### 4.2 CIFAR10

As we had done for the MNIST dataset, for CIFAR10 we used only the standard training dataset (50000 labelled images) for the training procedure. We trained a 9-layer architecture (6 convolutional + 3 linear layers) and a 15-layer architecture (12 convolutional + 3 linear layers). We experimented with data-dependent priors only, training $\mathcal{P}$ with 50% of the data for the 9-layer classifier, and with both 50% and 70% of the data in the case of the 15-layer one.

The results for the 9-layer architecture are reported in Table 2. Note that the best bound that we obtained in this setting was .2066, a result much tighter than the one reported by Pérez-Ortiz et al. (2021a), .2901. After some preliminary experiments, we chose to train both priors and posteriors via the Cond-Gauss algorithm with the `invKL` objective. We used a small factor $\kappa$ for the prior to avoid regularising too much, whilst $\kappa$ was 1 for the posterior. We tried different values for the dropout and the factor $\kappa$ in the training of the prior, as reported in Table 2. We trained via SGD with momentum for both prior and posterior. For $\mathcal{P}$, we used a schedule much longer than the one usually chosen for the prior in the literature. Essentially, this is because we were not just training the means of the

**Table 2:** CIFAR10 - 9 layers - Prior learnt on 50% of the dataset

| Prior | | | | | | | Posterior | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tm[a] | do[b] | pf[c] | iv[d] | l1[e] | l2[f] | p[g] | tm[a] | l1[e] | l2[f] | t[h] | p[g] | b[i] |
| G `invKL` | 0 | .01 | .001 | .0196 | .2233 | 3.778 | G `invKL` | .0196 | .2211 | $.2251_{\pm.0021}$ | 4.696 | .2376 |
| G `invKL` | 0 | .005 | .001 | .1127 | .2797 | 3.778 | G `invKL` | .1126 | .2782 | $.2814_{\pm.0019}$ | 4.319 | .2953 |
| G `invKL` | .1 | .01 | .001 | .0536 | .1930 | 3.778 | G `invKL` | .0536 | .1912 | $.1952_{\pm.0020}$ | 4.484 | **.2066** |
| G `invKL` | .1 | .005 | .001 | .0266 | .1930 | 3.778 | G `invKL` | .0266 | .1913 | $.1933_{\pm.0019}$ | 4.520 | .2067 |

[a] tm: Training method.
[b] do: Dropout probability for the prior's training.
[c] pf: Penalty factor $\kappa$ for the prior's training objective.
[d] iv: Initial value of the prior's variances.
[e] l1: Empirical error estimate on the prior dataset.
[f] l2: Empirical error estimate on the posterior dataset.
[g] t: Test error$_{\pm\text{standard deviation}}$ (from 1000 realisations).
[h] p: KL penalty `Pen` (2) in $10^{-4}$ units.
[i] b: Final PAC-Bayesian bound.

random parameters, but the variances as well. However, in this way we could already obtain for the priors competitive empirical errors on the posterior's dataset. Like with the MNIST dataset, the improvement due to the posterior's training was minimal.

For the 15-layer architecture, the full results and details are reported in Table SM2 in the supplementary material. Quite interestingly, to train $\mathcal{P}$, it was necessary to introduce an initial pre-training for the prior's means, as the Cond-Gauss algorithm alone could not significantly decrease the training objective. First, we initialised the means with an orthogonal initialisation, as suggested in Hu et al. (2020). Then we optimised them by training a deterministic network (with the same architecture) using the cross-entropy loss on the prior's dataset. Finally, via the Cond-Gauss algorithm, we completed the prior's training and proceeded with the posterior's tuning. The best final bounds obtained were .1855, with the prior learnt on 50% of the dataset, and .1595, when 70% of the dataset was used to train $\mathcal{P}$. Again, these values are tighter than those from Pérez-Ortiz et al. (2021a).

### 4.3 Summary

To summarise our results, Table 3 compares our best PAC-Bayesian generalisation bounds with those from Pérez-Ortiz et al. (2021a). The column 'C-G' features the best bounds we could obtain with the Cond-Gauss algorithm in our experiments. The figures in the column 'P-O' are the tightest bounds reported in Pérez-Ortiz et al. (2021a) for the same architectures and datasets. All the PAC-Bayesian generalisation bounds in the table hold with probability at least 0.965 on the choice of the training dataset.

**Table 3:** Comparison of our PAC-Bayesian bounds with those from Pérez-Ortiz et al. (2021a)

| dataset | architecture | prior | C-G | P-O |
|---|---|---|---|---|
| MNIST | 4 layers | data-free | .1348 | .2165 |
| MNIST | 4 layers | 50% | .0144 | .0155 |
| CIFAR10 | 9 layers | 50% | .2066 | .2901 |
| CIFAR10 | 15 layers | 50% | .1855 | .1954 |
| CIFAR10 | 15 layers | 70% | .1595 | .1667 |

## 5 CONCLUSION

We have introduced the Cond-Gauss training algorithm, which allows the optimisation of PAC-Bayesian bounds without relying on the use of a surrogate loss. Taking an estimate of the actual target bound as the optimisation objective is a natural choice. As confirmed by our experiments on the MNIST and the CIFAR10 classification tasks, it also leads to tighter bounds than the current state-of-the-art bounds obtained via PAC-Bayes with BackProp.

### Acknowledgements

The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work.[5] Eugenio Clerico is partly supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through the grant EP/R513295/1 (DTP scheme). Arnaud Doucet is partly supported by the EPSRC grant EP/R034710/1. He also acknowledges the support of the UK Defence Science and Technology Laboratory (DSTL) and EPSRC under grant EP/R013616/1. This is part of the collaboration between US DOD, UK MOD, and UK EPSRC, under the Multidisciplinary University Research Initiative.

[5] `http://dx.doi.org/10.5281/zenodo.22558`

## References

Z. Allen-Zhu, Y. Li, and Y. Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *NeurIPS*, 2019.

P. Alquier. User-friendly introduction to PAC-Bayes bounds. *arXiv:2110.11216*, 2021.

P. Alquier and G. Biau. Sparse single-index model. *Journal of Machine Learning Research*, 14, 2013.

P. Alquier, J. Ridgway, and N. Chopin. On the properties of variational approximations of Gibbs posteriors. *Journal of Machine Learning Research*, 17, 2016.

A. Ambroladze, E. Parrado-Hernández, and J. Shawe-Taylor. Tighter PAC-Bayes bounds. *NeurIPS*, 2007.

F. Biggs and B. Guedj. Differentiable PAC-Bayes objectives with partially aggregated neural networks. *Entropy*, 23(10), 2021.

O. Catoni. PAC-Bayesian supervised classification: The thermodynamics of statistical learning. *IMS Lecture Notes Monograph Series*, 2007.

E. Clerico, G. Deligiannidis, and A. Doucet. Wide stochastic networks: Gaussian limit and PAC-Bayesian training. *arXiv:2106.09798*, 2021.

A. S. Dalalyan and A. B. Tsybakov. Sparse regression learning by aggregation and Langevin Monte-Carlo. *Journal of Computer and System Sciences*, 78 (5), 2012.

L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6), 2012.

G. K. Dziugaite and D. M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *UAI*, 2017.

G. K. Dziugaite and D. M. Roy. Data-dependent PAC-Bayes priors via differential privacy. *NeurIPS*, 2018.

G. K. Dziugaite, K. Hsu, W. Gharbieh, G. Arpino, and D. M. Roy. On the role of data in PAC-Bayes. *AISTATS*, 2021.

P. Germain, A. Lacasse, F. Laviolette, and M. Marchand. PAC-Bayesian learning of linear classifiers. *ICML*, 2009.

B. Guedj. A primer on PAC-Bayesian learning. *Proceedings of the Second Congress of the French Mathematical Society*, 2019.

Benjamin Guedj and Pierre Alquier. PAC-Bayesian estimation and prediction in sparse additive models. *Electronic Journal of Statistics*, 7, 2013.

W. Hu, L. Xiao, and J. Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks. *ICLR*, 2020.

K. Kawaguchi, L. P. Kaelbling, and Y. Bengio. Generalization in deep learning. *arXiv:1710.05468*, 2017.

A. Khaled and P. Richtárik. Better theory for SGD in the nonconvex world. *arXiv:2002.03329*, 2020.

A. Krizhevsky. Learning multiple layers of features from tiny images. *MSc Thesis University of Toronto*, 2009.

J. Langford and R. Caruana. (Not) bounding the true error. *NeurIPS*, 2002.

J. Langford and M. Seeger. Bounds for averaging classifiers. *CMU technical report*, 2001.

J. Langford and J. Shawe-Taylor. PAC-Bayes & margins. *NeurIPS*, 2003.

G. Letarte, P. Germain, B. Guedj, and F. Laviolette. Dichotomize and generalize: PAC-Bayesian binary activated deep neural networks. *NeurIPS*, 2019.

A. Maurer. A note on the PAC Bayesian theorem. *arXiv:0411099*, 2004.

D. A. McAllester. Some PAC-Bayesian theorems. *COLT*, 1998.

D. A. McAllester. PAC-Bayesian model averaging. *COLT*, 1999.

D. A. McAllester. PAC-Bayesian stochastic model selection. *Machine Learning*, 51, 2004.

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. *NeurIPS*, 2017.

E. Parrado-Hernández, A. Ambroladze, J. Shawe-Taylor, and S. Sun. PAC-Bayes bounds with data dependent priors. *Journal of Machine Learning Research*, 13, 2012.

T. Poggio, A. Banburski, and Q. Liao. Theoretical issues in deep networks. *PNAS*, 117(48), 2020.

R. Price. A useful theorem for nonlinear devices having gaussian inputs. *IRE Transactions on Information Theory*, 4(2), 1958.

M. Pérez-Ortiz, O. Risvaplata, J. Shawe-Taylor, and C. Szepesvári. Tighter risk certificates for neural networks. *Journal of Machine Learning Research*, 22, 2021a.

M. Pérez-Ortiz, O. Rivasplata, B. Guedj, M. Gleeson, J. Zhang, J. Shawe-Taylor, M. Bober, and J. Kittler. Learning PAC-Bayes priors for probabilistic neural networks. *arXiv:2109.10304*, 2021b.

C. M. Stein. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics*, 9(6), 1981.

V. B. Tadić and A. Doucet. Asymptotic bias of stochastic gradient search. *The Annals of Applied Probability*, 27(6), 2017.

N. Thiemann, C. Igel, O. Wintenberger, and Y. Seldin. A strongly quasiconvex PAC-Bayesian bound. *ALT*, 2017.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64 (3), 2021.

W. Zhou, V. Veitch, M. Austern, R. P. Adams, and P. Orbanz. Non-vacuous generalization bounds at the imagenet scale: a PAC-Bayesian compression approach. *ICLR*, 2019.

## Supplementary material

## SM1 PROOFS

**Proposition 2.** *Denote the cumulative distribution function (CDF) of a standard normal as $\psi : u \mapsto \frac{1}{2}(1 + \mathrm{erf}(u/\sqrt{2}))$. Fix a pair $x, y$ and let*

$$\begin{aligned} L_1 =& \psi\left(\max_{i\neq y} \frac{F_i(x) - M_y(x)}{\sqrt{V_y(x)}}\right) , \\ L_2 =& 1 - \prod_{i\neq y} \psi\left(\frac{F_y(x) - M_i(x)}{\sqrt{V_i(x)}}\right) , \end{aligned}$$

*where $F(x) \sim \mathcal{N}(M(x), \mathrm{diag}(V(x)))$. Then*

$$\mathbb{E}[L_1] = \mathbb{E}[L_2] = \mathbb{P}(\hat{y} \neq y) \, , \tag{SM1}$$
$$\mathbb{E}[\nabla L_1] = \mathbb{E}[\nabla L_2] = \nabla \mathbb{P}(\hat{y} \neq y) \, , \tag{SM2}$$

*where the gradient is with respect to all the components of $M(x)$ and $V(x)$.*

*Proof.* We start by showing that $\mathbb{E}[L_1] = \mathbb{P}(\hat{y} \neq y)$. We have

$$\mathbb{P}(\hat{y} \neq y) = \mathbb{P}\left(F_y(x) < \max_{i\neq y} F_i(x)\right) = \mathbb{E}\left[\mathbb{P}\left(F_y(x) < \max_{i\neq y} F_i(x) \middle| \{F_i(x)\}_{i\neq y}\right)\right] = \mathbb{E}[L_1] \, .$$

For $L_2$ again we first use conditioning w.r.t. $F_y(x)$

$$\mathbb{P}(\hat{y} \neq y) = \mathbb{E}\left[\mathbb{P}\left(F_y(x) < \max_{i\neq y} F_i(x) \middle| F_y(x)\right)\right] = 1 - \mathbb{E}\left[\mathbb{P}\left(F_y(x) \geq \max_{i\neq y} F_i(x) \middle| F_y(x)\right)\right] .$$

As the events $\{F_y(x) \geq F_i(x) | F_y(x)\}_{i\neq y}$ are independent, we can write

$$\mathbb{P}(\hat{y} \neq y) = 1 - \mathbb{E}\left[\prod_{i\neq y} \mathbb{P}\left(F_i(x) \leq F_y(x) \middle| F_y(x)\right)\right] = \mathbb{E}[L_2] \, ,$$

and so (SM1) is proved.

Now, to show (SM2), we need to prove that it is possible to swap expectation and differentiation for both $L_1$ and $L_2$. For $L_2$ everything is straightforward, as it is a smooth function of $M$ and $V$ (as all the components of $V$ are assumed to be strictly positive) and its gradient can be easily bounded (uniformly in some neighbourhood of $(M_i(x), V_i(x))_{i\neq y}$) by a function of $F_y(x)$ with finite expectation. Hence we can apply Leibniz integral rule. For $L_1$, this is the case only for $\partial_{M_y}$ and $\partial_{V_y}$, as $\max_{i\neq y} \frac{F_i(x) - M_y(x)}{\sqrt{V_y(x)}} = \frac{\max i\neq y\{F_i(x)\} - M_y(x)}{\sqrt{V_y(x)}}$ is smooth in $M_y$ and $V_y$, and its gradient can be easily bounded (uniformly in some neighbourhood of $(M_y(x), V_y(x))$) by a function of $(F_i(x))_{i\neq y}$ with finite expectation. However, for any $j \neq y$, the integrand is not everywhere differentiable wrt $M_j$ and $V_j$. Yet, we can still swap expectation and differentiation using Proposition SM1, detailed below. □

The two results that follow are well known in the literature, and restated here for convenience. For completeness we give a proof for both of them. Denote as $\rho_{m,s}$ the density of a normal random variable with mean $m$ and standard deviation $s$. For convenience we let $\rho = \rho_{0,1}$. All integrals $\int$ are over $\mathbb{R}$.

The next proposition is essentially a reformulation of Price's theorem (Price, 1958).

**Proposition SM1.** *Let $Z \sim \mathcal{N}(0, 1)$ and $X = sZ + m$. Let $g : \mathbb{R} \to \mathbb{R}$ be a locally Lipschitz function with a polynomially bounded derivative. Then*

$$\nabla_{m,s} \mathbb{E}_{X\sim\mathcal{N}(m,s^2)}[g(X)] = \mathbb{E}_{Z\sim\mathcal{N}(0,1)}[\nabla_{m,s} g(sZ + m)] \, .$$

*Proof.* Recall that $\partial_m \rho_{m,s}(x) = \frac{x-m}{s}\rho_{m,s}(x)$ and $\partial_s \rho_{m,s}(x) = \frac{(x-m)^2-s^2}{s^3}\rho_{m,s}(x)$. Let $z = sx + m$, then $\rho_{m,s}(x)\mathrm{d}x = \rho(z)\mathrm{d}z$. Note that by the local Lipschitzianity $g'$ is defined almost everywhere. Since it is polynomially bounded, the expectation $\mathbb{E}[\nabla_{m,s} g(sZ+m)]$ makes sense. Note moreover that $g$ is polynomially bounded as $g'$ is.

We start by proving the equality for the $m$-derivative. We have

$$\partial_m \mathbb{E}[g(X)] = \partial_m \int \rho_{m,s}(x)g(x)\mathrm{d}x = \int (\partial_m \rho_{m,s}(x))g(x)\mathrm{d}x\,,$$

by Leibniz integration rule, as $\rho_{m,s}$ is smooth in its arguments and the continuity and polynomial boundedness of $g$ ensure that $\int x \to \partial_m \rho_{m,s}(x))g(x)\,\mathrm{d}x$ is well defined and finite. Now, we have

$$\int (\partial_m \rho_{m,s}(x))g(x)\mathrm{d}x = \int \frac{x-m}{s^2}\rho_{m,s}(x)g(x)\mathrm{d}x = \int \frac{z}{s}\rho(z)g(sz+m)\mathrm{d}z\,.$$

From Lemma SM1 below, we get

$$\int \frac{z}{s}\rho(z)g(sz+m)\mathrm{d}z = \int \frac{1}{s}\rho(z)sg'(sz+m)\mathrm{d}z = \int \rho(z)g'(sz+m)\mathrm{d}z\,.$$

Now, as $g'(sz+m) = \partial_m g(sz+m)$ we conclude that

$$\partial_m \mathbb{E}[g(X)] = \mathbb{E}[\partial_m g(sZ+m)]\,.$$

For the $s$-derivative, the proof is essentially analogous. Proceeding as above, we have

$$\partial_s \mathbb{E}[g(X)] = \int (\partial_s \rho_{m,s}(x))g(x)\mathrm{d}x = \int \frac{(x-m)^2-s^2}{s^3}\rho_{m,s}(x)g(x)\mathrm{d}x = \int \frac{z^2-1}{s}\rho(z)g(sz+m)\mathrm{d}z\,.$$

Again from Lemma SM1 we find that

$$\int \frac{z^2-1}{s}\rho(z)g(sz+m)\mathrm{d}z = \int \rho(z)zg'(sz+m)\mathrm{d}z\,.$$

We conclude that

$$\partial_s \mathbb{E}[g(x)] = \mathbb{E}[\partial_s g(sz+m)]\,,$$

since $\partial_s g(sz+m) = zg'(sz+m)$. ◻

The next lemma states Stein's identity (Stein, 1981) and a straightforward corollary.

**Lemma SM1.** *Let $Z \sim \mathcal{N}(0,1)$, and $g : \mathbb{R} \to \mathbb{R}$ a locally Lipschitz function with a polynomially bounded derivative. Then*

$$\mathbb{E}[Zg(Z)] = \mathbb{E}[g'(Z)]\,,$$
$$\mathbb{E}[(Z^2-1)g(Z)] = \mathbb{E}[Zg'(Z)]\,.$$

*Proof.* The first equality, known as Stein's identity, is established using integration by parts:

$$0 = \int (\rho(z)g(z))'\mathrm{d}z = \int \rho'(z)g(z)\mathrm{d}z + \int \rho(z)g'(z)\mathrm{d}z = -\int z\rho(z)g(z) + \int \rho(z)g'(z)\mathrm{d}z\,,$$

where we used that $g'$ exists almost everywhere as $g$ is locally Lipschitz, and that both $g$ and $g'$ are polynomially bounded, so all integral are finite and well defined. Now take $h(z) = zg(z)$. Then we have $h'(z) = zg'(z) + g(z)$ and so

$$\mathbb{E}[Z^2 g(Z)] = \mathbb{E}[Zh(Z)] = \mathbb{E}[h'(Z)] = \mathbb{E}[Zg'(Z)] + \mathbb{E}[g(Z)]\,,$$

which is the second equality. ◻

**Proposition 3.** *Assume that $\mathcal{B}$ is locally Lipschitz in the hidden stochastic parameters $\theta^{\mathcal{H}}$, and that $\nabla_{\theta^{\mathcal{H}}}\mathcal{B}$ is polynomially bounded. If $\mathcal{B}(\mathcal{E}_S(\mathcal{Q}), \mathtt{Pen})$ is an affine function of $\mathcal{E}_S(\mathcal{Q})$, then we have $\mathbb{E}[\hat{\mathcal{B}}] = \mathcal{B}$ and $\mathbb{E}[\nabla\hat{\mathcal{B}}] = \nabla\mathcal{B}$, the gradient being with respect to the trainable hyper-parameters $\mathtt{p}$.*

*Proof.* By linearity it is sufficient to show that $\mathbb{E}[\hat{\mathcal{E}}_S(\mathcal{Q})] = \mathcal{E}_S(\mathcal{Q})$ and $\mathbb{E}[\nabla\hat{\mathcal{E}}_S(\mathcal{Q})] = \nabla\mathcal{E}_S(\mathcal{Q})$. Note that, following the discussion of Section 3.1, we can write $\hat{\mathcal{E}}_S(\mathcal{Q}) = \sum_{x\in S}\hat{\mathcal{E}}_x$ where

$$\hat{\mathcal{E}}_x = E(M(x,\theta^{\mathcal{H}},\mathfrak{p}^{\mathcal{L}}), V(x,\theta^{\mathcal{H}},\mathfrak{p}^{\mathcal{L}}),\xi)\,,$$

for some suitable function $E$. If we are dealing with binary classification the variable $\xi$ can be omitted, otherwise it represents the random draws needed to obtain the estimate $L_1$ or $L_2$ (defined in Proposition 2).

Define $\mathcal{E}_x = \mathbb{E}[\hat{\mathcal{E}}_x]$, the expectation being over $\xi$ and $\theta^{\mathcal{H}}$. By Proposition 2 (if we are dealing with multiclass classification, otherwise by definition) we get that $\mathcal{E}_S(\mathcal{Q}) = \sum_{x\in S}\mathcal{E}_x$. Consequently we have

$$\mathcal{E}_S(\mathcal{Q}) = \sum_{x\in S}\mathbb{E}[\hat{\mathcal{E}}_x] = \mathbb{E}[\hat{\mathcal{E}}_S(\mathcal{Q})]\,.$$

Now, to show the unbiasedness of the gradient, it is enough to show that for all $x \in S$

$$\nabla_{\mathfrak{p}}\mathcal{E}_x = \mathbb{E}[\nabla_{\mathfrak{p}}\hat{\mathcal{E}}_x]\,.$$

First, again by Proposition 2 we can write

$$\mathbb{E}[\nabla_{\mathfrak{p}}\hat{\mathcal{E}}_x] = \mathbb{E}[\nabla_{\mathfrak{p}}\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]] = \mathbb{E}[\tfrac{\partial(M,V)}{\partial\mathfrak{p}}\mathbb{E}[\nabla_{M,V}\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]] = \mathbb{E}[\tfrac{\partial(M,V)}{\partial\mathfrak{p}}\nabla_{M,V}\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]] = \mathbb{E}[\nabla_{\mathfrak{p}}\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]]\,.$$

Now, $\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]$ is the probability that a component of a Gaussian vector with mean $M$ and covariance $\mathrm{diag}(V)$ is smaller than the maximum of the other components (cf. Section 3.1). This is a smooth function of $M$ and $V$, which in turn are smooth functions of the last layer's hyper-parameters $\mathfrak{p}^{\mathcal{L}}$. As a consequence we can write

$$\nabla_{\mathfrak{p}^{\mathcal{L}}}\mathcal{E}_x = \mathbb{E}[\nabla_{\mathfrak{p}^{\mathcal{L}}}\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]] = \mathbb{E}[\nabla_{\mathfrak{p}^{\mathcal{L}}}\hat{\mathcal{E}}_x]\,.$$

As for the hidden hyper-parameters, since we are assuming that all the hidden stochastic parameters are independent Gaussian random variables, we can apply Proposition SM1, which brings

$$\nabla_{\mathfrak{p}^{\mathcal{H}}}\mathcal{E}_x = \mathbb{E}[\nabla_{\mathfrak{p}^{\mathcal{H}}}\mathbb{E}[\hat{\mathcal{E}}_x|\mathcal{F}^{\mathcal{H}}]] = \mathbb{E}[\nabla_{\mathfrak{p}^{\mathcal{H}}}\hat{\mathcal{E}}_x]\,,$$

thus concluding our proof. □

## SM2 A NOTE ON UNBIASEDNESS

The previous results state that the gradient estimates used in the Cond-Gauss algorithm are unbiased, as long as the bound is affine in the empirical error. Under suitable regularity conditions, this ensures that stochastic gradient descent algorithms converge to a stationary point of the objective (Khaled and Richtárik, 2020). However, among the four bounds (4) that we used in our experiments, only (4b) and (4d) are actually affine. We argue here that in most cases of interest $\hat{\mathcal{E}}_S(\mathcal{Q})$ is concentrated enough that the bounds (4a) and (4c) are approximately affine in the empirical error. In the following, we detail this heuristic idea and then give some empirical evidence on MNIST in the case of (4a). This almost affine behaviour ensures that the gradient used by our stochastic optimisation procedure is almost unbiased, and hence we can expect the algorithm to converge to a point close to a stationary point of the objective (Tadić and Doucet, 2017).

Consider a generic bound $\mathcal{B} = B(\mathcal{E}_S(\mathcal{Q}))$, where $B$ might be a non-affine function. Our estimate is of the form $\hat{\mathcal{B}} = B(\hat{\mathcal{E}}_S(\mathcal{Q}))$. We can now consider a linearised version $\bar{B}$ of $B$, defined as

$$\bar{B}(\mathcal{E}) = B(\mathcal{E}_S(\mathcal{Q})) + (\mathcal{E} - \mathcal{E}_S(\mathcal{Q}))B'(\mathcal{E}_S(\mathcal{Q}))\,.$$

Clearly, in a sufficiently small neighborhood of $\mathcal{E}_S(\mathcal{Q})$, we can expect $B$ and $\bar{B}$ to almost coincide. In particular, if the law of $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates around $\mathcal{E}_S(\mathcal{Q})$, we can expect that with high probability

$$B(\hat{\mathcal{E}}_S(\mathcal{Q})) \simeq \bar{B}(\mathcal{E}_S(\mathcal{Q}))\,.$$

As $\bar{B}$ is affine, we can apply Proposition 3 and get

$$\mathbb{E}[\hat{\mathcal{B}}] = \mathbb{E}[B(\hat{\mathcal{E}}_S(\mathcal{Q}))] \simeq \mathbb{E}[\bar{B}(\hat{\mathcal{E}}_S(\mathcal{Q}))] = \bar{B}(\mathcal{E}_S(\mathcal{Q})) = B(\mathcal{E}_S(\mathcal{Q})) = \mathcal{B}\,,$$
$$\mathbb{E}[\nabla_{\mathfrak{p}}\hat{\mathcal{B}}] = \mathbb{E}[\nabla_{\mathfrak{p}}B(\hat{\mathcal{E}}_S(\mathcal{Q}))] \simeq \mathbb{E}[\nabla_{\mathfrak{p}}\bar{B}(\hat{\mathcal{E}}_S(\mathcal{Q}))] = \nabla_{\mathfrak{p}}\bar{B}(\mathcal{E}_S(\mathcal{Q})) = \nabla_{\mathfrak{p}}B(\mathcal{E}_S(\mathcal{Q})) = \nabla_{\mathfrak{p}}\mathcal{B}\,.$$

To empirically justify the above, we consider the bound (4a), which was used for most of our experiments. Figure SM1 and Figure SM2 show that indeed $\hat{\mathcal{E}}_S(\mathcal{Q})$ is sufficiently concentrated around its mean to see the bound as an affine function of the empirical error. Figure SM1 reports the data from the network achieving the best bound in our experiments with *data-dependent* priors on MNIST. On the other hand, among the networks trained with the `invKL` objectives on MNIST with *data-free* priors, the one achieving the tightest bound was used for Figure SM2. In both figures, the histogram represents the distribution of 10000 realisations of $\hat{\mathcal{E}}_S(\mathcal{Q})$. It is clear that in both cases the bound is essentially affine in the empirical loss, in the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates (zoomed portion of the plot).

Similar observations hold when the objective is derived from (4c).

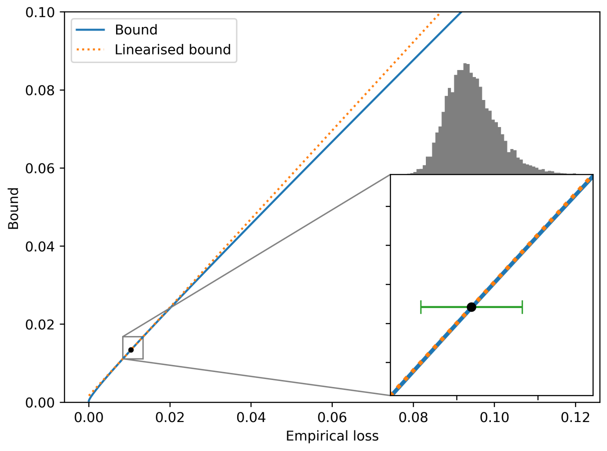

**Figure SM1:** (Same as Figure 1 from the main text.) Experimental evidence, from a network trained with a *data-dependent* prior on MNIST, that the bound (4a) is almost affine in the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates. The network used was the one achieving the best generalisation bound in our experiment on MNIST with *data-dependent* priors. 10000 realisations of $\hat{\mathcal{E}}_S(\mathcal{Q})$ were sampled. Their distribution is summarised by the histogram above the zoomed portion of the plot. The black dot is the bound for the average value found for $\hat{\mathcal{E}}_S(\mathcal{Q})$, while the green error bar has a total width of 4 empirical standard deviations. In the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates, the bound and its linearised version almost coincide. Along the green error bar, the bound's slope has a relative variation of $\pm 0.8\%$.

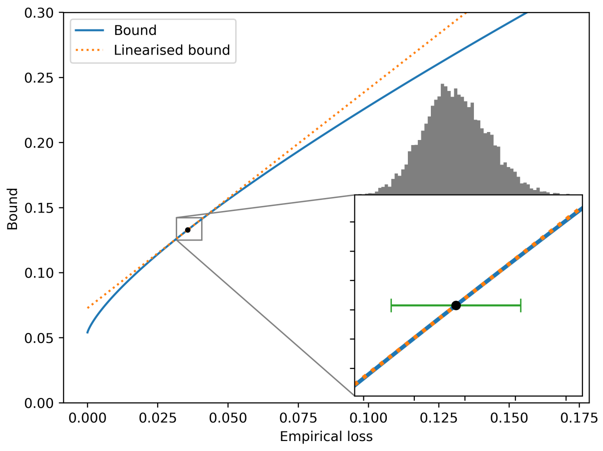

**Figure SM2:** Experimental evidence, from a network trained with a *data-free* prior on MNIST, that the bound (4a) is almost affine in the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates. Among the networks trained with the `invKL` objectives on MNIST with *data-free* priors, the one achieving the tightest bound was used in this experiment. 10000 realisations of $\hat{\mathcal{E}}_S(\mathcal{Q})$ were sampled. Their distribution is summarised by the histogram above the zoomed portion of the plot. The black dot is the bound for the average value found for $\hat{\mathcal{E}}_S(\mathcal{Q})$, while the green error bar has a total width of 4 empirical standard deviations. In the region where $\hat{\mathcal{E}}_S(\mathcal{Q})$ concentrates, the bound and its linearised version almost coincide. Along the green error bar, the bound's slope has a relative variation of $\pm 2\%$.

## SM3 PAC-BAYESIAN TRAINING FOR GENERAL ARCHITECTURES

In the main text we focused on the case of a network whose stochastic parameters are all Gaussian. This is not a necessary condition for the Cond-Gauss algorithm. What we need is actually to be able to express the KL between prior and posterior as a differentiable expression of the hyper-parameters, and to evaluate the gradient (wrt the hyper-parameters) of a single empirical loss's realisation. We can satisfy this last requirement if we are able to rewrite the stochastic parameters as a (differentiable) function $\Theta$ of the hyper-parameters $\mathfrak{p}$ and of some random variable $\tau$ (independent of $\mathfrak{p}$) such that $\Theta(\mathfrak{p},\tau)$ has the same law of $\theta$, namely $\mathcal{Q}_\mathfrak{p}$. In short, for any measurable function $\varphi$,

$$\mathbb{E}_{\theta\sim\mathcal{Q}_\mathfrak{p}}[\varphi(\theta)] = \mathbb{E}_\tau[\varphi(\Theta(\mathfrak{p},\tau))]\,.$$

In particular, to sample a realisation $\hat{\varphi}$ of $\varphi(\theta)$ we can sample a realisation $\hat{\tau}$ of $\tau$ and then define

$$\hat{\varphi} = \varphi(\Theta(\mathfrak{p},\hat{\tau}))\,.$$

As long as $\varphi\circ\Theta$ is differentiable in $\mathfrak{p}$, we can evaluate the gradient of $\hat{\varphi}$ wrt $\mathfrak{p}$.

For the Cond-Gauss algorithm to be implementable, we require that there exists a $\mathfrak{p}$-differentiable reparametrisation $\Theta$ for the hidden parameters $\theta^{\mathcal{H}}$. Clearly, this is the case if $\theta^{\mathcal{H}}$ is a Gaussian vector with independent components. Indeed, if we denote by $\mathfrak{m}^{\mathcal{H}}$ and $\mathfrak{s}^{\mathcal{H}}$ the vectors of means and standard deviations, we have

$$\theta^{\mathcal{H}} = \mathfrak{m}^{\mathcal{H}} + \mathfrak{s}^{\mathcal{H}} \odot \tau \,,$$

where $\tau$ is a vector with independent standard normal components and $\odot$ denotes the component-wise product. This is what was used for the networks in our experiments.

## SM4 NUMERICAL EVALUATION OF $\mathrm{kl}^{-1}$ AND ITS GRADIENT

When the training objective is `invKL`, it is necessary to evaluate $\mathrm{kl}^{-1}$ and its gradient, in order to implement the Cond-Gauss algorithm. Many of the most popular deep learning libraries, such as PyTorch and TensorFlow, do not provide an implementation for $\mathrm{kl}^{-1}$. However, as pointed out by Dziugaite and Roy (2017), a fast numerical evaluation can be done via a few iterations of Newton's method. This is what we used in our code.

We show here that the gradient of $\mathrm{kl}^{-1}$ can be expressed as a function of $\mathrm{kl}^{-1}$, so that the implementation of the latter allows the evaluation of the former. Recall that

$$\mathrm{kl}(u\|v) = u \log \frac{u}{v} + (1-u)\log\frac{1-u}{1-v}\,.$$

For $u > 0$, the mapping $v \mapsto \mathrm{kl}(u\|v)$ is not injective. However if we restrict its domain to $\{(u,v) \in [0,1]^2 : v \geq u\}$, then we find a bijective map, whose inverse coincides with $c \mapsto \mathrm{kl}^{-1}(u|c)$ (with the definition (3) for $\mathrm{kl}^{-1}$). It follows immediately that

$$\partial_c \mathrm{kl}^{-1}(u|c) = \frac{1}{\partial_v \mathrm{kl}(u\|v)}\bigg|_{v=\mathrm{kl}^{-1}(u|c)} = \left(\frac{1-u}{1-v} - \frac{u}{v}\right)^{-1}\bigg|_{v=\mathrm{kl}^{-1}(u|c)}\,.$$

To find an expression for $\partial_u \mathrm{kl}^{-1}(u|c)$ we can proceed as follow. Let $\mathrm{kl}^{-1}(u|c) = v$ and $\mathrm{kl}^{-1}(u+\varepsilon|c) = v+\varepsilon'$, with $\varepsilon' = \varepsilon\partial_u \mathrm{kl}^{-1}(u|c) + o(\varepsilon)$. This means that $\mathrm{kl}(u+\varepsilon\|v+\varepsilon') = \mathrm{kl}(u\|v)$, so that $\varepsilon\partial_u \mathrm{kl}(u\|v) + \varepsilon'\partial_v \mathrm{kl}(u\|v) = o(\varepsilon)$. Taking $\varepsilon \to 0$ we find

$$\partial_u \mathrm{kl}^{-1}(u|c) = -\frac{\partial_u \mathrm{kl}(u\|v)}{\partial_v \mathrm{kl}(u\|v)}\bigg|_{v=\mathrm{kl}^{-1}(u|c)} = \left(\log\frac{1-u}{1-v} - \log\frac{u}{v}\right)\bigg/\left(\frac{1-u}{1-v} - \frac{u}{v}\right)\bigg|_{v=\mathrm{kl}^{-1}(u|c)}\,.$$

## SM5 ADDITIONAL EXPERIMENTAL DETAILS AND RESULTS

In this section we give additional details about our experiments. The PyTorch code written for this paper is available at https://github.com/eclerico/CondGauss. In all our experiments we used the average of 100 independent estimates of $L^1$ (defined in Proposition 2) to evaluate the empirical error. To keep the standard deviations $\sigma$ positive during the training, we trained the parameters$\rho$ defined by $\sigma = |\rho|^{3/2}$. We found empirically that this transformation allowed for a much faster training compared to the usual exponential choices (Dziugaite and Roy, 2017; Pérez-Ortiz et al., 2021a).

### SM5.1 MNIST

For our experiments on MNIST, we only used the standard training dataset, which consists of 60000 labelled examples. We ran our experiments on a 4-layer ReLU stochastic network, whose parameters were independent Gaussians with trainable means and variances. The architecture used was the following:

$$x \mapsto y = L_2 \circ \phi \circ L_1 \circ \phi \circ f \circ C_2 \circ \phi \circ C_1(x)\,,$$

with

- $C_1$: convolutional layer; channels: IN 1, OUT 32; kernel: (3, 3); stride: (1, 1);
- $C_2$: convolutional layer; channels: IN 32, OUT 64; kernel: (3, 3); stride: (1, 1);
- $L_1$: linear layer; dimensions: IN 9216, OUT 128;

- $L_2$: linear layer; dimensions: IN 128, OUT 10;
- $f$: max pool (kernel size = 2) & flatten;
- $\phi$: ReLU activation component-wise.

All convolutional and linear layers were with bias.

### SM5.1.1 Data-free priors

We first experimented on data-free priors, whose means were initialised via the Pytorch default initialisation. We tried different values for the initial prior's variances: .01, .001, and .0001. We compared the performances of the standard PAC-Bayesian training algorithm (S), where the misclassification loss is replaced by a bounded version of the cross-entropy loss as in Pérez-Ortiz et al. (2021a), and the Cond-Gauss algorithm (G). We used the following four training objectives from (4):

$$\begin{aligned}
&\texttt{invKL}: && \mathrm{kl}^{-1}(\mathcal{E}_S(\mathcal{Q})|\texttt{Pen}_\kappa)\,;\\
&\texttt{McAll}: && \mathcal{E}_S(\mathcal{Q}) + \sqrt{\texttt{Pen}_\kappa/2}\,;\\
&\texttt{quad}: && \left(\sqrt{\mathcal{E}_S(\mathcal{Q}) + \texttt{Pen}_\kappa/2} + \sqrt{\texttt{Pen}_\kappa/2}\right)^2;\\
&\texttt{lbd}: && \tfrac{1}{1-\lambda/2}\left(\mathcal{E}_S(\mathcal{Q}) + \texttt{Pen}_\kappa/\lambda\right),
\end{aligned}$$

where the KL penalty is defined as

$$\texttt{Pen}_\kappa = \frac{\kappa}{m}\left(\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \log\frac{2\sqrt{m}}{\delta}\right). \tag{9}$$

The factor $\kappa$ in (9) can increase or reduce the weight of the KL term during the training. We experimented three different values for this parameter: 0.5, 1, and 2. For the last objective, `lbd`, the parameter $\lambda$ takes values in $(0,1)$ and is optimised during training[6].

For all the different training settings, the network was trained via SGD with momentum for 250 epochs with a learning rate $\eta = .005$ followed by 50 epochs with $\eta = .0001$. We tried using different values for the momentum: 0.5, 0.7, and 0.9. During the training, at the end of each epoch, we kept track of the bound (4a)'s empirical value in order to pick the best epoch at the end of the training.

Figure 2 and Table 1 in the main text report our results.

### SM5.1.2 Data-dependent priors

For the data-dependent priors, we used 50% of the dataset to train $\mathcal{P}$ and the remaining 50% to train $\mathcal{Q}$. We always used the Cond-Gaussian algorithm for both prior and posterior. All the posteriors were trained with the `invKL` objective and $\kappa = 1$, whilst for the prior, we experimented with both `invKL` (with $\kappa = 0.1$) and with direct empirical risk minimisation (`ERM`), meaning that the objective was simply $\mathcal{E}_S(\mathcal{Q})$. The initial prior's variances were set at 0.01, while the means were randomly initialised (via the default PyTorch initialisation for each layer). We used different dropout values, as shown in Table SM1. The prior's training consisted of 750 epochs with $\eta = .005$, followed by 250 epochs with $\eta = .0001$, the posterior's training of 750 epochs with $\eta = 10^{-5}$, followed by 250 epochs with $\eta = 10^{-6}$. We used SGD with a momentum of 0.9 for both priors and posteriors. The results of the experiment can be found in Table SM1.

## SM5.2 CIFAR10

As we had done for the MNIST dataset, for CIFAR10 we used only the standard training dataset (50000 labelled images). We trained a 9-layer architecture (6 convolutional + 3 linear layers) and a 15-layer architecture (12 convolutional + 3 linear layers). We experimented with data-dependent priors only, training $\mathcal{P}$ with 50% of the data for the 9-layer classifier and both with 50% and 70% for the 15-layer one.

[6] In our experiments, we initialised $\lambda$ at 0.5 and then doubled the number of epochs, alternating one epoch of $\lambda$'s optimisation with one of optimisation for $\mathfrak{m}$ and $\mathfrak{s}$.

**Table SM1:** MNIST - Prior learnt on 50% of the dataset

| Prior | | | | | | | Posterior | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tm[a] | do[b] | pf[c] | iv[d] | l1[e] | l2[f] | p[g] | tm[a] | l1[e] | l2[f] | t[h] | p[g] | b[i] |
| G `ERM` | 0 | - | .001 | .0010 | .0126 | 3.179 | G `invKL` | .0010 | .0122 | $.0122_{\pm .0006}$ | 3.671 | .0164 |
| G `invKL` | 0 | .01 | .001 | .0008 | .0125 | 3.179 | G `invKL` | .0008 | .0119 | $.0115_{\pm .0007}$ | 3.882 | .0162 |
| G `ERM` | .1 | - | .001 | .0010 | .0111 | 3.179 | G `invKL` | .0010 | .0107 | $.0110_{\pm .0006}$ | 3.688 | .0148 |
| G `invKL` | .1 | .01 | .001 | .0006 | .0113 | 3.179 | G `invKL` | .0006 | .0107 | $.0109_{\pm .0006}$ | 3.944 | .0149 |
| G `ERM` | .2 | - | .001 | .0011 | .0111 | 3.179 | G `invKL` | .0011 | .0107 | $.0101_{\pm .0005}$ | 3.742 | .0148 |
| G `invKL` | .2 | .01 | .001 | .0010 | .0108 | 3.179 | G `invKL` | .0010 | .0104 | $.0101_{\pm .0006}$ | 3.801 | **.0144** |

[a] tm: Training method.
[b] do: Dropout probability for the prior's training.
[c] pf: Penalty factor $\kappa$ for the prior's training objective.
[d] iv: Initial value of the prior's variances.
[e] l1: Empirical error estimate on the prior dataset.
[f] l2: Empirical error estimate on the posterior dataset.
[g] p: KL penalty `Pen` (2) in $10^{-4}$ units.
[h] t: Test error$_{\pm\text{standard deviation}}$ (from 1000 realisations).
[i] b: Final PAC-Bayesian bound.

### SM5.2.1 9-layer architecture

The 9-layer architecture had the following structure:

$$x \mapsto L_3 \circ \phi \circ L_2 \circ \phi \circ L_1 \circ \phi \circ f_2 \circ C_6 \circ \phi \circ C_5 \circ \phi \circ f_1 \circ C_4 \circ \phi \circ C_3 \circ \phi \circ f_1 \circ C_2 \circ \phi \circ C_1(x)\,.$$

Here are detailed the different layers:

- $C_1$: convolutional layer; channels: IN 3, OUT 32; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_2$: convolutional layer; channels: IN 32, OUT 64; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_3$: convolutional layer; channels: IN 64, OUT 128; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_4$: convolutional layer; channels: IN 128, OUT 128; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_5$: convolutional layer; channels: IN 128, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_6$: convolutional layer; channels: IN 256, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $L_1$: linear layer; dimensions: IN 4096, OUT 1024;
- $L_2$: linear layer; dimensions: IN 1024, OUT 512;
- $L_3$: linear layer; dimensions: IN 512, OUT 10;
- $f_1$: max pool (kernel size = 2, stride = 2);
- $f_2$: max pool (kernel size = 2, stride = 2) & flatten;
- $\phi$: ReLU activation component-wise.

All convolutional and linear layers are with bias.

The results for the 9-layer architecture are reported in Table 2 in the main text. After some preliminary experiments, we chose to train both priors and posteriors via the Cond-Gauss algorithm with the `invKL` objective. We used a small factor $\kappa$ for the prior, to avoid regularising too much, whilst $\kappa$ was 1 for the posterior. We tried different values for the dropout and $\kappa$ in the prior's training (see Table 2). We used SGD with momentum 0.9 for both prior and posterior. For $\mathcal{P}$ the training consisted of 1500 epochs with $\eta = .005$ followed by 500 epochs with $\eta = .0001$, whilst $\mathcal{Q}$ was trained for 1500 epochs with $\eta = 10^{-5}$, plus 500 epochs with $\eta = 10^{-6}$.

### SM5.2.2 15-layer architecture

The 15-layer architecture had the following structure:

$$\begin{aligned} x \mapsto & L_3 \circ \phi \circ L_2 \circ \phi \circ L_1 \circ \phi \circ f_2 \circ C_{12} \circ \phi \circ C_{11} \circ \phi \circ C_{10} \circ \phi \circ C_9 \circ \phi \circ f_1 \circ C_8 \circ \phi \\ & \circ C_7 \circ \phi \circ f_2 \circ C_6 \circ \phi \circ C_5 \circ \phi \circ f_1 \circ C_4 \circ \phi \circ C_3 \circ \phi \circ f_1 \circ C_2 \circ \phi \circ C_1(x)\,. \end{aligned}$$

Here are detailed the different layers:

- $C_1$: convolutional layer; channels: IN 3, OUT 32; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_2$: convolutional layer; channels: IN 32, OUT 64; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_3$: convolutional layer; channels: IN 64, OUT 128; kernel: (3, 3); stride: (1, 1); padding(1, 1);

**Table SM2:** CIFAR10 - 15 layers - Prior learnt on 50% and 70% of the dataset

| | Prior | | | | | | | Posterior | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tm[a] | do[b] | pf[c] | iv[d] | l1[e] | l2[f] | p[g] | tm[a] | l1[e] | l2[f] | t[h] | p[g] | b[i] |
| | **Prior trained on 50% of the dataset** | | | | | | | | | | | | |
| Pre-Train do=.1 | G `ERM` | 0 | - | .001 | .0090 | .1946 | 3.778 | G `invKL` | .0090 | .1924 | $.1933_{\pm.0020}$ | 4.775 | .2082 |
| Pre-Train do=.1 | G `invKL` | 0 | .01 | .001 | .0085 | .1937 | 3.778 | G `invKL` | .0084 | .1909 | $.1922_{\pm.0022}$ | 4.913 | .2068 |
| Pre-Train do=.1 | G `ERM` | .1 | - | .001 | .0139 | .1722 | 3.778 | G `invKL` | .0139 | .1709 | $.1736_{\pm.0018}$ | 4.386 | **.1855** |
| Pre-Train do=.1 | G `invKL` | .1 | .01 | .001 | .0222 | .1746 | 3.778 | G `invKL` | .0222 | .1725 | $.1760_{\pm.0020}$ | 4.703 | .1875 |
| Pre-Train do=.2 | G `ERM` | 0 | - | .001 | .0214 | .1996 | 3.778 | G `invKL` | .0214 | .1974 | $.1939_{\pm.0020}$ | 4.734 | .2133 |
| Pre-Train do=.2 | G `invKL` | 0 | .01 | .001 | .0169 | .1963 | 3.778 | G `invKL` | .0169 | .1941 | $.1930_{\pm.0022}$ | 4.859 | .2100 |
| Pre-Train do=.2 | G `ERM` | .1 | - | .001 | .0240 | .1772 | 3.778 | G `invKL` | .0240 | .1758 | $.1791_{\pm.0017}$ | 4.474 | .1907 |
| Pre-Train do=.2 | G `invKL` | .1 | .01 | .001 | .0394 | .1764 | 3.778 | G `invKL` | .0393 | .1747 | $.1734_{\pm.0019}$ | 4.606 | .1897 |
| | **Prior trained on 70% of the dataset** | | | | | | | | | | | | |
| Pre-Train do=.1 | G `ERM` | 0 | - | .001 | .0057 | .1616 | 6.127 | G `invKL` | .0057 | .1602 | $.1643_{\pm.0020}$ | 6.882 | .1774 |
| Pre-Train do=.1 | G `invKL` | 0 | .01 | .001 | .0062 | .1634 | 6.127 | G `invKL` | .0062 | .1617 | $.1648_{\pm.0021}$ | 7.203 | .1793 |
| Pre-Train do=.1 | G `ERM` | .1 | - | .001 | .0098 | .1443 | 6.127 | G `invKL` | .0098 | .1430 | $.1470_{\pm.0017}$ | 7.006 | **.1595** |
| Pre-Train do=.1 | G `invKL` | .1 | .01 | .001 | .0180 | .1467 | 6.127 | G `invKL` | .0178 | .1446 | $.1506_{\pm.0019}$ | 7.374 | .1616 |
| Pre-Train do=.2 | G `ERM` | 0 | - | .001 | .0151 | .1639 | 6.127 | G `invKL` | .0151 | .1622 | $.1696_{\pm.0018}$ | 7.161 | .1797 |
| Pre-Train do=.2 | G `invKL` | 0 | .01 | .001 | .0127 | .1629 | 6.127 | G `invKL` | .0127 | .1611 | $.1656_{\pm.0020}$ | 7.293 | .1787 |
| Pre-Train do=.2 | G `ERM` | .1 | - | .001 | .0175 | .1484 | 6.127 | G `invKL` | .0175 | .1471 | $.1506_{\pm.0016}$ | 7.043 | .1638 |
| Pre-Train do=.2 | G `invKL` | .1 | .01 | .001 | .0306 | .1500 | 6.127 | G `invKL` | .0305 | .1484 | $.1498_{\pm.0018}$ | 7.090 | .1652 |

[a] tm: Training method.
[b] do: Dropout probability for the prior's training.
[c] pf: Penalty factor $\kappa$ for the prior's training objective.
[d] iv: Initial value of the prior's variances.
[e] l1: Empirical error estimate on the prior dataset.
[f] l2: Empirical error estimate on the posterior dataset.
[g] p: KL penalty `Pen` (2) in $10^{-4}$ units.
[h] t: Test error$_{\pm\text{standard deviation}}$ (from 1000 realisations).
[i] b: Final PAC-Bayesian bound.

- $C_4$: convolutional layer; channels: IN 128, OUT 128; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_5$: convolutional layer; channels: IN 128, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_6$: convolutional layer; channels: IN 256, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_7$: convolutional layer; channels: IN 256, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_8$: convolutional layer; channels: IN 256, OUT 256; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_9$: convolutional layer; channels: IN 256, OUT 512; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_{10}$: convolutional layer; channels: IN 512, OUT 512; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_{11}$: convolutional layer; channels: IN 512, OUT 512; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $C_{12}$: convolutional layer; channels: IN 512, OUT 512; kernel: (3, 3); stride: (1, 1); padding(1, 1);
- $L_1$: linear layer; dimensions: IN 2048, OUT 1024;
- $L_2$: linear layer; dimensions: IN 1024, OUT 512;
- $L_3$: linear layer; dimensions: IN 512, OUT 10;
- $f_1$: max pool (kernel size = 2, stride = 2);
- $f_2$: max pool (kernel size = 2, stride = 2) & flatten;
- $\phi$: ReLU activation component-wise.

All convolutional and linear layers are with bias.

For the 15-layer architecture, we experimented different prior trainings, with 50% and 70% of the training dataset. In both cases, it was necessary to introduce an initial pre-training for the prior's means, as otherwise the Cond-Gauss algorithm alone could not significantly decrease the training objective. First, we initialised the means with an orthogonal initialisation, as suggested in Hu et al. (2020). Then we optimised them by training a deterministic network (with the same architecture) using the cross-entropy loss on the prior's dataset, for 50 epochs with $\eta = .005$. Finally, via the Cond-Gauss algorithm, we completed the prior's training and proceeded with the posterior's tuning following the same learning rate schedule as for the 9-layer case. We always used SGD with momentum 0.9. Different objectives and dropout factors were used for training the prior, as detailed in Table SM2, which also reports the results of our experiment.