# On Functional Priors and Cold Posteriors in Bayesian Neural Networks

Martin Marek[1]

MSc Machine Learning

Internal Supervisor: Brooks Paige
Adjunct Supervisor: Alex Hawkins-Hooker

Submission date: 18 September 2023

**Abstract**

Bayesian neural networks tend to perform poorly when trained on augmented data. A common fix is to temper the posterior distribution by "cooling" (sharpening) it, which dramatically improves performance. Several papers have considered this "cold posterior effect" problematic from a Bayesian perspective and suggested it results from prior misspecification. One proposed solution is to use a Dirichlet prior defined over model outputs, forcing the model to be more confident on the training data. We demonstrate that this proposed Dirichlet prior only works due to its inaccurate implementation, secretly approximating a cold posterior. We show that correctly sampling from a prior distribution over model outputs requires a "change of variables" term that has not been previously discussed. Using a novel approximation for this term, we are able to control the confidence of a BNN, closely matching the expected confidence of a Dirichlet prior. We argue that a cold likelihood is theoretically sound but we should be explicit about its use, rather than using sophisticated approximations that inadvertently achieve the same result. The source code to reproduce all experiments is available at `https://github.com/martin-marek/msc-thesis`.

# Contents

# Chapter 1

# Introduction

## 1.1 Bayesian neural networks

Compared to standard neural networks, Bayesian neural networks (BNNs) offer a more faithful representation of our beliefs over model parameters. A priori, we know very little about the parameters, so we start with a vague prior. As we observe data, we apply Bayes' rule to systematically update our beliefs, and our posterior distribution over parameters becomes increasingly concentrated. In a BNN, we use the full posterior distribution to generate predictions; in contrast a standard neural network is only parametrized by a single set of parameters, corresponding to a single mode of the posterior.

Modern neural networks are typically overparametrized – there are many different sets of model parameters that fit the training data perfectly but disagree on unseen data [1–3]. As a result, using any single set of parameters to generate predictions is problematic – it ignores our uncertainty over models, leading to over-confident predictions. In a BNN, we consider the full distribution over possible models, leading to both improved accuracy and uncertainty estimation [4]. Even if all posterior samples of a BNN were overconfident, if they disagreed on their predictions, their joint prediction would have low confidence, consistent with our uncertainty in that datum.

BNNs are also easier to reason about theoretically compared to standard neural networks. For example, in order for standard neural networks to achieve good generalization, they must be initialized correctly [1, 5], trained with a good learning rate schedule, and with

heuristics such as early stopping and dropout [6]. The advantage of a BNN is that we only need to define a prior and likelihood, and assuming we have access to an accurate sampler, we can generate valid samples from the posterior. This enables us to decouple any regularization effects of a prior from any side-effects of SGD optimization [7].

Despite the allure of BNNs, several challenges remain. Most notably, it remains unclear what prior to use. Fortuin et al. examined the distribution of NN weights trained using SGD under a uniform prior [8], but this only tells us something about the posterior distribution of parameters under an uninformative prior. What we really want is an informed prior that induces useful biases that help the model better fit the training data and generalize to unseen data. Wenzel et al. have suggested that the cold posterior effect could result from prior misspecification, although they did not suggest a concrete fix [9]. Kapoor et al. introduced a Dirichlet prior over model outputs as a remedy for the cold posterior effect, although we will show that their implementation only works because it numerically approximates a cold likelihood [10].

## 1.2 Posterior predictive distribution

A trained BNN is fully defined by its posterior distribution over parameters. Let's denote the parameters $\boldsymbol{\theta}$, inputs (e.g., images) $\mathbf{X}$ and labels $\mathbf{Y}$. Then we can see that the posterior $p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{X})$ is proportional to a likelihood $p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})$ times a prior $p(\boldsymbol{\theta}|\mathbf{X})$:

$$\begin{aligned} p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) &= p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) \\ p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{X})p(\mathbf{Y}|\mathbf{X}) &= p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta}|\mathbf{X}) \\ p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{X}) &\propto p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta}|\mathbf{X}) \end{aligned} \tag{1.1}$$

An interesting observation here is that the prior can depend on the input data $\mathbf{X}$ but not on the data labels $\mathbf{Y}$. In practice, the dependence on $\mathbf{X}$ is often ignored, for example by setting a Normal prior over parameters. However, the dependence of the prior on the input data is crucial for functional priors like the Dirichlet prior.

The distribution over the response variable $\tilde{\mathbf{y}}$, given a new observation $\tilde{\mathbf{x}}$ and training data

$(\mathbf{X}, \mathbf{Y})$ is given by the posterior predictive distribution:

$$\begin{aligned} p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \mathbf{X}, \mathbf{Y}) &= \int_{-\infty}^{\infty} p(\tilde{\mathbf{y}}, \boldsymbol{\theta}|\tilde{\mathbf{x}}, \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\theta} \\ &= \int_{-\infty}^{\infty} p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\theta} \\ &= \mathbb{E}_{\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}}[p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta})] \end{aligned} \tag{1.2}$$

Unfortunately, the integral in Eq. 1.2 does not have a closed-form solution and requires integrating over the high-dimensional parameters $\boldsymbol{\theta}$. Hence, we must resort to numerical methods to approximate it. A popular approach is *Monte Carlo simulation*: we express the integral as an expectation of the likelihood over the posterior distribution, draw $N$ samples from the posterior, and compute the empirical mean of the likelihood over these samples. This results in a *consistent* estimator of the posterior predictive distribution:

$$\begin{aligned} p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \mathbf{X}, \mathbf{Y}) &= \mathbb{E}_{\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}}[p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta})] \\ &\approx \sum_{i=1}^{N} p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) \\ &\approx \sum_{i=1}^{N} \frac{p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}_i)}{N}, \boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) \end{aligned} \tag{1.3}$$

Lastly, for better numerical precision, we must prevent underflow and work with log-probabilities instead of probabilities:

$$\begin{aligned} \log p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \mathbf{X}, \mathbf{Y}) &\approx \sum_{i=1}^{n} \exp \log p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}_i)) \frac{1}{N} \\ &\approx \log \left( \sum_{i=1}^{n} \exp \log p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}_i) \frac{1}{N} \right) \\ &\approx \mathrm{LogSumExp}_i(\log p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}}, \boldsymbol{\theta}_i)) - \log(N) \end{aligned} \tag{1.4}$$

## 1.3 Hamiltonian Monte Carlo

In section 1.2, we detailed a method to estimate the posterior predictive distribution of a BNN that requires access to samples from the posterior. Often these samples are generated using Markov-Chain Monte Carlo (MCMC), a family of algorithms used for sampling

continuous distributions where direct sampling is not possible. MCMC generates a Markov chain of samples from the target distribution where each sample in the chain depends on the previous sample but is conditionally independent of all the samples that came before. The more dependent (correlated) these samples are, the lower the accuracy of our posterior predictive distribution approximation. Hamiltonian Monte Carlo (HMC) is often considered the "gold standard" of MCMC algorithms because it is capable of generating samples that have very low autocorrelation [11].

The idea behind HMC is that instead of drawing samples directly from the target distribution $\pi(\boldsymbol{\theta})$, we define a new variable $\mathbf{v}$, called the *momentum*, and draw samples from the joint distribution $\pi(\boldsymbol{\theta}, \mathbf{v}) := \pi(\boldsymbol{\theta})\pi(\mathbf{v})$. Once we have the joint samples $(\boldsymbol{\theta}, \mathbf{v})$, we can discard the momentum $\mathbf{v}$ and we are left with samples from our target distribution $\pi(\boldsymbol{\theta})$.

HMC alternates between updating the momentum $\mathbf{v}$ on its own and jointly updating the parameters together with the momentum $(\boldsymbol{\theta}, \mathbf{v})$. In order to understand this process intuitively, it helps to think of $(\boldsymbol{\theta}, \mathbf{v})$ as the state of a physical particle rolling on a hill. The particle has a position $\boldsymbol{\theta}$, momentum $\mathbf{v}$, and the shape of this hill depends on the geometry of the target distribution. First, we flick the particle in a random direction to update its momentum. Then we let the particle roll for a few seconds, which jointly updates its position and momentum together.

More formally, when the momentum is updated on its own, it is drawn from a standard Normal distribution:

$$\mathbf{v} \sim \mathcal{N}(\mathbf{0}, I) \tag{1.5}$$

The joint proposal for $(\boldsymbol{\theta}, \mathbf{v})$ is generated by simulating *Hamiltonian dynamics*:

$$\begin{aligned} \mathrm{d}\boldsymbol{\theta} &= \mathbf{v}\mathrm{d}t \\ \mathrm{d}\mathbf{v} &= \nabla \log \pi(\boldsymbol{\theta})\mathrm{d}t \end{aligned} \tag{1.6}$$

In theory, if the Hamiltonian dynamics were simulated exactly, the sampled values of $\boldsymbol{\theta}$ would follow the target distribution $\pi(\boldsymbol{\theta})$ exactly. Unfortunately, Hamiltonian dynamics describe a continuous system that we have to approximate in discrete steps, typically using the leapfrog algorithm. After each simulation of Hamiltonian dynamics, the new proposal for $\boldsymbol{\theta}$ will get accepted or rejected at random, with the acceptance rate depending on the accuracy of the numerical simulation. To achieve any reasonable acceptance rate (i.e. above 10%), the discretization steps need to be extremely small. However, each step requires

computing the full-batch gradient of the posterior. For models with tens to hundreds of thousands of parameters, this might amount to more than $10,000$ steps (epochs) per proposal, or over 1 million epochs to generate 100 posterior samples. Naturally, this is extremely expensive and very rarely done in practice [4].

**Algorithm 1** HMC

**for** $i \leftarrow 1 \dots n_{\text{samples}}$ **do** ▷ generate $n_{\text{samples}}$ from target distribution
  resample $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, I)$ ▷ resample momentum
  $\boldsymbol{\theta}', \mathbf{v}' \leftarrow \text{Leapfrog}(\boldsymbol{\theta}, \mathbf{v})$ ▷ propose new parameters
  with prob. $\min\left(1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta})} \exp\left(\frac{1}{2}||\mathbf{v}||^2 - ||\mathbf{v}'||^2\right)\right)$, set $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'$ ▷ accept / reject proposal
  $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}$ ▷ save new value
**end for**

**Algorithm 2** Leapfrog

**for** $i \leftarrow 1 \dots n_{\text{steps}}$ **do** ▷ do $n_{\text{steps}}$ steps
  $\mathbf{v} \leftarrow \mathbf{v} + \frac{\epsilon}{2} \nabla \log \pi(\boldsymbol{\theta})$ ▷ do a half-step update of momentum
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon v$ ▷ do a full-step update of parameters
  $\mathbf{v} \leftarrow \mathbf{v} + \frac{\epsilon}{2} \nabla \log \pi(\boldsymbol{\theta})$ ▷ do a half-step update of momentum
**end for**

Crucially, HMC only requires evaluating the target distribution up to a normalizing constant, since the evidence cancels out when computing the posterior ratio in Algorithm 1 and it doesn't affect the gradient in Algorithm 2.

## 1.4 Stochastic Gradient Hamiltonian Monte Carlo

In HMC, each update of momentum requires computing the full-batch gradient: $\Delta \mathbf{v} = \epsilon \nabla \log \pi(\boldsymbol{\theta})$. If we replace the full-batch gradient $\nabla \log \pi(\boldsymbol{\theta})$ with a mini-batch estimate $\nabla \log \tilde{\pi}(\boldsymbol{\theta})$, we can think of this as using the full-batch gradient but adding noise that arises from the mini-batch estimate: $\Delta \mathbf{v} = \epsilon \nabla \log \tilde{\pi}(\boldsymbol{\theta}) = \epsilon \nabla \log \pi(\boldsymbol{\theta}) + \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{V})$. By the central limit theorem, the larger the mini-batch size is, the closer the noise approaches a Normal distribution. Denoting $\mathbf{B} := \frac{1}{2}\epsilon \mathbf{V}$, we can view the noisy mini-batch momentum update as a discretization of the following system:

$$\begin{aligned} \mathrm{d}\boldsymbol{\theta} &= \mathbf{v}\mathrm{d}t \\ \mathrm{d}\mathbf{v} &= \nabla \log \pi(\boldsymbol{\theta})\mathrm{d}t + \mathcal{N}(\mathbf{0}, 2\mathbf{B}\mathrm{d}t) \end{aligned} \tag{1.7}$$

Introducing noise to the dynamics biases the proposed samples toward a uniform distribution [12]. In theory, if we knew the exact scale of the noise, we could compensate for the noise exactly by introducing a friction term to the dynamics:

$$\begin{aligned} \mathrm{d}\boldsymbol{\theta} &= \mathbf{v}\mathrm{d}t \\ \mathrm{d}\mathbf{v} &= \nabla \log \pi(\boldsymbol{\theta})\mathrm{d}t - B\mathbf{v}\mathrm{d}t + \mathcal{N}(\mathbf{0}, 2\mathbf{B}\mathrm{d}t) \end{aligned} \tag{1.8}$$

Unfortunately, in practice, we don't know the true value of the noise scale, so we instead introduce a user-specified friction parameter $C$ to the system. This is where SGHMC turns from an exact method (with unbiased posterior samples) to an approximate method. During the momentum update, in addition to the stochastic mini-batch noise of scale $\mathbf{B}$, we inject Normal noise that exactly compensates for the friction:

$$\begin{aligned} \mathrm{d}\boldsymbol{\theta} &= \mathbf{v}\mathrm{d}t \\ \mathrm{d}\mathbf{v} &= \nabla \log \pi(\boldsymbol{\theta})\mathrm{d}t - C\mathbf{v}\mathrm{d}t + \mathcal{N}(\mathbf{0}, 2CI\mathrm{d}t) + \mathcal{N}(\mathbf{0}, 2\mathbf{B}\mathrm{d}t) \end{aligned} \tag{1.9}$$

The friction no longer compensates for the combined noise and so the generated samples become biased. Fortunately, as $\epsilon \to 0$, the injected noise of scale $C$ dominates the mini-batch noise of scale $\mathbf{B} = \frac{1}{2}\epsilon\mathbf{V}$. Hence, as the step size decreases, the samples become asymptotically unbiased.

Another distinction to HMC is that, unlike standard Hamiltonian dynamics, the system described in Eq. 1.9 is not time-reversible, so computing an acceptance probability is not even possible. This further biases the generated posterior samples. At the same time, since there is no accept/reject step, it is possible to sample the momentum only once and keep it throughout the whole Markov chain, which increases the average trajectory length.

**Algorithm 3** SGHMC

sample $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, I)$ ▷ sample momentum
**for** $i \leftarrow 1 \ldots n_{\text{samples}}$ **do** ▷ generate $n_{\text{samples}}$ from target distribution
  **for** $i \leftarrow 1 \ldots \mathrm{B}$ **do** ▷ iterate over mini-batches
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon\mathbf{v}$ ▷ position update
    $\mathbf{v} \leftarrow (1 - \epsilon C)\mathbf{v} + \epsilon\nabla \log \tilde{\pi}(\boldsymbol{\theta}) + \mathcal{N}(\mathbf{0}, 2CI\epsilon)$ ▷ mini-batch momentum update
  **end for**
  $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}$ ▷ save new value
**end for**

In the limiting behavior of $C \to 0$, SGHMC reduces to HMC although the posterior is

approximated from mini-batches, there is no accept/reject step and the Hamiltonian dynamics are approximated using Euler's method instead of the leapfrog algorithm (leapfrog is more accurate [11]). On the other end, if $C = 1$, SGHMC reduces to Stochastic gradient Langevin dynamics [13].

# Chapter 2

# Cold posteriors

## 2.1 Effect of data augmentation

Deep neural networks have very high model complexity since they are able to perfectly fit pure noise. The reason they generalize to new data at all is the result of their inductive biases [14]. One way to enforce these biases is to train on augmented data that displays various invariances of our physical world. When used together with a cold posterior (more on this later), data augmentation significantly improves the performance of Bayesian neural networks.

Typically, it is assumed that observations in the training dataset are independent, so the likelihood of the whole dataset $(\mathbf{X}, \mathbf{Y})$ factorizes into a product of likelihoods of the individual observations $((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2)...(\mathbf{x}_n, \mathbf{y}_n))$:

$$\begin{aligned} p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) &\propto p(\boldsymbol{\theta}|\mathbf{X})p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X}) \\ &\propto p(\boldsymbol{\theta}|\mathbf{X}) \prod_{i=1}^{N} p(\mathbf{y}_i|\boldsymbol{\theta}, \mathbf{x}_i) \\ \log p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) &\stackrel{c}{=} \log p(\boldsymbol{\theta}|\mathbf{X}) + \sum_{i=1}^{N} \log p(\mathbf{y}_i|\boldsymbol{\theta}, \mathbf{x}_i) \end{aligned} \tag{2.1}$$

Often, computing the full-batch posterior over all $N$ training observations is too expensive, so it is instead approximated using a mini-batch of data of size $B$. To ensure this posterior estimand is unbiased, the mini-batch likelihood has to be rescaled to the size of the full

dataset:

$$\log p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) \overset{c}{\approx} \log p(\boldsymbol{\theta}|\mathbf{X}) + \frac{N}{B}\sum_{i=1}^{B} \log p(\mathbf{y}_i|\boldsymbol{\theta}, \mathbf{x}_i) \tag{2.2}$$

Eq. 2.2 is an unbiased estimator of the posterior value (up to a normalizing constant) as long as it is computed on the original data. However, the way data augmentation is typically implemented is that during each epoch, each datum is randomly augmented but the likelihood is still evaluated in the same way as if the data was not augmented. This biases the posterior estimate since the dataset no longer comprises just $N$ unique data points. For example, if there are 100 unique augmentations for each image, the dataset now has $100N$ unique images but we are still treating the augmented dataset as if it only had $N$ images. At the same time, augmented images are no longer independent, so in this example, the "effective" dataset size would be somewhere between $1N$ and $100N$. If this increased dataset size is not addressed in any way, the true likelihood of the augmented data is "softened" (scaled to a smaller value than it should be).

Suppose we treat the augmented dataset as if it was $1/T$-times bigger than the original dataset, where $T < 1$. We can achieve this by multiplying the log-likelihood by $1/T$, or equivalently, exponentiating the likelihood to $1/T$. The parameter $T$ is called the *temperature* and setting $T < 1$ results in a *cold likelihood*. If we believe that data augmentation increases the effective dataset size at all, we should expect $T < 1$ to more faithfully represent our beliefs about the posterior distribution than $T = 1$. Indeed, many experiments have shown that when data augmentation is used, the performance of a BNN improves with a cold likelihood ($T < 1$) compared to the baseline $T = 1$ [8–10][1]. The categorical likelihood function raised to various temperatures is plotted in 2.1.

[1]Technically, most of these experiments tempered the *posterior* temperature rather than the *likelihood* temperature. However, a cold posterior is equivalent to a cold likelihood with a rescaled prior, and the effects are very closely related.

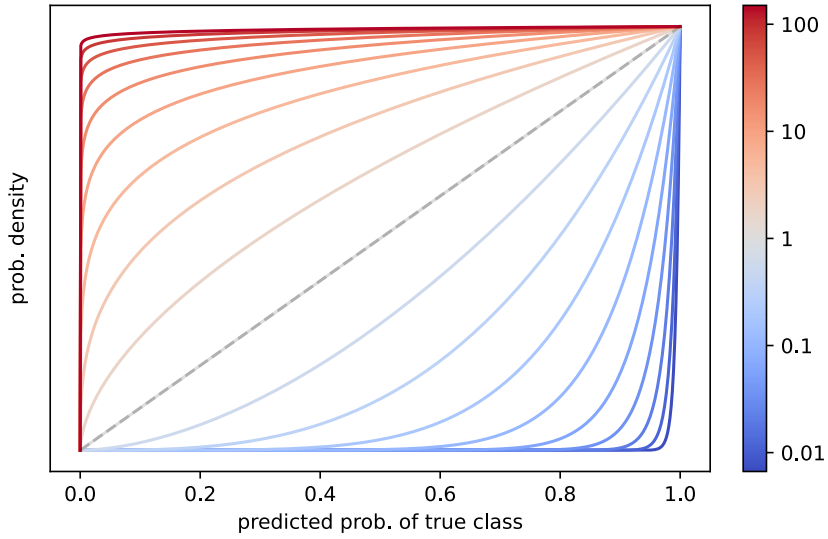


Figure 2.1: A visualization of the categorical likelihood probability density function tempered to various temperatures: $p(x) = x^{1/T}$. The colorbar on the right shows the temperature scale. A cold likelihood ($T < 1$) is "sharp" and penalizes wrong predictions heavily. A warm likelihood ($T > 1$) is "soft" and tolerates wrong predictions. The dashed parallel line shows the untempered likelihood ($T = 1$). Note: the plotted density functions do not integrate to one but MCMC samplers are invariant to normalizing constants, so normalization is not required.

## 2.2 What is a cold posterior

A cold posterior is achieved by exponentiating the posterior to $1/T$, where $T < 1$. Since the posterior factorizes into a prior and a likelihood, a cold posterior can be seen simply as a combination of cold likelihood (Fig. 2.1) with a cold prior:

$$p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X})^{1/T} \propto p(\boldsymbol{\theta}|\mathbf{X})^{1/T} p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})^{1/T} \tag{2.3}$$

If the prior is Normal, then raising it to a temperature $1/T$ is equivalent to adjusting its variance by a factor of $T$:

$$\begin{aligned} N(0, \sigma^2)^{1/T} &\propto \exp\left(-\frac{1}{2}\frac{\theta_i^2}{\sigma^2}\right)^{1/T} \\ &\propto \exp\left(-\frac{1}{2}\frac{\theta_i^2}{(T\sigma^2)}\right) \\ &\sim N(0, T\sigma^2) \end{aligned} \tag{2.4}$$

The decomposition of a cold posterior into a cold prior and a cold likelihood explains the *cold posterior effect* observed in several studies [8–10]. Since most of these experiments used Normal priors over parameters, the effect of the cold prior equated to a different prior scale combined with a cold likelihood (scaling the effective dataset size, which is naturally required for data augmentation).

Typically, a cold posterior is sampled by modifying the SGHMC algorithm instead of directly tempering the posterior. This is to avoid scaling the log-posterior, which could reduce numerical stability. More specifically, we scale the injected noise in line 5 of Algorithm 3 by the temperature $T$:

$$\mathbf{v} \leftarrow (1 - \epsilon C)\mathbf{v} + \epsilon \nabla \log \tilde{\pi}(\boldsymbol{\theta}) + \mathcal{N}(\mathbf{0}, 2TCI\epsilon) \tag{2.5}$$

Naturally, when $T = 1$, Eq. 2.5 is equivalent to the standard SGHMC algorithm but when $T < 1$, it corresponds to sampling $\pi(\boldsymbol{\theta})^{1/T}$ with scaled hyperparameters $C$ and $\epsilon$ [9].

In the limit of $T \to 0$, the cold posterior approaches a deep ensemble. There are two ways to see this: first, a cold temperature sharpens the posterior. As $T \to 0$, the exponent approaches $\infty$, so the distribution becomes increasingly sharp, approaching a distribution with point masses located at posterior modes and equal to zero everywhere else (i.e. a deep ensemble). At the same time, as $T \to 0$, the SGLD noise scale approaches zero, so SGLD becomes equivalent to SGD with momentum.

To summarize, when a Normal prior is used, a cold posterior is equivalent to reducing the prior scale and sharpening the likelihood. At the same time, as $T \to 0$, the cold posterior approaches a deep ensemble. To visualize these results, we trained a ResNet20 [15] on CIFAR-10 [16], with a Normal prior over parameters, using SGHMC and the same augmentation strategy as Wenzel et al. (left/right flip, border-pad, random crop) [9]. We

tested 15 different scales of the Normal prior and 16 different temperatures ranging from 0 to 1 inclusively. The results are shown in Fig. 2.2. Using $T = 1$ results in underfitting. Since the test accuracy can be at most equal to the training accuracy, in order for the test accuracy to improve, the training accuracy must improve, too. When $T \ll 1$, the model reaches 100% training accuracy, so the model *is* capable of fitting the training data perfectly. The main difference between $T = 1$ and $T \ll 1$ is the cold likelihood. Intuitively, a cold likelihood increases the loss associated with misclassifying an image, thus concentrating the posterior on models that fit the training data well.

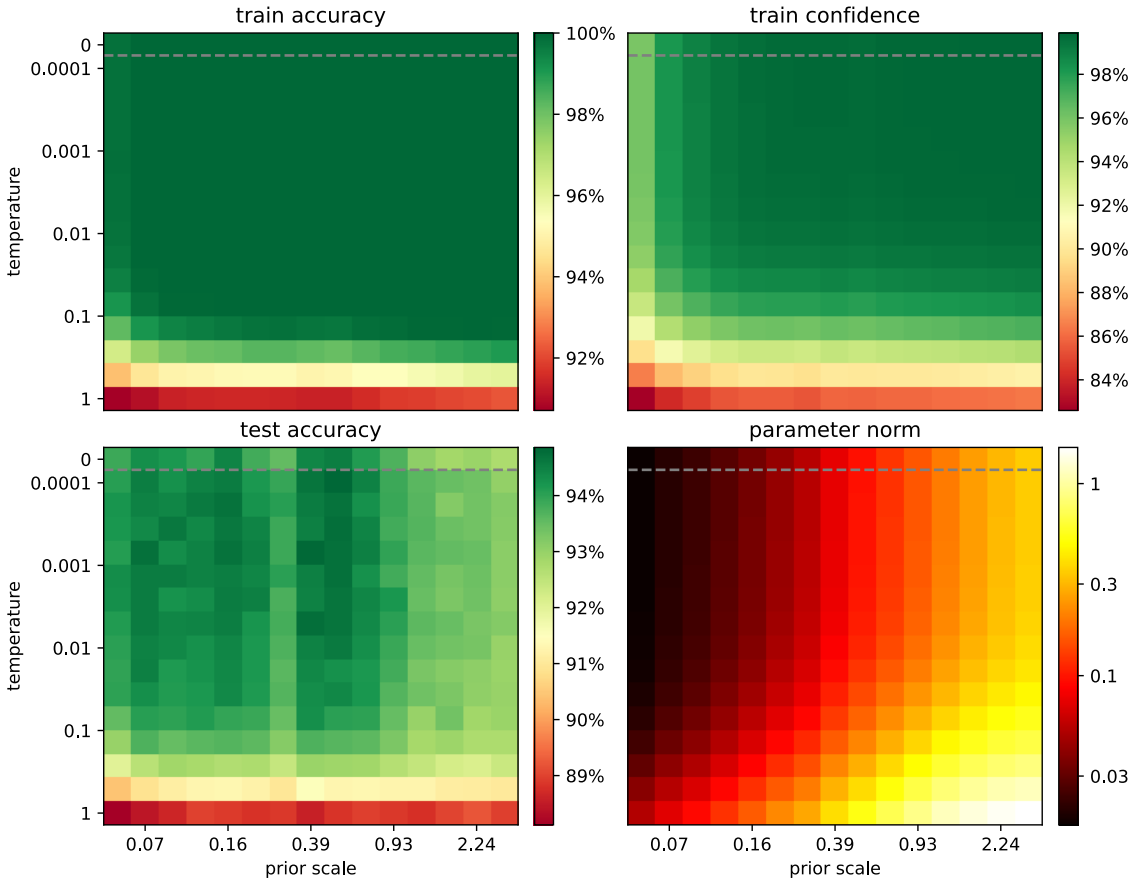


Figure 2.2: The effect of posterior temperature and Normal prior scale on a Bayesian neural network. The model is a ResNet20 trained on CIAR-10 with data augmentation turned on. When temperature = 1, the augmented data is undercounted, resulting in underfitting (low training accuracy). As temperature decreases, the fit to training data improves, increasing test accuracy. Lastly, as temperature decreases, the norm of the model parameters decreases, but only up to a point. With a decreasing temperature, the cold posterior approaches a deep ensemble, which is obtained by setting the temperature to exactly zero.

# Chapter 3

# Dirichlet prior

## 3.1 Motivation

Observe in Fig. 2.2 that training accuracy, training confidence, and test accuracy are all closely correlated. When $T = 1$, the model underfits the training data, resulting in low confidence and low test accuracy; when $T \ll 1$, both confidence and accuracy improve. To avoid having to temper the posterior (or likelihood) temperature, Kapoor et al. [10] have searched for a prior that would result in good posterior accuracy at $T = 1$. Presumably, if we could find a prior that results in high posterior *confidence* at $T = 1$, it would also improve training *accuracy*, thus eliminating underfitting.

Kapoor et al. [10] hypothesize that if a Normal prior was concentrated on models with very low confidence, then the posterior would also have low confidence. However, they never actually measured the confidence of models drawn from a Normal prior, even though it is simple to test. There's no need for expensive MCMC sampling – instead, we can directly sample each model parameter from a Normal distribution, thus obtaining completely unbiased and uncorrelated prior samples.

If we draw a single *sample* from the prior, we expect the model might be confident in a single class, since one of the logits happens to be larger than the other logits. However, the prior has no access to labeled data, so prior samples are equally likely to predict any class. As a consequence, the *prior predictive distribution* (i.e. average over prior samples, as defined in Eq. 1.2) will average out these random predictions, predicting roughly the same probability

for each class. While the prior predictive distribution provides little useful information, the predictions of prior samples *are* informative. For example, if prior samples have high confidence, we might expect posterior samples to also have high confidence, which *would* affect the posterior predictive distribution.

We varied the scale of the Normal prior, sampled parameters for a ResNet20 model, and evaluated the predictions on the training set of CIFAR-10. The dashed line in Fig. 3.1 shows the average confidence of prior samples as a function of the prior scale. Intuitively, as the prior scale tends to zero, the model outputs (logits) tend to zero, inducing uniform predictions over class probabilities, obtaining the lowest possible average confidence (10% for a dataset with 10 classes). Conversely, as the prior scale increases, so does the scale of logits. Inevitably, one of the logits will be larger than the other logits. As the scale of logits gets larger, the absolute differences between the logits grow and the largest logit's corresponding probability will get closer to 1, while others will get closer to 0. Hence as the prior scale gets larger, samples from the prior distribution will approach 100% confidence (in a random class).

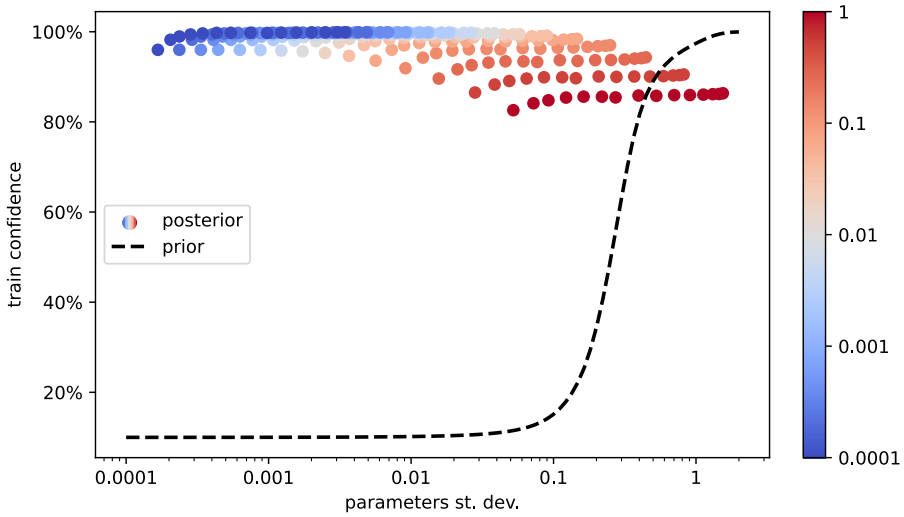


Figure 3.1: Confidence of ResNet20 trained on CIFAR-10 with a Normal prior and data augmentation turned on. The dashed line shows the average confidence of prior samples as a function of the prior scale (standard deviation). The relationship is one-to-one: the prior scale exactly determines prior confidence. Conversely, the scale of posterior samples has almost no effect on posterior confidence – each scatter point corresponds to a single trained model. The intuition that "larger parameters means larger logits" fails. Instead, posterior confidence depends mostly on the posterior temperature, visualized using the colorbar on the right.

However, the confidence of the prior distribution does not trivially translate into posterior confidence. This is empirically clear from Fig. 3.1. When a cold temperature is used, it is possible to obtain a model that has prior confidence near 10% (the lowest possible) *and* posterior confidence of nearly 100%.

These results cast doubt on the claim that underfitting results from low prior confidence. Still, a Normal prior supports a huge range of models, only a tiny fraction of which have meaningful posterior mass, so the link between prior and posterior confidence is not clear. Therefore it is worthwhile investigating if posterior confidence can be affected more *directly* through the prior. Kapoor et al. [10] have suggested doing this through a Dirichlet prior defined over the model outputs. The rest of this report is dedicated to exploring the Dirichlet prior: explaining what it is, introducing novel theory on how to correctly implement it, and experimentally verifying the theory.

## 3.2 Dirichlet distribution

Traditionally, Bayesian neural network priors are defined directly over parameters, and the distribution over parameters induces a distribution over predictions. For example, as discussed in section 3.1, a small-scale Normal prior tends to generate predictions with low confidence whereas a large-scale Normal prior tends to generate predictions with high confidence. The Dirichlet prior works the other way around: it defines a prior over model predictions, which *induces* a distribution over model parameters.

More formally, let's denote the model's predicted class probabilities as $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2 \dots \hat{y}_K)$, where $K$ is the number of classes and $\sum_{k=1}^{K} \hat{y}_k = 1$. The Dirichlet distribution is parametrized by a vector of concentration parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2 \dots \alpha_K)$ and assigns a probability density to any set of predictions $\hat{\mathbf{y}}$:

$$p(\hat{\mathbf{y}}) \propto \prod_{d=1}^{K} \hat{y}_k^{\alpha_k - 1} \tag{3.1}$$

The Dirichlet distribution is conjugate to the categorical distribution: if the prior over predictions is $\hat{\mathbf{y}} \sim \text{Dir}(\boldsymbol{\alpha})$, then the posterior after observing the image label $y$ is also Dirichlet, with updated parameters $\tilde{\boldsymbol{\alpha}}$:

$$\begin{aligned} p(\hat{\mathbf{y}}|y) &\propto p(y|\hat{\mathbf{y}})p(\hat{\mathbf{y}}) \\ &\propto \hat{y}_y \cdot \prod_{k=1}^{K} \hat{y}_k^{\alpha_k - 1} \\ &\propto \hat{y}_y \cdot \prod_{k=1}^{K} \hat{y}_k^{\alpha_k + \mathbb{1}(k=y) - 1} \\ &\sim \text{Dir}(\tilde{\boldsymbol{\alpha}}) \end{aligned} \tag{3.2}$$

We can visualize the Dirichlet prior and posterior on a toy problem. Suppose we have a random generator that outputs the values {`A`, `B`, `C`} and we want to estimate the frequency (probability) of each of these values. Before observing any data, we believe that the generator is heavily biased toward one of the values but we don't know which one. To represent this belief, we place a $\text{Dir}(\alpha = 0.1)$ prior over the frequencies. This prior is visualized on the left side of Fig. 3.2. We sample a single value from the generator and it turns out to be `A`. Combining this observation with our prior belief that the generator is

heavily biased, our posterior belief is that the generator is on average very likely to output the value `A` and unlikely to output values `B` or `C`. This posterior belief is plotted on the right side of Fig. 3.2.

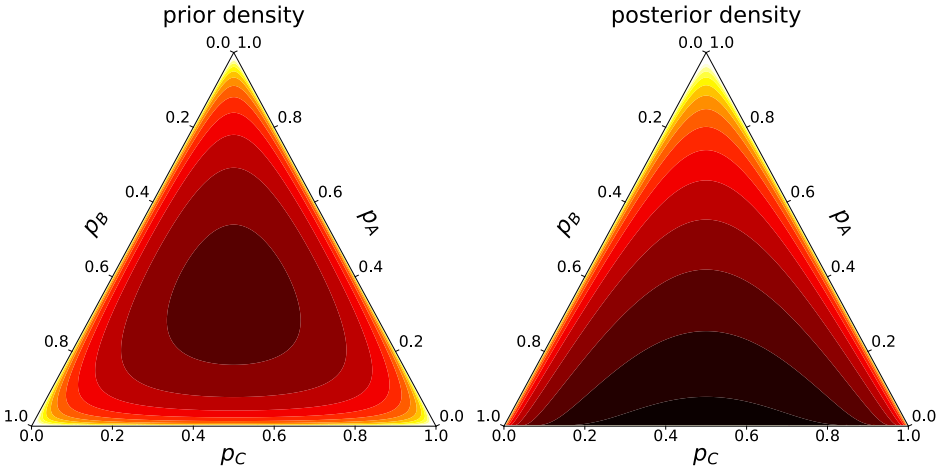


Figure 3.2: **Left:** Dir($\alpha = 0.1$) prior over three probabilities is concentrated around the vertices, where the frequency of one value is close to 1 and the other two are close to 0. **Right:** posterior contracts after a single observation.

Now imagine that we instead held a strong belief that the generator is fair (outputs each value with the same frequency). We could represent this belief using a Dirichlet prior with a much larger concentration parameter (e.g. $\alpha = 10$). In general, we can represent a large range of prior beliefs about the generator by controlling the concentration parameter $\alpha$. Fig. 3.3 shows that depending on which of these priors we select, the posterior might look very different.

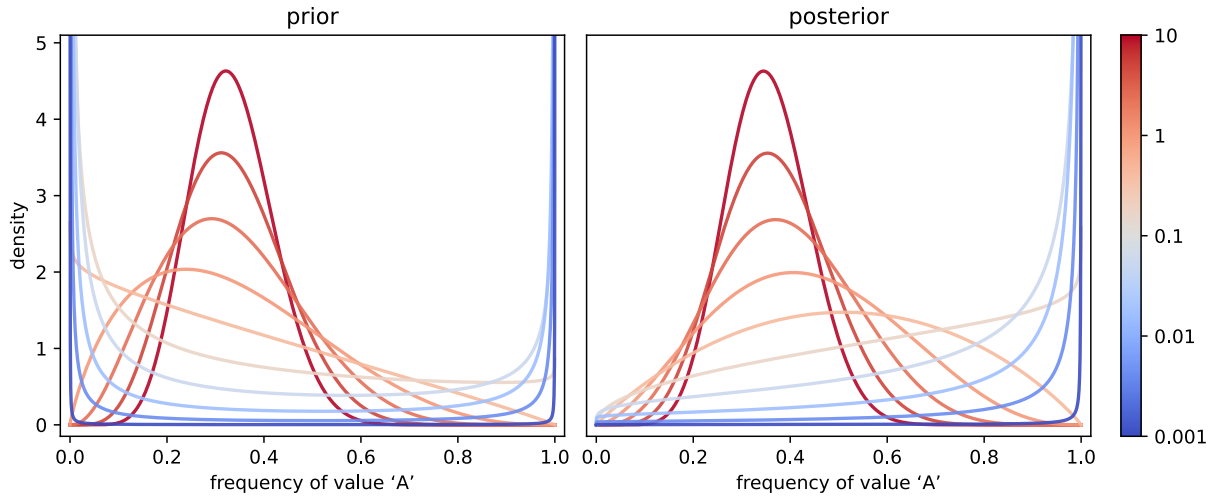


Figure 3.3: **Left:** Dirichlet prior for the generator's frequency of the value `A`. **Right:** posterior belief after a single observation of the value `A`. Each line corresponds to a different prior – the color of the line represents the value of $\alpha$ for that prior. If we believe that the generator mostly outputs a single value (blue), our prior belief is that the frequency of `A` is either 0 or 1. After a single observation of the value `A`, our posterior immediately contracts on the frequency being 1. In contrast, if we have a strong prior belief that the generator is unbiased (red), a single observation will have minimal effect on our posterior beliefs.

We can think of a BNN used for image classification analogously to this random generator. The BNN has a prior distribution over parameters, which induces a prior over outputs. We can set the prior distribution to have a very small concentration parameter $\alpha$, which means the posterior will immediately contract after a single image label, resulting in confident predictions on the training data. In theory, this might enable us to train BNNs that have high confidence at $T = 1$.

## 3.3 Gaussian approximation

Kapoor et al. [10] have observed that directly setting the Dirichlet distribution as the prior $p(\boldsymbol{\theta}) = p(\hat{\mathbf{y}})$ results in divergence. They assumed that this was due to numerical instability but it is not immediately clear where the instability would come from. The log probability density function (log-PDF) of the Dirichlet distribution simply amounts to

taking a weighted sum of the predicted log-probabilities:

$$\log p(\hat{\mathbf{y}}) \stackrel{c}{=} \sum_{k=1}^{K} (\alpha_k - 1) \log \hat{y}_k \tag{3.3}$$

At first glance, Eq. 3.4 doesn't appear numerically any less stable than evaluating the categorical likelihood (or equivalently, the categorical cross-entropy), which are both proportional to the log-probability of the true class. Nonetheless, Kapoor et al. decided to approximate the *posterior* that results from combing the Dirichlet prior with a categorical likelihood, as described in Eq. 3.2. They approximate the posterior over probabilities $\hat{\mathbf{y}}$ with a product of independent Gaussian distributions over the logits $\mathbf{z}$:

$$\begin{aligned} \text{DirGauss}(\hat{\mathbf{y}}|\tilde{\boldsymbol{\alpha}})_k \sim \mathcal{N}(z_k|\mu_k, \sigma_k^2), \text{ where} \\ \sigma_k = \log(\tilde{\alpha}_k^{-1} + 1) \\ \mu_k = \log(\tilde{\alpha}_k) - \sigma_k^2/2 \end{aligned} \tag{3.4}$$

Using this approximate Dirichlet prior, they completely eliminated the cold posterior effect that arises from data augmentation when a ResNet20 is trained on CIFAR-10. Their prior at $T = 1$ matched the performance of a Normal prior at $T \ll 1$, reaching almost 94% test accuracy.

Despite these promising experimental results, there are two immediate concerns with the approximation. First, we are not just using an approximate prior; instead, we are jointly approximating the prior *together* with the likelihood, despite the categorical likelihood being perfectly numerically stable. The main motivation for using the Dirichlet prior was to avoid *tempering* the likelihood, yet we are *tampering* it using an opaque approximation.

Second, the Gaussian approximation visually resembles a cold likelihood much more than it does the true Dirichlet posterior, as shown in Fig. 3.4. Unfortunately, Kapoor et al. [10] have used the Gaussian approximation across all of their experiments, never testing the true Dirichlet posterior. This raises the question: were their experiments successful because the Dirichlet prior actually removes the cold posterior effect or were they successful simply because they secretly approximated a cold likelihood?

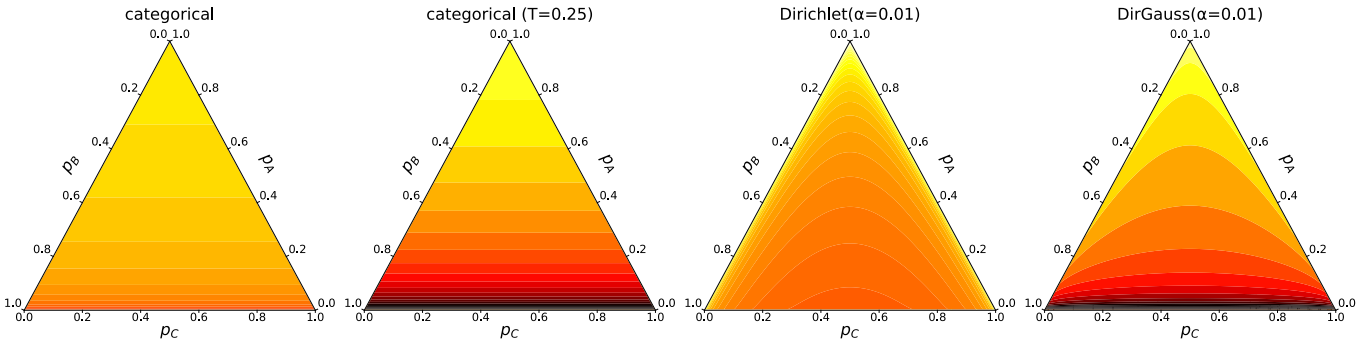


Figure 3.4: Imagine a BNN classifying images into three classes: {`A`, `B`, `C`}. We have passed a single image with label `A` to the model. Typically, we would use the standard categorical likelihood (plotted left) to assign a probability density to the prediction. A cold categorical likelihood (plotted second from left) increases the density assigned to correct predictions, forcing the model to make confident predictions. The Dirichlet posterior also assigns more density to correct predictions, although to a much smaller extent than a cold likelihood. Observe that the Gaussian approximation of the Dirichlet posterior (plotted right) resembles a cold categorical likelihood much more than the true Dirichlet posterior.

In chapter 4, we explain why a simple implementation of the Dirichlet prior diverges: it is, in fact, statistically incorrect. We show theoretically how to fix the implementation and provide early experimental validation in chapter 5.

# Chapter 4

# Change of variables

## 4.1 Priors over outputs

Both regression and classification models are just functions with some inputs, outputs, and parameters. For example, a convolutional neural network is a function that maps images to class probabilities. The goal of model training is to infer the values of parameters of this function that best describe the relationship between inputs and outputs. At the implementation level, we typically work with distributions over model parameters; but at the conceptual level, it is often more useful to think about distributions over functions.

In classical statistical models, a distribution over functions often maps trivially to a distribution over parameters. For example, in linear regression, the weight attached to an input feature tells us both how quickly and in which direction the output will change if we tweak the input. This makes defining informative priors straightforward. If we believe that most features only have a small impact on the output, we place a Normal prior on the parameters, resulting in small weights (ridge regression). Alternatively, if we believe that the output only depends on a small number of input features, we place a Laplace prior on the parameters, forcing many weights to be exactly zero (lasso regression).

Unfortunately, the more complex models get, the harder it becomes to relate a distribution over functions to a distribution over parameters. Perhaps the most challenging example is deep neural networks: their ability to learn hidden hierarchical representations of arbitrary data gives them both great flexibility and extremely low interpretability. In a

fully-connected network, in each layer, each input feature affects each output. This makes it extremely challenging to predict functional properties of the model: the effects of all parameters are inherently interlinked and highly non-linear. It is likely for this reason that, in practice, most Bayesian neural network implementations rely on simple (vague) priors over parameters [8].

One attempt to define a prior directly over model predictions (instead of parameters) is the Dirichlet prior introduced by Kapoor et al. [10]. However, both MCMC and SGD-based optimization methods for BNNs require access to the posterior density over parameters $p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) \propto p(\boldsymbol{\theta}|\mathbf{X})p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})$, rather than a distribution over predictions. How can we translate between the distribution of predictions and parameters? In general, mapping a distribution over one random variable to another variable is difficult. We first illustrate this problem on a simple toy example and then return to the more complicated case of BNN priors.

## 4.2 Toy problem

Imagine we want to use Hamiltonian Monte Carlo (HMC) to draw probability samples $\hat{\mathbf{y}}$ from a Dirichlet distribution with 10 classes: $\hat{\mathbf{y}} \sim \text{Dir}(\alpha = 0.1)$. However, HMC is a gradient-based method, and the domain of probabilities $\hat{\mathbf{y}}$ is constrained to the 9-dimensional simplex of valid probabilities that sum to 1. Without any constraint over the domain $\hat{\mathbf{y}}$, running Hamiltonian dynamics in the domain of probabilities could lead to generating invalid samples. To overcome this problem, we decide to run HMC on the logits $\mathbf{z}$, where $\hat{\mathbf{y}} := \text{softmax}(\mathbf{z})$ and $\mathbf{z} \in \mathbb{R}^{10}$. To summarize this toy problem, we know the distribution of $\text{softmax}(\mathbf{z})$ but the question is how this translates into a distribution over $\mathbf{z}$ directly:

$$\begin{aligned} \text{softmax}(\mathbf{z}) &\sim \text{Dir}(\alpha = 0.1) \\ \mathbf{z} &\sim ? \end{aligned} \tag{4.1}$$

The naive solution would be to assume that we can simply map $\mathbf{z} \to \text{softmax}(\mathbf{z})$ and evaluate the density of $\text{softmax}(\mathbf{z})$, which we already know: $p(\mathbf{z}) = p(\text{softmax}(\mathbf{z}))$. Unfortunately, this is incorrect and results in divergence. The left side of Fig. 4.1 shows the empirical cumulative distribution function (CDF) of eight different HMC chains of

softmax($\mathbf{z}$) generated this way. The logits in each chain diverge so that each chain only contains samples that are entirely concentrated on one of the ten classes, despite the distribution being symmetric. The figure shows the marginal CDF of the first class: it is either concentrated on 0 or 1, depending on which direction HMC diverged in. Note: HMC was run to propose 1,000 samples, each with 1,000 leapfrog steps and an average acceptance rate of 90%. The divergence observed in Fig. 4.1 does *not* result from numerical instability; it is a direct consequence of $p(\mathbf{z})$ being derived incorrectly.

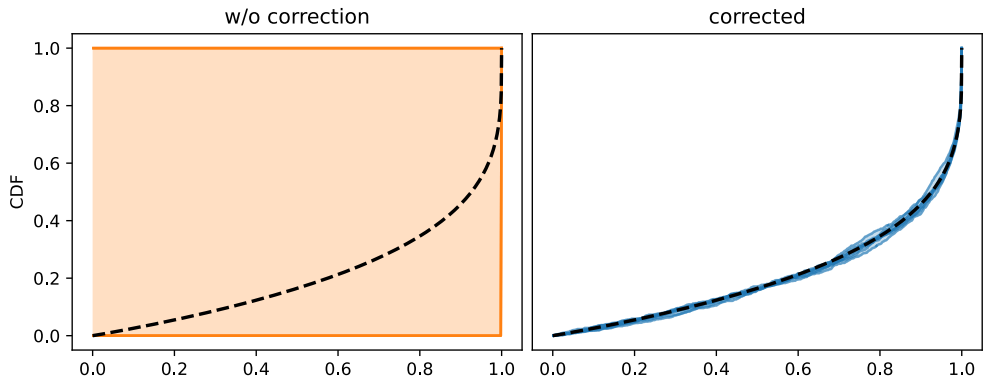


Figure 4.1: Marginal empirical cumulative distribution function (CDF) for softmax($\mathbf{z}$)$_1$, sampled using HMC. Both plots show eight empirical CDFs, estimated from eight independent HMC chains. Each empirical CDF is plotted using a single curve; the area between the minimum and maximum estimate is shaded. The black dashed line shows the true CDF, derived in closed-form. **Left:** naively sampling without a "change of variables" correction term results in divergence and each CDF is completely flat. **Right:** after adding the correction term, the empirical CDFs match the true CDF.

## 4.3 Jacobian determinant

The mistake in the previous example was assuming that probability density is conserved when transforming random variables. In fact, probability density is *not* conserved but probability mass *is* locally conserved. Since $\hat{\mathbf{y}}$ is continuous, the probability mass of $\hat{\mathbf{y}}$ being in some small volume $|\Delta\hat{\mathbf{y}}|$ is approximately $p(\hat{\mathbf{y}})|\Delta\hat{\mathbf{y}}|$. Applying the same logic to

$\mathbf{z}$, we get:

$$\begin{aligned} p(\mathbf{z})|\Delta\mathbf{z}| &= p(\hat{\mathbf{y}})|\Delta\hat{\mathbf{y}}| \\ p(\mathbf{z}) &= p(\hat{\mathbf{y}})\frac{|\Delta\hat{\mathbf{y}}|}{|\Delta\mathbf{z}|} \end{aligned} \tag{4.2}$$

To generalize, let's denote the transform $\mathbf{z} \to \hat{\mathbf{y}}$ as $g$. We know that $\hat{\mathbf{y}} = g(\mathbf{z})$, so to compute $p(\hat{\mathbf{y}})$ for some value of $\mathbf{z}$, we simply map $\mathbf{z} \to \hat{\mathbf{y}}$ and evaluate $p(\hat{\mathbf{y}})$. But what is the relationship between the volumes $|\Delta\mathbf{z}|$ and $|\Delta\hat{\mathbf{y}}|$? The Jacobian of $g$ is a linear approximation of the transform $\mathbf{z} \to \hat{\mathbf{y}}$, so for small values it holds that $\Delta\hat{\mathbf{y}} \approx \mathbf{J}_g(\mathbf{z})\Delta\mathbf{z}$. What we need at this point is to evaluate how $\mathbf{J}_g(\mathbf{z})$ scales the volume of $\Delta\hat{\mathbf{y}}$ as a function of $\Delta\mathbf{z}$. If $\mathbf{J}_g(\mathbf{z})$ is a square matrix, then the scaling is equal to the absolute value of the Jacobian determinant:

$$p(\mathbf{z}) = p(\hat{\mathbf{y}})|\det(\mathbf{J}_g(\mathbf{z}))| \tag{4.3}$$

Using the Jacobian determinant is a standard method when the number of dimensions is preserved. However, when the number of dimensions is not preserved, the Jacobian determinant is undefined and we need a more general method.

## 4.4 Generalization across dimensions

Let's consider the PCA of the Jacobian: $\mathbf{J}_g(\mathbf{z}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, hence $\Delta\hat{\mathbf{y}} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\Delta\mathbf{z}$. First, note that $\mathbf{V}^\top$ is a square matrix of orthonormal vectors, so it has unit determinant and preserves volume. Meanwhile, $\mathbf{U}$ maps $\Delta\hat{\mathbf{y}}$ to the dimensions of $\Delta\mathbf{z}$ using orthonormal vectors. It does not preserve the number of dimensions but it does preserve scaling. The only matrix that performs any scaling is $\mathbf{\Sigma}$, which is a diagonal matrix of singular values. The determinant of this matrix is equal to the product of its diagonal elements. Hence the change of variables formula becomes:

$$p(\mathbf{z}) = p(\hat{\mathbf{y}}) \prod_i \mathbf{\Sigma}_{ii} \tag{4.4}$$

As a sanity check, it is worth noting that if $\mathbf{J}_g(\mathbf{z})$ is square, the product of singular values is equal to $|\det(\mathbf{J}_g(\mathbf{z}))|$, so this general case reduces to the standard Jacobian determinant method. To the best of our knowledge, this generalized method hasn't been introduced before, so we will provide experimental verification to support it. Also, note that it is only

a *local* approximation that might fail for any function that is not one-to-one or due to the numerical instability of SVD.

## 4.5 Logit correction

Let's return to the example of sampling logits $\mathbf{z}$ given the distribution of probabilities $\hat{\mathbf{y}}$. It is easier to think about the transform $\mathbf{z} \to \hat{\mathbf{y}}$ as the combination of two simpler transforms. First, logits are mapped to log-probabilities using the 'logsoftmax' function. Second, log-probabilities are mapped to probabilities by exponentiating them:

$$\begin{aligned} \log \hat{\mathbf{y}} &= \text{logsoftmax}(\mathbf{z}) \\ \text{logsoftmax}(\mathbf{z})_i &= z_i - \log \sum_{k=1}^{K} \exp z_k \end{aligned} \tag{4.5}$$

The "change of variables" correction factor for the combined transform is equal to the product of correction factors for the two simpler transforms. The correction factor for the mapping from log-probabilities to probabilities is simple. The transform is an element-wise exponentiation, so the Jacobian is a diagonal matrix with values equal to the probabilities (Eq. 4.6). The determinant of this Jacobian is equal to the product of diagonals $\prod_{k=1}^{K} \hat{y}_k$.

$$\mathbf{J}_{\text{exp}}(\mathbf{z}) = \begin{bmatrix} \hat{y}_1 & 0 & \cdots & 0 \\ 0 & \hat{y}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{y}_K \end{bmatrix} \tag{4.6}$$

The transform from logits to log-probabilities is more complicated. The domain of logits is $\mathbb{R}^K$ whereas log-probabilities lie on a $K-1$ dimensional manifold. Therefore, the Jacobian determinant of this transform is zero. However, we can still attempt to map a unit volume in logits to a unit surface in log-probabilities using the generalized method introduced in section 4.4. Since log-probabilities have one less degree of freedom, the first $K-1$ singular values of the Jacobian are non-zero and only the last singular value is zero. Notice that the Jacobian can be written as the sum of an identity matrix and the outer product of two vectors:

$$\mathbf{J}_{\text{logsoftmax}}(\mathbf{z}) = I_K - \mathbf{1}_K \otimes \hat{\mathbf{y}} \tag{4.7}$$

The Jacobian has $K-2$ singular values equal to 1, one singular value equal to $\sqrt{K\hat{\mathbf{y}}^\top\hat{\mathbf{y}}}$, and one singular value equal to zero (full derivation is provided in Appendix B). Therefore, the correction factor for this transform is equal to $\sqrt{K\hat{\mathbf{y}}^\top\hat{\mathbf{y}}}$. Unfortunately, computing this factor requires using raw probabilities (rather than log probabilities), which might introduce numerical instability. Additionally, this correction factor is orders of magnitude smaller compared to the correction factor for mapping log-probabilities to probabilities. For these reasons, we neglect this correction factor and only consider the correction factor corresponding to the exponentiation. Intuitively, log probabilities are just logits shifted by a scalar, which approximately preserves scaling.

In summary, when sampling logits $\mathbf{z}$ from a distribution over probabilities $\hat{\mathbf{y}}$, the correction factor $\prod_{k=1}^{K}\hat{y}_k$ is required. For improved numerical stability, it is better to work with the log of the correction term, i.e. the sum of log-probabilities $\sum_{k=1}^{K}\log\hat{y}_k$.

## 4.6 Application to Dirichlet distribution

Recall the toy problem from section 4.2: sampling logits $\mathbf{z}$ given the distribution of probabilities $\hat{\mathbf{y}}$. By applying the correction term, we learn that if $\hat{\mathbf{y}}\sim\mathrm{Dir}(\alpha)$, the density of $\mathbf{z}$ is actually proportional to $\mathrm{Dir}(\alpha+1)$:

$$\begin{aligned} p(\mathbf{z}) &= p(\hat{\mathbf{y}})\frac{|\Delta\hat{\mathbf{y}}|}{|\Delta\mathbf{z}|} \\ &\propto \prod_{d=1}^{K}\hat{y}_k^{\alpha_k-1}\cdot\prod_{k=1}^{K}\hat{y}_k \\ &\propto \prod_{d=1}^{K}\hat{y}_k^{(\alpha_k+1)-1} \end{aligned} \tag{4.8}$$

The left side of Fig. 4.1 shows that if the correction term is omitted, HMC diverges. The right side of the figure shows that when the correction term is included, the distribution of transformed samples of $\mathbf{z}$ perfectly matches the true CDF of $\hat{\mathbf{y}}$. This difference can be understood intuitively. If we let the probability of one class go to zero ($\hat{y}_i\to 0$), the density of the Dirichlet distribution gets arbitrarily high but also the volume of $\hat{\mathbf{y}}$ with this density gets arbitrarily small; there is only a tiny region where $\hat{y}_i\approx 0$, so the probability *mass* of $\hat{y}_i\approx 0$ is actually small. In contrast, the domain of logits is $\mathbf{z}\in\mathbb{R}^K$. If probability

density increases as $z_i \to -\infty$, the distribution over $\mathbf{z}$ is not normalizable and thus invalid. Running HMC on this distribution results in divergence because the sampler continues to propose samples with increasing probability density, diverging toward $-\infty$.

The corrected distribution over $\mathbf{z}$ has density proportional to $\text{Dir}(\alpha+1)$, which interestingly has a mode at $\mathbf{z} = \mathbf{0}$, i.e. $\hat{\mathbf{y}} = \left(\frac{1}{K}, \frac{1}{K} \dots \frac{1}{K}\right)$. This is true even for a very small value of $\alpha$, concentrated at high-confidence probabilities. Note that when a random variable is transformed, its mode is not preserved.

# Chapter 5

# Revisiting the Dirichlet prior

## 5.1 Sampling with correction

Both MCMC and SGD-based optimization methods for BNNs operate on the posterior density over parameters $p(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{X}) \propto p(\boldsymbol{\theta}|\mathbf{X})p(\mathbf{Y}|\boldsymbol{\theta}, \mathbf{X})$. However, the Dirichlet prior is defined over model predictions $\hat{\mathbf{y}}$ rather than parameters $\boldsymbol{\theta}$. From a Bayesian perspective, both $\boldsymbol{\theta}$ and $\hat{\mathbf{y}}$ are random variables. In fact, we know the exact mapping between them: first, the neural network maps parameters $\boldsymbol{\theta}$ (and inputs $\mathbf{X}$, which are constant) to logits $\mathbf{z}$. Second, the softmax function maps logits $\mathbf{z}$ to probabilities $\hat{\mathbf{y}}$.

From chapter 4, we already know that when logits are mapped to probabilities, probability density is not conserved: $p(\hat{\mathbf{y}}) \neq p(\mathbf{z})$. The same argument applies to the mapping from model parameters $\boldsymbol{\theta}$ to logits $\mathbf{z}$. Therefore, the implementation of the Dirichlet prior in Kapoor et al. [10] is fundamentally incorrect. It is not possible to treat a prior over predictions $p(\hat{\mathbf{y}}|\mathbf{X})$ as if it were a prior over parameters $p(\boldsymbol{\theta}|\mathbf{X})$; a correction term that accounts for the mapping $\boldsymbol{\theta} \to \hat{\mathbf{y}}$ is required. Without the correction term, the model will diverge, just as in the toy example from section 4.2.

We can think of the mapping $\boldsymbol{\theta} \to \hat{\mathbf{y}}$ as a composition of two simpler mappings: the neural network maps $\boldsymbol{\theta} \to \mathbf{z}$ and the softmax function maps $\mathbf{z} \to \hat{\mathbf{y}}$. In section 4.5, we derived that the log of the softmax correction is equal to $\sum_{k=1}^{K} \log \hat{y}_k$, which is both numerically stable and cheap to compute. The correction for the mapping $\boldsymbol{\theta} \to \mathbf{z}$ is more problematic: the dimensionality of $\boldsymbol{\theta}$ and $\mathbf{z}$ differs, so the generalized method from section

4.4 is required. It involves computing the product of singular values of the Jacobian of model predictions w.r.t. model parameters, which has both high space *and* time complexity, as well as introducing potential numerical instability.

To experimentally validate these methods, we trained a small CNN and MLP on 1,000 images from MNIST [17]. We tested both the "full" correction (corresponding to the mapping from model parameters to predictions) and a simplified "softmax" correction (which only accounts for the mapping from logits to probabilities). The results are shown in Fig. 5.1. We set a Dir($\alpha$) prior over the model predictions and computed the expected confidence of the Dirichlet distribution for each value of $\alpha$. Next, we used SGHMC to sample both the model prior and the posterior. If the models are implemented correctly, their prior confidence should match the expected confidence of the Dirichlet distribution. Indeed, when the full correction term is used, the confidence of prior samples closely matches the expected confidence. When the simplified "softmax" correction is used, the models at least converge (which they wouldn't without a correction term) but their confidence no longer matches the expected value.

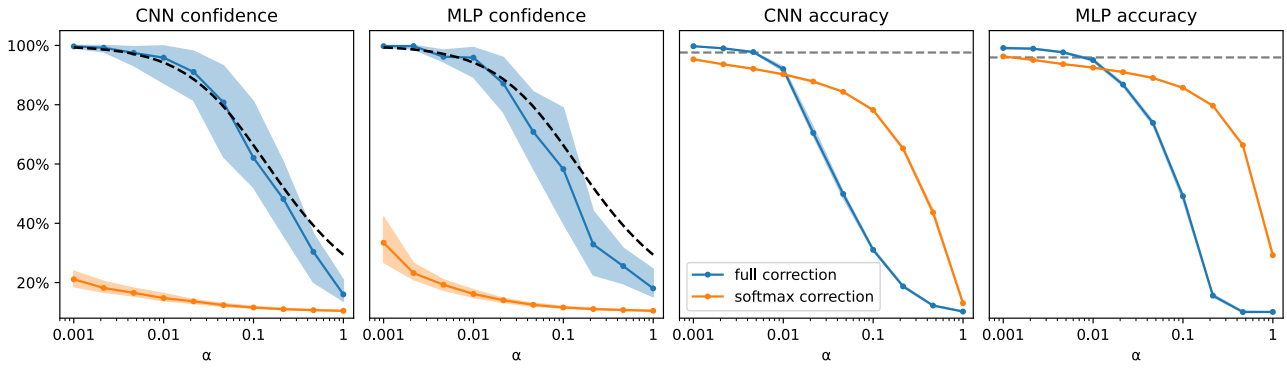


Figure 5.1: Average confidence and accuracy of a small CNN and MLP on MNIST training data. The black dashed line on the left shows the expected confidence of a Dir($\alpha$) distribution. The blue line corresponds to the "full" correction term ($\boldsymbol{\theta} \to \hat{\mathbf{y}}$). The orange line corresponds to the simplified "softmax" correction term ($\mathbf{z} \to \hat{\mathbf{y}}$). As a baseline, the dashed gray line on the right shows the average accuracy of the models under a uniform prior.

Interestingly, the concentration parameter $\alpha$ not only controls the confidence but also the accuracy on training data. A Dir($\alpha = 0.001$) prior results in 99.1% training accuracy, which is 8 standard errors higher than the 95.9% accuracy under the uniform prior. These results are consistent with the cold posterior effect observed in Fig. 2.2, where training

confidence and accuracy were highly correlated. Using a prior that forces a high training confidence, we increased the training accuracy.

Unfortunately, computing the full correction term is extremely expensive and has low numerical stability, requiring many SGHMC steps to achieve convergence. For these reasons, we were unable to train larger models or use more images from MNIST, despite burning over 2,000 TPU-v3-core hours on this task. More details on computational costs are provided in Appendix A.

# Chapter 6

# Discussion

When a Bayesian neural network is trained on augmented data, the likelihood is softened, resulting in underfitting. A simple solution, which has been observed to empirically work well is to cool the posterior [9]. In an attempt to find an alternative solution, Kapoor et al. [10] have proposed to use a Dirichlet prior, which forces the model to be more confident on the training data. However, they found the true Dirichlet prior to be "numerically unstable" and approximated it using a Gaussian distribution over logits across all of their experiments.

**The Dirichlet-Gaussian approximation is secretly a cold likelihood**

The Gaussian approximation introduced by Kapoor et al. is an approximation for the Dirichlet *posterior* (product of prior and likelihood). While this posterior approximation is not directly factorizable into a valid prior and likelihood, it visually resembles a cold likelihood much more than it resembles the true Dirichlet posterior. It is precisely because of this induced *cold likelihood* that their implementation eliminates the cold posterior effect. We argue that a cold likelihood is theoretically sound but we should be explicit about its use, rather than using sophisticated approximations that inadvertently achieve the same result.

**A correction term is required to implement predictive priors**

We demonstrated both theoretically and empirically that simply placing a Dirichlet prior over model outputs results in divergence. To correctly implement a prior over outputs, a "change of variables" term is required, which accounts for how model parameters are

mapped to predictions. We derived a novel approximation for this term that generalizes to mappings that do not preserve the number of dimensions.

**Our implementation enables controlling model confidence**

If a prior over outputs is implemented correctly, then the output of models sampled from the prior should match the specified prior distribution. More specifically, by setting a $\mathrm{Dir}(\alpha)$ prior over the outputs, we obtain prior model samples whose average confidence closely matches the expected confidence of $\mathrm{Dir}(\alpha)$. By forcing the model to be confident on the training data, we are able to achieve a significantly higher training accuracy compared to a uniform prior.

**Future work**

Our work is meant to provide neither a solution to the cold posterior effect nor a practical method of implementing functional priors. Computing the singular values of the full model Jacobian is extremely computationally expensive, numerically unstable, and hard to scale beyond the smallest neural networks. Instead, this work provides a useful first step for correctly thinking about functional priors. We leave scaling the method to larger models and applying it to new functional priors as an exciting direction for future work.

# Acknowledgements

This research was supported by Google's TPU Research Cloud (TRC) program: `https://sites.research.google/trc/`.

# Appendix A

# Implementation details

All experiments were implemented in JAX [18] using TPU v3-8 devices, using Google's TPU Research Cloud (TRC) program. The typical way to draw SGHMC samples is to generate a single chain where each sample depends on the previous samples. Since BNN posteriors are multi-modal [1], generating autocorrelated samples means that the posterior distribution is explored slowly. To eliminate this autocorrelation (and thus achieve more accurate posterior approximations), we instead generated each posterior sample from a different random initialization, completely independently of the other samples.

We followed the learning rate and temperature schedule depicted in Fig. A.1. Initially, the temperature is zero and the learning rate is high so that the sampler quickly converges to a local mode. Afterward, the temperature is increased to explore the posterior, and the learning rate is decreased to reduce the bias of the sampler. At the end of the cycle, only a single posterior sample is produced. We run this procedure in parallel across TPU cores to generate multiple posterior samples.

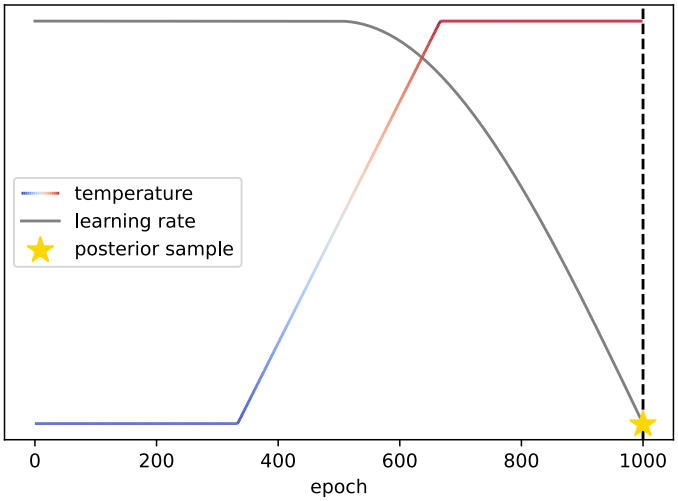


Figure A.1: SGHMC learning rate and temperate schedule for sampling ResNet20 posterior on CIFAR10. Temperature starts at zero, increases linearly after $\frac{1}{3}$ epochs and stays at the maximum value after $\frac{2}{3}$ epochs. Learning rate is constant for $\frac{1}{2}$ epochs and then follows a sine schedule to zero. The same schedules are used for MNIST experiments with the Dirichlet priors except the number of epochs is 10,000.

For the ResNet20+CIFAR10 experiments, we generated 16 posterior samples for each model (using two TPU devices with eight cores each). Fig. A.2 shows that we could further improve the performance of the BNN by generating more samples, although the gains would only be marginal and at a large computational expense. For the Dirichlet experiments on MNIST, we generated 80 posterior samples for each model, since the Jacobian correction term has lower numerical stability and comparison to a concrete theoretical prediction was important.

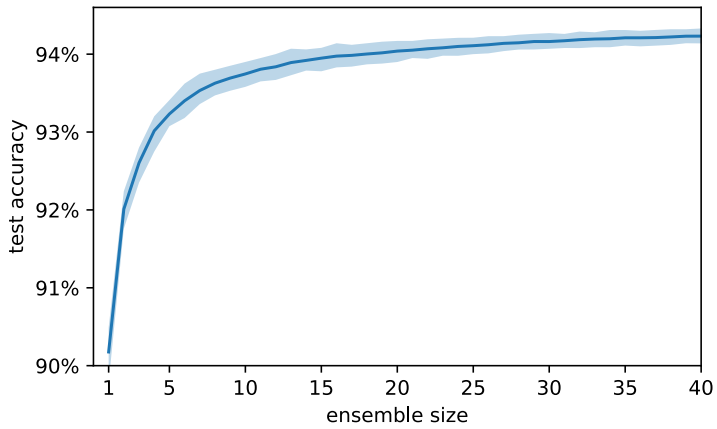


Figure A.2: Test accuracy of a ResNet20 BNN as a function of the number of posterior samples.

Generating a single ResNet20 posterior sample on CIFAR10 took approximately 0.5 TPU-v3 core hours. We spent roughly 4,000 TPU-v3 core hours to get 16 posterior samples for 16 different temperatures and 15 different scales. Sampling Dirichlet models with the Jacobian correction was also expensive, costing roughly an additional 2,000 TPU-v3 core hours, despite only using a small model on a subset of MNIST.

# Appendix B

# Singular values of a rank-1 update

Given $\mathbf{M} = I - \mathbf{1} \otimes \mathbf{u}$, where $\mathbf{1}$ is a vector of ones and $\mathbf{u}$ is a vector of probabilities of length $K$, the matrix $\mathbf{M}$ is a rank-1 update of the identity matrix. Note that since $\mathbf{u}$ is a vector of probabilities, $\mathbf{1} \cdot \mathbf{u} = 1$. The singular values of $\mathbf{M}$ are the square roots of the eigenvalues of $\mathbf{M}^\top\mathbf{M}$. First, let's expand out $\mathbf{M}^\top\mathbf{M}$:

$$\begin{aligned}
\mathbf{M}^\top\mathbf{M} &= (I - \mathbf{u}\mathbf{1}^\top)(I - \mathbf{1}\mathbf{u}^\top) \\
&= I - \mathbf{u}\mathbf{1}^\top - \mathbf{1}\mathbf{u}^\top + \mathbf{u}\mathbf{1}^\top\mathbf{1}\mathbf{u}^\top \\
&= I - \mathbf{u}\mathbf{1}^\top - \mathbf{1}\mathbf{u}^\top + \mathbf{u}K\mathbf{u}^\top \\
&= I - \mathbf{1}\mathbf{u}^\top + \mathbf{u}(K\mathbf{u}^\top - \mathbf{1}^\top)
\end{aligned} \tag{B.1}$$

For any eigenvector $\mathbf{x}$ that is orthogonal to both $\mathbf{1}$ and $\mathbf{u}$, the eigenvalue is 1. There are $K - 2$ such eigenvectors:

$$\begin{aligned}
\mathbf{M}^\top\mathbf{M}\mathbf{x} &= \mathbf{x} - \mathbf{u}(\mathbf{1}^\top\mathbf{x}) - \mathbf{1}(\mathbf{u}^\top\mathbf{x}) + K\mathbf{u}(\mathbf{u}^\top\mathbf{x}) \\
&= \mathbf{x}
\end{aligned} \tag{B.2}$$

Now let's plug in the eigenvector $\mathbf{x}$ that is a linear combination of $\mathbf{1}$ and $\mathbf{u}$; $\mathbf{x} = \alpha\mathbf{1} + \beta\mathbf{u}$:

$$\begin{aligned}
\mathbf{M}^\top\mathbf{M}\mathbf{x} &= (\alpha - \mathbf{u}^\top(\alpha\mathbf{1} + \beta\mathbf{u}))\mathbf{1} + (\beta + (K\mathbf{u}^\top - \mathbf{1}^\top)(\alpha\mathbf{1} + \beta\mathbf{u}))\mathbf{u} \\
&= (\alpha - \alpha\mathbf{u}^\top\mathbf{1} - \beta\mathbf{u}^\top\mathbf{u})\mathbf{1} + (\beta + \alpha K\mathbf{u}^\top\mathbf{1} - \alpha\mathbf{1}^\top\mathbf{1} + \beta K\mathbf{u}^\top\mathbf{u} - \beta\mathbf{1}^\top\mathbf{u})\mathbf{u} \\
&= (\alpha - \alpha - \beta\mathbf{u}^\top\mathbf{u})\mathbf{1} + (\beta + \alpha K - \alpha K + \beta K\mathbf{u}^\top\mathbf{u} - \beta)\mathbf{u} \\
&= -\beta(\mathbf{u}^\top\mathbf{u})\mathbf{1} + \beta K(\mathbf{u}^\top\mathbf{u})\mathbf{u}
\end{aligned} \tag{B.3}$$

Setting $c = \mathbf{u}^\top \mathbf{u}$, this simplifies to:

$$\mathbf{M}^\top \mathbf{M}\mathbf{x} = -\beta c\mathbf{1} + \beta K c\mathbf{u} \tag{B.4}$$

This equation is solved by two different eigenvectors. The first eigenvector is $\mathbf{x} = \mathbf{1}$ (i.e. $\beta = 0$) with corresponding eigenvalue $\lambda = 0$. The second eigenvector is $\mathbf{x} = -\frac{1}{K}\mathbf{1} + \mathbf{u}$ with eigenvalue $cK$.

In summary, the eigenvalues of $\mathbf{M}^\top \mathbf{M}$ are $(1, 1, 1 \ldots K\mathbf{u}^\top \mathbf{u}, 0)$. This corresponds to singular values $(1, 1, 1 \ldots \sqrt{K\mathbf{u}^\top \mathbf{u}}, 0)$.

# Bibliography


[1] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

[2] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.

[3] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

[4] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are bayesian neural network posteriors really like? In *International Conference on Machine Learning*, pages 4629–4640. PMLR, 2021.

[5] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[7] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

[8] Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Ratsch, Richard E Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

[9] Florian Wenzel, Kevin Roth, Bastiaan Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? In Hal DaumÃĺ III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10248–10259. PMLR, 13–18 Jul 2020.

[10] Sanyam Kapoor, Wesley Maddox, Pavel Izmailov, and Andrew Gordon Wilson. On uncertainty, tempering, and data augmentation in bayesian classification. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[11] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[12] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

[13] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[14] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-

Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.