Chapter 3

Bayesian Deep Learning

In previous chapters we reviewed Bayesian neural networks (BNNs) and historical techniques for approximate inference in these, as well as more recent approaches. We discussed the advantages and disadvantages of different techniques, examining their practicality. This, perhaps, is the most important aspect of modern techniques for approximate inference in BNNs. The field of deep learning is pushed forward by practitioners, working on real-world problems. Techniques which cannot scale to complex models with potentially millions of parameters, scale well with large amounts of data, need well studied models to be radically changed, or are not accessible to engineers, will simply perish.

In this chapter we will develop on the strand of work of [Graves, 2011; Hinton and Van Camp, 1993], but will do so from the Bayesian perspective rather than the information theory one. Developing Bayesian approaches to deep learning, we will tie approximate BNN inference together with deep learning stochastic regularisation techniques (SRTs) such as dropout. These regularisation techniques are used in many modern deep learning tools, allowing us to offer a practical inference technique.

We will start by reviewing in detail the tools used by Graves [2011], and extend on these with recent research. In the process we will comment and analyse the variance of several stochastic estimators used in variational inference (VI). Following that we will tie these derivations to SRTs, and propose practical techniques to obtain model uncertainty, even from existing models. We finish the chapter by developing specific examples for image based models (CNNs) and sequence based models (RNNs). These will be demonstrated in chapter 5, where we will survey recent research making use of the suggested tools in real-world problems.

3.1 Advanced techniques in variational inference

We start by reviewing recent advances in VI. We are interested in the posterior over the weights given our observables $\mathbf{X}, \mathbf{Y}$: $p\big(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}\big)$. This posterior is not tractable for a Bayesian NN, and we use variational inference to approximate it. Recall our minimisation objective (eq. (2.6)),

$$\mathcal{L}_{\text{VI}}(\theta) := -\sum_{i=1}^{N} \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})). \tag{3.1}$$

Evaluating this objective poses several difficulties. First, the summed-over terms $\int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega}$ are not tractable for BNNs with more than a single hidden layer. Second, this objective requires us to perform computations over the entire dataset, which can be too costly for large N.

To solve the latter problem, we may use data sub-sampling (also referred to as *mini-batch optimisation*). We approximate eq. (3.1) with

$$\widehat{\mathcal{L}}_{\text{VI}}(\theta) := -\frac{N}{M} \sum_{i \in S} \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \tag{3.2}$$

with a random index set S of size M.

The data sub-sampling approximation forms an unbiased stochastic estimator to eq. (3.1), meaning that $\mathbb{E}_S[\widehat{\mathcal{L}}_{\text{VI}}(\theta)] = \mathcal{L}_{\text{VI}}(\theta)$. We can thus use a stochastic optimiser to optimise this stochastic objective, and obtain a (local) optimum θ^* which would be an optimum to $\mathcal{L}_{\text{VI}}(\theta)$ as well [Robbins and Monro, 1951]. This approximation is often used in the deep learning literature, and was suggested by Hoffman et al. [2013] in the VI context (surprisingly) only recently[1].

The remaining difficulty with the objective in eq. (3.1) is the evaluation of the expected log likelihood $\int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega}$. Monte Carlo integration of the integral has been attempted by some, to varying degrees of success. We will review three approaches used in the VI literature and analyse their variance.

3.1.1 Monte Carlo estimators in variational inference

We often use Monte Carlo (MC) estimation in variational inference to estimate the expected log likelihood (the integral in eq. (3.2)). But more importantly we wish to estimate the expected log likelihood's derivatives w.r.t. the approximating distribution

[1] Although online VI methods processing one point at a time have a long history [Ghahramani and Attias, 2000; Sato, 2001]

parameters θ. This allows us to optimise the objective and find the optimal parameters θ^*. There exist three main techniques for MC estimation in the VI literature (a brief survey of the literature was collected by [Schulman et al., 2015]). These have very different characteristics and variances for the estimation of the expected log likelihood and its derivative. Here we will contrast all three techniques in the context of VI and analyse them both empirically and theoretically.

To present the various techniques we will consider the general case of estimating the *integral derivative*:

$$I(\theta) = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x \tag{3.3}$$

which arises when we optimise eq. (3.2) (we will also refer to a stochastic estimator of this quantity as a *stochastic derivative estimator*). Here $f(x)$ is a function defined on the reals, differentiable almost everywhere (a.e., differentiable on $\mathbb{R}$ apart from a zero measure set), and $p_\theta(x)$ is a probability density function (pdf) parametrised by θ from which we can easily generate samples. We assume that the integral exists and is finite, and that $f(x)$ does not depend on θ. Note that we shall write $f'(x)$ when we differentiate $f(x)$ w.r.t. its input (i.e. $\frac{\partial}{\partial x}f(x)$, in contrast to the differentiation of $f(x)$ w.r.t. other variables).

We further use $p_\theta(x) = \mathcal{N}(x;\mu,\sigma^2)$ as a concrete example, with $\theta = \{\mu,\sigma\}$. In this case, we will refer to an estimator of (3.3) as the *mean derivative estimator* (for some function $f(x)$) when we differentiate w.r.t. $\theta = \mu$, and the *standard deviation derivative estimator* when we differentiate w.r.t. $\theta = \sigma$.

Three MC estimators for eq. (3.3) are used in the VI literature:

1. The *score function* estimator (also known as a *likelihood ratio estimator* and *Reinforce*, [Fu, 2006; Glynn, 1990; Paisley et al., 2012; Williams, 1992]) relies on the identity $\frac{\partial}{\partial\theta}p_\theta(x) = p_\theta(x)\frac{\partial}{\partial\theta}\log p_\theta(x)$ and follows the parametrisation:

$$\begin{aligned}\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x &= \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)\mathrm{d}x \\ &= \int f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}p_\theta(x)\mathrm{d}x\end{aligned} \tag{3.4}$$

   leading to the unbiased stochastic estimator $\widehat{I}_1(\theta) = f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}$ with $x\sim p_\theta(x)$, hence $\mathbb{E}_{p_\theta(x)}[\widehat{I}_1(\theta)] = I(\theta)$. Note that the first transition is possible since x and $f(x)$ do not depend on θ, and only $p_\theta(x)$ depends on it. This estimator is simple and applicable with discrete distributions, but as Paisley et al. [2012] identify, it

has rather high variance. When used in practice it is often coupled with a variance reduction technique.

2. Eq. (3.3) can be re-parametrised to obtain an alternative MC estimator, which we refer to as a *pathwise derivative* estimator (this estimator is also referred to in the literature as the *re-parametrisation trick*, *infinitesimal perturbation analysis*, and *stochastic backpropagation* [Glasserman, 2013; Kingma and Welling, 2013, 2014; Rezende et al., 2014; Titsias and Lázaro-Gredilla, 2014]). Assume that $p_\theta(x)$ can be re-parametrised as $p(\epsilon)$, a parameter-free distribution, s.t. $x = g(\theta, \epsilon)$ with a deterministic *differentiable* bivariate transformation $g(\cdot, \cdot)$. For example, with our $p_\theta(x) = \mathcal{N}(x; \mu, \sigma^2)$ we have $g(\theta, \epsilon) = \mu + \sigma\epsilon$ together with $p(\epsilon) = \mathcal{N}(\epsilon; 0, I)$. Then the following estimator arises:

$$\widehat{I}_2(\theta) = f'(g(\theta, \epsilon))\frac{\partial}{\partial\theta}g(\theta, \epsilon). \tag{3.5}$$

Here $\mathbb{E}_{p(\epsilon)}[\widehat{I}_2(\theta)] = I(\theta)$.

Note that compared to the estimator above where $f(x)$ is used, in this case we use its derivative $f'(x)$. In the Gaussian case, both the derivative w.r.t. μ as well as the derivative w.r.t. σ use $f(x)$'s first derivative (note the substitution of ϵ with $x = \mu + \sigma\epsilon$):

$$\frac{\partial}{\partial\mu}\int f(x)p_\theta(x)\mathrm{d}x = \int f'(x)p_\theta(x)\mathrm{d}x,$$
$$\frac{\partial}{\partial\sigma}\int f(x)p_\theta(x)\mathrm{d}x = \int f'(x)\frac{(x-\mu)}{\sigma}p_\theta(x)\mathrm{d}x,$$

i.e. $\widehat{I}_2(\mu) = f'(x)$ and $\widehat{I}_2(\sigma) = f'(x)\frac{(x-\mu)}{\sigma}$ with $\mathbb{E}_{p_\theta(x)}[\widehat{I}_2(\mu)] = I(\mu)$ and $\mathbb{E}_{p_\theta(x)}[\widehat{I}_2(\sigma)] = I(\sigma)$. An interesting question arises when we compare this estimator ($\widehat{I}_2(\theta)$) to the one above ($\widehat{I}_1(\theta)$): why is it that in one case we use the function $f(x)$, and in the other use its derivative? This is answered below, together with an alternative derivation to that of Kingma and Welling [2013, section 2.4].

The pathwise derivative estimator seems to (empirically) result in a lower variance estimator than the one above, although to the best of my knowledge no proof to this has been given in the literature. An analysis of this parametrisation, together with examples where the estimator has *higher* variance than that of the score function estimator is given below.

3. Lastly, it is important to mention the estimator given in [Opper and Archambeau, 2009, eq. (6-7)] (studied in [Rezende et al., 2014] as well). Compared to both estimators above, Opper and Archambeau [2009] relied on the characteristic function of the Gaussian distribution, restricting the estimator to Gaussian $p_\theta(x)$ alone. The resulting mean derivative estimator, $\frac{\partial}{\partial \mu} \int f(x) p_\theta(x) \mathrm{d}x$, is identical to the one resulting from the pathwise derivative estimator.

   But when differentiating w.r.t. σ, the estimator depends on $f(x)$'s second derivative [Opper and Archambeau, 2009, eq. (19)]:

$$\frac{\partial}{\partial \sigma} \int f(x) p_\theta(x) \mathrm{d}x = 2\sigma \cdot \frac{1}{2} \int f''(x) p_\theta(x) \mathrm{d}x \tag{3.6}$$

   i.e. $\widehat{I}_3(\sigma) = \sigma f''(x)$ with $\mathbb{E}_{p_\theta(x)}[\widehat{I}_3(\sigma)] = I(\sigma)$. This is in comparison to the estimators above that use $f(x)$ or its first derivative. We refer to this estimator as a *characteristic function* estimator.

These three MC estimators are believed to have decreasing estimator variances (1) > (2) > (3). Before we analyse this variance, we offer an alternative derivation to Kingma and Welling [2013]'s derivation of the pathwise derivative estimator.

Remark (Auxiliary variable view of the pathwise derivative estimator)**.** Kingma and Welling [2013]'s derivation of the pathwise derivative estimator was obtained through a change of variables. Instead we justify the estimator through a construction relying on an auxiliary variable augmenting the distribution $p_\theta(x)$.
Assume that the distribution $p_\theta(x)$ can be written as $\int p_\theta(x, \epsilon) \mathrm{d}\epsilon = \int p_\theta(x|\epsilon) p(\epsilon) \mathrm{d}\epsilon$, with $p_\theta(x|\epsilon) = \delta(x - g(\theta, \epsilon))$ a dirac delta function. Then

$$\begin{aligned}
\frac{\partial}{\partial \theta} \int f(x) p_\theta(x) \mathrm{d}x &= \frac{\partial}{\partial \theta} \int f(x) \Big(\int p_\theta(x, \epsilon) \mathrm{d}\epsilon \Big) \mathrm{d}x \\
&= \frac{\partial}{\partial \theta} \int f(x) p_\theta(x|\epsilon) p(\epsilon) \mathrm{d}\epsilon \mathrm{d}x \\
&= \frac{\partial}{\partial \theta} \int \Big(\int f(x) \delta\big(x - g(\theta, \epsilon)\big) \mathrm{d}x \Big) p(\epsilon) \mathrm{d}\epsilon.
\end{aligned}$$

Now, since $\delta(x - g(\theta, \epsilon))$ is zero for all x apart from $x = g(\theta, \epsilon)$,

$$\frac{\partial}{\partial \theta} \int \Big(\int f(x) \delta\big(x - g(\theta, \epsilon)\big) \mathrm{d}x \Big) p(\epsilon) \mathrm{d}\epsilon = \frac{\partial}{\partial \theta} \int f(g(\theta, \epsilon)) p(\epsilon) \mathrm{d}\epsilon$$

$$= \int \frac{\partial}{\partial\theta} f(g(\theta,\epsilon))p(\epsilon)\mathrm{d}\epsilon$$
$$= \int f'(g(\theta,\epsilon))\frac{\partial}{\partial\theta} g(\theta,\epsilon)p(\epsilon)\mathrm{d}\epsilon.$$

This derivation raises an interesting question: why is it that here the function $f(x)$ depends on θ, whereas in the above (the score function estimator) it does not? A clue into explaining this can be found through a measure theoretic view of the score function estimator:

$$\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x = \frac{\partial}{\partial\theta}\int \Big(f(x)p_\theta(x)\Big)\mathrm{d}\lambda(x)$$

where $\lambda(x)$ is the Lebesgue measure. Here the measure does not depend on θ, hence x does not depend on θ. Only the integrand $f(x)p_\theta(x)$ depends on θ, therefore the estimator depends on $\frac{\partial}{\partial\theta}p_\theta(x)$ and $f(x)$. This is in comparison to the pathwise derivative estimator:

$$\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x = \frac{\partial}{\partial\theta}\int f(x)\mathrm{d}p_\theta(x).$$

Here the integration is w.r.t. the measure $p_\theta(x)$, which depends on θ. As a result the random variable x as a measurable function depends on θ, leading to $f(x)$ being a measurable function depending on θ. Intuitively, the above can be seen as "stretching" and "contracting" the space following the density $p_\theta(x)$, leading to a function defined on this space to depend on θ.

3.1.2 Variance analysis of Monte Carlo estimators in variational inference

Next we analyse the estimator variance for the three estimators above using a Gaussian distribution $p_\theta(x) = \mathcal{N}(x;\mu,\sigma^2)$. We will refer to the estimators as the score function estimator (1), the pathwise derivative estimator (2), and the characteristic function estimator (3). We will alternate between the estimator names and numbers indistinguishably.

We begin with the observation that none of the estimators has the lowest variance for *all* functions $f(x)$. For each estimator there exists a function $f(x)$ such that the estimator would achieve lowest variance when differentiating w.r.t. μ, $\frac{\partial}{\partial\boldsymbol{\mu}}\int f(x)p_\theta(x)\mathrm{d}x$. Further, for each estimator there exists a function $f(x)$ (not necessarily the same) such that the estimator would achieve lowest variance when differentiating w.r.t. σ, $\frac{\partial}{\partial\boldsymbol{\sigma}}\int f(x)p_\theta(x)\mathrm{d}x$.

$f(x)$	Score function	Pathwise derivative	Character. function
$x + x^2$	$1.7 \cdot 10^1$	$\mathbf{4.0 \cdot 10^0}$	$\mathbf{4.0 \cdot 10^0}$
$\sin(x)$	$3.3 \cdot 10^{-1}$	$\mathbf{2.0 \cdot 10^{-1}}$	$\mathbf{2.0 \cdot 10^{-1}}$
$\sin(10x)$	$\mathbf{5.0 \cdot 10^{-1}}$	$5.0 \cdot 10^1$	$5.0 \cdot 10^1$

Table 3.1 $\frac{\partial}{\partial \boldsymbol{\mu}} \int f(x) p_\theta(x) \mathrm{d}x$ variance

$f(x)$	Score function	Pathwise derivative	Character. function
$x + x^2$	$8.6 \cdot 10^1$	$9.1 \cdot 10^0$	$\mathbf{0.0 \cdot 10^0}$
$\sin(x)$	$8.7 \cdot 10^{-1}$	$\mathbf{3.0 \cdot 10^{-1}}$	$4.3 \cdot 10^{-1}$
$\sin(10x)$	$\mathbf{1.0 \cdot 10^0}$	$5.0 \cdot 10^1$	$5.0 \cdot 10^3$

Table 3.2 $\frac{\partial}{\partial \boldsymbol{\sigma}} \int f(x) p_\theta(x) \mathrm{d}x$ variance

Table 3.3 Estimator variance for various functions $f(x)$ for the score function estimator, the pathwise derivative estimator, and the characteristic function estimator ((1), (2), and (3) above). On the left is mean derivative estimator variance, and on the right is standard deviation derivative estimator variance, both w.r.t. $p_\theta = \mathcal{N}(\mu, \sigma^2)$ and evaluated at $\mu = 0, \sigma = 1$. In bold is lowest estimator variance.

We assess empirical sample variance of $T = 10^6$ samples for the mean and standard deviation derivative estimators. Tables 3.1 and 3.2 show estimator sample variance for the integral derivative w.r.t. μ and σ respectively, for three different functions $f(x)$. Even though all estimators result in roughly the same means, their variances differ considerably.

For functions with slowly varying derivatives in a neighbourhood of zero (such as the smooth function $f(x) = x + x^2$) we have that the estimator variance obeys (1) > (2) > (3). Also note that mean variance for (2) and (3) are identical, and that σ derivative variance under the characteristic function estimator in this case is zero (since the second derivative for this function is zero). Lastly, note that even though $f(x) = \sin(x)$ is smooth with bounded derivatives, the similar function $f(x) = \sin(10x)$ has variance (3) > (2) > (1). This is because the derivative of the function has high magnitude and varies much near zero.

I will provide a simple property a function has to satisfy for it to have lower variance under the pathwise derivative estimator and the characteristic function estimator than the score function estimator. This will be for the mean derivative estimator.

Proposition 1. *Let $f(x)$, $f'(x)$, $f''(x)$ be real-valued functions s.t. $f(x)$ is an indefinite integral of $f'(x)$, and $f'(x)$ is an indefinite integral of $f''(x)$. Assume that $Var_{p_\theta(x)}((x - \mu)f(x)) < \infty$, and $Var_{p_\theta(x)}(f'(x)) < \infty$, as well as $\mathbb{E}_{p_\theta(x)}(|(x-\mu)f'(x) + f(x)|) < \infty$ and $\mathbb{E}_{p_\theta(x)}(|f''(x)|) < \infty$, with $p_\theta(x) = \mathcal{N}(\mu, \sigma^2)$.*

If it holds that

$$\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x) + f(x)\Big)^2 - \sigma^4 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) \geq 0,$$

then the pathwise derivative and the characteristic function mean derivative estimators w.r.t. the function $f(x)$ will have lower variance than the score function estimator.

Before proving the proposition, I will give some intuitive insights into what the condition means. For this, assume for simplicity that $\mu = 0$ and $\sigma = 1$. This gives the simplified condition

$$\mathbb{E}_{p_\theta(x)}\Big(xf'(x) + f(x)\Big)^2 \geq \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big).$$

First, observe that all expectations are taken over $p_\theta(x)$, meaning that we only care about the functions' average behaviour near zero. Second, a function $f(x)$ with a large derivative absolute value will change considerably, hence will have high variance. Since the pathwise derivative estimator boils down to $f'(x)$ in the Gaussian case, and the score function estimator boils down to $xf(x)$ (shown in the proof), we wish the expected change in $\frac{\partial}{\partial x}xf(x) = xf'(x) + f(x)$ to be higher than the expected change in $\frac{\partial}{\partial x}f'(x) = f''(x)$, hence the condition.

Proof. We start with the observation that the score function mean estimator w.r.t. a Gaussian $p_\theta(x)$ is given by

$$\frac{\partial}{\partial \mu}\int f(x)p_\theta(x)\mathrm{d}x = \int f(x)\frac{x-\mu}{\sigma^2}p_\theta(x)\mathrm{d}x$$

resulting in the estimator $\widehat{I}_1 = f(x)\frac{x-\mu}{\sigma^2}$. The pathwise derivative and the characteristic function estimators are identical for the mean derivative, and given by $\widehat{I}_2 = f'(x)$. We thus wish to show that $\mathrm{Var}_{p_\theta(x)}(\widehat{I}_2) \leq \mathrm{Var}_{p_\theta(x)}(\widehat{I}_1)$ under the proposition's assumptions. Since $\mathrm{Var}_{p_\theta(x)}(\widehat{I}_1) = \frac{\mathrm{Var}_{p_\theta(x)}(xf(x)-\mu f(x))}{\sigma^4}$, we wish to show that

$$\sigma^4\mathrm{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \mathrm{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big).$$

Proposition 3.2 in [Cacoullos, 1982] states that for $g(x)$, $g'(x)$ real-valued functions s.t. $g(x)$ is an indefinite integral of $g'(x)$, and $\mathrm{Var}_{p_\theta(x)}(g(x)) < \infty$ and $\mathbb{E}_{p_\theta(x)}(|g'(x)|) < \infty$, there exists that

$$\sigma^2\mathbb{E}_{p_\theta(x)}\Big(g'(x)\Big)^2 \leq \mathrm{Var}_{p_\theta(x)}\Big(g(x)\Big) \leq \sigma^2\mathbb{E}_{p_\theta(x)}\Big(g'(x)^2\Big).$$

Substituting $g(x)$ with $(x-\mu)f(x)$ we have

$$\sigma^2\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x) + f(x)\Big)^2 \leq \mathrm{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big)$$

and substituting $g(x)$ with $f'(x)$ we have

$$\mathrm{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \sigma^2 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big).$$

Under the proposition's assumption that

$$\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x)+f(x)\Big)^2 - \sigma^4 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) \geq 0$$

we conclude

$$\begin{aligned}\sigma^4 \mathrm{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \sigma^6 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) &\leq \sigma^2 \mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x)+f(x)\Big)^2 \\ &\leq \mathrm{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big)\end{aligned}$$

as we wanted to show. □

For example, with the simple polynomial function $f(x) = x + x^2$ and $\mu = 0, \sigma = 1$ from table 3.3 we have

$$\begin{aligned}\mathbb{E}_{\mathcal{N}(0,1)}\Big(xf'(x)+f(x)\Big)^2 - \mathbb{E}_{\mathcal{N}(0,1)}\Big(f''(x)^2\Big) &= \mathbb{E}_{\mathcal{N}(0,1)}\Big(2x+3x^2\Big)^2 - 2^2 \\ &= 3^2 - 2^2 > 0,\end{aligned}$$

and indeed the score function estimator variance is higher than that of the pathwise derivative estimator and that of the characteristic function estimator. A similar result can be derived for the standard deviation derivative estimator.

From empirical observation, the functions $f(x)$ often encountered in VI seem to satisfy the variance relation (1) > (2). For this reason, and since we will make use of distributions other than Gaussian, we continue our work using the pathwise derivative estimator.

3.2 Practical inference in Bayesian neural networks

We now derive what would hopefully be a practical inference method for Bayesian neural networks. Inspired by the work of Graves [2011], we propose an inference technique that satisfies our definition of *practicality*, making use of the tools above. The work in this section and the coming sections was previously presented in [Gal, 2015; Gal and Ghahramani, 2015a,b,c,d, 2016a,b,c].

In his work, Graves [2011] used both delta approximating distributions, as well as fully factorised Gaussian approximating distributions. As such, Graves [2011] relied on Opper and Archambeau [2009]'s characteristic function estimator in his approximation of eq. (3.2). Further, Graves [2011] factorised the approximating distribution for each weight scalar, losing weight correlations. This approach has led to the limitations discussed in §2.2.2, hurting the method's performance and practicality.

Using the tools above, and relying on the pathwise derivative estimator instead of the characteristic function estimator in particular, we can make use of more interesting non-Gaussian approximating distributions. Further, to avoid losing weight correlations, we factorise the distribution for each weight row $\mathbf{w}_{l,i}$ in each weight matrix $\mathbf{W}_l$, instead of factorising over each *weight scalar*. The reason for this will be given below. Using these two key changes, we will see below how our approximate inference can be closely tied to SRTs, suggesting a practical, well performing, implementation.

To use the pathwise derivative estimator we need to re-parametrise each $q_{\theta_{l,i}}(\mathbf{w}_{l,i})$ as $\mathbf{w}_{l,i} = g(\theta_{l,i}, \epsilon_{l,i})$ and specify some $p(\epsilon_{l,i})$ (this will be done at a later time). For simplicity of notation we will write $p(\boldsymbol{\epsilon}) = \prod_{l,i} p(\epsilon_{l,i})$, and $\boldsymbol{\omega} = g(\theta, \boldsymbol{\epsilon})$ collecting all model random variables. Starting from the data sub-sampling objective (eq. (3.2)), we re-parametrise each integral to integrate w.r.t. $p(\boldsymbol{\epsilon})$:

$$\begin{aligned}\widehat{\mathcal{L}}_{\text{VI}}(\theta) &= -\frac{N}{M}\sum_{i\in S}\int q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \\ &= -\frac{N}{M}\sum_{i\in S}\int p(\boldsymbol{\epsilon})\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\epsilon} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))\end{aligned}$$

and then replace each expected log likelihood term with its stochastic estimator (eq. (3.5)), resulting in a new MC estimator:

$$\widehat{\mathcal{L}}_{\text{MC}}(\theta) = -\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i)) + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \tag{3.7}$$

s.t. $\mathbb{E}_{S,\boldsymbol{\epsilon}}(\widehat{\mathcal{L}}_{\text{MC}}(\theta)) = \mathcal{L}_{\text{VI}}(\theta)$.

Following results in stochastic non-convex optimisation [Rubin, 1981], optimising $\widehat{\mathcal{L}}_{\text{MC}}(\theta)$ w.r.t. θ would converge to the same optima as optimising our original objective $\mathcal{L}_{\text{VI}}(\theta)$. One thus follows algorithm 1 for inference.

Predictions with this approximation follow equation (2.4) which replaces the posterior $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ with the approximate posterior $q_\theta(\boldsymbol{\omega})$. We can then approximate the predictive

Algorithm 1 Minimise divergence between $q_\theta(\boldsymbol{\omega})$ and $p(\boldsymbol{\omega}|X, Y)$

1: Given dataset $\mathbf{X}, \mathbf{Y}$,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly.
4: **repeat**
5: Sample M random variables $\widehat{\boldsymbol{\epsilon}}_i \sim p(\boldsymbol{\epsilon})$, S a random subset of $\{1, .., N\}$ of size M.
6: Calculate stochastic derivative estimator w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{N}{M}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x}_i)) + \frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})).$$

7: Update θ:
$\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$.
8: **until** θ has converged.

distribution with MC integration as well:

$$\begin{aligned}\tilde{q}_\theta(\mathbf{y}^*|\mathbf{x}^*) := \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \widehat{\boldsymbol{\omega}}_t) &\xrightarrow[T\to\infty]{} \int p\Big(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}\Big)q_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\ &\approx \int p\Big(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}\Big)p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\mathrm{d}\boldsymbol{\omega} \\ &= p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\end{aligned} \tag{3.8}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta(\boldsymbol{\omega})$.

We next present distributions $q_\theta(\boldsymbol{\omega})$ corresponding to several SRTs, s.t. standard techniques in the deep learning literature could be seen as identical to executing algorithm 1 for approximate inference with $q_\theta(\boldsymbol{\omega})$. This means that existing models that use such SRTs can be interpreted as performing approximate inference. As a result, uncertainty information can be extracted from these, as we will see in the following sections.

3.2.1 Stochastic regularisation techniques

First, what are SRTs? Stochastic regularisation techniques are techniques used to regularise deep learning models through the injection of stochastic noise into the model. By far the most popular technique is *dropout* [Hinton et al., 2012; Srivastava et al., 2014], but other techniques exist such as multiplicative Gaussian noise (MGN, also referred to as *Gaussian dropout*) [Srivastava et al., 2014], or dropConnect [Wan et al., 2013], among many others [Huang et al., 2016; Krueger et al., 2016; Moon et al., 2015; Singh et al.,

2016]. We will concentrate on dropout for the moment, and discuss alternative SRTs below.

Notation remark. In this section and the next, in order to avoid confusion between matrices (used as weights in a NN) and stochastic random matrices (which are random variables inducing a distribution over BNN weights), we change our notation slightly from §1.1. Here we use $\mathbf{M}$ to denote a *deterministic* matrix over the reals, $\mathbf{W}$ to denote a *random variable* defined over the set of real matrices[2], and use $\widehat{\mathbf{W}}$ to denote a realisation of $\mathbf{W}$.

Dropout is a technique used to avoid over-fitting in neural networks. It was suggested as an ad-hoc technique, and was motivated with sexual breeding metaphors rather than through theoretical foundations [Srivastava et al., 2014, page 1932]. It was introduced several years ago by Hinton et al. [2012] and studied more extensively in [Srivastava et al., 2014]. We will describe the use of dropout in simple single hidden layer neural networks (following the notation of §1.1 with the adjustments above). To use dropout we sample two binary vectors $\widehat{\boldsymbol{\epsilon}}_1, \widehat{\boldsymbol{\epsilon}}_2$ of dimensions Q (input dimensionality) and K (intermediate layer dimensionality) respectively. The elements of the vector $\widehat{\boldsymbol{\epsilon}}_i$ take value 0 with probability $0 \leq p_i \leq 1$ for $i = 1, 2$. Given an input $\mathbf{x}$, we set p_1 proportion of the elements of the input to zero in expectation: $\widehat{\mathbf{x}} = \mathbf{x} \odot \widehat{\boldsymbol{\epsilon}}_1$[3]. The output of the first layer is given by $\mathbf{h} = \sigma(\widehat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b})$, in which we randomly set p_2 proportion of the elements to zero: $\widehat{\mathbf{h}} = \mathbf{h} \odot \widehat{\boldsymbol{\epsilon}}_2$, and linearly transform the vector to give the dropout model's output $\widehat{\mathbf{y}} = \widehat{\mathbf{h}}\mathbf{M}_2$. We repeat this for multiple layers.

We sample new realisations for the binary vectors $\widehat{\boldsymbol{\epsilon}}_i$ for every input point and every forward pass through the model (evaluating the model's output), and use the same values in the backward pass (propagating the derivatives to the parameters to be optimised $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$). At test time we do not sample any variables and simply use the original units $\mathbf{x}, \mathbf{h}$ scaled by $\frac{1}{1-p_i}$.

Multiplicative Gaussian noise is similar to dropout, where the only difference is that $\widehat{\boldsymbol{\epsilon}}_i$ are vectors of draws from a Gaussian distribution $\mathcal{N}(1, \alpha)$ with a positive parameter α, rather than draws from a Bernoulli distribution.

[2]The reason for this notation is that $\mathbf{M}$ will often coincide with the mean of the random matrix $\mathbf{W}$.
[3]Here $\odot$ is the element-wise product.

3.2.2 Stochastic regularisation techniques as approximate inference

Dropout and most other SRTs view the injected noise as applied in the feature space (the input features to each layer: $\mathbf{x}, \mathbf{h}$). In Bayesian NNs, on the other hand, the stochasticity comes from our uncertainty over the model parameters. We can transform dropout's noise from the feature space to the parameter space as follows[4]:

$$\begin{aligned}
\widehat{\mathbf{y}} &= \widehat{\mathbf{h}}\mathbf{M}_2 \\
&= (\mathbf{h} \odot \widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2 \\
&= (\mathbf{h} \cdot \text{diag}(\widehat{\boldsymbol{\epsilon}}_2))\mathbf{M}_2 \\
&= \mathbf{h}(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big(\widehat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big((\mathbf{x} \odot \widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1 + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big(\mathbf{x}(\text{diag}(\widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1) + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2)
\end{aligned}$$

writing $\widehat{\mathbf{W}}_1 := \text{diag}(\widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1$ and $\widehat{\mathbf{W}}_2 := \text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2$ we end up with

$$\widehat{\mathbf{y}} = \sigma\Big(\mathbf{x}\widehat{\mathbf{W}}_1 + \mathbf{b}\Big)\widehat{\mathbf{W}}_2 =: \mathbf{f}^{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}}(\mathbf{x})$$

with random variable realisations as weights, and write $\widehat{\boldsymbol{\omega}} = \{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}\}$.

This allows us to write dropout's objective in a more convenient form. Recall that a neural network's optimisation objective is given by eq. (1.3). For dropout it will simply be:

$$\widehat{\mathcal{L}}_{\text{dropout}}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}) := \frac{1}{M} \sum_{i \in S} E^{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}}(\mathbf{x}_i, \mathbf{y}_i) + \lambda_1 ||\mathbf{M}_1||^2 + \lambda_2 ||\mathbf{M}_2||^2 + \lambda_3 ||\mathbf{b}||^2, \tag{3.9}$$

with $\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i$ corresponding to new masks $\widehat{\boldsymbol{\epsilon}}_1^i, \widehat{\boldsymbol{\epsilon}}_2^i$ sampled for each data point i. Here we used data sub-sampling with a random index set S of size M, as is common in deep learning.

[4]Here the diag(·) operator maps a vector to a diagonal matrix whose diagonal is the elements of the vector.

As shown by Tishby et al. [1989], in regression $E^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x},\mathbf{y})$ can be rewritten as the negative log-likelihood scaled by a constant:

$$E^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) = \frac{1}{2}||\mathbf{y} - \mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})||^2 = -\frac{1}{\tau}\log p(\mathbf{y}|\mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})) + \text{const} \tag{3.10}$$

where $p(\mathbf{y}|\mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})) = \mathcal{N}(\mathbf{y}; \mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x}), \tau^{-1}I)$ with τ^{-1} observation noise. It is simple to see that this holds for classification as well (in which case we should set $\tau = 1$).

Recall that $\widehat{\boldsymbol{\omega}} = \{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}\}$ and write

$$\widehat{\boldsymbol{\omega}}_i = \{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}\} = \{\text{diag}(\widehat{\boldsymbol{\epsilon}}_1^i)\mathbf{M}_1, \text{diag}(\widehat{\boldsymbol{\epsilon}}_2^i)\mathbf{M}_2, \mathbf{b}\} =: g(\theta, \widehat{\boldsymbol{\epsilon}}_i)$$

with $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$, $\widehat{\boldsymbol{\epsilon}}_1^i \sim p(\boldsymbol{\epsilon}_1)$, and $\widehat{\boldsymbol{\epsilon}}_2^i \sim p(\boldsymbol{\epsilon}_2)$ for $1 \leq i \leq N$. Here $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ is a product of Bernoulli distributions with probabilities $1 - p_l$, from which a realisation would be a vector of zeros and ones.

We can plug identity (3.10) into objective (3.9) and get

$$\widehat{\mathcal{L}}_{\text{dropout}}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}) = -\frac{1}{M\tau}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2 \tag{3.11}$$

with $\widehat{\boldsymbol{\epsilon}}_i$ realisations of the random variable $\boldsymbol{\epsilon}$.

The derivative of this optimisation objective w.r.t. model parameters $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$ is given by

$$\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{dropout}}(\theta) = -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \frac{\partial}{\partial\theta}\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big).$$

The optimisation of a NN with dropout can then be seen as following algorithm 2.

There is a great sense of similarity between algorithm 1 for approximate inference in a Bayesian NN and algorithm 2 for dropout NN optimisation. Specifically, note the case of a Bayesian NN with approximating distribution $q(\boldsymbol{\omega})$ s.t. $\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ a product of Bernoulli distributions with probability $1 - p_l$ (which we will refer to as a *Bernoulli variational distribution* or a *dropout variational distribution*). The only differences between algorithm 1 and algorithm 2 are

1. the regularisation term derivatives ($\text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$ in algo. 1 and $\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2$ in algo. 2),
2. and the scale of $\widehat{\Delta\theta}$ (multiplied by a *constant* $\frac{1}{N\tau}$ in algo. 2).

Algorithm 2 Optimisation of a neural network with dropout

1: Given dataset $\mathbf{X}, \mathbf{Y}$,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly.
4: **repeat**
5: Sample M random variables $\widehat{\boldsymbol{\epsilon}}_i \sim p(\boldsymbol{\epsilon})$, S a random subset of $\{1, .., N\}$ of size M.
6: Calculate derivative w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \frac{\partial}{\partial\theta}\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big).$$

7: Update θ:
$\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$.
8: **until** θ has converged.

More specifically, if we define the prior $p(\boldsymbol{\omega})$ s.t. the following holds:

$$\frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = \frac{\partial}{\partial\theta}N\tau\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big) \tag{3.12}$$

(referred to as the *KL condition*), we would have the following relation between the derivatives of objective (3.11) and objective (3.7):

$$\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{dropout}}(\theta) = \frac{1}{N\tau}\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{MC}}(\theta)$$

with identical optimisation procedures!

We found that for a specific choice for the approximating distribution $q_\theta(\boldsymbol{\omega})$, VI results in identical optimisation procedure to that of a dropout NN. I would stress that this means that optimising *any* neural network with dropout is equivalent to a form of approximate inference in a probabilistic interpretation of the model[5]. This means that the optimal weights found through the optimisation of a dropout NN (using algo. 2) are the same as the optimal variational parameters in a Bayesian NN with the same structure. Further, this means that a network already trained with dropout *is* a Bayesian NN, thus possesses all the properties a Bayesian NN possesses.

We have so far concentrated mostly on the dropout SRT. As to alternative SRTs, remember that an approximating distribution $q_\theta(\boldsymbol{\omega})$ is defined through its re-parametrisation $\boldsymbol{\omega} = g(\theta, \boldsymbol{\epsilon})$. Various SRTs can be recovered for different re-parametrisations. For exam-

[5]Note that to get well-calibrated uncertainty estimates we have to optimise the dropout probability p as well as θ, for example through grid-search over over validation log probability. This is discussed further in §4.3

ple, multiplicative Gaussian noise [Srivastava et al., 2014] can be recovered by setting $g(\theta, \boldsymbol{\epsilon}) = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ (for $l = 1, 2$) a product of $\mathcal{N}(1, \alpha)$ with positive-valued α[6]. This can be efficiently implemented by multiplying a network's units by i.i.d. draws from a $\mathcal{N}(1, \alpha)$. On the other hand, setting $g(\theta, \boldsymbol{\epsilon}) = \{\mathbf{M}_1 \odot \boldsymbol{\epsilon}_1, \mathbf{M}_2 \odot \boldsymbol{\epsilon}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ a product of Bernoulli random variables for each weight scalar we recover dropConnect [Wan et al., 2013]. This can be efficiently implemented by multiplying a network's weight scalars by i.i.d. draws from a Bernoulli distribution. It is interesting to note that Graves [2011]'s fully factorised approximation can be recovered by setting $g(\theta, \boldsymbol{\epsilon}) = \{\mathbf{M}_1 + \boldsymbol{\epsilon}_1, \mathbf{M}_2 + \boldsymbol{\epsilon}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ a product of $\mathcal{N}(0, \alpha)$ for each weight scalar. This SRT is often referred to as *additive Gaussian noise*.

3.2.3 KL condition

For VI to result in an identical optimisation procedure to that of a dropout NN, the KL condition (eq. (3.12)) has to be satisfied. Under what constraints does the KL condition hold? This depends on the model specification (selection of prior $p(\boldsymbol{\omega})$) as well as choice of approximating distribution $q_\theta(\boldsymbol{\omega})$. For example, it can be shown that setting the model prior to $p(\boldsymbol{\omega}) = \prod_{i=1}^{L} p(\mathbf{W}_i) = \prod_{i=1}^{L} \mathcal{N}(0, \mathbf{I}/l_i^2)$, in other words independent normal priors over each weight, with *prior length-scale*[7]

$$l_i^2 = \frac{2N\tau\lambda_i}{1 - p_i} \tag{3.13}$$

we have

$$\frac{\partial}{\partial\theta}\text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \approx \frac{\partial}{\partial\theta} N\tau(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2)$$

for a large enough number of hidden units and a Bernoulli variational distribution. This is discussed further in appendix A. Alternatively, a discrete prior distribution

$$p(\mathbf{w}) \propto e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}}$$

[6]For the multiplicative Gaussian noise this result was also presented in [Kingma et al., 2015], which was done in parallel to this work.

[7]A note on *mean-squared-error* losses: the mean-squared-error loss can be seen as a scaling of the Euclidean loss (eq. (1.1)) by a factor of 2, which implies that the factor of 2 in the length-scale should be removed. The mean-squared-error loss is used in many modern deep learning packages instead (or as) the Euclidean loss.

defined over a *finite* space $\mathbf{w} \in X$ satisfies the KL condition (eq. (3.12)) exactly. This is discussed in more detail in §6.5. For multiplicative Gaussian noise, Kingma et al. [2015] have shown that an improper log-uniform prior distribution satisfies our KL condition.

Notation remark. In the rest of this work we will use $\mathbf{M}$ and $\mathbf{W}$ interchangeably, with the understanding that in deterministic NNs the random variable $\mathbf{W}$ will follow a delta distribution with mean parameter $\mathbf{M}$.

Remark (What is a prior length-scale l_i^2?)**.** It is interesting to explain why we consider the parameter l to be a prior *length-scale* in a Bayesian NN when setting a Gaussian prior $\mathcal{N}(0, I/l^2)$ over the weights. To see this, re-parametrise the input prior distribution as $\mathbf{W}_1' \sim \mathcal{N}(0, I)$, with $\mathbf{W}_1 = \mathbf{W}_1'/l$:

$$\widehat{\mathbf{y}} = \sigma\left(\mathbf{x}\frac{\mathbf{W}_1'}{l} + \mathbf{b}\right)\mathbf{W}_2 = \sigma\left(\frac{\mathbf{x}}{l}\mathbf{W}_1' + \mathbf{b}\right)\mathbf{W}_2$$

i.e. placing a prior distribution $\mathcal{N}(0, I/l^2)$ over $\mathbf{W}_1$ can be replaced by scaling the inputs by $1/l$ with a $\mathcal{N}(0, I)$ prior instead. For example, multiplying the inputs by 100 (making the function smoother) and placing a prior length-scale $l = 100$ would give identical model output to placing a prior length-scale $l = 1$ with the original inputs. This means that the length-scale's unit of measure is identical to the inputs' one.
What does the prior length-scale mean? To see this, consider a real valued function $f(x)$, periodic with period P, and consider its Fourier expansion with K terms:

$$f_K(x) := \frac{A_0}{2} + \sum_{k=1}^{K} A_k \cdot \sin\left(\tfrac{2\pi k}{P}x + \phi_k\right).$$

This can be seen as a single hidden layer neural network with a non-linearity $\sigma(\cdot) := \sin(\cdot)$, input weights given by the Fourier frequencies $\mathbf{W}_1 := [\frac{2\pi k}{P}]_{k=1}^K$ (which are fixed and not learnt), $\mathbf{b} := [\phi_k]_{k=1}^K$ is the bias term of the hidden layer, the Fourier coefficients are the output weights $\mathbf{W}_2 := [A_k]_{k=1}^K$, and $\frac{A_0}{2}$ is the output bias (which can be omitted for centred data). For simplicity we assume that $\mathbf{W}_1$ is composed of only the Fourier frequencies for which the Fourier coefficients are not zero. For example, $\mathbf{W}_1$ might be composed of high frequencies, low frequencies, or a combination of the two.

This view of single hidden layer neural networks gives us some insights into the role of the different quantities used in a neural network. For example, erratic functions have high frequencies, i.e. high magnitude input weights $\mathbf{W}_1$. On the other hand, smooth slow-varying functions are composed of low frequencies, and as a result the magnitude of $\mathbf{W}_1$ is small. The magnitude of $\mathbf{W}_2$ determines how much different frequencies will be used to compose the output function $f_K(x)$. High magnitude $\mathbf{W}_2$ results in a large magnitude for the function's outputs, whereas low $\mathbf{W}_2$ magnitude gives function outputs scaled down and closer to zero.

When we place a prior distribution over the input weights of a BNN, we can capture this characteristic. Having $\mathbf{W}_1 \sim \mathcal{N}(0, I/l^2)$ a priori with long length-scale l results in weights with low magnitude, and as a result slow-varying induced functions. On the other hand, placing a prior distribution with a short length-scale gives high magnitude weights, and as a result erratic functions with high frequencies. This will be demonstrated empirically in §4.1.

Given the intuition about weight magnitude above, equation (3.13) can be re-written to cast some light on the structure of the weight-decay in a neural network[a]:

$$\lambda_i = \frac{l_i^2(1-p_i)}{2N\tau}. \tag{3.14}$$

A short length-scale l_i (corresponding to high frequency data) with high precision τ (equivalently, small observation noise) results in a small weight-decay λ_i—encouraging the model to fit the data well but potentially generalising badly. A long length-scale with low precision results in a large weight-decay—and stronger regularisation over the weights. This trade-off between the length-scale and model precision results in different weight-decay values.

Lastly, I would comment on the choice of placing a distribution over the rows of a weight matrix rather than factorising it over each row's elements. Gal and Turner [2015] offered a derivation related to the Fourier expansion above, where a function drawn from a Gaussian process (GP) was approximated through a finite Fourier decomposition of the GP's covariance function. This derivation has many properties in common with the view above. Interestingly, in the multivariate $\mathbf{f}(\mathbf{x})$ case the Fourier frequencies are given in the columns of the equivalent weight matrix $\mathbf{W}_1$ of size Q (input dimension) by K (number of expansion terms). This generalises on the univariate case above where $\mathbf{W}_1$ is of dimensions $Q = 1$ by K and each entry (column) is a single frequency. Factorising the weight matrix approximating

distribution $q_\theta(\mathbf{W}_1)$ over its *rows* rather than columns captures correlations over the function's frequencies.

[a]Note that with a *mean-squared-error* loss the factor of 2 should be removed.

3.3 Model uncertainty in Bayesian neural networks

We next derive results extending on the above showing that model uncertainty can be obtained from NN models that make use of SRTs such as dropout.

Recall that our approximate predictive distribution is given by eq. (2.4):

$$q_\theta^*(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*))q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \tag{3.15}$$

where $\boldsymbol{\omega} = \{\mathbf{W}_i\}_{i=1}^L$ is our set of random variables for a model with L layers, $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)$ is our model's stochastic output, and $q_\theta^*(\boldsymbol{\omega})$ is an optimum of eq. (3.7).

We will perform moment-matching and estimate the first two moments of the predictive distribution empirically. The first moment can be estimated as follows:

Proposition 2. *Given* $p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)) = \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})$ *for some* $\tau > 0$, $\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]$ *can be estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}[\mathbf{y}^*] := \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) \xrightarrow[T\to\infty]{} \mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] \tag{3.16}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$.

Proof.

$$\begin{aligned}
\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] &= \int \mathbf{y}^* q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)\mathrm{d}\mathbf{y}^* \\
&= \int\int \mathbf{y}^* \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I}) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\mathrm{d}\mathbf{y}^* \\
&= \int \left(\int \mathbf{y}^* \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})\mathrm{d}\mathbf{y}^*\right) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\
&= \int \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega},
\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}[\mathbf{y}^*] := \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ following MC integration with T samples. □

When used with dropout, we refer to this Monte Carlo estimate (3.16) as *MC dropout*. In practice MC dropout is equivalent to performing T stochastic forward passes through the network and averaging the results. For dropout, this result has been presented in the literature before as *model averaging* [Srivastava et al., 2014]. We have given a new derivation for this result which allows us to derive mathematically grounded uncertainty estimates as well, and generalises to all SRTs (including SRTs such as multiplicative Gaussian noise where the model averaging interpretation would result in infinitely many models). Srivastava et al. [2014, section 7.5] have reasoned based on empirical experimentation that the model averaging can be approximated by multiplying each network unit $\mathbf{h}_i$ by $1/(1-p_i)$ at test time, referred to as *standard dropout*. This can be seen as propagating the mean of each layer to the next. Below (in section §4.4) we give results showing that there exist models in which standard dropout gives a bad approximation to the model averaging.

We estimate the second raw moment (for regression) using the following proposition:

Proposition 3.
Given $p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)) = \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})$ *for some* $\tau > 0$, $\mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\Big[(\mathbf{y}^*)^T(\mathbf{y}^*)\Big]$ *can be estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}\Big[(\mathbf{y}^*)^T(\mathbf{y}^*)\Big] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) \xrightarrow[T\to\infty]{} \mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\Big[(\mathbf{y}^*)^T(\mathbf{y}^*)\Big]$$

with $\widehat{\boldsymbol{\omega}}_t \sim q^*_\theta(\boldsymbol{\omega})$ *and* $\mathbf{y}^*, \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ *row vectors (thus the sum is over the outer-products).*

Proof.

$$\begin{aligned}
\mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\Big[(\mathbf{y}^*)^T(\mathbf{y}^*)\Big] &= \int \left(\int (\mathbf{y}^*)^T(\mathbf{y}^*) p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) \mathrm{d}\mathbf{y}^* \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\mathrm{Cov}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*] + \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*]^T \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*] \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\tau^{-1}\mathbf{I} + \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)^T \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*) \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}
\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}\Big[(\mathbf{y}^*)^T(\mathbf{y}^*)\Big] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ following MC integration with T samples. □

To obtain the model's predictive variance we use the unbiased estimator:

$$\widetilde{\mathrm{Var}}[\mathbf{y}^*] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) - \widetilde{\mathbb{E}}[\mathbf{y}^*]^T \widetilde{\mathbb{E}}[\mathbf{y}^*]$$

$$\xrightarrow[T\to\infty]{} \mathrm{Var}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]$$

which equals the sample variance of T stochastic forward passes through the NN plus the inverse model precision.

How can we find the model precision? In practice in the deep learning literature we often grid-search over the weight-decay λ to minimise validation error. Then, given a weight-decay λ_i (and prior length-scale l_i), eq. (3.13) can be re-written to find the model precision[8]:

$$\tau = \frac{(1-p)l_i^2}{2N\lambda_i}. \tag{3.17}$$

Remark (Predictive variance and posterior variance)**.** It is important to note the difference between the variance of the approximating distribution $q_\theta(\boldsymbol{\omega})$ and the variance of the predictive distribution $q_\theta(\mathbf{y}|\mathbf{x})$ (eq. (3.15)).
To see this, consider the illustrative example of an approximating distribution with fixed mean and variance used with the first weight layer $\mathbf{W}_1$, for example a standard Gaussian $\mathcal{N}(0, I)$. Further, assume for the sake of argument that delta distributions (or Gaussians with very small variances) are used to approximate the posterior over the layers following the first layer. Given enough follow-up layers we can capture any function to arbitrary precision—including the inverse cumulative distribution function (CDF) of any distribution (similarly to the remark in §2.2.1, but with the addition of a Gaussian CDF as we don't have a uniform on the first layer in this case). Passing the distribution from the first layer through the rest of the layers transforms the standard Gaussian with this inverse CDF, resulting in any arbitrary distribution as determined by the CDF.
In this example, even though the variance of each weight layer is constant, the variance of the predictive distribution can take any value depending on the learnt CDF. This example can be extended from a standard Gaussian approximating distribution to a mixture of Gaussians with fixed standard deviations, and to discrete distributions with fixed probability vectors (such as the dropout approximating distribution). Of course, in real world cases we would prefer to avoid modelling the

[8] Prior length-scale l_i can be fixed based on the density of the input data $\mathbf{X}$ and our prior belief as to the function's wiggliness, or optimised over as well (w.r.t. predictive log-likelihood over a validation set). The dropout probability is optimised using grid search similarly.

deep layers with delta approximating distributions since that would sacrifice our ability to capture model uncertainty.

Given a dataset $\mathbf{X}, \mathbf{Y}$ and a new data point $\mathbf{x}^*$ we can calculate the probability of possible output values $\mathbf{y}^*$ using the predictive probability $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$. The log of the predictive likelihood captures how well the model fits the data, with larger values indicating better model fit. Our predictive log-likelihood (also referred to as *test log-likelihood*) can be approximated by MC integration of eq. (3.15) with T terms:

$$\begin{aligned}\widetilde{\log} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) := \log\left(\frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}_t)\right) &\xrightarrow[T\to\infty]{} \log \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) q_\theta^*(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &\approx \log \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega} \\ &= \log p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\end{aligned}$$

with $\boldsymbol{\omega}_t \sim q_\theta^*(\boldsymbol{\omega})$ and since $q_\theta^*(\boldsymbol{\omega})$ is the minimiser of eq. (2.3). Note that this is a biased estimator since the expected quantity is transformed with the non-linear logarithm function, but the bias decreases as T increases.

For regression we can rewrite this last equation in a more numerically stable way[9]:

$$\widetilde{\log} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \mathrm{logsumexp}\left(-\frac{1}{2}\tau||\mathbf{y} - \mathbf{f}^{\widehat{\omega}_t}(\mathbf{x}^*)||^2\right) - \log T - \frac{1}{2}\log 2\pi + \frac{1}{2}\log\tau \tag{3.18}$$

with our precision parameter τ.

Uncertainty quality can be determined from this quantity as well. Excessive uncertainty (large observation noise, or equivalently small model precision τ) results in a large penalty from the last term in the predictive log-likelihood. On the other hand, an over-confident model with large model precision compared to poor mean estimation results in a penalty from the first term—the distance $||\mathbf{y} - \mathbf{f}^{\widehat{\omega}_t}(\mathbf{x}^*)||^2$ gets amplified by τ which drives the exponent to zero.

Note that the normal NN model itself is not changed. To estimate the predictive mean and predictive uncertainty we simply collect the results of stochastic forward passes through the model. As a result, this information can be used with existing NN models trained with SRTs. Furthermore, the forward passes can be done concurrently, resulting in constant running time identical to that of standard NNs.

[9] logsumexp is the log-sum-exp function.

3.3.1 Uncertainty in classification

In regression we summarised predictive uncertainty by looking at the sample variance of multiple stochastic forward passes. In the classification setting, however, we need to rely on alternative approaches to summarise uncertainty[10]. We will analyse three approaches to summarise uncertainty within classification: variation ratios [Freeman, 1965], predictive entropy [Shannon, 1948], and mutual information [Shannon, 1948]. These measures capture different notions of uncertainty: model uncertainty and predictive uncertainty, and will be explained below. But first, how do we calculate these quantities in our setting?

To use variation ratios we would sample a label from the softmax probabilities at the end of each *stochastic forward pass* for a test input $\mathbf{x}$. Collecting a set of T labels y_t from multiple stochastic forward passes on the same input we can find the mode of the distribution[11] $c^* = \arg\max_{c=1,\dots,C} \sum_t \mathbb{1}[y^t = c]$, and the number of times it was sampled $f_{\mathbf{x}} = \sum_t \mathbb{1}[y^t = c^*]$. We then set

$$\text{variation-ratio}[\mathbf{x}] := 1 - \frac{f_{\mathbf{x}}}{T}. \tag{3.19}$$

The variation ratio is a measure of dispersion—how "spread" the distribution is around the mode. In the binary case, the variation ratio attains its maximum of 0.5 when the two classes are sampled equally likely, and its minimum of 0 when only a single class is sampled.

Remark (Variation ratios and approximate inference)**.** The variation ratio as it was formulated in [Freeman, 1965] and used above can be seen as approximating the quantity

$$1 - p(y = c^* | \mathbf{x}, \mathcal{D}_{\text{train}})$$

with $c^* = \arg\max_{c=1,\dots,C} p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$. This is because for y^t the t'th class sampled for input $\mathbf{x}$ we have:

$$\frac{f_{\mathbf{x}}}{T} = \frac{1}{T} \sum_t \mathbb{1}[y^t = c^*] \xrightarrow[T \to \infty]{} \mathbb{E}_{q_\theta^*(y|\mathbf{x})} \Big[\mathbb{1}[y = c^*] \Big]$$

[10] These approaches are necessary since the probability vector resulting from a *deterministic* forward pass through the model does not capture confidence, as explained in figure 1.3.

[11] Here $\mathbb{1}[\cdot]$ is the indicator function.

$$
\begin{aligned}
&= q_\theta^*(y = c^*|\mathbf{x}) \\
&\approx p(y = c^*|\mathbf{x}, \mathcal{D}_{\text{train}})
\end{aligned}
$$

and,

$$
\begin{aligned}
c^* = \underset{c=1,\dots,C}{\arg\max} \sum_t \mathbb{1}[y^t = c] &= \underset{c=1,\dots,C}{\arg\max} \frac{1}{T} \sum_t \mathbb{1}[y^t = c] \\
&\xrightarrow[T\to\infty]{} \underset{c=1,\dots,C}{\arg\max} \mathbb{E}_{q_\theta^*(y|\mathbf{x})}[\mathbb{1}[y = c]] \\
&= \underset{c=1,\dots,C}{\arg\max} q_\theta^*(y = c|\mathbf{x}). \\
&\approx \underset{c=1,\dots,C}{\arg\max} p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})
\end{aligned}
$$

since $q_\theta^*(\boldsymbol{\omega})$ is a minimiser of the KL divergence to $p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})$ and therefore $q_\theta^*(y|\mathbf{x}) \approx \int p(y|\mathbf{f}^\omega(\mathbf{x}))p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}$ (following eq. (3.15)).

Unlike variation ratios, predictive entropy has its foundations in information theory. This quantity captures the average amount of information contained in the predictive distribution:

$$
\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] := -\sum_c p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \tag{3.20}
$$

summing over all possible classes c that y can take. Given a test point $\mathbf{x}$, the predictive entropy attains its maximum value when all classes are predicted to have equal uniform probability, and its minimum value of zero when one class has probability 1 and all others probability 0 (i.e. the prediction is certain).

In our setting, the predictive entropy can be approximated by collecting the probability vectors from T stochastic forward passes through the network, and for each class c averaging the probabilities of the class from each of the T probability vectors, replacing $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$ in eq. (3.20). In other words, we replace $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$ with $\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)$, where $p(y = c|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)$ is the probability of input $\mathbf{x}$ to take class c with model parameters $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$:

$$
[p(y = 1|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t), ..., p(y = C|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)] := \text{Softmax}(\mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x})).
$$

Then,

$$\begin{aligned}
\tilde{\mathbb{H}}[y|\mathbf{x},\mathcal{D}_{\text{train}}] :=& -\sum_c \left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\log\left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\\
\xrightarrow[T\to\infty]{} & -\sum_c \left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})q^*_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\right)\log\left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})q^*_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\right)\\
\approx & -\sum_c \left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}\right)\log\left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}\right)\\
=& -\sum_c p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\log p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\\
=& \mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}]
\end{aligned}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q^*_\theta(\boldsymbol{\omega})$ and since $q^*_\theta(\boldsymbol{\omega})$ is the optimum of eq. (3.7). Note that this is a biased estimator since the unbiased estimator $\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t) \xrightarrow[T\to\infty]{} \int p(y=c|\mathbf{x},\boldsymbol{\omega})q^*_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}$ is transformed through the non-linear function $\mathbb{H}[\cdot]$. The bias of this estimator will decrease as T increases.

As an alternative to the predictive entropy, the mutual information between the prediction y and the posterior over the model parameters $\boldsymbol{\omega}$ offers a different measure of uncertainty:

$$\begin{aligned}
\mathbb{I}[y,\boldsymbol{\omega}|\mathbf{x},\mathcal{D}_{\text{train}}] :=& \mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}] - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})}\Big[\mathbb{H}[y|\mathbf{x},\boldsymbol{\omega}]\Big]\\
=& -\sum_c p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\log p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\\
&+ \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})}\Big[\sum_c p(y=c|\mathbf{x},\boldsymbol{\omega})\log p(y=c|\mathbf{x},\boldsymbol{\omega})\Big]
\end{aligned}$$

with c the possible classes y can take. This tractable view of the mutual information was suggested in [Houlsby et al., 2011] in the context of active learning. Test points $\mathbf{x}$ that maximise the mutual information are points on which the model is uncertain on average, yet there exist model parameters that erroneously produce predictions with high confidence.

The mutual information can be approximated in our setting in a similar way to the predictive entropy approximation:

$$\begin{aligned}
\tilde{\mathbb{I}}[y,\boldsymbol{\omega}|\mathbf{x},\mathcal{D}_{\text{train}}] :=& -\sum_c \left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\log\left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\\
&+ \frac{1}{T}\sum_{c,t} p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\log p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\\
\xrightarrow[T\to\infty]{} & \mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}] - \mathbb{E}_{q^*_\theta(\boldsymbol{\omega})}\Big[\mathbb{H}[y|\mathbf{x},\boldsymbol{\omega}]\Big]
\end{aligned}$$

$$\approx \mathbb{I}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}]$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$.

Some intuition. To understand the different measures for uncertainty, we shall look at three concrete examples in binary classification of dogs and cats given an input image. More specifically, we will look at the sets of probability vectors obtained from multiple stochastic forward passes, and the uncertainty measures resulting from these sets. The three examples are where the probabilities for the class "dog" in all vectors are

1. all equal to 1 (i.e. the probability vectors collected are $\{(1,0), ..., (1,0)\}$),
2. all equal to 0.5 (i.e. the probability vectors collected are $\{(0.5,0.5), ..., (0.5,0.5)\}$), and
3. half of the probabilities sampled equal to 0 and half of the probabilities equal to 1 (i.e. the probability vectors collected are $\{(1,0), (0,1), (0,1), ..., (1,0)\}$ for example).

In example (1) the prediction has high confidence, whereas in examples (2) and (3) the prediction has low confidence. These are examples of *predictive* uncertainty. Compared to this notion of confidence, in examples (1) and (2) the *model* is confident about its output since it gives identical probabilities in multiple forward passes. On the other hand, in the last example (3) the model is uncertain about its output, corresponding to the case in figure 1.3 (where the layer before the softmax has high uncertainty). This is an example of *model uncertainty*.

In example (1), both variation ratios, predictive entropy, and the mutual information would return value 0, all measures indicating high *confidence*. In example (3) variation ratios, predictive entropy, and the mutual information would all return value 0.5, all measures indicating high *uncertainty*. All three measures of uncertainty agree on these two examples.

However, in example (2) variation ratios and predictive entropy would return value 0.5, whereas the mutual information would return value 0. In this case variation ratios and predictive entropy capture the *uncertainty in the prediction*, whereas the mutual information captures the *model's confidence* in its output. This information can be used for example in *active learning*, and will be demonstrated in section §5.2.

3.3.2 Difficulties with the approach

Our technique is simple: perform several stochastic forward passes through the model, and look at the sample mean and variance. But it has several shortcomings worth discussing.

First, even though the training time of our model is identical to that of existing models in the field, the test time is scaled by T—the number of averaged forward passes through the network. This may not be of real concern in some real world applications, as NNs are often implemented on distributed hardware. Distributed hardware allows us to obtain MC estimates in constant time almost trivially, by transferring an input to a GPU and setting a mini-batch composed of the same input multiple times. In dropout for example we sample different Bernoulli realisations for each output unit and each mini-batch input, which results in a matrix of probabilities. Each row in the matrix is the output of the dropout network on the same input generated with different random variable realisations (dropout masks). Averaging over the rows results in the MC dropout estimate.

Another concern is that the model's uncertainty is not calibrated. A calibrated model is one in which the predictive probabilities match the empirical frequency of the data. The lack of calibration can be seen through the derivation's relation to Gaussian processes [Gal and Ghahramani, 2015b]. Gaussian processes' uncertainty is known to not be calibrated—the Gaussian process's uncertainty depends on the covariance function chosen, which is shown in [Gal and Ghahramani, 2015b] to be equivalent to the non-linearities and prior over the weights. The choice of a GP's covariance function follows from our assumptions about the data. If we believe, for example, that the model's uncertainty should increase far from the data we might choose the squared exponential covariance function.

For many practical applications the lack of calibration means that model uncertainty can increase for large magnitude data points or be of different scale for different datasets. To calibrate model uncertainty in regression tasks we can scale the uncertainty linearly to remove data magnitude effects, and manipulate uncertainty percentiles to compare among different datasets. This can be done by finding the number of validation set points having larger uncertainty than that of a test point. For example, if a test point has predictive standard deviation 5, whereas almost all validation points have standard deviation ranging from 0.2 to 2, then the test point's uncertainty value will be in the top percentile of the validation set uncertainty measures, and the model will be considered as very uncertain about the test point compared to the validation data. However, another model might give the same test point predictive standard deviation of 5 with most of the validation data given predictive standard deviation ranging from 10 to 15. In this model the test point's uncertainty measure will be in the lowest percentile of validation set uncertainty measures, and the model will be considered as fairly confident about the test point with respect to the validation data.

One last concern is a known limitation of VI. Variational inference is known to underestimates predictive variance [Turner and Sahani, 2011], a property descendent from our objective (which penalises $q_\theta(\boldsymbol{\omega})$ for placing mass where $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ has no mass, but less so for not placing mass where it should). Several solutions exist for this (such as [Giordano et al., 2015]), with different limitations for their practicality. Uncertainty under-estimation does not seem to be of real concern in practice though, and the variational approximation seems to work well in practical applications as we will see in the next chapter.

3.4 Approximate inference in complex models

We finish the chapter by extending the approximate inference technique above to more complex models such as convolutional neural networks and recurrent neural networks. This will allow us to obtain model uncertainty for models defined over sequence based datasets or for image data. We describe the approximate inference using Bernoulli variational distributions for convenience of presentation, although any SRT could be used instead.

3.4.1 Bayesian convolutional neural networks

In existing convolutional neural networks (CNNs) literature dropout is mostly used after inner-product layers at the end of the model alone. This can be seen as applying a finite deterministic transformation to the data before feeding it into a Bayesian NN. As such, model uncertainty can still be obtained for such models, but an interesting question is whether we could use approximate Bayesian inference over the full CNN. Here we wish to integrate over the convolution layers (kernels) of the CNN as well. To implement a Bayesian CNN we could apply dropout after all convolution layers as well as inner-product layers, and evaluate the model's predictive posterior using eq. (3.8) at test time. Note though that generalisations to SRTs other than dropout are also possible and easy to implement.

Recall the structure of a CNN described in §1.1 and in figure 1.1. In more detail, the input to the i'th convolution layer is represented as a 3 dimensional tensor $\mathbf{x} \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times K_{i-1}}$ with height H_{i-1}, width W_{i-1}, and K_{i-1} channels. A convolution layer is then composed of a sequence of K_i kernels (weight tensors): $\mathbf{k}_k \in \mathbb{R}^{h \times w \times K_{i-1}}$ for $k = 1, ..., K_i$. Here we assume kernel height h, kernel width w, and the last dimension to match the number of channels in the input layer: K_{i-1}. Convolving

the kernels with the input (with a given stride s) then results in an output layer of dimensions $\mathbf{y} \in \mathbb{R}^{H'_{i-1} \times W'_{i-1} \times K_i}$ with H'_{i-1} and W'_{i-1} being the new height and width, and K_i channels—the number of kernels. Each element $y_{i,j,k}$ is the sum of the element-wise product of kernel $\mathbf{k}_k$ with a corresponding patch in the input image $\mathbf{x}$: $[[x_{i-h/2,j-w/2,1}, ..., x_{i+h/2,j+w/2,1}], ..., [x_{i-h/2,j-w/2,K_{i-1}}, ..., x_{i+h/2,j+w/2,K_{i-1}}]]$.

To integrate over the kernels, we reformulate the convolution as a linear operation. Let $\mathbf{k}_k \in \mathbb{R}^{h \times w \times K_{i-1}}$ for $k = 1, ..., K_i$ be the CNN's kernels with height h, width w, and K_{i-1} channels in the i'th layer. The input to the layer is represented as a 3 dimensional tensor $\mathbf{x} \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times K_{i-1}}$ with height H_{i-1}, width W_{i-1}, and K_{i-1} channels. Convolving the kernels with the input with a given stride s is equivalent to extracting patches from the input and performing a matrix product: we extract $h \times w \times K_{i-1}$ dimensional patches from the input with stride s and vectorise these. Collecting the vectors in the rows of a matrix we obtain a new representation for our input $\overline{\mathbf{x}} \in \mathbb{R}^{n \times hwK_{i-1}}$ with n patches. The vectorised kernels form the columns of the weight matrix $\mathbf{W}_i \in \mathbb{R}^{hwK_{i-1} \times K_i}$. The convolution operation is then equivalent to the matrix product $\overline{\mathbf{x}}\mathbf{W}_i \in \mathbb{R}^{n \times K_i}$. The columns of the output can be re-arranged into a 3 dimensional tensor $\mathbf{y} \in \mathbb{R}^{H_i \times W_i \times K_i}$ (since $n = H_i \times W_i$). Pooling can then be seen as a non-linear operation on the matrix $\mathbf{y}$. Note that the pooling operation is a non-linearity applied after the linear convolution counterpart to ReLU or Tanh non-linearities.

To make the CNN into a probabilistic model we place a prior distribution over each kernel and approximately integrate each kernels-patch pair with Bernoulli variational distributions. We sample Bernoulli random variables $\boldsymbol{\epsilon}_{i,j,n}$ and multiply patch n by the weight matrix $\mathbf{W}_i \cdot \text{diag}([\boldsymbol{\epsilon}_{i,j,n}]_{j=1}^{K_i})$. This product is equivalent to an approximating distribution modelling each kernel-patch pair with a distinct random variable, tying the means of the random variables over the patches. The distribution randomly sets kernels to zero for different patches. This approximating distribution is also equivalent to applying dropout for each element in the tensor $\mathbf{y}$ before pooling. Implementing our Bayesian CNN is therefore as simple as using dropout after every convolution layer before pooling.

The standard dropout test time approximation (scaling hidden units by $1/(1 - p_i)$) does not perform well when dropout is applied after convolutions—this is a negative result we identified empirically. We solve this by approximating the predictive distribution following eq. (3.8), averaging stochastic forward passes through the model at test time (using MC dropout). We assess the model above with an extensive set of experiments studying its properties in §4.4.

3.4.2 Bayesian recurrent neural networks

We next develop inference with Bernoulli variational distributions for recurrent neural networks (RNNs), although generalisations to SRTs other than dropout are trivial. We will concentrate on simple RNN models for brevity of notation. Derivations for LSTM and GRU follow similarly. Given input sequence $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ of length T, a simple RNN is formed by a repeated application of a deterministic function $\mathbf{f}_\mathbf{h}$. This generates a hidden state $\mathbf{h}_t$ for time step t:

$$\mathbf{h}_t = \mathbf{f}_\mathbf{h}(\mathbf{x}_t, \mathbf{h}_{t-1}) = \sigma(\mathbf{x}_t \mathbf{W}_\mathbf{h} + \mathbf{h}_{t-1} \mathbf{U}_\mathbf{h} + \mathbf{b}_\mathbf{h})$$

for some non-linearity σ. The model output can be defined, for example, as

$$\mathbf{f}_\mathbf{y}(\mathbf{h}_T) = \mathbf{h}_T \mathbf{W}_\mathbf{y} + \mathbf{b}_\mathbf{y}.$$

To view this RNN as a probabilistic model we regard $\boldsymbol{\omega} = \{\mathbf{W}_\mathbf{h}, \mathbf{U}_\mathbf{h}, \mathbf{b}_\mathbf{h}, \mathbf{W}_\mathbf{y}, \mathbf{b}_\mathbf{y}\}$ as random variables (following Gaussian prior distributions). To make the dependence on $\boldsymbol{\omega}$ clear, we write $\mathbf{f}_\mathbf{y}^\boldsymbol{\omega}$ for $\mathbf{f}_\mathbf{y}$ and similarly for $\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}$. We define our probabilistic model's likelihood as above (section 2.1). The posterior over random variables $\boldsymbol{\omega}$ is rather complex, and we approximate it with a variational distribution $q(\boldsymbol{\omega})$. For the dropout SRT for example we may use a Bernoulli approximating distribution.

Recall our VI optimisation objective eq. (3.1). Evaluating each log likelihood term in eq. (3.1) with our RNN model we have

$$\begin{aligned}\int q(\boldsymbol{\omega}) \log p(\mathbf{y}|\mathbf{f}_\mathbf{y}^\boldsymbol{\omega}(\mathbf{h}_T)) \mathrm{d}\boldsymbol{\omega} &= \int q(\boldsymbol{\omega}) \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^\boldsymbol{\omega}\left(\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_T, \mathbf{h}_{T-1})\right)\right) \mathrm{d}\boldsymbol{\omega} \\ &= \int q(\boldsymbol{\omega}) \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^\boldsymbol{\omega}\left(\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_T, \mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(...\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_1, \mathbf{h}_0)...))\right)\right) \mathrm{d}\boldsymbol{\omega}\end{aligned}$$

with $\mathbf{h}_0 = \mathbf{0}$. We approximate this with MC integration with a single sample:

$$\approx \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^{\widehat{\boldsymbol{\omega}}}\left(\mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_T, \mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(...\mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_1, \mathbf{h}_0)...))\right)\right),$$

with $\widehat{\boldsymbol{\omega}} \sim q(\boldsymbol{\omega})$, resulting in an unbiased estimator to each sum term[12].

[12] Note that for brevity we did not re-parametrise the integral, although this should be done to obtain low variance derivative estimators.

This estimator is plugged into equation (3.1) to obtain our minimisation objective

$$\widehat{\mathcal{L}}_{\text{MC}} = -\sum_{i=1}^{N} \log p\left(\mathbf{y}_i \middle| \mathbf{f}_{\mathbf{y}}^{\widehat{\boldsymbol{\omega}}_i}\left(\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(\mathbf{x}_{i,T}, \mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(...\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(\mathbf{x}_{i,1}, \mathbf{h}_0)...))\right)\right) + \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})). \quad (3.21)$$

Note that for each sequence $\mathbf{x}_i$ we sample a new realisation $\widehat{\boldsymbol{\omega}}_i = \{\widehat{\mathbf{W}}_{\mathbf{h}}^i, \widehat{\mathbf{U}}_{\mathbf{h}}^i, \widehat{\mathbf{b}}_{\mathbf{h}}^i, \widehat{\mathbf{W}}_{\mathbf{y}}^i, \widehat{\mathbf{b}}_{\mathbf{y}}^i\}$, and that each symbol in the sequence $\mathbf{x}_i = [\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,T}]$ is passed through the function $\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}$ with *the same weight realisations* $\widehat{\mathbf{W}}_{\mathbf{h}}^i, \widehat{\mathbf{U}}_{\mathbf{h}}^i, \widehat{\mathbf{b}}_{\mathbf{h}}^i$ *used at every time step* $t \leq T$.

In the dropout case, evaluating the model output $\mathbf{f}_{\mathbf{y}}^{\widehat{\boldsymbol{\omega}}}(\cdot)$ with sample $\widehat{\boldsymbol{\omega}}$ corresponds to randomly zeroing (masking) rows in each weight matrix $\mathbf{W}$ during the forward pass. In our RNN setting with a sequence input, each weight matrix row is randomly masked once, and importantly the same mask is used through all time steps. When viewed as a stochastic regularisation technique, our induced dropout variant is therefore identical to implementing dropout in RNNs with the *same network units dropped at each time step*, randomly dropping inputs, outputs, and recurrent connections. Predictions can be approximated by either propagating the mean of each layer to the next (referred to as the *standard dropout approximation*), or by performing dropout at test time and averaging results (MC dropout, eq. (3.16)).

Remark (Bayesian versus ensembling interpretation of dropout)**.** Apart from our Bayesian approximation interpretation, dropout in deep networks can also be seen as following an ensembling interpretation [Srivastava et al., 2014]. This interpretation also leads to MC dropout at test time. But the ensembling interpretation does not determine whether the ensemble should be over the network units or the weights. For example, in an RNN this view will *not* lead to our dropout variant, unless the ensemble is *defined to tie the weights of the network* ad hoc. This is in comparison to the Bayesian approximation view where the weight tying is forced by the probabilistic interpretation of the model.
The same can be said about the latent variable model view of dropout [Maeda, 2014] where a constraint over the weights would have to be added ad hoc to derive the results presented here.

Certain RNN models such as LSTMs [Graves et al., 2013; Hochreiter and Schmidhuber, 1997] and GRUs [Cho et al., 2014] use different *gates* within the RNN units. For example, an LSTM is defined by setting four gates: "input", "forget", "output", and an "input modulation gate",

$$\underline{\mathbf{i}} = \text{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_i + \mathbf{x}_t\mathbf{W}_i\right) \qquad \underline{\mathbf{f}} = \text{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_f + \mathbf{x}_t\mathbf{W}_f\right)$$

$$\underline{\mathbf{o}} = \text{sigm}\Big(\mathbf{h}_{t-1}\mathbf{U}_o + \mathbf{x}_t\mathbf{W}_o\Big) \qquad \underline{\mathbf{g}} = \tanh\Big(\mathbf{h}_{t-1}\mathbf{U}_g + \mathbf{x}_t\mathbf{W}_g\Big)$$
$$\mathbf{c}_t = \underline{\mathbf{f}} \odot \mathbf{c}_{t-1} + \underline{\mathbf{i}} \odot \underline{\mathbf{g}} \qquad \mathbf{h}_t = \underline{\mathbf{o}} \odot \tanh(\mathbf{c}_t) \tag{3.22}$$

with $\boldsymbol{\omega} = \{\mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_f, \mathbf{U}_f, \mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_g, \mathbf{U}_g\}$ weight matrices and sigm the sigmoid non-linearity. Here an internal state $\mathbf{c}_t$ (also referred to as *cell*) is updated additively.

Alternatively, the model could be re-parametrised as in [Graves et al., 2013]:

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \left(\mathbf{W} \cdot \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} \right) \tag{3.23}$$

with $\boldsymbol{\omega} = \{\mathbf{W}\}$, $\mathbf{W} = [\mathbf{W}_i, \mathbf{U}_i; \mathbf{W}_f, \mathbf{U}_f; \mathbf{W}_o, \mathbf{U}_o; \mathbf{W}_g, \mathbf{U}_g]$ a matrix of dimensions $4K$ by $2K$ (K being the dimensionality of $\mathbf{x}_t$). We name this parametrisation a *tied-weights* LSTM (compared to the *untied-weights* LSTM parametrisation in eq. (3.22)).

Even though these two parametrisations result in the same deterministic model output, they lead to different approximating distributions $q(\boldsymbol{\omega})$. With the first parametrisation one could use different dropout masks for different gates (even when the same input $\mathbf{x}_t$ is used). This is because the approximating distribution is placed over the matrices rather than the inputs: we might drop certain rows in one weight matrix $\mathbf{W}$ applied to $\mathbf{x}_t$ and different rows in another matrix $\mathbf{W}'$ applied to $\mathbf{x}_t$. With the second parametrisations we would place a distribution over the single matrix $\mathbf{W}$. This leads to a faster forward-pass, but with slightly diminished results (this tradeoff is examined in section §4.5).

Remark (Word embeddings dropout)**.** In datasets with continuous inputs we often apply SRTs such as dropout to the input layer—i.e. to the input vector itself. This is equivalent to placing a distribution over the weight matrix which follows the input and approximately integrating over it (the matrix is optimised, therefore prone to overfitting otherwise).
But for models with discrete inputs such as words (where every word is mapped to a continuous vector—a *word embedding*)—this is seldom done. With word embeddings the input can be seen as either the word embedding itself, or, more conveniently, as a "one-hot" encoding (a vector of zeros with 1 at a single position). The product of the one-hot encoded vector with an embedding matrix $\mathbf{W}_E \in \mathbb{R}^{V \times D}$ (where D is the embedding dimensionality and V is the number of words in the vocabulary) then gives a word embedding. Curiously, this parameter layer is the largest layer in

most language applications, yet it is often not regularised. Since the embedding matrix is optimised it can lead to overfitting, and it is therefore desirable to apply dropout to the one-hot encoded vectors. This in effect is identical to *dropping words at random* throughout the input sentence, and can also be interpreted as encouraging the model to not "depend" on single words for its output.
Note that as before, we randomly set rows of the matrix $\mathbf{W}_E \in \mathbb{R}^{V \times D}$ to zero. Since we repeat the same mask at each time step, we drop the same words throughout the sequence—i.e. we drop word types at random rather than word tokens (as an example, the sentence "the dog and the cat" might become "— dog and — cat" or "the — and the cat", but never "— dog and the cat"). A possible inefficiency implementing this is the requirement to sample V Bernoulli random variables, where V might be large. This can be solved by the observation that for sequences of length T, at most T embeddings could be dropped (other dropped embeddings have no effect on the model output). For $T \ll V$ it is therefore more efficient to first map the words to the word embeddings, and only then to zero-out word embeddings based on their word type.

In the next chapter we will study the techniques above empirically and analyse them quantitatively. This is followed by a survey of recent literature making use of the techniques in real-world problems concerning AI safety, image processing, sequence processing, active learning, and other examples.