Chapter 4
Bayesian Neural Networks

This chapter presents the ideas, derivations, advantages, and issues of four different algorithms for Bayesian Neural Network (BNN):

- Bayes by Backprop (BBB) [6];
- Probabilistic Backpropagation (PBP) [19];
- Monte Carlo Dropout (MCDO) [13];
- Variational Adam (Vadam) [26].

Each method approaches the problem in a considerably different manner. Still, they all share one trait in common: they all consider unstructured approximations to the posterior distribution.

By the end of this chapter, the reader should:

- Know the attributes a BNN should possess;
- Learn metrics to assess such characteristics;
- Discern the benefits and issues of each method;
- Understand the differences among them;
- Be capable of choosing the one that best suits its needs;
- Know where to search deeper if in need of structured BNNs.

4.1 Why BNNs?

Recently, BNNs have been object of renewed interest within the research community. As one may imagine by now, BNNs are essentially standard deterministic NNs enhanced with Bayesian methods. Instead of learning the optimal weights $\mathbf{w}^*$, they infer the posterior weight distribution $p(\mathbf{w} \mid \mathbf{D})$ given the data set $\mathcal{D}$. Thus, $\mathbf{w}^*$, the maximizer of the distribution, is only a single point in the entire support, as illustrated in Fig. 4.1.

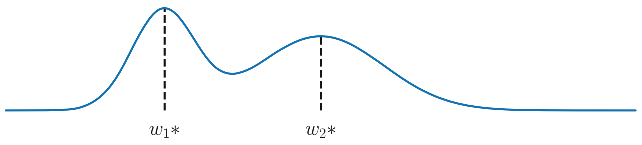

Fig. 4.1 Maximizer w_1^* of the posterior distribution and runner-up w_2^* corresponding to the maximum of another mode

First introduced in [55], BNNs saw a great advance during the following years (the 1990s) [20, 35, 39, 40]. However, due to their computational complexity, they ended up relegated for a decade. Standard NNs had not had success for a long time, only picking up momentum in 2006 [21] and effectively gaining attention in 2012 after a Deep Learning (DL) method [30] won an important image classification competition [48] by a large margin. It certainly would not go differently for BNNs, which faced an even more difficult scenario. Over the last few years, new practical approaches to BNNs [16, 57] allied to the concerns raised by adversarial attacks [43] and the cry for uncertainty measures quintessential for some practical applications sparked interest in Bayesian methods for DL.

The main reason for the late acceptance of BNNs (which is still to come) is that their computational complexity impedes scalability. Modern models and data sets have millions of parameters and instances, so nothing but very simplistic algorithms can handle well such large-scale regime. A clear example is the use of backpropagation and first-order optimization methods, though that does not mean they are not ingenious. Consequently, latest works in this field focus on scalable and (most of the time) practical approaches that can meet the current demand and still are comprehensible, or at least usable, by practitioners.

For those still not convinced about the benefits of being Bayesian, we quickly review the state of affairs for modern DL.

Even though backpropagation and maximum likelihood optimization allow fitting large non-linear models on massive amounts of data and find success on several tasks, they are sensitive to overfitting, specially if we try such models on not so large data sets. Employing common regularization techniques, such as ℓ_1 or ℓ_2 penalty, is equivalent to maximum a posteriori optimization (with Laplace and Gaussian priors, respectively). However, in spite of alleviating overfitting, it is far from solving the problem. What is more, it makes the solution dependent on the parameterization, that is, different parameterizations may lead to different optimal points. Then, the question arises of which parameterization leads to the best possible solution and how sensitive it is.

Even when resorting to invariant methods, we still have no measure of confidence. Although bootstrapping alleviates the issue, it does not solve the underlying problem: it approximates the probability distribution of the observed data, considering the unknown variables to be fixed. The Bayesian framework solves all this at once by allowing models to represent not only single point estimates but complete distributions over all possible parameter values. It offers a unified framework for model building, inference, prediction, and decision-making. Moreover, it provides a straightforward way to score models and select among them. BNNs have built-in regularization, offer the advantages of ensemble learning, allow uncertainty estimation and continual learning, besides weight quantization and compression.

There is no free lunch, and as already hinted above, BNNs have burdensome inference. They rely on conditioning and marginalization, so the main operation is integration. Thus, high-dimensional and/or complex models impose a real barrier to their deployment. We discuss approaches that mitigate this issue by employing distributional approximations (Sect. 3.2) to render computations amenable. Particularly, we focus on those that do not explicitly impose structure on weights, and instead assume them independent (mean-field approximation, Sect. 3.2.1).

For ease of notation, we shall use $\mathbf{w}$ as the random variable instead of $\mathbf{z}$. This change of notation is not only to keep similarity to the literature in BNNs, but also to remind our readers that the distributions are over the model's weights (the parameters) and not hidden units.

4.2 Assessing Uncertainty Quality

Bayesian and, more generally, probabilistic models output some measure of uncertainty on which we rely to make decisions. Can we really believe in these models? Do they reflect, approximately at least, the truth? We next present common approaches to address these questions.

4.2.1 *Predictive Log-Likelihood*

As explained in Sect. 2.4, the likelihood term $p(\mathbf{d} \mid \mathbf{W})$ measures how likely a specific configuration of the model is of generating the observed data. The predictive log-likelihood captures how well the model fits the data, taking the variance (or other measure of spread) of the prediction into account. It is an estimate of how well the model fits both the mean and uncertainty.

Intuitively, the lower the variance, the more reliable the prediction should be and, hence, the lower the score for being wrong. Still, the predictions ought to be reliable so large variances also receive lower scores.

Let us take as example a regression model $f(\cdot; w)$, parameterized by w, that predicts a scalar value $\hat{y}$, such that $\hat{y} = f(x)$. Our probabilistic model assumes a given level of noise and we thus place an observation noise model on top of the output, such that the true output is corrupted by a known process. For an additive Gaussian noise with variance σ^2, the log-likelihood estimate has the form

$$\begin{aligned}\log p(y \mid x, w) &= \log \mathcal{N}(y; f(x; w), \sigma^2)\\ &= -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(y - f(x)\right)^2 .\end{aligned} \tag{4.1}$$

What we wish is that the observation y is as close as possible to the predicted output $f(x)$, such that our model agrees with the data. Note that the prediction and the noise model could in principle be anything.

4.2.2 Calibration

Although posterior or predictive credible intervals are not necessarily calibrated, it is a natural measure of reliability. In a classification task, one would expect being correct $X\%$ of the time when the model assigns an $X\%$ probability of being correct. In a regression setting, one hopes that $X\%$ of the time the true value falls within an $X\%$ credible interval. A model with such coverage property is said to be well-calibrated and implies that the Bayesian credible intervals coincide with the frequentist confidence intervals.

The approach that asserts that inferences under a particular model should be Bayesian, but model assessment can and should involve frequentist ideas is called Calibrated Bayes [33].

A common diagnostic tool for calibration is the reliability (or calibration) plot, shown in Fig. 4.2. Ideally, the empirical and the predictive cumulative distribution functions should match, so plotting one against the other should give a graph as close as possible to the identity $y = x$. Namely, for each credible interval corresponding to a probability threshold p_i, we plot the observed number of times (empirical frequency) the prediction falls within the interval. We can measure the calibration error numerically by computing the expected error between the predicted and empirical frequencies for m different confidence intervals.

Still analyzing Fig. 4.2, one can notice that there are two other curves besides the identity. The one in red, with triangle markers, refers to the uncalibrated model, as the blue one, with square markers, is provided by the calibrated method applied after the model has been trained. This is a toy example so the reader can realize how significant the calibration process can be, clearly moving the uncalibrated curve towards the identity. However, the performance of this method varies according to the model you are calibrating, as pointed by the authors in [42]. In that paper, the authors also present and analyze the behavior of two well-known learning techniques that perform calibration: Platt Scaling [45] and Isotonic Regression.

Calibration is not enough for a good overall model, forecasts also need to be sharp [31]. Intuitively, credible intervals should be as tight and probabilities as binary as possible in regression and classification, respectively. A model that always predicts the mean value and adjusts its confidence accordingly is calibrated by definition, but not useful. There are various ways to measure spread, variance being one of them.

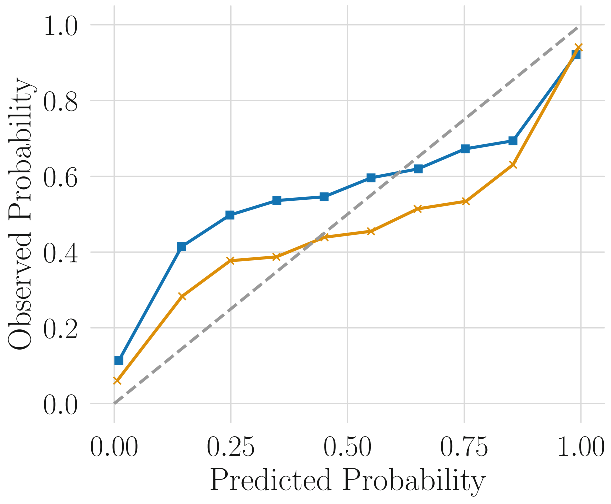

Fig. 4.2 Example of calibration plot. The gray dashed line is the identity $y = x$, the red curve is the uncalibrated model, and the blue curve refers to the post-calibration model, which were calibrated using the Platt's method [45]. Ideally, we want the blue and gray line to be superposed, indicating a perfectly well-calibrated model

4.2.3 Downstream Applications

It is worthwhile noting that even though the two previous metrics, predictive log-likelihood and expected calibration error, are standard measures for assessing uncertainty quality, it is still important to consider the context in which the uncertainty measures are applied. One should also evaluate uncertainty quality by measuring the performance of the downstream application of interest, e.g., outlier detection, active learning, or uncertainty-driven exploration, with the appropriate relevant metrics.

4.3 Bayes by Backprop

BBB has a quite long history preceding it. Bayes by Backprop [6], or BBB for short, continues the work of [16] on practical VI for NNs, who in turn extends on [20], the first to propose VI for NNs.

The essence of BBB's approach is choosing a variational posterior q from which probable samples can be drawn efficiently so that it becomes amenable to Monte Carlo (MC) integration.

Specifying a diagonal Gaussian posterior implies that all network weights w_i are independent, requiring separate means μ_i and variances σ_i^2. Consequently, each weight w_i is characterized by $\Psi_i = \{\mu_i, \sigma_i^2\}$ and the set of all parameters by $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$. We express the approximating variational posterior distribution by

$$q(\mathbf{w}; \Psi) = \prod_i q(w_i; \Psi_i) = \prod_i \mathcal{N}(w_i; \mu_i, \sigma_i^2). \tag{4.2}$$

As seen in Sect. 3.2.1, optimizing the variational approximation amounts to minimizing the negative ELBO as in (3.8), which writes

$$\begin{aligned}\mathcal{L}(q) &= -\text{ELBO}(q) \\ &= -\mathbb{E}_{q(\mathbf{w};\Psi)}\left[\log p(\mathcal{D} \mid \mathbf{W})\right] + D_{KL}\left(q(\mathbf{w}; \Psi) \| p(\mathbf{w})\right) \\ &= \mathcal{L}_{data} + \mathcal{L}_{prior},\end{aligned} \tag{4.3}$$

where we make explicit the presence of two cost functions of different nature. The first, $\mathcal{L}_{data}$, which we refer to as the likelihood cost, is data-dependent and quantifies the amount of error the model commits. The second, $\mathcal{L}_{prior}$, is prior-dependent and we call it the complexity cost. While the former drives the model towards best explaining the data, the latter acts as a regularizer pushing towards the prior $p(\mathbf{w})$, as already explained in Sect. 3.2.1.

The diagonal Gaussian posterior (4.2) results in a non-closed analytical form for the expectation in $\mathcal{L}_{data}$ and for its derivatives w.r.t. μ_i and σ_i, rendering direct evaluation and backpropagation unfeasible. To get around this issue one may resort to MC integration, i.e., drawing different weights $\mathbf{w}_t$ from the posterior $q(\mathbf{w}; \Psi)$, performing the desired computation for each sample and averaging the results. The main contribution from [6] is a reparameterization that gives unbiased gradient estimators and is actually not restricted to Gaussian distributions. It relies on the reparameterization trick (Sect. A.1) for a variational posterior $q(\mathbf{w}; \Psi)$ and a cost function $h(\mathbf{w}; \Psi)$, both dependent on the parameters Ψ, according to

$$\nabla_\Psi \mathbb{E}_{q(\mathbf{w};\Psi)}\left[h(\mathbf{w}; \Psi)\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\frac{\partial h(\mathbf{w}; \Psi)}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \Psi} + \frac{\partial h(\mathbf{w}; \Psi)}{\partial \Psi}\right], \tag{4.4}$$

where, as before, $\mathbf{w} = g(\boldsymbol{\epsilon}; \Psi)$, with $g(\cdot; \Psi)$ a smooth invertible deterministic transformation, and $\boldsymbol{\epsilon}$ is a base random variable.

Indeed, the above representation works for any distribution $q(\mathbf{w}; \Psi)$ that can be recast as a transformation $g(\boldsymbol{\epsilon}; \cdot)$ of base distribution $p(\boldsymbol{\epsilon})$. Still, the present case only deals with $q(\mathbf{w}; \Psi)$ as the product of independent univariate Gaussians (4.2) with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$. A convenient choice of transformation is $g(\boldsymbol{\epsilon}; \Psi) = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which boils down to $\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ for the uncorrelated Gaussian case, i.e., diagonal covariance matrix $\boldsymbol{\Sigma}$.

On a practical numerical note, one needs to prevent the σ_i from assuming negative values during the optimization since $\sigma_i \geqslant 0$. Instead of imposing explicit

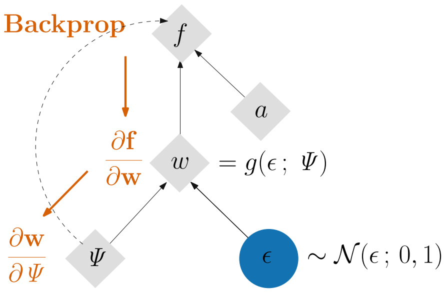
Backprop
f
a
$\frac{\partial \mathbf{f}}{\partial \mathbf{w}}$
w
$= g(\epsilon;\ \Psi)$
$\frac{\partial \mathbf{w}}{\partial \Psi}$
Ψ
ϵ
$\sim \mathcal{N}(\epsilon;\ 0, 1)$

Fig. 4.3 Computational graph after the reparameterization trick. The blue round node is a random node, while the gray rhombus nodes are deterministic. Black arrows represent the forward pass of the model and the red ones (part of) the backpropagation path. The black dashed line indicates the path for the computation of the KL divergence, that takes the distribution parameters Ψ as input. Note that thanks to the reparameterization trick the node w is no longer random and so we can compute its gradient as usual

constraints, the authors [6] suggest the *softplus* transform $\sigma_i = \log(1 + \exp \rho_i)$ that maps σ_i to ρ_i, whose value is confined to the range $(0, \infty)$.

Computing the derivatives of (4.4) w.r.t. both elements of $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and using the chosen transformation give

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = \frac{\partial h(\mathbf{w}, \Psi)}{\partial w_i} + \frac{\partial h(\mathbf{w}, \Psi)}{\partial \mu_i}, \tag{4.5}$$

$$\frac{\partial \mathcal{L}}{\partial \rho_i} = \frac{\partial h(\mathbf{w}, \Psi)}{\partial w_i} \frac{\epsilon}{1 + \exp(-\rho_i)} + \frac{\partial h(\mathbf{w}, \Psi)}{\partial \rho_i}. \tag{4.6}$$

The modification places the random component out of the gradient path followed by backpropagation, as illustrated in Fig. 4.3, where we depict a computational graph that computes a function f with weights $\mathbf{w}$ from input activations $\mathbf{a}$. This modification allows the direct computation of the gradients w.r.t. $\mathbf{w}$ and Ψ nodes in the computational graph just as done in any other deterministic node. Automatic differentiation tools available in common frameworks [1, 44, 53] handle this transparently, the only implementation difference being the need to explicitly reparameterize the weights $w = g(\boldsymbol{\epsilon}; \Psi)$ in the network definition and specify $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\rho}\}$ as the learnable parameters. More modern versions of the frameworks include built-in functions that automatically perform this reparameterization implicitly.

Figure 4.4 shows the final graphical model for BBB with independent Gaussian priors with parameters $\{\mu_p, \sigma_p^2\}$, an example for which the KL term in (4.3) can be evaluated analytically through the closed-form solution

$$\mathcal{L}_{prior} = \sum_i^W \log \frac{\sigma_p}{\sigma_i} + \frac{1}{2\sigma_p^2} \left[(\mu_i - \mu_p)^2 + \sigma_i^2 - \sigma_p^2 \right], \tag{4.7}$$

whose derivatives w.r.t. σ_i and μ_i are trivial to calculate.

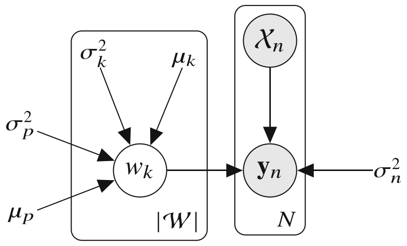

Fig. 4.4 PGM representation of the model underlying the BBB method. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter σ_n^2. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors

For non-conjugate priors, such as a mixture of Gaussians, we can instead compute the KL numerically with the samples drawn from the posterior. This estimation has the immediate advantage of allowing many more combinations of prior and variational posterior families. Even though we now have one more approximation in the system, more expressive priors can be used, i.e., non-Gaussian, what potentially leads to better results. In light of this change, instead of plugging (4.7) into (4.3), we write

$$\mathcal{L} \approx \sum_{i=1}^{T} - \log p(\mathbf{d} \mid \mathbf{W}^{(i)}) + \log q(\mathbf{w}^{(i)}; \Psi) - \log p(\mathbf{w}^{(i)}), \tag{4.8}$$

where $\mathbf{w}^{(i)}$ denotes the i-th out of T Monte Carlo samples drawn from the variational posterior $q(\mathbf{w}; \Psi)$.

When using mini-batch optimization such that $\mathcal{D} = \{\mathbf{d}_j \mid 1 \leq j \leq M\}$, it is important to scale the complexity cost $\mathcal{L}_{prior}$ in the objective accordingly. Equation (4.7) accounts for the whole data set, so naively computing the loss $\mathcal{L}_j$ in (4.3) M times will lead to accounting M times for the complexity loss $\mathcal{L}_{prior}$ instead of one. The $\mathcal{L}_{j_{prior}}$ terms should then be weighted so that $\mathcal{L}_{prior} = B_j \mathcal{L}_{j_{prior}}$. Although uniformly distributed weights $B_j = 1/M$ seem a natural choice, there are different ways of distributing them as long as $\sum_{j=1}^{M} B_j = 1$. In [6], the authors propose $B_j = 2^{M-j}/(2^M - 1)$. During the first iterations, the complexity cost dominates, and at later mini-batches, after more data is seen, the data likelihood cost $\mathcal{L}_{j_{data}}$ progressively gains more importance.

We summarize the resulting algorithm for optimizing a BNN in Algorithm 1. The case we illustrate is for a diagonal Gaussian variational posterior with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\rho}\}$, trained with a mini-batch of size 1 with non-uniformly distributed weighting of the complexity term $\mathcal{L}_{prior}$ across the mini-batches.

Even though the gradient estimators are unbiased, the MC predictive log-likelihood estimator is biased, because a non-linear function, i.e., the log, warps

Algorithm 1: Bayes by Backprop

```
1: while not converged do
2:     $\mathbf{w} \leftarrow \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3:     Randomly sample a data example $\mathbf{x}_i$
4:     $i \leftarrow (i + 1) \bmod N$
5:     $\pi_i \leftarrow 2^{N-i}/2^{N-1}$
6:     for $\mathbf{s} \in \{\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\rho}\}$ do
7:         $\mathbf{g}_\mathbf{s} \leftarrow -\nabla_\mathbf{s} \log p(\mathbf{x}_i|\mathbf{W}) + \pi_i \, (\nabla_\mathbf{s} \log q(\mathbf{w}; \Psi) - \nabla_\mathbf{s} \log p(\mathbf{w}))$
8:     end for
9:     $\Delta\boldsymbol{\mu} \leftarrow \mathbf{g}_\mathbf{w} + \mathbf{g}_{\boldsymbol{\mu}}$
10:    $\Delta\boldsymbol{\rho} \leftarrow \mathbf{g}_\mathbf{w} \odot \boldsymbol{\epsilon}/(1 + \exp(-\boldsymbol{\rho})) + \mathbf{g}_{\boldsymbol{\rho}}$
11:    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
12:    $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} - k\Delta\boldsymbol{\rho}$
13: end while
```

the expected value. This will in general be true for all MC estimators and can be mitigated by increasing the number of samples.

4.3.1 Practical VI

The BBB algorithm [6] actually builds upon the work of Graves [16], which gets around the non-closed analytical form of the derivatives of $\mathcal{L}_{data}$ in a different manner. Instead of using the reparameterization trick to compute the derivatives, Practical VI uses the fact that the expectations are over the Gaussian distribution and employs the identities [7, 46]

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \mu_i} = \mathbb{E}_q\left[\frac{\partial f(\mathbf{w})}{\partial w_i}\right], \tag{4.9}$$

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \sigma_i^2} = \frac{1}{2}\mathbb{E}_q\left[\frac{\partial^2 f(\mathbf{w})}{\partial w_i^2}\right]. \tag{4.10}$$

Here, the generic function $f = -\log p(\mathbf{d} \,|\, \mathbf{W})$ and its expected value $\mathbb{E}_q\left[f(\mathbf{w})\right] = \mathcal{L}_{data}$, the likelihood cost term of (4.3). These identities are useful because they enable unbiased gradient estimates and have low variance when doing MC integration. Nevertheless, (4.10) requires second-order derivatives and even though the mean-field assumption saves us from computing the full Hessian matrix $\nabla^2_\mathbf{w}\mathcal{L}_{data}$, its diagonal is still necessary.

Using the Generalized Gauss-Newton (GGN) approximation [8] to the Hessian in (4.10) (see Appendix A.3), we obtain

$$\frac{\partial \mathbb{E}_q\left[f(\mathbf{w})\right]}{\partial \sigma_i^2} = \frac{1}{2}\mathbb{E}_q\left[\frac{\partial^2 f(\mathbf{w})}{\partial w_i^2}\right] \approx \frac{1}{2}\mathbb{E}_q\left[\left(\frac{\partial f(\mathbf{w})}{\partial w_i}\right)^2\right]. \tag{4.11}$$

This approximation spares us from second-order derivatives, but introduces bias into the estimation of the gradient w.r.t. the variance, that is, its expected value no longer corresponds to the true gradient.

Putting together the gradients for both $\mathcal{L}_{prior}$ and $\mathcal{L}_{data}$ terms, we have

$$\frac{\partial \mathcal{L}}{\partial \mu_i} \approx \frac{\mu_i - \mu_p}{\sigma_p^2} + \sum_{\mathbf{x}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \frac{\partial \log p(\mathbf{x}\,|\,\mathbf{W}^{(k)})}{\partial w_i}, \tag{4.12}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_i^2} \approx \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_i^2}\right) + \sum_{\mathbf{x}\in\mathcal{D}} \frac{1}{T}\sum_{k=1}^{T} \left[\frac{\partial \log p(\mathbf{x}\,|\,\mathbf{W}^{(k)})}{\partial w_i}\right]^2, \tag{4.13}$$

where $\{w_i\}_{i=0}^T$ are the MC samples, $\mathbf{x}$ are the data points, i.e., input, target pairs. We then optimize the objective (4.3) with a gradient-descent method $\Psi_{m+1} = \Psi_m - k\frac{\partial \mathcal{L}}{\partial \Psi_m}$.

As with the BBB method, observing (4.13) we note that this parameterization may cause σ_i to assume negative values, thus calling for external constraints. Also similar to BBB, the Probabilistic Graphical Model (PGM) underlying Practical VI is the same as the one in Fig. 4.4. The difference between the two algorithms is rather a practical implementation issue, not a modeling assumption.

We summarize the resulting algorithm for optimizing a BNN with Practical ADF [16] in Algorithm 2. The case we illustrate is for a diagonal Gaussian variational posterior with parameters $\Psi = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ and centered Gaussian prior with diagonal covariance matrix $\sigma_p^2\mathbf{I}$, trained with a mini-batch of size 1 and uniformly distributed weighting of the complexity term $\mathcal{L}_{prior}$ across the mini-batches.

Algorithm 2: Practical ADF

```
1: while not converged do
2:     $\mathbf{w} \leftarrow \boldsymbol{\mu} + \sigma \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3:     Randomly sample a data example $\mathbf{x}_i$
4:     $\mathbf{g} \leftarrow -\nabla \log p(\mathbf{x}_i|\mathbf{w})$
5:     $\Delta\boldsymbol{\mu} \leftarrow (\boldsymbol{\mu} - \mu_p\mathbf{1})/(N\sigma_p^2) + \mathbf{g}$
6:     $\Delta\boldsymbol{\sigma}^2 \leftarrow (\boldsymbol{\sigma}^2 - \sigma_p^2\mathbf{1})/\left(N\sigma_p^2\boldsymbol{\sigma}^2\right) + (\mathbf{g} \odot \mathbf{g})$
7:     $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - k\Delta\boldsymbol{\mu}$
8:     $\boldsymbol{\sigma^2} \leftarrow \boldsymbol{\sigma^2} - k\Delta\boldsymbol{\sigma^2}$
9: end while
```

4.4 Probabilistic Backprop

Probabilistic Backpropagation (PBP) [19] solves the same problem as BBB but in a rather very different manner. While the algorithm of the previous section relies on optimizing the ELBO for the VI equation, PBP employs Assumed Density Filtering (ADF) and Expectation Propagation (EP), discussed in Sects. 3.2.2 and 3.2.3, respectively. The result is a parameter-free (not even learning rate) fully Bayesian method that has forward and backward phases as in common backpropagation. But instead of performing gradient descent in the parameter space, it incorporates information about the new data points into the posterior approximation at each iteration. Although another EP-based method had been proposed before [50], it focused on binary weights and its continuous extension performed poorly, not estimating the posterior variance.

In the year following PBP's publication [19], other researchers developed a variant for binary and multi-class classification problems [15]. In [52], the authors adopted the PBP framework to propose an online algorithm that models the correlations within the weights of the network with a matrix variate Gaussian distribution. However, here we shall focus solely on its original formulation for regression tasks since this already is enough work. PBP does not use the usual reverse mode automatic differentiation and requires non-trivial custom implementations, which is its major drawback and the reason why it has not seen widespread adoption. We start this section anticipating the reader that this is the most technically difficult section in the book.

Similar to the previous method, PBP assumes independence among the network weights and the existence of additive Gaussian noise $\mathcal{N}(\epsilon \mid 0, \gamma^{-1})$ with precision γ corrupting the observations. Although specifying the network architecture is not necessary for the other methods in this chapter, since they correctly function with any directed acyclic graph with no or almost none adaptations, the one at hand specializes in fully connected layers with Rectified Linear Unit (ReLU) [38], that is, $\max(0, x)$, as activation function. While modifying the model to conform to a different non-linearity is possible, it requires painstaking mathematical derivations as we can glance upon this section.

The graphical model for PBP is illustrated in Fig. 4.5 and its full posterior distribution over the parameters is given by

$$\begin{aligned} p(w, \gamma, \lambda \mid \mathbf{X}) &= \frac{p(y \mid W, \mathbf{X}, \gamma) p(w \mid \lambda) p(\lambda) p(\gamma)}{p(y \mid \mathbf{X})} \\ &\propto p(y \mid W, \mathbf{X}, \gamma) p(w \mid \lambda) p(\lambda) p(\gamma), \end{aligned} \tag{4.14}$$

where $p(y \mid X)$ is the model evidence, $p(y \mid W, X, \gamma)$ the observation model defining the likelihood factors, $p(w \mid \lambda)$ the prior distribution over the weights composed of univariate Gaussians with precision λ, that is,

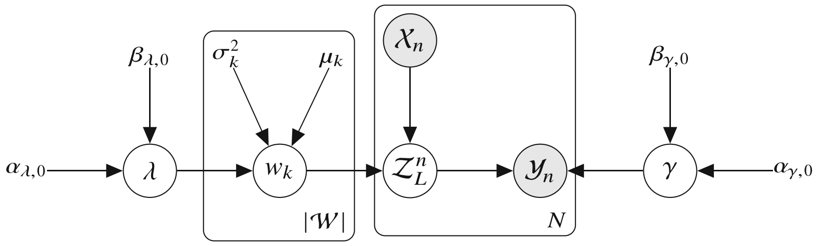

Fig. 4.5 PGM representation of the PBP model. The observed output y_n is a noisy observation of the model output z_L^n for the input x_n. The hyper-parameter λ governs the precision of the Gaussian prior distributions over the weights, whereas γ governs the precision noise of the Gaussian observation model

$$p(w_1, \cdots, w_{|\mathcal{W}|} \,|\, \lambda) = \prod_{w \in \mathcal{W}} \mathcal{N}(w \,|\, 0, \lambda^{-1}), \tag{4.15}$$

and $p(\lambda)$ and $p(\gamma)$ are hyper-prior distributions over the precision hyper-parameters of the likelihood and weight prior, respectively. We specify Gamma distributions $\text{Ga}(z \,|\, \alpha, \beta)$, given by

$$p(z \,|\, \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} z^{\alpha-1} \exp(-\beta z), \tag{4.16}$$

for both hyper-priors. In Sect. 2.5, we proved that Gamma is the conjugate prior for the Gaussian distribution with known mean and unknown precision parameter.

From the analysis of the influence of the hyper-parameters on the Gamma posterior (2.42), we choose them such that they impose a weak prior, not affecting the posterior distribution. Exactly the same reasoning is valid for the hyper-prior on γ.

PBP uses EP and ADF (Sects. 3.2.3 and 3.2.2 respectively) to update the parameters $w_1, \cdots, w_{|\mathcal{W}|}, \alpha_\gamma, \beta_\gamma, \alpha_\lambda$, and β_λ of the approximating distribution

$$\begin{aligned} &q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) \\ &\quad = \left[\prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \,|\, \mu_i, \sigma_i^2)\right] \text{Ga}\,(\lambda \,|\, \alpha_\lambda, \beta_\lambda)\, \text{Ga}\left(\gamma \,|\, \alpha_\gamma, \beta_\gamma\right), \end{aligned} \tag{4.17}$$

by cycling through the factors in (4.14) and including them one at a time. Thus, the total number of factors is the number of data points plus the (hyper-)priors, i.e., $|\mathcal{W}|$ for the weights and two for the precisions.

Since EP requires storing the approximate factors to compute the cavity distributions, it does not scale well with data. Its memory consumption grows linearly with the data set size. Thus, instead of performing EP updates for the likelihood factors, PBP repeatedly employs ADF multiple times, that is, instead of going through each data point only once, it incorporates the same factors N times. Although computationally more efficient, this approach has the risk of underestimating the parameter posterior variance. We are artificially observing more data, which in the limit of infinite data leads to the collapse of the posterior distribution onto the MLE as we assume the data points are (conditionally) independent and identically distributed (iid). However, this is clearly not the case when repeating the observations. Thus, the PBP should not run for many epochs. The authors [19] advise fewer than 100 and in our case study (Sect. 4.7) we run it for 40 epochs. Nevertheless, PBP is specifically designed for large data sets so this restriction does not matter much in practice. Yet, this is important to keep it in mind.

The models we analyze here and those employed in the original work have rather small sizes according to the current standards, i.e., one hidden layer with 50 units, so running EP updates is still feasible. Indeed, it is what the authors in [19] propose. In modern networks, which commonly contain hundreds of thousands of parameters, EP once again becomes a problem and ADF is the way to go.

The ADF update consists in including the true factor $f_i(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ into the current approximation $q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$, such that the updated approximation is

$$K^{-1} f(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma), \tag{4.18}$$

where K^{-1} is a normalization constant that assures $q(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ remains a proper probability distribution. This step usually causes the distribution to shift and no longer belong to the desired functional form. Then, to maintain the approximation manageable, we project it back to the same distribution class we had before the inclusion of the true factor, namely we minimize the KL divergence between the term in (4.18) and $q_{new}(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma)$ w.r.t. $w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma$, the parameters of the new distribution q_{new}. As already shown in (3.40), this is equivalent to matching the moments of both distributions, and each update consists of an iterative deterministic procedure so there is no learning rate to modulate the step size as for the other methods we discuss.

At the beginning, we have no information, so unless we have prior domain knowledge we initialize the parameters such that q is effectively uniform. This amounts to setting $\alpha_\lambda = \alpha_\gamma = 1$, $\beta_\lambda = \beta_\gamma = 0$, and $\mu = 0$, $\sigma^2 = \infty$ for every weight w.

The remainder of the section is split into three different subsections explaining how each type of factor is included into the model.

4.4.1 *Incorporating the Hyper-Priors $p(\lambda)$ and $p(\gamma)$*

The first factors to incorporate into the approximation are the priors over γ and λ. As shown in (2.42), the product of the prior precision Gamma and the Normal distribution results in a distribution with the same functional form as Gamma. This is exactly the case for (4.14), that is

$$\begin{aligned} q_{new}(w_1, \cdots, w_{|\mathcal{W}|}, \lambda, \gamma) &\propto \left[\lambda^{\alpha_\lambda - 1} \exp\left(-\lambda\beta_\lambda\right)\right]\left[\lambda^{\alpha_{\lambda,0}-1} \exp\left(-\lambda\beta_{\lambda,0}\right)\right] \\ &\propto \lambda^{(\alpha_\lambda + \alpha_{\lambda,0} - 1) - 1} \exp\left(-\lambda\left(\beta_{\lambda,0} + \beta_\lambda\right)\right). \end{aligned} \tag{4.19}$$

Thus, including the Gamma prior factors into q, and considering that $\alpha_\lambda = 0$, $\beta_\lambda = 1$, amounts to increment the values of the parameters γ and λ by

$$\begin{aligned} \alpha_{\gamma,\text{new}} &= \alpha_\gamma + \alpha_{\gamma,0} - 1 = \alpha_{\gamma,0} , \\ \beta_{\gamma,\text{new}} &= \beta_\gamma + \beta_{\gamma,0} = \beta_{\gamma,0} , \end{aligned} \tag{4.20}$$

where we have used the values defined above, i.e., $\alpha_\lambda = \alpha_\gamma = 0$, $\beta_\lambda = \beta_\gamma = 1$.

Since there are no approximations in these relationship, and, hence no loss of information, the hyper-priors need to be included only once.

4.4.2 *Incorporating the Priors on the Weights* $p(\mathbf{w} \mid \lambda)$

Next, we incorporate the priors over the weights $w \in \mathcal{W}$. The unnormalized shifted distribution after the inclusion of one such factor is

$$q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1}), \tag{4.21}$$

and the normalization constant is

$$\begin{aligned} K &= \int q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1}) d\mathbf{w} d\gamma d\lambda \\ &= \int \underbrace{\prod_{i=1}^{|\mathcal{W}|} \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) \mathrm{Ga}\left(\lambda \mid \alpha_\lambda, \beta_\lambda\right) \mathrm{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right)}_{q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)} \times \\ &\qquad\qquad \mathcal{N}(w_j \mid 0, \lambda^{-1}) dw_1 \cdots dw_{|\mathcal{W}|} d\gamma d\lambda \\ &= \int \mathcal{N}(w_j \mid \mu_j, \sigma_j^2) \left[\int \mathcal{N}(w_j \mid 0, \lambda^{-1}) \mathrm{Ga}\left(\lambda \mid \alpha_\lambda, \beta_\lambda\right) d\lambda\right] dw_j \\ &= \int \mathcal{N}(w_j \mid \mu_j, \sigma_j^2) \mathcal{T}_{2\alpha_\lambda}(w_j \mid 0, \beta_\lambda / \alpha_\lambda) dw_j , \end{aligned} \tag{4.22}$$

where we have used the result demonstrated in (A.34), that the integral of the product of a Gamma and a Gaussian distributions is the t-Student's distribution defined in (A.35). We continue the computation of K by approximating the t-Student's distribution $\mathcal{T}_{2\alpha_\lambda}(w_j \mid 0, \beta_\lambda/\alpha_\lambda)$ with a Gaussian with same mean and variance, what as we saw in Fig. A.1 is within reason for enough degrees of freedom ν, i.e., large α_λ. Thus, continuing the calculation of K:

$$\begin{aligned} K &\approx \int \mathcal{N}(w_j \mid \mu_j, \sigma_j^2)\mathcal{N}(w_j \mid 0, \beta_\lambda/(\alpha_\lambda - 1))dw_j \\ &= \int \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \mathcal{N}\left(w_j \middle| \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right) \int \mathcal{N}\left(w_j \middle| \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\frac{\mu}{\sigma^2}, \frac{\lambda(\alpha_\lambda - 1)}{\beta\lambda + \alpha - 1}\right) dw_j \\ &= \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \frac{\beta_\lambda}{\alpha_\lambda - 1}\right), \end{aligned} \tag{4.23}$$

where we resorted to the fact that the product of two Gaussians is also a Gaussian and is given by

$$\mathcal{N}(w_j \mid \mu_1, \sigma_1^2)\mathcal{N}(w_j \mid \mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 \mid \mu_2, \sigma_1^2 + \sigma_2^2)\mathcal{N}(w_j \mid \mu, \sigma^2), \tag{4.24}$$

with $\sigma^2 = \left(\sigma_1^{-2} + \sigma_2^{-2}\right)^{-1}$ and $\mu = \sigma^2\left(\mu_1\sigma_1^2 + \mu_2\sigma_2^2\right)$.

4.4.2.1 Update Equations for α_λ and β_λ

Updating the posterior approximation means matching its moments with those of the shifted distribution $s = K^{-1}q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1})$. However, the sufficient statistics for λ does not have closed form so we revise its parameters β_λ and α_λ by matching only its first and second moments, which still produces good results [36].

Let us now derive those update formulas. We start by noting that K in (4.23) is a function of μ_j, σ_j^2, β_λ, and α_λ, and make the dependency in the two latter terms explicit by writing $K(\beta_\lambda, \alpha_\lambda)$. Additionally, for brevity we denote $q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_j \mid 0, \lambda^{-1})$ as $f(\lambda)\text{Ga}(\lambda \mid \alpha_\lambda, \beta_\lambda)$ and compute

$$\begin{aligned} \mathbb{E}_q[\lambda] &= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \int \lambda f(\lambda)\text{Ga}(\lambda \mid \alpha_\lambda, \beta_\lambda)d\lambda \\ &= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \int \frac{\alpha_\lambda}{\beta_\lambda} f(\lambda)\text{Ga}(\lambda \mid \alpha_\lambda + 1, \beta_\lambda)d\lambda \\ &= \frac{1}{K(\beta_\lambda, \alpha_\lambda)} \left[\frac{\alpha_\lambda}{\beta_\lambda} K(\alpha_\lambda + 1, \beta_\lambda)\right] \end{aligned}$$

$$= \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda}. \tag{4.25}$$

Similarly, we obtain for the second moment

$$\mathbb{E}_q\left[\lambda^2\right] = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda\,(\alpha_\lambda + 1)}{K(\alpha_\lambda, \beta_\lambda)\beta_\lambda^2}. \tag{4.26}$$

Recalling the mean and variance formulas for the Gamma distribution (4.16), we equate them to the above expressions to obtain

$$\frac{\alpha_{\lambda,\text{new}}}{\beta_{\lambda,\text{new}}} = \frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda}, \tag{4.27}$$

$$\frac{\alpha_{\lambda,\text{new}}}{\beta^2_{\lambda,\text{new}}} = \frac{K(\alpha_\lambda + 2, \beta_\lambda)\alpha_\lambda(\alpha_\lambda + 1)}{K(\beta_\lambda, \alpha_\lambda)\beta^2} - \left[\frac{K(\alpha_\lambda + 1, \beta_\lambda)\alpha_\lambda}{K(\beta_\lambda, \alpha_\lambda)\beta_\lambda}\right]^2. \tag{4.28}$$

Solving the above equations for $\alpha_{\lambda,\text{new}}$ and $\beta_{\lambda,\text{new}}$, and abbreviating the normalizing coefficients $K_0 = K(\alpha_\lambda, \beta_\lambda)$, $K_1 = K(\alpha_\lambda + 1, \beta_\lambda)$, and $K_2 = K(\alpha_\lambda + 2, \beta_\lambda)$, we finally get

$$\alpha_{\lambda,\text{new}} = \left[K_0 K_2 K_1^{-2}(\alpha_\lambda + 1)\alpha_\lambda^{-1} - 1\right]^{-1}, \tag{4.29}$$

$$\beta_{\lambda,\text{new}} = \left[K_2 K_1^{-1}(\alpha_\lambda + 1)\beta_\lambda^{-1} - K_1 K_0^{-1}\alpha_\lambda\beta_\lambda^{-1}\right]^{-1}, \tag{4.30}$$

which are the update equations for the Gamma distribution over the precision parameter λ.

4.4.2.2 Update Equations for the μ and σ^2

It remains to establish how the mean and variance parameters of a given random weight change when we include its prior distribution into the posterior. The derivation in this section closely follows [17].

We first note that the shifted distribution can be conveniently written as $s = K^{-1} f(w_i)\mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$, where $f(w_i)$ comprises all factors in

$$q(w_1, \cdots, w_{|\mathcal{W}|}, \gamma, \lambda)\mathcal{N}(w_i \mid 0, \gamma^{-1}) \tag{4.31}$$

except the $\mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$, which we make explicit.

For μ_i, we start from the easily verifiable identity

$$\nabla_{\mu_i}\mathcal{N}(w_i \mid \mu_i, \sigma_i^2) = \sigma_i^{-2}(w_i - \mu_i)\mathcal{N}(w_i \mid \mu_i, \sigma_i^2), \tag{4.32}$$

which we rearrange to

$$w_i \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) = \mu_i \mathcal{N}(w_i \mid \mu_i, \sigma_i^2) + \sigma_i^2 \nabla_\mu \mathcal{N}(w_i \mid \mu_i, \sigma_i^2). \tag{4.33}$$

Multiplying on both sides by $K^{-1} f(w_i)$ and integrating over w_i leads to

$$\begin{aligned}\int w_i K^{-1} f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i &= \int \mu K^{-1} f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i \\ &\quad + \int \sigma^2 K^{-1} f(w_i) \nabla_\mu \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i\end{aligned} \tag{4.34}$$

$$\begin{aligned}\mathbb{E}_s\left[w_i\right] &= \mu + \sigma^2 K^{-1}\left[\nabla_\mu \int f(w_i) \mathcal{N}(w_i \mid \mu_i, \sigma^2) dw_i\right] \\ &= \mu + \sigma^2 K^{-1} \nabla_\mu K \\ &= \mu + \sigma^2 \nabla_\mu \log K.\end{aligned} \tag{4.35}$$

Since the first moment for the to-be-updated distribution $\mathcal{N}(w_i \mid \mu_i, \sigma^2)$ is μ_i, the update formula is

$$\mu_{i,\text{new}} = \mu + \sigma^2 \nabla_\mu \log K. \tag{4.36}$$

Through a similar identity for the derivative w.r.t. σ_i^2:

$$\nabla_{\sigma_i^2} \mathcal{N}(w_i \mid \mu_i, \sigma^2) = \frac{\sigma_i^{-2}}{2}\left(-1 + \sigma_i^{-2}(w_i - \mu_i)^2\right) \mathcal{N}(w_i \mid \mu_i, \sigma^2), \tag{4.37}$$

and following exactly the same procedure as before for μ_i, we arrive at $\mathbb{E}_s\left[w_i^2\right] = \sigma_i^2 + 2\left(\sigma_i^2\right)^2 \nabla_{\sigma_i^2} \log K$. Then, the variance of the shifted distribution is

$$\operatorname{Var}(w_i) = \mathbb{E}_s\left[w_i^2\right] - \left(\mathbb{E}_s\left[w_i\right]\right)^2 = \sigma_i^2 - \left(\sigma_i^2\right)^2\left[\left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K\right]. \tag{4.38}$$

From this, we establish the update for the variance of the normally distributed weight as

$$\sigma_{i,\text{new}}^2 = \sigma_i^2 - \left(\sigma_i^2\right)^2\left[\left(\nabla_\mu \log K\right)^2 - 2\nabla_{\sigma_i^2} \log K\right]. \tag{4.39}$$

Although we derived rules for performing ADF, that is, only including the individual true factors of the model, without ever removing the approximating factors to be updated, adapting them to EP is simple. The two key differences are:

1. Keep track of the parameters for each individual approximating factor;
2. Before the update, remove from the posterior the approximating factor corresponding to the true factor that will be incorporated (cavity distribution), effectively this means subtracting their contributions from the parameters of the posterior.

4.4.3 Incorporating the Likelihood Factors $p(\mathbf{y} \mid \mathbf{W}, \mathbf{X}, \gamma)$

In order to incorporate the information coming from a data point, we pass it forward through the network. Assuming the model to be a fully connected multi-layer network, at each layer following the input, PBP approximates the distribution of the resulting activations with a Gaussian distribution with same mean and variance, such that the input to the next layer is also Gaussian. At the last layer, we obtain the distribution of the output y_i given $\mathbf{x}_i$, to which we further apply the observation model, i.e., additive Gaussian noise with precision γ, which gives us $p(y_i \mid \mathbf{x}_i, \mathbf{W}, \gamma) = \mathcal{N}(y_i \mid f(\mathbf{x}_i, \mathbf{W}), \gamma^{-1})$. The likelihood factor is then included into the posterior approximation as usual: we shift the posterior by multiplying it by the likelihood factor stemming from the data point under consideration, compute the first and second moments of the resulting distribution, and update the parameters to obtain these moments.

Note that in the derivation of the update formulas (4.29), (4.30), (4.36), (4.39) we have not assumed any specific format for the factors being included into the posterior approximation. So the same equations can be used once again for the likelihood factors, the sole change being what the normalizing constant K is. In what follows we unveil the expression for K for the likelihood factors $\mathcal{N}(y_i \mid f(\mathbf{x}_i, \mathbf{W}), \gamma^{-1})$.

4.4.3.1 The Normalizing Factor

We consider a network with L layers and V_l units on each layer l, taking in vector-shaped inputs $\mathbf{x}_i$. Thus, the output $\mathbf{z}_l$ of each layer can be arranged into a vector, and the weights between two consecutive layers into a weight matrix $\mathbf{W}_l$ with dimensions $V_l \times (V_{l-1} + 1)$, where the $+1$ stems from the inclusion of a bias term. The pre-activation of a layer l is given by $\mathbf{a}_l = \mathbf{W}\mathbf{z}_{l-1}/\sqrt{V_{l-1} + 1}$, and for all except the last layer, this gets transformed according to the non-linear mapping $\max(a, 0)$, known as ReLU [38].

We make the simplifying assumption that the output z_L of the network at the last layer L is distributed as a Gaussian and proceed to compute the normalizing constant K of the associated shifted distribution as

$$K = \int q(\mathbf{w}, \gamma, \lambda)\mathcal{N}(\mathbf{y}_i \mid f(\mathcal{X}_i, \mathbf{w}), \gamma^{-1}) d\mathbf{w} d\gamma d\lambda$$

$$\begin{aligned}
&\approx \int q(\mathbf{w}, \gamma, \lambda) \mathcal{N}(\mathbf{y}_i \mid \mathbf{z}_L, \gamma^{-1}) \mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L}) d\mathbf{w} d\mathbf{z}_L d\gamma d\lambda \\
&= \int \mathrm{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right) \mathcal{N}(\mathbf{y}_i \mid \mathbf{z}_L, \gamma^{-1}) \mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L}) d\mathbf{z}_L d\gamma \\
&= \int \mathcal{T}_{2\alpha_\gamma}(\mathbf{y}_i \mid \mathbf{z}_L, \beta_\gamma/\alpha_\gamma) \mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L}) d\mathbf{z}_L \\
&\approx \int \mathcal{N}\left(\mathbf{y}_i \mid \mathbf{z}_L, \beta_\gamma/(\alpha_\gamma - 1)\right) \mathcal{N}(\mathbf{z}_L \mid \mu_{\mathbf{z}_L}, \sigma^2_{\mathbf{z}_L}) d\mathbf{z}_L \\
&= \mathcal{N}(\mathbf{y}_i \mid \mu_{\mathbf{z}_L}, \beta_\gamma/(\alpha_\gamma - 1) + \sigma^2_{\mathbf{z}_L}), \qquad (4.40)
\end{aligned}$$

where we have followed the same steps and performed the same approximations as in the derivation of (4.23).

Computing the mean $\mu_{\mathbf{z}_L}$ and variance $\sigma^2_{\mathbf{z}_L}$ of the last layer $\mathbf{z}_L$ amounts to propagating the input through the entire network. If we assume that the layer $l-1$ has output $\mathbf{z}_{l-1}$ with a diagonal covariance Gaussian distribution with mean and variance $\mu_{\mathbf{z}_{l-1}}$ and $\sigma^2_{\mathbf{z}_{l-1}}$, respectively, we can compute the mean and variance of the pre-activation $\mathbf{a}_l$ at the following layer according to

$$\mu_{\mathbf{a}_l} = \mathbb{E}\left[\mathbf{W}_l \mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right] = \bar{\mathbf{W}}_l \mathbf{z}_{l-1}/\sqrt{V_{l-1}+1} \qquad (4.41)$$

$$\begin{aligned}
\sigma^2_{\mathbf{a}_l} &= \mathrm{Var}\left(\mathbf{W}_l \mathbf{z}_{l-1}/\sqrt{V_{l-1}+1}\right), \\
&= \frac{1}{V_{l-1}+1}\left[(\mathbb{E}\,[\mathbf{W}_l])^2\, \mathrm{Var}\left(\mathbf{z}_{l-1}\right) + \mathrm{Var}\left(\mathbf{W}_l\right)\left(\mathbb{E}\left[\mathbf{z}_{l-1}\right]\right)^2 + \mathrm{Var}\left(\mathbf{W}_l\right) \mathrm{Var}\left(\mathbf{z}_{l-1}\right)\right] \\
&= \frac{1}{V_{l-1}+1}\left[\left(\bar{\mathbf{W}}_l \odot \bar{\mathbf{W}}_l\right)\sigma^2_{\mathbf{z}_{l-1}} + \mathbf{V}_l\left(\mu_{\mathbf{z}_{l-1}} \odot \mu_{\mathbf{z}_{l-1}}\right) + \mathbf{V}_l \sigma^2_{\mathbf{z}_{l-1}}\right], \qquad (4.42)
\end{aligned}$$

where $\bar{\mathbf{W}}_l$ and $\mathbf{V}_l$ are the mean and variance matrices for the weights in $\mathbf{W}_l$, whose values are determined by the corresponding Gaussian factors of the model.

If the number V_{l-1} of inputs to the layer l is large enough and we further assume the entries of $\mathbf{a}_l$ are independent, we can invoke the Central Limit Theorem and claim that the pre-activation $\mathbf{a}_l$ is Normally distributed with the above mean and variance [50].

We are now to consider the effect of the non-linear activation function on $\mathbf{a}_l$. The $\max(0, a_{i,l})$ operation causes all probability density spread over $\mathbb{R}^-$ to concentrate at zero as Fig. 4.6 indicates. The resulting distribution is called rectified Gaussian and has its PDF given by

$$\mathcal{N}^R\left(a_{i,l}; \mu_{i,l}, \sigma^2_{i,l}\right) = \Phi\left(-\frac{\mu_{i,l}}{\sigma_{i,l}}\right)\delta(a_{i,l}) + \frac{1}{\sqrt{2\pi\sigma^2_{i,l}}} e^{-\frac{(a_{i,l}-\mu_{i,l})^2}{2\sigma^2_{i,l}}}\, \mathrm{U}(a_{i,l}), \qquad (4.43)$$

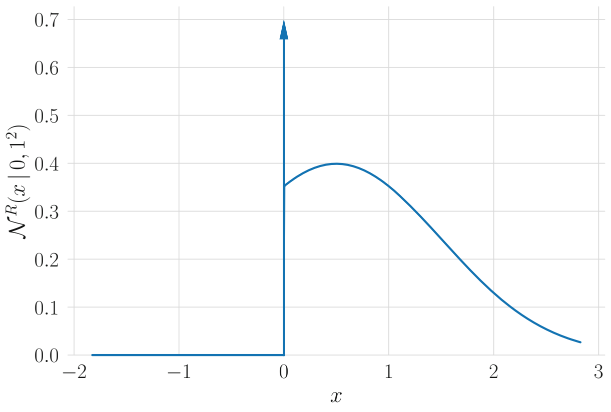

Fig. 4.6 PDF of the rectified Gaussian distribution $\mathcal{N}^R(x; 0.5, 1^2)$

where μ, σ^2 are the mean and variance of the Gaussian prior to rectification, $\Phi(\cdot)$ is the CDF of the standard Gaussian at the specified point, $\delta(\cdot)$ is Dirac's impulse function, and $\mathrm{U}(\cdot)$ is the unit step function. Its mean and variance are

$$\mu_{z_{i,l}} = \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\mu_{a_{i,l}} + \sigma_{a_{i,l}}\phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) \tag{4.44}$$

$$\sigma^2_{a_{i,l}} = m\left(\mu_{a_{i,l}} + \sigma_{a_{i,l}}\kappa\right)\Phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) + \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\sigma^2_{a_{i,l}}\left(1 - \kappa\left(\kappa + \frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)\right), \tag{4.45}$$

where $\kappa = \phi\left(-\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right) \Big/ \Phi\left(\frac{\mu_{a_{i,l}}}{\sigma_{a_{i,l}}}\right)$, and $\phi(\cdot)$ is the PDF of the standard Gaussian at the specified position.

The output distribution of the corresponding layer is then a Gaussian with entries determined by the above formulas plus an extra element we append for the bias term, which has mean $\mathbf{1}$ and variance $\mathbf{0}$. Then, finding the values μ_{z_L} and $\sigma^2_{z_L}$ consists in iteratively computing Eqs. (4.41), (4.42), (4.44), (4.45) from the first until the last layer for each data point $(\mathbf{x}_i, \mathbf{y}_i)$.

We summarize the many factors that compose PBP's distribution in Table 4.1 and highlight how many of each there are. The steps of the ADF-update-only method are available in Algorithm 3. We condense the forward pass responsible for computing the output distribution $\mathcal{N}(\mathcal{Y}_i \mid f(\mathcal{X}_i, \mathbf{w}), \gamma^{-1})$ into a single step in line 17.

It is important to note that the authors [19] assume the inputs are normalized, i.e., zero mean and unit variance. Hence, we need to normalize the data points before feeding them to the model and then to denormalize the obtained outputs.

Table 4.1 Summary of PBP's factors, their distributions, and quantities

Type	Symbol	Distribution	Quantity
Hyper-prior	$p(\lambda)$	$\text{Ga}\,(\lambda \mid \alpha_\lambda, \beta_\lambda)$	1
Hyper-prior	$p(\gamma)$	$\text{Ga}\left(\gamma \mid \alpha_\gamma, \beta_\gamma\right)$	1
Prior	$p(w_j \mid \lambda)$	$\mathcal{N}(w_j \mid 0, \lambda^{-1})$	$\lvert\mathcal{W}\rvert$
Likelihood	$p(\mathcal{Y}_j \mid \mathbf{W}, \mathcal{X}_j, \gamma)$	$\mathcal{N}(\mathcal{Y}_j \mid f(\mathcal{X}_j, \mathbf{w}), \gamma^{-1})$	N

Algorithm 3: Probabilistic Backprop

```
1:  Initialise parameters $\alpha_\lambda, \alpha_\gamma, \beta_\lambda, \beta_\gamma, \{\mu_j, \sigma_j^2\}_{j=0}^{|\mathcal{W}|}$
2:  for $s \in \{\lambda, \gamma\}$ do
3:      $\alpha_s \leftarrow \alpha_s + \alpha_{s,0} - 1$
4:      $\beta_s \leftarrow \beta_s + \beta_{s,0}$
5:  end for
6:  while not converged do
7:      for $j = 1$ to $|\mathcal{W}|$ do
8:          for $s = 0$ to $2$ do
9:              $K_s \leftarrow \mathcal{N}\left(\mu_j \middle| 0, \sigma_j^2 + \beta_\lambda/(\alpha_\lambda - 1 + s)\right)$
10:         end for
11:         $\alpha_\lambda \leftarrow \left[K_0 K_2 K_1^{-2}(\alpha_\lambda + 1)\alpha_\lambda^{-1} - 1\right]^{-1}$
12:         $\beta_\lambda \leftarrow \left[K_2 K_1^{-1}(\alpha_\lambda + 1)\beta_\lambda^{-1} - K_1 K_0^{-1}\alpha_\lambda\beta_\lambda^{-1}\right]^{-1}$
13:         $\mu_j \leftarrow \mu_j + \sigma^2 \nabla_\mu \log K_0$
14:         $\sigma_j^2 \leftarrow \sigma_j^2 - \left(\sigma_j^2\right)^2 \left[\left(\nabla_{\mu_j} \log K_0\right)^2 - 2\nabla_{\sigma_j^2} \log K_0\right]$
15:     end for
16:     for $j = 1$ to $N$ do
17:         $\mu_{\mathbf{z}_L}, \sigma_{\mathbf{z}_L}^2 \leftarrow f(\mathcal{X}_j, \mathcal{W})$
18:         for $s = 0$ to $2$ do
19:             $K_s \leftarrow \mathcal{N}(\mathbf{y}_i \mid \mu_{\mathbf{z}_L}, \sigma_{\mathbf{z}_L}^2) + \beta_\gamma/(\alpha_\gamma - 1 + s)$
20:         end for
21:         $\alpha_\gamma \leftarrow \left[K_0 K_2 K_1^{-2}(\alpha_\gamma + 1)\alpha_\gamma^{-1} - 1\right]^{-1}$
22:         $\beta_\gamma \leftarrow \left[K_2 K_1^{-1}(\alpha_\gamma + 1)\beta_\gamma^{-1} - K_1 K_0^{-1}\alpha_\gamma\beta_\gamma^{-1}\right]^{-1}$
23:         $\mu_j \leftarrow \mu_j + \sigma^2 \nabla_\mu \log K_0$
24:         $\sigma_j^2 \leftarrow \sigma_j^2 - \left(\sigma_j^2\right)^2 \left[\left(\nabla_{\mu_j} \log K_0\right)^2 - 2\nabla_{\sigma_j^2} \log K_0\right]$
25:     end for
26: end while
```

4.5 MC Dropout

The Monte Carlo Dropout [13], usually referred to as MC Dropout, stems from reinterpreting Dropout [51] as doing approximate Bayesian inference. Consequently, it suffices to use Dropout both during training and testing to obtain the advantages of Bayesian inference and model uncertainty measures.

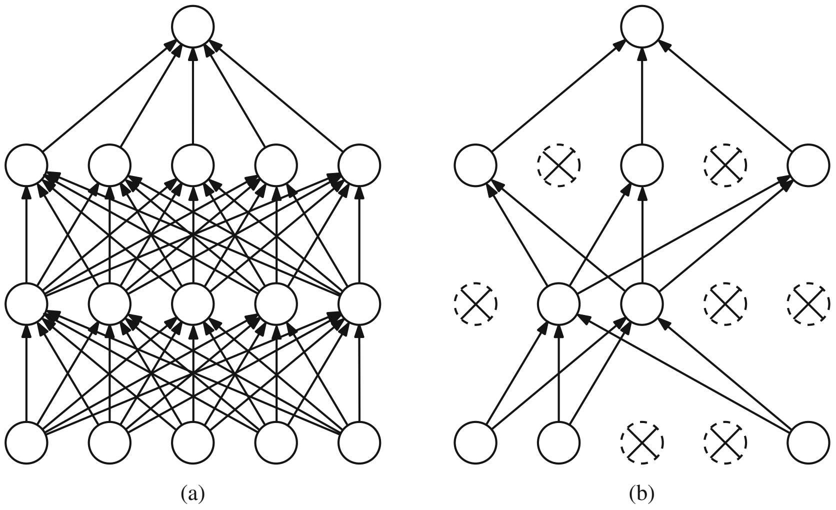

Fig. 4.7 Effect of Bernoulli dropout on the network. Setting the output of a unit to zero is equivalent to removing that unit from the network. That nodes' inputs also become irrelevant because they are no longer propagated forward and so we remove them from the drawing in (**b**). (**a**) Standard neural network. (**b**) Neural network after dropout

4.5.1 Dropout

First, we review Dropout [51]. Succinctly, it is a stochastic regularization technique to avoid overfitting the data. The basic idea is to corrupt the model's units with random multiplicative noise while training. Mathematically, it amounts to multiplying the input $\mathbf{h}_l$ of layer l pointwise by a realization of a random vector $\boldsymbol{\epsilon}_l$, such that $\hat{\mathbf{h}_\mathbf{l}} = \mathbf{h}_l \odot \boldsymbol{\epsilon}_l$.

In the case of Bernoulli dropout, each unit $h_{j,l}$ at layer l is randomly *dropped out* with probability $1 - p$, i.e., its output value is set to zero, at each iteration according to $\epsilon_{j,l} \sim Bern(p)$, as illustrated in Fig. 4.7. Dropping units causes different subnetworks with considerably less parameters to be used at each iteration (12 instead of 55 in Fig. 4.7, for instance). When testing, all units are kept as if an ensemble with all subnetworks was being used for evaluation.

Other works propose other types of noise. For example, in [28, 51], the authors study corrupting the activations with multiplicative Gaussian noise, and in [56], independently injecting noise on each weight, instead of on the input. The latter technique is called DropConnect.

4.5.2 A Bayesian View

Optimizing a model with dropout and an approximate Bayesian inference model leads to similar objective functions with similar stochastic gradient update steps. This similarity is so strong that under some conditions they are indeed equivalent [13]. Although we shall only consider here the Bernoulli Dropout, a similar development is possible for other types of noise.

Let us first review the cost function of a standard deterministic neural network $f(\cdot;\Theta)$ with deterministic parameters Θ:

$$\mathcal{L} = \mathcal{L}_{data}\left(\mathcal{D}, f(\cdot;\Theta)\right) + \mathcal{L}_{reg}(\Theta), \tag{4.46}$$

where the first term is data-dependent and measures the model's prediction error, and the second is a regularization term to help against overfitting. Considering a regression task with data points $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid 1 \leqslant i \leqslant N\}$, a model with parameters $\Theta = \{\mathbf{M}_l \mid 1 \leqslant l \leqslant L\}$, and $\mathcal{L}_{reg}$ as the usual ℓ_2-norm with strength factors $\lambda_{\mathbf{M}}$, (4.46) becomes

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \frac{1}{2}\left(\mathbf{y} - f(\mathbf{x};\Theta)\right)^2 + \sum_{\mathbf{M}\in\Theta} \lambda_{\mathbf{M}} \|\mathbf{M}\|_2{}^2 . \tag{4.47}$$

If we reinterpret Dropout as instead of corrupting the layers' inputs, corrupting the corresponding weights, we get for an arbitrary intermediate layer l with activation function $g_l(\cdot)$, the expression

$$\begin{aligned}
\mathbf{h}_l &= g_l\left(\mathbf{M}_l \hat{\mathbf{h}}_{l-1}\right) \\
&= g_l\left(\mathbf{M}_l\left(\boldsymbol{\epsilon}_l \odot \mathbf{h}_{l-1}\right)\right) \\
&= g_l\left(\mathbf{M}_l\left(\operatorname{diag}(\boldsymbol{\epsilon}_l)\mathbf{h}_{l-1}\right)\right) \\
&= g_l\left(\left(\mathbf{M}_l \operatorname{diag}(\boldsymbol{\epsilon}_l)\right)\mathbf{h}_{l-1}\right) \\
&= g_l\left(\mathbf{W}_l \mathbf{h}_{l-1}\right),
\end{aligned} \tag{4.48}$$

where $\mathbf{M}_l$ is the (deterministic) weight matrix, $\mathbf{h}_{l-1}$ the input, $\boldsymbol{\epsilon}_l$ the random noise, and $\mathbf{W}_l = \mathbf{M}_l \operatorname{diag}(\boldsymbol{\epsilon}_l)$.

We have demonstrated that multiplying the input is equivalent to multiplying the columns of the upcoming weight matrix. Considering a Bernoulli distribution on each element of $\boldsymbol{\epsilon}_l$, when one of its entries assumes value equal to 0, it zeros the corresponding column of $\mathbf{W}_l$ (as $\mathbf{W}_l = \mathbf{M}_l \operatorname{diag}(\boldsymbol{\epsilon}_l)$). Zeroing the column is equivalent to dropping every input of a neuron, which in turn is the same as dropping the neuron itself, as illustrated in Fig. 4.8.

Hence, applying dropout on a deterministic neural network can be interpreted as a transformation to a NN whose weights are sampled from a distribution. Looking

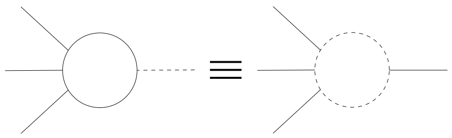
Fig. 4.8 Dropping every input of a neuron (figure in the left) is equivalent to dropping the neuron itself

from this perspective, dropout is a way of using BNNs. Figure 4.10 shows the resulting weight matrix after being transformed by realizations of different types of noise.

If we now rewrite (4.47) taking this into consideration and make the sampling explicit, we get

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left(\mathbf{y}_i - f^{(i)}(\mathbf{x}_i;\Theta)\right)^2 + \sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2{}^2, \tag{4.49}$$

where the notation $f^{(i)}(\cdot;\Theta)$ indicates a sample of the random parameters drawn for the data point $(\mathbf{x}_i, \mathbf{y}_i)$. Since Θ now defines distribution parameters, we replace it by Ψ in order to keep compliance with our notation of variational parameters and ease the comparison with other methods.

Substituting the first term of the above equation according to (4.1), we obtain

$$\begin{aligned}\mathcal{L} &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(\mathbf{y}_i \,|\, \mathbf{X}_i, \mathbf{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2{}^2 - \frac{\sigma_n^2}{2}\log\left(2\pi\sigma_n^2\right) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sigma_n^2 \log p(\mathbf{y}_i \,|\, \mathbf{X}_i, \mathbf{W}^{(i)}) + \sum_{l=1}^{L}\lambda_l \|\mathbf{M}_l\|_2{}^2 + \text{const}\,, \end{aligned} \tag{4.50}$$

where σ_n is the observation noise, $\mathbf{W}^{(i)}$ is one sample from the distribution. The term that only depends on σ_n is considered a constant since this hyper-parameter is set by cross-validation and not gradient-descent optimization.

Equation (4.50) is pretty similar to a one-sample MC estimator of the VI cost function $\hat{\mathcal{L}}_{VI}$ defined in (3.8), and (4.3), which after approximating with MC integration becomes

$$\begin{aligned}\hat{\mathcal{L}}_{VI} &= -\frac{1}{T}\sum_{k=1}^{T}\log p(\mathbf{d}\,|\,\mathbf{W}^{(k)}) + D_{KL}\left(q(\mathbf{W}^{(k)};\Psi)\|p(\mathbf{W}^{(k)})\right) \\ &= -\log p(\mathbf{d}\,|\,\mathbf{W}^{(1)}) + D_{KL}\left(q(\mathbf{W}^{(1)};\Psi)\|p(\mathbf{W}^{(1)})\right) \\ &= -\sum_{i=1}^{N}\log p(\mathbf{y}_i\,|\,\mathbf{X}_i,\mathbf{W}^{(1)}) + D_{KL}\left(q(\mathbf{W}^{(1)};\Psi)\|p(\mathbf{W}^{(1)})\right). \end{aligned} \tag{4.51}$$

Taking the derivative of both (4.50) and (4.51) w.r.t. their parameters, we note that they possess the same objective (up to a constant scale factor), as long as we assure that

$$\frac{\partial}{\partial \Psi} D_{KL}\left(q(\mathbf{W}^{(1)}; \Psi) \| p(\mathbf{W}^{(1)})\right) = \frac{N}{\sigma_n^2} \frac{\partial}{\partial \Psi} \sum_{l=1}^{L} \lambda_l \|\mathbf{M}_l\|_2^2. \tag{4.52}$$

This condition is now the sole thing impeding us from using dropout (or any other similar noise injection technique) as an approximate Bayesian model. For (4.52) to hold, we have to choose the hyper-parameters σ_n and $\Lambda = \{\lambda_l | 1 \leqslant l \leqslant L\}$ such that they induce a sensible prior $p(\mathbf{W})$ for the underlying variational distribution $q(\mathbf{W}; \Psi)$. In the Appendix of [12], the author goes deeper in the conditions necessary to be attended in order for Eq. (4.52) to hold, where they assume that the weights of the neural network are sampled from a centered Gaussian distribution, i.e., $w_{j,l} \sim \mathcal{N}(0, \gamma_l^{-1})$.

Let us stop here and digest this result. No specific assumption about the neural network architecture was assumed other than having a Dropout layer before each weight layer. This is the only restriction to obtain approximate Bayesian inference with the model in Fig. 4.9, the other being readily attended: to every choice of dropout probability $1 - p_l$, observation noise σ_n^2 (or, equivalently, noise precision τ_n), and regularization strength λ_l, corresponds a prior precision γ_l (or, according to [13] a prior length-scale l_l), whether or not its value is reasonable for the problem at hand. For other network architectures such as convolutional [12] and recurrent [14], few additional considerations are required to achieve a similar result. If the employed model does not have Dropout in between every layer, as is usually the case in pretrained models with only the last fully connected layers of the classifier possessing Dropout, we can think of them as having a deterministic feature-extractor part and a subsequent approximate Bayesian classifier. Although not as powerful, this is still a nice interpretation if we want to do inference and are bound to a given model.

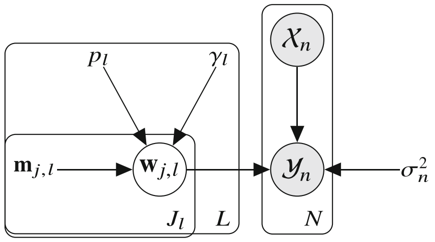

Fig. 4.9 PGM representation of the MCDO model. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter σ_n^2. The j-th weight vector $\mathbf{m}_{j,l}$ out of the J_l from the l-th layer get selected by Bernoulli random variables with success rate p_l and $\mathbf{w}_{j,l}$ have centered Gaussian priors with fixed precision γ_l, whose value is readily determined by the choice of the two previous hyper-parameters together with the regularization strength λ_l

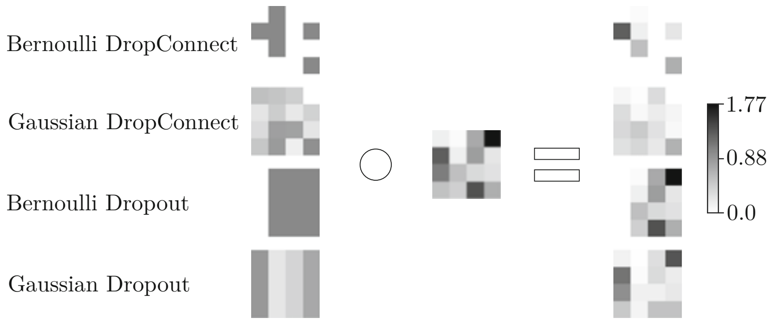

Fig. 4.10 The effect on the network weights of using different stochastic regularization techniques on the same deterministic weight matrix **M**. Each technique corresponds to a distinct base distribution and leads to distinct variational distribution. For the same weight matrix, different base variational distributions

Also, the posterior approximation for Bernoulli Dropout factorizes over different layers and over connections going out of the same unit, but not over the connections arriving at the same unit. As the same Bernoulli random variable acts on the same weight matrix column, naturally they are not independent. The other methods of this chapter use mean-field approximation to the posterior, completely missing any codependency among the weights. In this sense, MCDO is less restrictive.

However, the author of [12] warns that to get well-calibrated uncertainty estimates the dropout probability must be optimized as well. Since this is a variational parameter, it cannot be directly chosen by observing the ELBO objective [12]. The recommendation is then to set it by maximizing the log-likelihood over a validation set.

We summarize the resulting procedure in Algorithm 4, where we illustrate the case for the Bernoulli dropout trained with a mini-batch of size 1. However, as pointed out at the beginning of this section, other stochastic regularizers can be recast as performing approximate Bayesian inference by following a similar derivation, as Fig. 4.10 illustrates. For example, for DropConnect [56] the only difference is in using separate random variables for each weight instead of one for each column of the weight matrix. It is important to note that not all resignifications go without problems, Gaussian Dropout [51] as Bayesian inference with a log-uniform prior [28] has had some issues pointed out in [24]. If instead of using multiplicative we consider additive Gaussian noise for each weight parameter, we recover the algorithm of Sect. 4.3.1.

4.6 Fast Natural Gradient

The parameter space is in general Riemannian and not Euclidean, so learning methods should take the structure of the space into account [2]. Natural gradient

Algorithm 4: MC Dropout

```
1: while not converged do
2:     Randomly sample a data example $\{\mathbf{x}_i, \mathbf{y}_i\}$
3:     for $l = 1$ to $L$ do
4:         $\mathbf{W}_l \leftarrow \mathbf{M}_l \mathrm{diag}(\boldsymbol{\epsilon}_l)$, where $\boldsymbol{\epsilon}_l \sim \mathrm{Bern}(p_l)$
5:     end for
6:     $\mathbf{g} \leftarrow \frac{1}{2}\nabla\left(\mathbf{y}_i - f(\mathbf{x}_i; \{\mathbf{W}_l\}_{l=0}^{L})\right)^2 + \sum_{l=1}^{L} \lambda_l \nabla \|\mathbf{W}_l \mathrm{diag}(\boldsymbol{\epsilon}_l)\|_2{}^2$
7:     $\mathbf{m}_{j,l} \leftarrow \mathbf{m}_{j,l} - k\mathbf{g}$
8: end while
```

methods do that by warping the gradient according to the information geometry encoded into the Fisher information matrix (see Appendix A.4). As a consequence, they are invariant (up to first order) to changes in the parameterization of the problem, what is in stark contrast to standard gradient descent, whose efficiency and convergence rate are sensitive to the parameterization.

Current frameworks focus on MLE and adapting them for VI requires modifications in the code, increasing development time, memory requirements, and computation costs. For example, the algorithms of Sects. 4.3 and 4.4 have twice the number of parameters of a deterministic model with the same architecture, besides the additional implementation effort. Adaptive optimizers further enlarge the costs since each parameter has its own scaling variable that regulates the learning rate.

The authors in [26] build upon previous work [25] on natural gradient for Gaussian MFVI and propose a series of progressively more practical but less accurate optimizers. It is a lengthy read to grasp all the details, but certainly worth the effort. Here, we review and rederive the core algorithm of [26], named Vadam.

4.6.1 Vadam

From all reviewed methods, Vadam [26] is the more recent and practical one. Similar to the Adam optimizer [27] by construction, it is a natural gradient method (see Appendix A.4) with momentum designed specifically for MFVI. Starting from a parameter update equation proposal, the authors [26] embed several approximations defining different algorithms until reaching the method they name Vadam, for Variational Adam.

Gradient optimizers with momentum establish the update step as a linear combination between the steepest descent direction and the last displacement [8], such as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \bar{\alpha}_t \nabla_{\mathbf{w}} f(\mathbf{w}_t) + \bar{\gamma}_t(\mathbf{w}_t - \mathbf{w}_{t-1}), \tag{4.53}$$

where $\{\bar{\alpha}_t\}$ and $\{\bar{\gamma}_t\}$ form a sequence of scalars that determines the contribution of each term and must obey the convergence conditions discussed in Sect. 3.2.1.5.

The latter term in (4.53) keeps the algorithm's movement along previous search directions, and is thus named momentum. Reasoning about its dynamics since the first iteration, each step can be understood as an exponentially decaying average of past gradients, hence the tendency to accumulate contributions in directions of persistent descent, while directions that oscillate tend to cancel out, or at least remain small [8].

Instead of (4.53), the authors of [26] propose

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \bar{\alpha}_t \widetilde{\nabla}_{\eta} f(\boldsymbol{\eta}_t) + \bar{\gamma}_t (\boldsymbol{\eta}_t - \boldsymbol{\eta}_{t-1}), \tag{4.54}$$

where $\widetilde{\nabla}$ is the natural gradient and the optimization is on the natural parameter $\boldsymbol{\eta}$ of an exponential family member. For such family, the natural gradient assumes a simple and efficient form, requiring less memory and computations. Besides, it improves the convergence rate by exploiting the information geometry of posterior approximations.

Constraining the variational approximation to the exponential family allows the use of the relation [3]

$$\widetilde{\nabla}_{\eta} f(\boldsymbol{\eta}) = \mathcal{I}^{-1}(\boldsymbol{\eta}) \nabla_{\eta} f(\boldsymbol{\eta}) = \nabla_{\mathbf{m}} f(\mathbf{m}), \tag{4.55}$$

which states that the natural gradient w.r.t. the natural parameter is equal to the gradient w.r.t. the mean parameter $\mathbf{m}$ when $f(\cdot)$ is parameterized according to $\mathbf{m} = \mathbb{E}[\mathbf{u}(\mathbf{w})]$. The identity in (4.55) frees us from computing the Fisher matrix and its inverse, that is why it is so useful and many other practical natural gradient algorithms resort to it [22, 23, 25].

In the specific case of independent univariate Gaussian weights (mean-field assumption), writing (4.54) as a minimization problem with a KL constraint (see Appendix A.4), using (4.55) and solving the resulting Lagrangian lead to

$$\mu_{t+1} = \mu_t - \frac{\beta_t}{1-\alpha_t} \sigma_{t+1}^2 \nabla_{\mu} \mathcal{L}_t + \frac{\alpha_t}{1-\alpha_t} \sigma_{t+1}^2 \sigma_{t-1}^{-2} (\mu_t - \mu_{t-1}), \tag{4.56}$$

$$\sigma_{t+1}^{-2} = \frac{1}{1-\alpha_t} \sigma_t^{-2} - \frac{\alpha_t}{1-\alpha_t} \sigma_{t-1}^{-2} + \frac{2\beta_t}{1-\alpha_t} \nabla_{\sigma^2} \mathcal{L}_t. \tag{4.57}$$

The pair of update Eqs. (4.56) and (4.57) is the natural momentum extension of [25]. We immediately note that the learning rate of μ gets scaled by the variance. Additionally, σ^2 may assume negative values just like the methods in Sect. 4.3, thus one needs external constraints to sidestep this issue.

No specific knowledge of the cost function $\mathcal{L}$ has been absorbed into the algorithm so far. However, now we take into consideration that the cost function is the negative ELBO defined in (4.3) and also specify univariate Gaussian priors $p(w) = \mathcal{N}(w; 0, \sigma_p^2)$ for the weights. Recalling the derivatives of the KL term already calculated in (4.7)–(4.13) and again using the identities (4.9) and (4.10), we get

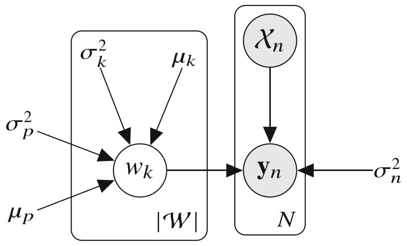

Fig. 4.11 PGM representation of the Vadam model. It is the same as the one in Sect. 4.3. The observed output $\mathbf{y}_n$ is a noisy observation of the model output for the input $\mathbf{x}_n$ with the variance noise determined by the fixed parameter σ_n^2. The constant values $\{\mu_p, \sigma_p^2\}$ govern the Gaussian prior distributions over the weights, while $\{\mu_k, \sigma_k^2\}$ their posteriors

$$\nabla_\mu \mathcal{L} = N\mathbb{E}_q\left[\nabla_w h(w)\right] + \frac{\mu}{\sigma_p^2}, \tag{4.58}$$

$$\nabla_{\sigma^2} \mathcal{L} = \frac{N}{2}\mathbb{E}_q\left[\nabla_w^2 h(w)\right] + \frac{1}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma^2}\right), \tag{4.59}$$

where N is the data set size and $h(w) = -\frac{1}{N}\sum_{i=1}^{N}\log(x_i \mid w)$, the average negative log-likelihood.

Now, after determining the prior and posterior distributions over the weights, we have completely defined the underlying Vadam model, shown in Fig. 4.11. t has the same structure as the one used in Sect. 4.3 (Fig. 4.4), the difference between both methods being the approximations included in Vadam to make it more computationally efficient.

As one might already expect, we use one-sample MC estimators for the expectations in (4.58) and (4.59), as well as replace the gradients, so far computed from the entire data set, with their stochastic versions $\widehat{\nabla}_w$ and $\widehat{\nabla}_w^2$, computed from a mini-batch. Since second derivatives are computationally expensive, besides (4.59) being able to lead σ^2 to negatives values, we resort to the GGN approximation for $\widehat{\nabla}_w^2$ (see Appendix A.3). This last step requires calculating the square of the first-order derivative for each mini-batch element; however, modern frameworks are not optimized to operate separately on each element of a batch after computing its derivatives. Thus, we incorporate yet another approximation

$$\widehat{\nabla}_w^2 h(w_t) \approx \frac{1}{M}\sum_{i=1}^{M}\widehat{\nabla}_w h(w_t;\mathbf{x}_i)^2 \approx \left(\frac{1}{M}\sum_{i=1}^{M}\widehat{\nabla}_w h(w_t;\mathbf{x}_i)\right)^2. \tag{4.60}$$

While the first approximation in (4.60) is the GGN, the last is known as the gradient magnitude approximation and employed by several usual optimizers [11, 27, 54]. It causes σ^2 to act as diagonal rescaling that simply assures equal progress along each axis of μ rather than closely approximating the curvature [8] (disregarding the momentum term of the update equation that counter-balances this effect by favoring historically good directions).

The gradient magnitude approximation biases the estimation even more than GGN, its expectation is in between that of GGN and the squared gradient of the full-batch. As the mini-batch size increases, the bias also increases: if the whole data set is used to compute this approximation, then all second-order information is lost, while if computed if a single data point it is equal to the GGN. Hence, there is a compromise between biasing estimations but converging quickly versus being "exact" (GGN-wise) but slow.

For practicality, the authors in [26] define the scaled prior precision $\widetilde{\lambda} = \sigma_p^{-2}/N$ and the new parameter $s_t = (\sigma_t^{-2} - \sigma_p^{-2})/N$. Moreover, they arbitrarily apply square root on the scaling vector s_t in the μ update formula so that the method gets more similar to Adam. Although this modification does not change the algorithm's fixed point solutions, it alters the dynamics [26]. The Vadam weight update equations are then

$$\mu_{t+1} = \mu_t - \bar{\alpha}_t \left[\frac{1}{\sqrt{s_t} + \widetilde{\lambda}} \right] \left(\widehat{\nabla}_w h(w_t) + \mu_t \widetilde{\lambda} \right) + \bar{\gamma}_t \left[\frac{\sqrt{s_t} + \widetilde{\lambda}}{\sqrt{s_{t+1}} + \widetilde{\lambda}} \right] (\mu_t - \mu_{t+1}), \tag{4.61}$$

$$s_{t+1} = (1 - \bar{\alpha}_t)\, s_t + \bar{\alpha}_t \widehat{\nabla}_w^2 h(w_t), \tag{4.62}$$

where $\widehat{\nabla}_w$ and $\widehat{\nabla}_w^2$ are the unbiased stochastic approximations of ∇_w and ∇_w^2, respectively.

Unwinding these update equations and using different step sizes γ_1 and γ_2 for μ and s instead of $\bar{\alpha}_t$ and $(1-\bar{\alpha}_t)$, respectively, we get the Algorithm 5. Remember that the scale factor s actually relates to σ^2 by $\sigma_t^{-2} = N s_t + \sigma_p^{-2}$ and each weight sample w_t is drawn from the distribution $\mathcal{N}(\mu_t, \sigma_t^2)$. The implementation differences from Adam [27], in red in Algorithm 5, are responsible for enabling ADF.

Algorithm 5: Vadam

```
1: while not converged do
2:     $\mathbf{w} \leftarrow \boldsymbol{\mu} + \sigma \odot \epsilon$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\sigma} \leftarrow 1/\sqrt{N\mathbf{s} + \sigma_p^{-2}}$
3:     Randomly sample a data example $\mathbf{x}_i$
4:     $\mathbf{g} \leftarrow -\nabla \log p(\mathbf{x}_i|\mathbf{W})$
5:     $\mathbf{m} \leftarrow \gamma_1 \mathbf{m} + (1 - \gamma_1)(\mathbf{g} + \sigma_p^{-2}\mu/N)$
6:     $\mathbf{s} \leftarrow \gamma_2 \mathbf{s} + (1 - \gamma_2)(\mathbf{g} \odot \mathbf{g})$
7:     $\hat{\mathbf{m}} \leftarrow \mathbf{m}/(1 - \gamma_1^t), \quad \hat{\mathbf{s}} \leftarrow \mathbf{s}/(1 - \gamma_2^t)$
8:     $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\, \hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{s}}} + \sigma_p^{-2}/N)$
9:     $t \leftarrow t + 1$
10: end while
```

Throughout the development of the Vadam algorithm, it is considered that the algorithm would already be running. Consequently, the exponential moving average would actually encode information about the geometry of the space. During the initial iterations, however, this estimation would be biased towards the starting point [27]. In order to reduce this effect, the authors [26] introduce a bias-correcting factor that decays exponentially as the optimization runs.

The final method is indeed very similar to Adam [27], but has the advantage of providing uncertainty estimates due to the implicit posterior inference it performs. Apart from being fast, Vadam offers a plug-and-play manner of performing ADF. Differently from the previous methods of this chapter, the user has only to define the model as if it were deterministic and optimize it with Vadam. There is no silver bullet and the price for such easiness and speed is inferior posterior estimates.

4.7 Comparing the Methods

In the remainder of this section, we compare the four algorithms studied during the chapter. We begin with one-dimensional toy examples, for which we can visually analyze the predicted curves and better grasp some of the discussed ideas. Next, we benchmark them on more complex regression tasks, whose results work as better guidelines on possible practical scenarios.

4.7.1 1-D Toy Example

As a first experiment, we evaluate the predictive distribution obtained from the approximation algorithms on toy regression data sets whose targets are given by

$$y = -(x+1)\sin(3\pi x) + \epsilon, \text{ where } \epsilon = \mathcal{N}(0, 0.3^2). \tag{4.63}$$

For this task, we uniformly sample a 20-point and a 400-point sets. We train one-hidden-layer networks with 100 units until convergence using the Adam optimizer (except for the Vadam algorithm, which is an optimizer itself). Figure 4.12 shows the results we obtain.

Let us first recall what was our intention with BNNs: to better model the underlying distribution of our problem and quantify the unknown. Thus, we would like to see our models' uncertainty increase in regions with few to none samples. In that sense, no matter how dense our knowledge about the function may be in the center region of Fig. 4.12, we essentially do not know much outside it, so our uncertainty estimation should not change much in these off-center regions. Conversely, the more samples we have in a given region, the better our accuracy should get and the more certain our model should be in that region.

We can notice that Vadam's uncertainty in the 400-sample scenario remain high whereas the other methods' estimate shrink considerably more in proportion to their 20-sample case. Even BBB, whose predictions look similar to Vadam's, more severely underestimates the variance of the posterior, specially in the 400-point set.

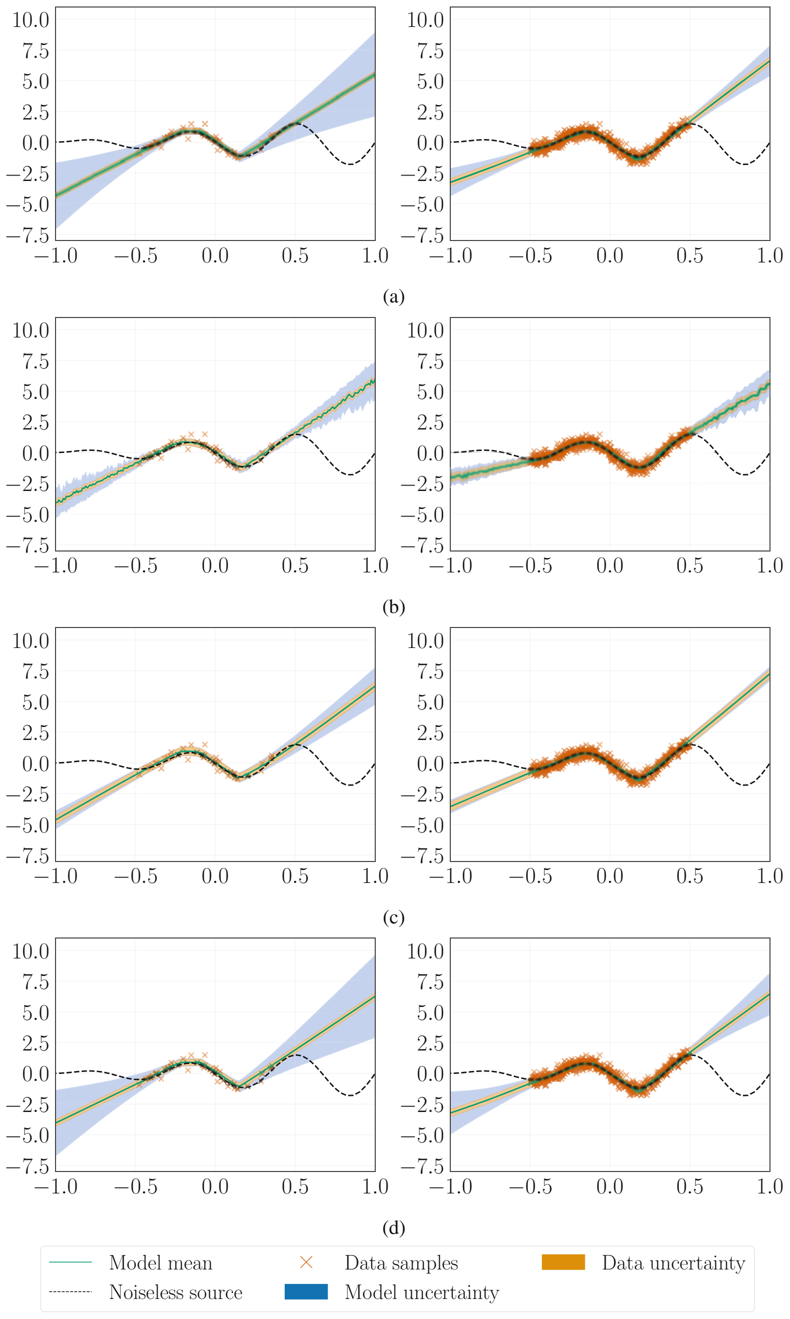

Fig. 4.12 Comparison of the resulting predictive distribution of a one-dimensional toy example with either 40 (left) or 400 (right) data point obtained by (**a**) BBB, (**b**) MCDO, (**c**) PBP, and (**d**) Vadam

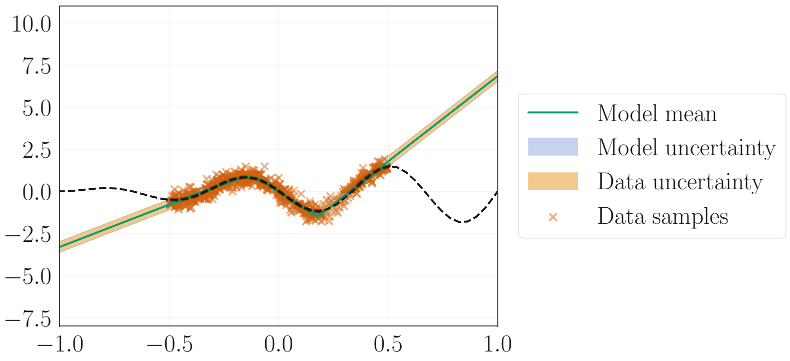

Fig. 4.13 Predictive posterior distribution for the PBP method when trained for 200 epochs. Note that the blue shades (model uncertainty) have disappeared

MCDO's predictions are considerably less smooth than those from other algorithms, needing more MC samples to obtain stable results. Nonetheless, the mean predictions accurately capture the underlying behavior of the target function. Although the algorithm's iterations are individually less computationally expensive, more iterations are required to adequately model the data. Even with a small number of samples, MCDO obtains good estimates for both the mean and the variance.

In larger data sets, where only a reduced number of passes over the data (fewer than 100) are possible, performing EP requires a unreasonably large memory footprint, so PBP actually performs multiple ADF passes through the data, treating each as a novel example. A disadvantage of this approach is that it can lead to underestimation of the posterior variance when too many passes are done over the data. This is the behavior we observe in Fig. 4.13 compared to the one in Fig. 4.12c for the same number of samples.

4.7.2 *UCI Data Sets*

Now, with a better understanding, we benchmark the algorithms in eight different regression data sets of the UCI Machine Learning Repository [10], a procedure that has recently become a standard in the related literature [13, 18, 19, 26, 34, 52, 58]. Below we give a brief description of each data set.

4.7.2.1 Boston Housing

Data about homes from various suburbs in Boston, Massachusetts, collected in 1978 by the US Census Service. The task is to predict the median value of owner-occupied homes from 13 variables measuring property and neighborhood characteristics such as average number of rooms per dwelling, nitric oxides concentration, per capita crime rate, among others. From now on, we refer to it as Boston only.

4.7.2.2 Concrete Compressive Strength

Concrete strength is one of the most important engineering properties of concrete and very important in civil engineering. It is a highly non-linear function of age and ingredients, usually obtained by testing samples under a compression testing machine. The aim of the data set is to predict the compressive strength given the age and seven ingredients, such as cement, water, fine and coarse aggregate, among other component concentrations. From now on, we call it Concrete.

4.7.2.3 Energy Efficiency

During the design of a building, simulations are performed to estimate its energy efficiency. The task is to predict the efficiency, which is expressed by the 2 different metrics heating and cooling load, from eight attributes, such as glazing area, surface area, orientation. The data set is composed of a collection of 768 simulated buildings with different characteristics and 12 different shapes. For brevity, we call this data set Energy.

4.7.2.4 Kin8nm

This data set consists of the angular positions of the joints of an 8-link all-revolute robotic arm, which is known to be highly non-linear. Data was synthetically generated from a simulation of its forward kinematics. The aim is to predict the distance of the end-effector from a given target.

4.7.2.5 Condition Based Maintenance of Naval Propulsion Plants

The behavior and interaction of the main components of propulsion systems cannot be easily modeled with a priori physical knowledge. Still, it is important to continuously monitor the propulsion equipment and take decisions based on their condition. The aim is to predict the compressor decay state coefficient using input features such as ship speed, fuel flow, torques from turbine and propellers, temperatures and pressures coming in and out of the compressor. The data set was generated from a numerical simulator of a Navy frigate characterized by a combined diesel-electric and gas propulsion plant. The simulator was fine-tuned and validated with real-data. In what follows, we refer to this data set as Naval.

4.7.2.6 Combined Cycle Power Plant

The aim of this data set is to predict the net hourly electrical energy output of a power plant. Power output prediction is an important element in managing a plant and its connection to the power grid. The features come from a real plant and were collected for over 6 years, when the plant was set to work with full load. They consist of hourly averages of temperature, ambient pressure, relative humidity, and exhaust vacuum. From this point on, we call this data set Powerplant.

4.7.2.7 Wine Quality

The data consists of 11 physicochemical characteristics of different brands of red and variants of the Portuguese "Vinho Verde" wine. The objective is to predict the quality of the wine, a score between 0 and 10. This data set is hereafter called Wine.

4.7.2.8 Yacht Hydrodynamics

Estimation of the residuary resistance of sailing yachts at the initial design stage is essential for evaluating the ship's performance and for assessing the required propulsive power. The data set contains results from 308 full-scale experiments of 22 different hull forms. The input features are aspects of hull geometry. From now on, we call this data set Yacht.

4.7.3 Experimental Setup

For each data set in Sect. 4.7.2, we compare algorithms according to their training time, predictive (Gaussian) log-likelihood (4.1), and Root Mean Squared Error (RMSE), which is defined as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}}. \tag{4.64}$$

While RMSE exclusively measures the prediction accuracy, thus assessing how close the predictions are to the target values, the log-likelihood takes into account the prediction variance and thus incorporates the prediction uncertainty into the evaluation. Intuitively, the lower the variance, the more reliable the prediction should be and, hence, the higher the penalty for being wrong; but still we want predictions to be reliable so large variances also receive higher penalties. Otherwise, constantly predicting uncertain values would amount to good scores, even though the model would not be of much use.

We do not directly measure structural uncertainty, that is, the uncertainty stemming from the model, which could be corrected with an infinite amount of data. However, highly uncertain weights, i.e., weights with large variances, lead to very different outputs for the same inputs each time we draw a different set of weight values. Consequently those outputs frequently fall far from the true value even if their mean is correct. This causes the estimated predictive log-likelihood, that should be ideally high, to be low. Hence, this metric gives us a sense of the model uncertainty, though indirectly.

We follow the setup proposed in [13]: for each data set we run the models on 20 random train-test splits after doing hyper-parameter search with 30 iterations of Bayesian Optimization (BO) [49] on each split. It is important to note that Bayesian Optimization (BO) has nothing to do with the previously discussed methods for training BNNs, we use it as a tool for efficiently searching the hyper-parameter space. We could have employed random or grid search instead and the arguments developed throughout the chapter would still be the same, as would our pipeline illustrated in Fig. 4.14.

4.7.3.1 Hyper-Parameter Search with Bayesian Optimization (BO)

Bayesian Optimization (BO) is a black-box approach to optimize objective functions that take a long time or are costly to evaluate. Bayesian Optimization (BO) builds a surrogate for the objective and quantifies the uncertainty in that surrogate through Gaussian Process regression [47]. At each iteration, we observe the objective at a new point (a new hyper-parameter configuration), update the posterior distribution that describes the potential objective values at each point, and sample a new point whose values maximize a given acquisition function, i.e., the expected improvement. Bayesian Optimization (BO) factors in all previously seen configurations to decide what point of the parameter space to investigate next, achieving good solutions for complex non-convex functions with considerably fewer iterations. On the other hand, the decision where to evaluate next makes each iteration computationally expensive to run, imposing an overhead.

We use the same 20 data splits[1] for the methods to avoid fluctuations in the results due to the reduced size of the data sets and the effect different splits may have. For each split, we set the optimal hyper-parameter configurations of the prior precision λ (or equivalently the prior variance σ_p^2), the observation noise precision γ, and, in the MC Dropout case, the dropout probability p by running 30 iterations of Bayesian Optimization (BO) on the training set for 40 epochs. Additionally, the

[1] Available at: https://github.com/yaringal/DropoutUncertaintyExps/tree/master/UCI_Datasets.

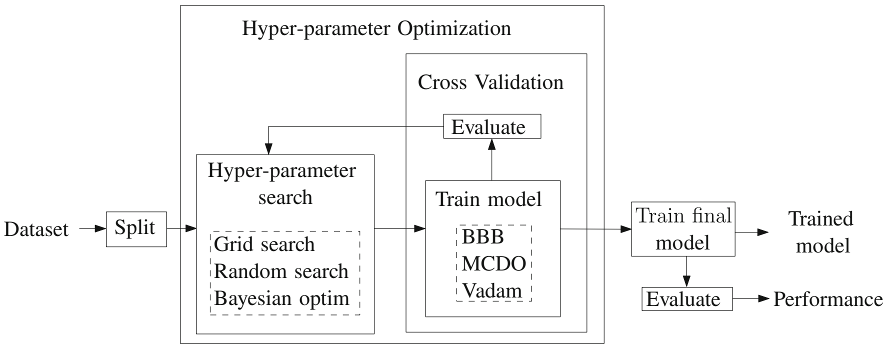

Fig. 4.14 Experimental pipeline for evaluating BNNs. Each data set is split as in [13], then we do 30 rounds of hyper-parameter optimization, training the model on a random subset of the split with one of the discussed methods in a cross-validation setting on each round. At the end, we use the best hyper-parameters found to train the final model on the whole split

performance of the hyper-parameter configuration of each Bayesian Optimization (BO) iteration is averaged over a 5-fold cross-validation, so for each setting we train and evaluate the model 5 times. After finding the best configuration for each split, we fit the model to the whole training set. All this procedure follows from [13]. The general pipeline of the approach is summarized in Fig. 4.14.

We observe that this structure escalates quickly, as for **each** data set and model we have:

$$\underbrace{20}_{\text{splits}} \times \left(\underbrace{30}_{\text{BO iters}} \times \underbrace{\left(5 \times \frac{4}{5}\right)}_{\text{CV iters}} + \underbrace{1}_{\text{full run}} \right) \times \underbrace{40}_{\text{epochs}} = \underbrace{96800}_{\text{epochs}}, \tag{4.65}$$

plus the time each Bayesian Optimization (BO) iteration takes to decide on the next point to test. We do small scale studies with single hidden-layer networks with 50 units to keep the computation under a viable amount of time.

The source codes for PBP[2] and Vadam[3] were borrowed from the authors' repositories. With exception of PBP which is implemented in Theano 1.0 [53], all remaining algorithms and supporting code are in PyTorch 1.0 [44].

[2]https://github.com/HIPS/Probabilistic-Backpropagation.

[3]https://github.com/emtiyaz/vadam.

Table 4.2 Number of MC samples during training for each algorithm. PBP does not employ MC integration, instead it uses analytical approximations

	Smaller sets	Larger sets
BBBx1	10	5
BBBx2	20	10
MCDOx1	1	1
MCDOx10	10	10
Vadam	10	5
PBP	–	–

4.7.4 Training Configuration

We maintain a training configuration similar to [26]. We use a mini-batch of size 32 on the four smaller data sets (Boston, Concrete, Energy, and Yacht), and of 128 on the other four (Wine, Powerplant, Naval, Kin8nm). Table 4.2 contains the number of MC samples used during training for each algorithm, for evaluation they all use 100 MC samples.

We run BBB and MCDO under two different conditions to further investigate their behavior. BBBx2 draws twice the number of MC samples of BBBx1 during training. As for MC Dropout, our MCDOx1 configuration follows the original MC Dropout implementation [13], i.e., just one MC sample during training and longer training time after hyper-parameter selection, namely 400 epochs instead of 40, since MC Dropout takes longer to converge [13]. On the other hand, MCDOx10's training procedure is more similar to the other algorithms' setup: 10 training MC samples and 40 epochs.

In PBP's original implementation [19], which we use, samples are individually processed, but mini-batching is possible at the cost of slightly reduced performance. PBP also has no MC approximation of the weights' posterior since it propagates entire distributions through the layers, analytically performing its approximations.

BBB, MCDO, and Vadam use gradient-descent optimizers. Both BBB and MC Dropout use the Adam optimizer [27], while Vadam is itself a (variational) optimizer and the experiment consists of using it in lieu of Adam. Following [26], in all three methods we set learning rate $k = 0.01$, and moving-average parameters $\gamma_1 = 0.99$ and $\gamma_2 = 0.9$ (instead of the usual $\gamma_1 = 0.9$ and $\gamma_2 = 0.999$) to encourage convergence within 40 epochs. The initial precision for the posterior approximation is set to 10 (attention, this is not the prior precision) for BBB and Vadam.

4.7.5 Analysis

For comparing the algorithms according to the performance in each individual data set, we use the Bayesian Correlated t-test [9]. This test is used for the analysis of cross-validation results and accounts for the correlation due to the overlapping

Table 4.3 Average amount of time in seconds each algorithm takes to complete a whole training cycle, that is, from finding the optimal hyper-parameters to finding the final posterior approximation to the weights

			Absolute avg. running time (s)					
Data set	*Size*	Dim	BBBx1	BBBx2	Vadam	**PBP**	MCDOx1	MCDOx10
Boston	506	13	1813	3279	2214	16	1286	1339
Concrete	1030	8	3510	6101	4333	28	2280	2442
Energy	768	8	2680	4312	3283	20	1541	912
Kin8nm	8192	8	4563	8433	4985	190	4493	4631
Naval	11,934	16	6923	14036	6835	279	6759	6916
Powerplant	9568	4	5349	9993	6117	188	5356	5500
Wine	1599	11	1009	2269	1226	41	1098	1076
Yacht	308	6	1139	1634	1291	10	612	275

training sets [5]. It is thus suited to our case where we have 20 random splits with 90% for training and 10% for testing.

PBP automatically sets all its hyper-parameters by the Bayesian framework thanks to the hyper-priors, thus dispensing with the hyper-parameter search. In this case, the number in (4.65) reduces from 96800 epochs to $20 \times 40 = 800$, that is, one 40-epoch run per random split. Table 4.3 shows the required (wall-clock) time each algorithm takes to complete the full training schedule (including Bayesian Optimization (BO)). Figure 4.15 illustrates a similar information, but depicts the ratio w.r.t. PBP training time for easier visualization.

PBP outspeeds all others being 34 times faster than the runner-up. This difference results from the absence of hyper-parameter tuning, which exempts the method from running the equivalent of $30 \times 4 = 120$ times to find a good hyper-parameter configuration prior to finally fitting to the full training set. On top of that, there is the overhead imposed by the Bayesian optimization inference. Instead of requiring computer time, PBP requires human time to workout all its derivations and approximations. However, if we were not performing hyper-parameter tuning, PBP's advantage would fade away and it would actually be the slowest method on average. There are actually different factors contributing to this:

- PBP's current implementation uses a mini-batch size of 1, and increasing it to 32, the same size as the others, makes the method once again the fastest [4], though not by that large of a margin as before;
- PBP uses the framework Theano, which is no longer officially supported, while the other 3 methods were implemented in Pytorch [44], a more recent and rapidly growing framework powered by Facebook Artificial Intelligence Research and developed by dedicated personnel, hence the operations are better optimized to GPU processing.

BBBx2 is by far the slowest, taking on average 80% more time to train than BBBx1. Although Vadam has a poorer average performance than BBBx2 as seen in the performance bar chars of Figs. 4.16 and 4.17, it trains faster: only 16%

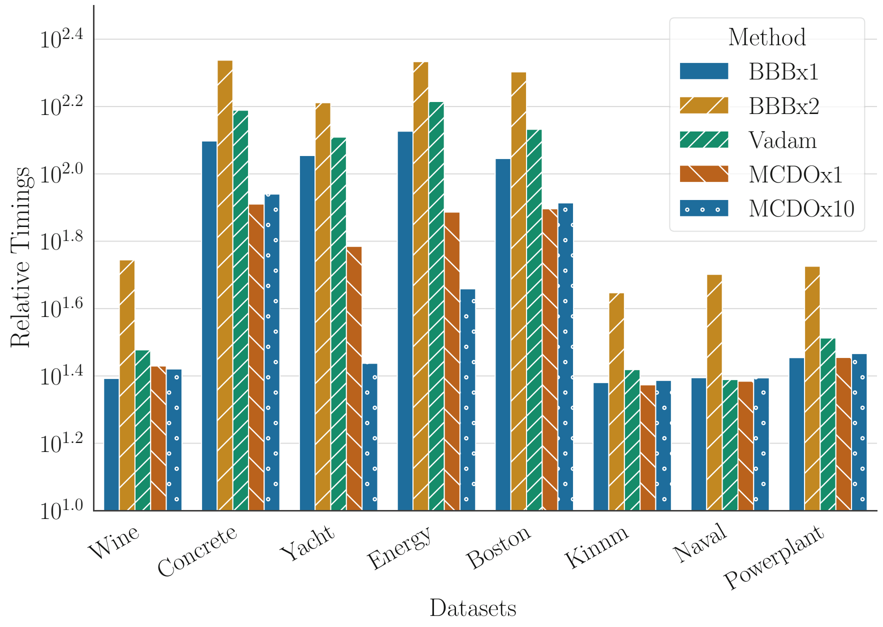

Fig. 4.15 Amount of time in logarithm scale for training each algorithm relative to PBP. We take into account hyper-parameter search and training of the final model

slower than BBBx1, instead of 80%. Vadam does not have additional parameters for the variances of weights; instead it directly computes them from the intermediate variable used to normalize the directions of the parameter space, something the optimizer Adam already does.

Both MCDO configurations take roughly the same amount of time and have similar performance as Figs. 4.15, 4.16, and 4.17 show. MCDOx1 is 10% slower to train and has slightly better performance than MCDOx10. Even though this difference is small, it is consistent. Overall, MCDOx1 is the best w.r.t both RMSE and log-likelihood, with MCDOx10 being a close second. PBP follows them in third place.

Despite its not so stellar performance, PBP has no need for hyper-parameter search and trains incredibly fast. Another strength PBP possesses, inherited from EP, is being naturally well-suited to data-parallelization across machines, and if using only ADF updates, to online learning.

We summarize the conclusions in Table 4.4 to make future reference easier. It rates the BNN algorithms without any number nor formula w.r.t. three fundamental practical aspects:

- The implementation effort to build a custom solution;
- The quality of the model predictions, both accuracy and uncertainty;
- The time it takes to train the model.

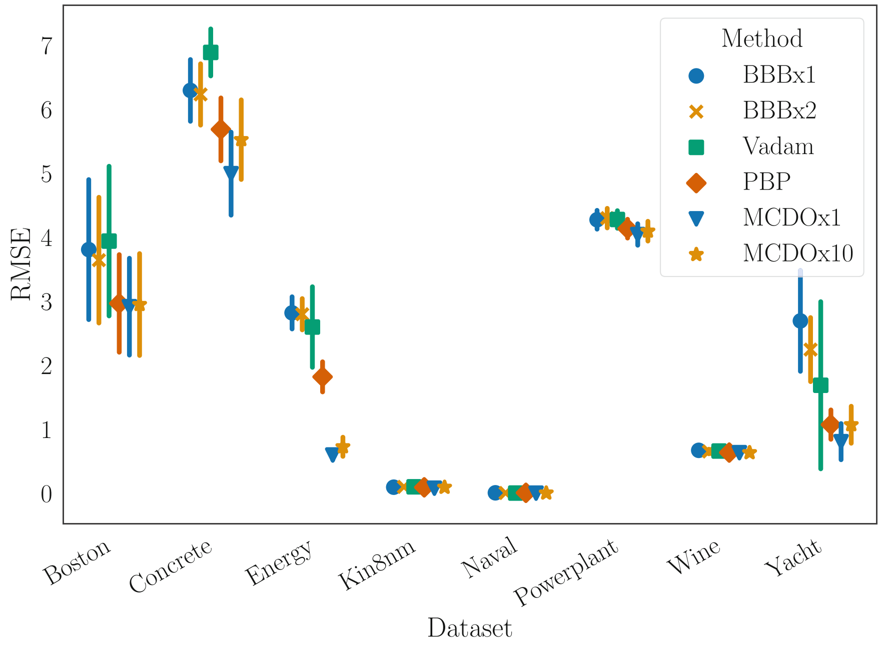

Fig. 4.16 The average RMSE (low values are better) over the 20 random resampled splits of the UCI regression data sets. Error bars represent the standard deviations over the 20 random splits

On a final note, we leave a general recommendation for those needing to develop a custom solution for a certain task: use Vadam [26], it is fast, out-of-the-box and has reasonable performance. It still needs hyper-parameter tuning, but at this point, almost every algorithm does. If the problem calls for better predictive accuracy or uncertainty estimation, resort to MCDO or other methods not covered here, a few of which are mentioned in Sect. 4.9.

4.8 Further References

Even though we treated here only algorithms that do not (explicitly) model the correlation structure between the weights, this also is an active research subject with many interesting works such as:

- Matrix variate Gaussian prior [52] and posterior approximation [34];
- Structured covariance with noisy natural gradient [58];
- Low-rank covariance approximation with natural gradient [37].

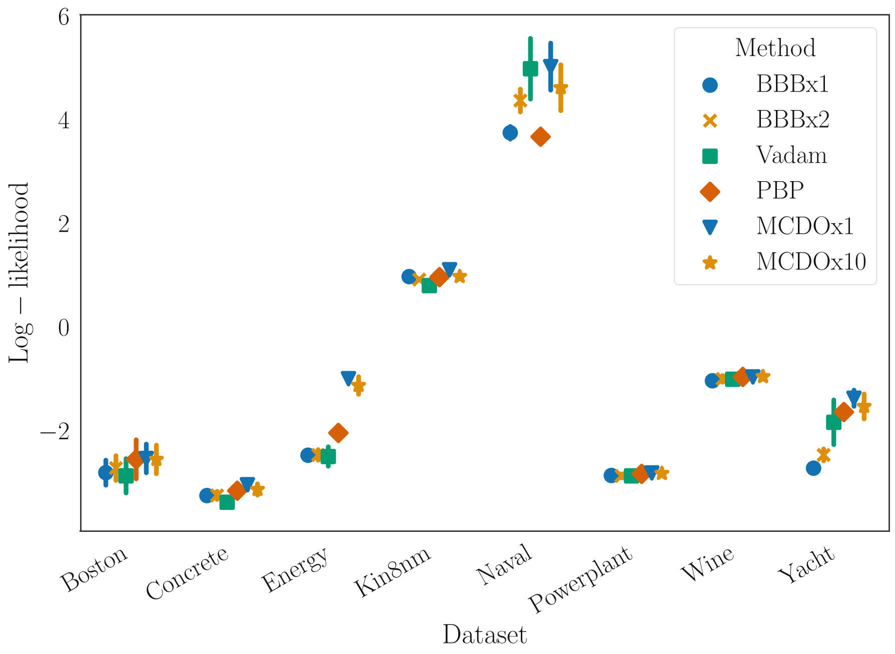

Fig. 4.17 The average log-likelihood (high values are better) over the 20 random resampled splits of the UCI regression data sets. Error bars represent the standard deviations over the 20 random splits

Table 4.4 Practitioner's Table: a rough comparison between the variational methods studied for BNNs. Although BBBx2 performance is better than Vadam's, it takes longer to train and a fairer comparison regarding the time would include BBBx1 instead

Method	Effort	Quality	Timing
BPB	Medium	Poor	Slow
PBP	Very hard	Good	Very fast
MCDO	Very easy	Good	OK
Vadam	None	OK	OK

Although the above methods also rely either on VI, ADF, or EP, by focusing on modeling the structure between the parameters, they achieve better posteriors approximations and uncertainty estimations.

There is a whole other sort of methods that rely on Markov Chain MC approximations to the posterior predictive density, which was not the focus of our discussion. Still, we name a few so that the interested reader knows where to start:

- Hamiltonian MC [41];
- Stochastic gradient Langevin dynamics [32, 57];
- Posterior distribution distillation [29].

4.9 Closing Remarks

In this chapter we have discussed BNNs, along with motivations for recurring to the computationally heavier Bayesian approach instead of contenting ourselves with traditional point estimates. Bayesian models offer a large number of advantages such as robustness to overfitting, principled model comparison, and uncertainty estimation not only in their outputs, but also in all of their parameters.

Additionally, we reviewed and experimentally compared four key variational algorithms throughout the chapter. Namely, Bayes by Backprop [6], Probabilistic Backprop [19], MC Dropout [13], and Vadam [26]. They all consider unstructured approximations to the posterior. Even though MC Dropout [13] is the only one that does not rely on mean-field approximation, it assumes dependency among groups of weights in a rather non-well-defined manner.

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org
2. Amari SI (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276
3. Amari SI (2016) Natural gradient learning and its dynamics in singular regions. Springer Japan, Tokyo, pp 279–314
4. Benatan M, Daresbury ST, Pyzer-Knapp EO (2018) Practical considerations for probabilistic backpropagation. In: Workshop on Bayesian deep learning (NeurIPS 2018), Montreal
5. Benavoli A, Corani G, Demšar J, Zaffalon M (2017) Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. J Mach Learn Res 18(77):1–36
6. Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) Weight uncertainty in neural networks. In: Proceedings of the international conference on machine learning, Lille, vol 37, pp 1613–1622
7. Bonnet G (1964) Transformations des signaux aléatoires a travers les systèmes non linéaires sans mémoire. Annales des Télé Communications 19(9):203–220
8. Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Rev 60(2):223–311. https://doi.org/10.1137/16M1080173
9. Corani G, Benavoli A (2015) A Bayesian approach for comparing cross-validated algorithms on multiple data sets. Mach Learn 100(2-3):285–304
10. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
11. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
12. Gal Y (2016) Uncertainty in deep learning. PhD thesis, University of Cambridge
13. Gal Y, Ghahramani Z (2016a) Dropout as a bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1050–1059
14. Gal Y, Ghahramani Z (2016b) A theoretically grounded application of dropout in recurrent neural networks. In: Advances in neural information processing systems, Barcelona, pp 1019–1027

15. Ghosh S, Fave FD, Yedidia J (2016) Assumed density filtering methods for learning Bayesian neural networks. In: Proceedings of the AAAI conference on artificial intelligence, Phoenix, pp 1589–1595
16. Graves A (2011) Practical variational inference for neural networks. In: Advances in neural information processing systems, Granada, pp 2348–2356
17. Herbrich R (2005) Minimising the Kullback-Leibler divergence. Tech. rep., Microsoft Research
18. Hernandez-Lobato J, Li Y, Rowland M, Bui T, Hernandez-Lobato D, Turner R (2016) Black-box alpha divergence minimization. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1511–1520
19. Hernández-Lobato JM, Adams RP (2015) Probabilistic backpropagation for scalable learning of bayesian neural networks. In: Proceedings of the international conference on machine learning, Lille, vol 37, pp 1861–1869
20. Hinton GE, van Camp D (1993) Keeping the neural networks simple by minimizing the description length of the weights. In: Annual conference on computational learning theory, COLT '93, Santa Cruz, pp 5–13
21. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
22. Hoffman MD, Blei DM, Wang C, Paisley J (2013) Stochastic variational inference. J Mach Learn Res 14:1303–1347
23. Honkela A, Tornio M, Raiko T, Karhunen J (2007) Natural conjugate gradient in variational inference. In: Proceedings of the international conference on neural information processing, Kitakyushu, pp 305–314
24. Hron J, Matthews A, Ghahramani Z (2018) Variational Bayesian dropout: pitfalls and fixes. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2019–2028
25. Khan M, Lin W (2017) Conjugate-computation variational inference :converting variational inference in non-conjugate models to inferences in conjugate models. In: International conference on artificial intelligence and statistics, Fort Lauderdale, vol 54, pp 878–887
26. Khan M, Nielsen D, Tangkaratt V, Lin W, Gal Y, Srivastava A (2018) Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2611–2620
27. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Proceedings of the international conference on learning representations, San Diego
28. Kingma DP, Salimans T, Welling M (2015) Variational dropout and the local reparameterization trick. In: Advances in neural information processing systems, Montreal, pp 2575–2583
29. Korattikara A, Rathod V, Murphy K, Welling M (2015) Bayesian dark knowledge. In: Advances in neural information processing systems, Montreal, pp 3438–3446
30. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, Lake Tahoe, pp 1097–1105
31. Kuleshov V, Fenner N, Ermon S (2018) Accurate uncertainties for deep learning using calibrated regression. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 2796–2804
32. Li C, Chen C, Carlson D, Carin L (2016) Preconditioned stochastic gradient Langevin dynamics for deep neural networks. In: Proceedings of the AAAI conference on artificial intelligence, Phoenix, pp 1788–1794
33. Little RJ (2006) Calibrated Bayes: a Bayes/frequentist roadmap. Am Stat 60(3):213–223
34. Louizos C, Welling M (2016) Structured and efficient variational deep learning with matrix Gaussian posteriors. In: Proceedings of the international conference on machine learning, New York, vol 48, pp 1708–1716
35. MacKay DJC (1992) A practical Bayesian framework for backpropagation networks. Neural Comput 4(3):448–472
36. Minka TP (2001) Expectation propagation for approximate Bayesian inference. In: Conference in uncertainty in artificial intelligence, San Francisco, pp 362–369

37. Mishkin A, Kunstner F, Nielsen D, Schmidt M, Khan ME (2018) SLANG: fast structured covariance approximations for Bayesian deep learning with natural gradient. In: Advances in neural information processing systems, Montreal, pp 6245–6255
38. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the international conference on machine learning, Haifa, pp 807–814
39. Neal RM (1993) Bayesian learning via stochastic dynamics. In: Advances in neural information processing systems, San Francisco, pp 475–482
40. Neal RM (1996) Bayesian learning for neural networks. PhD thesis, University of Toronto, Toronto
41. Neal RM (2011) MCMC using Hamiltonian dynamics. In: Handbook of Markov chain, chap 5. Monte Carlo, CRC Press, Boca Raton, pp 113–162
42. Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of the international conference on machine learning, Bonn, pp 625–632
43. Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A (2017) Practical black-box attacks against machine learning. In: Proceedings of the ACM Asia conference on computer and communications security, Abu Dhabi, pp 506–519
44. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, Vancouver, pp 8024–8035
45. Platt JC (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: Advances in large margin classifiers, pp 61–74
46. Price R (1958) A useful theorem for nonlinear devices having Gaussian inputs. Trans Inf Theor 4(2):69–72
47. Rasmussen CE, Williams CKI (2005) Gaussian processes for machine learning. The MIT Press, Cambridge
48. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
49. Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, Lake Tahoe, pp 2951–2959
50. Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in neural information processing systems, Montreal, pp 963–971
51. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
52. Sun S, Chen C, Carin L (2017) Learning structured weight uncertainty in Bayesian neural networks. In: International conference on artificial intelligence and statistics, Fort Lauderdale, vol 54, pp 1283–1292
53. Theano Development Team (2016) Theano: a python framework for fast computation of mathematical expressions. arXiv e-prints 1605.02688
54. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running aof its recent magnitude
55. Tishby N, Levin E, Solla SA (1989) Consistent inference of probabilities in layered networks: predictions and generalization. In: International joint conference on neural networks, vol 2, pp 403–409
56. Wan L, Zeiler M, Zhang S, Cun YL, Fergus R (2013) Regularization of neural networks using dropconnect. In: Proceedings of the international conference on machine learning, Atlanta, vol 28, pp 1058–1066
57. Welling M, Teh YW (2011) Bayesian learning via stochastic gradient Langevin dynamics. In: Proceedings of the international conference on machine learning, Bellevue, pp 681–688
58. Zhang G, Sun S, Duvenaud D, Grosse R (2018) Noisy natural gradient as variational inference. In: Proceedings of the international conference on machine learning, Stockholm, vol 80, pp 5852–5861