Bayesian Uncertainty Estimation for Batch Normalized Deep Networks

Mattias Teye [1 2 *] Hossein Azizpour [1 *] Kevin Smith [1 3]

Abstract

We show that training a deep network using batch normalization is equivalent to approximate inference in Bayesian models. We further demonstrate that this finding allows us to make meaningful estimates of the model uncertainty using conventional architectures, without modifications to the network or the training procedure. Our approach is thoroughly validated by measuring the quality of uncertainty in a series of empirical experiments on different tasks. It outperforms baselines with strong statistical significance, and displays competitive performance with recent Bayesian approaches.

1. Introduction

Deep learning has dramatically advanced the state of the art in a number of domains. Despite their unprecedented discriminative power, deep networks are prone to make mistakes. Nevertheless, they can already be found in settings where errors carry serious repercussions such as autonomous vehicles (Chen et al., 2016) and high frequency trading. We can soon expect automated systems to screen for various types of cancer (Esteva et al., 2017; Shen, 2017) and diagnose biopsies (Djuric et al., 2017). As autonomous systems based on deep learning are increasingly deployed in settings with the potential to cause physical or economic harm, we need to develop a better understanding of when we can be confident in the estimates produced by deep networks, and when we should be less certain.

Standard deep learning techniques used for supervised learning lack methods to account for uncertainty in the model. This can be problematic when the network encounters conditions it was not exposed to during training, or if the network is confronted with adversarial examples (Goodfellow et al., 2014). When exposed to data outside the distribution it was trained on, the network is forced to extrapolate, which can lead to unpredictable behavior.

If the network can provide information about its uncertainty in addition to its point estimate, disaster may be avoided. In this work, we focus on estimating such predictive uncertainties in deep networks (Figure 1).

The Bayesian approach provides a theoretical framework for modeling uncertainty (Ghahramani, 2015), which has prompted several attempts to extend neural networks (NN) into a Bayesian setting. Most notably, Bayesian neural networks (BNNs) have been studied since the 1990's (Neal, 2012), but do not scale well and struggle to compete with modern deep learning architectures. Recently, (Gal & Ghahramani, 2015) developed a practical solution to obtain uncertainty estimates by casting *dropout* training in conventional deep networks as a Bayesian approximation of a Gaussian Process (its correspondence to a general approximate Bayesian model was shown in (Gal, 2016)). They showed that *any network* trained with dropout is an approximate Bayesian model, and uncertainty estimates can be obtained by computing the variance on multiple predictions with different dropout masks.

The inference in this technique, called *Monte Carlo Dropout* (MCDO), has an attractive quality: it can be applied to any pre-trained networks with dropout layers. Uncertainty estimates come (nearly) for free. However, not all architectures use dropout, and most modern networks have adopted other regularization techniques. *Batch normalization* (BN), in particular, has become widespread thanks to its ability to stabilize learning with improved generalization (Ioffe & Szegedy, 2015).

An interesting aspect of BN is that the mini-batch statistics used for training each iteration depend on randomly selected batch members. We exploit this stochasticity and show that training using batch normalization, like dropout, can be cast as an approximate Bayesian inference. We demonstrate how this finding allows us to make meaningful estimates of the model uncertainty in a technique we call *Monte Carlo Batch Normalization* (MCBN) (Figure 1). The method we propose can be applied to any network using standard batch normalization.

[*] Co-first authorship [1]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden [2]Current address: Electronic Arts, SEED, Stockholm, Sweden. This work was carried out at Budbee AB. [3]Science for Life Laboratory. Correspondence to: Kevin Smith <ksmith@kth.se>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

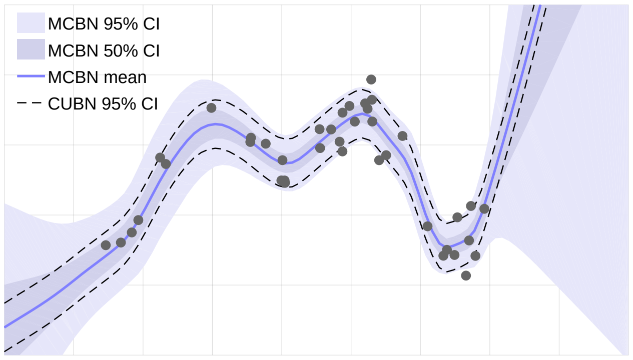

Figure 1. **Training a deep network using batch normalization is equivalent to approximate inference in Bayesian models.** Thus, uncertainty estimates can be obtained from any network using BN through a simple procedure. At inference, several mini-batches are constructed by taking random samples to accompany the query. The mean and variance of the outputs are used to estimate the predictive distribution (MCBN). Here, we show results on a toy dataset from a network with three hidden layers (30 units per layer). Training data is depicted as dots. The solid line is the predictive mean of 500 stochastic forward passes and the shaded areas represent the model's uncertainty. The dashed lines depict a minimal baseline for uncertainty (CUBN), see Section 4.1.

We validate our approach by empirical experiments on a variety of datasets and tasks, including regression and image classification. We measure uncertainty quality relative to a baseline of fixed uncertainty, and show that MCBN outperforms the baseline on nearly all datasets with strong statistical significance. We also show that the uncertainty quality of MCBN is on par with other recent approximate Bayesian networks.

2. Related Work

Bayesian models provide a natural framework for modeling uncertainty, and several approaches have been developed to adapt NNs to Bayesian reasoning. A common approach is to place a prior distribution (often a Gaussian) over each parameter. The resulting model corresponds to a Gaussian process for infinite parameters (Neal, 1995), and a Bayesian NN (MacKay, 1992) for a finite number of parameters. Inference in BNNs is difficult however (Gal, 2016), so focus has thus shifted to techniques that approximate the posterior, *approximate BNNs*. Methods based on variational inference (VI) typically rely on a fully factorized approximate distribution (Kingma & Welling, 2014; Hinton & Van Camp, 1993), but often do not scale. To alleviate these difficulties, (Graves, 2011) proposed a model using sampling methods to estimate a factorized posterior. Probabilistic backpropagation (PBP), estimates a factorized posterior via expectation propagation (Hernández-Lobato & Adams, 2015).

Using several strategies to address scaling issues, Deep Gaussian Processes show superior performance in terms of RMSE and uncertainty quality compared to state-of-the-art approximate BNNs (Bui et al., 2016)[1]. Another recent approach to Bayesian learning, Bayesian hypernetworks, use a NN to learn a distribution of parameters over another network (Krueger et al., 2017). Multiplicative Normalizing Flows for variational Bayesian networks (MNF) (Louizos & Welling, 2017) is a recent model that formulates a posterior dependent on auxiliary variables. MNF achieves a highly flexible posterior by the application of normalizing flows to the auxiliary variables.

Although these recent techniques address some of the difficulties with approximate BNNs, they all require modifications to the architecture or the way networks are trained, as well as specialized knowledge from practitioners. Recently, (Gal & Ghahramani, 2015) showed that a network trained with dropout implicitly performs the VI objective. Therefore *any* network trained with dropout can be treated as an approximate Bayesian model by making multiple predictions through the network while sampling different dropout masks for each prediction. The mean and variance of the predictions are used in the estimation of the mean and variance of the predictive distribution [2].

3. Method

In the following, we introduce Bayesian models and a variational approximation using Kullback-Leibler (KL) divergence following (Gal, 2016). We continue by showing that a batch normalized deep network can be seen as an approximate Bayesian model. Employing theoretical insights and empirical analysis, we study the induced prior on the parameters when using batch normalization. Finally, we describe the procedure for estimating the uncertainty of a batch normalized network's output.[3]

3.1. Bayesian Modeling

We assume a finite training set $\mathbf{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1:N}$ where each $(\mathbf{x}_i, \mathbf{y}_i)$ is a sample-label pair. Using $\mathbf{D}$, we are interested in learning an inference function $f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y})$ with parameters $\boldsymbol{\omega}$. In deterministic models, the estimated label $\hat{\mathbf{y}}$ is obtained as follows:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y})$$

In probabilistic models we let $f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})$. In Bayesian modeling, in contrast to finding a point estimate

[1]By uncertainty quality, we refer to predictive probability distributions as measured by PLL and CRPS.

[2]This technique is referred to as "MC Dropout" in the original work, though we refer to it here as MCDO.

[3]While the method applies to FC or Conv layers, the induced prior from weight decay (Section 3.3) is studied for FC layers.

of the model parameters, the idea is to estimate an (approximate) posterior distribution of the model parameters $p(\boldsymbol{\omega}|\mathbf{D})$ to be used for probabilistic prediction:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{D}) = \int f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) p(\boldsymbol{\omega}|\mathbf{D}) d\boldsymbol{\omega}$$

The predicted label, $\hat{\mathbf{y}}$, can then be accordingly obtained by sampling $p(\mathbf{y}|\mathbf{x}, \mathbf{D})$ or taking its maxima.

Variational Approximation In approximate Bayesian modeling, a common approach is to learn a parameterized approximating distribution $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ that minimizes $\mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\mathbf{D}))$; the Kullback-Leibler divergence of the true posterior w.r.t. its approximation. Minimizing this KL divergence is equivalent to the following minimization while being free of the data term $p(\mathbf{D})$ [4]:

$$\mathcal{L}_{\text{VA}}(\boldsymbol{\theta}) := -\sum_{i=1}^{N} \int q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \ln f_{\boldsymbol{\omega}}(\mathbf{x}_i, \mathbf{y}_i) \mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

During optimization, we want to take the derivative of the expected likelihood w.r.t. the learnable parameters $\boldsymbol{\theta}$. We use the same MC estimate as in (Gal, 2016) (explained in Appendix Section 1.1), such that one realized $\hat{\boldsymbol{\omega}}_i$ is taken for each sample i [5]. Optimizing over mini-batches of size M, the approximated objective becomes:

$$\hat{\mathcal{L}}_{\text{VA}}(\boldsymbol{\theta}) := -\frac{N}{M} \sum_{i=1}^{M} \ln f_{\hat{\boldsymbol{\omega}}_i}(\mathbf{x}_i, \mathbf{y}_i) + \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \quad (1)$$

The first term is the data likelihood and the second term is the divergence of the prior w.r.t. the approximated posterior.

3.2. Batch Normalized Deep Nets as Bayesian Modeling

We now describe the optimization procedure of a deep network with batch normalization and draw the resemblance to the approximate Bayesian modeling in Eq (1).

The inference function of a feed-forward deep network with L layers can be described as:

$$f_{\boldsymbol{\omega}}(\mathbf{x}) = \mathbf{W}^{L} a(\mathbf{W}^{L-1} ... a(\mathbf{W}^{2} a(\mathbf{W}^{1}\mathbf{x}))$$

where $a(.)$ is an element-wise nonlinearity function and $\mathbf{W}^l$ is the weight vector at layer l. Furthermore, we denote the input to layer l as $\mathbf{x}^l$ with $\mathbf{x}^1 = \mathbf{x}$ and we then set $\mathbf{h}^l = \mathbf{W}^l \mathbf{x}^l$. Parenthesized super-index for matrices (e.g. $\mathbf{W}^{(j)}$) and vectors (e.g. $x^{(j)}$) indicates jth row and element respectively. Super-index u refers to a specific unit at layer l, (e.g. $\mathbf{W}^u = \mathbf{W}^{l,(j)}, h^u = h^{l,(j)}$). [6]

Batch Normalization Each layer of a deep network is constructed by several linear units whose parameters are the rows of the weight matrix $\mathbf{W}$. Batch normalization is a unit-wise operation proposed in (Ioffe & Szegedy, 2015) to standardize the distribution of each unit's input. For FC layers, it converts a unit's input h^u in the following way:

$$\hat{h}^u = \frac{h^u - \mathbb{E}[h^u]}{\sqrt{\mathrm{Var}[h^u]}}$$

where the expectations are computed over the training set during evaluation, and mini-batch during training (in deep networks, the weight matrices are often optimized using back-propagated errors calculated on mini-batches of data)[7]. Therefore, during training, the estimated mean and variance on the mini-batch $\mathbf{B}$ is used, which we denote by $\boldsymbol{\mu}_{\mathbf{B}}$ and $\boldsymbol{\sigma}_{\mathbf{B}}$ respectively. This makes the inference at training time for a sample $\mathbf{x}$ a stochastic process, varying based on other samples in the mini-batch.

Loss Function and Optimization Training deep networks with mini-batch optimization involves a (regularized) risk minimization with the following form:

$$\mathcal{L}_{\text{RR}}(\boldsymbol{\omega}) := \frac{1}{M} \sum_{i=1}^{M} l(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \Omega(\boldsymbol{\omega})$$

where the first term is the empirical loss on the training data and the second term is a regularization penalty acting as a prior on model parameters $\boldsymbol{\omega}$. If the loss l is cross-entropy for classification or sum-of-squares for regression problems (assuming i.i.d. Gaussian noise on labels), the first term is equivalent to minimizing the negative log-likelihood:

$$\mathcal{L}_{\text{RR}}(\boldsymbol{\omega}) := -\frac{1}{M\tau} \sum_{i=1}^{M} \ln f_{\boldsymbol{\omega}}(\mathbf{x}_i, \mathbf{y}_i) + \Omega(\boldsymbol{\omega})$$

[4] Achieved by constructing the Evidence Lower Bound, called ELBO, and assuming i.i.d. observation noise; details can be found in Appendix Section 1.1.

[5] While a MC integration using a single sample is a weak approximation, in an iterative optimization for $\boldsymbol{\theta}$ several samples will be taken over time.

[6] For a (softmax) classification network, $f_{\boldsymbol{\omega}}(\mathbf{x})$ is a vector with $f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) = f_{\boldsymbol{\omega}}(\mathbf{x})^{(\mathbf{y})}$, for regression networks with i.i.d. Gaussian noise we have $f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) = \mathcal{N}(f_{\boldsymbol{\omega}}(\mathbf{x}), \tau^{-1}\mathbf{I})$.

[7] It also learns an affine transformation for each unit with parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, omitted for brevity: $\hat{x}^{(j)}_{\text{affine}} = \gamma^{(j)} \hat{x}^{(j)} + \beta^{(j)}$.

with $\tau = 1$ for classification. In a network with batch normalization, the model parameters include $\{\mathbf{W}^{1:L}, \gamma^{1:L}, \boldsymbol{\beta}^{1:L}, \boldsymbol{\mu}_{\mathbf{B}}^{1:L}, \boldsymbol{\sigma}_{\mathbf{B}}^{1:L}\}$. If we decouple the learnable parameters $\boldsymbol{\theta} = \{\mathbf{W}^{1:L}, \gamma^{1:L}, \boldsymbol{\beta}^{1:L}\}$ from the stochastic parameters $\boldsymbol{\omega} = \{\boldsymbol{\mu}_{\mathbf{B}}^{1:L}, \boldsymbol{\sigma}_{\mathbf{B}}^{1:L}\}$, we get the following objective at each step of the mini-batch optimization:

$$\mathcal{L}_{\text{RR}}(\boldsymbol{\theta}) := -\frac{1}{M\tau}\sum_{i=1}^{M} \ln f_{\{\boldsymbol{\theta}, \hat{\boldsymbol{\omega}}_i\}}(\mathbf{x}_i, \mathbf{y}_i) + \Omega(\boldsymbol{\theta}) \qquad (2)$$

where $\hat{\boldsymbol{\omega}}_i$ is the means and variances for sample i's mini-batch at a certain training step. Note that while $\hat{\boldsymbol{\omega}}_i$ formally needs to be i.i.d. for each training example, a batch normalized network samples the stochastic parameters once per training step (mini-batch). For a large number of epochs, however, the distribution of sampled batch members for a given training example converges to the i.i.d. case.

In a batch normalized network, $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ corresponds to the joint distribution of the weights, induced by the randomness of the normalization parameters $\boldsymbol{\mu}_{\mathbf{B}}^{1:L}, \boldsymbol{\sigma}_{\mathbf{B}}^{1:L}$, as implied by the repeated sampling from $\mathbf{D}$ during training. This is an approximation of the true posterior, where we have restricted the posterior to lie within the domain of our parametric network and source of randomness. With that, *we can estimate the uncertainty of predictions from a trained batch normalized network using the inherent stochasticity of BN* (Section 3.4).

3.3. Prior $p(\boldsymbol{\omega})$

Equivalence between the VA and BN training procedures requires $\frac{\partial}{\partial\theta}$ of Eq. (1) and Eq. (2) to be equivalent up to a scaling factor. This is the case if $\frac{\partial}{\partial\theta}\text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = N\tau\frac{\partial}{\partial\theta}\Omega(\boldsymbol{\theta})$.

To reconcile this condition, one option is to let the prior $p(\boldsymbol{\omega})$ imply the regularization term $\Omega(\boldsymbol{\theta})$. Eq. (1) reveals that the contribution of $\text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$ to the optimization objective is inversely scaled with N. For BN, this corresponds to a model with a small $\Omega(\boldsymbol{\theta})$ when N is large. In the limit as $N \to \infty$, the optimization objectives of Eq. (1) and Eq. (2) become identical with no regularization.[8]

Another option is to let some $\Omega(\boldsymbol{\theta})$ imply $p(\boldsymbol{\omega})$. In Appendix Section 1.4 we explore this with L2-regularization, also called weight decay ($\Omega(\boldsymbol{\theta}) = \lambda\sum_{l=1:L}||W^l||^2$). We find that unlike in MCDO (Gal, 2016), some simplifying assumptions are necessary to reconcile the VA and BN objectives with weight decay: no scale and shift applied to BN layers, uncorrelated units in each layer, BN applied on all layers, and large N and M. Given these conditions:

$$p(\mu_{\mathbf{B}}^{u}) = \mathcal{N}(\mu_{\mu,p}, \sigma_{\mu,p})$$
$$p(\sigma_{\mathbf{B}}^{u}) = \mathcal{N}(\mu_{\sigma,p}, \sigma_{\sigma,p})$$

where $\mu_{\mu,p} = 0$, $\sigma_{\mu,p} \to \infty$, $\mu_{\sigma,p} = 0$ and $\sigma_{\sigma,p} \to \frac{1}{2N\tau\lambda_l}$.

This corresponds to a wide and narrow distribution on BN units' means and std. devs respectively, where N accounts for the narrowness of the prior. Due to its popularity in deep learning, our experiments in Section 4 are performed with weight decay.

3.4. Predictive Uncertainty in Batch Normalized Deep Nets

In the absence of the true posterior, we rely on the approximate posterior to express an approximate predictive distribution:

$$p^*(\mathbf{y}|\mathbf{x}, \mathbf{D}) := \int f_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) q_{\boldsymbol{\theta}}(\boldsymbol{\omega}) d\boldsymbol{\omega}$$

Following (Gal, 2016) we estimate the first (for regression and classification) and second (for regression) moments of the predictive distribution empirically (see Appendix Section 1.5 for details):

$$\mathbb{E}_{p^*}[\mathbf{y}] \approx \frac{1}{T}\sum_{i=1}^{T} f_{\hat{\boldsymbol{\omega}}_i}(\mathbf{x})$$

$$\text{Cov}_{p^*}[\mathbf{y}] \approx \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{i=1}^{T} f_{\hat{\boldsymbol{\omega}}_i}(\mathbf{x})^{\intercal} f_{\hat{\boldsymbol{\omega}}_i}(\mathbf{x}) - \mathbb{E}_{p^*}[\mathbf{y}]^{\intercal}\mathbb{E}_{p^*}[\mathbf{y}]$$

where each $\hat{\boldsymbol{\omega}}_i$ corresponds to sampling the net's stochastic parameters $\boldsymbol{\omega} = \{\boldsymbol{\mu}_{\mathbf{B}}^{1:L}, \boldsymbol{\sigma}_{\mathbf{B}}^{1:L}\}$ the same way as during training. Sampling $\hat{\boldsymbol{\omega}}_i$ therefore involves sampling a batch $\mathbf{B}$ from the *training set* and updating the parameters in the BN units, just as if we were taking a training step with $\mathbf{B}$. From a VA perspective, training the network amounted to minimizing $\text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\mathbf{D}))$ wrt $\boldsymbol{\theta}$. Sampling $\hat{\boldsymbol{\omega}}_i$ from the training set, and keeping the size of $\mathbf{B}$ consistent with the mini-batch size used during training, ensures that $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ during inference remains identical to the approximate posterior optimized during training.

The network is trained just as a regular BN network, but instead of replacing $\boldsymbol{\omega} = \{\boldsymbol{\mu}_{\mathbf{B}}^{1:L}, \boldsymbol{\sigma}_{\mathbf{B}}^{1:L}\}$ with population values from $\mathbf{D}$ for inference, we update these parameters stochastically, once for each forward pass.[9] Pseudocode for estimating predictive mean and variance is given in Algorithm 1.

[8]To prove the existence and find an expression of $\text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$, in Appendix Section 1.3 we find that BN approximately induces Gaussian distributions over BN units' means and standard deviations, centered around the population values given by $\mathbf{D}$. We assume a factorized distribution and Gaussian priors, and find the corresponding $\text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$ components in Appendix Section 1.4 Eq. (7). These could be used to construct a custom $\Omega(\boldsymbol{\theta})$ for any Gaussian choice of $p(\boldsymbol{\omega})$.

[9]As an alternative to using the training set $\mathbf{D}$ to sample $\hat{\boldsymbol{\omega}}_i$,

Algorithm 1 MCBN Algorithm

```
Input: sample $x$, number of inferences $T$, batchsize $b$
Output: mean prediction $\hat{y}$, predictive uncertainty $\sigma^2$
1: $\mathbf{y} = \{\}$
2: loop for $T$ iterations
3:     $B \sim D$ // mini batch
4:     $\hat{\boldsymbol{\omega}} = \{\boldsymbol{\mu}_B, \boldsymbol{\sigma}_B\}$    // mini batch mean and variance
5:     $\mathbf{y} = \mathbf{y} \cup f_{\hat{\boldsymbol{\omega}}}(x)$
6: end loop
7: $\hat{y} = \mathbb{E}[\mathbf{y}]$
8: $\sigma^2 = \text{Cov}[\mathbf{y}] + \tau^{-1}\mathbf{I}$ // for regression
```

4. Experiments and Results

We assess the uncertainty quality of MCBN quantitatively and qualitatively. Our quantitative analysis relies on CIFAR10 for image classification and eight standard regression datasets, listed in Appendix Table 1. Publicly available from the UCI Machine Learning Repository (University of California, 2017) and Delve (Ghahramani, 1996), these datasets have been used to benchmark comparative models in recent related literature (see (Hernández-Lobato & Adams, 2015), (Gal & Ghahramani, 2015), (Bui et al., 2016) and (Li & Gal, 2017)). We report results using standard metrics, and also propose useful upper and lower bounds to normalize these metrics for an easier interpretation in Section 4.2.

Our qualitative results include the toy dataset in Figure 1 in the style of (Karpathy, 2015), *a new visualization of uncertainty quality* that plots test errors sorted by predicted variance (Figure 2 and Appendix), and image segmentation results (Figure 2 and Appendix).

4.1. Metrics

We evaluate uncertainty quality based on two standard metrics, described below: Predictive Log Likelihood (PLL) and Continuous Ranked Probability Score (CRPS). To improve the interpretability of the metrics, we propose to normalize them by upper and lower bounds.

Predictive Log Likelihood (PLL) Predictive Log Likelihood is widely accepted as the main uncertainty quality metric for regression (Hernández-Lobato & Adams, 2015; Gal & Ghahramani, 2015; Bui et al., 2016; Li & Gal, 2017). A key property of PLL is that it makes no assumptions about the form of the distribution. The measure is defined for a probabilistic model $f_{\boldsymbol{\omega}}(\mathbf{x})$ and a single observation $(\mathbf{y}_i, \mathbf{x}_i)$ as:

$$\text{PLL}(f_{\boldsymbol{\omega}}(\mathbf{x}), (\mathbf{y}_i, \mathbf{x}_i)) = \log p(\mathbf{y}_i | f_{\boldsymbol{\omega}}(\mathbf{x}_i))$$

where $p(\mathbf{y}_i | f_{\boldsymbol{\omega}}(\mathbf{x}_i))$ is the model's predicted PDF evaluated at $\mathbf{y}_i$, given the input x_i. A more detailed description is given in the Appendix Section 1.5. The metric is unbounded and maximized by a perfect prediction (mode at $\mathbf{y}_i$) with no variance. As the predictive mode moves away from $\mathbf{y}_i$, increasing the variance tends to increase PLL (by maximizing probability mass at $\mathbf{y}_i$). While PLL is an elegant measure, it has been criticized for allowing outliers to have an overly negative effect on the score (Selten, 1998).

Continuous Ranked Probability Score (CRPS) Continuous Ranked Probability Score is a measure that takes the full predicted PDF into account with less sensitivity to outliers. A prediction with low variance that is slightly offset from the true observation will receive a higher score form CRPS than PLL. In order for CRPS to be analytically tractable, we need to assume a Gaussian unimodal predictive distribution. CRPS is defined as

$$\text{CRPS}(f_{\boldsymbol{\omega}}(x_i), (y_i, x_i)) = \int_{-\infty}^{\infty} \left(F(y) - \mathbf{1}(y \geq y_i)\right)^2 \mathrm{d}y$$

where $F(y)$ is the predictive CDF, and $\mathbf{1}(y \geq y_i) = 1$ if $y \geq y_i$ and 0 otherwise (for univariate distributions) (Gneiting & Raftery, 2007). CRPS is interpreted as the sum of the squared area between the CDF and 0 where $y < y_i$ and between the CDF and 1 where $y \geq y_i$. A perfect prediction with no variance yields a CRPS of 0; for all other cases the value is larger. CRPS has no upper bound.

4.2. Benchmark models and normalized metrics

It is difficult to interpret the quality of uncertainty from raw PLL and CRPS values. We propose to normalize the metrics between useful lower and upper bounds. The normalized measures estimate the performance of an uncertainty model between the trivial solution (constant uncertainty) and *optimal uncertainty* for each prediction. For the lower bound, we define a baseline that predicts constant variance regardless of input. The variance is set to a fixed value that optimizes CRPS on validation data. We call this model Constant Uncertainty BN (CUBN). It reflects our best guess of constant variance on test data – thus, any improvement in uncertainty quality over CUBN indicates a sensible estimate of uncertainty. We similarly define a baseline for dropout, Constant Uncertainty Dropout (CUDO). The modeling of variance (uncertainty) by MCBN and CUBN are visualized in Figure 1.

An upper bound on uncertainty performance can also be defined for a probabilistic model f with respect to CRPS or PLL. For each observation (y_i, x_i), a value

we could sample from the implied $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ as modeled in the Appendix. This would alleviate having to store $\mathbf{D}$ for use during prediction. In our experiments we used $\mathbf{D}$ to sample $\hat{\boldsymbol{\omega}}_i$ however, and leave the evaluation of the modeled $q_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ for future research.

for the predictive variance T_i can be chosen that maximizes PLL or minimizes CRPS[10]. Using CUBN as a lower bound and the optimized CRPS score as the upper bound, uncertainty estimates can be normalized between these bounds (1 indicating optimal performance, and 0 indicating same performance as fixed uncertainty). We call this normalized measure $\overline{\text{CRPS}} = \frac{\text{CRPS}(f,(y_i,x_i))-\text{CRPS}(f_{CU},(y_i,x_i))}{\min_T \text{CRPS}(f,(y_i,x_i))-\text{CRPS}(f_{CU},(y_i,x_i))} \times 100$, and the PLL analogue $\overline{\text{PLL}} = \frac{\text{PLL}(f,(y_i,x_i))-\text{PLL}(f_{CU},(y_i,x_i))}{\max_T \text{PLL}(f,(y_i,x_i))-\text{PLL}(f_{CU},(y_i,x_i))} \times 100$.

4.3. Test setup

Our evaluation compares MCBN to MCDO (Gal & Ghahramani, 2015) and MNF (Louizos & Welling, 2017) using the datasets and metrics described above. Our setup is similar to (Hernández-Lobato & Adams, 2015), which was also followed by (Gal & Ghahramani, 2015). However, our comparison implements a different hyperparameter selection, allows for a larger range of dropout rates, and uses larger networks with two hidden layers.

For the regression task, all models share a similar architecture: two hidden layers with 50 units each, and ReLU activations, with the exception of Protein Tertiary Structure dataset (100 units per hidden layer). Inputs and outputs were normalized during training. Results were averaged over five random splits of 20% test and 80% training and cross-validation (CV) data. For each split, 5-fold CV by grid search with a RMSE minimization objective was used to find training hyperparameters and optimal n.o. epochs, out of a maximum of 2000. For BN-based models, the hyperparameter grid consisted of a weight decay factor ranging from 0.1 to 1^{-15} by a $\log 10$ scale, and a batch size range from 32 to 1024 by a $\log 2$ scale. For DO-based models, the hyperparameter grid consisted of the same weight decay range, and dropout probabilities in $\{0.2, 0.1, 0.05, 0.01, 0.005, 0.001\}$. DO-based models used a batch size of 32 in all evaluations. For MNF[11], the n.o. epochs was optimized, the batch size was set to 100, and early stopping test performed each epoch (compared to every 20th for MCBN, MCDO).

For MCBN and MCDO, the model with optimal training hyperparameters was used to optimize τ numerically. This optimization was made in terms of average CV CRPS for MCBN, CUBN, MCDO, and CUDO respectively.

Estimates for the predictive distribution were obtained by taking $T = 500$ stochastic forward passes through the network. For each split, test set evaluation was done 5 times with different seeds. Implementation was done in TensorFlow with the Adam optimizer and a learning rate of 0.001.

For the image classification test we use CIFAR10 (Krizhevsky & Hinton, 2009) which includes 10 object classes with 5,000 and 1,000 images in the training and test sets, respectively. Images are 32x32 RGB format. We trained a ResNet32 architecture with a batch size of 32, learning rate of 0.1, weight decay of 0.0002, leaky ReLU slope of 0.1, and 5 residual units. SGD with momentum was used as the optimizer.

Code for reproducing our experiments is available at https://github.com/icml-mcbn/mcbn.

4.4. Test results

The regression experiment comparing uncertainty quality is summarized in Table 1. We report $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$, expressed as a percentage, which reflects how close the model is to the upper bound, and check to see if the model significantly exceeds the lower bound using a one sample t-test (significance level is indicated by *'s). Further details are provided in Appendix Section 1.7.

In Figure 2 *(left)*, we present a novel visualization of uncertainty quality for regression problems. Data are sorted by estimated uncertainty in the x-axis. Grey dots show the errors in model predictions, and the shaded areas show the model uncertainty. A running mean of the errors appears as a gray line. If uncertainty estimation is working well, a correlation should exist between the mean error (gray line) and uncertainty (shaded area). This indicates that the uncertainty estimation recognizes samples with larger (or smaller) potential for predictive errors.

We applied MCBN on the image classification task of CIFAR10. The baseline in this case is the softmax distribution using the moving average for BN units. Log likelihood (PLL) is the metric used to compare with the baseline. The baseline achieves a PLL of **-0.32** on the test set, while MCBN obtains a PLL of **-0.28**. Table 2 shows the performance of MCBN when using different number of stochastic forward passes (the MCBN batchsize is fixed to the training batch size at 32). PLL improves as the number of the stochastic passes increases, until it is significantly better than the softmax baseline.

To demonstrate how model uncertainty can be obtained from an existing network with minimal effort, we applied MCBN to an image segmentation task using Bayesian SegNet with the main CamVid and PASCAL-VOC models in (Kendall et al., 2015). We simply ran multiple forward passes with different mini-batches randomly taken from the train set. The models obtained from the online model zoo have BN blocks after each layer. We recalculate mean and variance for the first 2 blocks only and use the training statistics for the rest of the blocks. Mini-batches of size 10 and 36 were used for CamVid and VOC respectively

[10] T_i can be found analytically for PLL, but must be found numerically for CRPS.

[11] Where we used an adapted version of the authors' code.

Table 1. **Uncertainty quality measured on eight regression datasets.** MCBN, MCDO and MNF are compared over 5 random 80-20 splits of the data with 5 different random seeds each split. We report $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$, uncertainty metrics CRPS and PLL normalized to a lower bound of constant variance and upper bound that maximizes the metric expressed as a percentage (described in Section 4.2). Higher numbers mean the model is closer to the upper bound. We check if the reported values for $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$ significantly exceed the lower bound using a one sample t-test (significance level indicated by *'s). See text for further details.

	$\overline{\text{CRPS}}$			$\overline{\text{PLL}}$		
Dataset	MCBN	MCDO	MNF	MCBN	MCDO	MNF
Boston	8.50 ****	3.06 ****	5.88 ****	10.49 ****	5.51 ****	1.76 ns
Concrete	3.91 ****	0.93 *	3.13 ***	-36.36 **	10.92 ****	-2.16 ns
Energy	5.75 ****	1.37 ns	1.10 ns	10.89 ****	-14.28 *	-33.88 ns
Kin8nm	2.85 ****	1.82 ****	0.53 ns	1.68 ***	-0.26 ns	0.42 ns
Power	0.24 ***	-0.44 ****	-0.89 ****	0.33 **	3.52 ****	-0.87 ****
Protein	2.66 ****	0.99 ****	0.57 ****	2.56 ****	6.23 ****	0.52 ****
Wine (Red)	0.26 **	2.00 ****	0.94 ****	0.19 *	2.91 ****	0.83 ****
Yacht	-56.39 ***	21.42 ****	24.92 ****	45.58 ****	-41.54 ns	46.19 ****

due to memory limits. The results in Figure 2 *(right)* were obtained from 20 stochastic forward passes, showing high uncertainty near object boundaries. The VOC results are more appealing because of larger mini-batches.

We provide additional experimental results in the Appendix. Appendix Tables 2 and 3 show the mean $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$ values from the regression experiment. Table 4 provides the raw CRPS and PLL scores. In Table 5 we provide RMSE results of the MCBN and MCDO networks in comparison with non-stochastic BN and DO networks. These results indicate that the procedure of multiple forward passes in MCBN and MCDO show slight improvements in the predictive accuracy compared to their non-Bayesian counterparts. In Tables 6 and 7, we investigate the effect of varying batch size while keeping other hyperparameters fixed. We see that performance deteriorates with small batch sizes ($\leq$16), a known issue of BN (Ioffe, 2017). Similarly, results varying the number of stochastic forward passes T is reported in Tables 8 and 9. While performance benefits from large T, in some cases $T = 50$ (i.e. 1/10 of T in the main evaluation) performs well. Uncertainty-error plots for all the datasets are provided in the Appendix.

5. Discussion

The results presented in Tables 1-2 and Appendix Tables 2-9 indicate that MCBN generates meaningful uncertainty estimates that correlate with actual errors in the model's prediction. In Table 1, we show statistically significant improvements over CUBN in the majority of the datasets, both in terms of $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$. The visualizations in Figure 2 and in the Appendix Figures 2-3 show correlations between the estimated model uncertainty and errors of the network's predictions. We perform the same experiments using MCDO and MNF, and find that MCBN generally performs on par with both methods. Looking closer, MCBN outperforms MCDO and MNF in more cases than not, measured by $\overline{\text{CRPS}}$. However, care must be used. The learned parameters are different, leading to different predictive means and confounding direct comparison.

Table 2. **Uncertainty quality for image classification varying number of stochastic forward passes.** Uncertainty quality for image classification measured by PLL. ResNet32 is trained on CIFAR10 with batch size 32. PLL improves as the sampling increases until it is significantly better than the softmax baseline (**-0.32**).

	Number of stochastic forward passes							
	1	2	4	8	16	32	64	128
PLL	-.36	-.32	-.30	-.29	-.29	-.28	-.28	**-.28**

The results on the Yacht Hydrodynamics dataset seem contradictory. The $\overline{\text{CRPS}}$ score for MCBN are extremely negative, while the $\overline{\text{PLL}}$ score is extremely positive. The opposite trend is observed for MCDO. To add to the puzzle, the visualization in Figure 2 depicts an extremely promising uncertainty estimation that models the predictive errors with high fidelity. We hypothesize that this strange behavior is due to the small size of the data set, which only contains 60 test samples, or due to the Gaussian assumption of CRPS. There is also a large variability in the model's accuracy on this dataset, which further confounds the measurements for such limited data.

One might criticize the overall quality of uncertainty estimates observed in all the models we tested, due to the magnitude of $\overline{\text{CRPS}}$ and $\overline{\text{PLL}}$ in Table 1. The scores rarely exceed 10% improvement over the lower bound. However, we caution that these measures should be taken in context. The upper bound is very difficult to achieve in practice – it is optimized for *each test sample individually* – and the lower bound is a quite reasonable estimate.

The study of MCBN sensitivity to batch size revealed that a certain batch size is required for the best performance, dependent on the data. When doing inference on a GPU, large

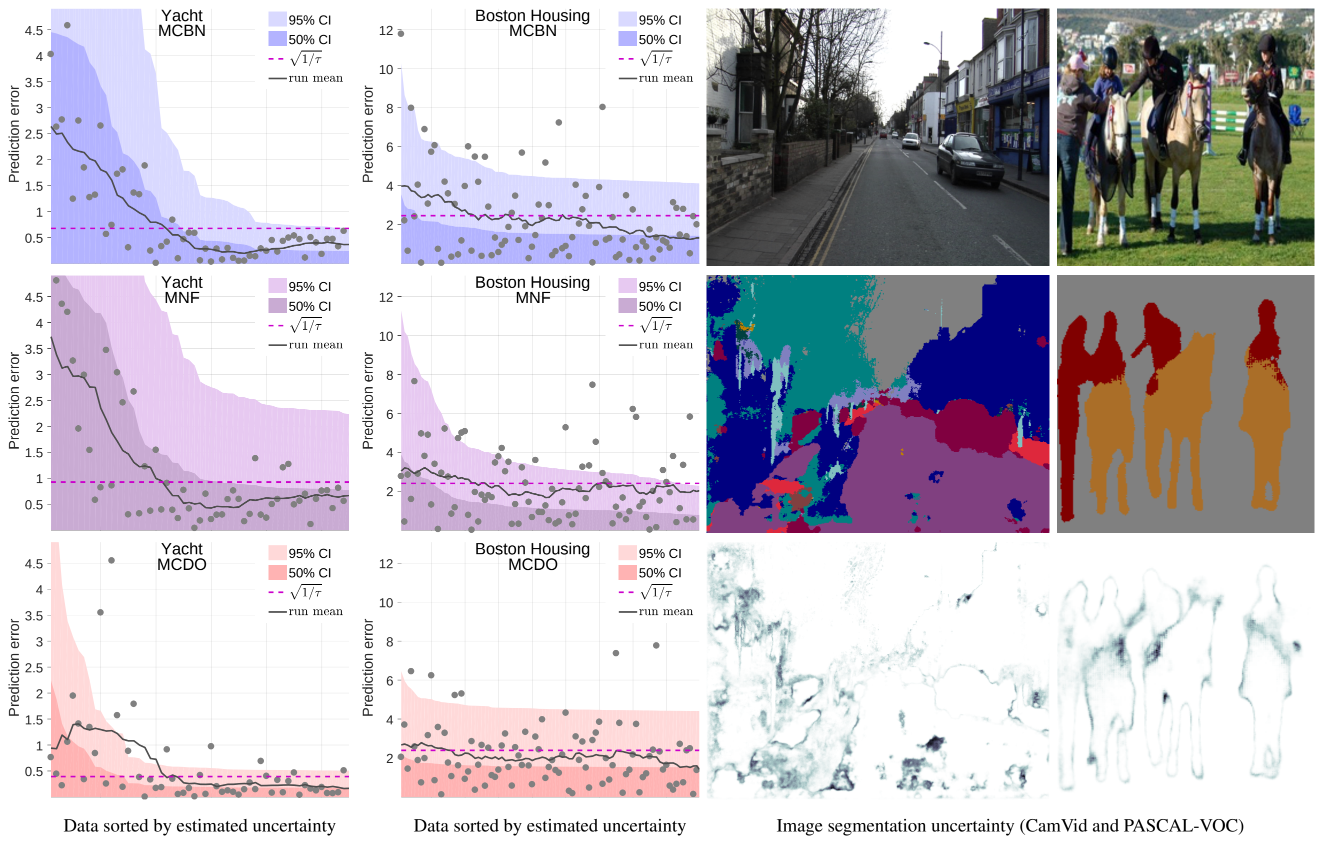

Figure 2. **Uncertainty-error plots (left) and segmentation and uncertainty results applying MCBN to Bayesian SegNet (right).** *(left)* Errors in predictions (gray dots) sorted by estimated uncertainty on select datasets. The shaded areas show model uncertainty for MCBN (blue), MNF (violet) and MCDO (red). The light area indicates 95% CI, dark area 50% CI. Gray dots show absolute prediction errors on the test set, and the gray line depicts a running mean of the errors. The dashed line indicates the optimized constant uncertainty. A correlation between estimated uncertainty (shaded area) and mean error (gray) indicates the uncertainty estimates are meaningful for estimating errors. *(right)* Applying MCBN to Bayesian SegNet (Kendall et al., 2015) on scenes from CamVid (3^{rd} column) and PASCAL-VOC (4^{th} column). Top: original image. Middle: the Bayesian estimated segmentation. Bottom: estimated uncertainty using MCBN for all classes. The uncertainty maps for both datasets are reasonable, but qualitatively better for PASCAL-VOC due to the larger mini-batch size (36) compared to CamVid (10). Smaller batch sizes were used for CamVid due to memory limitations (CamVid images are 360x480 while VOC are 224x224). See Appendix for complete results.

batch sizes may cause memory issues for cases where the input is large and the network has a large number of parameters, as is common for state-of-the-art image classification networks. However, there are various workarounds to this problem. One can store BN statistics, instead of batches, to reduce memory issues. Furthermore, we can use the Gaussian estimate of the BN statistics as discussed previously, which makes memory and computation extremely efficient.

6. Conclusion

In this work, we have shown that training a deep network using batch normalization is equivalent to approximate inference in Bayesian models. We show evidence that the uncertainty estimates from MCBN correlate with actual errors in the model's prediction, and are useful for practical tasks such as regression, image classification, and image segmentation. Our experiments show that MCBN yields a significant improvement over the optimized constant uncertainty baseline, on par with MCDO and MNF. Our evaluation also suggests new normalized metrics based on useful upper and lower bounds, and a new visualization which provides an intuitive explanation of uncertainty quality.

Finally, it should be noted that over the past few years, batch normalization has become an integral part of most – if not all – cutting edge deep networks. We have shown that it is possible to obtain meaningful uncertainty estimates from existing models without modifying the network or the training procedure. With a few lines of code, robust uncertainty estimates can be obtained by computing the variance of multiple stochastic forward passes through an existing network.

References

Bui, T. D., Hernández-Lobato, D., Li, Y., Hernández-Lobato, J. M., and Turner, R. E. Deep Gaussian Processes for Regression using Approximate Expectation Propagation. In *ICML*, 2016.

Chen, X., Kundu, K., Zhang, Z., Ma, H., Fidler, S., and Urtasun, R. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2147–2156, 2016.

Djuric, U., Zadeh, G., Aldape, K., and Diamandis, P. Precision histology: how deep learning is poised to revitalize histomorphology for personalized cancer care. *npj Precision Oncology*, 1(1):22, 2017.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, Feb 2017.

Gal, Y. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning. *ICML*, 48:1–10, 2015.

Ghahramani, Z. *Delve Datasets*. University of Toronto, 1996. URL http://www.cs.toronto.edu/{~}delve/data/kin/desc.html.

Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, May 2015.

Gneiting, T. and Raftery, A. E. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Graves, A. Practical Variational Inference for Neural Networks. *NIPS*, 2011.

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

Hinton, G. E. and Van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

Ioffe, S. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *CoRR*, abs/1702.03275, 2017. URL http://arxiv.org/abs/1702.03275.

Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, 2015. URL http://arxiv.org/abs/1502.03167.

Karpathy, A. Convnetjs demo: toy 1d regression, 2015. URL http://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html.

Kendall, A., Badrinarayanan, V., and Cipolla, R. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. *CoRR*, abs/1511.0, 2015. URL http://arxiv.org/abs/1511.02680.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. In *ICLR*, 2014.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Krueger, D., Huang, C.-W., Islam, R., Turner, R., Lacoste, A., and Courville, A. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Li, Y. and Gal, Y. Dropout Inference in Bayesian Neural Networks with Alpha-divergences. *arXiv*, 2017.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational Bayesian neural networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2218–2227, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/louizos17a.html.

MacKay, D. J. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Neal, R. M. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

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

Selten, R. Axiomatic characterization of the quadratic scoring rule. *Experimental Economics*, 1(1):43–62, 1998.

Shen, L. End-to-end training for whole image breast cancer diagnosis using an all convolutional design. *arXiv preprint arXiv:1708.09427*, 2017.

University of California, I. UC Irvine Machine Learning Repository, 2017. URL `https://archive.ics.uci.edu/ml/index.html`.