Training Binary Neural Networks using the Bayesian Learning Rule

Xiangming Meng [1]   Roman Bachmann [* 2]   Mohammad Emtiyaz Khan [1]

Abstract

Neural networks with binary weights are computation-efficient and hardware-friendly, but their training is challenging because it involves a discrete optimization problem. Surprisingly, ignoring the discrete nature of the problem and using gradient-based methods, such as the Straight-Through Estimator, still works well in practice. This raises the question: are there principled approaches which justify such methods? In this paper, we propose such an approach using the *Bayesian learning rule*. The rule, when applied to estimate a Bernoulli distribution over the binary weights, results in an algorithm which justifies some of the algorithmic choices made by the previous approaches. The algorithm not only obtains state-of-the-art performance, but also enables uncertainty estimation for continual learning to avoid catastrophic forgetting. Our work provides a principled approach for training binary neural networks which justifies and extends existing approaches.

1. Introduction

Deep neural networks (DNNs) have been remarkably successful in machine learning but their training and deployment requires a high energy budget and hinders their application to resource-constrained devices, such as mobile phones, wearables, and IoT devices. Binary neural networks (BiNNs), where weights and/or activations are restricted to binary values, are one promising solution to address this issue (Courbariaux et al., 2016; 2015). Compared to full-precision DNNs, e.g., using 32-bits, using BiNNs directly gives a 32 times reduction in the model size. Further computational efficiency is obtained by using specialized hardware,

e.g., by replacing the multiplication and addition operations with the bit-wise `xnor` and `bitcount` operations (Rastegari et al., 2016; Mishra et al., 2018; Bethge et al., 2020). In the near future, BiNNs are expected to play an important role in energy-efficient and hardware-friendly deep learning.

A problem with BiNNs is that their training is much more difficult than their continuous counterpart. BiNNs obtained by quantizing already trained DNNs do not work well, and it is preferable to optimize for binary weights directly. Such training is challenging because it involves a discrete optimization problem. Continuous optimization methods such as the Adam optimizer (Kingma & Ba, 2015) are not expected to perform well or even converge.

Despite such theoretical issues, a method called Straight-Through-Estimator (STE) (Bengio et al., 2013), which employs continuous optimization methods, works remarkably well (Courbariaux et al., 2015). The method is justified based on "latent" real-valued weights which are discretized at every iteration to get binary weights. The gradients used to update the latent weights, however, are computed at the binary weights (see Figure 1 (a) for an illustration). It is not clear why these gradients help the search for the minimum of the discrete problem (Yin et al., 2019; Alizadeh et al., 2019). Another recent work by Helwegen et al. (2019) dismisses the idea of latent weights, and proposes a new optimizer called Binary Optimizer (Bop) based on *inertia*. Unfortunately, the steps used by their optimizers too are derived based on intuition and are not theoretically justified using an optimization problem. Our goal in this paper is to address this issue and propose a principled approach to justify the algorithmic choices of these previous approaches.

In this paper, we present a Bayesian perspective to justify previous approaches. Instead of optimizing a discrete objective, the Bayesian approach relaxes it by using a distribution over the binary variable, resulting in a principled approach for discrete optimization. We use a Bernoulli approximation to the posterior and estimate it using a recently proposed approximate Bayesian inference method called the Bayesian learning rule (Khan & Lin, 2017; Khan & Rue, 2020). This results in an algorithm which justifies some of the algorithmic choices made by existing methods; see Table 1 for a summary of results. Since our algorithm is based on a well-defined optimization problem, it is easier to extend its
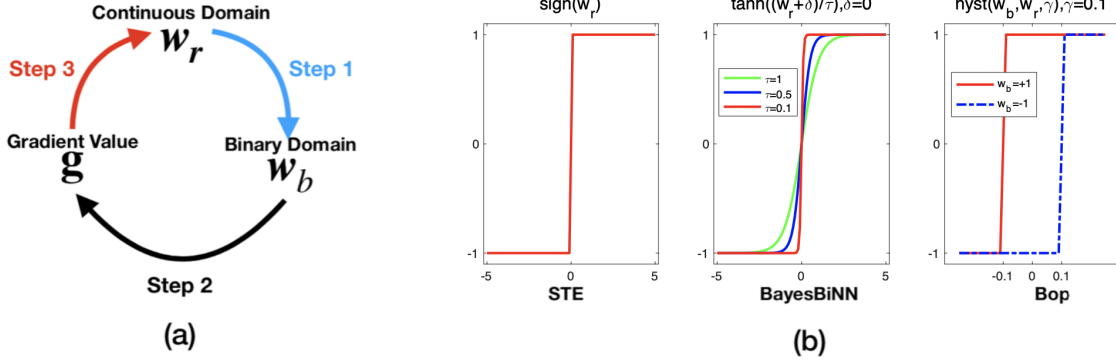
Figure 1. (a): The three steps involved in training BiNNs. In step 1, we obtain binary weights $\boldsymbol{w}_b$ from the real-valued parameters $\boldsymbol{w}_r$. In step 2, we compute gradients at $\boldsymbol{w}_b$ and, in step 3, update $\boldsymbol{w}_r$. (b): Different functions used to convert continuous parameters to binary weights. From left to right: Sign function used in STE; Tanh function used in BayesBiNN; Hysteresis function (see (25)) used in Bop.

	STE	Our BayesBiNN method	Bop
Step 1: Get $\boldsymbol{w}_b$ from $\boldsymbol{w}_r$	$\boldsymbol{w}_b \leftarrow \text{sign}(\boldsymbol{w}_r)$	$\boldsymbol{w}_b \leftarrow \tanh\left((\boldsymbol{w}_r + \boldsymbol{\delta})/\tau\right)$	$\boldsymbol{w}_b \leftarrow \text{hyst}(\boldsymbol{w}_r, \boldsymbol{w}_b, \gamma)$
Step 2: Compute gradient at $\boldsymbol{w}_b$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$
Step 3: Update $\boldsymbol{w}_r$	$\boldsymbol{w}_r \leftarrow \boldsymbol{w}_r - \alpha\mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha\mathbf{s} \odot \mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha\mathbf{g}$

Table 1. This table compares the steps of our algorithm BayesBiNN to the two existing methods, STE (Bengio et al., 2013) and Bop (Helwegen et al., 2019). Here, $\boldsymbol{w}_b$ and $\boldsymbol{w}_r$ denote the binary and real-valued weights. For step 1, where $\boldsymbol{w}_b$ are obtained from $\boldsymbol{w}_r$, STE uses the sign of $\boldsymbol{w}_r$ while BayesBiNN uses a $tanh$ function with a small noise δ sampled from a Bernoulli distribution and a temperature parameter τ. As Figure 1 (b) shows, as τ goes to 0, Step 1 of BayesBiNN becomes equal to that of STE. Step 1 of Bop uses the *hysteresis* function shown in Figure 1 (b) and becomes similar to sign function as the threshold γ goes to 0 (it is flipped but the sign is irrelevant for binary variables). Step 2 is the same for all algorithms. Step 3 of BayesBiNN is very similar to Bop, except that a scaling $\mathbf{s}$ is used (which is similar to the adaptive learning rate algorithms); see (12) in Section 3.

application. We show an application for continual learning to avoid catastrophic forgetting (Kirkpatrick et al., 2016). To the best of our knowledge, there is no other work on continual learning of BiNNs so far, perhaps because extending existing methods, like STE, for such tasks is not trivial. Overall, our work provides a principled approach for training BiNNs that justifies and extends previous approaches. The code to reproduce the results is available at https://github.com/team-approx-bayes/BayesBiNN.

1.1. Related Works

There are two main directions on the study of BiNNs: one involves the design of special network architecture tailored to binary operations (Courbariaux et al., 2015; Rastegari et al., 2016; Lin et al., 2017; Bethge et al., 2020) and the other is on the training methods. The latter is the main focus of this paper.

Our algorithm is derived using the Bayesian learning rule recently proposed by Khan & Lin (2017); Khan & Rue (2020), which is obtained by optimizing the Bayesian objective by using natural gradient descent (Amari, 1998; Hoffman et al.,

2013; Khan & Lin, 2017). It is shown in Khan & Rue (2020) that the Bayesian learning rule can be used to derive and justify many existing learning-algorithms in fields such as optimization, Bayesian statistics, machine learning and deep learning. In particular, the Adam optimizer can also be derived as a special case (Khan et al., 2018; Osawa et al., 2019). Our application is yet another example where the rule is used to justify existing algorithms that perform well in practice but whose mechanisms are not well understood.

Instead of using the Bayesian learning rule, it is possible to use other types of variational inference methods, e.g., Shayer et al. (2018); Peters & Welling (2018) used a *variational optimization* approach (Staines & Barber, 2012) along with the local reparameterization trick. The Gumbel-softmax trick (Maddison et al., 2017; Jang et al., 2017) is also used in Louizos et al. (2019) to train BiNNs. However, instead of specifying a Bernoulli distribution over the weights, Louizos et al. (2019) construct a noisy quantizer and their optimization objective is different from ours. Unlike our method, none of these methods (Shayer et al., 2018; Peters & Welling, 2018; Louizos et al., 2019) result in an update similar to either STE or Bop.

2. Training Binary Neural Networks (BiNNs)

Given $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, the goal is to train a neural network $f_{\boldsymbol{w}}(\boldsymbol{x})$ with binary weights $\boldsymbol{w} \in \{-1, +1\}^W$, where W is the number of weights. The challenge is in optimizing the following discrete optimization objective:

$$\min_{\boldsymbol{w} \in \{-1,1\}^W} \sum_{i \in \mathcal{D}} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)), \quad (1)$$

where $\ell(y_i, \hat{y}_i)$ is a loss function, e.g., cross-entropy loss for the model predictions $\hat{y}_i := f_{\boldsymbol{w}}(\boldsymbol{x}_i)$. It is clear that binarized weights obtained from pre-trained NNs with real-weights do not minimize (1) and therefore not expected to give good performance. Optimizing the objective with respect to binary weights is difficult since gradient-based methods cannot be directly applied. The gradient of the real-valued weights are not expected to help in the search for the minimum of a discrete objective (Yin et al., 2019).

Despite such theoretical concerns, the Straight-Through Estimator (STE) (Bengio et al., 2013), which utilizes gradient-based methods, works extremely well. There have been many recent works that build upon this method, including BinaryConnect (Courbariaux et al., 2015), Binarized neural networks (Courbariaux et al., 2016), XOR-Net (Rastegari et al., 2016), as well as the most recent MeliusNet (Bethge et al., 2020). The general approach of such methods is shown in Figure 1 in three steps. In step 1, we obtain binary weights $\boldsymbol{w}_b$ from the real-valued parameters $\boldsymbol{w}_r$. In step 2, we compute gradients at $\boldsymbol{w}_b$ and, in step 3, update $\boldsymbol{w}_r$ using the gradients. STE makes a particular choice for the step 1 where a sign function is used to obtain the binary weights from the real-valued weights (see Table 1 for a pseudocode). However, since the gradient of the sign function with respect to $\boldsymbol{w}_r$ is zero almost everywhere, it implies that $\nabla_{\boldsymbol{w}_r} \approx \nabla_{\boldsymbol{w}_b}$. This approximation can be justified in some settings (Yin et al., 2019) but in general the reasons behind its effectiveness are unknown.

Recently Helwegen et al. (2019) proposed a new method that goes against the justification behind STE. They argue that "latent" weights used in STE based methods do not exist. Instead, they provide a new perspective: the sign of each element of $\boldsymbol{w}_r$ represents a binary weight while its magnitude encodes some inertia against flipping the sign of the binary weight. With this perspective, they propose the Binary optimizer (Bop) method which keeps track of an exponential moving average (Gardner Jr, 1985) of the gradient $\mathbf{g}$ during the training process and then decide whether to flip the sign of the binary weights when they exceed a certain threshold γ. The Bop algorithm is shown in Table 1. However, derivation of Bop is also based on intuition and heuristics. It remains unclear why the exponential moving average of the gradient is used in Step 3 and what objective the algorithm is optimizing. The selection of the threshold

γ is another difficult choice in the algorithm.

Indeed, Bayesian methods do present a principled way to incorporate the ideas used in both STE and Bop. For example, the idea of "generating" binary weights from real-valued parameters can be though of as sampling from a discrete distribution with real-valued parameters. In fact, the sign function used in STE is related to the "soft-thresholding" used in machine learning. Despite this there exist no work on Bayesian training of BiNNs that can give an algorithm similar to STE or Bop. In this work, we fix this gap and show that, by using the Bayesian learning rule, we recover a method that justifies some of the steps of STE and Bop, and enable us to extend their application. We will now describe our method.

3. BayesBiNN: Binary NNs with Bayes

We will now describe our approach based on a Bayesian formulation of the discrete optimization problem in (1). A Bayesian formulation of a loss-based approach can be written as the following minimization problem with respect to a distribution $q(\boldsymbol{w})$ (Zellner, 1988; Bissiri et al., 2016)

$$\mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^N \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] + \mathbb{D}_{KL}[q(\boldsymbol{w}) \,\|\, p(\boldsymbol{w})], \quad (2)$$

where $p(\boldsymbol{w})$ is a prior distribution and $q(\boldsymbol{w})$ is the posterior distribution or its approximation. The formulation is general and does not require the loss to correspond to a probabilistic model. When the loss indeed corresponds to a negative log likelihood function, this minimization results in the posterior distribution which is equivalent to Bayes' rule (Bissiri et al., 2016). When the space of $q(\boldsymbol{w})$ is restricted, this results in an approximation to the posterior, which is then equivalent to variational inference (Jordan et al., 1999). For our purpose, this formulation enables us to derive an algorithm that resembles existing methods such as STE and Bop.

3.1. BayesBiNN optimizer

To solve the optimization problem (2), the Bayesian learning rule (Khan & Rue, 2020) considers a class of minimal exponential family distributions (Wainwright & Jordan, 2008)

$$q(\boldsymbol{w}) := h(\boldsymbol{w}) \exp \left[\boldsymbol{\lambda}^T \phi(\boldsymbol{w}) - A(\boldsymbol{\lambda}) \right], \quad (3)$$

where $\boldsymbol{\lambda}$ is the natural parameter, $\phi(\boldsymbol{w})$ is the vector of sufficient statistics, $A(\boldsymbol{\lambda})$ is the log-partition function, and $h(\boldsymbol{w})$ is the base measure. When the prior distribution $p(\boldsymbol{w})$ belongs to the same exponential-family as $q(\boldsymbol{w})$ in (3), and the base measure $h(\boldsymbol{w}) = 1$, the Bayesian learning uses the following update of the natural parameter (Khan & Lin,

2017; Khan & Rue, 2020)

$$\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha \left\{ \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] - \boldsymbol{\lambda}_0 \right\}, \quad (4)$$

where α is the learning rate, $\boldsymbol{\mu} = \mathbb{E}_{q(\boldsymbol{w})}[\phi(\boldsymbol{w})]$ is the expectation parameter of $q(\boldsymbol{w})$, and $\boldsymbol{\lambda}_0$ is the natural parameter of the prior distribution $p(\boldsymbol{w})$, which is assumed to belong to the same exponential-family as $q(\boldsymbol{w})$. The Bayesian learning rule is a natural gradient algorithm (Khan & Lin, 2017; Khan & Rue, 2020). An interesting point of this rule is that the gradient is computed with respect to the expectation parameter $\boldsymbol{\mu}$, while the update is performed on the natural parameter $\boldsymbol{\lambda}$. This particular choice leads to an update similar to STE for BiNNs, as we show next.

We start by specifying the form of $p(\boldsymbol{w})$ and $q(\boldsymbol{w})$. A priori, we assume that the weights are equally likely to be either -1 or $+1$, i.e., the prior $p(\boldsymbol{w})$ is a (symmetric) Bernoulli distribution with a probability of $\frac{1}{2}$ for each state. For the posterior approximation $q(\boldsymbol{w})$, we use the mean-field (symmetric) Bernoulli distribution

$$q(\boldsymbol{w}) = \prod_{j=1}^{W} p_j^{\frac{1+w_j}{2}} (1-p_j)^{\frac{1-w_j}{2}}, \quad (5)$$

where p_j is the probability that $w_j = +1$, and W is the number of parameters. Our goal is to learn the parameters p_j of the approximations. The Bernoulli distribution defined in (5) is a special case of the *minimal* exponential family distribution, where the corresponding natural and expectation parameters of each weight w_i are

$$\lambda_j := \frac{1}{2} \log \frac{p_j}{1-p_j}, \qquad \mu_j := 2p_j - 1. \quad (6)$$

The natural parameter $\boldsymbol{\lambda}_0$ of the prior $p(\boldsymbol{w})$ is therefore $\boldsymbol{0}$. Using these definitions, we can directly apply the Bayesian learning rule to learn the posterior Bernoulli distribution of the binary weights.

In addition to these definitions, we also require the gradient with respect to $\boldsymbol{\mu}$ to implement the rule (4). A straightforward solution is to use the REINFORCE method (Williams, 1992) which transforms the gradient of the expectation into the expectation of the gradient by using the log-derivative trick, i.e.,

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} [\ell(y, f_{\boldsymbol{w}}(\boldsymbol{x}))] = \mathbb{E}_{q(\boldsymbol{w})} [\ell(y, f_w(\boldsymbol{x})) \nabla_{\mu} \log q(\boldsymbol{w})].$$

This method, however, does not use the *minibatch gradient* (the gradient of the loss $\ell(y, f_{\boldsymbol{w}}(\boldsymbol{x}))$ on a minibatch of examples), which is essential to show the similarity to STE and Bop. The REINFORCE method also suffers from high variance. Due to these reasons, we do not use this method.

Instead, we resort to another reparameterization trick for discrete variables called the Gumbel-softmax trick (Maddison

et al., 2017; Jang et al., 2017), which, as we will see, leads to an update similar to STE/Bop. The idea of Gumbel-softmax trick is to introduce the *concrete distribution* that leads to a *relaxation* of the discrete random variables. Specifically, as shown in Appendix B in (Maddison et al., 2017), for a binary variable $w_j \in \{0, 1\}$ with $P(w_j = 1) = p_j$, we can use the following relaxed variable $w_r^{\epsilon_j, \tau}(p_j) \in (0, 1)$:

$$w_r^{\epsilon_j, \tau}(p_j) := \frac{1}{1 + \exp\left(-\frac{2\lambda_j + 2\delta_j}{\tau}\right)}, \quad (7)$$

where $\tau > 0$ is a temperature parameter, $\lambda_j := \frac{1}{2} \log \frac{p_j}{1-p_j}$ is the natural parameter, and δ_j is defined as follows,

$$\delta_j := \frac{1}{2} \log \frac{\epsilon_j}{1 - \epsilon_j}, \quad (8)$$

with $\epsilon_j \sim \mathcal{U}(0, 1)$ sampled from a uniform distribution. The $w_r^{\epsilon_j, \tau}(p_j)$ are samples from a Concrete distribution which has a closed-form expression (Maddison et al., 2017):

$$p\left(w_r^{\epsilon_j, \tau}(p_j)\right) \qquad (9)$$
$$:= \frac{\tau e^{2\lambda}(w_r^{\epsilon_j, \tau}(p_j))^{-\tau-1} \left(1 - (w_r^{\epsilon_j, \tau}(p_j))\right)^{-\tau-1}}{\left(e^{2\lambda}(w_r^{\epsilon_j, \tau}(p_j))^{-\tau} + \left(1 - (w_r^{\epsilon_j, \tau}(p_j))\right)^{-\tau}\right)^2}.$$

Instead of differentiating the objective with respect to binary variables w_j, we can differentiate with respect to $w_r^{\epsilon_j, \tau}(p_j)$. We will use this to approximate the gradient with respect to $\boldsymbol{\mu}$ in terms of the minibatch gradient.

In our case, entries w_j of $\boldsymbol{w}$ take value in $\{+1, -1\}$ rather than in $\{0, 1\}$. The relaxed version could be obtained by a linear transformation of the concrete variables $w_r^{\epsilon_j, \tau}(p_j)$ in (7), i.e.,

$$w_b^{\epsilon_j, \tau}(\lambda_j) := 2w_r^{\epsilon_j, \tau}(p_j) - 1 = \tanh\left((\lambda_j + \delta_j)/\tau\right),$$
$$(10)$$

where, unlike (7), we have explicitly written the dependency in terms of λ_j instead of the vector of p_j. Since $w_b^{\epsilon_j, \tau}(\lambda_j)$ are continuous, we can differentiate them with respect to μ_j by using the chain rule. The lemma below states the result where $\boldsymbol{w}_b^{\epsilon, \tau}(\boldsymbol{\lambda})$ and $\boldsymbol{\epsilon}$ denote the vectors of $w_b^{\epsilon_j, \tau}(\lambda_j)$ and ϵ_j, respectively, for all $j = 1, 2, \ldots, W$.

Lemma 1 *By using the Gumbel-softmax trick, we get the following approximation in terms of the minibatch gradient:*

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] \approx \mathbf{s} \odot \mathbf{g}, \quad (11)$$

where

$$\mathbf{g} := \frac{1}{M} \sum_{i \in \mathcal{M}} \nabla_{\boldsymbol{w}_r} \ell(y_i, f_{\boldsymbol{w}_r}(\boldsymbol{x}_i))|_{\boldsymbol{w}_r = \boldsymbol{w}_b^{\epsilon, \tau}(\boldsymbol{\lambda})}, \quad (12)$$

$$\mathbf{s} := \frac{N(1 - (\boldsymbol{w}_b^{\epsilon, \tau}(\boldsymbol{\lambda}))^2)}{\tau(1 - \tanh(\boldsymbol{\lambda})^2)}, \quad (13)$$

and $\mathcal{M}$ is a mini-batch of M examples.

Proof 1 *Using (10), we can first approximate the objective in terms of $\boldsymbol{w}_b^{\boldsymbol{\epsilon},\tau}(\boldsymbol{\lambda})$ and $\boldsymbol{\epsilon}$, and then push the gradient with respect to $\boldsymbol{\mu}$ inside the expectation as shown below:*

$$
\nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right]
$$

$$
\approx \mathbb{E}_{q(\boldsymbol{\epsilon})} \left[\sum_{i=1}^{N} \nabla_{\boldsymbol{\mu}} \ell(y_i, f_{\boldsymbol{w}_b^{\boldsymbol{\epsilon},\tau}(\boldsymbol{\lambda})}(\boldsymbol{x}_i)) \right]. \quad (14)
$$

Using the chain rule, the j-th element of the gradient on the right hand side can be obtained as follows:

$$
\nabla_{\mu_j} \ell(y_i, f_{\boldsymbol{w}_b^{\boldsymbol{\epsilon},\tau}}(\boldsymbol{x}_i)) = \nabla_{w_b^{\epsilon_j,\tau}} \ell(y_i, f_{\boldsymbol{w}_b^{\boldsymbol{\epsilon},\tau}}(\boldsymbol{x}_i)) \frac{dw_b^{\epsilon_j,\tau}(\lambda_j)}{d\mu_j}. \quad (15)
$$

According to the definition of natural parameter and expectation parameter in (6), we have $\lambda_j = \frac{1}{2} \log \frac{1+\mu_j}{1-\mu_j}$, therefore after some algebra we can write:

$$
\frac{dw_b^{\epsilon_j,\tau}}{d\mu_j} = \frac{1 - (w_b^{\epsilon_j,\tau}(\lambda_j))^2}{\tau \left(1 - \tanh^2(\lambda_j)\right)}. \quad (16)
$$

By using a mini-batch $\mathcal{M}$ of M examples and one sample $\boldsymbol{\epsilon}$, (14)-(16) give us (11). $\qquad\square$

Substituting the result of Lemma 1 into the Bayesian learning rule in (4), we obtain the following update:

$$
\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha \left[\mathbf{s} \odot \mathbf{g} - \boldsymbol{\lambda}_0 \right]. \quad (17)
$$

The resulting optimizer, which we call BayesBiNN, is shown in Table 1, where we assume $\boldsymbol{\lambda}_0 = \mathbf{0}$ (since the probability of $w_i = +1$ is 1/2 a priori). For the ease of comparison with other methods, the natural parameter $\boldsymbol{\lambda}$ is replaced with continuous variables $\boldsymbol{w}_r$.

At test-time, we can either use the predictions obtained using Monte-Carlo average (which we refer to as the "mean"):

$$
\hat{p}_k = \frac{1}{C} \sum_{c=1}^{C} p\left(y = k | \boldsymbol{x}, \boldsymbol{w}^{(c)}\right), \quad (18)
$$

with $\boldsymbol{w}^{(c)} \sim q(\boldsymbol{w})$ and C is the number of samples, or the predictions obtained by using the mode $\hat{\boldsymbol{w}}$ of of $q(\boldsymbol{w})$: $\hat{p}_k = p\left(y = k | \boldsymbol{x}, \hat{\boldsymbol{w}}\right)$, where $\hat{\boldsymbol{w}} = \text{sign}(\tanh(\boldsymbol{\lambda}))$ (which we refer to as the "mode").

3.2. Justification of Previous Approaches

In this section, we show how BayesBiNN justifies the steps of STE and Bop. A summary is shown in Table 1. First, BayesBiNN justifies the use of gradient based methods to solve the discrete optimization problem (1). As opposed to

(1), the new objective in (2) is over a continuous parameter $\boldsymbol{\lambda}$ and thus gradient descent can be used. The underlying principle is similar to stochastic relaxation for non-differentiable optimization (Lemaréchal, 1989; Geman & Geman, 1984), evolution strategies (Huning, 1976), and variational optimization (Staines & Barber, 2012).

Second, some of the algorithmic choices of previous methods such as STE and Bop are justified by BayesBiNN. Specifically, when the temperature τ in BayesBiNN is small enough, the $\tanh(\cdot)$ function in Table 1 behaves like the $\text{sign}(\cdot)$ function used in STE; see Figure 1 (b). From this perspective, the latent weights $\mathbf{w}_r$ in STE play a similar role as the natural parameter $\boldsymbol{\lambda}$ of BayesBiNN. In particular, when there is no sampling, i.e., $\delta \leftarrow 0$ in BayesBiNN, the two algorithms will become very similar to each other. BayesBiNN justifies the step 1 used in STE by using the Bayesian perspective.

BayesBiNN also justifies step 3 of Bop[1] in Table 1, where an exponential moving average of gradients is used. This is referred to as the momentum term in Bop which plays a similar role as the natural parameter in BayesBiNN. In Helwegen et al. (2019), the momentum is interpreted as a quantity related to *inertia*, which indicates the strength of the state of weights. Since the natural parameter in the binary distribution (5) essentially indicates the strength of the probability being -1 or $+1$ for each weight, BayesBiNN provides an alternative explanation for Bop.

A recent mirror descent view proposed in Ajanthan et al. (2019b) also interprets the continuous parameters as the dual of the quantized ones. As there is an equivalence between the natural gradient descent and mirror descent (Raskutti & Mukherjee, 2015; Khan & Lin, 2017), the proposed BayesBiNN also provides an interesting perspective on the mirror descent framework for BiNNs training.

3.3. Benefits of BayesBiNN

Apart from justifying previous methods, BayesBiNN has several other advantages. First, since its algorithmic form is similar to existing optimizers, it is very easy to implement BayesBiNN by using existing codebases. Second, as a Bayesian method, BayesBiNN provides uncertainty estimates, which can be useful for decision making. The uncertainty obtained using BayesBiNN can enable us to perform continual learning by using the variational continual learning (VCL) framework (Nguyen et al., 2018), as we discuss next.

In continual learning, our goal is to learn the parameters of the neural network from a set of sequentially arriving

---

[1]Note that the step 3 of Bop in Table 1 is an equivalent but "flipped" version of the one used by Helwegen et al. (2019); see Appendix A for details.

datasets $\mathcal{D}_t = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N_t}, t = 1, 2, \ldots, T$. While training the t-th task with dataset $\mathcal{D}_t$, we do not have access to the datasets of past tasks, i.e., $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{t-1}$. Training on task $\mathcal{D}_t$ using a deep-learning optimizer usually leads to a huge performance loss on the past tasks (Kirkpatrick et al., 2016). The goal of continual learning is to fix such catastrophic forgetting of the past.

For full-precision networks, a common approach to solve this problem is to use weight-regularization, e.g., the *elastic weight consolidation* (EWC) method (Kirkpatrick et al., 2016) uses a Fisher information matrix to regularize the weights:

$$\sum_{i \in \mathcal{D}_t} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) + \varepsilon(\boldsymbol{w} - \boldsymbol{w}_{t-1})^T \mathbf{F}_t(\boldsymbol{w} - \boldsymbol{w}_{t-1}),$$

(19)

where ε is a regularization parameter and $\mathbf{F}_t$ is the Fisher information matrix at $\boldsymbol{w}_t$. The hope is to keep the new weights close to the old weights, but in a discrete optimization problem, it is impossible to characterize such closeness using a quadratic regularizer as above. Therefore, it is unclear why such a regularizer will be useful. In addition, the use of the Fisher information matrix $\mathbf{F}_t$ typically assumes that the weights are continuous and the matrix does not provide a meaningful quantity for discrete weights. For these reasons, extending existing approaches such as STE and Bop to continual learning is a nontrivial task.

Fortunately, for BayesBiNN, this is very easy because the objective function is well-defined. We use the VCL framework (Nguyen et al., 2018) where we regularize the distributions instead of the weights. The Kullback-Leibler term is used as the regularizer. Denoting by $q_{t-1}(\boldsymbol{w})$ the posterior distribution at task $t - 1$, we can replace the prior distribution $p(\boldsymbol{w})$ in (2) by $q_{t-1}(\boldsymbol{w})$:

$$\mathbb{E}_{q_t(\boldsymbol{w})} \left[\sum_{i \in \mathcal{D}_t} \ell(y_i, f_w(\boldsymbol{x}_i)) \right] + \mathbb{D}_{KL}[q_t(\boldsymbol{w}) \| q_{t-1}(\boldsymbol{w})]. \quad (20)$$

This leads to a slight modification of the update in Bayes-BiNN where the prior natural parameter $\boldsymbol{\lambda}_0$ in (17) is replaced by the natural parameter $\boldsymbol{\lambda}_{t-1}$ of $q_{t-1}(\boldsymbol{w})$. The new update of the natural parameter $\boldsymbol{\lambda}_t$ of $q_t(\boldsymbol{w})$ is shown below:

$$\boldsymbol{\lambda}_t \leftarrow (1 - \alpha)\boldsymbol{\lambda}_t - \alpha \left[\mathbf{s} \odot \mathbf{g} - \boldsymbol{\lambda}_{t-1} \right]. \quad (21)$$

By using a posterior approximation $q(\boldsymbol{w})$ and a well-defined objective, BayesBiNN enables the application of STE/Bop like methods to such challenging continual learning problems.

4. Experimental Results

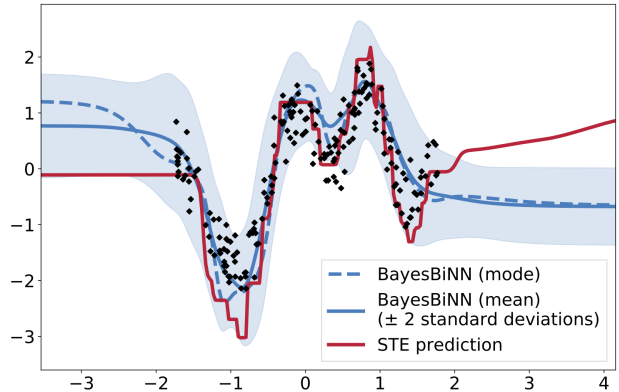In this section, we present numerical experiments to demonstrate the performance of BayesBiNN on both synthetic and

Figure 2. Regression on Snelson dataset (Snelson & Ghahramani, 2005). BayesBiNN (mean) gives much smoother curve than the STE. Uncertainty is low in areas with plenty of data points.

real image data for different kinds of neural network architectures. We also show an application of BayesBiNN to continual learning. The code to reproduce the results is available at https://github.com/team-approx-bayes/BayesBiNN.

4.1. Synthetic Data

First, we present visualizations on toy regression and binary classification problems. The STE (Bengio et al., 2013) algorithm is used as a baseline for which we employ Adam for training. We use a multi-layer perceptron (MLP) with two hidden layers of 64 units and tanh activation functions.

Regression In Figure 2, we show results for regression on the Snelson dataset (Snelson & Ghahramani, 2005). For this experiment, we add a Batch normalization (BN) (Ioffe & Szegedy, 2015) layer (but no learned gain or bias terms) after the last fully connected layer. As seen in Figure 2, predictions obtained using 'BayesBiNN (mean)' gives much smoother curves than STE, as expected. Uncertainty is lower in the areas with little noise and plenty of data points compared to areas with no data. Experimental details of the training process are provided in the Appendix B.1.

Classification Figure 3 shows STE and BayesBiNN on the two moons dataset (Moons) with 100 data points in each class. STE (the leftmost figure) gives a point estimate of the weights and results in a fairly deterministic classifier. When using the mode of the BayesBiNN distribution (the middle figure), the results are similar, with the fit being slightly worse but overall less overconfident, especially in the regions with no data. Using the mean over 10 samples drawn from the posterior distribution $q(\boldsymbol{w})$ (the rightmost figure), we get much better uncertainty estimates as we move away from the data. Experimental details of the training process
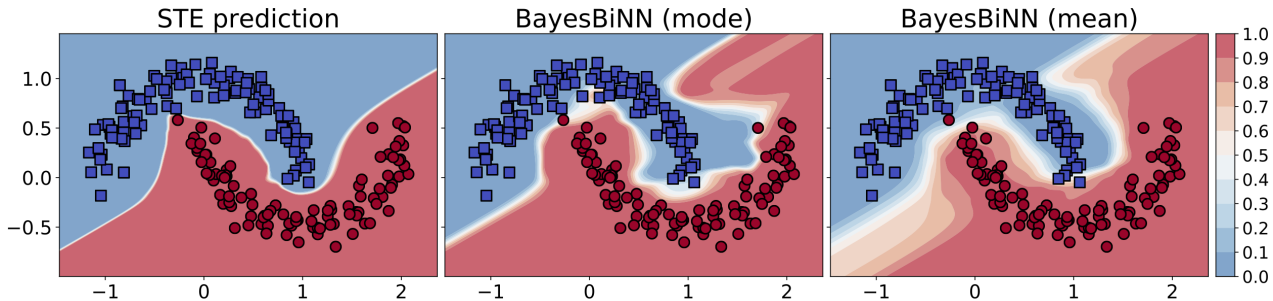
Figure 3. Classification on the two moons dataset with different optimizers. From left to right: STE, BayesBiNN using the mode, BayesBiNN using the predictive mean from 10 posterior Monte Carlo samples. STE is more overconfident than BayesBiNN, and BayesBiNN (mean) gives reasonable uncertainty in regions further away from the data.

Table 2. Test accuracy of different optimizers for MNIST, CIFAR-10 and CIFAR-100 (Averaged over 5 runs). In all the three benchmark datasets, BayesBiNN achieves similar performance as STE Adam and closely approaches the performance of full-precision networks.

Optimizer	MNIST	CIFAR-10	CIFAR-100
STE Adam	**98**.**85** $\pm$ 0.09 %	**93**.**55** $\pm$ 0.15 %	72.89 $\pm$ 0.21 %
Bop	98.47 $\pm$ 0.02 %	93.00 $\pm$ 0.11 %	69.58 $\pm$ 0.15 %
PMF	98.80 $\pm$ 0.06 %	91.43 $\pm$ 0.14 %	70.45 $\pm$ 0.25 %
BayesBiNN (mode)	**98.86** $\pm$ 0.05 %	**93.72** $\pm$ 0.16 %	**73.68** $\pm$ 0.31 %
BayesBiNN (mean)	**98.86** $\pm$ 0.05 %	**93.72** $\pm$ 0.15 %	**73.65** $\pm$ 0.41 %
Full-precision	99.01 $\pm$ 0.06 %	93.90 $\pm$ 0.17 %	74.83 $\pm$ 0.26 %

are provided in the Appendix B.1.

4.2. Image Classification on Real Datasets

We now present results on three benchmark real datasets widely used for image classification: MNIST (LeCun & Cortes, 2010), CIFAR-10 (Krizhevsky & Hinton, 2009) and CIFAR-100 (Krizhevsky & Hinton, 2009). We compare to three other optimizers[2]: STE (Bengio et al., 2013) using Adam with weight clipping and gradient clipping (Courbariaux et al., 2015; Maddison et al., 2017; Alizadeh et al., 2019); latent-free Bop (Helwegen et al., 2019); and the proximal mean-field (PMF) (Ajanthan et al., 2019a). An additional comparison with the LR-net method of Shayer et al. (2018) is given in Appendix B.3. For a fair comparison, we keep all conditions the same except for the optimization methods themselves. For our proposed BayesBiNN, we report results using both the mode and the mean. For all the experiments, standard categorical cross-entropy loss is used and we take 10% of the training set for validation and report the best accuracy on the test set corresponding to the highest validation accuracy achieved during training.

For MNIST, we use a multilayer perceptron (MLP) with

three hidden layers with 2048 units and rectified linear units (ReLU) (Alizadeh et al., 2019) activations. Both Batch normalization (BN)[3] (Ioffe & Szegedy, 2015) and dropout are used. No data augmentation is performed. For CIFAR-10 and CIFAR-100, we use the BinaryConnect CNN network in Alizadeh et al. (2019), which is a VGG-like structure similar to the one used in Helwegen et al. (2019). Standard data augmentation is used (Graham, 2014), where 4 pixels are padded on each side, a random 32×32 crop is applied, followed by a random horizontal flip. Note that no ZCA whitening is used as in Courbariaux et al. (2015); Alizadeh et al. (2019). The details of the experimental setting, including the detailed network architecture and values of all hyper-parameters, are provided in Appendix B.2 in the supplementary material.

As shown in Table 2, the proposed BayesBiNN achieves similar performances (slightly better for CIFAR-100) as STE Adam in all the three benchmark datasets and approaches the performance of full-precision DNNs. The detailed results, such as the train/validation accuracy as well as the training curves are provided in Appendix B.2 in the supplementary material.

---

[2]We use Bop code available at `https://github.com/plumerai/rethinking-bnn-optimization` and PMF code available at `https://github.com/tajanthan/pmf`.

[3]Here the parameters of BN layers are not learned. However, they could also be learned by applying a conventional optimizer such as Adam separately, which is easy to implement.
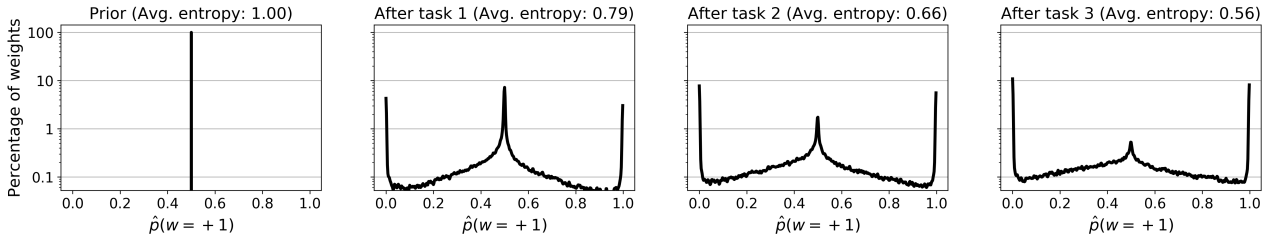
Figure 4. Evolution of the distribution over the binary weights during the learning process in continual learning. The histogram of the weight probabilities ($\hat{p}(w_j = +1)$ for all j) is shown after learning on different tasks. At the very beginning, all the weights are equal to -1 or +1 with prior probability 0.5 and thus have maximum average entropy 1.0. As the number of learned tasks increases, the distribution spreads and becomes flatter, implying that the average entropy of the binary weights decreases, i.e., the weights of BiNNs become more and more deterministic.

4.3. Continual learning with binary neural networks

We now show an application of BayesBiNN to continual learning. We consider the popular benchmark of permuted MNIST (Goodfellow et al., 2013; Kirkpatrick et al., 2016; Nguyen et al., 2018; Zenke et al., 2017), where each dataset $\mathcal{D}_t$ consists of labeled MNIST images whose pixels are permuted randomly. Similar to Nguyen et al. (2018), we use a fully connected single-head network with two hidden layers containing 100 hidden units with ReLu activations. No coresets are used. The details of the experiment, e.g., the network architecture and values of hyper-parameters, are provided in Appendix B.4.

As shown in Figure 5, using BayesBiNN with the posterior approximation $q_{t-1}(\boldsymbol{w})$ as the prior for task t (red solid line), we achieve significant improvements in overcoming catastrophic forgetting of the past. When the prior is fixed to be $\boldsymbol{\lambda}_0 = \boldsymbol{0}$ (blue dotted line), the network performs badly on the past tasks, e.g., in the top row, after trained on task 2 and task 3, the network performs badly on the previous task 1. The reason for better performance when using the posterior approximation $q_{t-1}(\boldsymbol{w})$ as the prior for task t is directly related to the uncertainty estimated by the posterior approximation $q_{t-1}(\boldsymbol{w})$. To visualize the uncertainty, Figure 4 shows the histogram of the weight probabilities ($\hat{p}(w_j = +1)$ for all j). The prior probability, shown in the first plot, is set to $\frac{1}{2}$ for all weights (entropy is 1.0). As we train on more tasks, the uncertainty decreases and the weights of BiNNs become more and more deterministic (the distribution spreads and becomes flatter). As desired, with more data, the network reduces the entropy of the distribution and the uncertainty is useful to perform continual learning.

5. Conclusion

Binary neural networks (BiNNs) are computation-efficient and hardware-friendly, but their training is challenging since in theory it involves a discrete optimization problem. How-
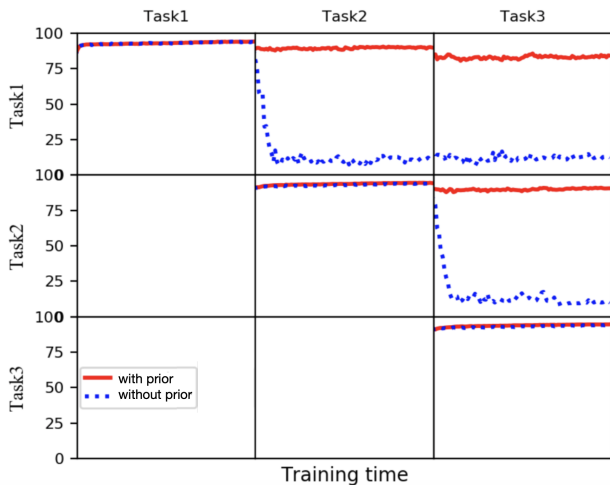

Figure 5. Test accuracy curves of continual learning for BiNNs using BayesBiNN on permuted MNIST. The neural network is trained with or without prior for BayesBiNN, respectively. Specifically, with prior (red solid line) indicates using BayesBiNN with the posterior approximation $q_{t-1}(\boldsymbol{w})$ as the prior for task t while without prior (blue dotted line) indicates using BayesBiNN with fixed prior where $\boldsymbol{\lambda}_0 = \boldsymbol{0}$. The test accuracy on the test set is averaged over 5 random runs. The X-axis shows the training time (epochs) and Y-axis shows the average test accuracy of different tasks as the training time increases.

ever, some gradient-based methods such as the STE work quite well in practice despite ignoring the discrete nature of the optimization problem, which is surprising and there is a lack of principled justification of their success. In this paper, we proposed a principled approach to train the binary neural networks using the Bayesian learning rule. The resulting optimizer, which we call BayesBiNN, not only justifies some of the algorithmic choices made by existing methods such as the STE and Bop but also facilitates the extensions of them, e.g., enabling uncertainty estimation for continual learning to avoid the catastrophic forgetting problem for binary neural networks.

References

Ajanthan, T., Dokania, P. K., Hartley, R., and Torr, P. H. S. Proximal mean-field for neural network quantization. In *IEEE International Conference on Computer Vision*, pp. 4871–4880, 2019a.

Ajanthan, T., Gupta, K., Torr, P. H., Hartley, R., and Dokania, P. K. Mirror descent view for neural network quantization. *arXiv preprint arXiv:1910.08237*, 2019b.

Alizadeh, M., Fernández-Marqués, J., Lane, N. D., and Gal, Y. An empirical study of binary neural networks' optimisation. *ICLR*, 2019.

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bethge, J., Bartz, C., Yang, H., Chen, Y., and Meinel, C. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936*, 2020.

Bissiri, P. G., Holmes, C. C., and Walker, S. G. A general framework for updating belief distributions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):1103–1130, 2016.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Gardner Jr, E. S. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1):1–28, 1985.

Geman, S. and Geman, D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *ICLR*, 2013.

Graham, B. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014.

Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.-T., and Nusselder, R. Latent weights do not exist: Rethinking binarized neural network optimization. *NeurIPS*, 2019.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Huning, A. Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution, 1976.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. *ICLR*, 2017.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Khan, M. E. and Lin, W. Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. *AISTATS*, 2017.

Khan, M. E. and Rue, H. Learning-Algorithms from Bayesian principles. 2020. https://emtiyaz.github.io/papers/learning_from_bayes.pdf.

Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. *ICML*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2015.

Kirkpatrick, J. N., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114 13:3521–3526, 2016.

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

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Lemaréchal, C. Nondifferentiable optimization. *Handbooks in operations research and management science*, 1:529–572, 1989.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 345–353, 2017.

Louizos, C., Reisser, M., Blankevoort, T., Gavves, E., and Welling, M. Relaxed quantization for discretized neural networks. *ICLR*, 2019.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017.

Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. WRPN: wide reduced-precision networks. *ICLR*, 2018.

Moons, T. Two moons datasets description. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. *ICLR*, 2018.

Osawa, K., Swaroop, S., Khan, M. E. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with Bayesian principles. In *Advances in Neural Information Processing Systems*, pp. 4289–4301, 2019.

Peters, J. W. and Welling, M. Probabilistic binary neural networks. *arXiv preprint arXiv:1809.03368*, 2018.

Raskutti, G. and Mukherjee, S. The information geometry of mirror descent. *IEEE Transactions on Information Theory*, 61(3):1451–1457, 2015.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Shayer, O., Levi, D., and Fetaya, E. Learning discrete weights using the local reparameterization trick. *ICLR*, 2018.

Snelson, E. and Ghahramani, Z. Sparse Gaussian processes using pseudo-inputs. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pp. 1257–1264, Cambridge, MA, USA, 2005. MIT Press.

Staines, J. and Barber, D. Variational optimization. *arXiv preprint arXiv:1212.4507v2*, 2012.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1–2:1–305, 2008.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., and Xin, J. Understanding straight-through estimator in training activation quantized neural nets. *ICLR*, 2019.

Zellner, A. Optimal information processing and bayes's theorem. *The American Statistician*, 42(4):278–280, 1988.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3987–3995. JMLR. org, 2017.