Appendix

A. Two equivalent forms of hysteresis function in Bop

The original update rule and the corresponding definition of the *hysteresis* function hyst$(\cdot)$ in Bop is (Helwegen et al., 2019)

$$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r + \alpha \mathbf{g} \tag{22}$$

$$\begin{aligned} y1 &= \text{hyst1}\left(w_r, w_b, \gamma\right) \\ &\equiv \begin{cases} -w_b & if\ |w_r| > \gamma \ \&\ \text{sign}(w_r) = \text{sign}(w_b) \\ w_b & otherwise, \end{cases} \end{aligned} \tag{23}$$

One could however also modify the update rule to $\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{g}$, as shown in Step 3 of Bop in Table 1. In this case, the update rule and the corresponding *hysteresis* function becomes

$$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{g} \tag{24}$$

$$\begin{aligned} y2 &= \text{hyst2}\left(w_r, w_b, \gamma\right) \\ &\equiv \begin{cases} -w_b & if\ |w_r| > \gamma \ \&\ \text{sign}(w_r) = \text{-sign}(w_b) \\ w_b & otherwise, \end{cases} \end{aligned} \tag{25}$$

It could be easily seen that these two update equations with two representations of the *hysteresis* function are equivalent to each other. The rightmost figure in Figure 1 (b) shows the curve of $y2 = \text{hyst2}\left(w_r, w_b, \gamma\right)$. The corresponding curve of $y1 = \text{hyst1}\left(w_r, w_b, \gamma\right)$ is simply a upside-down flipped version of the rightmost figure in Figure 1 (b).

B. Experimental details

In this section we list the details for all experiments shown in the main text.

Note that after training BiNNs with BayesBiNN, there are two ways to perform inference during test time:

(1). **Mean**: One method is to use the predictive mean, where we use Monte Carlo sampling to compute the predictive probabilities for each test sample $\boldsymbol{x}_j$ as follows

$$\hat{p}_{j,k} \approx \frac{1}{C}\sum_{c=1}^{C} p\left(y_j = k | \boldsymbol{x}_j, \boldsymbol{w}^{(c)}\right), \tag{26}$$

where $\boldsymbol{w}^{(c)} \sim q(\boldsymbol{w})$ are samples from the Bernoulli distributions with the natural parameters $\boldsymbol{\lambda}$ obtained by BayesBiNN.

(2). **Mode**: The other way is simply to use the mode of the posterior distribution $q(\boldsymbol{w})$, i.e., the sign value of the posterior mean, i.e., $\hat{\boldsymbol{w}} = \text{sign}(\tanh\left(\boldsymbol{\lambda}\right))$, to make predictions, which will be denoted as $C = 0$.

B.1. Synthetic Data

Binary Classification We used the Two Moons dataset with 100 data points in each class and added Gaussian noise with standard deviation 0.1 to each point. We trained a Multilayer Perceptron (MLP) with two hidden layers of 64 units and tanh activation functions for 3000 epochs, using Cross Entropy as the loss function. Additional train and test settings with respect to the optimizers are detailed in Table 3. The learning rate α was decayed at fixed epochs by the specified learning rate decay rate. For the STE baseline, we used the Adam optimizer with standard settings.

Table 3. Train settings for the binary classification experiment using the Two Moons dataset.

Setting	BayesBiNN	STE
Learning rate α	10^{-3}	10^{-1}
Learning rate decay	0.1	0.1
Learning rate decay epochs	[1500, 2500]	[1500, 2500]
Momentum(s) β	0.99	0.9, 0.999
MC train samples S	5	-
MC test samples C	0/10	-
Temperature τ	1	-
Prior $\boldsymbol{\lambda}_0$	$\mathbf{0}$	-
Initialization λ	± 15 randomly	-

Regression We used the Snelson dataset (Snelson & Ghahramani, 2005) with 200 data points to train a regression model. Similar to the Binary Classification experiment, we used a MLP with two hidden layers of 64 units and tanh activation functions, but trained it for 5000 epochs using Mean Squared Error as the loss function. Additionally, we added a batch normalization layer (without learned gain or bias terms) after the last fully connected layer. The learning rate is adjusted after every epoch to slowly anneal from an initial learning rate α_0 to a target learning rate α_T at the maximum epoch T using

$$\alpha_{t+1} = \alpha_t \left(\frac{\alpha_T}{\alpha_0}\right)^{-T}. \tag{27}$$

The learning rates and other train and test settings are detailed in Table 4.

Table 4. Train settings for the regression experiment using the Snelson dataset (Snelson & Ghahramani, 2005).

Setting	BayesBiNN	STE
Learning rate start α_0	10^{-4}	10^{-1}
Learning rate end α_T	10^{-5}	10^{-1}
Momentum(s) β	0.99	0.9, 0.999
MC train samples S	1	-
MC test samples C	0/10	-
Temperature τ	1	-
Prior $\boldsymbol{\lambda}_0$	$\mathbf{0}$	-
Initialization $\boldsymbol{\lambda}$	± 10 randomly	-

B.2. MNIST, CIFAR-10 and CIFAR-100

In this section, three well-known image datasets are considered, namely the MNIST, CIFAR-10 and CIFAR-100 datasets. We compare the proposed BayesBiNN with four other popular algorithms, STE Adam, Bop and PMF for BiNNs as well as standard Adam for full-precision weights. For dataset and algorithm specific settings, see Table 9.

MNIST All algorithms have been trained using the same MLP detailed in Table 5 on mini-batches of size 100, for a maximum of 500 epochs. The loss used was Categorical Cross Entropy. We split the original training data into 90% train and 10% validation data and no data augmentation except normalization has been done. We report the best accuracy (averaged over 5 random runs) on the test set corresponding to the highest validation accuracy achieved during training (we do not retrain using the validation set). Note that we tune the hyper-parameters such as learning rate for all the methods including the baselines. The search space for the learning rate is set to be $\left[10^{-2}, 3\cdot10^{-3}, 10^{-3}, 3\cdot10^{-4}, 10^{-4}, 3\cdot10^{-5}, 10^{-5}, 10^{-6}\right]$ for all methods. Moreover, Table 6 and Table 7 shows the results of MNIST with BayesBiNN for different choices of learning rate and temperature.

CIFAR-10 and CIFAR-100 We trained all algorithms on the Convolutional Neural Network (CNN) architecture detailed in Table 8 on mini-batches of size 50, for a maximum of 500 epochs. The loss used was Categorical Cross Entropy. We split

Table 5. The MLP architecture used in all MNIST experiments, adapted from (Alizadeh et al., 2019).

Dropout (p = 0.2)
Fully Connected Layer (units = 2048, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Dropout (p = 0.2)
Fully Connected Layer (units = 2048, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Dropout (p = 0.2)
Fully Connected Layer (units = 2048, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Dropout (p = 0.2)
Fully Connected Layer (units = 2048, bias = False)
Batch Normalization Layer (gain = 1, bias = 0)
Softmax

Table 6. Test accuracy of MNIST for different initial learning rates. The temperature is 10^{-10}. Results are averaged over 5 random runs.

Learning rate	10^{-1}	$3 \cdot 10^{-3}$	10^{-3}	$3 \cdot 10^{-4}$
Training Accuracy	99.46 ± 0.15 %	99.58 ± 0.16 %	99.67 ± 0.09 %	99.76 ± 0.09 %
Validation Accuracy	98.90 ± 0.14 %	98.94 ± 0.17 %	98.96 ± 0.13 %	98.97 ± 0.12 %
Test Accuracy	98.73 ± 0.11 %	98.81 ± 0.07 %	98.83 ± 0.05 %	98.84 ± 0.08 %
Learning rate	10^{-4}	$3 \cdot 10^{-5}$	10^{-5}	10^{-6}
Training Accuracy	99.85 ± 0.05 %	99.83 ± 0.06 %	99.76 ± 0.09 %	99.78 ± 0.03 %
Validation Accuracy	99.02 ± 0.13 %	99.02 ± 0.13 %	99.04 ± 0.11 %	99.02 ± 0.17 %
Test Accuracy	98.86 ± 0.05 %	98.86 ± 0.05 %	98.84 ± 0.08 %	98.85 ± 0.05 %

the original training data into 90% train and 10% validation data. For data augmentation during training, the images were normalized, a random 32×32 crop was selected from a 40×40 padded image and finally a random horizontal flip was applied. In the same manner as Osawa et al. (2019), we consider such data augmentation as effectively increasing the dataset size by a factor of 10 (4 images for each corner, and one central image, and the horizontal flipping step further doubles the dataset size, which gives a total factor of 10). We report the best accuracy (averaged over 5 random runs) on the test set corresponding to the highest validation accuracy achieved during training. In addition, we tune the hyper-parameters, such as the learning rate, for all the methods including the baselines. The search space for the learning rate is set to be $\left[10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}, 3 \cdot 10^{-5}, 10^{-5}, 10^{-6}\right]$ for all methods.

B.3. Comparison with LR-net

We also compare the proposed BayesBiNN with the LR-net method in Shayer et al. (2018) for MNIST and CIFAR-10. As the code for the LR-net is not open-source, we performed experiments with BayesBiNN following the same experimental settings in Shayer et al. (2018) and then compared the results with the reported results in their paper. In specific, the network architectures for MNIST and CIFAR-10 are the same as Shayer et al. (2018), except that we added BN after the FC layers. However, we kept all layers binary and did not learn the BN parameters, nor did we use dropout as in Shayer et al. (2018). The dataset pre-processing follows the same settings in Shayer et al. (2018) and is similar to that described in subsection 4.2, except that there is no split of the training set into training and validation sets. As a result, as in Shayer et al.

Table 7. Test accuracy of MNIST for different temperatures. The initial learning rate is 10^{-4}. Results are averaged over 5 random runs.

Temperature	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
Training Accuracy	89.25 ± 0.22 %	87.55 ± 0.50 %	90.22 ± 0.42 %	97.37 ± 0.13 %	98.27 ± 0.10 %
Validation Accuracy	90.06 ± 1.04 %	90.28 ± 0.43 %	93.35 ± 0.48 %	98.10 ± 0.17 %	98.55 ± 0.16 %
Test Accuracy	90.40 ± 0.97 %	90.72 ± 0.42 %	93.67 ± 0.50 %	98.01 ± 0.05 %	98.41 ± 0.10 %
Learning rate	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
Training Accuracy	99.48 ± 0.08 %	99.75 ± 0.14 %	99.85 ± 0.05 %	99.81 ± 0.04 %	99.82 ± 0.07 %
Validation Accuracy	98.92 ± 0.13 %	99.00 ± 0.13 %	99.02 ± 0.14 %	99.02 ± 0.12 %	99.02 ± 0.13 %
Test Accuracy	98.82 ± 0.05 %	98.81 ± 0.08 %	98.86 ± 0.05 %	98.86 ± 0.06 %	98.84 ± 0.04 %

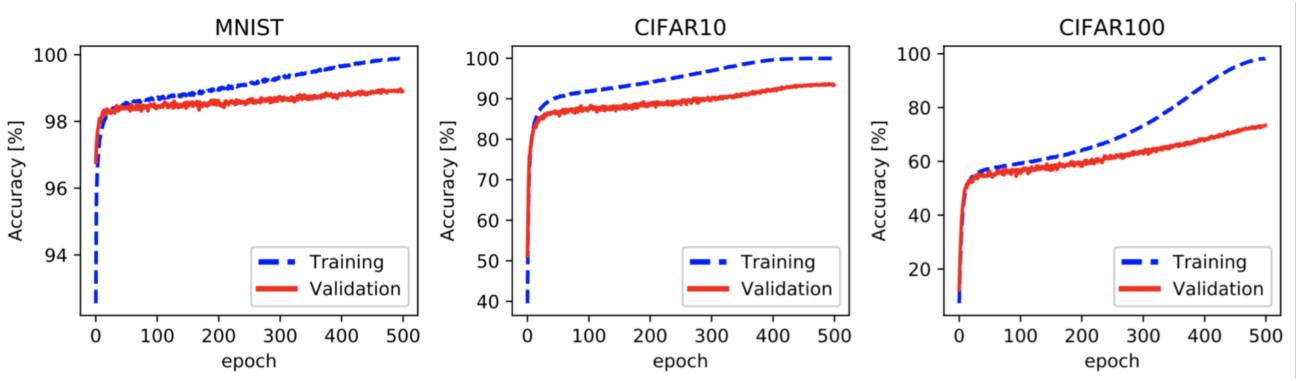

Figure 6. Training/Validation accuracy for MNIST, CIFAR-10 and CIFAR100 with BayesBiNN optimizer (Averaged over 5 runs).

(2018), we report the test accuracies after 190 epochs and 290 epochs for MNIST and CIFAR-10, respectively. Note that the hyper-parameter settings of BayesBiNN are the same as those in Table 9 for MNIST and CIFAR-10. The results are shown in Table 11. The proposed BayesBiNN achieves similar performance (slightly better for CIFAR-10) to the LR-net. Note that the LR-net method used pre-trained models to initialize the weights of BiNNs, while BayesBiNN trained BiNNs from scratch without using pre-trained models.

B.4. Continual learning with binary neural networks

For the continual learning experiment, we used a three-layer MLP, detailed in Table 12, and trained it using the Categorical Cross Entropy loss. Specific training parameters are given in Table 13. There is no split of the original MNIST training data in the continual learning case. No data augmentation except normalization has been performed.

C. Author Contributions Statement

M.E.K. conceived the idea of training Binary neural networks using the Bayesian learning rule. X.M. derived the BayesBiNN algorithm, studied its connections to STE and Bop, and wrote the first proof-of-concept experiments. R.B. fixed a few issue with the original implementation and re-organized the PyTorch code. R.B. also designed and performed the experiments on synthetic data presented in Section 4.1. X.M. did most of the experiments with some help from R.B. All the authors were involved in writing, revising and proof-reading the paper.

Table 8. The CNN architecture used in all CIFAR-10 and CIFAR-100 experiments, inspired by VGG and used in Alizadeh et al. (2019).

Convolutional Layer (channels = 128, kernel-size = 3×3, bias = False, padding = same)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Convolutional Layer (channels = 128, kernel-size = 3×3, bias = False, padding = same)
ReLU
Max Pooling Layer (size = 2×2, stride = 2×2)
Batch Normalization Layer (gain = 1, bias = 0)
Convolutional Layer (channels = 256, kernel-size = 3×3, bias = False, padding = same)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Convolutional Layer (channels = 256, kernel-size = 3×3, bias = False, padding = same)
ReLU
Max Pooling Layer (size = 2×2, stride = 2×2)
Batch Normalization Layer (gain = 1, bias = 0)
Convolutional Layer (channels = 512, kernel-size = 3×3, bias = False, padding = same)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Convolutional Layer (channels = 512, kernel-size = 3×3, bias = False, padding = same)
ReLU
Max Pooling Layer (size = 2×2, stride = 2×2)
Batch Normalization Layer (gain = 1, bias = 0)
Fully Connected Layer (units = 1024, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Fully Connected Layer (units = 1024, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Fully Connected Layer (units = 1024, bias = False)
Batch Normalization Layer (gain = 1, bias = 0)
Softmax

Table 9. Algorithm specific train settings for MNIST, CIFAR-10, and CIFAR-100.

Algorithm	**Setting**	**MNIST**	**CIFAR-10**	**CIFAR-100**
BayesBiNN	Learning rate start α_0	10^{-4}	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
	Learning rate end α_T	10^{-16}	10^{-16}	10^{-16}
	Learning rate decay	Cosine	Cosine	Cosine
	MC train samples S	1	1	1
	MC test samples C	0	0	0
	Temperature τ	10^{-10}	10^{-10}	10^{-8}
	Prior $\boldsymbol{\lambda}_0$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
	Initialization $\boldsymbol{\lambda}$	± 10 randomly	± 10 randomly	± 10 randomly
STE Adam	Learning rate start α_0	10^{-2}	10^{-2}	10^{-2}
	Learning rate end α_T	10^{-16}	10^{-16}	10^{-16}
	Learning rate decay	Cosine	Cosine	Cosine
	Gradient clipping	Yes	Yes	Yes
	Weights clipping	Yes	Yes	Yes
Bop	Threshold τ	10^{-8}	10^{-8}	10^{-9}
	Adaptivity rate γ	10^{-5}	10^{-4}	10^{-4}
	γ-decay type	Step	Step	Step
	γ-decay rate	$10^{\frac{-3}{500}}$	0.1	0.1
	γ-decay interval (epochs)	1	100	100
PMF	Learning rate start	10^{-3}	10^{-2}	10^{-2}
	Learning rate decay type	Step	Step	Step
	LR decay interval (iterations)	7k	30k	30k
	LR-scale	0.2	0.2	0.2
	Optimizer	Adam	Adam	Adam
	Weight decay	0	10^{-4}	10^{-4}
	ρ	1.2	1.05	1.05
Adam (Full-precision)	Learning rate start α_0	$3 \cdot 10^{-4}$	10^{-2}	$3 \cdot 10^{-3}$
	Learning rate end α_T	10^{-16}	10^{-16}	10^{-16}
	Learning rate decay	Cosine	Cosine	Cosine

Table 10. Detailed results of different optimizers trained on MNIST, CIFAR-10 and CIFAR-100 (Averaged over 5 runs).

Dataset	Optimizer	Train Accuracy	Validation Accuracy	Test Accuracy
MNIST	STE Adam	99.78 ± 0.10 %	99.02 ± 0.11 %	$\mathbf{98.85} \pm 0.09$ %
	Bop	99.23 ± 0.04 %	98.55 ± 0.05 %	98.47 ± 0.02 %
	PMF		99.06 ± 0.01 %	98.80 ± 0.06 %
	BayesBiNN (mode)	99.85 ± 0.05 %	99.02 ± 0.13 %	$\mathbf{98.86} \pm 0.05$ %
	BayesBiNN (mean)	99.85 ± 0.05 %	99.02 ± 0.13 %	$\mathbf{98.86} \pm 0.05$ %
	Full-precision	99.96 ± 0.02 %	99.15 ± 0.14 %	99.01 ± 0.06 %
CIFAR-10	STE Adam	99.99 ± 0.01 %	94.25 ± 0.42 %	$\mathbf{93.55} \pm 0.15$ %
	Bop	99.79 ± 0.03 %	93.49 ± 0.17 %	93.00 ± 0.11 %
	PMF		91.87 ± 0.10 %	91.43 ± 0.14 %
	BayesBiNN (mode)	99.96 ± 0.01 %	94.23 ± 0.41 %	$\mathbf{93.72} \pm 0.16$ %
	BayesBiNN (mean)	99.96 ± 0.01 %	94.23 ± 0.41 %	$\mathbf{93.72} \pm 0.15$ %
	Full-precision	100.00 ± 0.00 %	94.54 ± 0.29 %	93.90 ± 0.17 %
CIFAR-100	STE Adam	99.06 ± 0.15 %	74.09 ± 0.15 %	72.89 ± 0.21 %
	Bop	90.09 ± 0.57 %	69.97 ± 0.29 %	69.58 ± 0.15 %
	PMF		69.86 ± 0.08 %	70.45 ± 0.25 %
	BayesBiNN (mode)	98.02 ± 0.18 %	74.76 ± 0.41 %	$\mathbf{73.68} \pm 0.31$ %
	BayesBiNN (mean)	98.02 ± 0.18 %	74.76 ± 0.41 %	$\mathbf{73.65} \pm 0.41$ %
	Full-precision	99.89 ± 0.02 %	75.89 ± 0.41 %	74.83 ± 0.26 %

Table 11. Test accuracy of BayesBiNN and LR-net trained on MNIST, CIFAR-10. Results for BayesBiNN are averaged over 5 random runs.

Optimizer	MNIST	CIFAR-10
LR-net Shayer et al. (2018)	99.47 %	93.18%
BayesBiNN (mode)	99.50 ± 0.02 %	93.97 ± 0.11 %

Table 12. The MLP architecture used for continual learning (Nguyen et al., 2018)

Fully Connected Layer (units = 100, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Fully Connected Layer (units = 100, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Fully Connected Layer (units = 100, bias = False)
ReLU
Batch Normalization Layer (gain = 1, bias = 0)
Softmax

Table 13. Algorithm specific train settings for continual learning on permuted MNIST.

Algorithm	**Setting**	**Permuted MNIST**
BayesBiNN	Learning rate start α_0	10^{-3}
	Learning rate end α_T	10^{-16}
	Learning rate decay	Cosine
	MC train samples S	1
	MC test samples C	100
	Temperature τ	10^{-2}
	Prior $\boldsymbol{\lambda}_0$	learned $\boldsymbol{\lambda}$ of the previous task
	Initialization $\boldsymbol{\lambda}$	± 10 randomly
	Batch size	100
	Number of epochs	100