UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2020

[Re] Training Binary Neural Networks using the Bayesian Learning Rule

Prateek Garg[1, ID], Lakshya Singhal[1, ID], and Ashish Sardana[2]
[1]Indian Institute of Technology Delhi, New Delhi, India – [2]NVIDIA, Bangalore, KA, India

Edited by
(Editor)

Received
01 November 2018

Published
–

DOI
–

Reproducibility Summary

Meng, Bachmann, and Khan[1] gives a mathematically principled approach to solve the discrete optimization problem that occurs in the case of Binary Neural Networks and claims to give a similar performance on various classification benchmarks such as MNIST, CIFAR-10, and CIFAR-100 as compared to their full-precision counterparts, as well as other recent algorithms to train BNNs like PMF and Bop. The paper also claims that the BayesBiNN method has an application in the continual learning domain as it helps in overcoming catastrophic forgetting of the past by using the posterior approximation of the previous task as a prior for the upcoming task. We try to reproduce all the results presented in the original paper by making a separate and independent codebase.

Scope of Reproducibility

We try to verify the performance of our re-implementation of the BayesBiNN optimizer on various classification and regression benchmarks. We also implemented the STE optimizer which was the central baseline model used in the paper. Finally, we tried to evaluate the results of BayesBiNN on the continual learning benchmark to get a better insight.

Methodology

We developed our separate code-base, consisting of an end-to-end trainer with a Keras-like interface, for the reproduction which includes the implementation of the Bayes-BiNN and STE optimizer. We did refer to the author's code open-sourced on GitHub to get some insights about the hyperparameters and other doubts that emerged during code development.

Results

We reproduced the accuracy of the BayesBiNN optimizer within less than 0.5% of the originally reported value, which upholds the conclusion that it performs nearly as well as its full-precision counterpart in classification tasks. When we tried this in a semantic segmentation context, we found that the results were very underwhelming and in

Correspondence should be addressed to Prateek Garg (prateekgarg.iitd@gmail.com)
The authors have declared that no competing interests exist.
Code is available at https://github.com/prateekstark/training-binary-neural-network – DOI 10.5281/zenodo.4716863. – SWH swh:1:dir:904c01b8e4ac0d4acd1f5fe667511d5a9234d8da.
Open peer review is available at https://openreview.net/forum?id=bhiGno-Cxq.

contrast with the seemingly good results by the STE optimizer even with much hyper-parameter tuning. We can conclude that, like other Bayesian methods, it is difficult to train BayesBiNN on more complex tasks.

What was easy

After we worked out the mathematics behind the BayesBiNN approach, we developed a pseudo-code for the optimization process which along with references from the author's code, helped us a lot in our reproduction study.

What was difficult

Some of the hyperparameters were not mentioned by the authors in their paper so it was difficult to approximate the values of those parameters. The lack of resources was the next big difficulty that we faced.

Communication with original authors

We had a very fruitful conversation with the authors, which helped us in better understanding the BayesBiNN approach and its extension to the segmentation domain. The detailed pointers are given at the end of this report.

1 Introduction

Deep Learning is moving towards larger and larger parameters day-by-day, which often makes it difficult to run on resource-constraint devices like mobile phones. Binary Neural Networks (BNNs) could act as a savior in such situations, helping in largely saving storage and computational costs. The problem of optimizing this binary set of weights is clearly a discrete optimization problem. Previous approaches like Straight-Through Estimator (STE) and Binary Optimizer (Bop) tend to ignore this and use gradient-based methods, which still worked in practice. The paper presents a mathematically principled approach for training BNNs which also justifies the current approaches.

2 Scope of reproducibility

The paper mentions a bayesian approach to solve the discrete optimization problem in the case of Binary Neural Networks (BNNs). The outcome of this approach was a BayesBiNN optimizer which could be used to train BNNs and achieve similar accuracy as compared to their full-precision counterparts. To verify the claims given in the paper, we target to achieve the following objectives:

- Work out and present the mathematics behind BayesBiNN in a simpler way and prepare a pseudo-code to the optimizer.
- Implement the BayesBiNN optimizer and STE optimizer to verify the accuracy on tasks of varying domains, as reported in the original paper.
- Reproduce the results for other baselines present in the paper such as proximal mean-field (PMF) according to the hyper-parameters given in the paper.
- Evaluate the performance of BayesBiNN optimizer in more complex domains like semantic segmentation.

3 Methodology

We have re-implemented the algorithm proposed in the paper from scratch using PyTorch and created an end-to-end model trainer with a Keras-like interface. We referred to the code given by the authors for the baseline model hyperparameters and the source of synthetic datasets. The algorithm presented by the original authors in their paper can be represented as follows:

Algorithm 1: Bayesian Learning rule for BayesBiNN

Input: Input: Initialize λ

for *number of training epochs* **do**
  for *i = 1,...,number of mini-batch examples* **do**
    Sample $\epsilon \sim \mathcal{U}(0,1)$ and set $\delta = \frac{1}{2}\log\frac{\epsilon}{1-\epsilon}$
    Initialize $w_b = \tanh((\lambda + \delta)/\tau)$
    Compute following using *gumbel-softmax* trick

$$g_i := \frac{1}{M}\nabla_{w_b} l(y_i, f_{w_r}(x_i))$$

$$s_i := \frac{N(1-w_b^2)}{\tau(1-\tanh(\lambda)^2)}$$

  end
  Update μ and λ using following equation

$$\mu \leftarrow \tanh(\lambda)$$

$$\lambda \leftarrow (1-\alpha)\lambda - \alpha[\sum_{i=1}^{M}(s_i \odot g_i) - \lambda_0]$$

end

This would make the paper easier to interpret and this implementation on code. Some of the mathematical expressions mentioned in the original paper were presented from various sources and missed out several intermediate steps which we found to be very important while reproducing the paper from scratch. Here we present a step-wise derivation of some important expressions written in the original paper:
Bayesian formulation of the discrete optimization problem, in which loss has to be minimized w.r.t posterior $q(w)$, given prior $p(w)$ can be written as:

$$\mathbb{E}_{q(w)}[\sum_{i=1}^{N} l(y_i, f_w(x_i))] + \mathcal{D}_{KL}[q(w)\|p(w)]$$

To solve the above optimization problem, Bayesian learning rule given in Khan and Rue[2] is applied, assuming solution to be a part of minimal exponential family of distribution, given by:

$$q(w) = h(w)\exp[\lambda^T\phi(w) - A(\lambda)]$$

where base measure $h(w)$ is assumed to be 1. Following is the update rule used to learn λ:

$$\lambda \leftarrow (1-\rho)\lambda - \rho[\nabla_\mu \mathbf{E}_{q(w)}[l(y_i f_w(x_i))] - \lambda_0]$$

where ρ is the learning rate, $\mu = \mathbf{E}_{q(w)}[\phi(w)]$. Bernoulli distribution being a special case of minimal exponential family distribution, we assume prior $p(w) \sim \text{Bern}(p)$ with

$p = 0.5$, and posterior $q(w)$ to be mean-field Bernoulli distribution:

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

For weight j,

$$\begin{aligned} q(w_j) &= \exp(\frac{1}{2}(1+w_j)\log p_j + \frac{1}{2}(1-w_j)\log(1-p_j)) \\ &= \exp(\underbrace{w_j}_{\phi(w)} \underbrace{\frac{1}{2}\log\frac{p}{1-p}}_{\lambda}) + \frac{1}{2}\log(p(1-p)) \end{aligned}$$

Comparing above expression with minimal exponential family distribution, we can say:

$$\lambda = \frac{1}{2}\log\frac{p}{1-p} \text{ and } \phi(w) = w.$$

We defined $\mu = \mathbb{E}_{q(w)}[\phi(w)]$,

$$\begin{aligned} \mu = \int wq(w)dw = \mathbb{E}[q(w)] &= \sum_{w^i \in \{-1,1\}} w^i q(w^i) \\ = \sum_{w^i \in \{-1,1\}} w^i p^{\frac{1+w^i}{2}} (1-p)^{\frac{1-w^i}{2}} &= -(1-p) + p \\ &= 2p - 1 \end{aligned}$$

From above derivations we can say that, $p = 1/(1+\exp(-2\lambda)) = \text{Sigmoid}(2\lambda)$ and $q(w) \sim \text{Bern}(p)$.
To implement the update rule, we need to compute the gradient with respect to μ. Original paper uses a reparamaterization trick called gumbel-softmax trick Maddison, Mnih, and Teh[3], which is used to relax the discrete random variables of a concrete distribution (for eg, bernoulli distribution). Binary concrete relaxation Maddison, Mnih, and Teh[3] of binary concrete random variable $X \in (0,1)$ with distribution $X \sim \text{BinConcrete}(\alpha, \lambda)$ with temperature λ and location α,

$$X = \frac{1}{1+\exp(-(\log\alpha + L)/\lambda)}$$

where $L \sim$ Logistic. And its density is given by

$$p_{\alpha,\lambda}(x) = \frac{\lambda\alpha x^{-\lambda-1}(1-x)^{-\lambda-1}}{(\alpha x^{-\lambda} + (1-x)^{-\lambda})^2}$$

Using above expressions, for binary weights $w_j \in \{0,1\}$, relaxed variable $w_r^{\epsilon_j,\tau}(p_j) \in (0,1)$ can be used with temperature τ and $\alpha = e^{2\lambda}$ given by

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

where $\delta_j \sim$ Logistic and its density is given by

$$p(w_r^{\epsilon_j,\tau}(p_j)) = \frac{\tau e^{2\lambda} w_r^{\epsilon_j,\tau}(p_j)^{-\tau-1}(1-w_r^{\epsilon_j,\tau}(p_j))^{-\tau-1}}{(e^{2\lambda} w_r^{\epsilon_j,\tau}(p_j)^{-\tau} + (1-w_r^{\epsilon_j,\tau}(p_j))^{-\tau})^2}$$

4 Experimental setup

4.1 Model descriptions

We kept the model architectures the same as mentioned in the original paper to maintain uniformity and implemented them ourselves. For the MNIST classification task we used the BinaryConnect architecture and for the CIFAR classification task we used the VGGBinaryConnect architecture. The authors also compared their BayesBiNN method with the LR-Net method in Shayer, Levi, and Fetaya[4]. We implemented the same model architecture as in the LR-Net paper. The detailed architectures are mentioned in the supplementary material provided with this report. For the segmentation task, we used the original U-Net architecture detailed in Ronneberger, Fischer, and Brox[5] with a minor difference that we introduced a BatchNorm layer after every convolution layer.

4.2 Datasets

The datasets used for image classification tasks are MNIST, CIFAR-10, and CIFAR-100. For generating visualizations for the BayesBiNN and STE methods, we used small toy datasets, the Snelson dataset[6] for regression problems, and Two Moon's dataset Snelson and Ghahramani[7] for classification problems. For the segmentation part, we used the Brain Tissue segmentation dataset from Ronneberger, Fischer, and Brox[5], and for the continual learning visualizations we used the permuted MNIST dataset Goodfellow et al.[8]. The pre-processing of inputs has been kept the same as mentioned in the original paper and has been detailed below.
Pre-processing: For the MNIST dataset we simply normalize the images and do not perform data augmentation. We keep our validation split as 0.1 uniformly across all sets of experiments except the comparison with the LR-Net method Shayer, Levi, and Fetaya[4]. For the CIFAR datasets also, we perform the normalization of images along with data-augmentation where we generate images by randomly cropping a 32x32 image from a 40x40 padded image. Finally, for our semantic segmentation task, we had a very small dataset of 30 images out of which 24 were chosen for training and 6 for validation. No other pre-processing has been done.

4.3 Hyperparameters

We have used the hyper-parameters given in the original paper. Table Table 1 contains the list of all the parameters we used for our experiments:

Optimizer	Parameter	MNIST	CIFAR10	CIFAR100	Snelson Dataset	2 Moons Dataset
BayesBiNN	MC steps	1	1	1	1	5
	Initial LR	10^{-4}	3.10^{-4}	3.10^{-4}	10^{-4}	10^{-3}
	Final LR	10^{-16}	10^{-16}	10^{-16}	10^{-5}	10^{-5}
	LR Scheduler	Cosine	Cosine	Cosine	MultiStepLR	MultiStepLR
	Temperature τ	10^{-10}	10^{-10}	10^{-8}	1	1
	Initialization λ	± 10	± 10	± 10	± 10	± 15
STE	Initial LR	10^{-2}	10^{-2}	10^{-2}	10^{-1}	10^{-1}
	Final LR	10^{-16}	10^{-16}	10^{-16}	10^{-1}	10^{-3}
	LR Scheduler	Cosine	Cosine	Cosine	MultiStepLR	MultiStepLR
Adam (Full Precision)	Initial LR	10^{-5}	10^{-4}	10^{-4}	-	-
	Final LR	Step	Step	Step	-	-
	LR Scheduler	1	100	100	-	-

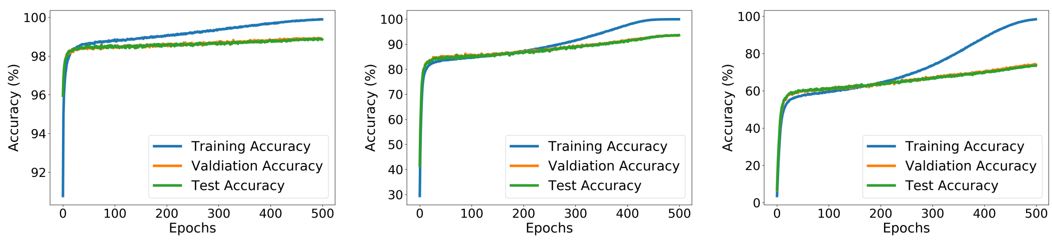
Table 1. Training setting for different optimizers on MNIST, CIFAR10, and CIFAR100 datasets.

4.4 Computational requirements

All our final experimental results were performed on a machine having 1 NVIDIA Tesla V100 GPU with 16 GB memory. Training the Binary Network with BayesBiNN optimizer for a single run, takes around 2.5 GPU hours for MNIST, 5.5 GPU hours for CIFAR-10, and around 8.5 GPU hours for the CIFAR-100 dataset, in the current experimental setup.

5 Results

In Table Table 1 we report our results for various classification benchmarks using our implemented BayesBiNN and STE optimizer. We notice that we get a difference of less than 0.1% as compared to that in the original paper. We generated the results for baseline STE optimizer and full-precision networks by evaluating our implementation of these methods. We also generated the results of PMF, by modifying its original open-sourced code and using the hyperparameters mentioned in the original paper.



(a) MNIST **(b)** CIFAR10 **(c)** CIFAR100

Figure 1. Training/Validation/Test accuracy using BayesBiNN optimizer

Datasets	Optimizer	Training Accuracy	Validation Accuracy	Test Accuracy
MNIST	BayesBiNN(ours)	$99.90 \pm 0.01\%$	$99.89 \pm 0.07\%$	$98.87 \pm 0.06\%$
	BayesBiNN(orig.)	$99.85 \pm 0.05\%$	$99.02 \pm 0.13\%$	$98.86 \pm 0.05\%$
	STE	$99.90 \pm 0.01\%$	$98.86 \pm 0.09\%$	$98.89 \pm 0.05\%$
	PMF	-	98.73%	-
	Adam (Full Precision)	$99.98 \pm 0.01\%$	$99.02 \pm 0.04\%$	$99.02 \pm 0.01\%$
CIFAR10	BayesBiNN(ours)	$99.96 \pm 0.01\%$	$93.59 \pm 0.45\%$	$93.54 \pm 0.26\%$
	BayesBiNN(orig.)	$99.96 \pm 0.01\%$	$94.23 \pm 0.41\%$	$93.72 \pm 0.16\%$
	STE	$99.99 \pm 0.01\%$	$93.77 \pm 0.06\%$	$93.54 \pm 0.08\%$
	PMF	-	91.98%	-
	Adam (Full Precision)	$99.99 \pm 0.01\%$	$94.27 \pm 0.15\%$	$94.38 \pm 0.16\%$
CIFAR100	BayesBiNN(ours)	$98.35 \pm 0.1\%$	$74.13 \pm 0.78\%$	$73.56 \pm 0.06\%$
	BayesBiNN(orig.)	$98.02 \pm 0.18\%$	$74.76 \pm 0.41\%$	$73.68 \pm 0.31\%$
	STE	$99.22 \pm 0.03\%$	$72.74 \pm 0.06\%$	$73.25 \pm 0.26\%$
	PMF	-	70.82%	-
	Adam (Full Precision)	$99.89 \pm 0.02\%$	$75.04 \pm 0.71\%$	$74.80 \pm 0.39\%$

Table 2. Results of different optimizers trained on MNIST, CIFAR10, and CIFAR100.

5.1 Comparison with LR-Net

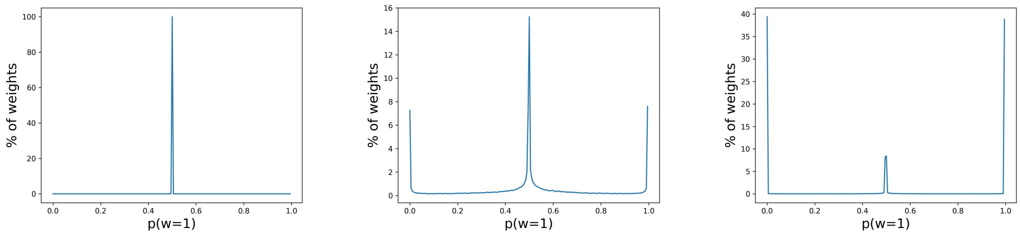
Authors compared their BayesBiNN approach to the LR-Net method presented in Shayer, Levi, and Fetaya[4]. We tried to reproduce the result for the same setting. In this comparison, the data pre-processing and augmentation methods remain the same as mentioned in section 4.2, but we do not split the data in training and validation sets in this case. We denote the test accuracies after 190 epochs in the case of MNIST and 290 epochs in the case of CIFAR-10, as done in the original paper to maintain uniformity. Note that, our accuracy is matching with that of the original authors in the case of MNIST but not in the case of CIFAR-10. We suspect that this is due to some difference in Batch-Norm layers used.

Optimizer	MNIST	CIFAR10
BayesBiNN (ours)	**99.52%**	84.49%
BayesBiNN (orig.)	99.50%	**93.97%**
LR-net Shayer, Levi, and Fetaya[4]	99.47%	93.18%

Table 3. Test accuracy of BayesBiNN and LRNet.

5.2 Continual Learning

As mentioned in the original paper, we try to reproduce the author's claims about weight distribution across tasks in a simple continual learning domain tested on Permuted MNIST. Clearly, as we learn across the tasks, the curve becomes flat from the middle conveying that the weights become more deterministic. Our result matches with the claims in the original paper.



(a) Prior λ **(b)** λ after task 1 **(c)** λ after task 2

Figure 2. Distribution of $p(w=1)$ across consecutive learning tasks

5.3 Visualization using Synthetic Dataset

In the original paper, the authors present visualizations on binary classification (Two moons datasetSnelson and Ghahramani[7]) and toy regression (Snelson dataset[6]) using STE and BayesBiNN optimizer. For the classification task, the authors claimed that STE is a more deterministic classifier compared to BayesBiNN. We reproduced this experiment and the results depicted in Figure Figure 3 seem to be consistent with the author's claim. For the regression task, we conclude that the author's claim about BayesBiNN (mean) giving a smoother curve compared to STE is true, which can also be seen in Figure Figure 4.

5.4 Extended Results (Semantic Segmentation)

We tried to validate the performance of the BayesBiNN optimizer on more complex tasks like semantic segmentation. Unfortunately, the results with BayesBiNN were quite underwhelming as compared to STE and its full-precision counterpart. We tried various parameters to improve its performance but none seemed to work. We had a brief discussion with the authors regarding this issue and the authors suggested that Bayesian

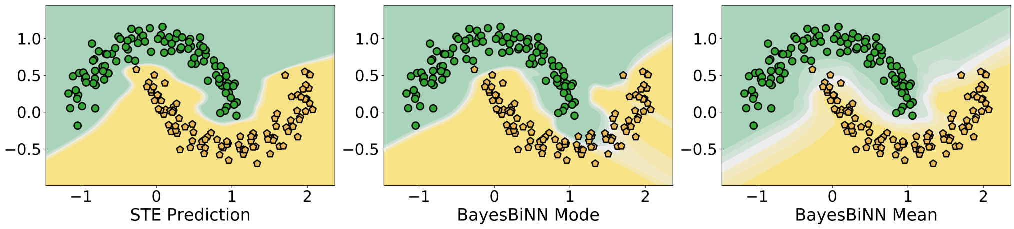


Figure 3. Classification on Two Moons dataset using STE and BayesBiNN optimizer.

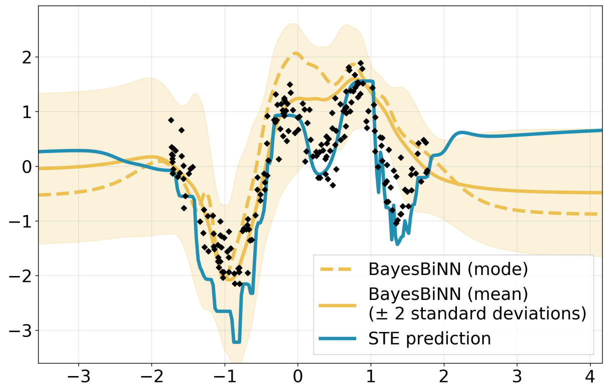
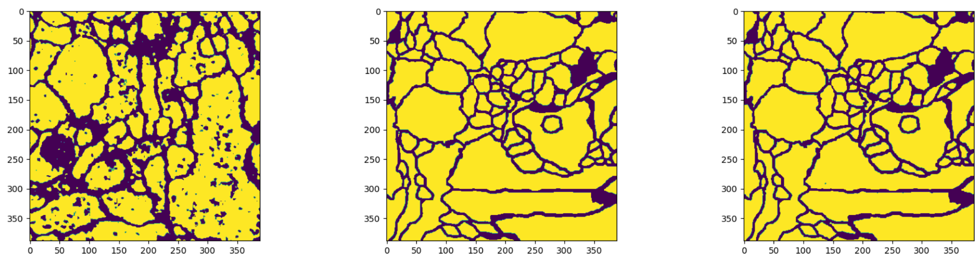


Figure 4. Regression on Snelson dataset using STE and BayesBiNN optimizer.

models are intrinsically very difficult to train. For the results shown in Table Table 5 and Figure Figure 5, we have used the hyperparameters denoted in Table Table 1.



(a) BayesBiNN **(b)** STE **(c)** Adam

Figure 5. Some samples of segmented image outputs

6 Discussion

We reproduced almost all the experiments given in the original paper and most of our results match with the original claims. While this BayesBiNN approach is mathematically principled, we tried to take a step forward by using that optimizer on a single segmentation task. However the results were against our expectation and the result of segmentation was a zoomed segmented image of the input with lots of noise. In addition to this, in the case of comparison with the LR-Net method, our accuracy differs from that of the original authors, which we feel might be due to some difference in architecture chosen. The major contribution of our work is developing a code base library

Temperature	10	1	0.1	10^{-2}	10^{-3}	10^{-4}
MSE Loss	1.313	0.208	2.151	0.443	0.231	0.199
Temperature	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
MSE Loss	0.156	0.127	0.173	0.122	0.195	0.173

Table 4. Mean square error loss of Snelson dataset for different temperatures.

	BayesBiNN	STE	Adam (Full Precision)
Validation Score	0.4102	0.3108	0.2943

Table 5. (1 - IoU) score for validation set

based on PyTorch with a Keras type interface for training BNNs with several different methods in its arsenal. This could reduce the coding efforts required for training BNNs and could help in future research as benchmarking library.

6.1 What was easy

The original paper contained a very good explanation of the mathematics behind the BayesBiNN approach. After we worked that out the pseudo-code as pointed out in Algorithm algorithm 1, the basic implementation of the optimizer became easy and easily verifiable by the author's original code. The appendix in the original paper contained a list of various hyper-parameters used for experiments. This helped us a lot while running the experiments and deciding the range of hyper-parameters while doing ablation studies.

6.2 What was difficult

The most difficult part here was running a large number of experiments in lack of many computational resources. This difficulty was increased since we are taking an average of 5 runs while reporting all our results. Apart from this, we also faced some difficulty in taking care of the hyper-parameters, which were not mentioned in the original paper (like momentum coefficient). To cater to that, we had to guess some possible values of the hyper-parameters and run small random searches to find a good candidate. Finally, we also faced difficulty while reproducing the results for the baselines PMF and Bop, and adapting their experimental settings to match with those used in the original BayesBiNN paper. Since their code was written a long time ago and used older software stack, this task took us a lot of time.

6.3 Communication with original authors

We did not understand the intent of the authors for choosing temperature as 1 in the case of experiments on synthetic datasets. We were also curious about the author's view on segmentation tasks using BayesBiNN. Hence, we reached out to the authors via email along with the review of their paper, to ask for some pointers. They gave the following major pointers:

- It is reasonable that at high temperatures the learned distribution will have high variance. The mode mentioned in the paper refers to the sign($\hat{(w)}$), where $\hat{(w)}$ denotes the expectation of the learned posterior Bernoulli distribution. It is not appropriate to directly use the continuous $\hat{(w)}$ as the mode. Another way is to use mean, which samples from the learned posterior Bernoulli distribution, and then make predictions using ensemble learning.

- STE is more stable and suggested by the authors to act as a baseline, in particular, Adam STE first, to make sure binary networks work. As shown in the paper, there is literally very little difference between STE and BayesBiNN but indeed the latter is difficult to train, as most Bayesian optimizers.

Broader Impact

Recent researches Strubell, Ganesh, and McCallum[9] mention that training a single big transformer model could emit around 626,155 lbs CO_2 which is around 5 times of average carbon emission by a car in its total lifetime. Clearly, Deep Learning takes a huge toll on the environment which is why there has been an increased focus on much more energy-efficient ”Green AI”. BNNs intrinsically have far less computational and space complexity as compared to their full-precision counterparts and as we can see above they can also achieve accuracy close to the full-precision networks, at least in the classification tasks, and also show the potential of expanding well to more complex segmentation tasks. This can help us a lot in moving towards cleaner Deep Learning. This field of research also provides a huge set of opportunities in extending AI to edge devices with much smaller and low-energy systems. We feel that its potential impact on the environment and sustainability is at par with its academic importance.

Acknowledgement

We would like to thank NVIDIA and IIT Delhi HPC facility for providing necessary computational resources. The computational requirements were also partly met by Google Colab and Code Ocean. We would also like to thank Weights & Biases for providing free teams to people in academic sphere which proved to be most valuable for our experiments and collaboration.

References

1. X. Meng, R. Bachmann, and M. E. Khan. **Training Binary Neural Networks using the Bayesian Learning Rule**. 2020. arXiv:2002.10778 [cs.LG].
2. M. Khan and H. Rue. **Learning Algorithms from Bayesian Principles**. URL: https://emtiyaz.github.io/papers/learning_from_bayes.pdf.
3. C. J. Maddison, A. Mnih, and Y. W. Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables.” In: **CoRR** (2016). URL: http://arxiv.org/abs/1611.00712.
4. O. Shayer, D. Levi, and E. Fetaya. “Learning Discrete Weights Using the Local Reparameterization Trick.” In: **CoRR** (2017). URL: http://arxiv.org/abs/1710.07739.
5. O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” In: **CoRR** (2015). URL: http://arxiv.org/abs/1505.04597.
6. **Moons, T. Two moons datasets description**. 2018. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html.
7. E. Snelson and Z. Ghahramani. “Sparse Gaussian Processes Using Pseudo-Inputs.” In: **Proceedings of the 18th International Conference on Neural Information Processing Systems**. MIT Press, 2005, pp. 1257–1264.
8. I. J. Goodfellow, M. Mirza, X. Da, A. C. Courville, and Y. Bengio. “An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks.” In: **2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings**. 2014. URL: http://arxiv.org/abs/1312.6211.
9. E. Strubell, A. Ganesh, and A. McCallum. “Energy and Policy Considerations for Deep Learning in NLP.” In: **CoRR** (2019).

Appendix

MLP Binary Connect Architecture

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

VGG Binary Connect Architecture

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 = 10, bias = False)
Batch Normalization Layer (gain = 1, bias = 0)
Softmax

MLP Binary Connect Architecture for Continual Learning

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

LRNet Architecture (MNIST)

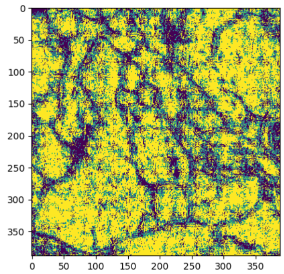
Convolutional Layer (channels = 32, kernel-size = 5×5, bias = False, padding = same) Max Pooling Layer (size = 2×2, stride = 2×2) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Convolutional Layer (channels = 64, kernel-size = 5×5, bias = False, padding = same) Max Pooling Layer (size = 2×2, stride = 2×2) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Fully Connected Layer (units = 512, bias = False) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Fully Connected Layer (units = 10, bias = False) Batch Normalization Layer (gain = 1, bias = 0) Softmax

LRNet Architecture (CIFAR-10)

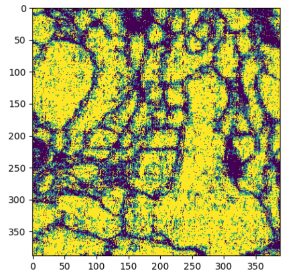
Convolutional Layer (channels = 128, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Convolutional Layer (channels = 128, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) Max Pooling Layer (size = 2×2, stride = 2×2) ReLU
Convolutional Layer (channels = 256, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Convolutional Layer (channels = 256, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) Max Pooling Layer (size = 2×2, stride = 2×2) ReLU
Convolutional Layer (channels = 512, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Convolutional Layer (channels = 512, kernel-size = 3×3, bias = False, padding = same) Batch Normalization Layer (gain = 1, bias = 0) Max Pooling Layer (size = 2×2, stride = 2×2) ReLU
Fully Connected Layer (units = 1024, bias = False) Batch Normalization Layer (gain = 1, bias = 0) ReLU
Fully Connected Layer (units = 10, bias = False) Batch Normalization Layer (gain = 1, bias = 0) Softmax

Semantic Segmentation using BayesBiNN with augmented dataset

We generated 1260 images from 30 original images using the rotation, random horizontal flip, random vertical flip operations. The result for BayesBiNN with this extended dataset was still very poor and inconsistent with the other methods (STE and Full Precision). The results presented in Section 5.4 were to show the extent of difficulty to train BayesBiNN for segmentation task as even with such a small dataset and large number of epochs, it was still not even able to overfit. Following are some of the images obtained by using BayesBiNN with this bigger dataset:



(a) Mask example 1



(b) Mask example 2