Are alternatives to backpropagation useful for training Binary Neural Networks? An experimental study in image classification

Ben Crulis, Barthelemy Serres, Cyril de Runz, Gilles Venturini

Are alternatives to backpropagation useful for training Binary Neural Networks? An experimental study in image classification

Ben Crulis
University of Tours
ben.crulis@etu.univ-tours.fr

Barthelemy Serres
University of Tours
barthelemy.serres@univ-tours.fr

Cyril de Runz
University of Tours
cyril.derunz@univ-tours.fr

Gilles Venturini
University of Tours
gilles.venturini@univ-tours.fr

ABSTRACT

Current artificial neural networks are trained with parameters encoded as floating point numbers that occupy lots of memory space at inference time. Due to the increase in size of deep learning models, it is becoming very difficult to consider training and using artificial neural networks on edge devices such as smartphones. Binary neural networks promise to reduce the size of deep neural network models as well as increasing inference speed while decreasing energy consumption and so allow the deployment of more powerful models on edge devices. However, binary neural networks are still proven to be difficult to train using the backpropagation based gradient descent scheme. We propose to adapt to binary neural networks two training algorithms considered as promising alternatives to backpropagation but for continuous neural networks. We provide experimental comparative results for image classification including the backpropagation baseline on the MNIST, Fashion MNIST and CIFAR-10 datasets in both continuous and binary settings. The results demonstrate that binary neural networks can not only be trained using alternative algorithms to backpropagation but can also be shown to lead better performance and a higher tolerance to the presence or absence of batch normalization layers.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**;

KEYWORDS

Binary neural networks, backpropagation, DFA, DRTP

ACM Reference Format:
Ben Crulis, Barthelemy Serres, Cyril de Runz, and Gilles Venturini. 2023. Are alternatives to backpropagation useful for training Binary Neural Networks? An experimental study in image classification. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27 – March 31, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555776.3577674

SAC'23, March 27 – March 31, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3577674

1 INTRODUCTION

Artificial Neural Networks (ANN) are known for their good performance and wide ranges of tasks, albeit at great costs at training and inference times. Nowadays, the research focuses on scaling up ANNs as their scaling curves predict performance gains when adding more parameters to the models, and generally make models cumbersome and slower. Unfortunately, this practice make deploying these bigger models on edge devices such as smartphones harder due to limited storage and memory space.

Yet, deploying models on smartphones directly has several benefits, such as reducing the amount of possibly insecure communications as well as reducing dependencies to cloud computing and to a permanent internet connection. The size reduction is particularly important in order to bring larger models such as large language models and large vision models on resource constrained devices and increase the number of deep learning models that can be stored and used at the same time on the device. The perspective of having more efficient training algorithms for deep learning enables the possibility of training and re-training models directly on the user's device, which enables more possibilities of personalization and user centered deep learning.

Binary Neural Networks (BNN) were proposed to make models smaller as well as improving their inference speed by leveraging low level binary operations. BNNs are thus a natural choice for deployment of models on portable devices, for instance on smartphones by benefiting from their eventual Graphical Processing Units[11] or on specialized hardware such as Field-Programmable Gate Array[31].

Binary neural networks have been used for voice recognition[24], pedestrian detection [22], stereo (depth) estimation[5], human activity recognition[7] and face mask wear and positioning correction[9].

Unfortunately, BNNs are much more difficult to train and sometimes suffer from degraded performance compared to their continuous counterparts. The main modern method to train BNNs was introduced in [13]. They achieve training by slightly altering the backpropagation (BP) and gradient descent (GD) scheme to allow gradients to flow through binarization steps. Due to the various estimation and quantization errors present in this method, training BNNs is still difficult and lead to lower performance compared to their full-precision counterparts.

However, recent advances in alternative training algorithms to backpropagation that match BP performance in full-precision network let us consider using them for training BNNs as well. To

the best of our knowledge, trying alternatives to BP for training BNNs was not attempted before.

In this paper, we propose to compare the training performance of the backpropagation algorithm with two other alternatives for training binary neural networks in the context of image classification. We also examine the qualitative differences between the algorithms and architecture choices. The main contributions of the paper are:

(1) Proposing adaptations to promising backpropagation alternatives for training BNNs.
(2) Showing the importance of the batch normalization layer in the different training algorithms
(3) Providing extensive tests on MNIST, Fashion MNIST and CIFAR-10 to show the viability of alternative algorithms for training BNNs.

We also provide the source code of all experiments at https://github.com/BenCrulis/binary_nn.

The rest of the paper is structured as follows: First, in Section 2 we present several alternatives to the backpropagation algorithm that allow to train neural networks with similar performances. Then in Section 3 we introduce the chosen method of binarization compatible with all training algorithms. In Section 4, we detail the experiments and results on three image classification tasks. Before summing up the paper in the conclusion, we discuss the results and their implications in Section 5.

2 BACKGROUND

In this section, we present the classical way of training Binary Neural Networks as well as some approaches that may replace the backpropagation mechanism.

2.1 Binary Neural Networks

This subsection introduces the main characteristic of BNNs. For more details about BNNs, please refer to recent surveys on BNNs [25, 27, 30].

Contrary to classical ANNs with real-valued parameters, every single parameter of a binary neural network is coded on a single bit, which can be seen as an extreme form of quantization of the continuous weights. A bit representing a single weight can encode one element among a pair of values, usually -1 and 1 for binary neural networks as it allows to replace the floating point multiplication by a logical XNOR operation. The immediate consequence is that binary neural networks are much smaller than their continuous counterparts, one can expect a reduction in size of a factor 32 compared to a network using 32 bits floating point weights. Provided the activation function used in the hidden layers also output binary values, the following computation of the next binary layer can be optimised with efficient binary operation and also provide inference speedups.

Training binary neural networks using gradient descent. Training a multilayer binary neural network by searching naively for a valid set of binary weights is a combinatorial problem that does not scale well to models beyond a few tens of parameters due to the non-linear binary optimization nature of the problem. Training neural network models with only three nodes and linear threshold functions is NP-Complete in a worst case sense [4]. For this reason, in order to train deep BNNs the usual method consists in training a continuously parametrized model as a support for the binarized model. The classical backpropagation algorithm is modified so that we can compute a non-zero gradient of the parameters with respect to the loss when traversing binary activation layers. This is the role of the so called Straight-Through Estimator (STE) [12] which turns the unusable derivative of the *sign* function into a useful approximation, usually an identity function.

The deep neural network is then trained in a classical manner, with a few modifications as proposed in [13]. In the forward pass, the model parameters are binarized using a threshold function, which turns the continuous parameters into binary values, in our case we use the *sign* function that outputs -1 or 1. These binary weights are used to compute the predicted values and the loss. During this forward pass, the data can also pass through binary activation layer such as the *sign* function yet again.

In the backward pass, the chosen STE is used at the binarization steps to compute a gradient that will be used to update the real valued parameters of the model. In the classical backpropagation scheme, the transposed matrices of the binarized weights are used to propagate the error to the previous layer. It should be mentioned that the computed gradient in the backward pass is not binary in general.

When deploying the model, the continuous parameters can be used to compute the final binary weights using the *sign* function and discarded, reducing the model size for both storage and inference.

The activation function of a BNN can be any differentiable function (tanh, ELU, ReLU,…), or any non-differentiable function (step functions, for instance) as long as an appropriate estimator of the gradient is provided. However, only the activation functions outputting binary vectors, e.g. *sign* function, leads to space reductions. This also conducts to obtain inference speed gains if the following layer is binary, thanks to the possibility of rewriting the floating point multiply and accumulation operations into efficient low level binary operations. The former point is particularly important in convolutional neural networks since the activation buffers usually take much more space than the kernels used to compute them, so a reduction in the activation storage requirements can be very beneficial at inference time.

The difficulty of training BNNs comes from the approximations that are made to enable training of the parameters using gradients. The STE being one of these approximation used in the binary activation layers, in a normal backpropagation pass, the error signal will flow through several of these estimators. We can conjecture traversing several of these estimator increase the level of approximation of the teaching signal to the point it starts to hurt the learning performance of the model.

Batch normalization. In the original paper [13] introducing the method for training binary neural network we consider, a shift-based batch normalization layer is used to approximate the original batch normalization layer [14]. This approximation is used in order to avoid some multiplication operations. As we are not interested in measuring the wall clock performance of the low level operations, in this work we only test the original non-approximated batch

normalization layer. We will nonetheless note that in both the non-approximated and approximated batch norm layers, the output of the layer is a vector of real numbers, implying a change of datatype and thus a possible greater storage cost for intermediate variables. The batch normalization layer is placed before the activation layer and is supposed to accelerate training as well as reducing the impact of the weight scale [13].

2.2 Backpropagation and alternatives

Backpropagation. The Backpropagation algorithm (BP) is the most common method for the end-to-end training of ANNs. BP allows to decrease the error of a differentiable model by performing Gradient Descent (GD) on the loss landscape induced by the differentiable training loss objective. After a forward pass on a data batch and the computation of the loss, a gradient of the parameters with respect to the loss is computed in all layers using the chain rule, in a step called the backward pass. This gradient is then used to apply a small modification of the weights which slightly improves the model performance on the training set. This training scheme is illustrated in Fig. 1a.

Feedback Alignment. Feedback Alignment (FA) is an alternative to backpropagation introduced in [18] with the explicit goal of proposing a more biologically plausible learning mechanism for training neural networks. FA demonstrate the surprising fact that, under some conditions, backpropagating the error of the output layer using the transpose matrix of each linear layer is not needed, and that a constant random matrix is sufficient to provide a useful learning signal to the upstream layers. By using random matrices, one might have the intuition that no useful features could be learned in the upstream layers. Yet, [18] shows that the weight matrices adapt to the constant random matrices and in turn allows the network to extract useful features in all layers.

Direct Feedback Alignment. In [21], the author goes a step further and proposes Direct Feedback Alignment (DFA) a method that improves on FA by propagating the output error signal directly to each layer using constant random matrices. Contrary to FA where the error signal goes through up to $K-1$ layers, where K is the number of layers in the network, in DFA the output error goes through exactly 0 layer in reverse. As the error effectively skips all downstream layers to train a particular layer, this method allows the training of very deep networks (with more than 100 layers) where BP would fail to converge. Although [21] suggests this is due to the difference in the initialisation of parameters, it is also explained if downstream layers are frozen, DFA cannot decrease the error whereas BP can. This training scheme is illustrated in Fig. 1b.

Direct Random Target Propagation. All previous mentioned methods necessitated to compute the output loss of the model in order to provide an error signal to train the layers. This effectively prevents the model's parameters to be updated until the forward pass is completed, this phenomenon is referred to as *update locking*. However another algorithm called Direct Random Target Projection (DRTP) was recently introduced to address this problem [10] and provide a less costly and more biologically plausible learning algorithm for deep neural networks. DRTP project the target labels directly to each layer using constant random matrices in a way similar to DFA and use the result as a learning signal to train each layer. By design, DRTP allows each layer to be updated as soon as its input activations are available, it is said to be *update-unlocked*. The parameters of models trained using DRTP can be updated while the forward pass is not completed, which also saves memory as the input buffers can be released once the layer parameters are updated. This training scheme is illustrated in Fig. 1c.

Other candidate alternatives for training BNNs. Very different types of training algorithms have been proposed to overcome limitations of BP or to provide biologically plausible alternatives. Among these alternatives, an interesting possibility is to use the Hilbert-Schmidt Independence Criterion (HSIC) to create a local loss for training each layer [19][23]. As these HSIC based losses only require the output of the layers and the target labels to update the parameters, these algorithms are thus very similar to DRTP. The Synthetic Gradient method also allows training without gradient locking [15]. Associated Learning proposes to train the layers to encode the inputs and the labels to similar embedded representation [28]. Some approaches try to view synapses or neurons as Reinforcement Learning agents that learn to predict the weight modifications that will improve the loss [17][3]. Other methods manage to get notable results by performing random modifications of the model parameters and accepting or rejecting the modifications [20][1]. Kickback shows that backpropagation can be decomposed into several local optimizations that optimize the same global error signal [2]. [6] proposes to enable online learning of deep networks by focusing on learning the hidden representations that decrease the error. Finally, an approach based on mixed-integer linear programming solvers was experimented in [26].

Due to limited space in the paper, we leave these other algorithms out of our experiments. In addition, these algorithms have a higher training cost and thus they are less adapted to edge computing.

3 ALGORITHMS

In this section we describe the binarization scheme used in the experiments.

Figure 1 summarises the differences between algorithms. BP is the only algorithm tested where the error traverses the layers in reverse, the other two algorithms directly send an error directly to each layer in parallel. BP (a) and DFA (b) are the only algorithms where the information learned in the downstream layers affect the computation of the errors in the upstream layers, although only indirectly in the case of DFA. DRTP (c) can directly train the layers in the forward pass.

3.1 Binarization of the weights and activations

In order to make the algorithms compatible with weight binarization, we slightly modify the way the forward pass and backward pass are computed. Firstly, we change the activation function of the hidden layers to be a *sign* function with a Straight-Through Estimator (STE), i.e. the derivative of the *sign* function is replaced with another function that will provide a non-zero gradient to previous layers. In the experiments we use two variants of the STE. Let δy_k be the incoming gradient at layer k and δz_k the estimated output gradient of the *sign* function. One of the STE variant is ignoring the

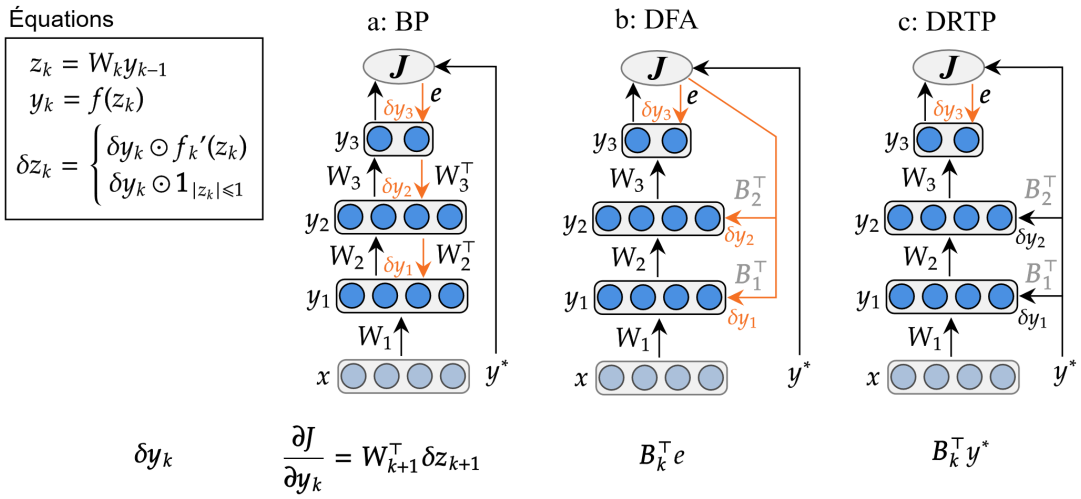


Figure 1: Summary of the algorithms tested in the following experiments. The B_k matrices are constant random matrices initialised once before starting to train the model. Adapted from [10].

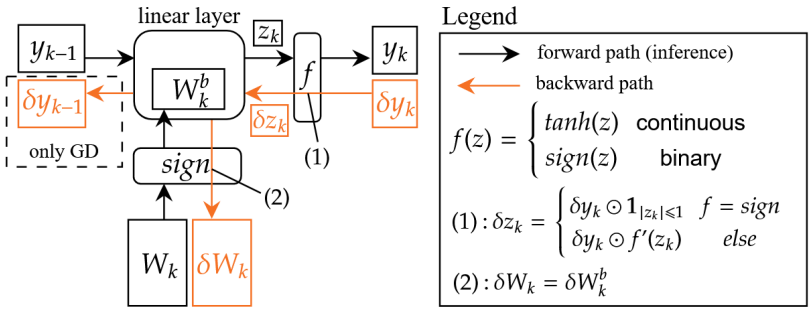


Figure 2: Summary of the chosen binarization scheme effective at each binary weight layer. Here a single layer of the model at index k is represented with both the linear layer using the binary weights W_k^b and the activation function f.

gradient of the *sign* function as if the activation was the identity function, so in this case $\delta z_k = \delta y_k$. This is the non-saturating STE. Another variant introduced in [13] is equivalent to propagating the gradient through a hard *tanh* function ($Htanh(x) = clip(x, -1, 1)$), in which case $\delta z_k = \delta y_k \odot \mathbf{1}_{|z_k|\leq 1}$. This STE is saturating as the gradient is 0 when the neuron as reached saturation, that is when $|x| > 1$, x being the neuron pre-activation.

In the experiments, we use the non-saturating STE for the binarization of the weights when applicable and the saturating STE for the activation function when using the *sign* function as the activation function of the hidden layers.

Figure 2 shows the flow of the information in the forward and backward pass when learning. The vector z_k is the pre-activation and y_k is the activation at layer k. When in the forward pass, the real valued weights are first binarized to get the binary weight matrix W_k^b using the *sign* function, and then used to compute the pre-activations vector z_k from y_{k-1} in order to get y_k as follows: $y_k = f(W_k^b y_{k-1})$. This value is then used as input to the next layer or as output of the network.

In the backward pass, the error δy_k is sent through the STE of the activation function if it is binary, or computed normally if the activation function is *tanh*. This gives us the error δz_k that is itself used together with y_{k-1} to compute the gradient of the real valued weights δW_k by sending it through the non-saturating STE. In the particular case of the BP algorithm, the error δz_k is also used to compute the output error of the previous layer of index $k-1$ as follows: $\delta y_{k-1} = (W_k^b)^\top \delta z_k$. This means the error signal that already went through a STE will go through another one upon reaching the previous layer. The parameters at layer k thus receive an error that went through $K-k$ STEs, K being the total number of layers in the model. The other algorithms all compute the error δy_k either from the loss error or directly from the target labels y^* and so receive an error that went through a constant number of STEs, only once in our case.

4 EXPERIMENTS

In this section we present a test protocol to run a comparison of the algorithms on three benchmark datasets and describe the results. The goal of this experiment is not to reach state of the art

results on the datasets but to assess the qualitative and quantitative differences between algorithms when training with binary weights compared to their continuous versions. The code of the experiments is available at https://github.com/BenCrulis/binary_nn.

These experiments aim to answer a few questions:

(1) What is the impact of the binarization of the neuron activations when training with the different algorithms?
(2) What is the impact of the binarization of the weights when training with the different algorithms?
(3) What importance does the batch normalization layers have in training?
(4) Which method(s) should be preferred and which ones should be avoided when training models with both binary weights and activations?

4.1 Protocol

We now propose to measure the relative difference performance of the algorithms and their binary variants on the MNIST[8], Fashion MNIST[29] datasets and CIFAR-10[16] datasets. These are 10-class classification tasks with respectively 70000, 70000, 60000 tiny images of resolution 28×28, 28×28 and 32×32. Each dataset contains 10000 images reserved for test.

All layers and algorithms where re-implemented from scratch in Numpy to ensure identical experimental conditions and implementation details. The fully connected layers have no bias term in order to make the binary and continuous architectures as close as possible. The loss function used is a $L2$ squared error and a batch size of 128.

We run each training algorithm in their continuous and binarized variants on the datasets and report the results. We also separate experiments for each type of activation ($tanh$ or $sign$) in order to evaluate the impact of using a binary activation function in the hidden layers. For the MNIST and Fashion MNIST datasets, we also repeat the experiments without the batch normalization layers.

We first run a grid search on the learning rate and initialisation value hyperparameters to find the combinations leading to the highest accuracy on MNIST in 100 epochs for each algorithm-binarization-activation triplet. The possible values for the learning rate are 10^{-4}, 10^{-5} and 10^{-6}. All models are trained from scratch using random initial weights taken from a uniform distribution. For the initialization parameter values of the uniform distribution, we only search between two values, small (10^{-3}) and large (0.1). The hyperparameters found on MNIST are used for Fashion MNIST. For CIFAR-10, we run another grid search on the same hyperparameter set on the CIFAR-10 train set.

All common hyperparameters are reported in Table 1. The architecture row describes the number of output neurons at each hidden layer and at the last layer. The retained architectures were chosen arbitrarily with the goal of being deep because shallow networks would pose no difficulty with binarization. The retained architecture has a total of 1110800 trainable parameters (ignoring batch normalization layers). As all tasks are classifications with 10 classes, the last layer is a continuous (non-binarized) layer with 10 output neurons and a $tanh$ activation in all experiments. This last layer is always trained at the same time as all other layers using a normal gradient descent on the gradient of the loss at the last layer.

Table 1: Common hyperparameters used in experiments

	MNIST, Fashion MNIST, CIFAR10
architecture	700,500,300,200,10
batch size	128
number of epochs	100

Then for each algorithm we repeat the experiment 10 times with the best set of hyperparameters. The experiments are duplicated with both continuous and binary weights as well as with both the continuous $tanh$ activation function and the binary $sign$ function. This will let us assess the individual effect of the choice of binarization in both weights and activations regarding the performance of the final model.

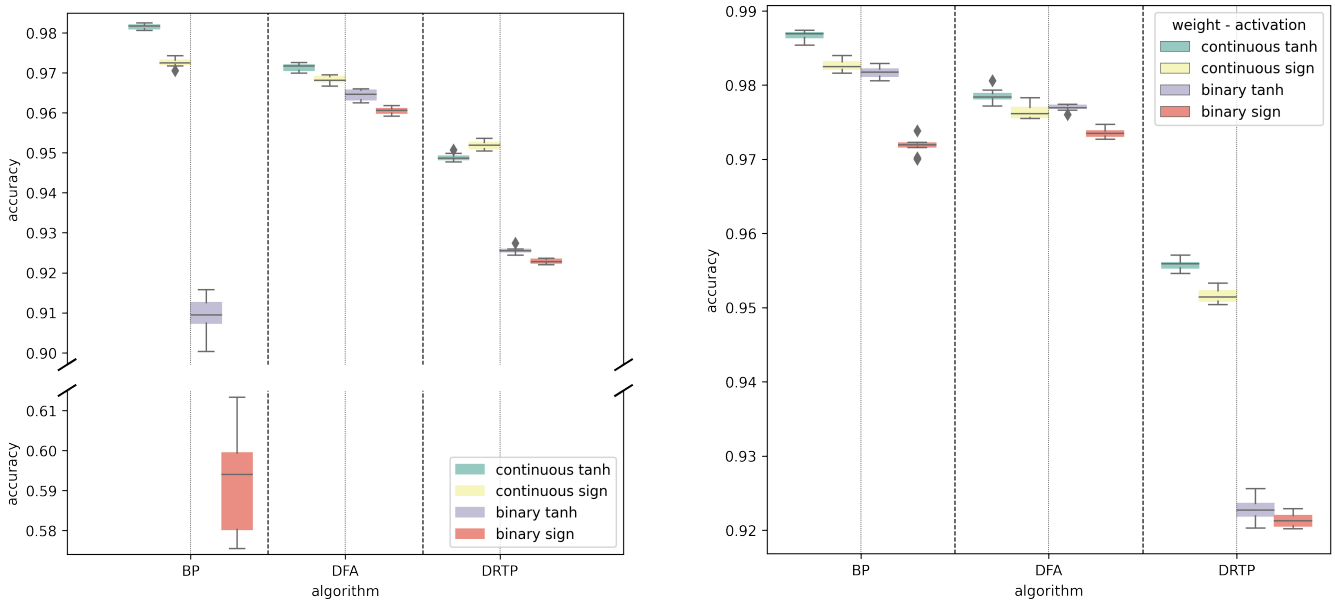
4.2 Results

In this section we present the results of the experiments in terms of test accuracy on the three datasets.

Figure 3 reports the best accuracies obtained in the repeated runs. The immediate observation is that the transition from continuous to binary weights as well as the transition from continuous to binary activation tend to cause a loss in accuracy whose magnitude depends on the particular algorithm considered. Without batch normalization, the sharpest drop observed is for BP which loses about 30% accuracy when using the binary activation instead of the continuous $tanh$ activation function when also using binary weights. The other algorithms are less affected by these changes, the biggest cause of loss of performance appears to be the weight binarization and not the use of a binary activation function. In particular, DRTP seems to be heavily impacted by the change from continuous to binary weights, both with and without batch normalization. Despite being roughly equal in term of accuracy on the train set (not shown here), DFA is slightly behind BP in general on the test set, while being slightly above BP in the case of both binary weights and activations, as confirmed by an ANOVA test yielding a p-value of less than 0.03% on the equality of the two group means.

Similar observations can be made for the Fashion MNIST dataset in Figure 4. The drops in accuracies appear to be slightly greater and the differences between algorithms also become more visible. Again, without batch normalization BP is outperformed by the other algorithms in the binary weights case by a greater margin compared to MNIST, but still dominates in the non-fully binarized cases. It can also be noted that the performance ranking of the algorithms is preserved from MNIST to Fashion MNIST. In particular, in the case of both binary weights and activations, DFA is again a few points ahead of BP, only slightly ahead with batch normalization and by a great margin without it. This time the ANOVA test yields a p-value of approximately 2.2×10^{-5}.

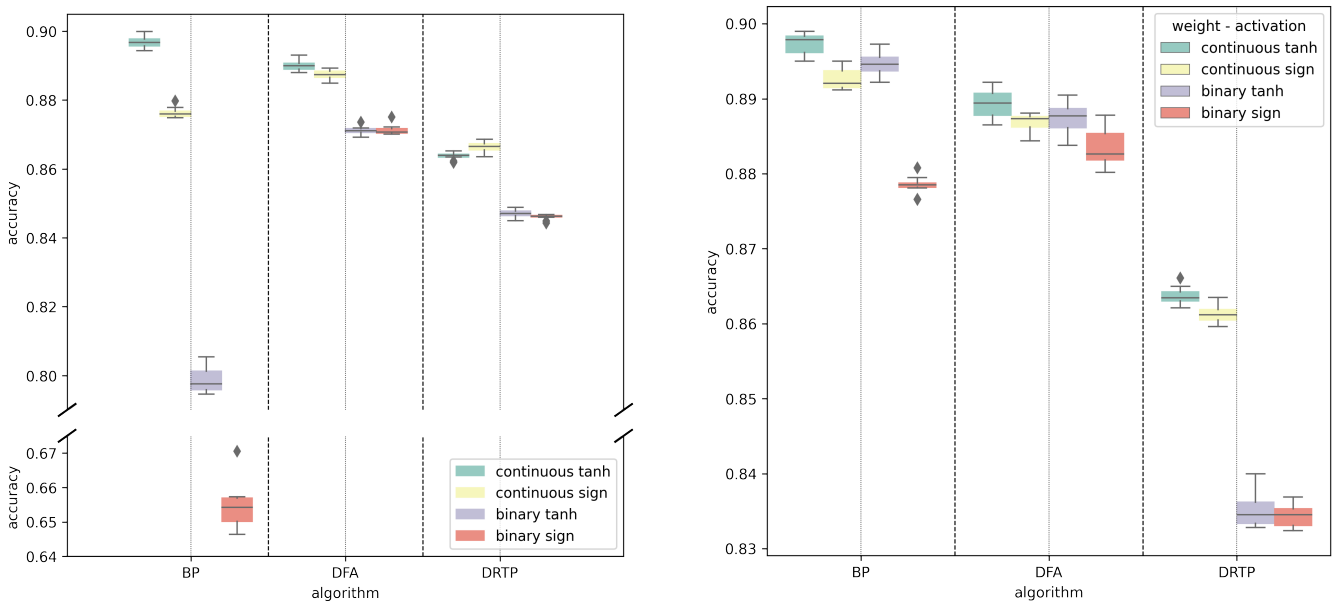
The experiments on CIFAR-10 summarized in Figure 5 are slightly different regarding BP and DFA. BP is no longer the best algorithm in the fully continuous version as it is well under DFA but still slightly outperforms DRTP. Here again, DFA outperforms BP in the fully binarized case by a large margin, with a p-value of 7.9×10^{-13}.



(a) without batch normalization layers

(b) with batch normalization layers

Figure 3: Best test accuracy in 100 epochs on MNIST. Each box plot correspond to 10 independent executions with exactly the same hyperparameters but using different seeds. The weight-activation boxes for each algorithm are presented in the same order as in the legend.



(a) without batch normalization layers

(b) with batch normalization layers

Figure 4: Best test accuracy in 100 epochs on Fashion MNIST. Each box plot correspond to 10 independent executions with exactly the same hyperparameters but using different seeds. The weight-activation boxes for each algorithm are presented in the same order as in the legend.

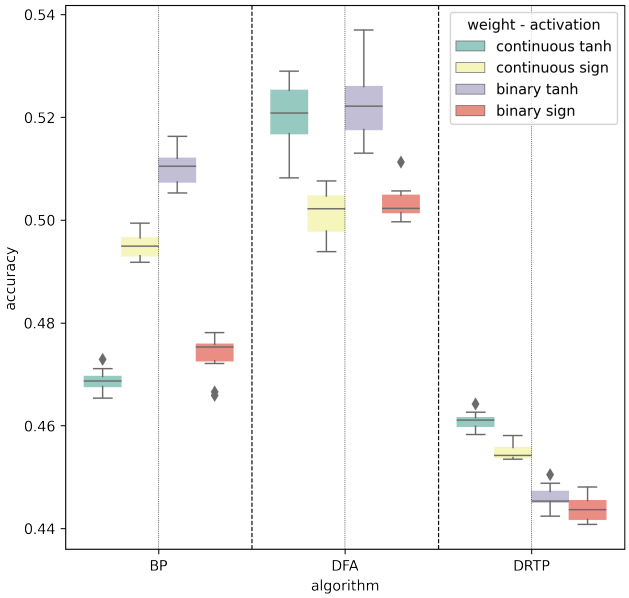


Figure 5: Best test accuracy in 100 epochs on CIFAR-10 with batch normalization. Each box plot correspond to 10 independent executions with exactly the same hyperparameters but using different seeds. The weight-activation boxes for each algorithm are presented in the same order as in the legend.

5 DISCUSSION

What is the impact of the binarization of the neuron activations when training with the different algorithms? Overall, the impact of using the sign function in place of the tanh function seems to be responsible for a small decrease in accuracy for all the algorithms considered and for both types of weights, continuous or binarized. This probably should not be seen as surprising given the STE used with the sign function was shown to be a very effective to train binary neural networks in practice.

What is the impact of the binarization of the weights when training with the different algorithms? The binarization of the weights appears to penalize the accuracy more strongly compared to the binarization of the activations, especially on DRTP and BP when the batch normalization layers are removed. This is most likely due to the pre-activation values being too large in magnitude because of the disappearance of small weights in the dense layers. These pre-activation values saturates the neurons and cause the gradient to vanish for most of the neurons, with both the STE and tanh derivative. BP seems to be particularly impacted, most likely due to the gradient having to traverse multiple activation layers recursively, perhaps inhibiting the ability of the model to form useful features in the first layers.

What importance does the batch normalization layers have in training? The batch normalization layers thus appear to have the important role of preventing saturation of the neurons throughout the model. Since BP requires the traversal of multiple layers, it is strongly negatively affected in the fully binarized case since saturation is more likely to happen, and the traversal of a STE in a saturated units means the gradient is set to 0, providing no feedback at all the to previous layers. The alternatives to BP are less affected as their teaching signal only traverse a constant amount of approximation no matter the depth of the model. This is coherent with the ability of these types of algorithms to train very deep networks, whereas BP usually fails because it requires a very small initialization of the weights[21], which is impossible with binary weights.

Which method(s) should be preferred and which ones should be avoided when training models with both binary weights and activations? These experiments seems to show BP is very sensitive to binarization in both weights and activations as well as to the presence of batch normalization, whereas DFA shows a lesser degradation of generalisation performance in its fully binarised version. DRTP and DFA appear to be very little sensitive to the presence of batch normalization whatever the level of binarization, which may come as a surprise given the impact it appears to have on the performance of the binary models trained with BP.

Overall, DFA seems to be the most promising algorithm for training fully-binarised neural networks among the algorithms tested. Contrary to the other algorithms, DFA is the only one that both make the error traverse a constant amount of STEs and use the information learned in the downstream layers in the previous iteration to compute the error. This combination of qualitative properties might explain its good performance in the context of binarization. DFA is also less affected by the choice of using batch normalization or not compared to BP. This is of importance because using batch normalization layers introduce learning parameters, intermediate variables and additional operations slowing down training and inference computations.

Since DRTP already provides a way to save memory by releasing the memory for the buffers of previous layers, further memory gains will have to be found elsewhere. The most obvious way to further save memory when training would be to store only binary weights and train on them directly instead of storing continuous versions of the parameter and binarizing them in the forward pass. This would effectively provide the same space gain at inference and train time, along with probable computational benefits if the new method can benefit from binary operations.

By design, the alternative algorithms already provide performance benefits and memory cost reduction in the continuous case with the full precision weights. Combined with the benefits of binarization that further reduce the memory and energy costs, it is now possible to consider enable the training of binary models at a manageable cost directly on edge devices. This should enable the update of deep neural networks models directly on user devices with their own data without requiring communication of personal data with external servers, increasing privacy.

6 CONCLUSION

Binary Neural Networks are more suited to use on edge devices compared to full-precision networks because of their lower memory and computational costs as well as their lower energy usage. The current approach to deal with BNN learning is to use backpropagation with the STE. Nevertheless, several alternatives to

backpropagation, such as DFA and DRTP, were recently proposed but not tested in the context of BNNs. These alternatives have lower complexity and memory cost in comparison to backpropagation. To the best of our knowledge, we are the first to study the impact of the different learning schemes and their binarization in terms of performance (accuracy).

With the notable exception that DFA outperforms BP in the fully continuous case on the CIFAR-10 experiment, we showed that alternatives to the backpropagation algorithm are in general behind BP in the continuous case. However, it is possible to approach BP performance and even outperform BP using the DFA algorithm in the fully binarized case, that is when using both weights and activations. Moreover, the alternative algorithms appears to be less sensitive to the use of batch normalization compared to BP, potentially allowing us to simplify the design and training of binary neural networks models. These results let us consider that the backpropagation algorithm may not be the best training method for binary neural networks with both binary weights and binary activations. Compared to backpropagation, these alternative algorithms are also themselves less costly in terms of memory and computations by design, opening up possibility of training binary neural networks directly on edge devices. In the future we will exploit these ideas to train and deploy more efficient models on edge devices, in particular, vision models on smartphones.

REFERENCES

[1] G. Akshat and N. R. Prasad. 2022. Blind Descent: A Prequel to Gradient Descent. *Lecture Notes in Electrical Engineering* 783 (2022), 473–479. https://doi.org/10.1007/978-981-16-3690-5_41

[2] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. 2015. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. *Proceedings of the National Conference on Artificial Intelligence* 1 (2015), 485–491.

[3] Aman Bhargava, Mohammad R. Rezaei, and Milad Lankarany. 2022. Gradient-Free Neural Network Training via Synaptic-Level Reinforcement Learning. *AppliedMath* 2 (2022), 185–195. Issue 2. https://doi.org/10.3390/appliedmath2020011

[4] Avrim L. Blum and Ronald L. Rivest. 1992. Training a 3-node neural network is NP-complete. *Neural Networks* 5, 1 (1992), 117–127. https://doi.org/10.1016/S0893-6080(05)80010-3

[5] Gang Chen, Yehua Ling, Tao He, Haitao Meng, Shengyu He, Yu Zhang, and Kai Huang. 2020. StereoEngine: An FPGA-Based Accelerator for Real-Time High-Quality Stereo Estimation with Binary Neural Network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), 4179–4190. Issue 11. https://doi.org/10.1109/TCAD.2020.3012864

[6] Anna Choromanska, Benjamin Cowen, Sadhana Kumaravel, Ronny Luss, Mattia Rigotti, Irina Rish, Brian Kingsbury, Paolo DiAchille, Viatcheslav Gurev, Ravi Tejwani, and Djallel Bouneffouf. 2019. Beyond backprop: Online alternating minimization with auxiliary variables. *36th International Conference on Machine Learning, ICML 2019* 2019-June (2019), 2041–2050.

[7] Francesco Daghero, Chen Xie, Daniele Jahier Pagliari, Alessio Burrello, Marco Castellano, Luca Gandolfi, Andrea Calimera, Enrico MacIi, and Massimo Poncino. 2021. Ultra-compact binary neural networks for human activity recognition on RISC-V processors. In *Proceedings of the 18th ACM International Conference on Computing Frontiers, CF 2021*. ACM, Virtual, 3–11. https://doi.org/10.1145/3457388.3458656

[8] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29 (2012), 141–142. Issue 6.

[9] Nael Fasfous, Manoj Rohit Vemparala, Alexander Frickenstein, Lukas Frickenstein, Mohamed Badawy, and Walter Stechele. 2021. BinaryCoP: Binary Neural Network-based COVID-19 Face-Mask Wear and Positioning Predictor on Edge Devices. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2021 - In conjunction with IEEE IPDPS 2021*. IEEE, Portland, OR, USA, 108–115. https://doi.org/10.1109/IPDPSW52791.2021.00024

[10] Charlotte Frenkel, Martin Lefebvre, and David Bol. 2021. Learning Without Feedback: Fixed Random Learning Signals Allow for Feedforward Training of Deep Neural Networks. *Frontiers in Neuroscience* 15 (2021), 1–13. Issue February. https://doi.org/10.3389/fnins.2021.629892

[11] Shengyu He, Haitao Meng, Zhaoheng Zhou, Yongjun Liu, Kai Huang, and Gang Chen. 2021. An efficient GPU-accelerated inference engine for binary neural network on mobile phones. *Journal of Systems Architecture* 117 (2021), 102156. Issue March. https://doi.org/10.1016/j.sysarc.2021.102156

[12] Geoffrey Hinton. 2012. Neural networks for machine learning coursera video lectures.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016,*. Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). Curran Associates, Inc., Barcelona, Spain, 4107–4115. https://proceedings.neurips.cc/paper/2016/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html

[14] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015* 1 (2015), 448–456.

[15] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. 2017. Decoupled neural interfaces using synthetic gradients. *34th International Conference on Machine Learning, ICML 2017* 4 (2017), 2558–2577.

[16] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto, Canada.

[17] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Körding. 2020. Learning to solve the credit assignment problem. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, Addis Ababa, Ethiopia, 1–19. https://openreview.net/forum?id=ByeUBANtvB

[18] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* 7 (2016), 1–10. https://doi.org/10.1038/ncomms13276

[19] Wan Duo Kurt Ma, J. P. Lewis, and W. Bastiaan Kleijn. 2020. The HSIC bottleneck: Deep learning without back-propagation. In *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*. AAAI Press, New York, New York, USA, 5085–5092. https://doi.org/10.1609/aaai.v34i04.5950

[20] Gregory Morse and Kenneth O. Stanley. 2016. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*. ACM, Denver, Colorado, USA, 477–484. https://doi.org/10.1145/2908812.2908916

[21] Arild Nøkland. 2016. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems, Nips*. Curran Associates Inc., Barcelona, Spain, 1045–1053.

[22] Fernando Cladera Ojeda, Anthony Bisulco, Daniel Kepple, Volkan Isler, and Daniel D Lee. 2020. On-Device Event Filtering with Binary Neural Networks for Pedestrian Detection Using Neuromorphic Vision Sensors. In *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, Abu Dhabi, United Arab Emirates, 3084–3088. https://doi.org/10.1109/ICIP40778.2020.9191148

[23] Roman Pogodin and Peter E. Latham. 2020. Kernelized information bottleneck leads to biologically plausible 3-factor Hebbian learning in deep networks. In *Advances in Neural Information Processing Systems, NeurIPS 2020*. Curran Associates Inc., Red Hook, NY, USA, 12. Issue NeurIPS.

[24] Yan-min Qian and Xu Xiang. 2019. Binary neural networks for speech recognition. *Frontiers of Information Technology and Electronic Engineering* 20 (2019), 701–715. Issue 5. https://doi.org/10.1631/FITEE.1800469

[25] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. 2020. Binary neural networks: A survey. *Pattern Recognition* 105 (2020), 107281. https://doi.org/10.1016/j.patcog.2020.107281

[26] Hasan Sildir and Erdal Aydin. 2022. A Mixed-Integer linear programming based training and feature selection method for artificial neural networks using piecewise linear approximations. *Chemical Engineering Science* 249 (2022), 117273. https://doi.org/10.1016/j.ces.2021.117273

[27] Taylor Simons and Dah Jye Lee. 2019. A review of binarized neural networks. *Electronics* 8 (2019), 661. Issue 6. https://doi.org/10.3390/electronics8060661

[28] Dennis Y Wu, Di nan Lin, Vincent F Chen, and Hung hsuan Chen. 2022. Associated Learning: an Alternative to End-to-End Backpropagation that Works on CNN, RNN, and Transformer. In *International Conference on Learning Representations*. OpenReview.net, virtual, 1–18.

[29] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. http://arxiv.org/abs/1708.07747 cite arxiv:1708.07747Comment: Dataset is freely available at https://github.com/zalandoresearch/fashion-mnist Benchmark is available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/.

[30] Wenyu Zhao, Teli Ma, Xuan Gong, Baochang Zhang, and David Doermann. 2020. A Review of Recent Advances of Binary Neural Networks for Edge Computing. *IEEE Journal on Miniaturization for Air and Space Systems* 2 (2020), 25–35. Issue 1. https://doi.org/10.1109/jmass.2020.3034205

[31] Yuteng Zhou, Shrutika Redkar, and Xinming Huang. 2017. Deep learning binary neural network on an FPGA. *Midwest Symposium on Circuits and Systems* 2017-Augus (2017), 281–284. Issue 1. https://doi.org/10.1109/MWSCAS.2017.8052915