A Bop and Beyond: A Second Order Optimizer for Binarized Neural Networks

Cuauhtemoc Daniel Suarez-Ramirez, Miguel Gonzalez-Mendoza, Leonardo Chang,
Gilberto Ochoa-Ruiz, Mario Alberto Duran-Vega
Department of Computer Science
Tecnologico de Monterrey, School of Engineering and Sciences
Monterrey, NL
a01206503@exatec.tec.mx, mgonza@tec.mx, lchang@tec.mx,
gilberto.ochoa@tec.mx, a00755076@exatec.tec.mx

Abstract

The optimization of Binary Neural Networks (BNNs) relies on approximating the real-valued weights with their binarized representations. Current techniques for weight-updating use the same approaches as traditional Neural Networks (NNs) with the extra requirement of using an approximation to the derivative of the sign function - as it is the Dirac-Delta function - for back-propagation; thus, efforts are focused adapting full-precision techniques to work on BNNs. In the literature, only one previous effort has tackled the problem of directly training the BNNs with bit-flips by using the first raw moment estimate of the gradients and comparing it against a threshold for deciding when to flip a weight (Bop). In this paper, we take an approach parallel to Adam which also uses the second raw moment estimate to normalize the first raw moment before doing the comparison with the threshold, we call this method Bop2ndOrder. We present two versions of the proposed optimizer: a biased one and a bias-corrected one, each with its own applications. Also, we present a complete ablation study of the hyperparameters space, as well as the effect of using schedulers on each of them. For these studies, we tested the optimizer in CIFAR10 using the BinaryNet architecture. Also, we tested it in ImageNet 2012 with the XnorNet and BiRealNet architectures for accuracy. In both datasets our approach proved to converge faster, was robust to changes of the hyperparameters, and achieved better accuracy values.

1. Introduction

Artificial Intelligence (AI) is having a great momentum in terms of new applications and positive impacts on society. Deep Learning (DL) is a sub-area of AI that has demonstrated outstanding capabilities for solving complex tasks in many areas. In computer vision in particular, it has out-performed previous approaches in tasks such as Image Classification, Image Recognition, and Image Segmentation[16, 10, 26, 24, 18].

Current approaches for optimizing DL methods (i.e. for edge computing applications) are either based on constructing and training lighter neural networks or pruning larger ones. In particular, in regard to lighter neural networks, approaches based on Binarized Neural Networks (BNNs), which uses weights constrained to $\{-1, +1\}$, result in models which are much less computationally expensive, and lead to noticeable reductions in energy consumption when implemented on specialized hardware, and far less memory usage. Moreover, the nature of the BNNs impacts other types of applications such as for Neuromorphic Computing [13, 15, 28, 19] and Quantum Computing [9], thus, highlighting the importance of these type of networks.

Anderson and Berg [1] proved theoretically and experimentally that BNNs maintain the geometrical properties of the Convolutional Neural Networks (CNNs), specifically the properties of the convolution operation. This means that the angle between a stochastic vector and its binarized counter-part converges to a small value with an increasing number of dimensions. Also, the matrix-product is preserved.

BinaryNet [4] was the pioneering work proving that BNNs are viable for complex tasks such as image classification on ImageNet [25, 7]. XnorNet [23] proved that the binarized convolution operation could be done by just using `xnor` and `pop-count` operations which dramatically reduces the convolution's complexity. Since then, the subject has attracted attention and various papers have been published for reducing the loss function and training/validation errors, training deeper BNNs, and using multiple binary bases for the matter [17, 5, 6, 8, 14, 21, 2].

Even though BNNs have been greatly improved, the methods for training them have remained mostly unchanged; they use Stochastic Gradient Descent or an equivalent method. Helwegen, *et al.* [11] proposed a Binary Optimizer (Bop) which instead of using the gradients to update a "latent weight", it uses this information to determine when to "flip" the weights, directly training the net with 0s and 1s. This method calculates a raw average of the gradients (first raw moment), and compares it to a threshold to assess when to modify the weight.

The main focus of this paper is to implement an optimizer which only takes into account when to flip the weights sign/values inspired by Adam [12] instead of using adapted full-precision methods. Our contributions are:

1. Introduce a second order optimizer for BNNs which uses the first and second momentum of the gradients. This optimizer yields better results in terms of accuracy (tested on CIFAR-10 and ImageNet 2012) than the start-of-the-art methods including the first order optimizer Bop.

2. Explore the effects of each hyperparameter, and study the effects of using schedulers on them.

With this paper we introduce a specialized optimizer for BNNs which uses the first and second raw moment estimates of gradients to assess when to modify the sign of the binarized weight. This method is based on Bop while introducing the advantages of Adam. The obtained results outperform the other learning methods for BNNs.

2. Background

Consider a neural network, $y = f(x, w)$, with $w \in \mathbb{R}^n$, and a defined loss function $L(y, y_{label})$, where y_{label} is the ground truth (real label). Then, the binarization problem is defined as:

$$w^*_{bin} = \underset{w_{bin}\in\{-1,+1\}^n}{\arg\min} \quad \mathbb{E}_{x,y}[L(f(x, w_{bin}), y_{label})] \quad (1)$$

As global optimums usually cannot be found, approximate solutions via Stochastic Gradient-Descent (SGD) are used instead. The problem arises when evaluating the gradient $\frac{\partial L}{\partial w}$ depends on $\frac{\partial w_{bin}}{\partial w}$. During the forward pass, the binarization of the inputs and weights is achieved by using the $sign$ function:

$$w_{bin} = sign(w) \quad (2)$$

As the gradient $sign$ function is the Dirac Delta function, it vanishes on every point except on 0 (as seen in Figure 1). Thus, approximations must be used for calculating the derivative and use it for the gradients of the weights.

One of the most common approximations is the Straight-Through Estimator (STE) [29] which is defined as:

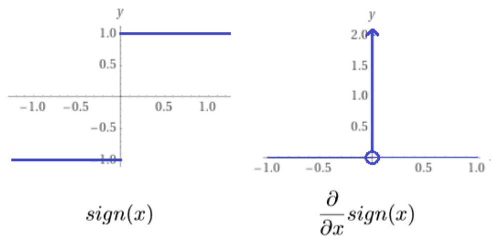


Figure 1: Sign function and its derivative, the Dirac delta function. The derivative banishes everywhere but in 0.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w_{bin}} 1_{\|w\| \le t_{clip}} \quad (3)$$

In other words, it permits the gradient to go through, except for those values where the weights have a large magnitude. The most common case is to use $t_{clip} = 1$, but Bethge, *et al.* [2] tested various values and found that values between 1.25 and 1.5 work better.

The first approaches towards improving the optimization methods for BNNs used different approximations to the sign function so it could be used for back-propagation. Some of these approaches use a first or second order polynomial approximation [17] to the sign function or extra mathematical operations to consider the magnitude of the weight or the gradient. Still, the most common approximation is the STE or Clip [29, 23, 3, 4, 22, 27] and the ApproxSign [17]. Recently, Bethge *et al.* [2] proved that there is no need for using a different approximation to STE, but the threshold must be tuned with values different from 1.

2.1. Real-valued approximation with binarized weights

The most common algorithm for training BNNs is described in 1.

Algorithm 1 Traditional training algorithm for BNNs using Stochastic-Gradient Descent on latent weights as presented by Helwegen *et al.* [11]

```
1: input: Loss function $L(f(x, w), y)$, Batch size $K$
2: input: Optimizer $\mathcal{P} : g \mapsto \delta_w$, learning rate $\alpha$
3: input: Pseudo-Gradient $\Phi : L(f(x, w), y) \mapsto g \in \mathbb{R}^n$
4: initialize $w \leftarrow w_0 \in \mathbb{R}^n$
5: while stopping criterion not met do
6:     Sample mini-batch $\{x^{(1)}, \dots, x^{(K)}\}$;
7:     Gradient: $g \leftarrow \frac{1}{K}\Phi \sum_k L\left(f(x^{(k)}; w), y^{(k)}\right)$;
8:     Update latent weights: $w \leftarrow w + \alpha\mathcal{P}(g)$;
```

Conceptually, this seems as a flawed approach as the only aspect that is enforced is to find when to change the magnitude of the weight, requiring the use of some pseudoweight to be optimized and then binarized. One of the problems of using this approach is that the gradient with respect to the binary weight is not enough to trigger the change of its sign [17]. Therefore, a magnitude-aware gradient was proposed by Liu *et al*. which takes into account the magnitude of the gradient for changing the weights.

The problem of using "latent weights" as a way of storing a non-real value by proxy that will be optimized upon for obtaining the real binary weight, was tackled by Helwegen *et al*. They demonstrated in their paper that the binarization problem could actually be seen as the sign of the weight multiplied by its magnitude (momentum). Thus, latent weights were not needed as the problem was almost identical to using momentum for training full-precision weights.

3. Related Work

There has been various attempts to adapt full-precision optimizers to BNNs, but only one that directly tackles the problem of building a specialized one, Binary Optimizer (Bop), proposed by Helwegen *et al*. [11]. This method is based on the concepts of momentum, and more notably, it showed that only flipping the sign of the weights is enough for correctly binarized the weights.

If the sign of the weights is all that matters, an optimization algorithm that only decides when to flip values makes a more natural manner to learn the weights. Helwegen *et al*. [11] realized this, and proposed the idea to design a momentum-like algorithm that stores the gradients of each weight (with an exponential decay) and when the accumulated value surpasses a threshold, the weight flips. This function is shown in algorithm 2.

Algorithm 2 Bop, an optimizer for BNNs [11]. It uses the first raw moment estimate to decide when to flip the weight value

```
1:  input: Loss function $L(f(x,w),y)$, Batch size $K$
2:  input: Threshold $\tau$, adaptivity rate $\gamma$;
3:  initialize $w \leftarrow w_0 \in \{-1,1\}^n, m \leftarrow m_0 \in \mathbb{R}^n$
4:  while stopping criterion not met do
5:      Sample mini-batch $\{x^{(1)},\dots,x^{(K)}\}$;
6:      Gradient: $g \leftarrow \frac{1}{K}\frac{\partial L}{\partial w}\sum_k L\left(f(x^{(k)};w),y^{(k)}\right)$;
7:      Update momentum: $m \leftarrow (1-\gamma)m + \gamma g$;
8:      for $i \leftarrow 1$ to n do
9:          if $\|m_i\| > \tau$ and $sign(m_i) = sign(w_i)$ then
10:             $w_i \leftarrow -w_i$;
```

The latent weights are better understood when thinking of the magnitude and the sign separately:

$$\begin{aligned}\tilde{w} &= sign(w)\cdot\|w\| := w_{bin}\cdot m,\\ w_{bin} &\in \{-1,+1\}, \quad m \in [0,\infty)\end{aligned} \tag{4}$$

The main rationale for this approach is that the latent weights encode the inertia values m of the binary weights w_{bin}. The bigger the magnitude is, the stronger the effect of this binarized weight. Thus, the gradient can be seen as an indicator of the "necessity" of changing the sign of the weight.

4. Second Order Binary Optimizer (Bop2ndOrder)

As determining when to change the value of a given weight (when to flip it) is the main purpose of the optimization step, our optimizer uses the gradient for this purpose, as outlined above. For this, we analyzed some full-precision optimizers and we found a way to adapt them to obtain this information. The Adam optimizer [12] seemed as the best choice to be adapted both conceptually and practically, due to two characteristics: 1) it resembles the work done in Bop [11], and 2) it further improves it by also using the second raw moment estimate.

As mentioned above, Bop is a first order binarized optimization method that calculates the first raw moment estimate of the gradients in the following way:

$$m_t = (1-\gamma)m_{t-1} + \gamma g_t = \sigma\sum_{r=0}^{t}(1-\gamma)^{t-r}g \tag{5}$$

Then, this value is compared against a pre-defined threshold for deciding when to flip the weight:

$$w_t^i = \begin{cases} -w_{t-1}^i & \text{if } \|m_t^i\| \geq \tau \text{ and } \mathrm{sign}(m_t^i) = \mathrm{sign}(w_{t-1}^i) \\ w_{t-1}^i & \text{otherwise.}\end{cases} \tag{6}$$

One thing to note, is that not only the absolute value of m_t should be higher than the threshold, but the sign of it should be the same as the previous weight. This is due to the gradient indicating the way of the maximum rate of change; if the weight points already in that direction, there is no reason to flip it again.

The natural iteration over this algorithm, is to also include a second raw moment estimate value for regularizing the gradients in the form of:

$$v_t = (1-\sigma)v_{t-1} + \sigma g_t^2 = \sigma\sum_{r=0}^{t}(1-\sigma)^{t-r}g^2 \tag{7}$$

As inspired by Adam, this would normalize the gradient making it invariant to re-scaling, and would make the training smoother and faster (in terms of the number of iterations). Thus, the previous quantity m_t is converted into:

$$s_t = \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{8}$$

Transforming the comparison rule into:

$$w_t^i = \begin{cases} -w_{t-1}^i & \text{if } \|s_t^i\| \geq \tau \text{ and } \operatorname{sign}(s_t^i) = \operatorname{sign}(w_{t-1}^i) \\ w_{t-1}^i & \text{otherwise.} \end{cases} \tag{9}$$

This iteration, a novelty of our method is shown in algorithm 3. If we compare this approach to Adam [12], we notice that no method for correcting bias is presented in the latter. Although, according to the results of Helwegen *et al*. [11] this is not really necessary as the threshold value takes on this role through schedulers. However, we decided to include an unbiased version of the algorithm as it would mitigate overfitting by reducing the bias of the mean and variance raw estimates. This is done by changing (8) to:

$$s_t = \frac{m_t/\gamma}{\sqrt{v_t/\sigma} + \epsilon} \tag{10}$$

Algorithm 3 Second Order Bop. This algorithm uses both the first and second raw moment estimates. Depending on if we choose the biased or unbiased algorithm, is the value of s_t calculated.

```
1:  input: Loss function $L(f(x,w),y)$, Batch size $K$
2:  input: Threshold $\tau$, adaptivity rate $\gamma$, standard rate $\sigma$;
3:  initialize $w \leftarrow w_0 \in \{-1,1\}^n, m \leftarrow m_0 \in \mathbb{R}^n, v \leftarrow v_0 \in \mathbb{R}^n$
4:  while stopping criterion not met do
5:      Sample mini-batch $\{x^{(1)}, \dots, x^{(K)}\}$;
6:      Gradient: $g \leftarrow \frac{1}{K}\frac{\partial L}{\partial w}\sum_k L\left(f(x^{(k)}; w), y^{(k)}\right)$;
7:      Update momentum: $m \leftarrow (1-\gamma)m + \gamma g$;
8:      Update raw variance: $v \leftarrow (1-\sigma)v + \sigma g^2$;
9:      Standardized momentum: $s \leftarrow s\_value(m, v)$;
10:     for $i \leftarrow 1$ to n do
11:         if $\|s_i\| > \tau$ and $sign(s_i) = sign(w_i)$ then
12:             $w_i \leftarrow -w_i$;
13: function $s\_value(m, v)$
14:     if biased then
15:         return $\frac{m}{\sqrt{v}+\epsilon}$
16:     else
17:         return $\frac{m/\gamma}{\sqrt{v/\sigma}+\epsilon}$
```

5. Experimental Results

In this section, we tested thoroughly the behavior of the Bop2ndOrder algorithm (both biased and unbiased) exploring the effects of its three hyperparameters (γ, σ and τ) and the effect of scheduling policies (both increasing and decreasing the values) to those hyperparameters. Additionally, we tested whether the batch or layer normalization works best for our optimizer. Lastly, we compared our results against those obtained for Bop.

For the hyperparameters exploration, we used the BinaryNet [4] architecture and tested the code in Google Colab using either Tesla P-100 or V-100 GPUs.

We included the metric of π_t introduced by Helwegen *et al*. [11] which monitors the ratio of weights flipped at each step. It is defined as:

$$\pi_t = \log\left(\frac{\text{\# flipped weights at time } t}{\text{Total number of weights}} + e^{-9}\right) \tag{11}$$

We used this evaluation parameter for analyzing the effects of each of the hyperparameters γ, σ and τ.

5.1. Second Order Binary Optimizer (Bop2ndOrder)

In this section, we present the complete ablation studies, comparisons, and tests on CIFAR10 and ImageNET of our optimizer.

5.1.1 Biased or Unbiased. Batch or Layer Normalization

As previously stated, the Second Order Bop can be formulated in an unbiased or a biased case. Batch and Layer Normalization can also be used. In order to test this, in Figure 2 we present a combination of the 4 possible cases. For choosing the correct hyperparameters, we did a search over all possible combinations of powers of 10 of each of the hyperparameters arriving to the combination $\gamma = 1e-7, \sigma = 1e-3, \tau = 1e-6$.

In the original work [11], they used Batch Normalization (BN) in all of the tested architectures. Recently, Bethge et al [2] compared this layer against Layer Normalization (LN) in their Binary Dense Net architecture obtaining better results with LN. Also, Nayak et al [20] tested this idea using the Bop optimizer with the BinaryNet architecture in CIFAR10 obtaining similar accuracies but less cross-entropy loss with LN.

As shown in Figure 2, the best results for the validation accuracy (top-1) are obtained when combining the BN with the unbiased version of the algorithm. As a matter of fact, the accuracy values for each experiment are not as far apart from each of the cases. The real issue is that using LN introduces lower losses without impacting the accuracy.

5.1.2 Hyperparameters exploration

To understand the effect of each of the three hyperparameters, we performed an ablation study based upon the optimal values that we previously obtained. Also, we implemented different schedulers for each of the hyperparameters.

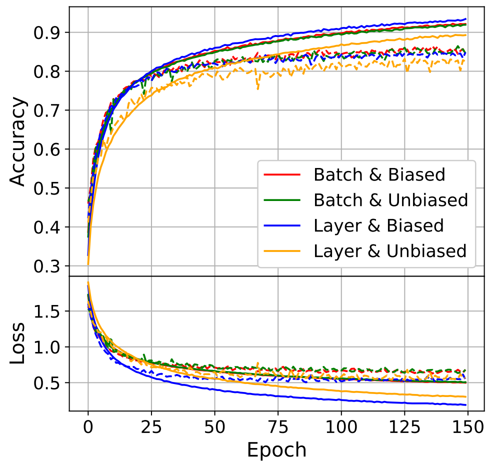


Figure 2: Bop2ndOrder tested with Batch Normalization and Layer Normalization combined with the Biased and Unbiased algorithms using the hyperparameters $\gamma = 1e-7$, $\sigma = 1e-3$, $\tau = 1e-6$ and a batch size of 100. The line (-) refers to the training and the dashed line (--) refers to the validation

Ablation Studies.
Taking as the starting point the "optimal" values that we obtained for CIFAR-10 (100 epochs, BinaryNet, batch size of 100), we tested two magnitudes of 10 lower and two higher for each hyperparameter while letting the values of the other two remain constant.

In figure 3, we show the results of this exploration. γ and τ have similar effects to the learning rate (as expected from Bop [11]). σ works differently as its impact on accuracy is negligible. Its effect relies on increasing or decreasing the number of bit-flips. This could be used for stabilizing the network after some epochs by decreasing the value of this hyper-parameter.

The effect of schedulers.
In order to get a better grasp of the behaviour caused by these hyperparameters, we tested exponential schedulers (both increasing and decreasing by factors of 10 every 100 epochs show in figure 4), for 350 epochs for the biased and the unbiased versions of the algorithm. For the base values, we chose the ones previously obtained $\gamma = 1e-7, \sigma = 1e-3, \tau = 1e-6$,

In figure 4, we show the results of applying these schedulers in the unbiased version of the algorithm. As expected, decreasing or increasing γ directly affects how well the network learns. Decreasing it affects positively while increasing the value causes the algorithm to behave worse for a period of time before trying to set to the new value. For σ both scheduling policies affect the hyperparameter equally; thus, it seems to be only a normalizing parameter. Lastly, τ seems to not affect the accuracy in a major way; this is counter-intuitive as having a higher value should avoid more weights to flip values and vice versa. Thus, we decided to explore the same schedulers with the biased algorithm. The behavior remains largely the same (for γ and σ), but τ does affect the behavior of the algorithm; increasing the value of the threshold positively affects the accuracy. This could be explained by thinking of τ as a regularizer value by restricting or enabling more weights to be flipped. Thus, we want the algorithm to be more stable towards the end by increasing the value of the threshold.

With these results, we can hypothesize that the biased version is more akin to be tuned by the hyperparameters than the unbiased version. Thus, to fine-tune the network, the biased algorithm should work better.

5.1.3 Bop vs. Bop2ndOrder

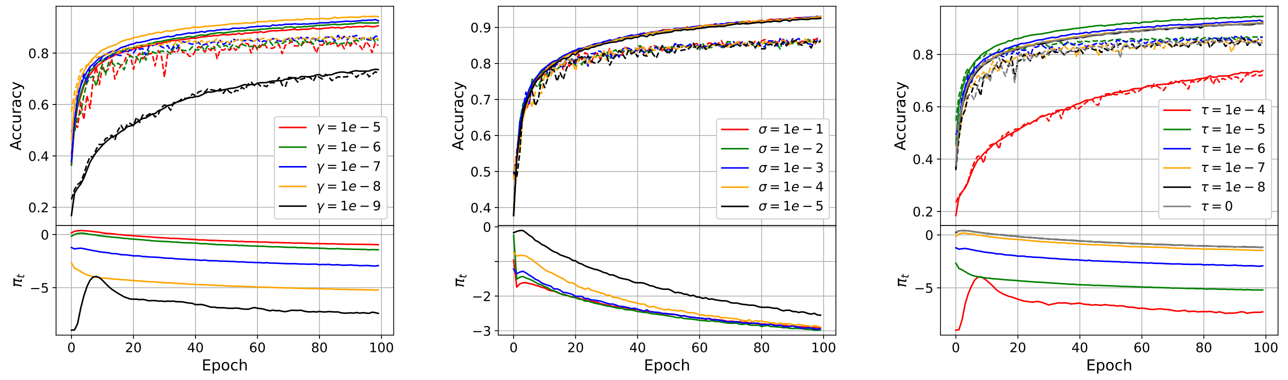
Bop2ndOrder has doubled the full-precision values as it is storing the second order momentum; however, the training times are increased by 15%. Considering this, we compared both optimizers (with their optimal hyperparameters). This is shown in Figure 5. As it can be seen, Bop2ndOrder is marginally superior with a validation accuracy (top-1) of **85.6%** against the Bop accuracy of **79.6%**. Also, it can be seen that Bop2ndOrder tends to overfit (Adam [12] exhibits the same problem). Thus, a threshold increasing scheduler was used as a mean of coping with this issue.

5.2. CIFAR-10

In order to carry out a fair comparison with Bop [11], we used the BinaryNet architecture running for 500 epochs and using a batch size of 50. The difference here is the type of schedulers used. In our case, we used a polynomial scheduler with $\gamma = 1e-5 \rightarrow 1e-8, \sigma = 1e-2 \rightarrow 1e-5, \tau = 1e-7 \rightarrow 1e-2$ and a learning rate for the Adam optimizer of 0.01 that goes to 0.001 (polynomially) and otherwise default settings ($\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1e-7$). Also, we run the experiments with both the biased and the unbiased version of the algorithms.

In figure 6, we show the results of this run. The validation accuracy (top-1) obtained for the biased version was of **91.9%** and **91.4%** for the unbiased one. Both results are higher than the Bop result of **91.3%**. Running the given code for the Bop algorithm we obtained and accuracy of **91.0%**.

Both of our results only reach the best values at the last iterations - which is a behavior also presented with ImageNet -, while the Bop algorithm reaches an stable result 100-150 epochs before that. Also, the unbiased version is much more steady while the biased version is erratic. The results are summarized in Table 1.



(a) Testing different γ values leaving $\sigma = 1e-3, \tau = 1e-6$

(b) Testing different σ values leaving $\gamma = 1e-7, \tau = 1e-6$

(c) Testing different τ values leaving $\gamma = 1e-7, \sigma = 1e-3$

Figure 3: Results of testing diverse hyperparameter values for Bop2ndOrder (unbiased) in BinaryNet for 100 epochs.

Optimizer	Training Acc	Validation Acc
Bop [11]	96.7%	91.0%
Bop2ndOrder Unbiased (ours)	**98.3%**	91.5%
Bop2ndOrder Biased(Ours)	98.1%	**91.9%**

Table 1: CIFAR-10 comparison between optimizers using Binary-Net [4]

Optimizer	Acc	XnorNet [23]	BirealNet [17]
Bop2ndOrder (ours)	Top-1	**46.9%**	**57.2%**
	Top-5	70.9%	79.5%
Bop [11]	Top-1	45.9%	56.6%
	Top-5	70.0%	79.4%
Latent weights	Top-1	44.2%	56.4%
	Top-5	69.2%	79.5%

Table 2: ImageNet comparison between optimizers using two common BNNs.

5.3. ImageNet

We tested Bop2ndOrder on ImageNet using the binarized networks: XnorNet [23] and BirealNet [17]. We also tried training with the BinaryNet architecture [4], but the increase in memory usage caused by storing the second order values proved to be high enough not to be feasible in a personal computer. The tests were done in a computer with 8 NVIDIA Tesla-P100 GPUs.

We trained XnorNet for 100 epochs and BirealNet for 150 epochs with a batch size of 1024, and standard preprocessing with random flips and resize. This is the same as in Bop [11] for comparison purposes.

For the hyperparameters, we used polynomial schedulers with $\gamma = 1e-4 \rightarrow 1e-9, \sigma = 1e-5 \rightarrow 1e-2, \tau = 1e-8 \rightarrow 1e-5$ and a learning rate for the Adam optimizer of $2.5e-3$ that goes to $5e-6$ (polynomially), and otherwise default settings ($\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1e-7$). In the case of XnorNet, we also used the l2-regularization of $5e-7$ that was used in Bop [11]. The obtained results are summarized in Table 2.

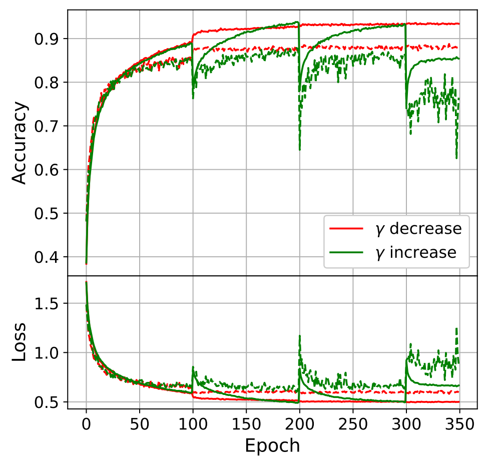
To further analyze the behavior of our optimizer, we also trained both architectures for 300 epochs with the unbiased version. In Figure 7 (orange line) the results for the run using XnorNet are shown. The obtained validation accuracies are: **46.9%** (top-1) and **71.3%** (top-5). Still, we get better results than Bop (1% in accuracy).

In Figure 7 (blue line) the results for the run using BirealNet are shown. The obtained validation accuracies are: **56.7%** (top-1) and **79.4%** (top-5). We have a 0.1% advantage over Bop, almost anecdotical.

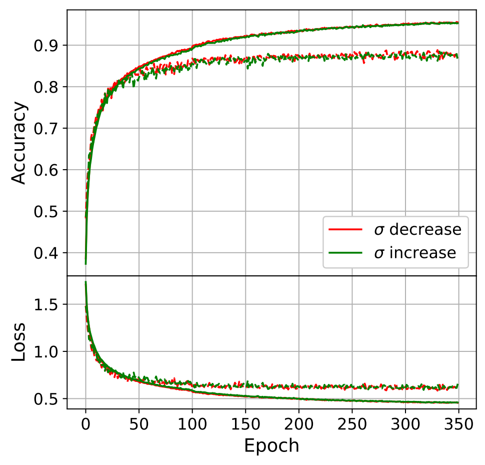
There is a interesting fact about the graph of BirealNet, at 300 epochs the trend is an "exponential" growth; thus, we decided to freeze the hyperparameters and train for 30 epochs more. With these new epochs, the results are: **57.1%** (top-1) and **79.6%** (top-5). We have a 0.5% advantage over Bop.

6. Discussion

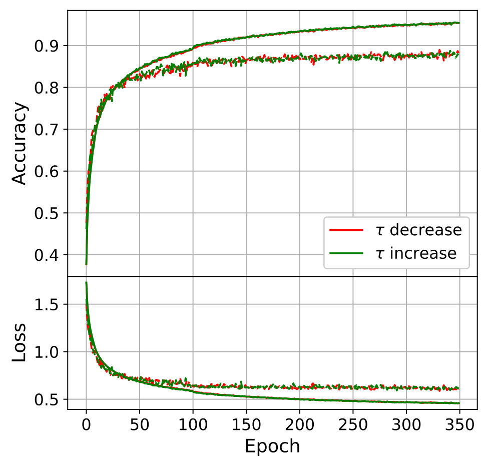
Referring to the results with CIFAR10 (Table 1) both of our algorithms present better results. Also, the biased version has higher validation accuracy even though its behavior is more chaotic. This could be due to its more tunable capacity as it is more affected by the modification of the



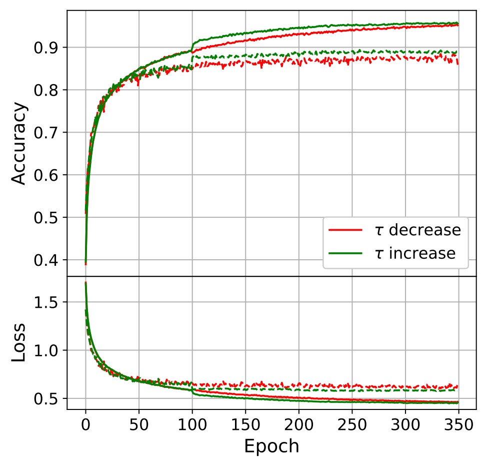
(a) Exponential Scheduler (by powers of 10) applied to γ in the unbiased version of the algorithm.



(b) Exponential Scheduler (by powers of 10) applied to σ in the unbiased version of the algorithm.



(c) Exponential Scheduler (by powers of 10) applied to τ in the unbiased version of the algorithm. Unbiased algorithm. The scheduler does not change the behavior.



(d) Exponential Scheduler (by powers of 10) applied to τ in the unbiased version of the algorithm. Biased algorithm. Here, the scheduler does change the behavior.

Figure 4: Exponential schedulers applied to the optimal hyperparameters of Bop2ndOrder unbiased and biased. For γ and σ the behavior is the same for both. For τ, we have different effects on both algorithms.

hyperparameters (as previously discussed).

With respect to the ImageNet results (Table 2) again our algorithm present better results than both Bop (by **0.6%** with BirealNet and **1.0%** with XnorNet) and the latent weights algorithms.

Our algorithm presents a peculiar behavior in both datasets (CIFAR-10 and ImageNet 2012) where the occurs a sudden increase in accuracy during the last epochs of the training. This could be due to the optimizer suddenly escaping from a local minimum, but there is no certainty of the reason of this behavior since this trend did not continue after increasing the number of iterations.

6.1. Robustness

While testing different configurations of the hyperparameters on BinaryNet trained for 100 epochs with CIFAR-10, we noticed that the hyperparameters can be changed in power of 10 or less without having great impact in the validation accuracy, at least for the unbiased algorithm. The only thing to take into account is that the proportion be-

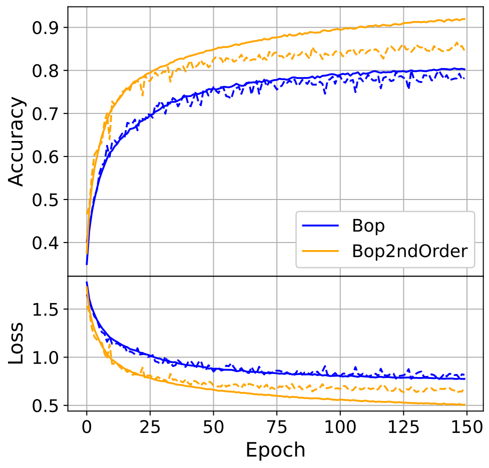


Figure 5: Bop and Bop2ndOrder compared in CIFAR10 using BinaryNet trained for 150 epochs. The line (-) refers to the training and the dashed line (- -) refers to the validation

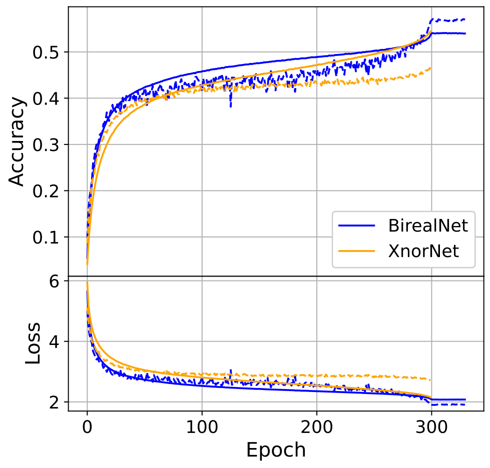


Figure 7: Optimum ImageNet using Bop2ndOrder and both the XnorNet and the BirealNet architectures. The line (-) refers to the train accuracy and the dashed line (- -) refers to the validation one

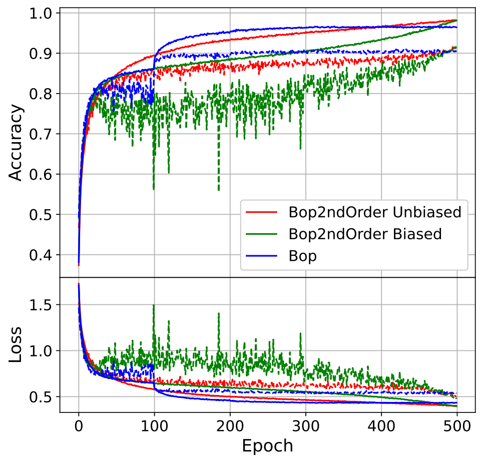


Figure 6: Optimum CIFAR10 run using Bop2ndOrder (biased and unbiased) compared to the Bop run. The line (-) refers to the training and the dashed line (- -) refers to the validation

tween hyperparameters must be maintained. This indicates that Bop2ndOrder is robust enough to perform almost optimally when the exact hyperparameters are now known (at least for these experiments).

7. Conclusions and Future Work

In this paper we presented a new algorithm based on the interpretation of the BNN training methods as latent-weights encoding inertia [11]. Thus, we decided to take this approach (parallel to that of momentum) and offer a second order approach which also uses the second raw moment estimate akin to Adam [12]. With this new optimizer, we have surpassed the state-of-the-art results of BinaryNet on CIFAR-10, and the same for XnorNet and BirealNet on ImageNet 2012.

In general, the results surpass those presented in previous methods for optimizing BNNs. The caveat is that Bop2ndOrder uses more memory and training time (between 15% - 25% for the biased version and between 20 - 32% for the unbiased one) than the other binary optimizer, Bop. Instead of choosing one over the other, there could be the case that both are used similar to how for training full-precision CNNs one optimizer (normally Adam) is used before another with less overfitting (such as RMSprop). Here, as Bop2ndOrder achieves faster a higher accuracy, could be used for the first epochs, and then change to Bop in order to stabilize the network and get less overfitting.

One exciting development that we foresee in the near future for these type of optimizers, is the introduction of specialized regularizers for BNNs (acting on the binary weights). Thus, Bop2ndOrder would be highly improved by using these methods as the training accuracies are way beyond those obtained using previous attempts.

References

[1] Alexander G. Anderson and Cory P. Berg. The high-dimensional geometry of binary neural networks, 2017.

[2] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Back to simplicity: How to train accurate bnns from scratch?, 2019.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2015.

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016.

[5] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Regularized binary network training, 2018.

[6] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. BNN+: Improved binary network training, 2019.

[7] Jia Deng, Wei Dong, Richard Socher, Lia-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[8] T. Ceren Deveci, Serdar Cakir, and A. Enis Cetin. Energy efficient hadamard neural networks, 2018.

[9] Abdulah Fawaz, Paul Klen, Sebastien Piat, Simone Severini, and Peter Mountney. Training and meta-training binary neural networks with quantum computing. pages 1674–1681, 07 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization, 2019.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[13] Olga Krestinskaya and Alex Pappachen James. Binary weighted memristive analog deep neural network for near-sensor edge processing, 2018.

[14] Fayez Lahoud, Radhakrishna Achanta, Pablo Márquez-Neila, and Sabine Süsstrunk. Self-binarizing networks, 2019.

[15] Corey Lammie, Olga Krestinskaya, Alex James, and Mostafa Rahimi Azghadi. Variation-aware binarized memristive networks, 2019.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[17] Zechun Liu, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Binarizing deep network towards real-network performance, 2018.

[18] Octavio Loyola-González, Miguel Angel Medina-Pérez, and Kim-Kwang Raymond Choo. A review of supervised classification based on contrast patterns: Applications, trends, and challenges. *Journal of Grid Computing*, 10 2020.

[19] Sen Lu and Abhroni Senguputa. Exploring the connection between binary and spiking neural networks. *Frontiers in Neuroscience*, 14:535, 2020.

[20] Nancy Nayak, Vishnu Raj, and Sheetal Kalyani. A comprehensive study on binary optimizer and its applicability. *ReScience C*, 6(2), 2020. Accepted at NeurIPS 2019 Reproducibility Challenge.

[21] Jorn W. T. Peters and Max Welling. Probabilistic binary neural networks, 2018.

[22] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, Sep 2020.

[23] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016.

[24] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[26] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[27] Taylor Simons and Lee Dah-Jye. A review of binarized neural networks. *Electronics*, 8:661, 06 2019.

[28] Gopalakrishnan Srinivasan and Kaushik Roy. Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing, 2019.

[29] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets, 2019.