[Re] Latent Weights Do Not Exist: Rethinking Binarized Neural Network optimisation

Nik Vaessen
vaessen@kth.se

Peter Mastnak
mastnak@kth.se

Sri Datta Budaraju
budaraju@kth.se

Abstract

This report examines the reproducibility of the paper "Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization" [10]. This paper proposes a novel view on how binary neural networks are trained and proposes a new optimisation algorithm called *Bop* (Binary optimisation) implemented in TensorFlow. By re-implementing *Bop* in PyTorch, we reproduce experiments that study the effect of the hyperparameters τ and γ used by the algorithm. We also reproduce training a binary neural network on CIFAR-10 with *Bop* and achieve similar accuracy. Our code repository can be found at https://github.com/nikvaessen/Rethinking-Binarized-Neural-Network-Optimization.

1 Introduction

Contemporary deep neural networks are trained and deployed on GPU-accelerated computing instances hosted in data centers. While achieving groundbreaking results in fields such as Computer Vision [8], Speech Recognition [1], NLP [6, 15] and Reinforcement Learning [11], they require expensive GPU hardware while also consuming a large amount of energy [3]. By restricting weights and activations to $\{-1, 1\}$, binary neural networks (BNNs) have a substantially lower energy requirement at run-time compared to equivalent architectures with real-valued weights and activation [5]. This allows them to be deployed in situations where computational power, energy and network access to data centers are limited. Applications such as self-driving vehicles, autonomous drones, mobile devices in areas with underdeveloped infrastructure or robotic space exploration meet those constraints and could benefit from deploying BNNs [10].

Currently, BNNs are trained with "latent weights", a term proposed by [10]. A real-valued network with latent weights $\theta_r \in \mathbb{R}$ is initialized. During the forward pass the binary weights $\theta_b = sign(\theta_r)$ are used to compute a prediction. In the backward pass, the gradient $\frac{\delta \mathcal{L}}{\delta \theta_r}$ of the loss function $\mathcal{L}$ with respect to real-valued, latent weights θ_r is computed. Consequently θ_r can be updated with an optimisation strategy such as SGD or Adam. After convergence, the binary weights $\theta_b = sign(\theta_r)$ represent the trained BNN. See section 2.1 for a detailed description.

[10] proposes a novel view on latent weights and presents a state-of-the-art method for BNN optimisation called Binary optimisation *(Bop)*. They argue that a latent weight $w_r \in \theta_r$ can be better understood as

$$w_r = |w_r| \cdot sign(w_r) := magnitude \cdot \{-1, 1\}$$

that is, it defines both a magnitude and a sign. The sign represents the binary weight w_b in the network, while the magnitude encodes some inertia against flipping the sign of w_b during the training process. Based on this view, *Bop* trains a BNN by keeping track of the moving average m with adaptivity rate γ of the gradient $\frac{\delta \mathcal{L}}{\delta w_b}$. When m exceeds a threshold τ, the sign of w_b is flipped. This process is simpler than the real-valued counterpart in that involves only 2 hyperparameters to tune (γ

and τ). It also requires less memory during training compared to optimizing a real-valued weight with SGD and momentum, or Adam. See section 2.2 for a detailed description.

This article aims to report on the reproducibility of [10], hereafter referred to as **the original paper** or [10]. To reproduce the experiments of the original paper the *Bop* algorithm was re-implemented in the PyTorch framework from scratch. We reproduced section 5.1 (the effect of the hyperparameters τ and γ) and section 5.2 (training on CIFAR-10) of the original paper while ignoring section 5.3 (training on ImageNet). We also conducted some additional experiments only briefly mentioned in the original paper which assisted the authors of [10] to make some theoretical claims.

This reproducibility report is structured as follows. Section 2 contains a detailed description of latent-weight and *Bop* optimisation. Section 3 describes the experiments which were conducted to verify reproducibility and section 4 reports on the results of those experiments. Lastly, in section 5 the results are analyzed and discussed while section 6 concludes the reproducibility report.

2 Methodology

2.1 Binary Neural Networks

This subsection gives a brief overview of the existing literature on the binarization of neural networks. The principle idea is to convert the real-valued weights, as well as the activations, to either -1 or +1. To binarize these real-valued weights both [5] and [4] propose the following binarization function.

$$x_{bin} = sign(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

where x_{bin} is the binary value and x is the real value.

However, using the sign as the binarization functions lead to problems during backpropagation as the derivative of the sign function is going to be zero. To overcome this problem Hinton in his 2012 lecture 15b proposed a "Straight-Through Estimator" [12]. Courbariaux et al. in [4] use a similar implementation of a straight-through estimator as Hinton except that they take into account the saturation effect and they use a deterministic rather than a stochastic sampling of the bit. Given that we got the estimator g_q of the gradient $\frac{\partial C}{\partial q}$ the straight-through estimator of the gradient $\frac{\partial C}{\partial r}$ is then:

$$g_r = g_q 1_{|r| \leq 1} \tag{2}$$

From equation 2 we can see that the gradient is canceled when r gets bigger or equal to 1, which is done to prevent the performance drop [4].

We need to compute the gradients only during the training phase, these latent weights are not used in run-time and thus do not defeat the purpose of binarizing the network. The behavior of the network does not change as long as the sign of the latent weight does not change. These weights can be viewed as inertia for the network and as this inertia increases stronger gradient signal is required to flip the binary weight and thus contributing to the stability of the network. From this point of view, the optimizer is rather altering the inertia of the network rather than the binary weights.

2.2 Binary optimisation

This subsection describes the ideas on BNN training presented in the original paper. The latent weight free binary optimizer referred to as *Bop* is specifically designed for binary neural networks. The optimizer is mainly based on 3 intuitions.

First, as the only goal of the latent weights and the optimizer is to decide whether or not to flip a particular weight, the desired optimizer should be able to clearly address this task. Secondly, gradient signal consistency is an important feature. It could be inferred by taking the previous gradients into consideration by averaging the gradient signals over training steps. Third, strength is equally important as consistency. The strength is the magnitude of the gradient signal and ignoring the weak signal reduces the noise in the training phase. The consistent signals are obtained by the exponential moving average of the gradient over training steps by:

$$m_t = (1-\gamma)m_{t-1} + \gamma g_t = \gamma \sum_{r=0}^{t} (1-\gamma)^{t-r} g_r \tag{3}$$

where, m_t is the moving average at time t, γ is the adaptivity rate and g is the gradient at time t. A higher adaptivity rate makes it easier to adapt to the new gradient. It could, therefore, be related to the learning rate in SGD. We flip the weight if this exponential moving average, the consistency, is more than a certain threshold of τ.

$$w_t^i = \begin{cases} -w_{t-1}^i & \text{if} \quad \left|m_t^i\right| \geq \tau \quad \text{and} \quad \operatorname{sign}\left(m_t^i\right) = \operatorname{sign}\left(w_{t-1}^i\right) \\ w_{t-1}^i & \text{otherwise.} \end{cases} \tag{4}$$

Here w_{t-1} is the weight before computing the exponential moving average from the gradient. Thus we use γ to control the consistency of the gradient to be considered and τ to control the strength of the gradient that would affect the behavior of the binary network.

2.3 Reproducing experiments of [10] with the original paper's code

As a sanity check, we ran the experiments of section 5.1 and 5.2 of [10] using their code, which uses TensorFlow and Larq[1]. We ran the experiments on Google Cloud Platform on Google's Deep Learning Virtual Machine with Cuda 10.0 and 1 NVIDIA K100 GPU. As the Deep Learning Virtual Machine is based on Debian 9, only python 3.5 was available. We got the equivalent results as published in [10].

2.4 Re-implementation in PyTorch

The code accompanied by the original paper only implements the binary optimizer from scratch. The binary convolutional and fully connected layers were taken from Larq. As this library only supports TensorFlow we had to implement the following in our port to PyTorch: the *bop* algorithm, binary fully connected layer, binary 2d convolutional layer, a custom sign function which computes a clipped straight-through estimator as gradient and the metric for keeping track of the number of flips each epoch. Our implementation uses PyTorch Lightning [7] to bootstrap the implementation of the training and testing pipeline.

3 Experiments

3.1 Theoretical claims

The authors of [10] mention two experiments that aided their understanding of latent weight optimisation. In the first experiment, they challenge the approximation viewpoint hypothesis that a binary weight vector acts as an approximation to the real-valued latent weight vector. The second experiment shows empirical proof for a corollary that states that under latent weight optimisation, the learning rate can be set arbitrary for each weight as long as its initialization is scaled accordingly. However, a detailed description of these experiments is not included in the original paper and they are also not included in the accompanying source code. The following subsections detail the steps we took as the best guess for reproducing the experiments.

3.1.1 Approximation viewpoint

To challenge the approximation viewpoint hypotheses, [10] only mentions that they trained a BNN with latent-weight optimisation. Following that, they evaluate the trained network with both the real-valued latent weights and the binary weights while using the sign function as the activation function. They also mention that they retrain the batch-normalization layers on the real-valued latent

[1]See `https://larq.dev`. This framework exposes an API for building BNNs with Keras. It's built and released by the authors of [10].

weight network. They observed that the real-valued latent weights do not perform better than the binary weights, even after retraining the batch statistics.

This experiment was replicated on the MNIST and CIFAR-10 datasets retrieved via the Tensorflow Keras dataset API[2]. All pixel values were normalized between -1 and 1. The network was built with the Larq and Tensorflow Keras framework [2]. The network architecture was 3 binary 2d-convolution blocks with respectively 32, 64 and 64 filters, 3x3 kernel and stride 1. Each convolution block also included a 2x2 max pooling and a batch normalization layer. The convolution blocks were followed by 2 binary linear blocks with 64 and 10 nodes. Each binary linear block also included a batch normalization layer. All binary layers had their latent weights clipped to $[-1, 1]$. The activation function was the sign function (Equation 1), which was applied after batch normalization. The last linear layer had a softmax activation instead. Adam was used for optimisation with default *learning rate* $= 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The loss function was sparse categorical cross-entropy. The training was done on a CPU. No train/validation split was applied. The weights of the last epoch were used to calculate test accuracy.

First, a model with binary weights is trained from scratch with latent-weight optimisation for 6 epochs. The test accuracy is then evaluated on both the binary weights and the latent real-valued weights. Then the binary layers are frozen, and the network is retrained for an additional 6 epochs to update the batch normalization statistics. The expectation is that both the real-valued evaluation and the real-valued evaluation with retrained batch statistics perform equal or worse in comparison with the binary weight evaluation.

3.1.2 Arbitrary learning rate

Theorem 1 in paper [10] states that the binary weights of BNNs trained with latent weights are invariant to the learning rate of the optimizer as long as the initial latent weights are scaled accordingly. It is also required that the pseudo gradient Φ mentioned in equation 2 does not depend on the magnitude of the weights.

To show that the performance difference between several learning rates can also be achieved by scaling the initial latent real-valued weights, the following experiment was conducted. A BNN is trained on MNIST and CIFAR-10 with the same architecture and training process as described in section 3.1.1. However, different configurations of Random and Glorot uniform initialization [9], Adam [13] and SGD [16] optimisation with different learning rates were tested:

1. Glorot uniform initialization and SGD with learning rate 0.01
2. Glorot uniform initialization and SGD with learning rate 1
3. Glorot uniform initialization scaled by 0.01 and SGD with learning rate 0.01
4. Uniform random initialization and SGD with learning rate 0.01
5. Uniform random initialization and Adam with learning rate 0.01, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

The expectation is that configuration 2 and 3 perform equally well due to the scaled initialization, while configuration 1 performs differently from 2 and 3. We expect that configuration 1 and 4 perform equally well, as the magnitude of the weights (which differs between initialization strategies) does not influence the pseudo gradient.

3.2 Effect of hyperparameters

Binary optimisation, as described in section 2.2, requires two hyperparameters: The threshold τ and the moving average adaptivity rate γ. To explore the behavior of these hyperparameters, a network was trained with 6 different combinations: A constant $\tau = 10^{-5}$ with varying $\gamma = [10^{-1}, 10^{-2}, 10^{-3}]$ and a constant $\gamma = 10^{-2}$ with varying $\tau = [0, 10^{-4}, 10^{-5}]$. These choices differ to the hyperparameters examined in the original paper by a single magnitude [3]. The network and training procedure were identical to what is described in section 3.3 except for the number of epochs. During training, the

[2]See `https://www.tensorflow.org/api_docs/python/tf/keras/datasets`.

[3]A mistake was made while interpreting the numbers by writing 10e-x instead of 1e-x in the code. This is expected to be fixed before the final submission.

number of flips in the sign of the weights per optimisation step was monitored. Because this value is very large at the start of training but tends to zero, the value is reported in the natural log domain as

$$\pi_t = ln(\frac{\textit{sign changes at step t}}{\textit{total number of parameters}} + 10^{-9}) \qquad (5)$$

This equation is described in the original paper. The authors of [10] also add the term 10^{-9} to prevent a natural log of 0 when a step does not result in any weight flip.

3.3 Training on CIFAR-10

The authors of [10] trained a VGG-inspired binary neural network on CIFAR-10. The exact architecture was unclear and was retrieved from the original paper's source implementation. The network architecture is as follows:

$$\begin{gathered}(3 \times 128\ BC3) - BatchNorm - (128 \times 128\ BC3) - BatchNorm - MaxPool\\(128 \times 256\ BC3) - BatchNorm - (256 \times 256\ BC3) - BatchNorm - MaxPool\\(256 \times 512\ BC3) - BatchNorm - (512 \times 512\ BC3) - BatchNorm - MaxPool\\1024\ BFC - BatchNorm - 1024\ BFC - BatchNorm - 10\ BFC - BatchNorm - Softmax\end{gathered}$$

$x \times y$ BC3 is a binary convolutional layer with a x input channels, y output channels, 3x3 kernel and stride 1. x BFC is a binary fully connected layer with x units. Activation is accomplished by the fact that all convolutional and fully connected layers apply the sign function (equation 1) on their input except for the first layer, which accepts the real-valued normalized input data.

The data was received from the torchvision datasets API[4]. The last 10% of the training data was used for validation. The training and test data were preprocessed equivalently to the original paper. The training data was transformed by applying a 4x4 padding, taking a random 32x32 crop, applying a random horizontal flip with probability 0.50 and normalizing between $[-1, 1]$. The test data was only normalized between $[-1, 1]$. Binary weights were initialized by first creating real-valued weights with He-initialization[5] and then applying the sign function. *Bop* was used as optimisation method for the binary layers with threshold $\tau = 10^{-7}$ and adaptivity rate $\gamma = 10^{-3}$. as mentioned in [10]. These choices differ to the hyperparameters examined in the original paper by a single magnitude [6]. Similar to the original paper, the adaptivity rate was decayed with 0.1 every 100 epochs. The training was halted manually when no weight flips had occurred for at least 10 consecutive epochs to save computational resources. The batch normalization layers were optimized with Adam with default parameters *learning rate* $= 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The weights with the lowest validation loss were used to compute the test accuracy.

Both the experiments of Section 3.2 and 3.3 were conducted on Google Cloud Platform with using Google's Deep Learning VM and a single NVIDIA V100 GPU with CUDA 10.1, and PyTorch 1.3.0 using our PyTorch re-implementation of *Bop*.

4 Results

4.1 Theoretical claims

4.1.1 Approximation viewpoint

The result of evaluating the test set of MNIST and CIFAR-10 can be seen in Table 1. For MNIST, the binary weights show a mean accuracy of 0.97, which is decreased to 0.10 when using the real-valued latent weights instead. However, retraining the batch statistics makes the latent weights outperform the binary weights with a mean accuracy of 0.98. The CIFAR-10 test accuracies show a similar

[4]See `https://pytorch.org/docs/stable/torchvision/datasets.html`.

[5]This is the default method for linear and convolutional layers in PyTorch.

[6]A mistake was made while interpreting the numbers by writing 10e-x instead of 1e-x in the code. This is expected to be fixed before the final submission.

pattern where the latent weights with retrained batch statistics perform best. Both the reported MNIST and CIFAR-10 test accuracies are significantly different according to a one-way ANOVA test with p-values respectively 1.01e-43 < 0.05 and 3.06e-37 < 0.05.

Table 1: Mean and variance of test accuracy after training a binary CNN on MNIST and CIFAR-10 with 10 random initializations. The *binary* model is evaluated with binarized weights, while the *latent* model is evaluated without binarizing the weights. The *latent+retrained batch norm* model only differs in the batch normalization weights compared to the other 2 models.

	MNIST		CIFAR-10	
model	mean	standard deviation	mean	standard deviation
binary	0.9663	0.0180	0.5731	0.0142
latent	0.1039	0.0018	0.0978	0.0092
latent+retrained batch norm	**0.9807**	0.0013	**0.5942**	0.0050

4.1.2 Arbitrary learning rate

The trends in training the BNN with different configurations for 100 epochs on both MNIST and CIFAR-10 can be seen in Figure 1. We observe that uniform random, Adam and $lr = 0.01$ performs best. It's closely matched in performance by Glorot, SGD, $lr = 1$ and Glorot scaled by 0.01, SGD, $lr = 0.01$. Note that for both data sets the performance of the models with a learning rate of 1 is similar to those that had a learning rate of 0.01 and had their weights rescaled by 0.01. There is no significant difference between using Glorot initialization or uniform random initialization with SGD and learning rate 0.01.

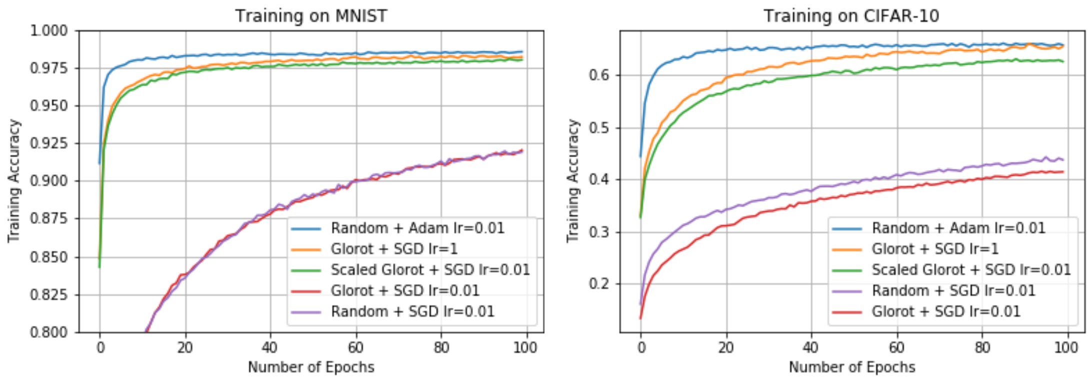

Figure 1: Training BNN on MNIST and CIFAR-10 for 100 epochs with latent weights optimisation for different configurations of optimisation strategies and weight initializations

4.2 Effect of hyperparameters

Figure 2 shows the result of exploring the behavior of γ and τ. We check the training accuracies and the number of flips of different values of γ and τ in *Bop* on CIFAR-10 (train: $-$, validation: $--$). The training accuracy curves were very noisy to illustrate and thus smoothed with a 1D Gaussian filter with a standard deviation of 100. The left column of Figure 2 shows that increasing the adaptivity rate leads to a lower amount of flips. The right column of Figure 2 shows that increasing the threshold also leads to a lower amount of flips. In both accuracy plots (the first row) we see that the highest accuracy is achieved when the amount of flips is neither too high nor too low (seen in orange). When the amount of flips is high (seen in blue) training is noisy and convergence is inhibited. When the amount of flips is low (seen in green) training is stable but slow.

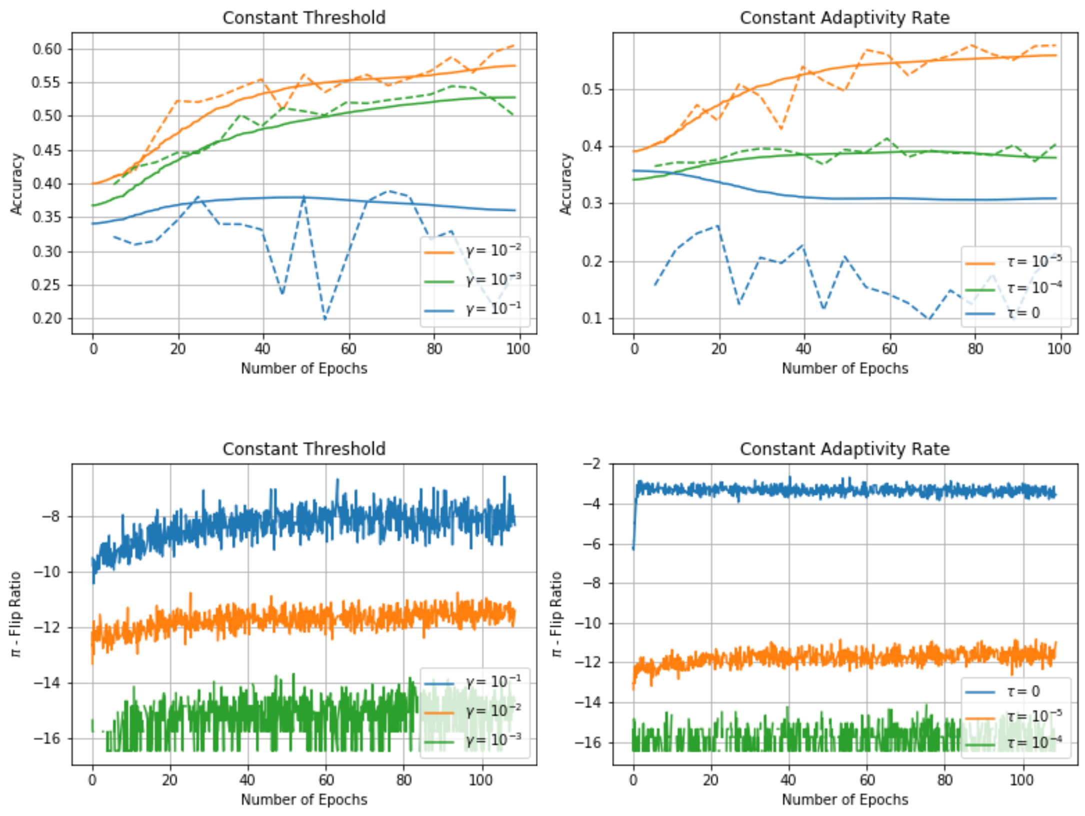

Figure 2: On the left τ is fixed at 10^{-5} and we compare three values for γ. On the right γ is fixed at 10^{-2} and we compare three values for τ. The tensorboard graphs can be found at `https://tensorboard.dev/experiment/up4UbhojT6uZKjnKPyZaRQ`

4.3 Training on CIFAR-10

We trained the described network on CIFAR-10 with slightly different hyperparameters from the original paper twice. The training and validation accuracy of one of the runs can be seen in Figure 3. We can observe that the flip ratio of the entire network falls significantly as the adaptivity rate of γ decreases. After 300 epochs when γ is 10^{-6} no weights are flipped with two exceptions where only 1 weight is flipped across all the layers. Therefore training was stopped after 327 epochs instead of running for 500 epochs. The training and validation accuracy trends similar to the figures from [10]. Our network achieves a test accuracy of **88.67**% and **88.70**% with the best model on the validation set. This is lower than the accuracy of **91.3**% presented in [10]. We also trained a network without a validation split for 320 epochs. The final model had a test accuracy of **89.2**%.

5 Discussion

We were confronted with several issues during re-implementing the code based on the description of *Bop* in the original paper. Without batch normalization, binary neural networks can barely outperform random predictions. When debugging *Bop* we thought that we made a mistake in implementing the algorithm, while we erred by testing on a straight forward 3-layer fully connected network without batch normalization layers. The second issue we ran into when implementing *Bop* was that batch normalization layers are (by definition) non-binary as the mean and variance must be real-valued. Thus implementation of *Bop* must still use a method like SGD or Adam to update the batch normalization parameters and needs to be intertwined with the binary optimisation strategy.

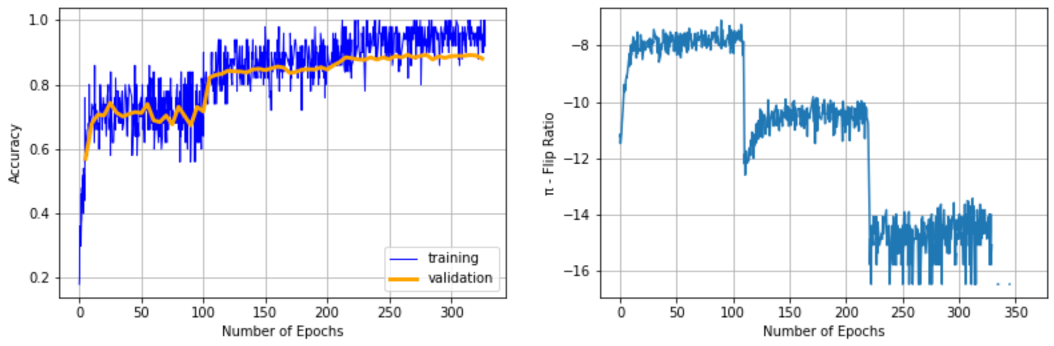

Figure 3: On the left are the training and validation accuracies and on the right are the flip ratios of the overall network while training on CIFAR-10. The tensorboard graph can be found at `https://tensorboard.dev/experiment/OXOeA57LTVWBtYzGRSdwVA`.

Our experiment to disprove the approximation viewpoint was inconclusive. While simply using the real-valued weights instead of the binary weights lead to an accuracy of 10% (random predictions), retraining the batch statistics improved the accuracy by around 1.5% compared to the binary weights on both MNIST and CIFAR-10. Perhaps this experiment should be conducted on more difficult data sets or different, more deep network architectures. Our experiment does not disprove the approximation viewpoint but indicates that batch normalization can recover accuracy after binarizing the weights. This matches the current believe that batch normalization is required for good training convergence of BNNs [14].

From our experiments to verify the arbitrary learning rate, we observe that the effect of scaling the learning rate can be reversed by also scaling the initial weights. We also see that a BNN trained using SGD with a learning rate of 0.01 has almost the same trends when trained with Glorot and uniform random initialization. These observations are more nuanced on MNIST than on CIFAR-10. This supports the statement in the original paper that "*Nevertheless, in experiments we have observed that the advantages of various learning rates can also be achieved by scaling the initialization*" [10]. This is due to the fact that the initialization techniques mostly contribute to the magnitudes of the weights and not the signs. This experiment also supports the statement made in [10] that Adam is preferred over SGD for training a BNN with latent weights.

We didn't reach the same accuracy with our hyperparameters on CIFAR-10 stated in [10]. However, our accuracies are relatively close to the reported accuracy in the original report. We have no reason to believe that using the hyperparameters as reported in the original paper would lead to significantly different result. We expect to show this before the final submission deadline. The *Bop* algorithm was successfully re-implemented solely on the description of the original paper and the hyperparameters behave similarly as described in section 5.1 of [10].

6 Conclusion

We show that the experiments in section 5.1 and 5.2 of [10] are reproducible. The architecture details and implementation of binary layers, both of which are not related to *Bop*, are not directly mentioned in the original paper and had to be found by performing a literature study on binary neural networks. Because the authors of [10] published the source code of their experiments, there is an additional way to gain this information. We were unable to reproduce empirical evidence that the approximation viewpoint hypothesis is incorrect. We will need to contact the authors for more information regarding their experiments and observations on this hypothesis.

Acknowledgments

We would like to thank CodeOcean (under the NeurIPS 2019 reproducibility challenge) and Google Cloud Platform (educational grant for the KTH course *Deep Learning, Advanced course*) for providing computational resources. We would also like to thank Matteo Gamba for providing feedback.

References

[1] Dario Amodei et al. "Deep speech 2: End-to-end speech recognition in english and mandarin". In: *International conference on machine learning*. 2016, pp. 173–182.

[2] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[3] Adam Coates et al. "Deep learning with COTS HPC systems". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1337–1345. URL: `http://proceedings.mlr.press/v28/coates13.html`.

[4] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830 (2016). arXiv: `1602.02830`. URL: `http://arxiv.org/abs/1602.02830`.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *CoRR* abs/1511.00363 (2015). arXiv: `1511.00363`. URL: `http://arxiv.org/abs/1511.00363`.

[6] Jacob Devlin et al. "Fast and Robust Neural Network Joint Models for Statistical Machine Translation". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 1370–1380. DOI: `10.3115/v1/P14-1129`. URL: `https://www.aclweb.org/anthology/P14-1129`.

[7] W.A. Falcon. *PyTorch Lightning*. `https://github.com/williamFalcon/pytorch-lightning`. 2019.

[8] Yuying Ge et al. *DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images*. 2019. arXiv: `1901.07973 [cs.CV]`.

[9] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: `http://proceedings.mlr.press/v9/glorot10a.html`.

[10] Koen Helwegen et al. "Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization". In: *arXiv preprint arXiv:1906.02107* (2019).

[11] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: `1710.02298 [cs.AI]`.

[12] Geoffrey Hinton. *Neural networks for machine learning. In: Coursera Video Lectures (2012).*

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: `1412.6980 [cs.LG]`.

[14] Eyyüb Sari, Mouloud Belbahri, and Vahid Partovi Nia. *How Does Batch Normalization Help Binary Training?* 2019. arXiv: `1909.09139 [cs.LG]`.

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112. URL: `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

[16] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. URL: `http://proceedings.mlr.press/v28/sutskever13.html`.