REPRODUCIBILITY OF EQUI-NORMALIZATION OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Normalization techniques in Deep Neural Networks have become increasingly prevalent to solve the vanishing gradient problem. Equi-Normalization is a new normalization technique that was developed to be superior to the commonly used Batch Normalization in terms of memory efficiency and training time (Stock et al., 2019). We investigate the reproducibility of the Equi-Normalization technique on the fully connected CIFAR-10 network architecture described in the paper. We found that we were able to accurately reproduce the results obtained by the authors. We report our findings as well as some important considerations (input normalization method, inclusion of bias terms, learning rate decay technique) when implementing this technique.

1 INTRODUCTION

Deep Neural Networks (DNN) have had an enormous success across a wide variety of domain applications. In particular, DNNs are used in tasks such as speech recognition (Hinton et al., 2012), image classification (Krizhevsky et al., 2012), and many other interesting applications. The growth in successful applications of DNNs has also led to many improvements in the methods used to train these types of models.

The normalization technique of Equi-Normalization (ENorm) that is introduced by the authors of the paper we are reviewing (Stock et al., 2019) is presented as an improvement over other normalization schemes. Normalization schemes attempt to solve the issue of vanishing/exploding gradients. Vanishing and exploding gradients is a a challenge that arises during the training of deep networks where the gradients computed during the backpropagation become either too small or too large. This has a more significant effect on the first layers in the DNN, and thus slows down the learning process significantly.

Batch Normalization (BN) is a commonly used normalization technique that generally achieves outstanding accuracy (Ioffe & Szegedy, 2015). This technique introduces intermediate layers in the network that normalize the features by utilizing the mean and variance computed within the current batch. BN is particularly good for reducing training time, diminishing the need for careful initialization, and providing better generalization capabilities.

However, BN has a few limitations that make it not optimal for certain memory-consuming networks. In particular, one of the issues with BN is that it must be done on batches of sufficient size, often at least 16 or 32 for it to work in its optimal range. This is particularly limiting in use cases that involve video processing, since the batch size must be restricted to a non-optimal batch size due to memory constraints, which severely impacts the performance of the model. Another limitation to the utilization of Batch Normalization is that it is computationally intensive, typically consuming 20% to 30% of the training time.

Equi-Normalization is an improvement over Batch Normalization in terms of both the training speed and generalization accuracy of networks. The algorithm of Equi-Normalization involves the scaling of two consecutive matrices with rescaling coefficients that minimize the joint *p* norm of these two matrices. The paper (Stock et al., 2019) that introduces this technique also presents a proof of convergence to a unique parameterization of the network that minimizes the global *p* norm of the weights.

2 METHODOLOGY

Due to time and compute resource constraints, this reproducibility study focuses solely on reproducing the results shown in Figure 1 of the fully-connected CIFAR-10 experiments proposed in the original paper (Stock et al., 2019). In particular, we focus on reproducing the results for the baseline, BN, ENorm and ENorm + BN network architectures for odd numbers of intermediary layers from 1 to 19. The baseline model is a network that contains no form of normalization. We use the ENorm algorithm with the optimal value of $c = 1.2$ that the authors report.

These specific experiments were chosen to demonstrate that ENorm is a versatile normalization technique, that could potentially be used in combination with BN to significantly improve the performance of higher layered networks. Furthermore, experiments with networks using linear layers with a bias term were done to show the effects of bias on the performance of the normalization technique. Because bias is used so often, we felt it necessary to observe the effects of adding bias to the network.

The algorithms and networks were implemented in Python with the help of the PyTorch library. The code was written and run using Googles Colaboratory platform, in order to take advantage of the free GPU resources they offer. The code is made public and available at the following URL: `https://github.com/jeromepl/enorm-reproducibility`.

For the baseline case, a DNN was created by using a linear input layer of size 3072x500, followed by *p* intermediary linear layers of size 500x500 before a final linear output layer of size 500x10. Each linear layer, with the exception of the output layer, uses a ReLU activation function and no layers use a bias. The weights are initialized using He's initialization He et al. (2015). The network is trained for 60 epochs using stochastic gradient descent (SGD) without momentum, using a batch size of 256 and a weight decay of $10e^{-3}$. During training, the learning rate is decayed linearly to 0. The input images from CIFAR-10 were normalized by subtracting mean (0.4914, 0.4822, 0.4465) and dividing by the standard deviation (0.2023, 0.1994, 0.2010) computed over the whole training set and independently for each of the channels. The images were then flattened into a size 3072 input. The value of *p* corresponds to the number of intermediary layers and was varied in a range from 1 to 19 in increments of 2 to get a clear picture of the impact of ENorm on progressively deeper networks.

For the BN case, the same architecture as the baseline is used. The only difference is the use of batch normalization on each linear layer with the exception of the output layer.

For the ENorm case, again, the same baseline architecture is used. Algorithm's 1 and 2, as described in the original paper (Stock et al., 2019) were implemented using the PyTorch tensor library. The normalization is thus applied after each training step. Note that since SGD without momentum was used, the updating of momentum buffers, as described in the paper, was not implemented.

The ENorm + BN case is simply a combination of the above two scenarios, where batch normalization is used on all intermediary layers and the input layer, and where ENorm is used to normalize the weights after each training step.

For the baseline and ENorm cases only, the test performances we report correspond to the average of the results from two independent runs. For all other results, the experiment were only ran once because of time constraints.

The optimal learning rates for each of these experiments were kindly provided to us by the authors and are reported in Table 1. However, we modified the learning rates for *p* = 15, 17, 19 in the ENorm + BN case from 0.1 to 0.05 as we found those values to generally perform better.

Table 1: Initial learning rates used for all experiments

p	Baseline	BN	ENorm	ENorm + BN
1	0.1	0.1	0.05	0.1
3	0.1	0.1	0.05	0.1
5	0.1	0.1	0.05	0.1
7	0.1	0.1	0.05	0.1
9	0.05	0.1	0.05	0.1
11	0.05	0.1	0.05	0.1
13	0.05	0.1	0.05	0.1
15	0.05	0.1	0.05	0.05
17	0.05	0.1	0.01	0.05
19	0.05	0.1	0.01	0.05

We also attempted to run these experiments on ENorm and baseline networks with a bias term. The bias was always initialized to zero. All hyperparameters were taken to be the same as in the experiments for ENorm and baseline without bias.

3 RESULTS

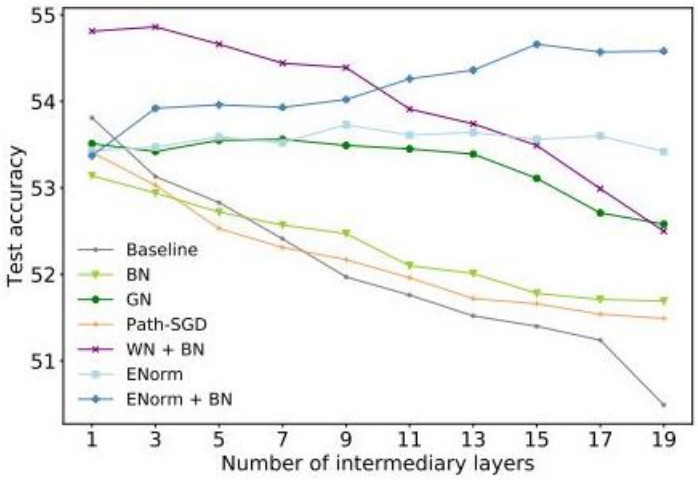

Figure 1: The authors' original results on the CIFAR-10 dataset using a fully-connected neural network architecture. We focus on replicating the plots for Baseline, BN, ENorm and ENorm + BN.

In Figure 2, we can see that for almost all numbers of intermediary layers the accuracy when using the ENorm algorithm is superior to the accuracy achieved when not using any normalization at all. This is consistent with the authors findings and intentions when developing this technique.

For most values of intermediary layers, ENorm + BN is the best performing. It consistently outperforms BN alone. However, we still observed that the accuracy for $p = 19$ was relatively low.

Additionally, our baseline accuracy was slightly higher (approximately 1%) higher than the baseline reported in the original paper.

When bias terms are included at each node, it was found that the network became unstable when attempting to run the ENorm algorithm for larger numbers of intermediary layers. When p exceeded 9, the value for the loss was always *NaN* after a single epoch, and the network was unable to compute gradients correctly and perform updates to weights as a result. These results can be viewed in Figure 3.

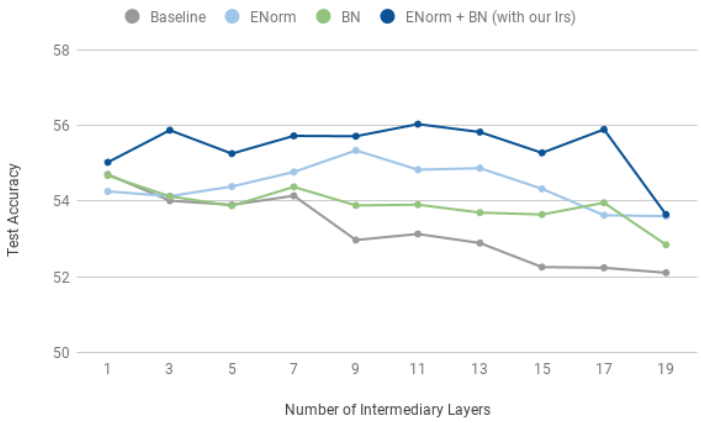

Figure 2: The results from running our implementation of the ENorm algorithm and comparing it to the baseline (no normalization of any kind) for odd values of intermediary layers from 1-19. The results for our implementation when using batch normalization (BN) and when combing BN and ENorm are the results of a single run.

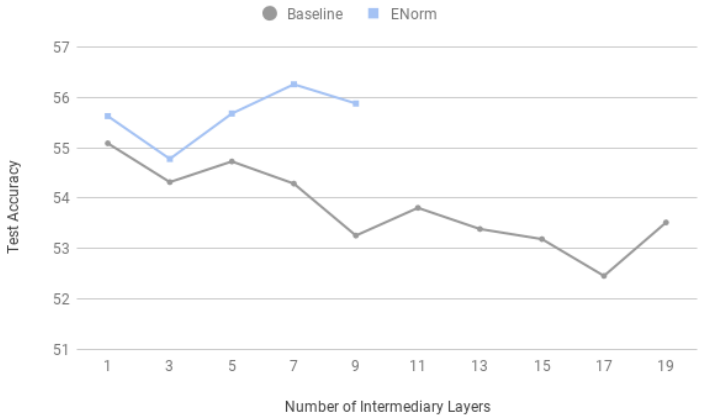

Figure 3: Results for the baseline and ENorm with nodes that included bias terms. When using ENorm with this configuration, there was always NaN loss when the number of intermediary layers was beyond 9.

For the values of p that could be run, it was observed that ENorm still outperformed the baseline case.

The baseline test accuracy for the network that included bias terms was generally slightly better than the baseline for the network that did not include bias terms. When the initial loss was not *NaN* for network that included bias terms, the test accuracy for ENorm also outperformed ENorm in the network with no bias terms.

4 Discussion

4.1 Reproducibility

We found that we were able to successfully replicate the paper and achieve approximately the same results. For most values of p, the best normalization scheme was ENorm + BN, followed by ENorm,

then BN. All the normalization schemes outperform the baseline case. This ranking matches the authors' precisely.

The only place where we observe some divergence from the authors' findings is in the $p = 19$ case. It is likely that this discrepancy is simply due to the fact that this data is the result of a single test, which has a certain amount of variance. Averaging out several training sessions would likely result in BN + ENorm performing slightly better, according to the trend seen in the rest of the data.

We experience occasional instability in the network when running the ENorm procedure with high values of p, which resulted in the loss function evaluating to *NaN*, and leaving the network unable to learn anything. This is due to the lack of an activation function on the output layer of the network. We attempted to reduce this instability by adding a softmax to the final layer. This prevented the loss function from evaluating as *NaN*, but we found that this modification caused the gradient to vanish and again rendered the network unable to learn. This problem was handled by simply restarting the training procedure and allowing the network to initialize with different random weights.

When bias terms were included the network, it became unstable when trying to use ENorm with larger values of p. However, it was noted that for the values of p that could be run, the network that included biases outperformed the network that did not for both the ENorm and the baseline cases. This is logical since the inclusion of the bias terms gives the network greater expressiveness and makes it better able to predict the class labels. It would have been interesting to see the results of the network with higher numbers of intermediary layers, in order to see if the improved performance would hold. Investigating the cause of the instability in this network architecture and a way to stabilize it may represent an interesting area of future research.

Reproducing the work reported in the paper in its entirety would take a substantial amount of time. It would be one of the major constraints to consider if a more in depth reproducibility study were to be performed. To run the code that we used to produce the results presented in this work took about 24 hours of run time. This does not include time spent running code that was tested before obtaining additional details from the authors or for debugging different sections of the code. If time was spent to perform cross validation to select the learning rates, and the other normalization schemes (GN, WN + BN, Path-SGD) were done for comparison, the time taken would have been increased by an order of magnitude. If the other models studied in the paper were also included, it would likely increase the time by another order of magnitude. Note that these timings are related to the hardware we had access to (Google Colab GPUs).

4.2 CHALLENGES

We encountered several issues while implementing this technique that we were unable to resolve until we contacted the authors (anonymously, through openreview.net forums) to clarify sections of the paper.

The first issue we encountered was how the input images from the CIFAR-10 dataset were normalized. In the paper they state "Input data is normalized by subtracting mean and dividing by standard deviation independently for each channel". Our interpretation of this statement was that each image had its own respective mean and standard deviation for each of the channels. However, the mean and standard deviation they are referring to is the mean and standard deviation of the complete CIFAR-10 dataset(mean = (0.4914, 0.4822, 0.4465), standard deviation = (0.2023, 0.1994, 0.2010).

The second issue that we encountered was that the authors do not use any bias terms in their fully connected network. Our initial attempts to reproduce the work included bias terms with our implementation of the ENorm extension for bias terms described by the authors in Section 3.5 of their paper. It is not explicitly stated in the paper to exclude the bias terms. Our attempts to run the E-Norm procedure with a bias term resulted in a very unstable network that produced *NaN* loss for larger numbers of intermediary layers ($p \geq 11$). We report the results we obtained when running these experiments in this work as additional information. We found that the occasional instability while using ENorm we encountered in implementing the fully connected network identically to the paper was much more prominent when a bias term was included. In fact, we were unable to get $p =$ 11, 13, 15, 17, 19 to produce non-*NaN* loss. As a result, we were unable to get any results for these architectures.

The third issue we faced was managing the learning rate. Our original attempt involved separating out a validation set from the training set (80-20 split), keeping a constant learning rate and then computing the test accuracy using the model weights that had the best accuracy on the validation set, regardless of epoch count. We found this strange because they also state how "cross-validation is used to pick an initial learning rate", which suggests some sort of learning rate decay was performed. It turns out that the authors decay the learning rate linearly to zero from the initial value over the course of the 60 epochs. They then use the final weights of the model to compute the accuracy on the test set.

5 CONCLUSION

We found that we able to substantially reproduce the authors' findings. The authors' Equi-Normalization algorithm was implemented and compared to Batch Normalization and a baseline with no normalization. A combination of BN and ENorm was also tested. Fully connected feed forward networks were evaluated on the CIFAR-10 dataset with a range of intermediary layers. It was found that for most values of intermediary layers the best performing normalization scheme was BN + Enorm, followed by ENorm, then BN; all of which outperformed the baseline case. This ordering matches the authors' findings exactly. The only difference in our findings compared to the authors' was that for the network with 19 intermediary layers the BN + ENorm scheme relatively poorly. However, this outlier is easily explained by the variance in our data.

ACKNOWLEDGMENTS

We would like to thank the authors for responding to our questions and providing us with the relevant portions of their code.

REFERENCES

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, and Tara Sainath. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97, November 2012. URL https://www.microsoft.com/en-us/research/publication/deep-neural-networks-for-acoustic-modeling-in-speech-recognition/.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pp. 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Pierre Stock, Benjamin Graham, Rmi Gribonval, and Herv Jgou. Equi-normalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1gEqiC9FX.