Path-BN: Towards Effective Batch Normalization in the Path Space for ReLU Networks Supplementary Materials

Xufang Luo[1] **Qi Meng**[1] **Wei Chen**[1] **Yunhong Wang**[2] **Tie-Yan Liu**[1]

[1]Microsoft Research, Beijing, China
[2]State Key Laboratory of Virtual Reality Technology and System, Beihang University, Beijing, China,

This draft is the supplemenatry materials for the paper "Path-BN: Towards Effective Batch Normalization in the Path Space for ReLU Networks".

1 OTHER CHOICES ON PATH-REPARAMETERIZATION

In this section, we discuss other choices on path-reparameterization and the influence. The path-reparameterization method using path-values showed in the proof of Theorem 1 is not unique. For example, we can obtain another kind of path-reparameterization, via multiplying and dividing the incoming and outgoing weights of O_1^1 by v_{11}^1 in step 4 of Figure.2 in the main paper. This will not influence analyses much in this paper for the following two reasons. First, whatever path-reparameterization method used, each path-reparameterized network can serve as a sufficient representation for ReLU networks in the path space, because the outputs of the network will keep unchanged for any input after path-reparameterization. Second, our studies are based on Theorem 1 in the main paper, which can be generalized to other path-reparameterization methods.

2 PROOF FOR PROPOSITION 2

Proposition 2 *Suppose* $\hat{o}_j^{l-1}(x^i) \neq 0$ *and* $\hat{o}_j^l(x^i) = \hat{o}_j^{l-1}(x^i) + \sum_{k=1,k\neq j}^d \frac{v_{jk}^l}{v_{kk}^1} \cdot \hat{o}_k^{l-1}(x^i)$. *If the scale of the trained parameters* $\{\frac{v_{jk}^l}{v_{kk}^1}\}_{k=1,\cdots,d;k\neq j}$ *is changed by a positive scalar* $c, (c \neq 1)$ *as* $\{c \cdot \frac{v_{jk}^l}{v_{kk}^1}\}_{k=1,\cdots,d;k\neq j}$ *, the value of* $BN(\hat{o}_j^l(x^i))$ *will be changed.*

Proof: If the scale of the trained parameters $\{\frac{v_{jk}^l}{v_{kk}^1}\}_{k=1,\cdots,d;k\neq j}$ is changed by a positive scalar c, we have

$$
\begin{aligned}
&BN\left(\hat{o}_j^{l-1}(x^i) + \sum_{k=1,k\neq j}^d c \cdot \frac{v_{jk}^l}{v_{kk}^1} \cdot \hat{o}_k^{l-1}(x^i)\right) \\
&\neq BN\left(c \cdot \hat{o}_j^{l-1}(x^i) + \sum_{k=1,k\neq j}^d c \cdot \frac{v_{jk}^l}{v_{kk}^1} \cdot \hat{o}_k^{l-1}(x^i)\right) \\
&= BN\left(\hat{o}_j^{l-1}(x^i) + \sum_{k=1,k\neq j}^d \frac{v_{jk}^l}{v_{kk}^1} \cdot \hat{o}_k^{l-1}(x^i)\right).
\end{aligned}
$$

■

3 DETAILS ON EXPERIMENTS

3.1 EXPERIMENTAL SETTING DETAILS

In particular, following the settings in Meng et al. [2019], we apply G-SGD or Path-BN with residual blocks, because there is no positive scaling invariance across residual blocks. In addition, we introduce a coefficient $\lambda(\lambda < 1)$ for $\tilde{z}_j^{l-1}(x^i)$ for experiments of ResNet, to prevent the output exploding during the forward process, and λ for all layers in ResNet is set to be 0.1.

Here we describe pre-processing steps, which is same as He et al. [2016]. For CIFAR-10 and CIFAR-100, we randomly crop the input image by size of 32 with padding size of 4, and normalize every pixel value to $[0, 1]$. Then the random horizontal flipping to the image is applied. For ImageNet, we randomly crop the input image by size of 224, and normalize every pixel value to $[0, 1]$. Then the random horizontal flipping to the image is also applied.

We use 1 NVIDIA Tesla P100 or P40 GPU to run the experiments on CIFAR, and use 4 NVIDIA Tesla P100 or P40 GPUs in one machine to run the experiments on ImageNet. All experiments are averaged over 5 runs with different random seeds, and PyTorch Paszke et al. [2017] is used for implementation.

Dataset	Method	PlainNet		ResNet		
		18	34	18	34	50
CIFAR-10	SGD+$Path$-BN	7.00% ($\pm$0.08)	7.68% ($\pm$0.21)	6.78% ($\pm$0.15)	6.57% ($\pm$0.06)	6.47% ($\pm$0.23)
CIFAR-100	SGD+$Path$-BN	28.57% ($\pm$0.46)	32.98% ($\pm$0.81)	26.71% ($\pm$0.22)	26.80% ($\pm$0.51)	25.68% ($\pm$0.28)

Table 1: Test error rate on CIFAR-10 and CIFAR-100 for SGD+$Path$-BN.

Here we list hyper-parameters used for CIFAR-10 and CIFAR-100. As for the hyper-parameters of methods mentioned in their corresponding papers, i.e., SGD+BN He et al. [2016] and (G-SGD)+BN(wnorm) Meng et al. [2019], we use the same settings as their original ones. Specifically, for SGD+BN, the initial learning rate is set to 0.1, and for G-SGD+BN (wnorm), the initial learning rate is set to 1.0. Besides, we tune hyper-parameters for (G-SGD)+BN and the proposed (G-SGD)+(Path-BN). Specifically, the initial learning rate is searched from $\{0.1, 0.2, 0.5, 1.0\}$. Then, we use the method proposed in Zheng et al. [2018] as the weight decay in the path space, and set the coefficient of the weight decay to 1×10^{-4}, for these two methods. Besides, we use the SGD without momentum in our experiments, because the way to utilize momentum in the path space remains unclear now. Moreover, for all models and algorithms, the mini-batch size is set to be 128 and the training process is conducted for 64k iterations. The learning rates are multiplied by 0.1 after 32k and 48k iterations in all experiments, and the coefficient of weight decay for methods is set to be 0.0001.

Here we list hyper-parameters used for ImageNet. For all experiments here, the mini-batch size is set to be 256 and the hyper-parameter for weight decay is set to be 0.0001 He et al. [2016]. For SGD+BN, the initial learning rate is set to 0.1 He et al. [2016]. For G-SGD+Path-BN, we only tune the initial learning rate from $\{0.1, 0.2, 0.5\}$. The training process is conducted for 90 epochs, and the learning rate is multiplied by 0.1 after 30 and 60 epochs.

As for the stacked CNN used for observations in Section 5.2 in the main paper, we use the following settings: kernel size 3, stride 1, padding 1, and no bias. For CNN, the network width is the the number of channels, which is set to 128 for all hidden layers.

3.2 SUPPLEMENTARY RESULTS

Here, we add other baseline, i.e., SGD+$Path$-BN, in which we use SGD to train the network with $Path$-BN, and in Table 1, we show the test error for SGD+$Path$-BN. We can observe that G-SGD+$Path$-BN outperforms SGD+$Path$-BN, which demonstrate the benefits of optimizing in the path space.

3.3 ANALYSIS OF COMPUTATIONAL COSTS

Different from the conventional BN, Path-BN is to identify the diagonal elements of weight matrices. For convolutional networks, the outputs of hidden nodes in MLP always correspond to the feature maps. Hence, on the one hand, if the sizes of feature maps in different layers are the same, and the stride length is 1, we can then set the diagonal elements in weight matrices to 0 and add skip connections for all layers (except the *1st* layer), which will not bring extra computational cost. On the other hand, if the stride length is larger than 1, we can also set the diagonal elements in weight matrices to 0, and additionally, use a fixed-weight convolutional layer, with only diagonal elements fixed to 1 and others fixed to 0. Thus, with the skip connection or the fixed-weight convolutional layer, the back propagation process can be easily implemented. Therefore, according to the above implementation, Path-BN will not bring much extra computational cost compared with the conventional BN. Statistically, in our experiments, the ratio of run time between G-SGD+Path-BN and SGD+BN, is less than $4:3$, and the extra computational cost is mainly brought by the update of G-SGD instead of Path-BN.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

Qi Meng, Shuxin Zheng, Huishuai Zhang, Wei Chen, Zhi-Ming Ma, and Tie-Yan Liu. G-SGD: Optimizing ReLU neural networks in its positively scale-invariant space. In *ICLR*, 2019.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS-W*, 2017.

Shuxin Zheng, Qi Meng, Huishuai Zhang, Wei Chen, Nenghai Yu, and Tie-Yan Liu. Capacity control of ReLU neural networks by basis-path norm. *arXiv preprint arXiv:1809.07122*, 2018.