OPT2025: 17th Annual Workshop on Optimization for Machine Learning

BatchNorm Layers have an Outsized Effect on Adversarial Robustness

Noam Zeise
Tiffany Vlaar TIFFANY.VLAAR@GLASGOW.AC.UK
University of Glasgow, United Kingdom

Abstract

Training different layers differently may affect resulting adversarial robustness and clean accuracy in adversarial training. We focus on the BatchNorm layers and study their unique role in adversarial training. Through a partial adversarial (pre-)training methodology we investigate how different optimization strategies for the BatchNorm layers affect adversarial robustness, and interplay with other model design choices.

1. Introduction

Neural networks are well-known to be vulnerable to adversarial attacks: subtle, crafted input perturbations that cause models to misclassify [5, 14]. Adversarial training [5, 9] uses adversarial examples during training to strengthen models against such attacks. Projected gradient decent (PGD) has been found effective at generating adversarial examples for training [9]. However, a major drawback to adversarial training is the decreased test accuracy on clean data, with an inherent trade-off between robustness and clean test accuracy [15, 20].

It is well-known that different neural network layers play different roles [19], and this has also been considered within the context of adversarial training. In particular, Bakiskan et al. [1] hypothesize that earlier layers are more crucial in obtaining robustness against adversarial attacks. Nguyen et al. [11] show that different layers affect robust overfitting differently and argue that changing the optimization of later layers can mitigate overfitting. Relatedly, Cianfarani et al. [2] found that "deeper layers overfit during robust training". This prior research suggests that training different neural network layers differently may benefit adversarial training.

Batch normalization (BatchNorm) [7] is a common part of various deep learning architectures [6]. BatchNorm was argued to be a cause of adversarial vulnerability [4], was considered in a specific adversarial fine-tuning setting [16], and removing BatchNorm layers was shown to have potential to enhance adversarial training [17]. Although BatchNorm layers comprise only a very small amount of the trainable parameters of a model (typically less than 0.2%), it was shown that by only training BatchNorm layers non-trivial test accuracy can be obtained [3]. Inspired by this, we perform a thorough investigation into how BatchNorm layers interact with adversarial training.

Our contributions:

- We find that naturally re-training the BatchNorm layers of an adversarially trained model has a big impact: significantly reducing adversarial robustness and increasing clean test accuracy. In contrast, BatchNorm layers can be frozen during adversarial re-training of a naturally trained model without significant impact (Table 1, Table 2).

© N. Zeise & T. Vlaar.

- We show that adversarially re-training only the BatchNorm layers of a naturally trained model can achieve non-trivial adversarial robustness (Table 1, Table 2), and that this occurs both in the fine-tuning and re-initializing setting (Table 6). We demonstrate that adversarially re-training the same number of randomly-selected parameters per channel is not able to achieve similar robustness, while significantly lowering clean accuracy (Table 3).
- When adversarially re-training only the BatchNorm layers of a naturally trained model, we show that the resulting adversarial robustness and clean test accuracy improves with batch size (Table 5) and significantly improves with depth (Table 1, Table 2, Table 4, Figure 1).
- We consider the distributions of the trainable BatchNorm parameters and find a large spike at zero when only adversarially re-training BatchNorm layers (Figure 2). Further, we suggest that early BatchNorm layers may play a more important role (Table 3).

Code is available at: https://github.com/NoamZeise/bn-layers-adv-robustness.

2. Related Work

Recently, various authors have commented on the role that different neural network layers play in adversarial training. Bakiskan et al. [1] found that earlier layers play a more important role in delivering robustness, by partially re-training either earlier or later parts of an (adversarially or naturally trained) network. Cianfarani et al. [2] find that later layers exhibit more overfitting during robust training, whereas early layers converge quickly, and thus comment that development of layer-wise regularization methods can benefit robust training. Similarly, Nguyen et al. [11] find that robust overfitting is mostly connected to the training of later layers and suggest two ways (through learning rate adjustments or adversarial weight perturbation [18]) to regularize the later layers.

BatchNorm is a common part of various deep learning architectures [6]. BatchNorm was originally suggested to reduce internal covariate shift [7], but this has since been disputed [12], leaving the reason for its success as an open question. Frankle et al. [3] found that by only training the BatchNorm layers, one can obtain non-trivial test accuracy, showing the expressive abilities of the BatchNorm affine parameters. Although BatchNorm is thought to be a cause of adversarial vulnerability [4], it was shown in a limited setting that adversarially fine-tuning these layers may lead to non-trivial robustness [16], whereas [17] showed that removing BatchNorm layers could enhance adversarial training. Only perturbing the BatchNorm (or LayerNorm) layers in the adversarial step of sharpness aware minimisation was found to improve both test accuracy and robustness [10].

3. Background

We investigate the effect of BatchNorm (Sec. 3.1) on adversarial robustness. The partial adversarial training methodology (Sec. 3.2) is inspired by [1], who focused on the role of earlier layers.

3.1. Batch Normalization

Batch normalization (BatchNorm) [7] applies the following transformation to input $\mathbf{x}$:

$$\gamma \frac{\mathbf{x} - \mu}{\sigma} + \beta, \tag{1}$$

where γ and β are trainable parameters (also referred to as the affine parameters), and μ and σ^2 are the mean and variance, respectively, computed over the batch dimension.

3.2. Partial Adversarial Training

Following Bakiskan et al. [1], we consider the following 2-step process to study the effect of partial adversarial training on resulting test performance and adversarial robustness:

1. The whole model is trained either adversarially or naturally.
2. Part of the model is frozen. The remaining unfrozen layers are reinitialized and re-trained (adversarially or naturally).

4. Results

We use partial adversarial training (Sec. 3.2) to investigate the role of BatchNorm in adversarial training. We first train a whole model adversarially (All_A) or naturally (All_N), and then freeze part of the network, in our case either all BatchNorm layers (BN) or all non-BatchNorm layers (Rest). Unfrozen layers are re-initialized and re-trained. The same optimizer settings are used for the re-training. The BatchNorm statistics are never frozen as this would significantly impact training. Our baseline is a ResNet-18 trained on CIFAR-10 data. For adversarial training we used a PGD attack [9]. Results are averaged over 3 random seeds. For full experimental details see Appx A.

Impact of BatchNorm on Adversarial Training. We show that only naturally retraining the BatchNorm layers of an adversarially trained network (All_ABN_N) leads to very low adversarial accuracy and higher clean accuracy compared to a fully adversarially trained model (All_A), despite the majority of the model being adversarially trained (Table 1). In contrast, freezing the BatchNorm layers during adversarial re-training of a natural model ($\text{All}_N\text{Rest}_A$) does not have a big effect.

Furthermore, we find that adversarially re-training only the BatchNorm layers of a naturally trained network (All_NBN_A) leads to non-trivial adversarial robustness (Table 1). We show that adversarially re-training the same number of parameters, by selecting two random parameters per channel, is unable to obtain a similar level of clean and adversarial accuracy (Table 3, compare with 2 rand params). This indicates BatchNorm layers have an outsized effect on adversarial robustness.

model	retrain params	test loss	test acc.	adversarial loss	adversarial acc.
All_NBN_A	9600	1.243 $\pm$0.009	0.620 $\pm$0.004	1.860 $\pm$0.012	0.292 $\pm$0.003
All_ABN_N	9600	0.417 $\pm$0.008	0.867 $\pm$0.003	12.232 $\pm$0.252	0.051 $\pm$0.004
$\text{All}_N\text{Rest}_A$	11164362	0.616 $\pm$0.005	0.802 $\pm$0.001	1.781 $\pm$0.026	0.420 $\pm$0.005
$\text{All}_A\text{Rest}_N$	11164362	0.271 $\pm$0.016	0.918 $\pm$0.005	35.566 $\pm$1.249	0.000$\pm$0.000

Table 1: Number of retrained parameters (retrain params), clean/adversarial loss & accuracy (acc.) when re-training all BatchNorm layers (BN) or non-BatchNorm layers (Rest) of an adversarially (All_A) or naturally (All_N) trained ResNet-18 on CIFAR-10. For reference, All_N obtains: test loss 0.251; test acc. 0.926; adversarial loss 40.034; adversarial acc. 0.000. All_A obtains: test loss 0.598; test acc. 0.808 adversarial loss 1.7; adversarial acc. **0.428**.

Role of Depth. We show that the same trends occur for a ResNet-50 model (Table 2). In particular, retraining only the BatchNorm layers adversarially (All_NBN_A) also results in non-trivial adversarial accuracy. However, additionally there is a noticeable improvement in the resulting clean accuracy of

model	retrain params	test loss	test acc.	adversarial loss	adversarial acc.
All_NBN_A	53120	1.108 $\pm$0.013	0.663 $\pm$0.006	1.833 $\pm$0.005	0.304 $\pm$0.002
All_ABN_N	53120	0.381 $\pm$0.008	0.881 $\pm$0.002	22.283 $\pm$0.187	0.002 $\pm$0.000
$\text{All}_N\text{Rest}_A$	23467722	0.562 $\pm$0.013	0.817 $\pm$0.006	1.885 $\pm$0.006	0.405 $\pm$0.005
$\text{All}_A\text{Rest}_N$	23467722	0.253 $\pm$0.010	0.925 $\pm$0.004	35.330 $\pm$0.889	0.000 $\pm$0.000

Table 2: Re-training all BatchNorm layers (BN) or non-BatchNorm layers (Rest) of an adversarially (All_A) or naturally (All_N) trained ResNet-50 on CIFAR-10. For comparison, All_N obtains: test loss 0.270; test acc. 0.914; adversarial loss 43.727; adversarial acc. 0.000. All_A obtains: test loss 0.637; test acc. 0.799; adversarial loss 1.605; adversarial acc. **0.439**.

All_NBN_A compared to a fully adversarially trained (All_A) network (Table 2) relative to our ResNet-18 findings (Table 1). The clean accuracy and adversarial robustness of All_NBN_A further improves for a ResNet-152 model (Table 4). Overall, we find that the gap between All_A and All_NBN_A decreases with model depth (Fig. 1). This echoes Frankle et al. [3]'s findings (albeit in a different context) that depth can further enhance generalization when only training the BatchNorm layers.

Role of Early Layers. We consider freezing all layers of a naturally trained network, including either the first or the last BatchNorm layer, while adversarially re-training only the remaining BatchNorm layers. We find that the first BatchNorm layer has a bigger impact on adversarial robustness despite having a smaller number of parameters (Table 3, Appendix). This would be consistent with Bakiskan et al. [1]'s findings on the importance of the earlier layers in adversarial training.

Role of Batch Size. BatchNorm is known to be sensitive to the choice of mini-batch size [8, 13]. We find that there is a small but noticeable trend where the clean and adversarial accuracies both decrease with the batch size when adversarially retraining only the BatchNorm layers (Table 5).

Adversarial Fine-tuning. We find that our results are sustained whether the BN layers are re-initialized or not before adversarial re-training (Table 6). Surprisingly, we find that both adversarial robustness and clean accuracy is slightly decreased without re-initializing.

Values of BatchNorm Parameters. When the BatchNorm layers of a naturally trained network are retrained adversarially, a significant spike occurs at 0 for γ and β (Fig. 2). This matches observations of Frankle et al. [3] when only the BatchNorm layers of a network are trained. This occurs both in the re-initialization and fine-tuning setting. [3] argue that observing these kind of values for the BatchNorm parameters indicates the network has learned to impose per-feature sparsity, which allows models to still achieve non-trivial performance when only the BatchNorm layers are trained.

5. Discussion

We find that BatchNorm layers have an outsized effect on adversarial robustness, despite representing only a small part of the total number of parameters, and that their importance seems to increase with model depth (Fig. 1). This suggests further investigation into BatchNorm may be valuable to better understand and enhance adversarial training. In particular, since robust overfitting was shown to be layer-sensitive [2, 11], it would be insightful to study this in the specific context of BatchNorm. Further, although the vast majority of related work consider ResNets with BatchNorm, it would be valuable to investigate if our findings generalize to LayerNorm and different architectures.

References

[1] C. Bakiskan, M. Cekic, and U. Madhow. Early Layers Are More Important For Adversarial Robustness. *New Frontiers in Adversarial Machine Learning*, 2022.

[2] C. Cianfarani, A. N. Bhagoji, V. Sehwag, B.Y. Zhao, P. Mittal, and H. Zheng. Understanding robust learning through the lens of representation similarities. *NeurIPS*, 2022.

[3] J. Frankle, D.J. Schwab, and A.S. Morcos. Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs. *ICLR*, 2021.

[4] A. Galloway, A. Golubeva, T. Tanay, M. Moussa, and G.W. Taylor. Batch normalization is a cause of adversarial vulnerability. *ICML Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019.

[5] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML*, 2015.

[8] X. Lian and J. Liu. Revisit Batch Normalization: New understanding and refinement via composition optimization. *AISTATS*, 2019.

[9] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ICLR*, 2018.

[10] M. Müller, T. Vlaar, D. Rolnick, and M. Hein. Normalization Layers Are All That Sharpness-Aware Minimization Needs. *NeurIPS*, 2023.

[11] D. Nguyen, C. Yu, V. Nandakumar, Y. Lee, and T. Liu. On Intriguing Layer-Wise Properties of Robust Overfitting in Adversarial Training. *Transactions on Machine Learning Research*, 2024.

[12] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? *NeurIPS*, 2018.

[13] C. Summers and M.J. Dinneen. Four things everyone should know to improve batch normalization. *ICLR*, 2020.

[14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *ICLR*, 2014.

[15] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness may be at odds with accuracy. *ICLR*, 2019.

[16] N.P. Walter, D. Stutz, and B. Schiele. On fragile features and batch normalization in adversarial training. *arXiv:2204.12393 preprint*, 2022.

[17] H. Wang, A. Zhang, S. Zheng, X. Shi, M. Li, and Z. Wang. Removing batch normalization boosts adversarial training. *ICML*, 2022.

[18] D. Wu, S. Xia, and Y. Wang. Adversarial weight perturbation helps robust generalization. *NeurIPS*, 2020.

[19] C. Zhang, S. Bengio, and Y. Singer. Are all layers created equal? *JMLR*, 2022.

[20] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically principled trade-off between robustness and accuracy. *ICML*, 2019.

Appendix A. Experimental Details

- Unless otherwise specified we use a ResNet18 architecture trained on CIFAR 10 for 80 epochs with a batch size of 128. We use an SGD optimizer with momentum 0.9 and a weight decay of 0.0002. We use a learning rate of 0.1 which decays to 0.01 after 75 epochs.
- For adversarial training we use the same SGD optimizer and an ℓ_∞ bounded PGD attack with an epsilon of $\frac{8}{255}$ and a step size of $\frac{2}{255}$ over 10 steps.
- For adversarial testing we used an l_∞ PGD attack with a step size of $\frac{1}{255}$ and with 100 steps. All other attack settings are the same as during adversarial training.

Appendix B. Impact of BatchNorm on Adversarial Training

adv. retrained	retrain params	test loss	test acc.	adversarial loss	adversarial acc.
BN (baseline)	9600	1.243 $\pm$0.009	0.620 $\pm$0.004	1.860 $\pm$0.012	**0.292** $\pm$0.003
2 rand params	9600	2.171 $\pm$0.035	0.204 $\pm$0.012	2.229 $\pm$0.019	0.133 $\pm$0.022
BN not first	9472	1.297 $\pm$0.005	0.597 $\pm$0.007	1.913 $\pm$0.003	0.273 $\pm$0.003
BN not last	8576	1.273 $\pm$0.017	0.592 $\pm$0.005	1.887 $\pm$0.011	0.283 $\pm$0.005

Table 3: Number of retrained parameters (retrain params), clean/adversarial loss & accuracy (acc.) when adversarially re-training: all BatchNorm layers (BN); two random parameters per channel (2 rand params); all BatchNorm layers except the first BN layer (BN not first); all BatchNorm layers except the last BN layer (BN not last) of a naturally trained ResNet-18.

Appendix C. Role of Depth

In Table 4 we show that adversarially retraining only the BatchNorm layers of a naturally trained ResNet-152 model (All_NBN_A) results in non-trivial adversarial accuracy. The clean accuracy and adversarial robustness are improved relative to a fully adversarially trained model (All_A) compared to the ResNet-50 (Table 2) and ResNet-18 (Table 1) setting. In Figure 1 we comprehensively show the role of model depth. Use of deeper architectures significantly increases clean accuracy and adversarial robustness when only adversarially re-training the BatchNorm layers (All_NBN_A) relative to a fully adversarially trained model (All_A). In contrast, the adversarial accuracy decreases with model depth when freezing the BatchNorm layers during adversarial re-training ($\text{All}_N\text{Rest}_A$). This shows the increased importance of BatchNorm layers on adversarial robustness with model depth.

model	retrain params	test loss	test acc.	adversarial loss	adversarial acc.
All_NBN_A	151424	0.988 ± 0.023	0.703 ± 0.009	1.711 ± 0.017	0.356 ± 0.007
All_ABN_N	151424	0.418 ± 0.009	0.887 ± 0.002	20.733 ± 0.461	0.008 ± 0.002
$\text{All}_N\text{Rest}_A$	58005194	0.598 ± 0.015	0.817 ± 0.004	2.913 ± 0.017	0.374 ± 0.002
$\text{All}_A\text{Rest}_N$	58005194	0.246 ± 0.016	0.930 ± 0.004	36.469 ± 1.369	0.000 ± 0.000

Table 4: Re-training all BatchNorm layers (BN) or non-BatchNorm layers (Rest) of an adversarially (All_A) or naturally (All_N) trained ResNet-152 on CIFAR-10. For comparison, All_N obtains: test loss 0.280; test acc. 0.911; adversarial loss 40.872; adversarial acc. 0.000. All_A obtains: test loss 0.593; test acc. 0.815; adversarial loss 1.646; adversarial acc. **0.439**.

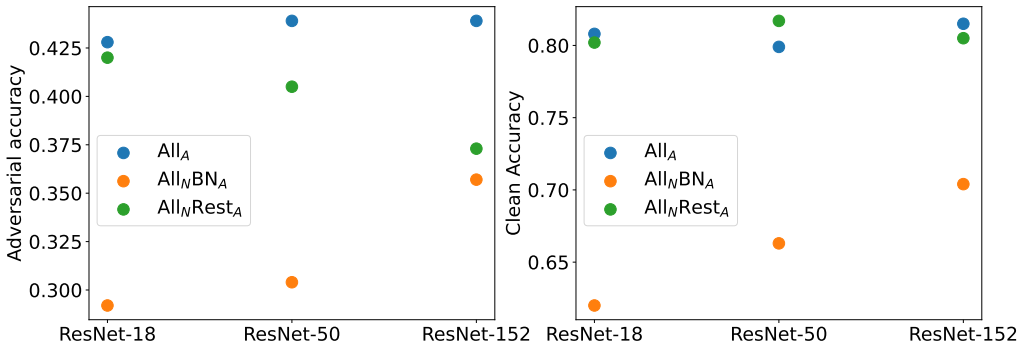


Figure 1: Adversarially re-training all BatchNorm layers (All_NBN_A,orange) or all non-BatchNorm layers ($\text{All}_N\text{Rest}_A$,green) of naturally trained ResNet architectures with different model depth. We also show the performance of a fully adversarially model (All_A,blue).

Appendix D. Role of Batch Size

Model: All_NBN_A	test loss	test acc.	adversarial loss	adversarial acc.
batch size 128 (baseline)	1.243 ± 0.009	0.620 ± 0.004	1.860 ± 0.012	0.292 ± 0.003
batch size 64	1.266 ± 0.004	0.607 ± 0.006	1.885 ± 0.013	0.283 ± 0.005
batch size 32	1.313 ± 0.006	0.583 ± 0.001	1.930 ± 0.012	0.262 ± 0.005

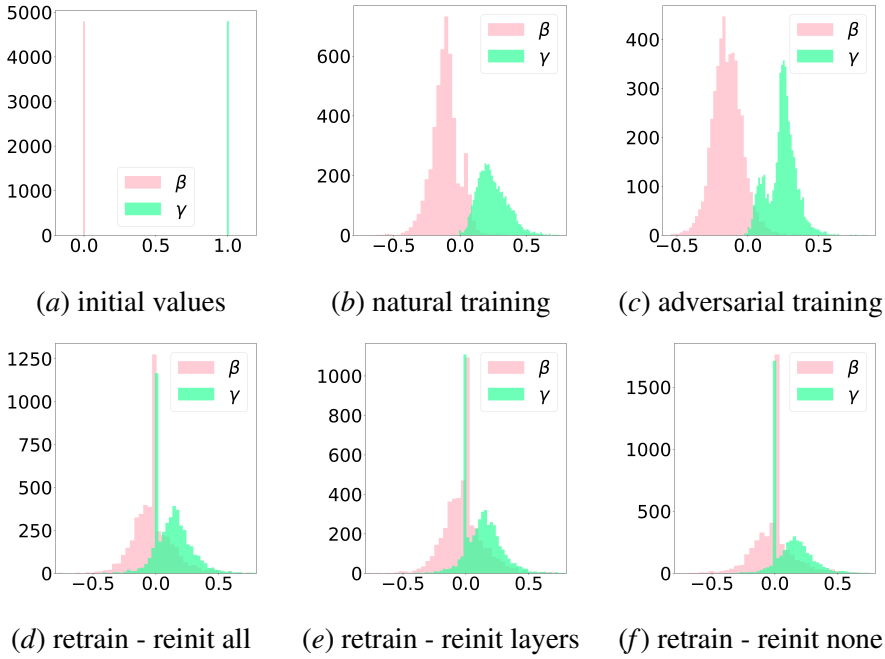
Table 5: Adversarially re-training all BatchNorm layers of a naturally trained (All_NBN_A) ResNet-18 on CIFAR-10 for different choices of batch size.

Appendix E. Adversarial Fine-tuning

Model: All_NBN_A	test loss	test accuracy	adversarial loss	adversarial accuracy
reinit (baseline)	1.243 $\pm$0.009	0.620 $\pm$0.004	1.860 $\pm$0.012	0.292 $\pm$0.003
reinit stats only	1.249 $\pm$0.004	0.615 $\pm$0.004	1.864 $\pm$0.004	0.293 $\pm$0.002
no reinit	1.294 $\pm$0.005	0.599 $\pm$0.004	1.904 $\pm$0.012	0.280 $\pm$0.004

Table 6: Adversarially re-training all BatchNorm layers of a naturally trained (All_NBN_A) ResNet-18 on CIFAR-10. We either re-initialize the BatchNorm parameters before re-training (reinit, our baseline), re-initialize only the BatchNorm statistics (reinit stats only), or use the adversarial fine-tuning setting (no reinit).

Appendix F. BatchNorm Parameters



(*a*) initial values (*b*) natural training (*c*) adversarial training

(*d*) retrain - reinit all (*e*) retrain - reinit layers (*f*) retrain - reinit none

Figure 2: Distributions of the trainable BatchNorm parameters (β and γ, see Eq. 1) for a ResNet-18 architecture under different training methods. For (d), (e), and (f) we start with a natural network and retrain only the batch norm layers adversarially. We observe that retrained batch norm layers have large spikes around zero for both weight and bias, no matter whether parameters or statistics are reinitialised before retraining.

Appendix G. Retraining Single Layers

model	test loss	test accuracy	adversarial loss	adversarial accuracy
conv1	2.419	0.501	23.135	0.002
bn1	1.417	0.671	16.128	0.013
layer1.0.conv1	1.319	0.698	14.143	0.029
layer1.0.bn1	0.698	0.815	26.202	0.0
layer1.0.conv2	2.848	0.209	39.164	0.0
layer1.0.bn2	2.669	0.45	18.59	0.001
layer1.1.conv1	0.801	0.796	21.774	0.004
layer1.1.bn1	0.477	0.866	36.321	0.0
layer1.1.conv2	0.722	0.816	26.459	0.0
layer1.1.bn2	0.615	0.834	33.548	0.0
layer2.0.conv1	2.592	0.216	74.166	0.0
layer2.0.bn1	0.528	0.855	38.871	0.0
layer2.0.conv2	3.058	0.119	22.013	0.001
layer2.0.bn2	2.991	0.1	8.998	0.048
layer2.0.shortcut.0	0.443	0.877	35.737	0.0
layer2.0.shortcut.1	1.103	0.716	42.314	0.0
layer2.1.conv1	1.667	0.626	252.353	0.0
layer2.1.bn1	0.343	0.897	40.779	0.0
layer2.1.conv2	2.233	0.433	100.29	0.0
layer2.1.bn2	3.025	0.208	44.652	0.0
layer3.0.conv1	2.055	0.271	63.394	0.0
layer3.0.bn1	2.657	0.112	11.889	0.0
layer3.0.conv2	2.032	0.303	27.199	0.0
layer3.0.bn2	3.25	0.197	74.731	0.0
layer3.0.shortcut.0	0.329	0.907	50.911	0.0
layer3.0.shortcut.1	3.464	0.204	44.522	0.0
layer3.1.conv1	1.667	0.443	30.451	0.0
layer3.1.bn1	0.518	0.855	54.123	0.0
layer3.1.conv2	1.754	0.411	17.89	0.0
layer3.1.bn2	2.501	0.189	55.927	0.0
layer4.0.conv1	2.077	0.371	39.624	0.0
layer4.0.bn1	0.941	0.75	37.804	0.0
layer4.0.conv2	1.869	0.531	16.906	0.0
layer4.0.bn2	2.446	0.154	5.579	0.0
layer4.0.shortcut.0	0.391	0.886	55.133	0.0
layer4.0.shortcut.1	2.504	0.143	25.684	0.0
layer4.1.conv1	2.303	0.134	3.032	0.0
layer4.1.bn1	1.912	0.324	4.426	0.0
layer4.1.conv2	1.94	0.862	3.265	0.0
layer4.1.bn2	2.268	0.149	2.873	0.001
linear	2.295	0.145	2.358	0.0

Table 7: Retraining only one layer of naturally (All_N) trained ResNet-18 on CIFAR-10.