Spectrum Extraction and Clipping for Implicitly Linear Layers

Ali Ebrahimpour-Boroojeny
UIUC
ae20@illinois.edu

Matus Telgarsky
NYU
mjt10041@nyu.edu

Hari Sundaram
UIUC
hs1@illinois.edu

Abstract

We show the effectiveness of automatic differentiation in efficiently and correctly computing and controlling the spectrum of *implicitly* linear operators, a rich family of layer types including all standard convolutional and dense layers. We provide the first clipping method which is correct for general convolution layers, and illuminate the representational limitation that caused correctness issues in prior work. By comparing the accuracy and performance of our methods to existing methods, using various experiments, show they lead to better generalization and adversarial robustness of the models. In addition to these advantages over the state-of-the-art methods, we show they are much faster than the alternatives.

1 Introduction

Implicitly linear layers are key components of deep learning models and include any layer whose output can be written as an affine function of their input. This affine function might be trivial, such as a dense layer, or non-trivial, such as convolutional layers. These layers inherit appealing properties of linear transformations; not only are they flexible and easy to train, but also they have the same Jacobian as the transformation that the layer represents and a Lipschitz constant equal to its largest singular value. Therefore controlling the largest singular value of these layers, which is the same as the largest singular value of their Jacobians, not only contributes to the generalization of the model (Bartlett et al., 2017), but makes the model more robust to adversarial perturbations (Szegedy et al., 2013; Weng et al., 2018), and prevents the gradients from exploding or vanishing during backpropagation. Although efficient algorithms have been introduced to bound the spectral norm of dense layers (Miyato et al., 2018), computing and bounding them efficiently and correctly has been a challenge for the general family of *implicitly* linear layers, such as convolutional layers.

Convolutional layers are a major class of implicitly linear layers that are used in many models in various domains. There has been an ongoing effort to design methods for controlling the spectrum of these layers in either trained models or during the training (Miyato et al., 2018; Cisse et al., 2017; Sedghi et al., 2018; Farnia et al., 2018; Virmaux & Scaman, 2018; Trockman & Kolter, 2021; Singla et al., 2021; Gouk et al., 2021; Xu et al., 2022; Senderovich et al., 2022; Delattre et al., 2023); however, neither of these methods work correctly for all standard convolutional layers (see section D for a discussion of prior works). Moreover, they are either computationally extensive or rely on heuristics. In this work, we study both problems of extracting the spectrum and controlling them for the general family of implicitly linear layers. We give efficient algorithms for both tasks that scale to large models without incurring noticeable computational barriers. We also unveil some previously neglected limitations of convolutional layers in representing arbitrary spectrums. Up to this point, researchers have assumed they can modify the spectrum of convolutional layers in an arbitrary way. Although the experiments center around dense layers and convolutional layers, as we will discuss, our algorithms are general and do not assume properties that are specific to these layers. The only assumption is that their transformation can be represented as an affine function $f(x) = M_W x + b$, in

Mathematics of Modern Machine Learning Workshop at NeurIPS 2023.

which W represents the parameters of the layer, but the transformation matrix M_W is not necessarily known. In details, the contributions of this work are as follows:

1. An implicit way of **computing the spectral norm via shifted subspace iteration** on any implicitly linear layer or their concatenation **using automatic differentiation** and showing how it can be efficiently used during training Section 2.1.
2. An algorithm for **exact clipping of the spectral norms** to arbitrary values for implicitly linear layers in Section 2.2. We also design a pipeline to perform clipping of the spectral norms in large models **during the training with negligible computational overhead** in Appendix B.1.
3. Theoretical proof of the **limitation of convolutional layers** with circular padding in representing arbitrary spectrums in Section 2.3.
4. Showing the effect of our clipping pipeline in **boosting the test accuracy and increasing the robustness** in **table 1**.

2 Methods

This section is organized as follows. We introduce our algorithm, PowerQR, for extracting top-k singular values and vectors of any implicitly linear layer or their concatenations in Section 2.1. In Section 2.2, we introduce our accurate and efficient algorithm, FastClip, for clipping the spectral norm of implicitly linear layers. In Section 2.3, we establish the inability of convolutions to attain any arbitrary spectrum. The reader can find the proofs in Appendix A. Next, we introduce notation.

Notation. An implicitly linear layer can be written as an affine function $f(x) = M_W x + b$, where $x \in \mathbb{R}^n$, $M_W \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. M_W is not necessarily the explicit form of the parameters of the layer, in which case the explicit form is shown with W (e.g., the kernel of convolutional layers); otherwise, M_W is the same as W. The spectral norm of M_W, which is the largest singular value of M_W, is shown with $\|M_W\|_2$. We use $\sigma_i(W)$ to represent the i-th largest singular value of matrix M_W. We might refer to the largest singular value as either $\|W\|_2$ or $\|M_W\|_2$. We use $\omega = \exp(2\pi i/n)$ (the basic n-th roots of unity), where n is the rank of the linear transformation. $\Re(.)$ returns the real part of its input, and we also define the symmetric matrix A_W as $M_W^\top M_W$.

2.1 Spectrum Extraction

We introduce our *PowerQR* algorithm (Algorithm 1), which performs an implicit version of shifted subspace iteration algorithm (see chapter 5 of Saad (2011) for more details) using automatic differentiation, to compute the singular values of M. Regarding the complexity of Algorithm 1, other than the $O(n^2k)$ complexity of QR decomposition in line 6, it has an additional cost of computing the gradients of the layer at line 4 for a batch of k data points, which is dominated by the former cost.

Lemma 2.1. *Let $f(x) = Mx + b$, and $g(x) = f(x) - f(0)$. Then Algorithm 1 correctly performs the shifted subspace iteration algorithm with μ as the shift value.*

Algorithm 1 PowerQR ($f, X, N, \mu = 1$)

1: **Input:** Affine function f, Initial matrix $X \in \mathbb{R}^{n \times k}$, Number of iterations N, Shift value μ
2: **Output:** Top k singular values and corresponding right singular vectors
3: **for** $i = 1$ **to** N **do**
4: $\quad X' \leftarrow \nabla_X \frac{1}{2}\|f(X) - f(0)\|^2$ // $\nabla_x \frac{1}{2}\|f(x) - f(0)\|^2 = M^\top Mx$
5: $\quad X \leftarrow \mu X + X'$
6: $\quad (Q, R) \leftarrow \mathrm{QR}(X)$ // QR *decomposition of* X
7: $\quad X \leftarrow Q$
8: **end for**
9: $S \leftarrow \sqrt{\mathrm{diag}(R - \mu I)}$ // *Singular Values*
10: $V \leftarrow X$ // *Right singular vectors*
11: **Return** S, V

Since SGD makes small updates to W in each training step, by reusing the matrix V (top-k right singular vectors) computed for W_{i-1} as initial X for computing the spectrum of W_i, we can get much faster convergence. The shift parameter μ helps in faster convergence as well, especially if it is chosen to be close to the largest singular value, which is the case when clipping the singular values because the largest one gets closer to the target clipping value as the training proceeds. Our experiments show that by using this technique, it is enough to perform only one iteration of PowerQR per SGD step to converge to the spectrum of W after a few steps. Using a warm start helps to correctly follow the top-k singular values during the training. The prior work on estimating and clipping the singular values has exploited these small SGD updates for the parameters in a similar manner (Farnia et al., 2018; Gouk et al., 2021; Senderovich et al., 2022).

2.2 Clipping the Spectral Norm

Algorithm 2 shows our stand-alone clipping method. The outer `while` loop clips the singular values of the linear operator one by one. After clipping each singular value, the PowerQR method in line `10` computes the new largest singular value and vector, and the next iteration of the loop performs the clipping on them. The clipping of a singular value is done by the `for` loop. Considering that in the `for` loop $M := M_W = M_{W'} = \sum_{i=1}^{n} u_i \sigma_i v_i^\top$, since v_is are orthogonal and $v_i^\top v_i = 1$, for line `6` we have:

$$W'_\delta = \nabla_{W'} \frac{1}{2} \|f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\|^2 = \left(f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\right) \nabla_{W'} f_{W'}(v_1)$$
$$= \left(Mv_1 + b - c\sigma_1^{-1} M v_1 - b\right) v_1^\top = (u_1\sigma_1 - u_1 c)\, v_1^\top = u_1\sigma_1 v_1^\top - u_1 c v_1^\top,$$

where $\nabla_{W'} f_{W'}(v_1) = v_1^\top$ because it is as if we are computing the gradient of the linear operator with respect to its transformation matrix. Therefore in line `7` if $\lambda = 1$ we compute the new transformation matrix as $\sum_{i=1}^{n} u_i\sigma_i v_i^\top - u_1\sigma_1 v_1^\top + u_1 c v_1^\top = u_1 c v_1^\top + \sum_{i=2}^{n} u_i\sigma_i v_i^\top$. Notice that W'_δ is in the same format as the parameters of the linear operator (e.g., kernel in convolutional layers).

If we set $\lambda = 1$, the largest singular value will be clipped to the desired value c, without requiring the `for` loop. That is indeed the case for linear layers. The reason that we let λ be a parameter and use the `for` loop is that for convolutional layers, we noticed that a slightly lower value of λ is required for the algorithm to work stably (we used 0.7 in all the experiments). The `for` loop allows this convergence to singular value c. By intertwining PowerQR and this clipping algorithm we derive our *FastClip* algorithm, which is explained in Appendix B.1.

Algorithm 2 Clip ($f_W, v_1, \sigma_1, c, N, P, \lambda = 1.$)

1: **Input:** Affine function $f_W = M_W x + b$, Clip value c, Number of iterations N, Learning rate λ, Number of iterations of PowerQR P
2: **Output:** Affine function $f_{W'}$ with singular values clipped to c
3: $\sigma_1, v_1 \leftarrow \text{PowerQR}(f_W, v, P)$ (v: Random vector if v_1 is empty)
4: **while** $\sigma_1 > c$ **do**
5: $\quad W' \leftarrow W$ // $W' = \sum_{i=1}^{n} u_i\sigma_i v_i^\top$
6: $\quad$ **for** $i = 1$ **to** N **do**
7: $\quad\quad W'_\delta \leftarrow \nabla_{W'} \frac{1}{2} \|f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\|^2$ // $W'_\delta = u_1(\sigma_1 - c)v_1^\top$
8: $\quad\quad W' \leftarrow W' - \lambda W'_\delta$
9: $\quad$ **end for**
10: $\quad W \leftarrow W'$
11: $\quad \sigma_1, v_1 \leftarrow \text{PowerQR}(f_W, v, P)$ (v: Random vector) // *Updated σ_1 and v_1*
12: **end while**
13: **Return** $f_{W'}, \sigma_1, v_1$

2.3 Limitations of convolutional layers

In this section, we shed light on the formerly overlooked limitation of convolutional layers to represent any arbitrary spectrum. In some prior work, the proposed method for clipping computes the whole spectrum, clips the spectral norm, and then tries to form a new convolutional layer with the new spectrum (Sedghi et al., 2018; Senderovich et al., 2022). We also introduce a new simple optimization method that uses our PowerQR algorithm and adheres to this procedure in Appendix B.2.

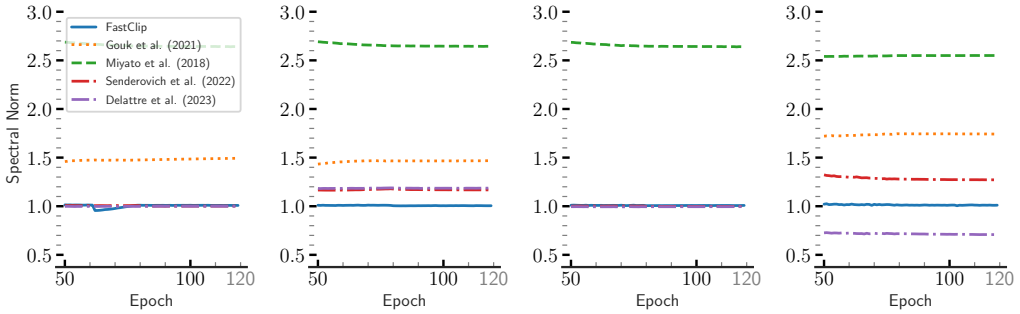


Figure 1: Comparison of the clipping methods for different types of convolutional layers in a simple network with only one convolutional layer and one linear layer, where **the target value is 1**. Our method is the only one that clips this layer correctly for all different settings.

In section A.2, we start with a simple example that shows an issue with this procedure, and then give the proof for the following theorem that shows a more general and fundamental limitation for a family of convolutional layers.

Theorem 2.2. *Any convolutional layer with circular padding with either* 1 *input or output channels has at most two simple (without duplicate) singular values.*

3 Experiments

We perform various experiments to show the effectiveness of our algorithms and compare them to the state-of-the-art methods to show their advantages. Since the method introduced by Senderovich et al. (2022) extends the method of Sedghi et al. (2018) and (Farnia et al., 2018) use the same method as (Gouk et al., 2021), the methods we use in our comparisons are the methods introduced by Miyato et al. (2018); Gouk et al. (2021); Senderovich et al. (2022); Delattre et al. (2023). For all these methods we used the settings recommended by the authors in all experiments. Implementations of our methods and experiments can be found in the supplementary material. All the experiments were performed on a single node with an NVIDIA A40 GPU.

As presented in Algorithm 3, for all our experiments, the values N and P in Algorithm 2 are set to 1 and 10, respectively. We also ignore the condition of the while loop from the latter algorithm when using our FastClip algorithm during training, and instead run it for only 5 iterations. This means that we only perform 5 iterations of lines 7, 8, and 11 (the update for the new values of the largest singular value and vector). Therefore, the complexity of our clipping method is dominated by the complexity of our PowerQR method, which as mentioned in Section 2.1, is $O(n^2)$ (where n is the dimensions of the input tensor). We also compared how fast and accurate our method is on ResNet-18 model compared to the other methods. As Figure 3 shows, our method is the most precise clipping method while being much faster than the relatively accurate alternatives.

To further compare the precision of our method to prior works, we used a simple model with one convolutional layer and one dense layer and used each of the clipping methods on the convolutional layer while the model is being trained on MNIST and the target clipping value is 1. We computed the true spectral norm after each epoch. Figure 3 shows the results of this experiment for 4 convolutional layers with different settings (e.g., kernel size and padding type). This figure shows our method is the only one that correctly clips various convolutional layers.

Since the Lipschitz constant of a network may be upper-bounded by multiplying together the spectral norms of its constituent dense and convolutional layers, it is intuitive that regularizing per-layer spectral norms improves model generalization (Bartlett et al., 2017), and also adversarial robustness (Szegedy et al., 2013). Therefore, we tested all the clipping models by using them during the training and computing the accuracy of the corresponding models on the test set and adversarial examples generated by two common adversarial attacks, Projected Gradient Descent (PGD) (Madry et al., 2017) and Carlini & Wanger Attack (CW) (Carlini & Wagner, 2017).

Table 1: The accuracy on the test set and adversarial examples generated with PGD-(50) and CW for ResNet-18 trained on MNIST and CIFAR-10, for the models with their convolutional and dense layers clipped to 1 (`With BN`) and clipped models with their batch norm layers removed (`No BN`). The accuracy of the original model on test set, PGD-generated examples, and CW-generated examples for MNIST are $\mathbf{99.37 \pm 0.02}$, $\mathbf{15.30 \pm 9.11}$, and $\mathbf{77.46 \pm 5.89}$, respectively. For CIFAR-10, these values are $\mathbf{94.77 \pm 0.19}$, $\mathbf{22.15 \pm 0.54}$, and $\mathbf{13.85 \pm 0.74}$.

METHOD	MNIST		CIFAR-10	
	WITH BN	NO BN	WITH BN	NO BN
	ACCURACY ON THE TEST SET			
MIYATO ET AL. (2018)	99.40 ± 0.06	98.00 ± 0.22	94.82 ± 0.11	88.83 ± 1.41
GOUK ET AL. (2021)	99.25 ± 0.04	21.94 ± 6.01	89.98 ± 0.38	19.80 ± 5.55
SENDEROVICH ET AL. (2022)	99.40 ± 0.03	62.63 ± 24.01	94.19 ± 0.13	68.29 ± 10.63
DELATTRE ET AL. (2023)	99.29 ± 0.05	97.27 ± 0.03	93.17 ± 0.13	39.35 ± 9.84
FASTCLIP (ALGORITHM 3)	$\mathbf{99.41 \pm 0.04}$	$\mathbf{99.31 \pm 0.02}$	$\mathbf{95.28 \pm 0.07}$	$\mathbf{92.08 \pm 0.28}$
	ACCURACY ON SAMPLES FROM PGD ATTACK			
MIYATO ET AL. (2018)	21.77 ± 12.98	32.67 ± 14.08	23.48 ± 0.11	35.18 ± 7.72
GOUK ET AL. (2021)	2.40 ± 2.94	8.41 ± 3.03	16.13 ± 1.28	14.66 ± 3.99
SENDEROVICH ET AL. (2022)	30.99 ± 9.28	15.97 ± 4.84	21.74 ± 0.72	39.84 ± 7.87
DELATTRE ET AL. (2023)	30.87 ± 4.77	71.75 ± 1.49	21.08 ± 0.84	16.22 ± 3.17
FASTCLIP (ALGORITHM 3)	$\mathbf{47.90 \pm 5.49}$	$\mathbf{78.50 \pm 2.85}$	$\mathbf{24.48 \pm 0.32}$	$\mathbf{41.37 \pm 0.95}$
	ACCURACY ON SAMPLES FROM CW ATTACK			
MIYATO ET AL. (2018)	86.25 ± 2.18	73.56 ± 10.38	16.68 ± 0.95	48.48 ± 6.40
GOUK ET AL. (2021)	66.59 ± 21.91	21.94 ± 6.01	18.79 ± 2.99	12.63 ± 4.33
SENDEROVICH ET AL. (2022)	87.72 ± 2.75	58.71 ± 20.67	20.53 ± 0.77	43.82 ± 9.57
DELATTRE ET AL. (2023)	83.97 ± 1.79	$\mathbf{96.93 \pm 0.06}$	24.05 ± 1.72	11.92 ± 5.34
FASTCLIP (ALGORITHM 3)	$\mathbf{90.21 \pm 1.80}$	95.35 ± 1.06	$\mathbf{24.31 \pm 0.96}$	$\mathbf{56.28 \pm 0.96}$

As Table 1 shows, our model leads to the best improvement in test accuracy while making the models more robust to adversarial attacks. The reason for the lack of the expected boost in the robustness of the models when clipping their spectral norms is shown in Figure 4. This figure shows that although the models are being clipped, the concatenation of some convolutional layers with batch normalization layers form linear operators with large spectral norms. Therefore, we also trained a version of the model with all the batch norm layers removed. As Table 1 shows, this leads to a huge improvement in the robustness of the clipped models; however, the clipped models achieve worse accuracy on the test set.

4 Conclusions

We introduced efficient and accurate algorithms for extracting and clipping the spectrum of implicitly linear layers. We showed they are fast and more accurate than prior works, and our experiments confirmed they are more effective than existing methods in improving the generalization and robustness of large models. Also, our algorithms are unique in that they can be applied to not only convolutional and dense layers, but also the concatenation of these layers with other implicitly linear ones, such as batch normalization. This capability has to be further explored in future works.

References

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Revisiting batch normalization for improving corruption robustness. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 494–503, 2021.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. Ieee, 2017.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pp. 854–863. PMLR, 2017.

Blaise Delattre, Quentin Barthélemy, Alexandre Araujo, and Alexandre Allauzen. Efficient bound of lipschitz constant for convolutional layers by gram iteration. *arXiv preprint arXiv:2305.16173*, 2023.

Farzan Farnia, Jesse M Zhang, and David Tse. Generalizable adversarial training via spectral normalization. *arXiv preprint arXiv:1811.07457*, 2018.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

Anil K Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989.

Stamatios Lefkimmiatis, John Paul Ward, and Michael Unser. Hessian schatten-norm regularization for linear inverse problems. *IEEE transactions on image processing*, 22(5):1873–1888, 2013.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

Alexandra Senderovich, Ekaterina Bulatova, Anton Obukhov, and Maxim Rakhuba. Towards practical control of singular values of convolutional layers. *Advances in Neural Information Processing Systems*, 35:10918–10930, 2022.

Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learning*, pp. 9756–9766. PMLR, 2021.

Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on cifar-10 and cifar-100. *arXiv preprint arXiv:2108.04062*, 2021.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. *arXiv preprint arXiv:2104.07167*, 2021.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. *arXiv preprint arXiv:1906.03787*, 2019.

Xiaojun Xu, Linyi Li, and Bo Li. Lot: Layer-wise orthogonal training on improving l2 certified robustness. *Advances in Neural Information Processing Systems*, 35:18904–18915, 2022.

Tan Yu, Jun Li, Yunfeng Cai, and Ping Li. Constructing orthogonal convolutions in an explicit manner. In *International Conference on Learning Representations*, 2021.

A omitted proofs

A.1 proofs of section 2.1

We start by proving lemma 2.1:

Proof. Notice that $\nabla_x \frac{1}{2}\|g(x)\|^2 = m^t m x = ax$. Let define $a' = a + \mu i$. Putting steps 4 and 5 of the algorithm together, we have $(q, r) = \mathrm{qr}(ax + \mu x) = \mathrm{qr}(a'x)$. Therefore, the algorithm above can be seen as the subspace iteration for matrix a'. Therefore, the diagonal values of r will converge to the eigenvalues of a', and columns of q converge to the corresponding eigenvectors. If λ is an eigenvalue of $a' = a + \mu i$, then $\lambda - \mu$ is an eigenvalue of a. Therefore, by subtracting μ from the diagonal elements of r, we get the eigenvalues of a, which are squared of singular values of m. The eigenvectors of a' are the same as the eigenvectors of a and, therefore, the same as the right singular vectors of m.

□

A.2 Theorems and proofs of section 2.3

We start with a simple example that shows an issue with this procedure, and then, present our theoretical results that show a more general and fundamental limitation for a family of convolutional layers.

Consider a 2d convolutional layer whose kernel is 1×1 with a value of c. Applying this kernel to any input of size $n \times n$ scales the values of the input by c. The equivalent matrix form of this linear transformation is an $n \times n$ identity matrix scaled by c. We know that this matrix has a rank of n, and all the singular values are equal to 1. Therefore, if the new spectrum, s', does not represent a full-rank transformation, or if its singular values are not all equal, we cannot find a convolutional layer with a 1×1 kernel with s' as its spectrum.

To prove theorem 2.2, we first prove the following lemma, which gives a closed form for the singular values of a convolutional layer with circular padding.

Lemma A.1. *For a convolutional layer with* 1 *input channel and* m *output channels (the same result holds for convolutions with* 1 *output channel and* m *input channels) and circular padding, if the vectorized form of the* j*-th channel of the filter is given by* $\boldsymbol{f^{(j)}} = [f_0^{(j)}, f_1^{(j)}, \ldots, f_{k-1}^{(j)}]$*, the singular values are:*

$$\int(\omega) = \left\{ \sqrt{\sum_{j=1}^{m} \int_k^{(j)^2}(\omega)},\ k = 0, 1, \ldots, n-1 \right\}, \tag{1}$$

where for $j \in 1, \ldots, m$*:*

$$\int^{(j)}(\omega) = \left[\sqrt{c_0^{(j)} + 2\sum_{i}^{k-1} c_i^{(j)} \mathrm{e}(\omega^{k \times i})},\ k = 0, 1, 2, \ldots, n-1 \right]^\top$$

in which $c_i^{(j)}$*'s are defined as:*

$$\begin{aligned}
c_0^{(j)} &:= f_0^{(j)^2} + f_1^{(j)^2} + \cdots + f_{k-1}^{(j)^2}, \\
c_1^{(j)} &:= f_0^{(j)} f_1^{(j)} + f_1^{(j)} f_2^{(j)} + \cdots + f_{k-2}^{(j)} f_{k-1}^{(j)}, \\
&\vdots \\
c_{k-1}^{(j)} &:= f_0^{(j)} f_{k-1}^{(j)}.
\end{aligned}$$

For proving lemma A.1, we first present and prove lemma A.2 which proves a similar result for convolutional layers with only 1 input and output channel.

Lemma A.2. *Given a convolutional layer with single input and output channel and circular padding, if the vectorized form of the kernel is given by* $\boldsymbol{f} = [f_0, f_1, \ldots, f_{k-1}]$*, the singular values are:*

$$\mathcal{S}(\omega) = \left\{ \sqrt{c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^{j\times i})},\ j = 0, 1, 2, \ldots, n-1 \right\} \tag{2}$$

where c_i*'s are defined as:*

$$\begin{aligned} c_0 &:= f_0^2 + f_1^2 + \cdots + f_{k-1}^2, \\ c_1 &:= f_0 f_1 + f_1 f_2 + \cdots + f_{k-2} f_{k-1}, \\ &\vdots \\ c_{k-1} &:= f_0 f_{k-1}. \end{aligned}$$

Proof. The above convolutional operator is equivalent to a circulant matrix A with $[f_m, f_{m+1}, \ldots, f_{k-1}, 0, \ldots, 0, f_0, f_1, \ldots, f_{m-1}]^T$ as its first row, where $m = \lfloor \frac{k}{2} \rfloor$ Jain (1989); Sedghi et al. (2018). Singular values of A are the eigenvalues of $M = A^T A$. Circulant matrices are closed with respect to multiplication, and therefore M is a symmetric circulant matrix. It is easy to check that the first row of M is $r = [c_0, c_1, \ldots, c_{k-1}, 0, \ldots, 0, c_{k-1}, \ldots, c_1]^T$.

Now, we know the eigenvalues of a circulant matrix with first row $v = [a_0, a_1, \ldots, a_{n-1}]^T$ are given by the set $\Lambda = \{v\Omega_j, j = 0, 1, \ldots, n-1\}$, where $\Omega_j = [\omega^{j\times 0}, \omega^{j\times 1}, \ldots, \omega^{j\times n-1}]$. So eigenvalues of M are given by the set below:

$$\begin{aligned} \{r\Omega_j,\ j = 0, 1, \ldots, n-1\} = \{&c_0 + c_1\omega^{j\times 1} + c_2\omega^{j\times 2} + \cdots + c_{k-1}\omega^{j\times(k-1)} \\ &+ c_{k-1}\omega^{j\times(n-k+1)} + c_{k-2}\omega^{j\times(n-k+2)} + \cdots + c_1\omega^{j\times(n-1)}, \\ &j = 0, 1, \ldots, n-1\}. \end{aligned}$$

Note that $\omega^{j\times i}$ has the same real part as $\omega^{j\times(n-i)}$, but its imaginary part is mirrored with respect to the real axis (the sign is flipped). Therefore, $\omega^{j\times i} + \omega^{j\times(n-i)} = 2\Re(\omega^{j\times i})$. Using this equality the summation representing the eigenvalues of M can be simplified to:

$$\{c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^{j\times i}),\ j = 0, 1, \ldots, n-1\},$$

and therefore the singular values can be derived by taking the square roots of these eigenvalues. □

Now using lemma A.2, we can easily prove lemma A.1.

Proof. The convolutional layer with m output channels and one input channel can be represented by a matrix $M = [M_1, M_2, \ldots, M_m]^T$, where each M_i is an $n \times n$ circulant matrix representing the $i-$th channel. Therefore $A = M^T M$ (or MM^T when there are multiple input channels) can be written as $\sum_{j=1}^{m} M_j^T M_j$. So, for circulant matrix A we have $c_i = \sum_{j=1}^{m} c_i^{(j)}$, and the proof can be completed by using Lemma A.2. □

Now, we present the proof of theorem 2.2 simply by using lemma A.1.

Proof. Note that in the closed form of singular values given by lemma A.1, we only have the real parts of the roots of unity. So, for any non-real root of unity, we get a duplicate singular value because mirroring that root around the real axis (i.e., flipping the sign of the imaginary part) gives another root of unity with the same real component. Only the roots that are real might derive singular values that do not have duplicates. The only such roots with real parts are 1 and -1 (-1 is a root only for even n). Therefore, except for at most two singular values, other ones always have duplicates. This shows a more general limitation in the representation power of convolutional layers in representing arbitrary spectrums. □

Next, we state corollary A.3, which uses the results of lemma A.1 to give an easy-to-compute lower and upper bounds for the spectral norm of the convolutional layers with either one input or output channel. When the filter values are all positive, these bounds become equalities and provide and easy way to compute the exact spectral norm.

Corollary A.3. *Consider a convolutional layer with* 1 *input channel and* m *output channels or* 1 *output channel and* m *input channels and circular padding. if the vectorized form of the kernel of the* j*-th channel is given by* $\boldsymbol{f}^{(j)} = [f_0^{(j)}, f_1^{(j)}, \dots, f_{k-1}^{(j)}]$*, and the largest singular value of the layer is* σ_1*, then:*

$$\sqrt{\sum_{j=1}^{m}\left(\sum_{i=0}^{k-1} f_i^{(j)}\right)^2} \leq \sigma_1 \leq \sqrt{\sum_{j=1}^{m}\left(\sum_{i=0}^{k-1} |f_i^{(j)}|\right)^2}, \tag{3}$$

and therefore the equalities hold if all $f_i^{(j)}$*s are non-negative.*

To make the proof more clear, we first present a similar corollary for lemma A.2, which proves similar results for convolutions with only 1 input and output channel.

Corollary A.4. *Consider a convolutional layer with single input and output channels and circular padding. If the vectorized form of the kernel is given by* $\boldsymbol{f} = [f_0, f_1, \dots, f_{k-1}]$*, and the largest singular value of the layer is* σ_1*, then:*

$$\sum_{i=0}^{k-1} f_i \leq \sigma_1 \leq \sum_{i=0}^{k-1} |f_i|, \tag{4}$$

and therefore, the equalities hold if all f_i*s are non-negative.*

Proof. From lemma A.2, we can write:

$$\begin{aligned}
\int(\omega) &= \sqrt{c_0 + 2\sum_{i}^{k-1} c_i \mathrm{e}(\omega^i)} = \sqrt{|c_0 + 2\sum_{i}^{k-1} c_i \mathrm{e}(\omega^i)|} \\
&\leq \sqrt{|c_0| + 2\sum_{i}^{k-1} |c_i \mathrm{e}(\omega^i)|} \leq \sqrt{|c_0| + 2\sum_{i}^{k-1} |c_i|},
\end{aligned}$$

where the last inequality is due to the fact that $\mathrm{e}(\omega^i) \leq 1$. Now, for c_is we have:

$$\begin{aligned}
|c_0| &= f_0^2 + f_1^2 + \cdots + f_{k-1}^2, \\
2|c_1| &\leq 2\left(|f_0||f_1| + |f_1||f_2| + \cdots + |f_{k-2}||f_{k-1}|\right), \\
&\vdots \\
2|c_{k-1}| &\leq 2\left(|f_0||f_{k-1}|\right).
\end{aligned}$$

Note that the summation of the right-hand sides of the above inequalities is equal to $(\sum_{i=0}^{k-1} |f_i|)^2$ therefore:

$$\int(\omega) \leq \sum_{i=0}^{k-1} |f_i| \implies \sigma_1 \leq \sum_{i=0}^{k-1} |f_i|.$$

Since 1 is always one of the $n-$th roots of unity, if all the f_is are non-negative, all the above inequalities hold when $\omega = 1$ is considered. Now, note that when $\omega = 1$:

$$
\begin{aligned}
&\int^2(1) = c_0 + 2\sum_{i}^{k-1} c_i = \\
&f_0^2 + f_1^2 + \cdots + f_{k-1}^2 \quad (c_0) \\
&+ 2f_0f_1 + 2f_1f_2 + \cdots + 2f_{k-2}f_{k-1} \quad (c_1) \\
&\vdots \\
&+ 2f_0f_{k-1} \quad (c_{k-1}) \\
&= (\sum_{i=0}^{k-1} f_i)^2
\end{aligned}
$$

Therefore, $\sum_{i=0}^{k-1} f_i$ is a singular value of the convolution layer, which gives a lower bound for the largest one and this completes the proof.

□

Now we give the proof for corollary A.3.

Proof. From lemma A.1, we can write:

$$
\int(\omega) = \sqrt{\sum_{j=1}^{m} \int^{(j)^2}(\omega)} \leq \sqrt{\sum_{j=1}^{m} (\sum_{i=0}^{k-1} |f_i^{(j)}|)^2},
$$

where the inequality is a result of using theorem A.4 for each $\int^{(j)}(\omega)$. For showing the left side of inequality, we set $\omega = 1$ (1 is one of the n-th roots of unity), and again use lemma A.1 to get:

$$
\int(1) = \sqrt{\sum_{j=1}^{m} \int^{(j)^2}(1)} = \sqrt{c_0^{(j)} + 2\sum_{i}^{k-1} c_i^{(j)} \mathrm{e}(1)} = \sqrt{\sum_{j=1}^{m} (\sum_{i=0}^{k-1} f_i^{(j)})^2},
$$

where the last equality was shown in proof of theorem A.4. This shows the left side of the inequality in (4) is one of the singular values of the layer, and hence is a lower bound for the spectral norm (the largest singular value of the layer).

□

B Other algorithms

B.1 Fast Clipping

First, we should note that to project a linear operator $m = usv^\top$ to the space of operators with a bounded spectral norm of c, it is enough to construct $a' = us'v^\top$, where $s'_{i,i} = \min(s_{i,i}, c)$ (Lefkimmiatis et al., 2013). Therefore, for this projection, we will not need to extract and modify the whole spectrum of a (as in (Sedghi et al., 2018; Senderovich et al., 2022)), and only clipping the singular values that are larger than c to be exactly c would be enough. Note that after computing the largest spectral norm (e.g., using PowerQR method) by simply dividing the parameters of the affine model by the largest singular value, the desired bound on the spectral norm would be achieved, and that is the basis of some of the prior work (Miyato et al., 2018; Farnia et al., 2018; Gouk et al., 2021); however, this procedure scales the whole spectrum rather than projecting it to a norm ball. Therefore, it might result in suboptimal optimization of the network due to replacing the layer with one that is not the most similar to it in the norm ball. This adverse effect can be seen in table 1 as well. Thus, to

Algorithm 3 FastClip (f_W, X, c, N, η)

1: **Input:** Affine function f_W, Dataset X, Clip value c, Number of SGD iterations N, Learning rate η
2: **Output:** Trained affine function $f_{W'}$ with singular values clipped to c
3: $v \leftarrow$ Random input vector
4: $\sigma_1, v_1 \leftarrow \text{PowerQR}(f_W, v_1, 10)$
5: **for** $i = 1$ **to** N **do**
6: $\quad X_b \leftarrow \text{SampleFrom}(X)$ *// Sample a batch*
7: $\quad W = W - \eta \nabla_W \ell(f_W(X_b))$ *// SGD step*
8: $\quad \sigma_1, v_1 \leftarrow \text{PowerQR}(f_W, v_1, 1)$
9: $\quad$ **if** i isDivisibleBy 100 **then**
10: $\quad\quad f_W, \sigma_1, v_1 \leftarrow \text{Clip}(f_W, \sigma_1, v_1, c, 1, 10)$
11: $\quad$ **end if**
12: **end for**
13: **Return** f_W

resolve both aforementioned issues, our algorithm iteratively shrinks the largest singular value by substituting the corresponding rank 1 subspace in an implicit manner.

To speed up our clipping algorithm enough to be used during the training of large models without adding much overhead, we intertwined it with the outer SGD optimization used for training the model and the PowerQR method, as shown in algorithm 3. As we mentioned in section 2.1, the PowerQR method with warm start is able to track the largest singular values and corresponding vectors by running as few as one iteration per SGD step (lines 4 and 8 in algorithm 3). Whenever the clipping method is called, we use this value and its corresponding vector as the input to the algorithm. If we call algorithm 2 every few iterations of SGD, since the number of calls to this method becomes large and the changes to the weight matrix are slow, we do not need to run its `while` loop many times. Our experiments showed performing the clipping method every 100 step and using only 1 iteration of `while` and `for` loops (5 iteration of `while` when batch norm is used) is enough for clipping the trained models (lines 9 and 10). Because after the clipping, the corresponding singular value of v has shrunk, we need to perform a few iterations of PowerQR on a new randomly chosen vector (since v is orthogonal to the other right singular vectors) to find the new largest singular value and corresponding right singular vector.

B.2 Modifying the whole spectrum

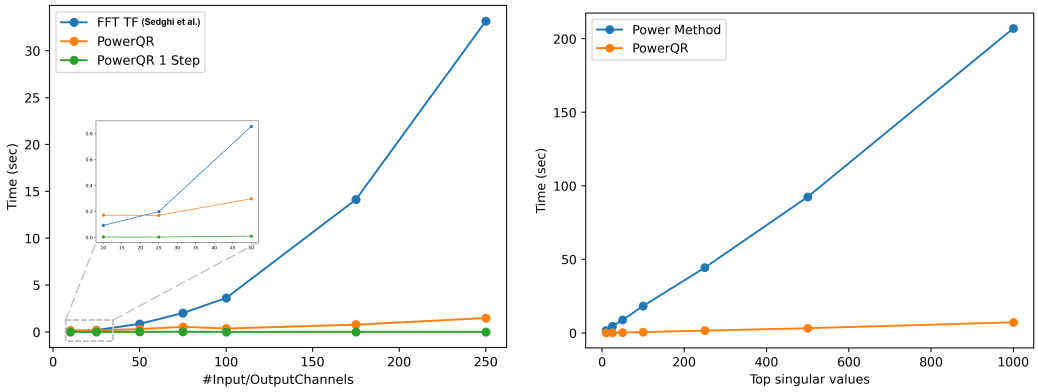
If we use PowerQR to extract singular values and right singular vectors s and v from $m_w = usv^t$, then given new singular values s', we can modify the spectrum of our function generate a linear operator $f' : x \to m'_w x + b$, where $m'_w = us'v^t$, without requiring the exact computation of u or m_w:

$$f_w(vs^{-1}s'v^t x) - f(0) = usv^t vs^{-1}s'v^t x = us'v^t x = f'(x) - b.$$

Although f' is a linear operator with the desired spectrum, it is not necessarily in the desired form. For example, if f_w is a convolutional layer with w as its kernel, $f_w(vss'v^t x)$ will not be in the form of a convolutional layer. To find a linear operator of the same form as f_w, we have to find new parameters w' that give us f'. For this, we can form a convex objective function and use SGD to find the parameters w' via regression:

$$\min_{w'} {}_x \left\| f_{w'}(x) - f_w(vs^{-1}s'v^t x) \right\|_{\text{f}}^2, \tag{5}$$

where x is randomly sampled from the input domain. Note that we do not need many different vectors x to be sampled for solving 5. if the rank of the linear operator $f_w(.)$ is n, then sampling as few as n random data points would be enough to find the optimizer $f_{w'}$. This procedure can be seen as two successive projections, one to the space of linear operators with the desired spectrum and the other one to the space of operators with the same form as f_w (e.g., convolutional layers). This method can be very slow because the matrix v can be very large. One might reduce the computational cost by working with the top singular values and singular vectors and deriving a low-ranked operator. This computation can be made more efficient by choosing the order of multiplication to be $(v(ss'))(v^t x)$



(a) Spectral norm during training

(b) Test loss

Figure 2: **a.** Comparison of the run-time of PowerQR (algorithm 1), with either 1 or 100 iterations, to `FFT TF` Sedghi et al. (2018); Senderovich et al. (2022). In this plot, we considered a 2d-convolutional layer with 3×3 filters and different input and output channels. The convolution is applied to a 16×16 image. **b.** Comparison of the run-time of PowerQR (algorithm 1) to that of the pipeline used by Virmaux & Scaman (2018) for computing the top-k singular values. We considered a 2d-convolutional layer with 3×3 filters and different numbers of input/output channels. The convolution is applied to a 32×32 image.

and using the fast multiplication with the diagonal matrix (ss') by extracting the diagonal elements and performing broadcasted element-wise multiplication. However, this method is still not practical for controlling the spectral norm during training.

C Experiments

In this section, we elaborate on the experiments and results pointed out in section 3 and present some additional experiments to show the advantages of our proposed methods.

C.1 Comparison of PowerQR

Figure 2a shows the comparison in the run-time of our method `PowerQR` (algorithm 1) to the method introduced by Sedghi et al. (2018); Senderovich et al. (2022) (`FFT TF`), for a convolutional layer when the number of channels vary. Figure 2b shows the run-time comparison of our method, PowerQR, to successive calls to power method Virmaux & Scaman (2018), for extracting the top-k singular values and vectors. Our method is extremely faster as k increases.

C.2 Comparison of FastClip to other methods

We compare the accuracy of our clipping method (`fastclip`) to the ones introduced by Gouk et al. (2021), Miyato et al. (2018), Senderovich et al. (2022), and Delattre et al. (2023). For comparing the correctness of the clipping methods, we use a simple model constituting one convolutional layer and a linear layer and train it for 120 epochs on MNIST. The convolutional layer has 1 input channel, 32 output channels. For the filter, we choose 4 different settings, and plot the spectral norm of convolutional layer during training and present the results in fig. 3. The four different settings, from left to right are: 1. kernel of size 3 with zero padding, 2. kernel of size 3 with reflection padding, 3. kernel of size 3 and dilation 2, and 4. kernel of size 5 with replicate padding and stride of 2, respectively. As the figure shows, our method converges to the desired spectral norm in all settings. This is not the case for any of the other methods, and there are cases for either of them that the spectral norm of the layer converges to a wrong values.

We also computed the layer-wise spectral norm of all the convolutional and linear layers for the ResNet18 models that was trained using any of the clipping methods on MNIST and CIFAR10. As Figure 3 shows, our method has done the most accurate clipping for all the layers, while being must

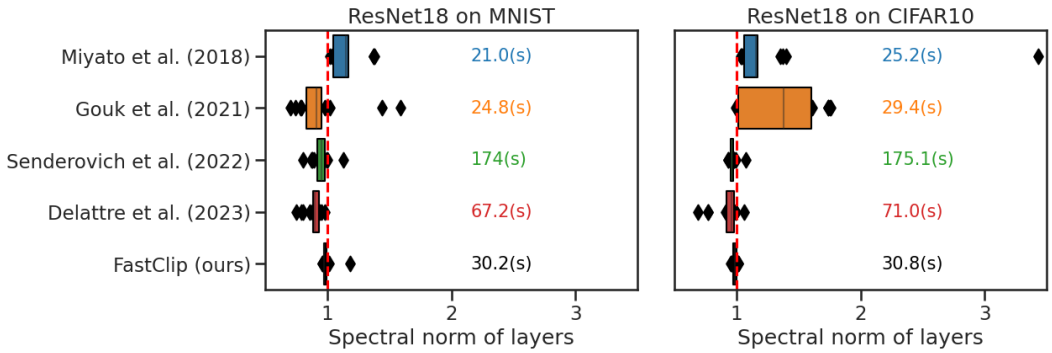


Figure 3: Comparison of the final spectral norm of the linear and convolutional layers of ResNet18 model trained on MNIST and CIFAR10 for different clipping methods. **the target value is 1**. For each of the clipping method, we have reported the average time per epoch on a NVIDIA A40 GPU. Our method is the most accurate clipping method while being very fast.

faster than the clipping method of Senderovich et al. (2022); Delattre et al. (2023) which are the next most accurate methods.

C.3 Generalization and Robustness

Since the Lipschitz constant of the composition of functions can be upper-bounded by the multiplication of their Lipschitz constants, clipping the spectral norm of the linear and convolutional layers contributes to training a model that has better generalization (Bartlett et al., 2017) and is more robust to adversarial perturbations Szegedy et al. (2013). Therefore, we tested all the clipping methods by using them during the training and computing the accuracy of the corresponding models on the test set and adversarial examples generated by two common adversarial attacks. For this experiment, we used the same model that was used by Delattre et al. (2023) and Senderovich et al. (2022), which is a ResNet18 model for which the sum of residual connections has is divided by two to make sure the Lipschitz factor of the module is upper bounded by 1 when both the connections are 1-Lipschitz. We trained this model on MNIST and CIFAR10. As table 1 shows, our clipping method achieves the best improvement in the accuracy on both the test set and the adversarial examples. The improvement in the robustness of the model, however, is not as much as when the batch normalization layers are removed from the model. The reason for the lack of the expected boost in the robustness of the models is shown in Figure 4, which shows although the clipping method is used, the concatenation of some convolutional layers with batch normalization layers form linear operators with large spectral norms. This adverse effect of batch norm layers on the robustness of the models has been empirically shown Xie & Yuille (2019); Benz et al. (2021). As table 1 shows, removing the batch normalization layers leads to a huge improvement in the robustness of the clipped models; however, the models achieves worse accuracy on the test set.

D Further Related Works

Convolutional layers are a major class of implicitly linear layers that are used in many models in various domains. They are compressed forms of linear transformations with an effective rank that depends on the dimensions of input rather than their filters. A 2d-convolutional layer with a stride of 1 and zero padding, whose kernel has dimensions (c_{out}, c_{in}, k, k) and is applied to an input of size $n \times n$ represents a linear transformation of rank $\min(c_{in}, c_{out})n^2$. This compression is beneficial for training large and deep models as it reduces the number of parameters drastically while keeping the flexibility of performing the higher-rank transformation. Nevertheless, this compression makes it challenging to compute and control the spectrum. Clearly, a straightforward way of computing the spectrum of the convolutional layers is to unroll them to build the explicit matrix form of the transformation and compute its singular values. In addition to the complexity of computing the matrix representation for various padding types and sizes and different strides, this procedure is very slow, as represented in prior work (Sedghi et al., 2018), and forfeits the initial motivation for using them.

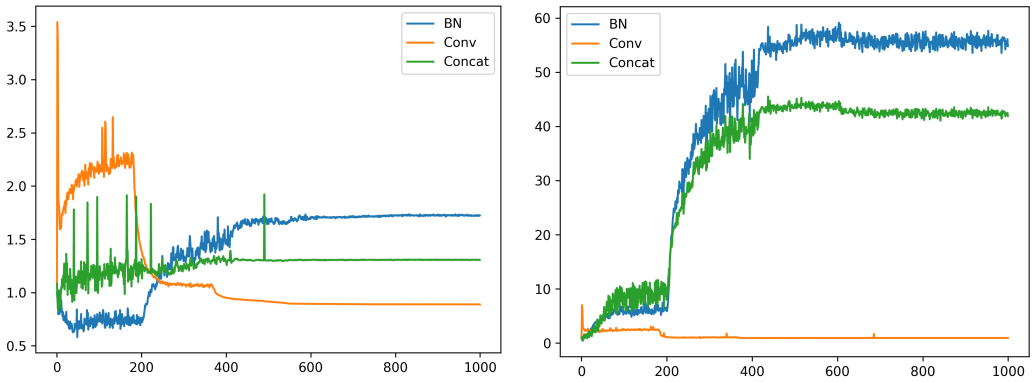


Figure 4: Spectral norm of two convolutional layers and their succeeding batch norm layers from the clipped ResNet-18 model trained on CIFAR-10. For each layer, spectral norms of the clipped convolution layer, its succeeding batch norm layer, and the concatenation of the two are plotted.

The problem becomes much more challenging if one decides to bound the spectral norms of these layers during training without heavy computations that make this regularization impractical.

For computing the singular values of linear and convolutional layers, Virmaux & Scaman (2018) perform the power method implicitly using the automatic differentiation that is similar to our implicit approach. However, they do not provide any algorithm for bounding (clipping) the spectral norms and leave that for future work. Also, for extracting multiple singular values, they use successive runs of their algorithm followed by deflation, which is much less efficient than our extraction method, as shown in appendix C.1. They also do not investigate the application of their method to a concatenation of the layers and ignore the behavior of convolutional layers in the presence of batch normalization layers in their study.

Some prior work relies on approximate methods to control the spectral norm of convolutional layers. Gouk et al. (2021), at each SGD iteration, approximate the spectral norm of the layer by considering the output of the layer on samples of the previous batch. They compute the ratio of the length of the output to the length of the input, consider the largest one as the approximate value, and scale the weights of the layer by that, which leads to scaling all the singular values, including the largest one. Miyato et al. (2018) approximate the original transformation by reshaping the kernel to a $n_{in}k^2 \times n_{out}$ matrix and perform the method they designed for linear the layers to control the spectral norm of the convolutional layer. Farnia et al. (2018) use the convolution transpose operator to perform a similar clipping method to the one introduced by Miyato et al. (2018) for linear layers.

Some other works consider additional constraints. Cisse et al. (2017) constrained the weight matrices to be orthonormal and performed the optimization on the manifold of orthogonal matrices. Sedghi et al. (2018) gave a solution that works only for convolutional layers with circular padding and no stride. For other types of convolutional layers, they approximate the spectral norm by considering the circular variants, however, the approximation error can be large for these methods. (Senderovich et al., 2022) extends the method of Sedghi et al. (2018) to support convolutional layers with strides. It also increases its speed and memory efficiency, albeit with the cost of expressiveness of convolutions, through specific compressions. Delattre et al. (2023) use Gram iteration to derive an upper-bound on the spectral norm of the circulant approximation in an effieint way. Therefore, in the best case, they will have the same approximto error as (Sedghi et al., 2018; Senderovich et al., 2022). A parallel line of research on controlling the spectrum of linear layers seeks to satisfy an additional property, gradient norm preserving, in addition to making each of the linear and convolutional layers 1-Lipschitz Singla & Feizi (2021); Yu et al. (2021); Trockman & Kolter (2021); Singla et al. (2021); Xu et al. (2022). Some of the state-of-the-art methods in this line of work are also based on the clipping method introduced by Sedghi et al. (2019) Yu et al. (2021); Xu et al. (2022). Therefore, they are even slower than the FFT method by Sedghi et al. (2019). We hope our new efficient method provides an opportunity for researchers to develop more efficient methods for this line of work in the future.