OPT2022: 14th Annual Workshop on Optimization for Machine Learning

A Better Way to Decay: Proximal Gradient Training Algorithms for Neural Nets

Liu Yang LIU.YANG@WISC.EDU
Jifan Zhang JIFAN@CS.WISC.EDU
Joseph Shenouda JSHENOUDA@WISC.EDU
Dimitris Papailiopoulos DIMITRIS@PAPAIL.IO
Kangwook Lee KANGWOOK.LEE@WISC.EDU
Robert D. Nowak RDNOWAK@WISC.EDU
University of Wisconsin, Madison, USA

Abstract

Weight decay is one of the most widely used forms of regularization in deep learning, and has been shown to improve generalization and robustness. The optimization objective driving weight decay is a sum of losses plus a term proportional to the sum of squared weights. This paper argues that stochastic gradient descent (SGD) may be an inefficient algorithm for this objective. For neural networks with ReLU activations, solutions to the weight decay objective are equivalent to those of a different objective in which the regularization term is instead a sum of products of ℓ_2 (not squared) norms of the input and output weights associated each ReLU. This alternative *(and effectively equivalent)* regularization suggests a novel proximal gradient algorithm for network training. Theory and experiments support the new training approach, showing that it can converge much faster to the *sparse* solutions it shares with standard weight decay training.

1. Introduction

Weight decay is the most prevalent form of explicit regularization in deep learning, which corresponds to regularizing the sum of squared weights in the model. It has been shown to improve the generalization performance of deep neural networks [1, 17, 41] and even plays a role in making models more robust [8, 28]. This paper argues that weight decay regularization can be equivalently and more effectively incorporated into training via shrinkage and thresholding. Moreover, the new training algorithms produce better solutions with more desirable properties compared to standard weight decay (see initial result in Fig. 1 and 2). Weight decay can produce solutions with similar qualities, but only after an inordinate number of training epochs. This may explain why characteristics like sparsity are not associated with weight decay training. Shrinkage and thresholding converges to such solutions *much* faster.

To gain some intuition into the connection between weight decay and thresholding let us consider a key aspect of most neural networks. Deep architectures include many types of processing steps, but the basic neuron or unit is common in almost all. Consider a single unit of the form $\boldsymbol{v}\sigma(\boldsymbol{w}^T\boldsymbol{x}+b)$, where σ is a fixed activation function and $\boldsymbol{v}, \boldsymbol{w}, b$ denote its trainable output, input, and bias weights. This single unit is homogeneous if $\boldsymbol{v}\sigma(\boldsymbol{w}^T\boldsymbol{x}+b) = \alpha\boldsymbol{v}\sigma(\alpha^{-1}(\boldsymbol{w}^T\boldsymbol{x}+b))$ for all $\alpha > 0$. Weight decay regularization of this unit corresponds to adding a term proportional to $\frac{1}{2}\left(\|\boldsymbol{w}\|_2^2 + \|\boldsymbol{v}\|_2^2\right)$ to the optimization objective. Among all the equivalent representations of the unit, it is easy to verify that $\alpha^2 = \|\boldsymbol{w}\|_2/\|\boldsymbol{v}\|_2$ produces the smallest regularization term by the AM-GM inequality. Thus, at a minimum of the objective we have $\frac{1}{2}\left(\|\boldsymbol{w}\|_2^2 + \|\boldsymbol{v}\|_2^2\right) = \|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$. This simple fact is understood [9, 26], albeit perhaps not widely. The form $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ (coined as ℓ_2-PATH-NORM) is reminiscent of ℓ_1-type regularization functions, such as the lasso and group lasso regularizers.

In this paper, we propose to replace the weight decay term with the effectively equivalent proximal operation that corresponds to a shrinkage and thresholding step for each homogeneous unit. Theory (Sec. 3) and experiments (Sec. 5) have shown this leads to solutions that are sparser, have smaller Lipschitz constants, and are more robust than solutions based on weight decay training.

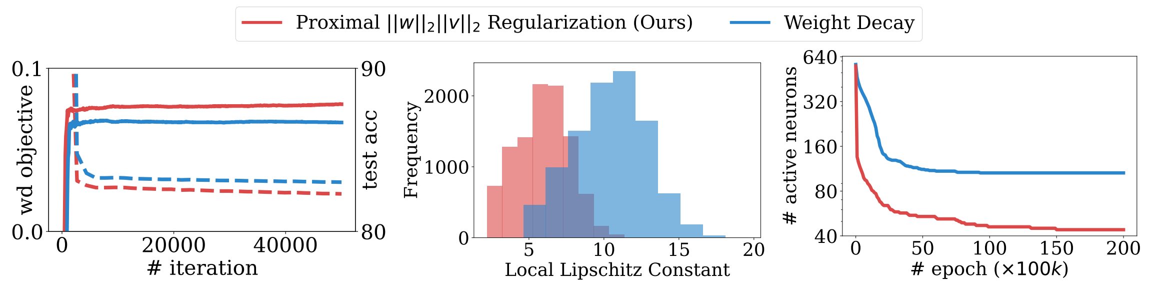

Figure 1: Comparison between Weight Decay and our proposed proximal algorithm for ℓ_2-PATH-NORM on (*left*) weight decay objective (dashed line) and test accuracy convergence (solid line), (*middle*) the histogram of local Lipschitz constant on test dataset, and (*right*) number of active neurons. Please refer to Appendix C.2 for detailed experiments setup.

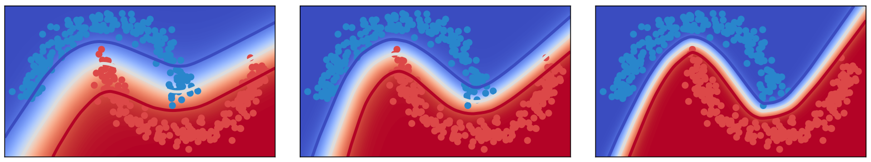

Figure 2: Decision Boundary of a network trained for binary classifier with Weight Decay (*left*) and our proposed proximal algorithm for ℓ_2-PATH-NORM (*middle*) after the same amount of training iterations. Both algorithm eventually converge to the same function (*right*). The white line is the decision boundary while the red and blue lines represent 90% confidence. Please refer to Appendix C.2 for detailed experiments setup.

2. Related Works

It has been well-understood that weight decay helps improve the robustness and generalization of the trained model [17, 20]. Lines of work have focused on theoretically analyzing the equivalent form of weight decay: [9] was the first work to highlight an equivalence between weight decay and lasso on the output weights of a neural network. Later [25, 26] proved that for a shallow, single output network, training the network with squared ℓ_2 regularization on all the weights (i.e. weight decay) is equivalent to regularizing with the $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_1$ norm in the $(\boldsymbol{w}, \boldsymbol{v})$ homogeneous units. [29, 30] demonstrated regularizing the shallow neural network with $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_1$ will lead to finite width neural network which are the optimal solution to learning problems in second-order Radon-domain Bounded Variation $\mathcal{R}\mathrm{BV}^2$ function space. Furthermore they demonstrate how the $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_1$ "path regularizer" is indeed the natural norm for the functions in $\mathcal{R}\mathrm{BV}^2$. This result was later generalized for the deep neural network case in [31]. In [32] they also used the equivalent form of weight decay to understand the minimum norm required to fit any function to an infinitely wide univariate shallow network. More recently, [6] used this equivalence to utilize convex geometry in understanding the optimal solution for neural networks trained with weight decay. Based on the rescaling equivalence on homogeneous units, [35] proposed an algorithm to iteratively minimize the sum of squared weights for the weight decay objective. Another property of weight decay is that it encourages sparse structure, as confirmed in [14]. [5, 38] also indicate weight decay explicitly promotes low-rank solution for parallel network. However, the sparse structure is not fully revealed when training with weight decay for finite iterations. Our proposed algorithm, on the other hand, directly exploits the sparse structure with a weight decay equivalent form of regularization. Other recent works propose related forms of regularization, and argue that these are sometimes better than weight decay: In [24] introduced the “path regularizer", a generalization of the regularizer in [26] for deep neural networks and showed how it can lead to solutions that generalize better and are more robust [4, 15]. Similarly, [21] utilize the homogeneity of ReLU neural network and proposed “scale shift invariant” algorithm. Proximal gradient type of algorithm has been proposed for 1-path-norm in [18], where they focus on the $\|\boldsymbol{w}\|_1\|\boldsymbol{v}\|_1$ norm of a homogeneous unit $(\boldsymbol{w}, \boldsymbol{v})$ in shallow networks.

3. Training Neural Networks with Weight Decay

Let $\boldsymbol{W}$ denote the weights of a multilayer neural network. Most standard “training" algorithms fit neural networks to data by minimizing an objective of the form

$$F_\lambda(\boldsymbol{W}) \;:=\; L(\boldsymbol{W}) + \frac{\lambda}{2}\, R(\boldsymbol{W})$$

where $L(\boldsymbol{W})$ is a loss function on the training data, $R(\boldsymbol{W})$ is the sum of squared weights, and $\lambda \geq 0$. When minimized by gradient descent methods, $R(\boldsymbol{W})$ leads to the common practice known as “weight decay." We will call $F_\lambda(\boldsymbol{W})$ the *weight decay objective* and λ the *weight decay parameter*.

The neural network may have a general architecture (fully connected, convolutional, etc) and may involve many types of units and operations (e.g., nonlinear activation functions, pooling/subsampling, etc). This paper focuses on those units in the architecture that are *homogeneous*. A function σ is homogeneous if it satisfies $\sigma(\alpha x) = \alpha\sigma(x)$ for any $\alpha > 0$. For example, the popular Rectified Linear Unit (ReLU), Leaky ReLU and PReLU are homogeneous. Consider a neuron with input weights $\boldsymbol{w} \in \mathbb{R}^p$ and output weights $\boldsymbol{v} \in \mathbb{R}^q$. The neuron produces the mapping $\boldsymbol{x} \mapsto \boldsymbol{v}\,\sigma(\boldsymbol{w}^T\boldsymbol{x})$. Because σ is homogeneous, $\boldsymbol{v}\,\sigma(\boldsymbol{w}^T\boldsymbol{x}) = \alpha\boldsymbol{v}\,\sigma(\alpha^{-1}\boldsymbol{w}^T\boldsymbol{x})$, for every $\alpha > 0$. The following **Neural Balance Theorem** provides an important characterization of representations with the minimum sum of squared weights.

Theorem 1 *(Neural Balance Theorem) Let f be a function represented by a neural network and consider a representation of f with the minimum sum of squared weights. Then the weights satisfy the following* balancing constraints. *Let $\boldsymbol{w}$ and $\boldsymbol{v}$ denote the input and output weights of any homogeneous unit in this representation. Then $\|\boldsymbol{w}\|_2 = \|\boldsymbol{v}\|_2$.*

Proof Assume there exists a representation f with minimum sum of squared weights, but does not satisfy the constraint for a certain unit. Because the homogeneous unit is homogeneous, its input and output weights, $\boldsymbol{w}$ and $\boldsymbol{v}$, can be scaled by $\alpha > 0$ and $1/\alpha$, respectively, without changing the function. The solution to the optimization $\min_{\alpha>0} \|\alpha\,\boldsymbol{w}\|_2^2 + \|\alpha^{-1}\boldsymbol{v}\|_2^2$ is $\alpha = \sqrt{\|\boldsymbol{v}\|_2/\|\boldsymbol{w}\|_2}$. Thus, we can rescale the input and output weights to meet the constraint while preserving f yet reducing the sum of squared weights, contradicting the beginning assumption in the proof. ■

Remark 2 *Versions of Neural Balance Theorem (NBT) and its consequences have been discussed in the literature [9, 24, 25, 27, 30–32], but usually in the setting of fully connected ReLU architectures. We note here that NBT holds for any architecture (fully connected, convolutional, pooling layers, etc.) and every homogeneous unit in the architecture.*

Theorem 1 tells us that the norms of the input and output weights of each homogeneous unit must be equal to each other at a minimum of F_λ (see Appendix A.1 for further discussion). To illustrate a key implication of this, consider a neural network with L layers of homogeneous neurons. Let $\boldsymbol{w}_{i,k}$ and $\boldsymbol{v}_{i,k}$ denote the input and output weights of the ith neuron in the kth layer. Theorem 1 shows that a global minimizer of the weight decay objective F_λ with any $\lambda > 0$ must satisfy the balancing constraints of the theorem. At a global minimum the ith neuron in the kth layer contributes the term $\frac{1}{2}\left(\|\boldsymbol{w}_{i,k}\|_2^2 + \|\boldsymbol{v}_{i,k}\|_2^2\right) = \|\boldsymbol{w}_{i,k}\|_2\|\boldsymbol{v}_{i,k}\|_2$ to the overall sum of squares $R(\boldsymbol{W})$. Therefore, at a global minimum of F_λ

$$R(\boldsymbol{W}) \; = \; \frac{1}{2}\sum_{i=1}^{n_1} \|\boldsymbol{w}_{i,1}\|_2^2 + \frac{1}{2}\sum_{i=1}^{n_L} \|\boldsymbol{v}_{i,L}\|_2^2 + \sum_{k=1}^{L}\sum_{i=1}^{n_k} \|\boldsymbol{w}_{i,k}\|_2 \, \|\boldsymbol{v}_{i,k}\|_2 \tag{1}$$

n_k is the number of neurons in the kth layer. The expression accounts for the fact that certain weights may appear twice (associated with the input/output of two different neurons). Though we mainly discuss in the multi-layer perceptron regime, Theorem 1 also applied to convolutional neural network, as presented in Appendix A.2. Note that $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ is the Lipschitz constant of the function $\boldsymbol{v}\,\sigma(\boldsymbol{w}^T\boldsymbol{x})$, which shows that weight decay encourages solutions that are Lipschitz smooth (detailed proof in Appendix A.3).

4. A New Algorithm for Minimizing the Weight Decay Objective

In this section we propose a new neural network training algorithm. It exploits the fact that a global minimizer to the weight decay objective is related to a sum of norm-products, as shown in Eqn. (1). The following expression for the sum of squared weights will be helpful in deriving the new algorithm. Again consider a neural network with homogeneous neurons and L layers. Since the inputs weights of a neuron in layer k involve the output weights of neurons in the preceding layer $k-1$, we will associate the weights with odd number layers: $R(\boldsymbol{W}) \; = \sum_{j=1}^{\lfloor L/2 \rfloor}\sum_i \left(\|\boldsymbol{w}_{i,2j-1}\|_2^2 + \|\boldsymbol{v}_{i,2j-1}\|_2^2\right) \; + \; c\sum_i \|\boldsymbol{v}_{i,L}\|_2^2$, where $c = 0$ if L is even and 1 if L is odd. Each weight appears only once in the expression above, which is convenient for optimization. Theorem 1 implies for any weights that minimize F_λ we have

$$R(\boldsymbol{W}) \; = 2\sum_{j=1}^{\lfloor L/2 \rfloor}\sum_{i=1}^{n_{2j-1}} \|\boldsymbol{w}_{i,2j-1}\|_2 \, \|\boldsymbol{v}_{i,2j-1}\|_2 \; + \; c\sum_{i=1}^{n_L} \|\boldsymbol{v}_{i,L}\|_2^2 \tag{2}$$

Theorem 3 *For any weights* $\boldsymbol{W}$ *let* $\widetilde{R}(\boldsymbol{W}) := \sum_{j=1}^{\lfloor L/2 \rfloor}\sum_i \|\boldsymbol{w}_{i,2j-1}\|_2 \, \|\boldsymbol{v}_{i,2j-1}\|_2 \; + \; \frac{c}{2}\sum_i \|\boldsymbol{v}_{i,L}\|_2^2$. *Then the solutions to* $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \frac{\lambda}{2}R(\boldsymbol{W})$ *and* $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \lambda\widetilde{R}(\boldsymbol{W})$ *are equivalent. Specifically, a minimizer of the first optimization is a minimizer of the second, and a minimizer of the second minimizes the first (after possibly rescaling the weights so* $\|\boldsymbol{w}_{i,2j-1}\|_2 = \|\boldsymbol{v}_{i,2j-1}\|_2$ *for each term).*

The proof is presented in Appendix A.4. The product $\|\boldsymbol{w}_{i,j}\|_2 \, \|\boldsymbol{v}_{i,j}\|_2$ is sometimes called the *path-norm* of the neuron. So we will call the objective

$$G_\lambda(\boldsymbol{W}) \; := \; L(\boldsymbol{W}) \; + \; \lambda\widetilde{R}(\boldsymbol{W})$$

the *path-norm objective*, where again $\widetilde{R}(\boldsymbol{W}) = \sum_{j=1}^{\lfloor L/2 \rfloor}\sum_i \|\boldsymbol{w}_{i,2j-1}\|_2 \, \|\boldsymbol{v}_{i,2j-1}\|_2 \; + \; \frac{c}{2}\sum_i \|\boldsymbol{v}_{i,L}\|_2^2$. Theorem 3 shows that minimizing G_λ is equivalent to minimizing the weight decay objective F_λ. Our new neural network training algorithm is designed to minimize the G_λ. The key observation is that the product terms $\|\boldsymbol{w}\|_2 \, \|\boldsymbol{v}\|_2$ are non-smooth, which means that minimizers may be sparse. In fact, as λ increases fewer and fewer terms (neurons) will be nonzero. This is remarkable, since the sparsity of solutions is not apparent from simple inspection of the original weight decay objective, as indicate in Fig. 1 (*right*). The product terms are reminiscent of *group lasso* regularization terms. Proximal gradient descent methods have been widely applied to this (group) lasso type of regularization schemes in the linear/convex cases [7, 10, 12, 40]. This motivates us to the following Algorithm 1. In the algorithnm, we project each $\boldsymbol{w}$ to the unit sphere, which is also equivalent with the proximal operator of the indicator function $\infty \cdot \mathbf{1}\{\|\boldsymbol{w}\|_2 \neq 1\} = \begin{cases} 0 & \text{if } \|\boldsymbol{w}\|_2 = 1 \\ \infty & \text{o.w.} \end{cases}$. Shown in [3], with mild assumptions on gradient noise and proper learning rate scheduler beyond T, when running nonstop, the objective value evaluated at every step of our Algorithm 1 converges. Moreover, every limit point of the iterates $\{\boldsymbol{W}^k\}_{k=1}^{\infty}$ is composite critical almost surely. The algorithm is presented as a full batch gradient step, but can be easily modified to SGD/mini-batch style algorithms.

Algorithm 1 ℓ_2-PATH-NORM Proximal Optimization of G_λ as discussed in Thm. 3

```
Input: loss functions $L$, learning rate $\gamma > 0$, weight decay parameter $\lambda > 0$, total number of iterations $T$.
for $t = 1, 2, ..., T$ do
    for $j = 1, 2, ..., \lfloor L/2 \rfloor$ do
        For each homogeneuous unit $i$ in $(2j-1, 2j)$ coupled layer:
        Update homogeneous input weights:
        update based on batch gradient: $\boldsymbol{y} \leftarrow \boldsymbol{w}_{i,2j-1}^{t-1} - \gamma \frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{w}_{i,2j-1}^{t-1}}|_{\boldsymbol{W}^{t-1}}$.
        project to have unit norm: $\boldsymbol{w}_{i,2j-1}^{t} \leftarrow \arg\min_{\|\boldsymbol{w}\|_2=1} \|\boldsymbol{w} - \boldsymbol{y}\|_2 = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2}$.
        Update homogeneous output weights:
        update based on batch gradient: $\boldsymbol{z} \leftarrow \boldsymbol{v}_{i,2j-1}^{t-1} - \gamma \frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{v}_{i,2j-1}^{t-1}}|_{\boldsymbol{W}^{t-1}}$.
        apply proximal operator: $\boldsymbol{v}_{i,2j-1}^{t} \leftarrow \text{Prox}_2(\boldsymbol{z})$ with $\text{Prox}_2(\boldsymbol{z})_i = \begin{cases} 0 & ||\boldsymbol{z}||_2 \leq \lambda \cdot \gamma \\ \boldsymbol{z}_i - \lambda \cdot \gamma \frac{\boldsymbol{z}_i}{||\boldsymbol{z}||_2} & \text{o.w.} \end{cases}$.
    end for
end for
```

As indicated in Eqn. 2, we may have weights associated to non-homogeneous units: $\{\boldsymbol{v}_{i,L}\}_{i=1}^{n_L}$. Standard weight decay will be applied to these additional weights. Algorithm 1 can also be applied to convolutional neural networks, where we treat each channel of a given convolutional layer as an 1-homogeneous unit (see details in Appendix B.1). Motivated by the homogeneity of layers and Theorem 1, we also apply a layer-wise balancing procedure (described in Appendix B.2) to ensure the total ℓ_2-PATH-NORM among each coupling of layers to be equal at every iteration.

5. Experimental Results

In this section, we will present experimental results to support the following claims: Proximal algorithm for ℓ_2-PATH-NORM provides a) faster convergence b) better generalization c) sparser solutions than weight decay.

For simplicity, we use *MLP-d-n* for d fully-connected feedforward layers, each with n neurons. *MLP-d-n factorized* indicates each layer of *MLP-d-n* is factorized into 2 linear layers, with hidden neurons to be n as well. Our proximal algorithm is evaluated on the following task: **(Task 1)** MNIST [19] subset on factorized MLP-3-400, **(Task 2)** MNIST on MLP-6-400, **(Task 3)** CIFAR10 [16] on VGG19 [34], **(Task 4)** SVHN on VGG19, **(Task 5)** MNIST on MLP-3-800. For

MNIST subset, we randomly subsample 100 images per class. Please refer to details of the experiment settings in Appendix C.1. The detail of applying Algorithm 1 to convolutional neural networks such as VGG19 is discussed in Appendix B.1.

5.1. Generalization on Test Data and Corrupted Data

Performance on the unseen dataset, or dataset sampled from other distribution, measures the generalization ability of the model. Following the experimental design in [11, 33, 41], the following modification of the data and label are investigated: i) **True labels**: the original dataset without modification. ii) **Corrupted dataset**: for MNIST, we train on an unmodified dataset, then evaluate the average accuracy across different types of corruption on the MNIST-C dataset [22]. For CIFAR10, we train on unmodified dataset, then evaluate the average accuracy across different types of corruption on the CIFAR10-C dataset [13]. iii) **Partially corrupted labels**: the label of training image is corrupted to be uniform random class with probability p. We train on this modified dataset, and report the result on unmodified dataset.

In this section, we present the result of **(Task 1)** on modification i, ii, and iii; result of **(Task 2)** on modification i and ii; result of **(Task 3)** on modification i and ii; and result of **(Task 4)** on modification i in Table 1. For each experiment, we did a grid search on the hyper-parameter choice of λ and *learning rate*, and pick the best set of parameter based on the validation accuracy. The details of hyper-parameter search is in Appendix C.1. Notice that although we did not explicitly prune the model, our proposed $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ proximal algorithm inherently enforce the sparse structure: in the Table 1, we calculate the structural sparsity of the model, namely the percentage of the active units in the grouped layers, and the result confirm our proximal method can naturally prune the model to have some level of sparsity. We also investigate the efficiency of finding the structural sparse solution when explicitly enforce the model to prune, and present the result in Appendix C.3.

Table 1: Generalization result for different modification on weight decay, proximal algorithm for $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$. Numbers are highlighted if the gap between weight decay and proximal algorithm is at least the sum of both standard errors. Structural sparsity of the model is in the parentheses.

Task	Modification	Weight Decay	$\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ Proximal
1	True labels	90.87 ± 0.1 (100%)	$\mathbf{91.46 \pm 0.11}$ (99.92 ± 0.14%)
	Corrupted data	64.86 ± 0.37 (100%)	65.52 ± 0.65 (99.92 ± 0.14%)
	Corrupted labels ($p = 0.3$)	81.125 ± 0.44 (100%)	81.87 ± 0.42 (100%)
	Corrupted labels ($p = 0.7$)	47.82 ± 1.57 (100%)	$\mathbf{55.27 \pm 0.95}$ (100%)
2	True labels	98.29 ± 0.03 (100%)	98.21 ± 0.07 (98.3 ± 0.78%)
	Corrupted data	71.54 ± 0.22 (100%)	$\mathbf{72.65 \pm 0.44}$ (99.95 ± 0.04%)
3	True labels	90.41 ± 0.1 (100%)	$\mathbf{90.79 \pm 0.06}$ (48.29 ± 10.68%)
	Corrupted data	60.78 ± 0.17 (100%)	60.71 ± 0.26 (60.34 ± 7.69 %)
4	True labels	94.68 ± 0.12 (100%)	$\mathbf{95.51 \pm 0.12}$ (44.02 ± 14.03%)

6. Conclusion and Future Work

This work shows that proximal gradient algorithms may offer advantages in neural network training compared to standard weight decay. There are several directions for possible future work, including investigating alternative formulations of the proximal gradient method that treat all homogeneous units in the same manner (rather than grouping weights into disjoint sets) and adaptive learning rate procedures like those used in other proximal gradient methods.

References

[1] Peter Bartlett. For valid generalization the size of the weights is more important than the size of the network. *Advances in neural information processing systems*, 9, 1996.

[2] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2:183–202, 2009.

[3] Damek Davis, Dmitriy Drusvyatskiy, Sham Kakade, and Jason D Lee. Stochastic subgradient method converges on tame functions. *Foundations of computational mathematics*, 20(1): 119–154, 2020.

[4] Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M. Roy. In search of robust measures of generalization. *ArXiv*, abs/2010.11924, 2020.

[5] Tolga Ergen and Mert Pilanci. Path regularization: A convexity and sparsity inducing regularization for parallel relu networks. *ArXiv*, abs/2110.09548, 2021.

[6] Tolga Ergen and Mert Pilanci. Convex geometry and duality of over-parameterized neural networks. *Journal of machine learning research*, 2021.

[7] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[8] Angus Galloway, Thomas Tanay, and Graham W Taylor. Adversarial training versus weight decay. *arXiv preprint arXiv:1804.03308*, 2018.

[9] Yves Grandvalet. Least absolute shrinkage is equivalent to quadratic penalization. In *International Conference on Artificial Neural Networks*, pages 201–206. Springer, 1998.

[10] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

[11] Hrayr Harutyunyan, Kyle Reing, Greg Ver Steeg, and Aram Galstyan. Improving generalization by controlling label-noise information in neural network weights. In *International Conference on Machine Learning*, pages 4071–4081. PMLR, 2020.

[12] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. Statistical learning with sparsity. *Monographs on statistics and applied probability*, 143:143, 2015.

[13] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *ArXiv*, abs/1903.12261, 2019.

[14] Arthur Jacot, Eugene Golikov, Clément Hongler, and Franck Gabriel. Feature learning in l2-regularized dnns: Attraction/repulsion and sparsity. *ArXiv*, abs/2205.15809, 2022.

[15] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *ArXiv*, abs/1912.02178, 2020.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

[18] Fabian Latorre, Paul Rolland, Nadav Hallak, and Volkan Cevher. Efficient proximal mapping of the 1-path-norm of shallow networks. *ArXiv*, abs/2007.01003, 2020.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[21] Ziquan Liu, Yufei Cui, and Antoni B Chan. Improve generalization and robustness of neural networks via weight scale shifting invariant regularizations. *arXiv preprint arXiv:2008.02965*, 2020.

[22] Norman Mu and Justin Gilmer. Mnist-c: A robustness benchmark for computer vision. *ArXiv*, abs/1906.02337, 2019.

[23] Yuval Netzer, Tao Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[24] Behnam Neyshabur, Russ R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. *Advances in neural information processing systems*, 28, 2015.

[25] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *COLT*, 2015.

[26] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations (Workshop)*, 2015.

[27] Greg Ongie, Rebecca Willett, Daniel Soudry, and Nathan Srebro. A function space view of bounded norm infinite width relu nets: The multivariate case. In *International Conference on Learning Representations*, 2019.

[28] Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of tricks for adversarial training. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Xb8xvrtB8Ce`.

[29] Rahul Parhi and Robert D. Nowak. The role of neural network activation functions. *IEEE Signal Processing Letters*, 27:1779–1783, 2020.

[30] Rahul Parhi and Robert D Nowak. Banach space representer theorems for neural networks and ridge splines. *J. Mach. Learn. Res.*, 22(43):1–40, 2021.

[31] Rahul Parhi and Robert D. Nowak. What kinds of functions do deep neural networks learn? Insights from variational spline theory. *SIAM Journal on Mathematics of Data Science*, 4(2): 464–489, 2022. doi: 10.1137/21M1418642.

[32] Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In *Conference on Learning Theory*, pages 2667–2690. PMLR, 2019.

[33] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*, pages 5739–5748. PMLR, 2019.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[35] Pierre Stock, Benjamin Graham, Rémi Gribonval, and Hervé Jégou. Equi-normalization of neural networks. *arXiv preprint arXiv:1902.10416*, 2019.

[36] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the royal statistical society series b-methodological*, 58:267–288, 1996.

[37] Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. *ArXiv*, abs/2103.03936, 2021.

[38] Yifei Wang, Tolga Ergen, and Mert Pilanci. Parallel deep neural networks have zero duality gap. *ArXiv*, abs/2110.06482, 2021.

[39] Stephen J. Wright, Robert D. Nowak, and Mário A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57:2479–2493, 2008.

[40] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[41] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ArXiv*, abs/1611.03530, 2017.

Appendix Content

Appendix A. Supplementary of Theory

A.1. Discussion on the Theorem 1

At minimum sum of squared weights, Theorem 1 state that each homogeneous unit in the function f have equivalent ℓ_2 norm for the input and output weights of the unit:

$$\|\boldsymbol{w}\|_2 = \|\boldsymbol{v}\|_2$$

Note that the balancing constraint is a necessary, but not a sufficient condition for the minimum sum of squared weights representation. In details, there are different equivalence of function:

Definition 4 (Rescaling Equivalence) *Function f with weights $\boldsymbol{W}_f$ and g with weights $\boldsymbol{W}_g$ are rescaling equivalent if $\boldsymbol{W}_f = s(\boldsymbol{W}_g)$ where $s(\cdot)$ is the scaling operator on the homogeuous units in weights $\boldsymbol{W}$.*

This rescaling equivalence is the main force we leverage for finding the effectively equivalent norm.

Definition 5 (Functional Equivalence) *Function f with weights $\boldsymbol{W}_f$ and g with weights $\boldsymbol{W}_g$ are functional equivalent if $\forall \boldsymbol{x}, f(\boldsymbol{x}) = g(\boldsymbol{x})$.*

It is clear from the definition that rescaling equivalence implies functional equivalence, but the reverse is not true. With this definition, we will properly state the relation of balancing constraint and minimum representation as follows (also discussed in [35]):

Remark 6 *For function f with weights $\boldsymbol{W}$, the minimum norm up to rescaling equivalence may not be the minimum representation up to functional equivalence. So the minimum sum of squared weights representation implies balancing constraint, however balancing constraint doesn't necessarily imply minimum norm representation.*

We are not aware of the algorithm to find the minimum sum of squared weights representation. However up to rescaling equivalence, [35] proposed an iterative algorithm to minimize the sum of squared weights, which requires C cycles to converge. The number of cycles after each SGD step is an hyperparameter, by default they set to 1. If set to C, which means constraint the norm to be balanced for every homogeneous units after each SGD step, from our experiment findings, the algorithm converge to worse solution compared to without this constraint.

A.2. Extension of Theorem 1 to Convolutional Layers

Let $\circledast$ denote the sliding-window convolutional operator, where for any two-dimensional tensors X, Y and Z, we have $Z_{i,j} = (X \circledast Y)_{i,j} = \sum_{a\in[L_1],b\in[L_2]} X_{a,b}Y_{i+a,j+b}$ with L_1 and L_2 denoting the width and height of X. A convolutional homogeneous unit then takes the following form with one hidden channel

$$\mu(\boldsymbol{x}) = \left[\boldsymbol{v}_j \circledast \sigma\left(\sum_{i\in[C_1]} \boldsymbol{w}_i \circledast \boldsymbol{x}_i\right)\right]_{j\in[C_2]}$$

where $\boldsymbol{x}$, $\boldsymbol{w}$ and $\boldsymbol{v}$ are three-dimensional tensors and with slight abuse of notation $\sigma(\cdot)$ is applied element-wise. x takes the *channel first* notation, where the first dimension indexes number of channels while the last two dimensions indexes width and height. Both $\boldsymbol{x}$ and $\boldsymbol{w}$ have channel size C_1 while $\boldsymbol{v}$ has channel size C_2. Below, we restate the equivalence from Theorem 1 for convolutional units:

Theorem 7 *(Convolutional Neural Network Balance Theorem) Let f be a function represented by a neural network and consider a representation of f with the minimum sum of squared weights. Furthermore, let $vec(\cdot)$ denote the vectorize operator of a tensor. Then the weights satisfy the following* balancing constraints. *Let $\boldsymbol{w}$ and $\boldsymbol{v}$ denote the input and output weights of any convolutional homogeneous unit $\mu(\boldsymbol{x}) = [\boldsymbol{v}_j \circledast \sigma(\sum_{i\in[C_1]} \boldsymbol{w}_i \circledast \boldsymbol{x}_i)]_{j\in[C_2]}$ in this representation. Then $\|vec(\boldsymbol{w})\|_2 = \|vec(\boldsymbol{v})\|_2$.*

We also note that one can take a channel-wise homogeneous pooling operation on the hidden channel, i.e., a homogeneous unit of the form $\mu(\boldsymbol{x}) = [\boldsymbol{v}_j \circledast \mathcal{P}(\sigma(\sum_{i\in[C_1]} \boldsymbol{w}_i \circledast \boldsymbol{x}_i))]_{j\in[C_2]}$, where $\mathcal{P}$ is some homogeneous pooling function such as max pooling or average pooling. Due to the homogeneity of the pooling layer, the above results still hold for these homogeneous units with pooling layers. This suggests we can apply the argument across blocks in a convolutional network.

A.3. Proof for Lipschitz constant bound

As indicated in Sec. 3, at the global minimum of F_λ, we can write the weight decay objective as Eqn. (1). This shows that increasing the weight decay parameter λ penalizes the average of the norm-products $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$. This observation provides insight into the effects of weight decay regularization: Let $\eta(\boldsymbol{x}) := \boldsymbol{v}\,\sigma(\boldsymbol{w}^T\boldsymbol{x})$ be a homogeneous neuron with input weights $\boldsymbol{w} \in \mathbb{R}^p$ and output weights $\boldsymbol{v} \in \mathbb{R}^q$, and assume that σ is 1-Lipschitz. Hölder's inequality implies that for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^m$,

$$\|\eta(\boldsymbol{x}) - \eta(\boldsymbol{x}')\|_2 \;\le\; \|\boldsymbol{w}\|_2 \, \|\boldsymbol{v}\|_2 \, \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \, .$$

which shows $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ is a bound on the Lipschitz constant of η. Thus, weight decay regularization encourages solutions in which the individual unit functions have small Lipschitz constants on average, a property known to be related to generalization and robustness.

A.4. Proof for Theorem 3

Restate the Theorem 3 Here.

Theorem *For any weights $\boldsymbol{W}$ let $\widetilde{R}(\boldsymbol{W}) := \sum_{j=1}^{\lfloor L/2 \rfloor} \sum_i \|\boldsymbol{w}_{i,2j-1}\|_2 \, \|\boldsymbol{v}_{i,2j-1}\|_2 \; + \; \frac{c}{2} \sum_i \|\boldsymbol{v}_{i,L}\|_2^2$. Then the solutions to $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \frac{\lambda}{2} R(\boldsymbol{W})$ and $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \lambda \widetilde{R}(\boldsymbol{W})$ are equivalent. Specifically, a minimizer of the first optimization is a minimizer of the second, and a minimizer of the second minimizes the first (after possibly rescaling the weights so $\|\boldsymbol{w}_{i,2j-1}\|_2 = \|\boldsymbol{v}_{i,2j-1}\|_2$ for each term).*

Proof Suppose that $\widehat{\boldsymbol{W}}$ is a solution to $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \lambda \widetilde{R}(\boldsymbol{W})$, but there exists a $\boldsymbol{W}$ such that

$$L(\widehat{\boldsymbol{W}}) + \frac{\lambda}{2} R(\widehat{\boldsymbol{W}}) > L(\boldsymbol{W}) + \frac{\lambda}{2} R(\boldsymbol{W}) \, .$$

Theorem 1 shows that $R(\boldsymbol{W}) = 2\widetilde{R}(\boldsymbol{W})$, which contradicts the claim that $\widehat{\boldsymbol{W}}$ is a solution to $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \lambda \widetilde{R}(\boldsymbol{W})$. Next let $\boldsymbol{W}$ be a solution to $\min_{\boldsymbol{W}} L(\boldsymbol{W}) + \frac{\lambda}{2} R(\boldsymbol{W})$, but suppose it does not minimize $L(\boldsymbol{W}) + \lambda \widetilde{R}(\boldsymbol{W})$. Then there exists a $\widehat{\boldsymbol{W}}$ such that

$$L(\boldsymbol{W}) + \lambda \widetilde{R}(\boldsymbol{W}) > L(\widehat{\boldsymbol{W}}) + \lambda \widetilde{R}(\widehat{\boldsymbol{W}}) \, .$$

If necessary, rescale the weights $\widehat{\boldsymbol{W}}$ so that $\|\widehat{\boldsymbol{w}}_{i,2j-1}\|_2 = \|\widehat{\boldsymbol{v}}_{i,2j-1}\|_2$ for each term in $\widetilde{R}(\boldsymbol{W})$; this does not affect the value of $L(\widehat{\boldsymbol{W}})$. Then $R(\widehat{\boldsymbol{W}})/2 = \widetilde{R}(\widehat{\boldsymbol{W}})$, which shows that $L(\boldsymbol{W}) + \frac{\lambda}{2} R(\boldsymbol{W}) > L(\widehat{\boldsymbol{W}}) + \frac{\lambda}{2} R(\widehat{\boldsymbol{W}})$, contradicting the assumption that $\boldsymbol{W}$ is a solution to the first optimization. ■

Appendix B. Supplementary of Algorithm

B.1. Homogeneous Units are Everywhere

In the paper we mainly discuss the theorem and algorithm related to the multi-layer perceptron, but the identification of homogeneous units is not limited to multi-layer perceptron. In this section, we formally identify the homogeneous units in common neural network architectures: multi-layer perceptron (MLP) and convolutional neural network (CNN).

Multi-layer Perceptron (MLP). A multi-layer perceptron $f_{\text{MLP}}(x; W) : \mathcal{X} \to \mathcal{Y}$ with M linear layers takes the following recursive parameterization

$$f_{\text{MLP}}(x; W) = W^L \begin{bmatrix} 1 \\ h^L \end{bmatrix}$$

$$h^{k+1} = \sigma\left(W^k \begin{bmatrix} 1 \\ h^k \end{bmatrix} \right), \;\; \forall k \in [L-1] \quad \text{and} \quad h^1 = x.$$

Here, the linear layers are parameterized by weights $W = \{W^k \in \mathbb{R}^{n_{k+1}\times n_k}\}_{k=1}^L$, where n_k is the dimension of the $(k-1)$-th hidden layer, n_1 is the input dimension, and n_{L+1} is the output dimension. The activation function $\sigma(x) = \max\{0, x\}$ is applied element-wise.

Let $\widetilde{W}$ denote the matrix of W with its first column removed, W_i and $W_{:,i}$ denote the i-th row and column of W, respectively. Consider for every two layers, we have

$$\widetilde{W}^{k+1}\sigma\left(W^k\begin{bmatrix}1\\h^k\end{bmatrix}\right) = \sum_{i=1}^{n_k}\widetilde{W}^{k+1}_{:,i}\sigma\left(W^k_i\begin{bmatrix}1\\h^k\end{bmatrix}\right)$$

as part of the computation for h^{k+2}. Therefore, for every two consecutive layers with weights W^{2j-1} and $\widetilde{W}^{2j}$ where $j \in \left[\lfloor\frac{L}{2}\rfloor\right]$, we identify n_{2j-1} homogeneous units — one for each hidden neuron. In our experiments, for regular multi-layer perceptron, we use exactly this coupling scheme, where every two layers are combined and viewed as n_{2j-1} homogeneous units.

Factorized MLP. Multi-layer perceptron can be equivalent factorized as shown in [37]. The linear layers are parameterized by weights $W = \{W^k \in \mathbb{R}^{n_{k+1}\times n_k}\}_{k=1}^L$, where n_k is the dimension of the $(k-1)$-th hidden layer. For each W^k, we can further factorize it into $W^k = Q^kP^k$ for $k \in \{2, 3, \cdots, L-1\}$, where $P^k \in \mathbb{R}^{(n_{k+1}-1)\times n_k}$, $Q^k \in \mathbb{R}^{n_{k+1}\times(n_{k+1}-1)}$, and

$$Q^1 = W^1 \quad P^L = W^L$$

Here the bias term is not required for the factorization. Now let $g^k = P^k\begin{bmatrix}1\\h^k\end{bmatrix} \in \mathbb{R}^{n_{k+1}}$, and let $\widetilde{P}$ denote the matrix of P with its first column removed, then $\widetilde{W} = Q\widetilde{P}$ has the first column removed. We have part of the computation for h^{k+2} to be:

$$Q^{k+1}\widetilde{P}^{k+1}\sigma\left(Q^kP^k\begin{bmatrix}1\\h^k\end{bmatrix}\right) = Q^{k+1}g^{k+1}$$

and

$$g^{k+1} = \widetilde{P}^{k+1}\sigma\left(Q^kg^k\right) = \sum_{i=1}^{n_{k+1}-1}\widetilde{P}^{k+1}_{:,i}\sigma(Q^k_ig^k)$$

where Q_i and $P_{:,i}$ denote the i-th row and column of Q and P, respectively. Therefore, for every consecutive layers with factorized weights Q^k and P^{k+1}, we identify $n_{k+1}-1$ homogeneous units — one for each hidden neuron.

Note that for fixed dimensions and number of layers, the class of factorized MLP is equivalent with the class of the original MLPs with only activated layers. Furthermore, with a factorized MLP, we bypass the issue of having to group even number of layers. Instead, we are able to optimize an equivalent class of function and identify a homogeneous unit for each hidden neuron.

Convolutional neural networks. A 2D convolutional *backbone network* is a sequence of function composition of convolutional and pooling layers that maps 3D tensors to 3D tensors. The output of a backbone network is usually then flattened and passed through an MLP. We focus on an L layers convolutional backbone network here, which takes the following recursive parameterization

$$f_{\text{CNN}}(x;W) = \left[\left(\sum_j W^L_{i,j} \circledast h^L_j\right) \oplus b^L_i\right]_{i\in[n_{L+1}]}$$

$$h^{k+1} = \left[\mathcal{P}^k\left(\sigma\left(\left(\sum_j W^k_{i,j} \circledast h^k_j\right) \oplus b^k_i\right)\right)\right]_{i\in[n_{k+1}]}, \ \forall k \in [L-1] \quad \text{and} \quad h^1 = x.$$

Here n_k denotes the number of hidden/output channels and $W = \{(W^k, B^k)\}_{k=1}^L$ are the weights parameterizing the neural network. Each layer weight W^k is a four-dimensional tensor of dimensions $n_{k+1} \times n_k \times l \times l'$ and $b^k \in \mathbb{R}^{n_{L+1}}$. $\oplus$ is an element-wise addition operator which adds the later scalar argument to the former tensor. $\mathcal{P}^k$ is a channel-wise pooling layer such as average pooling and max pooling, or the identity function. Each hidden layer h^k is then a three-dimensional tensor of dimensions $n_k \times l \times l'$. By substituting in one more recursive step and for any homogeneous activation function σ, we get

$$\begin{aligned} h^{k+1} &= \left[\mathcal{P}^k\left(\sigma\left(\left(\sum_j W^k_{i,j} \circledast \mathcal{P}^{k-1}\left(\sigma\left(\left(\sum_{j'} W^{k-1}_{j,j'} \circledast h^{k-1}_{j'}\right) \oplus b^k_j\right)\right)\right) \oplus b^k_i\right)\right)\right]_{i\in[n_{k+1}]} \\ &= \mathcal{P}^k\left(\sigma\left(\left[\left(\sum_j W^k_{i,j} \circledast \mathcal{P}^{k-1}\left(\sigma\left(\left(\sum_{j'} W^{k-1}_{j,j'} \circledast h^{j-1}_{j'}\right) \oplus b^k_j\right)\right)\right) \oplus b^k_i\right]_{i\in[n_{k+1}]}\right)\right). \end{aligned}$$

Therefore, for each $j \in [n_k]$, we have $\left[W^k_{i,j} \circledast \mathcal{P}^{k-1}\left(\sigma\left(\left(\sum_{j'} W^{k-1}_{j,j'} \circledast h^{k-1}_{j'}\right) \oplus b^{k-1}_j\right)\right)\right]_{i\in[n_{k+1}]}$ as part of the computation. With the one-homogeneity of the channel-wise pooling layer $\mathcal{P}^{k-1}$, we can get an equivalent function by scaling $\left(\alpha W^{k-1}_{j,j'}, \alpha b^{k-1}_j\right)$ and $\frac{1}{\alpha} W^k_{i,j}$, we can therefore obtain a homogeneous unit for every hidden channel j of the k-th layer h^k.

Since the channel-wise pooling layer can be either identity, or max pooling, or average pooling layer, we can group the convolutional layers across max/average pooling layers. Therefore, for every two consecutive convolutional layer (possibly across the pooling layer), with weights $\left(W^{2k-1}_{j,j'}, b^{2k-1}_j\right)$ and $W^{2k}_{i,j}$, where $k \in \{1, 2, \cdots, \lfloor \frac{L}{2} \rfloor\}$, we identify n_{2k-1} homogeneous units, one for each channel.

B.2. Layer-wise Balancing Procedure

In this section, we present a layer-wise balancing procedure that enforces balancing constraints across layers. This is motivated by the fact that the sums of ℓ_2-PATH-NORM of every two consecutive layers should be equal at the minimum norm solution. Below, we will first formally show this observation as a corollary of Theorem 1 and then present the specific layer-wise balancing algorithm.

B.2.1. COROLLARY OF THEOREM 1: LAYER-WISE BALANCING

As indicated in Theorem 1, for the minimum sum of squared weight representation, the ℓ_2 norm of the input vector and output vector of the unit are the same. Thus for the coupling of $(j-1, j)$-th layer, we have

$$\sum_{i=1}^{n_{j-1}} \|\boldsymbol{w}_{i,j-1}\|_2^2 = \sum_{i=1}^{n_{j-1}} \|\boldsymbol{v}_{i,j-1}\|_2^2$$

and for the coupling of $(j, j+1)$-th layer, we have

$$\sum_{i=1}^{n_j} \|\boldsymbol{w}_{i,j}\|_2^2 = \sum_{i=1}^{n_j} \|\boldsymbol{v}_{i,j+1}\|_2^2$$

which indicate the $j-1$, j and $j+1$-th layer have the same amount of sum of squared weights. Since j is arbitrary, it is easily verified that at the minimum norm representation, each layer share the same amount of sum of squared weights. Now consider the $(j, j+1)$-th and $(k, k+1)$-th coupling layer, we have

$$\frac{1}{2}\sum_{i=1}^{n_j} \|\boldsymbol{w}_{i,j}\|_2^2 + \|\boldsymbol{v}_{i,j+1}\|_2^2 = \frac{1}{2}\sum_{i=1}^{n_k} \|\boldsymbol{w}_{i,k}\|_2^2 + \|\boldsymbol{v}_{i,k+1}\|_2^2$$

Again, as in Theorem 1 indicate, at minimum norm representation, we have $\|\boldsymbol{w}\|_2 = \|\boldsymbol{v}\|_2$, therefore

$$\sum_{i=1}^{n_j} \|\boldsymbol{w}_{i,j}\|_2 \, \|\boldsymbol{v}_{i,j+1}\|_2 = \sum_{i=1}^{n_k} \|\boldsymbol{w}_{i,k}\|_2 \, \|\boldsymbol{v}_{i,k+1}\|_2$$

So the sum of the ℓ_2-PATH-NORM per coupling of layers are the same for the minimum norm solution. Our proposed proximal algorithm doesn't naturally enforce this, so we will apply the layer-wise balance procedure along the proximal algorithm, as indicated in the next section.

B.2.2. LAYER-WISE BALANCE ALGORITHM

In Algorithm 1, we consider bias term as part of the weight parameter, and thus calculate ℓ_2-PATH-NORM with the bias term added. However to apply layer-wise balance, we will consider only regularizing the weight parameter, and leave the bias term for standard weight decaying.

Let $\widetilde{W}$ denote the matrix of W with its first column removed, W_i and $W_{:,i}$ denote the i-th row and column of W, respectively. For every group of layers $j \in \{1, 2, \cdots, \lfloor \frac{L}{2} \rfloor\}$, we will obtain equality in the following term

$$\sum_{i=1}^{n_{2j-1}} \left\|\widetilde{W}_i^{2j-1}\right\|_2 \left\|\widetilde{W}_{:,i}^{2j}\right\|_2$$

through the *layer-wise balance* procedure as indicated in Algorithm 2, and make sure the following function is equivalent after adjustment:

$$\widetilde{W}^{2j+2}\sigma\left(\widetilde{W}^{2j+1}\sigma\left(\widetilde{W}^{2j}\sigma\left(\widetilde{W}^{2j-1}h^{2j-1} + b^{2j-1}\right) + b^{2j}\right) + b^{2j+1}\right)$$

where $h^{2j-1} \in \mathbb{R}^{n_{2j-1}}$ is the input to the $(2j-1)$-th layer, $b^{2j-1} = W_{:,1}^{2j-1}$, $b^{2j} = W_{:,1}^{2j}$ and $b^{2j+1} = W_{:,1}^{2j+1}$ are the bias terms for $(2j-1)$-th, $2j$-th and $(2j+1)$-th layer respectively. In practice, this *layer-wise balance* is processed after doing batch gradient update on the homogeneous output weights, and before applying proximal operator.

For CNN backbone network, similar layer-wise balance procedure is applied on every group of the layers with proper defined ℓ_2-PATH-NORM norm as indicated in Sec. A.2. Since the convolution operator is piece-wise linear in weights, the layer-wise balancing procedure still holds for adjusting the 4D weight parameter in CNN.

Algorithm 2 Layer-wise balance for ℓ_2-PATH-NORM Proximal Optimization of (2) on each iteration

```
**Input:** The homogeneous output weights updated based on batch gradient $\widetilde{W}^{2j}$ for $j = 1, 2, \cdots, \lfloor \frac{L}{2} \rfloor$, and bias term $\{b_1, b_2, \cdots, b_L\}$
**Note:** After the proximal step for homogeneous input weights, we always have $\left\| \widetilde{W}_i^{2j-1} \right\|_2 = 1$, thus this term is omitted in the calculation of $\ell_2$-PATH-NORM norm below.
Calculated the geometric mean of the $\ell_2$-PATH-NORM norm of each layer:
    $$X = \sqrt[\lfloor \frac{L}{2} \rfloor]{\prod_{j=1}^{\lfloor \frac{L}{2} \rfloor} \left( \sum_{i=1}^{n_{2j-1}} \left\| \widetilde{W}_{:,i}^{2j} \right\|_2 \right)}$$
**for** $j = 1, 2, \cdots, \lfloor \frac{L}{2} \rfloor$ **do**
    Calculate the scale $\alpha$ applied to layer $2j$: $\alpha_j = X / \left\| \widetilde{W}^{2j} \right\|_2$
    Update the output weight: $\widetilde{W}^{2j} \leftarrow \alpha_j \cdot \widetilde{W}^{2j}$, where $\cdot$ is applied in an element-wise manner.
    For the following block of function:
        $$\widetilde{W}^{2j+2} \sigma \left( \widetilde{W}^{2j+1} \sigma \left( \underbrace{\boldsymbol{W^{2j}} \sigma \left( \widetilde{W}^{2j-1} h^{2j-1} + b^{2j-1} \right)}_{\times \alpha_j} + b^{2j} \right) + b^{2j+1} \right)$$
    Adjust other parameter to maintain the equivalence of the neural network function:
        $$b^{2j} \leftarrow \alpha_j \cdot b^{2j}$$
        $$b^{2j+1} \leftarrow \alpha_j \cdot b^{2j+1}$$
        $$\widetilde{W}^{2j+2} \leftarrow \widetilde{W}^{2j+2} / \alpha_j$$
        $$\alpha_{j+1} \leftarrow \alpha_{j+1} / \alpha_j$$
**end for**
```

Appendix C. Supplementary of Experiments

C.1. Experiment Setup

In this section we will describe the dataset and network we use in our experiments. All our experiments are conducted on Nvidia 3090 GPUs.

C.1.1. DATASETS

In this work, we demonstrate the performance of the proposed proximal algorithm for ℓ_2-PATH-NORM NORM on the following dataset:

a) **MNIST** [19] consist of 10 classes of hand-written digits, each class has 6000 training images and 1000 test images. When training, we randomly split the dataset into 55000 training data, and 5000 validation data. We use the validation data to decide the hyper-parameters. Each image has shape 28×28. When training, we normalize the input data.

b) **MNIST-C** [22] is a robustness benchmark on MNIST dataset, which applies 15 standard corruption to the MNIST dataset, namely 1) shot noise, 2) impluse noise, 3) glass blur, 4) motion blur, 5) shear, 6) scale, 7) rotate, 8) brightness, 9) translate, 10) stripe, 11) fog, 12) spatter, 13) dotted line, 14) zigzag, and 15) canny edge. When validate our result on MNIST-C dataset, we train on clean MNIST dataset, and pick the best model based on clean validation set, then test its performance on the corrupted MNIST-C dataset to measure the generalization.

c) **CIFAR10** [16] has 10 classes of real images. Each class has 5000 training images and 1000 test images. When training, we randomly split 45000 images for training, and 5000 images for validation. Each image has shape 32×32. When training, we random crop, random horizontal flip, and normalize the input data.

d) **CIFAR10-C** [13] is a robustness benchmark on CIFAR10 dataset, which applies 18 standard corruption to the CIFAR10 dataset, namely 1) Gaussian noise, 2) shot noise, 3) impluse noise, 4) defocus blur, 5) frosted glass blur, 6) motion blur, 7) zoom blur, 8) snow, 9) frost, 10) fog, 11) brightness, 12) contrast, 13) elastic, 14) pixelate, and 15) JPEG. When validate our result on CIFAR10-C dataset, we train on clean CIFAR10 dataset, and pick the best model based on clean validation set, then test its performnce on the corrupted CIFAR10-C dataset.

e) **SVHN** [23] is the Street View House Numbers dataset, with 73257 digits for training, 26032 digits for testing. We randomly split 67257 images for training, and 6000 images for validation. Each image has shape 32×32. When training, we normalize the input data. We didn't use the additional dataset to boost the performance.

C.1.2. MODELS

In this work, we demonstrate the performance of the proposed proximal algorithm for ℓ_2-PATH-NORM NORM on the following models:

a) **MLP-d-n** model consist of d fully connected layers, each with n neurons. Details of the *MLP-d-n* architecture is shown in Table 2.

b) **MLP-d-n factorized** model consist of d fully connected layers, each with n neurons. For each layer, we factorize it into two linear layers, with hidden neurons to be n as well. Details of the *MLP-d-n factorized* architecture is shown in Table 2.

c) **VGG19** is introduced in [34], which is widely used for computer vision task. Instead of 3 fully-connected layer as the classifier, to apply VGG19 on CIFAR10, we use 1 fully-connected layer instead[1]. There are 16 convolutional layers. After every two or four convolutional layers, there follows a max-pooling layer to reduce the feature map size by half.

In Sec. 5.1, we demonstrate the generalization result of weight decay and proximal algorithm for $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ on:

(Task 1) MNIST subset on MLP-3-400 factorized

(Task 2) MNIST on MLP-6-400

(Task 3) CIFAR10 on VGG19

(Task 4) SVHN on VGG19

In Sec. C.3, we evaluate the ability to obtain sparse solution of weight decay, lasso and group lasso on

(Task 5) MNIST on MLP-3-800

When evaluating the proximal algorithm for $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$, we need to couple the layers into groups. We choose to factorize it, and thus evaluate the algorithm on:

(Task 5) MNIST on MLP-3-800 factorized

Table 2: The MLP architecture used in the experiments. For MLP-3-400, MLP-6-400, and MLP-3-800 factorized, we group each coupling layer together. For MLP-3-800, we refer to each layer as a group.

Parameter	MLP-3-400 factorized	MLP-6-400	MLP-3-800	MLP-3-800 factorized
Group 1	784×400 ReLU 400×400	784×400 ReLU 400×400 ReLU	784×800 ReLU	784×800 ReLU 800×800
Group 2	400×400 ReLU 400×400	400×400 ReLU 400×400 ReLU	800×800 ReLU	800×800 ReLU 800×800
Group 3	400×400 ReLU 400×10	400×400 ReLU 400×10	800×800 ReLU	800×800 ReLU 800×10
Group 4			800×10	

1. Code adapted from `https://github.com/kuangliu/pytorch-cifar`

C.1.3. HYPER PARAMETER CHOICE

For generalization experiments in Sec. 5.1, we did grid search for the learning rate as well as the λ as follows:

(Task 1) & (Task 2):

- *learning rate*: 0.003, 0.01, 0.03, 0.1, 0.3, 0.5
- λ: 0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003, 0.01

(Task 3):

- *learning rate*: 0.01, 0.03, 0.1, 0.3
- λ: 0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003, 0.01

(Task 4):

- *learning rate*: 0.03, 0.1, 0.3, 0.5
- λ: 0.0001, 0.0003, 0.001, 0.003

(Task 5):

- *learning rate*: 0.01, 0.03
- λ: 0.0001, 0.001, 0.003, 0.01

C.1.4. STANDARD ERROR CALCULATION

For each task, we run the grid search experiments on random seed 42, and pick the set of hyper parameter (*learning rate**, λ^*) based on the best validation accuracy. Then for each task, we run with (*learning rate**, λ^*) for another three times, with random seed 43, 44, and 45. Again in each run, the test accuracy is picked based on the best validation accuracy. With four runs, we evaluate the mean and standard error of the experiments, and present the result in the table.

C.2. Experiments Details for Figure 1 and 2

Left figure in Fig. 1. For the convergence figure on the left, we evaluate on the **(Task 1)** with *learning rate* $= 0.01$, and $\lambda = 0.0001$. With same λ and *learning rate*, the proximal algorithm for $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ finds better solution faster, and minimize the weight decay objective faster.

Middle figure in Fig. 1. For the histogram in the middle, we evaluate on the **(Task 2)**. The models we investigate have the best validation accuracy with *learning rate*$= 0.1$ and $\lambda = 0.00003$. To measure the local Lipschitz constant for the unseen data, we compare the spectral norm of the Jacobian (of the model with respect to the input) on all 10,000 MNIST test samples, and plot their values in the histogram shown. We see that for the model trained with weight decay the spectral norm of the Jacobian is generally larger than the model trained with our approach, indicating that the proximal $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ regularization leads to models with a lower local Lipschitz constant and therefore more robust.

Right figure in Fig. 1. For the sparsity figure on the right, we do experiment on a synthetic dataset with spatial variants. To generate the dataset, we sample in uniform randomness 64 data point from the following function:

$$f(x) = \begin{cases} \sin(8\pi x) & 0 < x \leq \frac{1}{2} \\ \sin\left(32\pi\left(x - \frac{1}{2}\right)\right) & \frac{1}{2} < x \leq \frac{3}{4} \\ \sin\left(8\pi\left(x - \frac{3}{4}\right)\right) & \frac{3}{4} < x \leq 1 \end{cases}$$

which has variant spatial frequency. The model used to fit the data is an one-hidden layer fully-feedforward network, with 640 neurons, input dimension 1, and output dimension 1. We train with weight decay, as well as $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ proximal algorithm.

Consider i-th neuron with $w_i \in \mathbb{R}$ input and $v_i \in \mathbb{R}$ output, we count an neuron to be active if $|w_i||v_i| > 0.001$. With this, we obtain the number of active neurons during the training time, and plot the figure as shown. From the plot, we can see weight decay can also sparsify the model, though with much slower speed compared to the $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ proximal algorithm.

Fig. 2. The decision boundary figure is generated on the training task of fitting a 1-hidden-layer shallow network to the data points presented in the figure. Fix the weight decay parameter λ to be 0.0001 and *learning rate* to be 0.1, we run both algorithm for same amount of iterations (till converge).

The choice of learning rate and weight decay parameter is arbitrary. In general, we found that with same weight decay parameter (the same objective), larger learning rate will see the model getting over-thresholding (for both algorithm), and thus turning sparse. The sparse structure may degrade the performance. Many approaches [2, 39] adapt learning rate in proximal gradient algorithm, which is a good direction for the future work.

C.3. Proximal Algorithm Finds Structural Sparse Solution Faster

Weight decay will eventually find sparse solution, however with an inordinate number of training steps as indicate in Fig. 1 (*right*). Proximal algorithm helps accelerating the training process. To demonstrate this, we perform experiments on **(Task 5)**, and compare the sparse solution found by 1) weight decay, 2) proximal algorithm for ℓ_2-PATH-NORM ; as well as well-known regularizations that enforce sparsity in the solution: 3) lasso [36], and 4) group lasso [40]. As the proximal algorithm operates on each one homogeneous units (instead of weight parameter), in this section we mainly focus on structural sparsity of the model.

For each experiments, we first prune the model in each training step (prune one unit if either its input vector $\boldsymbol{w}$ or output vector $\boldsymbol{v}$ is zeroed out). After training and pruning for 30000 iterations, we take the model checkpoint from iterations $\{5000, 10000, \cdots, 30000\}$, set the unit to be inactive if $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2 < 10^{-5}$, and then train this sparse model for another 10000 iterations. Since different λ and *learning rate* may lead to different level of sparsity, we try with *learning rate* in $\{0.01, 0.03\}$, and $\lambda \in \{0.0001, 0.001, 0.003, 0.01\}$ for all method.

Performance of the sparse solution is presented in Fig. 3. For each sparsity level s, we present the best solution with sparsity $< s$. Compared with proximal algorithm for ℓ_2-PATH-NORM , weight

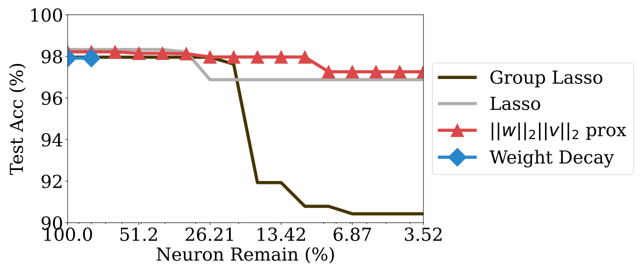

Figure 3: Sparse solution found by 1) weight decay, 2) proximal algorithm for ℓ_2-PATH-NORM , 3) lasso, and 4) group lasso. We highlight the sparse solution of weight decay and $\|\boldsymbol{w}\|_2\|\boldsymbol{v}\|_2$ proximal algorithm, which solves the equivalent objective function. However weight decay couldn't find as sparse solution as the ℓ_2-PATH-NORM proximal.

decay only finds solution of sparsity $> 64\%$. Though the proximal algorithm doesn't enforce sparsity, it finds competitive solution as lasso, and outperform group lasso in the sparse regime.