The Impact of Reinitialization on Generalization in Convolutional Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

We study the impact of different reinitialization methods in several convolutional architectures for small-size image classification datasets. We analyze the potential gains of reinitialization and highlight limitations. We also study a new layerwise reinitialization algorithm that outperforms previous methods and suggest explanations of the observed improved generalization. First, we show that layerwise reinitialization increases the margin on the training examples without increasing the norm of the weights, hence leading to an improvement in margin-based generalization bounds for neural networks. Second, we demonstrate that it settles in flatter local minima of the loss surface. Third, it encourages learning general rules and discourages memorization by placing emphasis on the lower layers of the neural network.

1 Introduction

Deep neural networks demonstrate state-of-the-art performance on many classification tasks. While often highly overparameterized, modern deep architectures exhibit a remarkable ability to generalize beyond the training sample even when trained without an explicit form of regularization (Zhang et al., 2017). A large body of work has been devoted to offering insights into this "benign" overfitting phenomenon, including explanations based on the margin (Neyshabur et al., 2015; Bartlett et al., 2017; Neyshabur et al., 2017; Arora et al., 2018; Soudry et al., 2018), the curvature of the local minima (Keskar et al., 2017; Chaudhari et al., 2019; Neyshabur et al., 2020), and the speed of convergence (Hardt et al., 2016), among others.

Recently, however, a number of works suggest that generalization in convolutional neural networks (CNNs) could be improved further using reinitialization. Precisely, let $\mathbf{w} \in \mathbb{R}^d$ be a vector that contains all of the parameters in a neural network (e.g. filters in convolutional layers and weight matrices in fully-connected layers). Let $\mathbf{s} \in \{0, 1\}^d$ be a binary mask that is generated at random according to some probability mass function. Then, "reinitialization" is selecting a subset of parameters and reinitializing them during training:

$$\mathbf{w} \leftarrow (1 - \mathbf{s}) \odot \mathbf{w} + \mathbf{s} \odot \eta, \tag{1}$$

where $\odot$ is an element-wise multiplication and η is a random initialization of the model parameters. For example, η may correspond to the weights of He or Xavier initializations (He et al., 2015; Glorot & Bengio, 2010). In the following, we refer to the update in (1) as a "reinitialization round." Reinitialization methods differ in how the binary mask $\mathbf{s}$ is selected. Four prototypical approaches are:

- **Random subset**: A random subset of the parameters of a fixed size is chosen uniformly at random in each round. This includes, for example, the random weight level splitting (WELSR) method studied in (Taha et al., 2021), in which about 20% of the parameters are selected for reinitialization.
- **Weight magnitudes**: The smallest parameters in terms of their absolute magnitudes are reinitialized at each round. This can be interpreted as a generalization to the dense-sparse-dense (DSD) workflow of Han et al. (2017) in which reinitialization occurs only once.
- **Fixed subset**: A subset is chosen at random prior to training and is fixed afterwards. It corresponds to the weight level splitting (WELS) method of Taha et al. (2021).

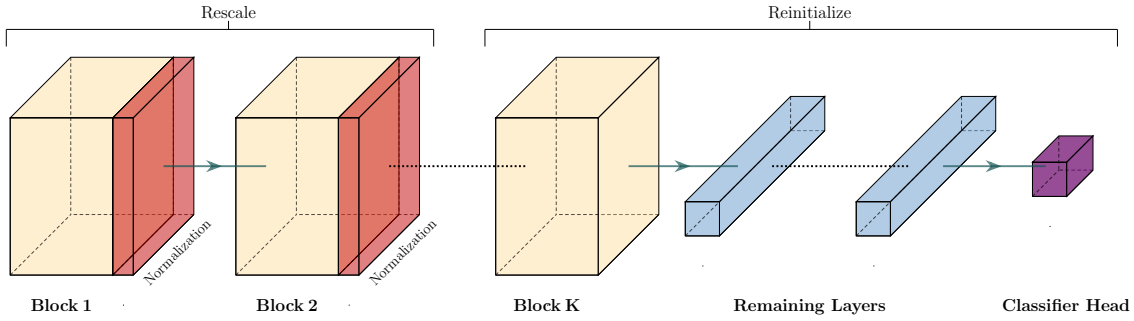

Figure 1: Given a deep neural network starting with K convolutional blocks followed by other layers, LW proceeds sequentially from bottom to top (see Algorithm 1). When in block k (e.g. $k = 2$ in the figure above), the weights of all early blocks $\{1, \ldots, k\}$ are rescaled while subsequent layers are reinitialized. In addition, a normalization layer is inserted following block K shown in red. Importantly, the network may contain other normalization layers within each block, such as batch or layer normalization (Ioffe & Szegedy, 2015; Ba et al., 2016). Red layers correspond to the normalization layers inserted by LW, which are fixed (non-trainable) at each round.

- **Fully-connected layers**: Only the last fully-connected layers are reinitialized. This includes, for example, the method proposed in (Li et al., 2020). In (Zhao et al., 2018), only the classifier head is reinitialized.

We denote these four methods as WELSR, DSD, WELS, and FC, respectively. Moreover, we denote the baseline method of training once until convergence as BL.

In this paper, we also study a new reinitialization algorithm, which we denote as LW for its LayerWise approach. The new algorithm is motivated by the common observation that lower layers in the neural network tend to learn general rules while upper layers specialize (Yosinski et al., 2014; Arpit et al., 2017; Raghu et al., 2019; Maennel et al., 2020; Baldock et al., 2021). While all reinitialization methods improve generalization in CNNs, we demonstrate in Section 3 that LW often outperforms the other methods. It encourages learning general rules by placing more emphasis on training the early layers of the neural network. A more formal statement is presented in Section 3.

- **Layerwise**: A convolutional neural network is partitioned into K blocks (see Figure 1 and Algorithm 1). At round k, the parameters at the lowest K blocks are rescaled back to their original norm during initialization (see Algorithm 1) while the rest of the network is reinitialized. In addition, a new normalization layer is inserted/updated following block K. This is repeated for a total of $N \geq 1$ iterations for each block.

It is worth noting that FC is a special case of LW, in which $K = 1$ and $N > 1$. In addition, the concurrent work of Zhou et al. (2021) also corresponds to $K = 1$ where the upper L layers are reinitialized at each round for some fixed $L > 1$. Besides the prominent role of reinitialization, LW includes normalization and rescaling, which we show in an ablation study in Appendix D to be important. Next, we illustrate the basic principles of these reinitialization methods on a minimal example with synthetic data.

1.1 Synthetic Data Example

Setup. Let $\mathbf{x} \in \mathbb{R}^{128}$ be the instance and $\mathbf{y}$ be its label, which is sampled uniformly at random from the set $\{0, 1, ..., 7\}$. For the instances, on the other hand, each of the first 3 coordinates of $\mathbf{x}$ is chosen from $\{-1, 1\}$ to encode the label $\mathbf{y}$ in binary form. For example, instances in class 0 would have their first three coordinates as $(-1, -1, -1)$, whereas instances in class 5 would have $(1, -1, 1)$. Consequently, the first three coordinates of an instance correspond to its "signal." The remaining 125 entries of $\mathbf{x}$ are randomly sampled i.i.d. from $\mathcal{N}(0, 1)$.

Algorithm 1 Pseudocode of LW

Input: (1) Neural network with identified sequence of $K \geq 1$ conv blocks; (2) Training dataset; (3) $N \geq 1$.

Output: Trained model parameters.

Training:

```
Initialize the neural network architecture;
For each layer l, compute s_l = ||W_l||_2, where W_l are the weights of layer l.
for k ∈ (1, 2, ..., K) do
    for n ∈ (1, 2, ..., N) do
        for j ∈ (1, 2, ..., k) do # rescaling
            for layer ∈ Block j do
                W_layer ← (s_layer / ||W_layer||_2) · W_layer
            end for
        end for
        Pick a batch X of training set uniformly at random;
        Compute Z: the output of Block k of X;
        Compute μ, σ ∈ ℝ: mean and standard deviation of Z;
        if n = 1 then
            Insert lambda layer λx : (x − μ)/σ after block k;
        else
            Update lambda layer with new values of μ and σ;
        end if
        Reinitialize all layers above block k;
        Fine-tune the entire model until convergence;
    end for
end for
```

Table 1: Test accuracy [%] for the synthetic data experiment of Section 1.1 with different signal strengths α and different reinitialization methods. We observe that all initialization methods (with the exception of DSD) improve generalization in this example setting with LW performing best. In addition, reinitialization methods also tend to reduce the variance of the test accuracy.

α	BL	WELSR	DSD	WELS	FC	LW
0.5	20.3 ± 0.6	24.6 ± 1.0	22.9 ± 0.6	23.1 ± 1.4	23.6 ± 3.0	25.2 ± 0.8
1.0	50.7 ± 5.4	72.9 ± 0.9	53.4 ± 0.7	66.1 ± 2.1	68.6 ± 2.1	72.3 ± 3.6
2.0	94.6 ± 2.0	98.2 ± 0.4	90.3 ± 1.4	96.8 ± 0.1	99.0 ± 0.2	99.8 ± 0.2

Although we focus in this work on convolutional neural networks, we use a multilayer perceptron (MLP) in this synthetic data experiment because the inputs are not images but generic feature vectors. The MLP contains two hidden layers of 32 neurons with ReLU activations (Nair & Hinton, 2010) followed by a classifier head with softmax activations. It optimizes the cross–entropy loss. We train on 256 examples using gradient descent with learning rate 0.05.

Methods. Treating every layer as a block, we have $K = 3$. If 200 training steps are used per round of reinitialization and $N = 3$, LW trains the model once for 200 steps after which the 2nd and 3rd layers are reinitialized (in addition to rescaling and normalization). This is carried out $N = 3$ times in the first layer before k is incremented. The same process is repeated on each layer making a total of $200 \times N \times K = 1,800$ training steps overall. In WELS, WELSR, DSD, and FC, the model is trained for 200 steps before reinitialization is applied, and this is repeated $K \times N$ times for the same total of $1,800$ steps. The baseline method corresponds to training the model once without reinitialization for a total of $1,800$ training steps.

Results. When trained for 1,800 steps, the baseline (BL) achieves 100% training accuracy, but only around 51% test accuracy. The large gap between training and test accuracy for such a simple task is reminiscent of the classical phenomenon of overfitting. Note that the number of training examples is 256, which is generally small for 128 features of equal variance. On the other hand, reinitialization improves accuracy as shown

in Table 1 even though these reinitialization methods do *not* have access to any additional data and use the same optimizer and hyper-parameters as baseline training. The training accuracy is 100% in all cases. We also observe that reinitialization tends to reduce the variance of the test accuracy (with respect to the random seed).

In the above experiment, both the signal part (first three coordinates) and the noise part (remaining coordinates) have the same scale (standard deviation 1). We can make the classification problem easier or harder by multiplying the signal part by a signal strength $\alpha > 1$ or $\alpha < 1$, respectively. We present the average test accuracy in Table 1 for a selection of values of α with $N = 3$. Appendix A contains additional results when weight decay is added.

1.2 Contribution

In this work, we study a new layerwise reinitialization algorithm LW, which often outperforms other methods. We provide two explanations, supported by experiments, for why it improves generalization in convolutional neural networks. First, we show that LW improves the margin on the training examples without increasing the norm of the weights, hence leading to an improvement in known margin-based generalization bounds in neural networks. Second, we show that LW settles in flatter local minima of the loss surface.

Furthermore, we provide a comprehensive study comparing previous reinitialization methods: First, we evaluate different methods within the same context. For example, the comparison in (Taha et al., 2021) uses only a single reinitialization round of the dense-sparse-dense approach (DSD) when DSD can be extended to multiple rounds. Also, (Zhao et al., 2018) uses an ensemble of classifiers when reinitializing the fully-connected layers, which could (at least partially) explain the improvement in performance. By contrast, we follow a coherent training protocol for all methods. Second, we use our empirical evaluation to analyze the effect of the experiment's design, such as augmentation, dropout and momentum. The goal is to determine if the effect of reinitialization could be achieved by tuning such settings. Third, we employ decision tree classifiers to identify when each reinitialization method is likely to outperform others. In summary, we:

1. Study a new reinitialization method, denoted LW, which is motivated by common observations of generalization and memorization effects across the neural network's layers. We show that it outperforms other methods with a statistically significant evidence at the 95% level.

2. Suggest two explanations, supported by experiments, for why LW is more successful at improving generalization in CNNs compared to other methods.

3. Present a comprehensive evaluation study of reinitialization methods covering more than 1,000 experiments for four convolutional architectures: (1) simplified CNN, (2) VGG16 (Simonyan & Zisserman, 2015), (3) MobileNet (Howard et al., 2017) and (4) ResNet50 (He et al., 2016a). We conduct the evaluation over 6 benchmark image classification datasets of small size (up to 12,000 training examples per dataset). We do not observe consistent gains of reinitialization with large datasets, so we omit those from the comparison and focus on the small-data regime.

2 Related Work

Reinitialization. As stated earlier, a number of works suggest that reinitializing a subset of the neural network parameters during training can improve generalization. This includes, the dense-sparse-dense (DSD) training workflow proposed by Han et al. (2017), in which reinitialization occurs only once during training. However, as the authors argue, the improvement in accuracy in DSD could be attributed to the effect of introducing sparsity, not reinitialization. Another example is "Knowledge Evolution", including weight level splitting (WELS) and its randomized version (WELSR) (Taha et al., 2021). It was noted that WELS outperformed WELSR, which agrees with our observations. Finally, some recent works propose to reinitialize the fully-connected layers only (Li et al., 2020; Zhao et al., 2018). In particular, reinitializing the last layer several times and combining the models into an ensemble can improve performance (Zhao et al., 2018). However, the improvement in accuracy could (at least partially) be attributed to the ensemble of predictors,

not to reinitialization *per se*. For fair comparison, we extend DSD to multiple rounds of reinitialization and do not use an ensemble of predictors.

Generalization Bounds. Several generalization bounds for neural networks have been proposed in the literature. Of those, a prototypical approach is to bound the generalization gap by a particular measure of the *size of weights* normalized by the *margin* on the training set. Examples of measures of the size of weights include the product of the ℓ_1 norms (Bartlett, 1998) and the product of the Frobenius norms of layers (Neyshabur et al., 2015), among others (Bartlett et al., 2017; Neyshabur et al., 2017; Arora et al., 2018). While such generalization bounds are often loose, they were found to be useful for ranking models (Neyshabur et al., 2017). The fact that rich hypothesis spaces could still generalize if they yield a large margin over the training set was used previously to explain the performance of boosting (Schapire et al., 1997). In Section 4, we show that LW boosts the margin on the training examples without increasing the size of the weights.

Flatness of the Local Minimum. Another important line of work examines the connection between generalization and the curvature of the loss at the local minimum (Keskar et al., 2017; Neyshabur et al., 2017; Foret et al., 2021). Deep neural networks are known to converge to local minima with sparse eigenvalues (>94% zeros) in their Hessian (Chaudhari et al., 2019). Informally, a flat local minimum is robust to data perturbation, and this robustness can, in turn, be connected to regularization (Bishop, 1995). In fact, some of the benefits of transfer learning were attributed to the flatness of the local minima (Neyshabur et al., 2020). For a precise treatment, one may use the PAC-Bayes framework to derive a generalization bound that comprises of two terms: (1) sharpness of the local minimum, and (2) the weight norm over noise ratio (Neyshabur et al., 2017). Similar terms also surface in the notion of "local entropy" (Chaudhari et al., 2019). We show in Section 4 that LW improves both terms.

Generalization vs. Memorization. Several works point out that early layers in a neural network tend to learn general-purpose representations whereas later layers specialize, e.g. (Raghu et al., 2019; Arpit et al., 2017; Yosinski et al., 2014; Maennel et al., 2020). This can be observed, for instance, using probes, in which classifiers are trained on the layer embeddings. As demonstrated in (Cohen et al., 2018) and (Baldock et al., 2021), deep neural networks learn to separate classes at the early layers with real labels (generalization) but they only separate classes at later layers when the labels are random (memorization). One explanation for why LW improves generalization is that it encourages learning general rules at early layers and discourages memorization at later layers.

3 Analysis

3.1 Layerwise Reinitialization as a Stochastic Gradient Descent

Before presenting our empirical study of the different reinitialization methods, we discuss briefly why reinitialization can be interpreted as a stochastic gradient descent (SGD) procedure to a well-defined stochastic loss. Later in Section 4, we present arguments for the improved generalization in LW that are linked to the margin on the training examples as well as the flatness of the local minimum.

Consider the following simplified training protocol. In a multi-layer neural network, let $w_0 \in \mathbb{R}^{d_0}$ be the set of weights at the first layer and write $\bar{w}_0 \in \mathbb{R}^{d-d_0}$ for the set of weights at all later layers. Let w_0^t and $\bar{w}_0^t$ be the set of weights after round t. We will simplify discussion by denoting $w^t = (w_0^t, \bar{w}_0^t) \in \mathbb{R}^d$ and focusing on the first layer alone.

Given a loss function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, training via stochastic gradient descent (SGD) leads to a stationary point of the loss surface (i.e. a solution w such that $\nabla\mathcal{L}(w) = 0$). Let $\mathcal{S}$ be the set of stationary points of $\mathcal{L}$. To mimic the behavior of LW, suppose that training proceeds at round t by reinitializing $\bar{w}_0$ and applying the *proximal* operator to the full set of weights w:

$$w^t = \arg\min_{w \in \mathcal{S}} ||w - w^{t-1}||_2^2. \tag{2}$$

Table 2: Test accuracy results [%] for the five reinitialization methods across 6 benchmark datasets. Values highlighted in **bold**/underlined correspond to the **best**/second-best results. The symbols B,R,D,W,F,L are for baseline, WELSR, DSD, WELS, FC, and LW, respectively. In LW, $N = 1$. Every reinitialization uses the same number of rounds K (cf. Appendix B) and they are trained for the same number of epochs (including the baseline). In WELS, WELSR, and DSD, we reinitialize 20% of the parameters, following Taha et al. (2021).

		B	R	D	W	F	L		B	R	D	W	F	L
OXFORD-IIIT	scnn	13.7	12.0	13.4	14.0	14.0	**15.0**	vgg16	16.2	16.7	16.9	16.4	**29.7**	25.6
	mobilenet	13.4	13.8	17.2	15.9	14.8	**30.1**	resnet	24.0	24.2	22.9	24.9	23.7	**28.5**
DOGS	scnn	5.8	5.2.	5.2	5.1	5.1	**6.1**	vgg16	11.4	11.8	10.7	10.3	**24.2**	19.1
	mobilenet	9.7	8.6	9.0	11.6	9.9	**19.2**	resnet	14.0	17.5	19.2	16.0	13.9	**22.0**
CARS196	scnn	3.4.	3.4	3.4	3.5	3.2	**3.8**	vgg16	6.5	5.5	7.7	7.0	**20.9**	9.9
	mobilenet	6.1	8.9	5.3	7.4	6.8	**22.2**	resnet	10.1	12.8	13.4	**13.7**	10.4	12.6
CALTECH101	scnn	**51.1**	49.9	49.8	48.4	50.8	50.7	vgg16	54.1	55.4	54.2	55.9	**69.1**	57.3
	mobilenet	36.7	43.0	44.9	40.9	42.3	**47.0**	resnet	50.4	54.6	**55.0**	52.9	50.7	53.3
CASSAVA	scnn	59.4	58.6	58.6	59.8	61.6	**63.9**	vgg16	58.4	59.5	58.4	**62.1**	58.5	59.3
	mobilenet	52.6	57.9	63.6	62.2	55.4	**70.0**	resnet	61.9	57.3	62.8	58.1	56.7	**63.9**
BIRDS2010	scnn	2.0	**2.6**	2.3	2.2	2.4	2.4	vgg16	3.8	4.3	4.1	3.4	**8.5**	5.1
	mobilenet	4.5	3.9	5.2	5.9	5.3	**8.1**	resnet	6.9	8.7	**10.0**	**10.0**	6.5	**10.0**
+ Augmentation														
OXFORD-IIIT	scnn	15.1	14.6	14.3	14.7	16.0	**16.6**	vgg16	22.0	20.0	20.9	20.7	**39.1**	29.8
	mobilenet	27.7	24.1	23.2	24.3	23.5	**41.7**	resnet	29.7	33.3	39.1	31.9	**36.6**	34.4
DOGS	scnn	7.4	8.2	7.7	8.0	8.1	**8.6**	vgg16	16.0	16.9	15.5	16.7	**37.3**	31.2
	mobilenet	19.0	19.5	27.1	18.8	20.7	**35.8**	resnet	26.4	30.7	28.4	35.2	33.3	**36.9**
CARS196	scnn	5.6.	6.0	5.3	5.5	**7.3.**	5.8	vgg16	11.7	10.7	11.2.	11.4	**43.6**	22.4
	mobilenet	6.9	16.2	9.1	11.9	13.8	**44.0**	resnet	21.8	42.5	33.2	43.1	36.7	**43.6**
CALTECH101	scnn	50.1	52.2	52.2	51.5	**54.4**	52.8	vgg16	55.9	55.8	57.1	56.3	**67.1**	59.1
	mobilenet	41.0	41.5	47.4	41.8	**48.0**	46.3	resnet	50.2	51.9	53.1	**57.5**	52.1.	50.5
CASSAVA	scnn	58.9	60.6	59.4	62.2	67.0	**69.3**	vgg16	70.3	70.0	70.0	71.2	**71.5**	68.3
	mobilenet	62.3	72.8	**80.1**	76.1	77.3	**81.1**	resnet	46.5	79.2	**82.9**	82.6	77.6	73.9
BIRDS2010	scnn	3.8	3.7	4.0	3.2	**4.2**	3.8	vgg16	5.5	6.5	5.6	5.8	**13.2**	9.8
	mobilenet	6.6	9.0	8.1	6.2	7.4	**9.8**	resnet	8.5	12.5	11.2	**13.0**	11.9	10.6
+ Augmentation + Dropout														
OXFORD-IIIT	scnn	15.8	13.6	14.3	15.1	**19.3**	17.1	vgg16	25.3	26.7	26.6	26.4	**43.4**	34.4
	mobilenet	28.4	29.1	27.8	27.9	22.0	**41.0**	resnet	33.1	35.2	**41.2**	37.4	32.6	35.6
DOGS	scnn	8.4	8.9	7.6	8.0	**9.6**	9.1	vgg16	17.7	19.7	19.5	18.9	34.9	**35.9**
	mobilenet	17.8	23.5	27.3	22.4	20.5	**35.0**	resnet	30.7	33.3	33.1	33.8	34.5	**40.1**
CARS196	scnn	6.3	6.2	5.9	6.8	**7.4**	6.4	vgg16	14.3	18.9	12.6	16.6	**45.2**	34.2
	mobilenet	9.5	16.1	30.8	24.1	16.4	**44.5**	resnet	19.8	45.7	43.0	**48.1**	45.0	47.5
CALTECH101	scnn	51.4	**53.5**	51.1	52.6	52.4	**53.6**	vgg16	59.6	61.7	60.8	61.0	**68.1**	62.7
	mobilenet	47.1	43.0	47.5	45.6	**51.3**	49.4	resnet	50.6	54.1	54.6	**55.7**	51.4	50.3
CASSAVA	scnn	61.8	62.9	60.7	61.9	68.4	**68.6**	vgg16	70.1	71.4	**73.7**	71.0	71.1	71.9
	mobilenet	68.5	70.0	78.6	74.3	73.7	**80.5**	resnet	74.2	77.6	**81.5**	80.9	73.5	78.6
BIRDS2010	scnn	4.0	4.0	3.5	3.7	3.3	**4.4**	vgg16	6.5	8.1	8.5	8.1	**18.6**	12.6
	mobilenet	2.5	**9.6**	5.1	6.3	8.4	9.0	resnet	11.5	14.8	13.3	**15.2**	13.5	13.3

Table 3: Statistical significance: a star ($\star$) implies that the column method outperforms the row with statistically significant evidence at the 95% level, computed using the exact binomial test. A circle ($\circ$) implies that statistical significance holds even after applying Holm's step-down correction for multiple hypothesis tests (Demšar, 2006). Only LW performs better than the baseline across all architectures. For `resnet50`, reinitialization methods except FC perform better than the baseline with no clear winner among them.

	scnn						vgg16						mobilenet						resnet50					
	B	R	D	W	F	L	B	R	D	W	F	L	B	R	D	W	F	L	B	R	D	W	F	L
B						$\circ$				$\star$	$\circ$	$\circ$						$\circ$		$\star$	$\circ$	$\circ$		$\star$
R						$\star$					$\star$	$\star$						$\star$				$\star$		
D						$\star$					$\star$	$\star$						$\star$						
W					$\star$	$\star$					$\circ$	$\star$						$\circ$						
F																		$\star$						
L											$\star$													

Table 4: Overview of the 6 benchmark datasets.

Name	|Training|	|Test|	# Classes
OXFORD-IIIT (Parkhi et al., 2012)	3,680	3,669	37
DOGS (Khosla et al., 2011)	12,000	8,580	120
CARS196 (Krause et al., 2013)	8,144	8,041	196
CALTECH101 (Fei-Fei et al., 2004)	3,060	6,084	101
CASSAVA (Mwebaze et al., 2019)	5,656	1,885	4
BIRDS2010 (Welinder et al., 2010)	3,000	3,033	200

Informally, (2) selects a stationary point of $\mathcal{L}$ that is nearest to the current solution in the reinitialization round. If we write $\Psi_{\mathcal{S}}$ for the indicator function of the set $\mathcal{S}$:

$$\Psi_{\mathcal{S}}(w) = \begin{cases} 0, & w \in \mathcal{S} \\ \infty, & w \notin \mathcal{S}, \end{cases} \tag{3}$$

then a single training round in (2) corresponds to a gradient descent step to the *Moreau envelope* of $\Psi_{\mathcal{S}}$ (Parikh & Boyd, 2014), which in this case is the distance to $\mathcal{S}$. That is, a training round in (2) is equivalent to the gradient update step:

$$w^{t+1} = w^t - \nabla M_{\Psi_{\mathcal{S}}}(w^t). \tag{4}$$

However, for the set of weights at the first layer w_0, reinitialization transforms the update rule in (4) into a *stochastic* gradient step $w_0^{t+1} = w_0^t - \nabla f_t(w_0^t)$, in which f_t is a stochastic loss function whose randomness is derived from the randomness of $\bar{w}_0$ and satisfies:

$$f_t(w_0) = M_{\Psi_{\mathcal{S}}}((w_0, \bar{w}_0^t)) = \min_{(x,\bar{x})\in\mathcal{S}} \left\{||x - w_0||_2^2 + ||\bar{x} - \bar{w}_0||_2^2\right\}. \tag{5}$$

Hence, a single reinitialization round, where the weights of the first layer are retained while the rest is reinitialized, can be interpreted as a stochastic gradient descent step to the loss in (5), which penalizes weights at the first layer w_0 that change significantly when the rest of the network is reinitialized and retrained. Repeating this several times for the first layer is analogous to choosing a large value of $N \gg 1$ in LW. Once the first layer is trained, its output can be normalized before proceeding to the next layer, which is what LW achieves.

3.2 Empirical Study

We begin by evaluating the performance of the five reinitialization methods discussed in Section 1 for four convolutional architectures on 6 small-size benchmark image classification datasets (see Table 4 for details). All images are resized to 224×224. The architectures are (1) simplified CNN, (2) VGG16 (Simonyan & Zisserman, 2015), (3) MobileNet (Howard et al., 2017) and (4) ResNet50 (He et al., 2016a). We denote these by `scnn`, `vgg16`, `mobilenet`, and `resnet50`, respectively. We use He-initialization (He et al., 2015) unless stated otherwise. Details about each architecture and the hyper-parameters are provided in Appendix B.

To recall, every reinitialization method trains the same model on the same dataset for several rounds. After each round, a binary mask of the model parameters is selected according to the reinitialization criteria and the update in Eq. (1) is applied for some random initialization η. After that, the model is fine-tuned on the same data. Blocks in LW correspond to the standard blocks of the architecture (e.g. a block in Figure 1 would correspond to either an identity or a convolutional block in ResNet50). Also, 10% of the training split is reserved for validation, which is used for early stopping in all methods.

To evaluate the relative performance of the reinitialization methods, we perform a set of experiments in which we *fix* the hyperparameters for all architectures and datasets to the same values. The hyperparameters were chosen to work reasonably well across all combinations; in particular they enable reaching 100% training accuracy in all cases. We use SGD with an initial learning rate of 0.003 and momentum 0.9. The learning rate is decreased by a factor of 2 whenever the validation error does not improve for 20 epochs. The

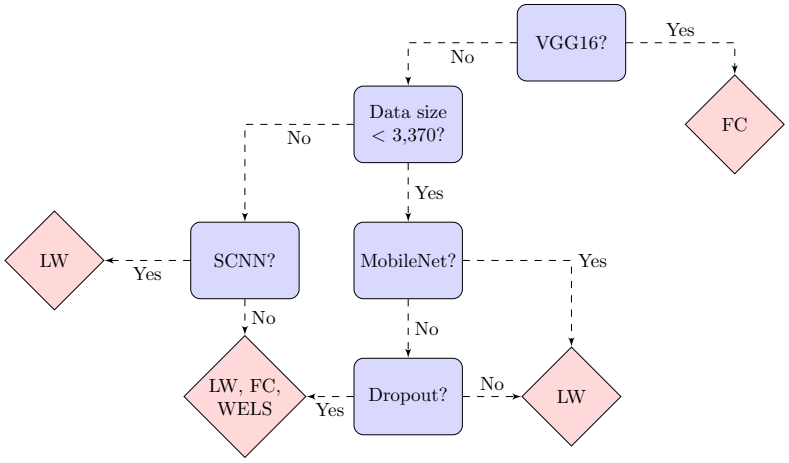

Figure 2: A decision tree classifier trained to predict the best reinitialization method based on the experiment design. The features are the training set size, number of classes, neural network architecture, dropout rate and augmentation. In general, LW performs best overall except in `vgg16`, where FC performs better. The decision tree classifier has a maximum depth of 4 and a minimum number sample leaf of 7.

batch size is 256 and a maximum of 100k minibatch steps are used. We run all experiments, as explicitly stated, without data augmentation or with mild augmentation consisting of horizontal flipping and random cropping (in which the size is increased to 248×248 before a crop of size 224×224 is selected). Such fixed hyperparameters are suboptimal for some combinations of architectures and datasets and therefore the resulting numbers can be worse than state-of-the-art results. However, they enable reaching 100% training accuracy in all combinations of models and datasets. For example, increasing the learning rate to 0.01 would prevent ResNet50 from progressing its training error beyond that of random guessing on CASSAVA.

Table 2 provides the detailed results of the five reinitialization methods across the benchmark datasets in three settings: (1) no augmentation or dropout is used, (2) with augmentation, and (3) with both augmentation and dropout rate 0.25. We perform an exact binomial test to evaluate which method performs statistically significantly better across the settings. In Table 3, we summarize these results. We observe that only LW outperform the baseline across all architectures with statistically significant evidence, and outperforms the other reinitialization methods in all architectures except `resnet50`. In `resnet50`, reinitialization methods except FC perform better than the baseline but with no clear winner among them. Moreover, FC performs generally better than WELS, WELSR, and DSD. It is worth reiterating, that FC is a special case of LW that corresponds to $K = 1$ and $N > 1$.

3.3 Effect of Experiment Design

To determine when a particular reinitialization method outperforms others, we train a decision tree classifier on the outcomes of several experiments that vary in design by, for example, number of classes, size of the training dataset, augmentation, and dropout. Every setting contains experiment runs of each of the 5 reinitialization methods in addition to the baseline for the four architectures and 6 benchmark datasets.

We use the decision tree classifier, implemented using the Scikit-Learn package (Pedregosa et al., 2011), for interpretability. We use a minimum leaf size of 7 in the decision tree and a maximum depth of 4. Figure 2 displays the resulting decision tree. In general, LW performs best overall except in `vgg16`, where FC performs better.

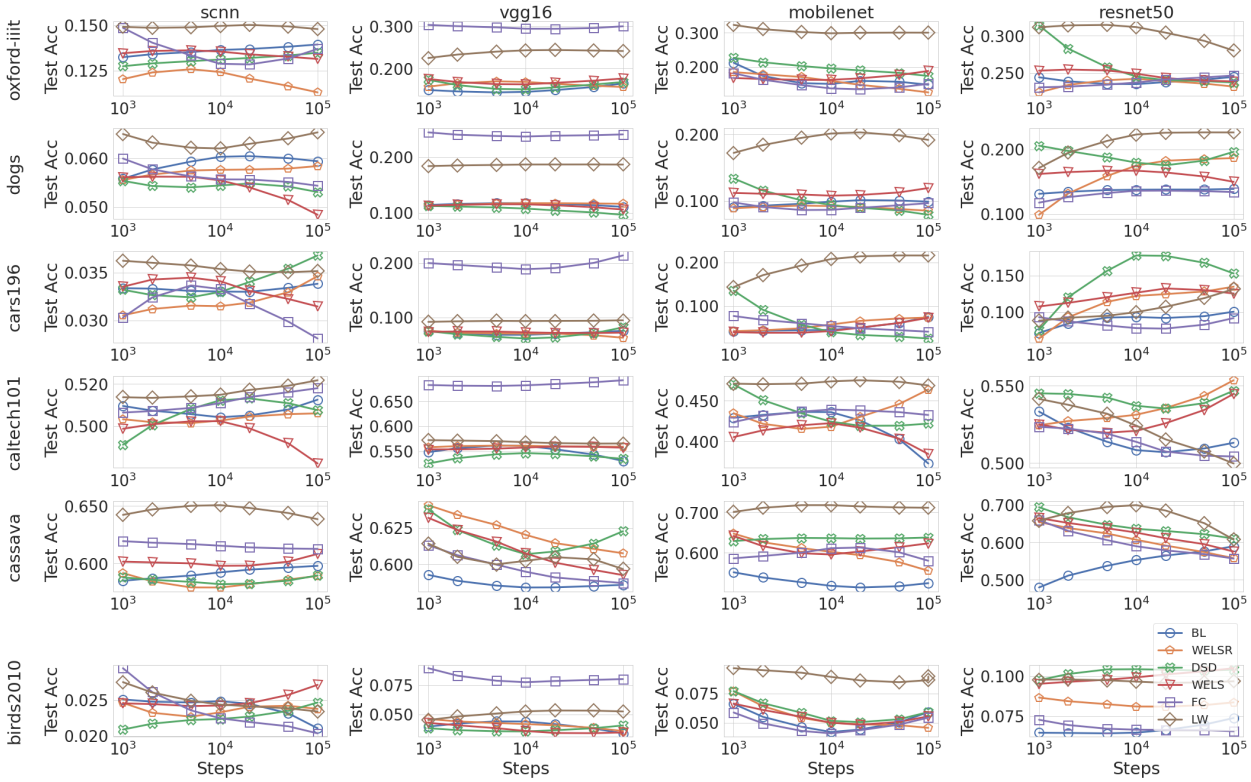

Figure 3: The test accuracy of reinitialization methods with different compute budgets (no augmentation or dropout) is plotted for each dataset. The x-axis is the number of training steps per reinitialization round. For the baseline, the test accuracy is plotted over the same total number of steps as reinitialization methods. Most reinitialization methods quickly surpass the accuracy of the baseline for the same amount of compute and can reap benefit of reinitialization without having to train until convergence in each round.

3.4 Compute

In Table 2, every reinitialization round is trained until convergence. However, improvement in generalization can also be obtained at lower computational overhead by stopping early in each round. This is demonstrated in Figure 3 for all six benchmark datasets. As shown in the figure, early stopping allows to realize the gain of reinitialization without incurring significant additional overhead. In addition, we show in Appendix E that training is faster in subsequent rounds of reinitialization.

4 Relations to the Generalization Risk

Boosting the Margin. As discussed earlier in Section 2, a typical approach for bounding the generalization gap in deep neural networks is to use a particular measure of the size of the weights normalized by the *margin* on the training sample. Let D be the number of layers in a neural network, whose output is a composition of functions: $f(x) = f_1 \circ f_2 \circ \cdots f_D(x)$, where each $f_i(x)$ is of the form $f_i(x) = \sigma(W_i x)$ for some matrix W_i and ReLU activation σ. Then, one measure of the size of the weight that relates to generalization is the product of the Frobenius norms of layers $\prod_{i=1}^{d} ||W||_F^2$ (Neyshabur et al., 2015; 2017). This is normalized by the margin $\gamma > 0$ on the training examples, which is the smallest difference between the score assigned to the true label and the next largest score. For a better visualization, we use the margin of the softmax output in the interval [0, 1].

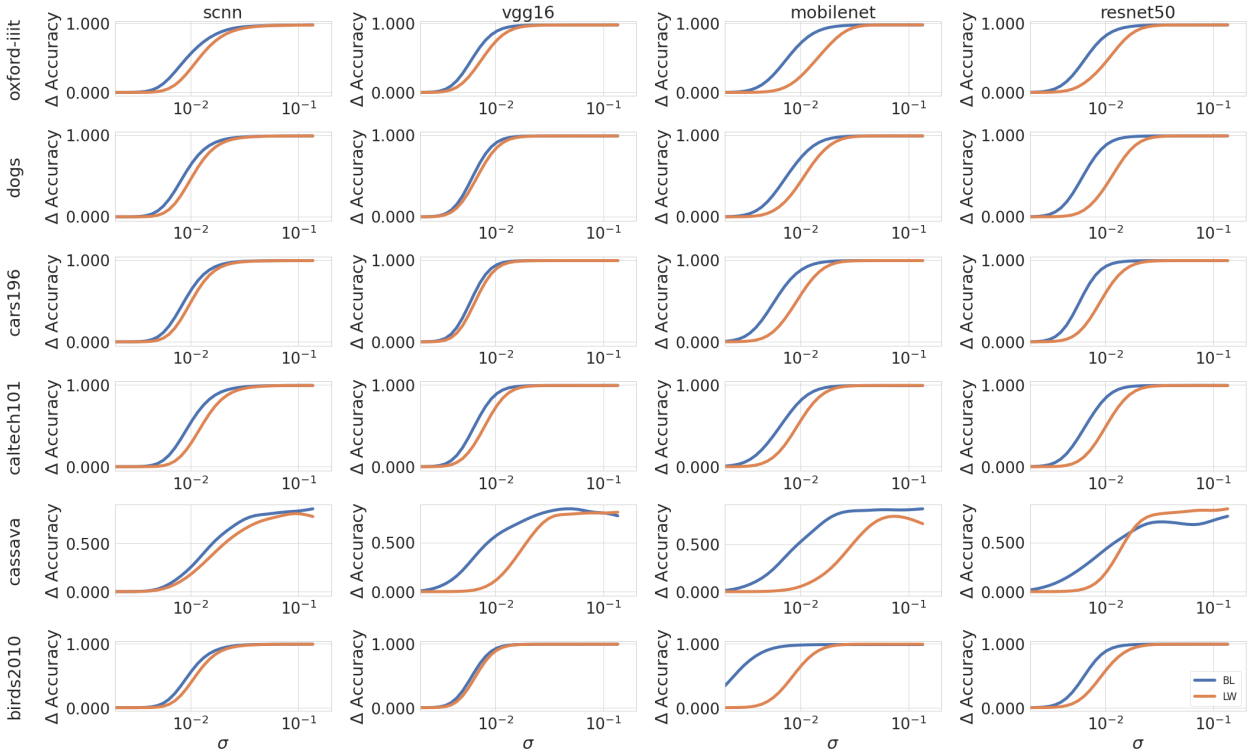

Figure 4: Bi-criteria plots for the change in training accuracy (y-axis) when the model parameters are perturbed by standard Gaussian noise $\mathcal{N}(0, \sigma^2 I)$ for each dataset. Lower curves suggest flatter local minima and better generalization.

Figure 5 displays the smallest 400 margins on the training sample for each of the benchmark datasets. As shown in the figure, LW boosts the margin on the training sample considerably when compared to previous reinitialization methods. Most importantly, LW achieves this *without* increasing the size of the weights. To take the contribution of the normalization layers into account when calculating the product $\prod_{i=1}^{d} ||W||_F^2$, we compare the product of the norms of the input to the classifier head (activations) and the norm of the weights of the classifier head in each method. We observe that LW tends to maintain the same size of the weights as the baseline. Appendix C provides further details.

We provide an informal argument for why this happens in LW. First, the product of the norms of the weights in the identified K blocks in LW (cf. Figure 1 and Algorithm 1) tend to remain unchanged due to the normalization layers inserted after each round. What changes is the norm of the *final* layers (following block K), but their norm tends to shrink because they train from scratch faster with each round. As for the margin, because the network classifies all examples correctly in a few epochs in the final round of LW, any additional epochs have the effect of increasing the margin to reduce the cross entropy loss.

Sharpness of the Local Minima. Finally, we observe that the final solution provided by LW seems to reside in a "flatter" local minima of the loss surface than in the baseline. One method for quantifying flatness is to compare the impact on the training loss when the model parameters are perturbed by Gaussian noise, which has been linked to generalization (Neyshabur et al., 2017). To recall, both LW and BL share the same size of the weights (cf. Appendix C). Figure 4 shows that the solution reached by LW is more robust to model perturbation than in standard training. More precisely, for every amount of noise added into the model parameters $\mathbf{w}$, the change in the training loss in LW is smaller than in standard training suggesting that the local minimum is flatter in LW.

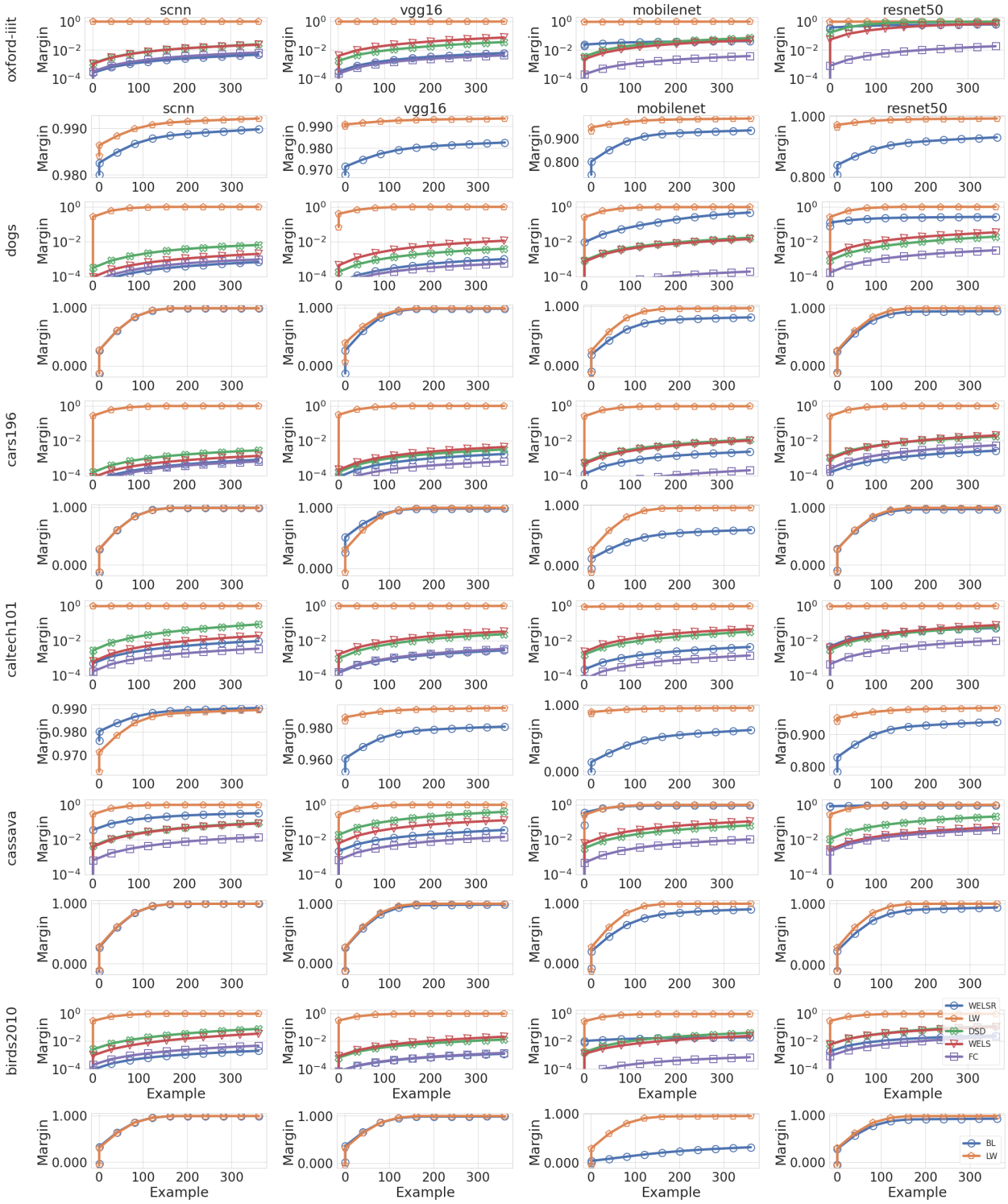

Figure 5: For each dataset, the top figure shows the smallest 400 margins in the training sample for different reinitialization methods. LW (orange) boosts the margin considerably compared to the other reinitialization methods. The bottom figure of each dataset provides the same comparison between LW and BL. The curves are displayed separately for a better visualization, as they almost coincide in the wide ranged log-scale in the top figure.

5 Discussion and Limitations

In this paper, we study a new reinitialization method for deep neural networks. Empirical results show that it improves generalization better than previous methods across a wide range of architectures and hyper-parameters. It relates to prior works that distinguish learning general rules in earlier layers from exceptions to the rules in later layers, because LW places more emphasis on the early layers of the neural network. We also argue that the improved generalization can be connected to the sharpness of the local minima and the margins on the training data.

To assess the limitations of the proposed method, we conducted ablation studies, statistical tests as well as failure analysis using decision trees. Those revealed that layerwise reinitialization yields a significant improvement in cases where the generalization gap is large, such as when using poor hyper-parameters or small datasets. The improvement is small, however, when the generalization gap is small, such as when the training data is large.

Our takeaway message is that the accuracy of convolutional neural networks can be improved for small datasets using bottom-up layerwise reinitialization, where the number of reinitialized layers may vary depending on the available compute budget. At one extreme, one would benefit from reinitializing the classifier's head alone, but reinitializing all layers in sequence with rescaling and normalization yields better results. We hope that the description of the observed positive effects will inspire others to study them more and to develop more efficient alternatives.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. *arXiv:1603.04467*, 2015.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *ICML*, 2018.

Devansh Arpit, Stanisław Jastrzkebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *ICML*, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. In *NeurIPS Deep Learning Symposium*, 2016.

Robert JN Baldock, Hartmut Maennel, and Behnam Neyshabur. Deep learning through the lens of example difficulty. In *NeurIPS*, 2021.

Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44 (2):525–536, 1998.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *NeurIPS*, 2017.

Christopher M Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7 (1):108–116, 1995.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

Gilad Cohen, Guillermo Sapiro, and Raja Giryes. DNN or k-NN: That is the generalize vs. memorize question. *arXiv:1805.06822*, 2018.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshops*, 2004.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *ICLR*, 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. DSD: Dense-sparse-dense training for deep neural networks. In *ICLR*, 2017.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *ICML*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016a.

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

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *CVPR Workshop on Fine-Grained Visual Categorization*, 2011.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *Int. IEEE Workshop on 3D Representation and Recognition*, 2013.

Xingjian Li, Haoyi Xiong, Haozhe An, Cheng-Zhong Xu, and Dejing Dou. RIFLE: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer. In *ICML*, 2020.

Hartmut Maennel, Ibrahim Alabdulmohsin, Ilya Tolstikhin, Robert JN Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. What do neural networks learn when trained with random labels? In *NeurIPS*, 2020.

Ernest Mwebaze, Timnit Gebru, Andrea Frome, Solomon Nsumba, and Jeremy Tusubira. iCassava 2019 fine-grained visual categorization challenge. *arXiv:1908.02900*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

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

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *NeurIPS*, 2017.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *NeurIPS*, 2020.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *CVPR*, 2012.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *NeurIPS*, 2019.

Robert E. Schapire, Yoav Freund, Peter L Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML*, 1997.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

Ahmed Taha, Abhinav Shrivastava, and Larry Davis. Knowledge evolution in neural networks. *arXiv:2103.05152*, 2021.

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Kaikai Zhao, Tetsu Matsukawa, and Einoshin Suzuki. Retraining: A simple way to improve the ensemble accuracy of deep neural networks for image classification. In *ICPR*, 2018.

Hattie Zhou, Ankit Vani, Hugo Larochelle, and Aaron Courville. Fortuitous forgetting in connectionist networks. In *ICLR*, 2021.

A Synthetic Data Experiment

We use the same type of data as described in Section 1, but look in more detail at the more difficult case $\alpha = 0.5$, this means the first three entries of the data encode the 8 possible labels as the 8 corners of the cube $[-0.5, 0.5]^3$, whereas the remaining entries are still sampled from the standard normal distribution. In addition, one may add a weight decay penalty to the task and examine the impact of rescaling alone. Specifically, we consider two cases:

- *Rescaling*: Instead of training once for N epochs, we train 5 times for $N/5$ epochs, and in between we scale back all weights such that the norm of each layer matches the norm after initialization.
- *Reinitialization* ("LW"): In addition to rescaling, we re-initialize the layers above the first one in the first two rounds, above the second layer in the next two rounds, and only the top layer in the last round.

The results are shown in Table 5. We use the same type of data as described above, but focus now at the more difficult case of $\alpha = 0.5$.

We observe that one can get significantly better results with weight decay. Nevertheless, LW gives an additional benefit on top of the L2 regularization: The best baseline result is 0.77, but the best results with Rescaling or LW are at 0.89 / 0.86. Note that the best L2 penalty needs to be estimated (e.g. by cross validation) for each data set and training procedure. In this case, less L2 penalty is needed if we apply Rescaling / LW .

In this particular experiment, rescaling seems to already give the full effect but this is not generally the case in natural image datasets, in which the gain seems to be modest without reinitialization.

Table 5: Test accuracies (average of 100 runs)

L2 penalty	Baseline	Rescaling	LW
0.0	0.19	0.21	0.25
0.005	0.51	0.67	0.82
0.01	0.54	**0.89**	**0.85**
0.02	0.58	**0.87**	**0.86**
0.05	**0.77**	0.78	0.79
0.1	0.59	0.63	0.64

B Experiment Setup

Throughout the main text, we use four different architectures: one simple convolutional neural network, and three standard deep convolutional models.

In all architectures, we use weight decay with penalty 10^{-5}. We also use layer normalization (Ba et al., 2016), implemented in TensorFlow (Abadi et al., 2015) using `GroupNormalization` layers with `groups=1`. Similar results are obtained when using Batch Normalization (Ioffe & Szegedy, 2015).

In all experiments, we use SGD as an optimizer with a learning rate of 0.003 and momentum 0.9. Also, we use a batch size of 256. All experiments are executed on Tensor Processing Units (TPU) for a maximum of 100,000 minibatch steps per reinitialization round. We resize images to 224×224 in all experiments.

Simple CNN (`scnn`). This architecture contains four convolutional blocks followed by one dense layer before the classifier head. The number of convolutional blocks K used in this architecture is 4. Every convolutional block is a 2D convolutional layer, followed by layer normalization and ReLU activation. Precisely:

```
conv2d              32 filters
layer_norm; activation_relu
```

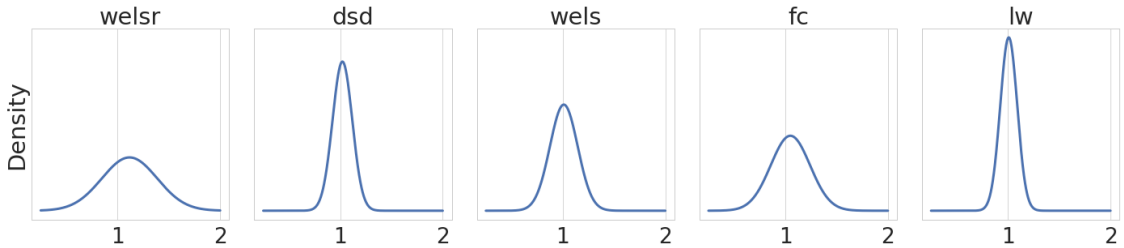

Figure 6: For each reinitialization method, the Gaussian approximation of the density of the *ratio* of the size of the weights over the size of the weights in the baseline method is shown. The density of the ratio in LW is concentrated around 1, which implies that LW tends to not increase the size of the weights. See Appendix C for details.

```
conv2d              32 filters
layer_norm; activation_relu
max_pooling2d

conv2d              64 filters
layer_norm; activation_relu

conv2d              64 filters
layer_norm; activation_relu
max_pooling2d

flatten
dense               512 units
layer_norm; activation_relu

dropout
classifier_head
```

MobileNetV1 (`mobilenet`). This is the standard shallow MobileNet architecture (Howard et al., 2017). The standard blocks in this architecture are either convolutional blocks with layer normalization and ReLU or depthwise separable convolutions with depthwise and pointwise layers followed by layer normalization and ReLU (see Figure 3 in (Howard et al., 2017)). In the shallow architecture, the number of convolutional blocks K is 7.

VGG16 (`vgg16`). This is the standard VGG16 architecture (Simonyan & Zisserman, 2015). The standard blocks in this architecture are convolutional layers with layer normalization and ReLU (Table 1 in (Simonyan & Zisserman, 2015)). The number of convolutional blocks K is 13.

ResNet50 (`resnet50`). This is the standard ResNet50 architecture (He et al., 2016b). The standard blocks in this architecture either identity blocks or convolutional blocks (see Table 1 in (He et al., 2016b)). The number of convolutional blocks K used in this architecture is 16.

C Size of the Weights

To calculate the norm of the weights while taking the contribution of the normalization layers into account, we compute the norm of the input to the classifier head (activations) for a random training sample of size 256. Then, we compute the Frobenius norm of the weights at the classifier head. Finally, we compute their product, which reflects the product of the Frobenius norm of layers stated in the generalization bound. Figure 6 shows a a Gaussian approximation to the ratio of the size of the weights of each reinitialization method over the size of the weights in the baseline. As shown in the figure, LW tends to maintain the size of the weights, while also boosting the margin on the training examples as discussed in the main paper.

D Ablation

LW includes rescaling, normalization, and reinitialization. In some cases, these may not all be required and reinitialization alone suffices, but this is not always the case. We observe a consistent improvement in LW when rescaling and normalization are included, in addition to fine-tuning the whole model at each round. In general:

- The improvement in generalization in LW cannot be attributed to rescaling or normalization alone. Reinitialization has the main effect.
- There exist experiment designs in which reinitialization fails without fine-tuning the model.
- We observe cases in which rescaling alone helps but adding reinitialization improves performance further.
- The gain from LW cannot be obtained by just training the baseline longer (i.e. using the same computational budget).

In this section, we show that the primary effect in LW comes from reinitialization, and that the improvement in generalization cannot be attributed to rescaling or normalization alone. We also show that fine-tuning the whole model performs better than freezing the early layers. Finally, we illustrate a case where LW without normalization fails.

Rescaling. In the synthetic data experiment in Appendix A, we show that rescaling improves performance compared to the baseline but adding re-initializations improves it further.

Reinitialization. We use the `vgg16` architecture with the same hyperparameters as listed in Appendix B. First, we observe that applying the same sequential process with rescaling and normalization but without reinitialization does not have a noticeable impact on the test accuracy. The test accuracy remains unchanged in all rounds. When reinitialization is added, we obtain the familiar looking curves where the test accuracy improves steadily with each round. This shows that the improvement in LW cannot be attributed to normalization or rescaling alone.

Fine-tuning vs. Freezing. LW fine-tunes the entire model in each round. One alternative approach is to *freeze* the early blocks. However, because of the co-adaptability between neurons that arises during training (Yosinski et al., 2014), freezing some layers and fine-tuning the rest is difficult to optimize and can harm its performance (Yosinski et al., 2014). This is also true for reinitialization methods in general. Hence, the entire model including the kept layers is fine-tuned at each round. In our experiments, we observe hyperparameter settings in which reinitialization with freezing drops the test accuracy to the random guessing level. Fine-tuning the whole model does not exhibit this behavior.

Training Longer. The improvement in LW cannot be obtained by simply training longer even with learning rate scheduling. Throughout our experiments (e.g. Tables 2), we also train the baseline longer to have the same number of training steps in total as reinitialization methods. Despite that, reinitialization methods improve performance considerably.