Weight Compander: A Simple Weight Reparameterization for Regularization

Rinor Cakaj
Image Processing
Robert Bosch GmbH & University of Stuttgart
71229 Leonberg, Germany
Rinor.Cakaj@de.bosch.com

Jens Mehnert
Image Processing
Robert Bosch GmbH
71229 Leonberg, Germany
JensEricMarkus.Mehnert@de.bosch.com

Bin Yang
ISS
University of Stuttgart
70550 Stuttgart, Germany
bin.yang@iss.uni-stuttgart.de

***Abstract*—Regularization is a set of techniques that are used to improve the generalization ability of deep neural networks. In this paper, we introduce *weight compander* (WC), a novel effective method to improve generalization by reparameterizing each weight in deep neural networks using a nonlinear function. It is a general, intuitive, cheap and easy to implement method, which can be combined with various other regularization techniques. Large weights in deep neural networks are a sign of a more complex network that is overfitted to the training data. Moreover, regularized networks tend to have a greater range of weights around zero with fewer weights centered at zero. We introduce a weight reparameterization function which is applied to each weight and implicitly reduces overfitting by restricting the magnitude of the weights while forcing them away from zero at the same time. This leads to a more *democratic* decision-making in the network. Firstly, individual weights cannot have too much influence in the prediction process due to the restriction of their magnitude. Secondly, more weights are used in the prediction process, since they are forced away from zero during the training. This promotes the extraction of more features from the input data and increases the level of weight redundancy, which makes the network less sensitive to statistical differences between training and test data. From an optimizational point of view, the second effect of WC can be seen as a reactivation of "dead" (near zero) weights to participate in the training. This increases the probability to find an ensemble of weights which performs better in the given task. We extend our method to learn the hyperparameters of the introduced weight reparameterization function. This avoids hyperparameter search and gives the network the opportunity to align the weight reparameterization with the training progress. We show experimentally that using weight compander in addition to standard regularization methods improves the performance of neural networks. Furthermore, we empirically analyze the weight distribution with and without weight compander after training to confirm the companding effects of our method on the weights.**

I. Introduction

Deep neural networks contain multiple non-linear hidden layers which make them powerful machine learning systems [1]. However, such networks are prone to overfitting [2] due to the limited size of training data, the high capacity of the model and the presence of to many noises in the training data [3]. Overfitting describes the phenomenon where a neural network (NN) perfectly fits the training data while achieving poor performance on the test data. Regularization is a set of techniques used to reduce overfitting and therefore a key element in deep learning [4]. It allows the model to generalize well to unseen data.

Many methods have been developed to regularize NNs, e.g. early stopping [5], weight penalties as L_1-regularization [6] and weight decay [7], soft weight sharing [8], dropout [1], data augmentation [9] and ensemble learning methods [10].

The architectures of NNs like ResNet [11], EfficientNet [12] or Transformer [13] differ widely across applications, but they typically share the same building blocks, e.g. weights, neurons and convolutional kernels.

We take a closer look at the weights of a NN. During the training, the weights are adjusted to decrease the loss function on the training set. The success of the optimization procedure depends on the parameterization of the weights [14]. The reparameterization of a weight $w \in \mathbb{R}$ is given by a reparameterization function $\Psi\colon \mathbb{R} \to \mathbb{R},\ v \mapsto \Psi(v)$ used to express the weight w in terms of a new learnable weight v by $w = \Psi(v)$. The idea to use reparameterization functions has been proposed before to prune or accelerate the training of deep NNs [14], [15].

In this work we introduce *weight compander*, a novel method using a reparameterization function to regularize general NNs, i.e. to improve their generalization ability.

Large weights in deep neural networks are a sign of a more complex network that is overfitted to the training data. Furthermore, regularized networks tend to have a greater range of weights around zero with fewer weights centered at zero [16].

Motivated by these observations, we propose a reparameterization function, which restricts the magnitude of the weights while forcing them away from zero at the same time. Therefore, our reparameterization function encourages the model to *not* rely on a few large weights but rather use many small weights to benefit from all the available information. Thus, our method promotes *weight democracy*, i.e. the influence of individual weights is limited and more weights are involved in the prediction process.

Our reparameterization is a *comp*ressor for weights with large magnitude and an exp*ander* for weights with small magnitude. Hence, our method is called *weight compander*.

The nonlinear reparameterization function $\Psi\colon \mathbb{R} \to \mathbb{R},\ v \mapsto a \cdot \arctan(\frac{v}{b})$ with hyperparameters $a, b > 0$, which are shared

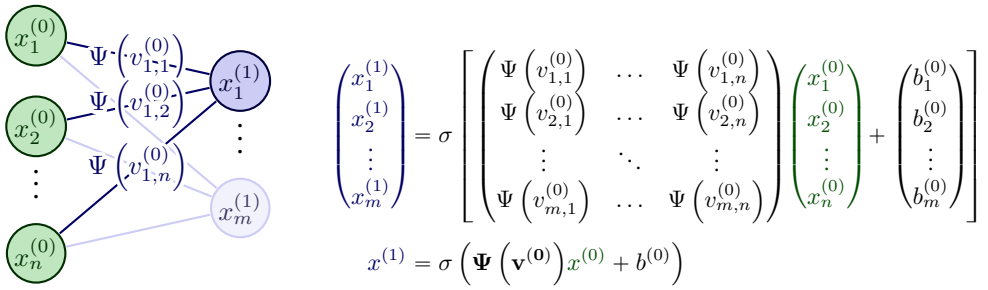


Fig. 1: Graphical illustration of the weight compander method applied to a fully connected network. We replace the weights w by $\Psi(v)$. Throughout the training v is adjusted to decrease the loss function on the training set.

across all weights in the NN, and new learnable weight $v \in \mathbb{R}$ meets our requirements. The parameter a is used to scale and restrict the absolute value of the weights w and the parameter b to squeeze and stretch the function in the v axis. The function is bounded, i.e. $-\frac{a\pi}{2} < a \cdot \arctan(\frac{v}{b}) < \frac{a\pi}{2}$ for all $v \in \mathbb{R}$. This restricts the magnitude of the weights. The derivative of $\Psi(v)$ is relatively larger for small $|v|$ than for large $|v|$. Since the derivative of $\Psi(v)$ appears in the gradient of v, this property is used during the training to force the weights away from zero.

In summary, weight compander reparameterizes each weight in a NN using the nonlinear function $\Psi(v)$. Throughout the training v is adjusted to decrease the loss function on the training set. A graphical illustration of a reparameterized fully connected NN is shown in Figure 1.

A. Contributions of this Paper and Applications

In this work we present *weight compander*, a novel method which reparameterize the weights w to reduce overfitting and which can be applied to any baseline NN. Our core contributions are:

- Defining a reparameterization function $\Psi(v)$ which regularizes NNs.
- Analyzing the impact of our method during the training process.
- Experimentally showing that using weight compander in addition to standard regularization methods increases the performance of different models on CIFAR-10, CIFAR-100, TinyImageNet and ImageNet. We improved the test accuracy of ResNet50 on CIFAR-10 by 0.75%, on CIFAR-100 by 1.56%, on TinyImageNet by 0.86% and on ImageNet by 0.22%.

II. Related Work

A. Regularization

Regularization is one of the key elements of deep learning [4], allowing the model to generalize well to unseen data even when trained on a finite training set or with an imperfect optimization procedure [17]. There are several techniques to regularize NNs which can be categorized in groups. Data augmentation methods like cropping, flipping, and adjusting brightness or sharpness [9] and cutout [18] transform the training dataset to avoid overfitting. Regularization techniques like dropout [1], dropblock [19] and dropconnect [2] drop neurons or weights from the NN during training to prevent units from co-adapting too much [1]. Furthermore, NNs can be regularized using penalty terms in the loss function. Weight decay [7] encourages the weights of the NN to be small in magnitude. The L_1-regularization [6] forces the weights of non-relevant features to zero. Normalization techniques [20]–[22] like batch normalization [20], which normalizes the features by the mean and variance computed within a mini-batch, act in some cases as regularizers. The resulting stochastic uncertainty of the batch statistics can benefit generalization [22] and in some cases eliminate the need for dropout [20]. Moreover, NNs can be regularized using early stopping [5] and ensemble learning methods [10]. Weight reparameterization has not been used for regularization yet. Therefore our introduced method is novel.

B. Weight Reparameterization

The idea of using reparameterization functions has been proposed before to prune NNs [15] or to accelerate their training [14]. Since these reparameterization functions have other goals, e.g. many low-magnitude weights to easily prune networks, they are not well suited for regularizing NN.

1) Pruning: Pruning methods are used to reduce the amount of connections in a network [23] with negligible performance reduction of the network [24].

Powerpropagation [15] is a weight reparameterization method, which focuses on the inherent effect of gradient based training on model sparsity. The idea is to reparameterize the weights w of a NN as $\Psi(v) = v \cdot |v|^{\alpha-1}$ for any $\alpha \geqslant 1$. Due to the chain rule of calculus, the magnitude of the parameters raised to $\alpha - 1$ appear in the gradient computation. Therefore, smaller magnitude parameters receive smaller gradient updates, while larger magnitude parameters receive larger updates. The network is then pruned by removing the low-magnitude weights.

Interspace pruning [25] is another pruning method which reparameterizes convolutional filters as a linear combinations of adaptive filter basis vectors. Let $h \in \mathbb{R}^{K \times K}$ be a filter with kernel size $K \times K$, then h can be also modeled in the linear space $\{\sum_{n=1}^{K^2} \lambda_n \cdot g^{(n)} \colon \lambda_n \in \mathbb{R}\}$ where $F := \{g^{(1)}, \dots, g^{(K^2)}\} \subset \mathbb{R}^{K \times K}$ is the filter basis. This method

sets pruned basis coefficients to zero while training un-pruned basis coefficients and basis vectors jointly, which leads to a state-of-the-art performance for unstructured pruning methods.

2) Accelerate Training of Deep Neural Networks: Reducing the training time for a NN with negligible performance reduction is an important task in deep learning. There are various approaches to accelerate the training, e.g. normalization methods like batch normalization [20], progressively freezing layers [26], prioritizing examples with high loss at each iteration [27] or dynamically pruning data samples [28].

Weight normalization [14] is a weight reparameterization approach that accelerates the convergence of SGD optimization. It reparameterizes the weight vectors of each layer such that the length of those weight vectors is decoupled from their direction. In detail, they express the weight vector $w \in \mathbb{R}^d$ by $w = \Psi(v, g) = \frac{g}{\|v\|} v$, where $v \in \mathbb{R}^d$ is the new weight vector and $g \in \mathbb{R}$ the scalar parameter. In contrast to earlier works [29], they directly perform SGD on the new weight vector v and scalar parameter g.

III. Weight Compander

Inspired by the weight reparameterizations for pruning [15] and faster training [14], we introduce *weight compander* to improve the generalization abilities of NNs. We present the reparameterization function and analyze how established deep learning practices can be combined with weight compander. Moreover, we extend our method by learning the new hyperparameters of weight compander with SGD instead of hyperparameter optimization.

A. Method

Our aim is to find a function $\Psi(v)$ to reparameterize the weights $w \in \mathbb{R}$ of a NN to reduce overfitting. In other words, we want to rewrite the weights w in an equivalent form $w = \Psi(v)$ and optimize the new weights $v \in \mathbb{R}$ to improve the performance of the NN.

To choose the optimal reparameterization function, we first analyze the effects of weight decay [7] and dropout [1] on the weights of a NN. Weight decay limits the growth of the weights and therefore prevents the weights from growing too large [7]. Dropout randomly omits hidden units from the network with a probability such that hidden units cannot rely on other hidden units being present. This prevents units from co-adapting too much [1]. Since hidden units are randomly omitted during the training, the NN cannot rely on a few weights. Therefore, the trained weights have a greater range around zero with fewer centered at zero [16].

Hence, we want to find a reparameterization function which strengthens the regularizing effects of both dropout and weight decay, i.e.

1) We want to restrict the magnitude of weights.
2) We want to have a greater range of weights around zero with fewer weights centered at zero.

Restricting the magnitude of weights restricts the decision-making power of individual weights. In contrast to weight decay [7] our method prevents large weights from the first epoch (see Section IV).

Having a greater range of weights around zero with fewer weights centered at zero yields an environment where more weights are involved with more and less equal contributions and more input information are used in the prediction process. Therefore our method promotes *weight democracy*, i.e. the NN should not rely on a few large weights but rather use many small weights to benefit from all the available information. In contrast to dropout [1], our method does not have a conflict with batch normalization [30]. Our method does not cause any "variance shiftes" that can harm the performance on DNNs using batch normalization (see Section IV). Moreover, our method can be used in addition to dropout (e.g. in VGGs in Section IV]), i.e. further increasing the performance of DNNs.

While the scope of this paper is regularization, from an optimization viewpoint, the second effect of WC can be seen as reactivating "dead" (near zero) weights to participate in the training, which otherwise would stay near zero during the training. This increases the probability to find an ensemble of weights which performs better in the given task.

At this point one could think that our method contradicts pruning for model reduction, e.g. [31], which claim that sparse models are superior in the sense of model complexity since they have a smaller amount of connections while negligible performance reduction of their networks. In Section IV, we analyze the weight distribution in networks with and without WC. Figure 4(b) shows that nearly 40% of the weights of the first convolutional layer in a trained ResNet34 are approximately 0 despite the usage of WC. Moreover, our analysis shows that the differences in the weight distribution between standard DNNs and networks using WC were less pronounced in the middle and at the end of the networks. Hence, WC does not force the network to use *all* weights. It stimulates the optimization process to use *more* weights in the training, to find the optimal ensemble of weights.

Note that the second demand is not in conflict with the aim of the weight decay, since weight decay only prevents weights from growing too large but does not force them to zero [7]. Although the L_1-regularization has a regularizing effect, generally it does not regularize as well as weight decay [32]. The L_1-regularization leads to a sparse weight matrix, i.e. more weights are zero. In contrast to that, weight decay leads to NNs which have a higher density of weights, i.e. weight decay only shrinks the weights close to zero, rather than being zero. These networks use more information from the input data for their prediction, which explains the increased performance compared to L_1-regularization.

To strengthen the regularizing effects, we introduce the following reparameterization function:

$$w = \Psi(v) = a \cdot \arctan\left(\frac{v}{b}\right) \tag{1}$$

for scalar values $a, b > 0$, which are shared across all weights in the NN. Figure 2 shows a plot of $\Psi(v)$ and its derivative. Since our method targets the weights of a NN, we do not reparameterize batch normalization layers or biases.

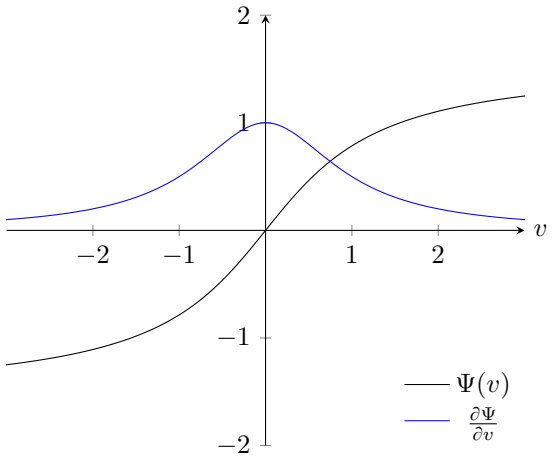


Fig. 2: The reparameterization function $\Psi(v)$ and the derivative of $\Psi(v)$ for $a = 1$ and $b = 1$, i.e. $\Psi(v) = \arctan(v)$.

The reparameterization restricts the magnitude of the weights to be in the interval $(-\frac{a\pi}{2}, \frac{a\pi}{2})$. Hence, the parameter a can be used to scale *and* restrict the absolute value of the weights. The gradient of the reparameterized loss $\mathcal{L}(\cdot, \Psi(v))$ w.r.t. v is

$$\frac{\partial \mathcal{L}(\cdot, \Psi(v))}{\partial v} = \frac{\partial \mathcal{L}(\cdot, \Psi(v))}{\partial \Psi(v)} \cdot \frac{\partial \Psi(v)}{\partial v} \tag{2}$$

$$= \frac{\partial \mathcal{L}(\cdot, \Psi(v))}{\partial \Psi(v)} \cdot \frac{a}{b \cdot (1 + \frac{v^2}{b^2})}. \tag{3}$$

The parameter b can be used to squeeze and stretch the function in the v axis, therefore changing the slope of the function, i.e. the gradient. Note that both parameters a and b have an effect on the gradient. In Equation 2, we see that our reparameterization adds the multiplicative factor $\frac{\partial \Psi(v)}{\partial v}$ to the derivative $\frac{\partial \mathcal{L}(\cdot, \Psi(v))}{\partial \Psi(v)}$. The derivative $\frac{\partial \Psi(v)}{\partial v}$ is relatively larger for small $|v|$ than for large $|v|$ (see Figure 2). Thus, weights near zero are updated relatively stronger than weights farther from zero. This promotes the weights near zero to have a greater range around zero with fewer weights centered at zero.

We now investigate how established deep learning practices can be used with weight compander. Since we want to compare standard NNs with reparameterized NNs, the initialization scheme and weight decay regularization have to be similar to standard NNs.

1) Initialization: Two important aspects of reliable training are the initialization of weights [33] and normalization layers such as batch-norm [20] or layer-norm [21]. Motivated by [15], we initialize v such that $\Psi(v)$ is equal to the original initialization of w (e.g. Kaiming initialization [33]). Hence, v is initialized by computing the inverse function of (1):

$$v = \Psi^{-1}(w) = b \cdot \tan\left(\frac{w}{a}\right). \tag{4}$$

This ensures that our weight reparameterization maintains all properties of typical initialization schemes. This initialization method is used in all experiments in Section IV.

In general, the weights of standard deep NNs are randomly initialized with small values, i.e. $-\frac{\pi}{2} \ll w \ll \frac{\pi}{2}$. Generally, Equation (4) is well defined for $a \geqslant 0.5$ since then $\frac{w}{a}$ is far away from the poles of the *tangent*. In our experiments, we found out that if $b > 0.5$, then $a < 0.5$ generally harms the performance. In the following, we will therefore assume that $a, b \geqslant 0.5$.

2) Weight Decay: Weight Decay is a regularization method applied to the weights of a NN [7]. Let $\mathbf{w}$ be a vector of all weights in a NN. Weight decay (or L_2-regularization) adds a penalty on the L_2-norm of the weights to the loss function, i.e.

$$\bar{\mathcal{L}}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}, \tag{5}$$

where λ is a value determining the strength of the penalty and $\mathbf{w}$ a weight vector containing all weights of a NN. To get comparable results in our experiments we apply weight decay on $w = \Psi(v)$ and *not* on v.

3) Impact of modern optimization Algorithms: Weight compander works fine with the SGD optimizer, since SGD computes the same gradient step as stated in Equation 2. However, it is important to analyze the impact of modern adaptive optimization algorithms [34]–[36] on our method. The Adam optimizer [34] computes moving averages of the gradient and the squared gradient to obtain an adaptive gradient step that is invariant to re-scaling of the gradient. Therefore, Adam and other adaptive optimization algorithms [34], [36] could reduce the effect of our method, as weight compander explicitly re-scales the gradient. To mitigate this issue, we introduce the *modified adaptive optimizer* from [15]. As stated in Section III-A, we can split the gradient $\frac{\partial \mathcal{L}}{\partial v}$ in two components $\frac{\partial \mathcal{L}}{\partial \Psi(v)}$ and $\frac{\partial \Psi(v)}{\partial v}$. The modified adaptive optimizer first computes the adaptive gradient step based on the raw gradient $\frac{\partial \mathcal{L}}{\partial \Psi(v)}$. Secondly, it multiplies $\frac{\partial \Psi(v)}{\partial v}$ to the computed adaptive gradient step to re-scale it. More details can be found in [15].

4) Inference: The inference time is the amount of time it takes for a NN to make a prediction on the test data. The reparameterized NNs have a slightly higher computational effort, since they need to evaluate the nonlinear weight reparameterization function for every weight in the NN.

However, this only affects training ($\approx$ 7%–14% runtime overhead depending on the size of the DNN) since we can get rid off the reparameterization function after the training has finished. We can copy the values $w = \Psi(v)$ for optimized v to another network of the same architecture. Instead of evaluating the function $\Psi(\cdot)$ for every weight v in the network, we directly save the values $\Psi(v)$ as weights. This yields a NN with the same inference time as the standard NN. Hence, weight compander only improves the generalization of NNs during training and has no negative impact to memory and computational complexity during inference.

B. Learnable Parameters

Until now we handled the parameters a and b as hyperparameters with fixed values set before the training. We now

present how these parameters can be learned by SGD. In this sense, a and b become learnable parameters instead of hyperparameters. Since we do not want to increase the number of parameters in the NN drastically, we will consider two different versions of *learnable weight compander*, which differ in their *granularity*. The *coarse* version shares the parameters a and b across all weights in the NN. In the *fine* version every layer shares a pair of parameters a and b.

In contrast to the standard weight compander with fixed hyperparameters a and b, we do *not* apply weight decay on $\Psi(v)$. Assume that weight decay is applied on $\Psi(v)$. The parameters a and b are learned through SGD, hence they are influenced by weight decay. Since a and b are shared across one layer or the whole network, weight decay influences many individual weights at the same time. We got better results applying weight decay only on the weights v and *not* on the parameters a and b. Sharing the parameters a and b across a layer or across the whole network implicitly regularizes a and b. We will provide empirical results in Section IV.

IV. Experiments

We experimentally validate the usefulness of our method in supervised image recognition, i.e. on CIFAR-10/100 [37], on TinyImageNet [38] and on ImageNet [39]. Furthermore, we analyze the differences in the weight distribution between NNs with and without reparameterization and confirm that weight compander indeed (i) restricts the magnitude of the weights and (ii) promotes the weights to have a greater range around zero with fewer weights centered at zero. We compare NNs using weight compander in addition to standard regularization methods against NNs using only standard regularization methods.

A. Image Classification on CIFAR-10/100

We evaluate the performance of WC on the CIFAR-10/100 classification dataset [37]. The CIFAR-10/100 dataset consists of 60,000 32×32 color images, containing 50,000 training images and 10,000 test images. We used various different network architectures, the CNN's ResNet18, ResNet34, ResNet50 [11], WideResNet-28-10 [40] and VGG16/19 with BN [41].

For our experiments we used PyTorch 1.10.1 [42] and one Nvidia GeForce 1080Ti GPU. The experiments were run five times with different random seeds for 200 epochs, resulting in different network initializations, data orders and additionally in different data augmentations. For every case we report the mean test accuracy and standard deviation. We used a 9/1-split between training examples and validation examples and saved the best model on the validation set. This model was then used for evaluation on the test dataset.

All networks were trained with a batch size of 128. We used the SGD optimizer with momentum 0.9 and initial learning rate 0.1. For ResNet18, ResNet34 and ResNet50 trained on CIFAR-10 we decayed the learning rate by 0.1 at epoch 90 and 136 and used weight decay with the factor $1e-4$ (as in [11]). For all networks trained on CIFAR-100 and for VGG16 with BN, VGG19 with BN, we decayed the learning rate by 0.2 at epoch 60, 120 and 160 and used weight decay with the factor $5e-4$ (as in [40]). For WideResNet-28-10 we used a dropout rate of 0.3. Dropout is also used in the fully connected layers in VGG16 with batch normalization and VGG19 with batch normalization. We want to point out that we have not done a hyperparameter search for the learning rate, weight decay, etc. We used the same setting as in [40].

TABLE I: Results of the hyperparameter search for the parameters a and b.

Model	CIFAR10	CIFAR100
ResNet18	$a = 1.0, b = 0.6$	$a = 0.6, b = 0.6$
ResNet34	$a = 0.8, b = 0.5$	$a = 0.7, b = 0.8$
ResNet50	$a = 1.0, b = 0.6$	$a = 1.0, b = 0.8$
VGG16	$a = 0.7, b = 0.7$	$a = 0.7, b = 1.0$
VGG19	$a = 0.5, b = 0.7$	$a = 0.7, b = 0.8$

The ResNet networks are initialized Kaiming-uniform [33]. In the VGG networks the linear layers are initialized Kaiming-uniform. The convolutional layers are initialized with a Gaussian normal distribution with mean 0 and standard deviation $\frac{2}{n}$, where n is the size of the kernel multiplied with the number of output channels in the same layer. In order to achieve a fair comparison, we initialize the weights v such that the initialization of $\Psi(v)$ is equal to the initialization of the standard network, as explained in Section III-A1.

For completeness, we compare weight compander with powerpropagation [15] and weight normalization [14] using ResNets on CIFAR-10/100.

1) Hyperparameters: For *every* network architecture and *both* datasets we first did a grid search for the parameters a and b (i.e for $a, b \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$) for one seed and picked the values that produced the best results. Then we conducted the experiments for the other seeds with the hyperparameters a and b from the first seed. The parameters a and b yielding the best results are shown in Table I. For WideResNet-28-10 trained on CIFAR-100 the parameters that produced the best results were $a = 0.7$ and $b = 0.7$.

2) Results: Table II and III present the results of our experiments on CIFAR-10 and CIFAR-100. The additions to the names of the models describe the reparameterization method, i.e. PP = Powerpropagation, WN = Weight Normalization, WC = Weight Compander. Our method improves the accuracy for all ResNets and VGGs. The additional gain in performance by reparameterizing the weights in the ResNet50 network are worth noting, i.e. $+0.75\%$ on CIFAR-10 and $+1.56\%$ on CIFAR-100. We observe that standard ResNet networks tend to generalize worse if more parameters are trainable. This can be counteracted by using weight compander.

Powerpropagation [15] allows parameters with larger magnitudes to adapt, while smaller magnitude parameters are restricted. Hence, low-magnitude parameters are largely unaffected by learning. This strong restriction lead to a decrease of the model performance.

Weight normalization [14] improves the conditioning of the optimization problem and speeds up convergence of SGD

TABLE II: Accuracy of experiments on CIFAR-10.

Model	Accuracy
ResNet18	$94.04\% \pm 0.08\%$
ResNet18 + PP	$92.94\% \pm 0.17\%$
ResNet18 + WN	$94.26\% \pm 0.14\%$
ResNet18 + WC	$\mathbf{94.44\% \pm 0.12\%}$
ResNet34	$93.69\% \pm 0.30\%$
ResNet34 + PP	$93.29\% \pm 0.16\%$
ResNet34 + WN	$94.17\% \pm 0.22\%$
ResNet34 + WC	$\mathbf{94.43\% \pm 0.25\%}$
ResNet50	$93.31\% \pm 0.36\%$
ResNet50 + PP	$93.47\% \pm 0.14\%$
ResNet50 + WN	$93.89\% \pm 0.23\%$
ResNet50 + WC	$\mathbf{94.06\% \pm 0.27\%}$
VGG16-BN	$93.27\% \pm 0.12\%$
VGG16-BN + WC	$\mathbf{93.52\% \pm 0.19\%}$
VGG19-BN	$93.21\% \pm 0.08\%$
VGG19-BN + WC	$\mathbf{93.33\% \pm 0.25\%}$

TABLE III: Accuracy of experiments on CIFAR-100.

Model	Accuracy
ResNet18	$76.47\% \pm 0.20\%$
ResNet18 + PP	$74.70\% \pm 1.32\%$
ResNet18 + WN	$\mathbf{76.83\% \pm 0.22\%}$
ResNet18 + WC	$76.73\% \pm 0.09\%$
ResNet34	$77.07\% \pm 0.45\%$
ResNet34 + PP	$74.70\% \pm 1.32\%$
ResNet34 + WN	$76.98\% \pm 0.39\%$
ResNet34 + WC	$\mathbf{77.54\% \pm 0.42\%}$
ResNet50	$76.17\% \pm 0.69\%$
ResNet50 + PP	$75.23\% \pm 0.25\%$
ResNet50 + WN	$76.10\% \pm 0.25\%$
ResNet50 + WC	$\mathbf{77.73\% \pm 0.40\%}$
WRN-28-10	$80.09\% \pm 0.14\%$
WRN-28-10 + WC	$\mathbf{80.24\% \pm 0.36\%}$
VGG16-BN	$72.48\% \pm 0.35\%$
VGG16-BN + WC	$\mathbf{72.74\% \pm 0.25\%}$
VGG19-BN	$71.34\% \pm 0.14\%$
VGG19-BN + WC	$\mathbf{71.86\% \pm 0.26\%}$

[14]. This slightly increases the performance compared to the baseline networks. However, weight compander outperforms weight normalization (except for ResNet18 on CIFAR-10). Especially for deep networks (ResNet34, ResNet50) trained on more complex tasks (CIFAR-100) weight compander performs much better than weight normalization.

B. Image Classification on TinyImageNet

TinyImageNet [38] contains 100,000 images of 200 classes (500 for each class) downsized to 64×64 colored images. Each class has 500 training images and 50 validation images.

For our experiments on TinyImageNet, we used PyTorch 1.10.1 [42] and one Nvidia GeForce 1080Ti GPU. The experiments were run three times with different random seeds for 300 epochs using a ResNet50. Following the common practice, we report the top-1 classification accuracy on the validation set. Analogously, to the experiments on CIFAR, we determined the optimal hyperparameters $a = 0.7$ and $b = 0.8$ for ResNet50. We trained all the models using the SGD optimizer with momentum 0.9 and initial learning rate 0.1 decayed by factor 0.1 at epochs 75, 150, and 225. Moreover, we used weight decay with the factor $1e-4$. The batch size is set to 128. The training data is augmented by using random crop with size 224, random horizontal flip and normalization. The validation data is augmented by resizing to 256×256, using center crop with size 224 and normalization.

TABLE IV: Top-1 Accuracy of experiments on TinyImageNet.

Model	Accuracy
ResNet50	$65.08\% \pm 0.38\%$
ResNet50 + WC	$\mathbf{65.93\% \pm 0.12\%}$

TABLE V: Top-1 Accuracy of experiments on ImageNet.

Model	Accuracy
ResNet50	$76.80\% \pm 0.11\%$
ResNet50 + WC	$\mathbf{77.02\% \pm 0.08\%}$

1) Results: Table IV presents the top-1 accuracy of our experiments on TinyImageNet. Using WC increases the accuracy of ResNet50 on TinyImageNet by 0.86%.

C. ImageNet

The ILSVRC 2012 classification dataset [39] contains 1.2 million training images, 50,000 validation images, and 150,000 testing images. Images are labeled with 1,000 categories.

For our experiments on ImageNet, we used PyTorch 1.10.1 [42] and 4 Nvidia GeForce 1080Ti GPU. The experiments were run three times with different random seeds for 300 epochs using ResNet50. Following the common practice, we report the top-1 classification accuracy on the validation set. Analogously, to the experiments on CIFAR and TinyImageNet, we determined the optimal hyperparameters $a = 0.6$ and $b = 0.7$. We trained all the models using the SGD optimizer with momentum 0.9 and initial learning rate 0.1 decayed by factor 0.1 at epochs 75, 150, 225. Moreover, we used weight decay with the factor $1e-4$. The batch size is set to 256. The training data and validation data is augmented as described for TinyImageNet.

1) Results: Table V presents the top-1 accuracy of our experiments on ImageNet. Using WC increases the accuracy of ResNet50 on ImageNet by 0.22%.

D. Weight Distribution

In the following, we analyze the effect of weight compander on the weight distribution of the networks. We experimentally confirm that weight compander strengthens the two phenomenons mentioned in Section III-A, i.e. (i) it restricts the magnitude of the weights and (ii) it forces the network to have a greater range of weights around zero with fewer weights centered at zero.

Setting $a = 0.8, b = 0.5$ produced the best results for ResNet34 on CIFAR-10 using weight compander in addition to standard regularization methods. Figure 3 shows the weight

distribution of the first convolutional layer of a ResNet34 trained on CIFAR-10 per epoch with and without weight compander. Note that both NNs have the same weight distribution at epoch 0. Figure 3(a) shows that the weights in a standard neural network can grow very large at the beginning of the training. Weight decay does not play an important role at the beginning of the optimization procedure due to the high training loss and therefore does not prevent the weights to grow very large in the first epochs. Figure 3(b) illustrates that weight compander bounds the maximal and minimal value of the weights and therefore forces the weights to stay in the interval $(-\frac{a\pi}{2}, \frac{a\pi}{2})$ during the training. Since each weight has a small magnitude, the influence of individual weighs in the prediction process is limited.

Standard NNs generally do not have very large weights at the end of the training. Since we restrict the magnitude of the weights from the beginning of the training, our method forces SGD to search from the first epoch for a solution with smaller weights. This also explains the increased performance of our method.

Figure 4 shows that weight compander encourages the weights to spread around zero, i.e. to have a greater range of weights around zero with fewer weights centered at zero. Therefore, more weights are involved in the prediction process. Due to the increased level of weight redundancy, the model is less sensitive to statistical differences between training and test data.

The weight distributions show that weight compander promotes *weight democracy* in NNs, i.e. the NN does not rely on a few large weights but rather use many small weights to benefit from all the available information.

The differences in weight distribution were less pronounced in the middle and at the end of the networks. Early convolutional layers detect low-level features such as edges and curves, while filters in higher layers are learned to encode more abstract features. Hence, reparameterized deep NNs extract more information from the input data than standard NNs. Nevertheless, each layer must be reparameterized to achieve the best performance.

E. Learnable Parameters

Since a hyperparameter search takes a lot of time and can be computationally extensive, we analyze the performance of *learnable weight regulation* where the parameters a and b are learned by SGD. The experiments were done in the same setting as described at the beginning of Section IV. We did a small hyperparameter search to find the optimal initialization values for a and b, i.e. $a, b \in \{0.5, 1.0\}$. As described in Section III-B, we compare two versions of learnable weight compander, i.e. the *fine* version where the parameters a and b are shared across each layer and the *coarse* version where the parameters are shared across all weights in the NN. However, the *fine* version performed significantly better than the *coarse* version. This implies that different layers in the network need different reparameterizations. Therefore, we only present the results of the *fine* version.

TABLE VI: Accuracy of learnable weight compander on CIFAR-10 for the *fine* version.

Model	Accuracy
ResNet18	$94.04\% \pm 0.08\%$
ResNet18 + WC	$\mathbf{94.16\% \pm 0.08\%}$
ResNet34	$93.69\% \pm 0.30\%$
ResNet34 + WC	$\mathbf{94.46\% \pm 0.07\%}$
ResNet50	$93.31\% \pm 0.36\%$
ResNet50 + WC	$\mathbf{94.80\% \pm 0.14\%}$

Initializing the *fine* version with $a = 0.5$ and $b = 1.0$ yields the best performance. Table VI shows the results for ResNets trained on CIFAR-10. In contrast to weight compander with fixed hyperparameters a and b, learnable weight compander could achieve significantly better performance on ResNet50.

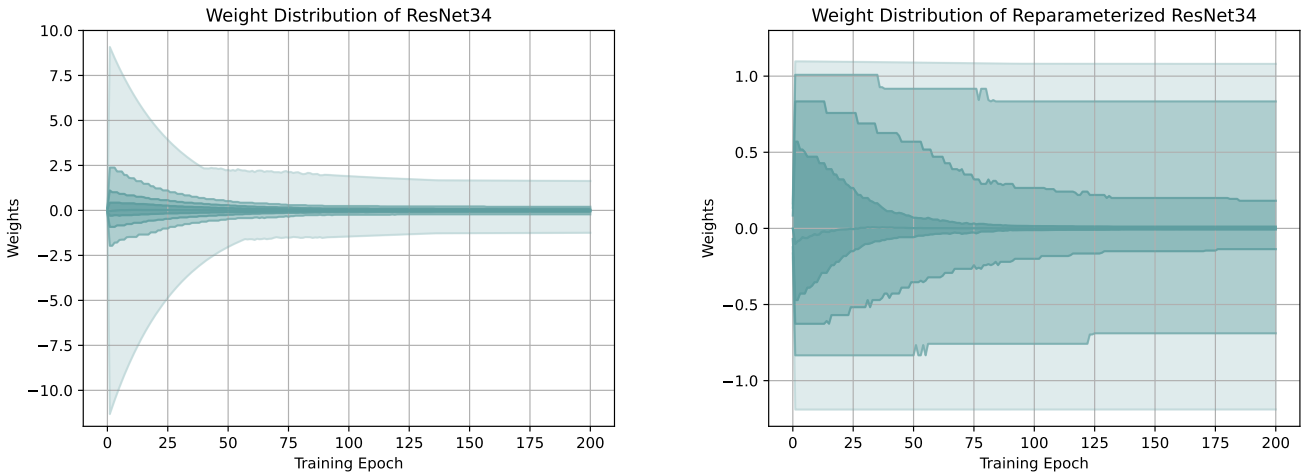
Training NNs requires minimizing a high-dimensional non-convex loss function. Different network architectures affect the loss landscape and therefore the generalization error and trainability [43]. In the case of learnable weight compander, SGD has the opportunity to change the reparameterization of each layer during the training. Therefore the loss landscape changes during the training. We think that learnable weight compander performs better since the optimization procedure has now the opportunity to affect the loss landscape. The performance on ResNet18 was slightly worse compared to the case with fixed hyperparameters. These results could be improved by finding the optimal hyperparameters for the learnable version.

We logged the values of the parameters a and b for each layer to understand their impact on the performance. The parameters a in the convolutional layers have not changed much compared to the linear layer. They stayed close to their initial values or moved in a neighborhood around their initial value (± 0.35). Interestingly, the parameter a in the last linear layer had a very unusual development. Their absolute values were very high, e.g. 8.5 for ResNet18 on one seed. Similar or even bigger values for a were observed on ResNet34 and ResNet50.

The parameters b have changed more in the optimization process. For the convolutional layers they were mostly in the interval $[0.5, 2.5]$ at the end of the training. Interestingly, the values for the parameters b were higher at earlier layers than at later layers in the network. Again, the linear layer is an exception where the parameters b took values from 0.5 to 11.15 over the five seeds. However, the network automatically converges to the desired distribution of weights described in the previous sections. For some layers the differences of the weight distribution were less pronounced.

F. Comparison with other S-shaped function

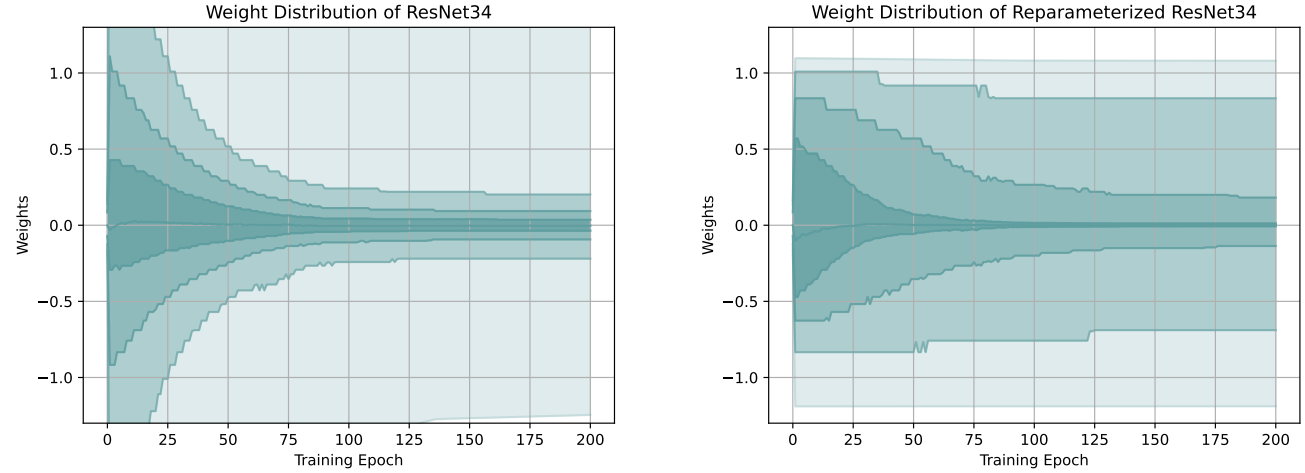
Additionally to our reparameterization function presented in the paper, we investigated other odd S-shaped functions, i.e. functions that have a rotational symmetry with respect to the origin. This means that their graph remains unchanged after rotation of 180 degrees about the origin. Note that it is not possible to use an arbitrary reparameterization function, e.g.



(a) Without weight compander

(b) With weight compander

Fig. 3: Weight distribution of w per epoch in the first convolutional layer of ResNet34 trained on CIFAR10 without weight compander 3(a) and with weight compander 3(b). From top to bottom, the lines represent percentiles with values: maximum, $93\%, 84\%, 69\%, 50\%, 31\%, 16\%, 7\%$, minimum.



(a) Without weight compander

(b) With weight compander

Fig. 4: Weight distribution of w per epoch in the first convolutional layer of ResNet34 on CIFAR10 without weight compander 4(a) (zoomed view of Figure 3(a)) and with weight compander 4(b).

using $w = \Psi(v) = v^2$ restricts the weights to be only positive and the derivative of v^2 grows for growing $|v|$. These effects harm the optimization process.

We had a look at the following other reparameterization functions $w = \Psi(v) = a \cdot \operatorname{arsinh}\left(\frac{v}{b}\right)$ and $w = \Psi(v) = a \cdot \operatorname{erf}\left(\frac{v}{b}\right)$ for $a, b > 0$. In contrast to the $\arctan$ and erf, arsinh is unbounded. We included it in our investigations since it is similar to S-shaped curves in the neighborhood around zero. Analogous to the experiments in Section IV in the paper, we did a grid search for every network and dataset for the parameters a and b (i.e for $a, b \in \{0.5, \ldots, 1.0\}$) for one seed and picked the values that produced the best results. Then we conducted the other experiments for the other seeds with the hyperparameters a and b from the first seed. We used the same settings as presented in Section IV.

Table VII present the results of ResNets trained on CIFAR-10 and CIFAR-100, which are reparameterized using Equation arsinh and Equation erf. Reparameterizing ResNets with arsinh or erf improves the accuracy for all ResNets compared to the baseline. However, reparameterizing the NNs with arsinh performs worse than reparameterizing the networks with $\arctan$ except for ResNet34 trained on CIFAR-10.

TABLE VII: Results of ResNets trained on CIFAR-10 and CIFAR-100 reparameterized with arsinh and erf.

Model	Accuracy CIFAR-10	Accuracy CIFAR-100
ResNet18_arcsinh	$94.34\% \pm 0.24\%$	$76.59\% \pm 0.25\%$
ResNet18_erf	$94.36\% \pm 0.26\%$	$\mathbf{76.80\% \pm 0.11\%}$
ResNet18 + WC	$\mathbf{94.44\% \pm 0.12\%}$	$76.73\% \pm 0.09\%$
ResNet34_arcsinh	$\mathbf{94.68\% \pm 0.39\%}$	$77.21\% \pm 0.23\%$
ResNet34_erf	$94.52\% \pm 0.14\%$	$77.26\% \pm 0.39\%$
ResNet34 + WC	$94.43\% \pm 0.25\%$	$\mathbf{77.54\% \pm 0.42\%}$
ResNet50_arcsinh	$93.82\% \pm 0.34\%$	$76.76\% \pm 0.42\%$
ResNet50_erf	$93.87\% \pm 0.30\%$	$77.51\% \pm 0.37\%$
ResNet50 + WC	$\mathbf{94.06\% \pm 0.27\%}$	$\mathbf{77.73\% \pm 0.40\%}$

G. Vanishing Gradients

We did not have vanishing gradients using WC. Our method is applied to the weights. Since most of the weights are distributed around zero (see Figure 4(b)), the gradient of the reparameterization function is for the most weights unequal to zero. Moreover, for some cases ($a > b$) the derivative of the reparameterization function evaluated at 0 is even greater than 1. It is important to note that our nonlinear *reparameterization* function should not be compared with nonlinear *activation* functions.

V. Conclusion and Future Work

We presented weight compander, a simple reparameterization of the weights in a NN to reduce overfitting. It is based on two premises (i) large weights are a sign of a more complex network that is overfitted to the training data and (ii) regularized networks tend to have a greater range of weights around zero with fewer weights centered at zero. Our proposed reparameterization function implicitly supports the network to restrict the magnitude of weights while forcing them away from zero at the same time. Therefore, our method promotes *weight democracy*, i.e. the influence of individual weights is limited and more weights are involved in the prediction process.

Using weight compander in addition to commonly used regularization methods (i.e. [1], [7], [9], [20], [44]) increased the performance of ResNets on CIFAR-10, CIFAR-100, Tiny-ImageNet and ImageNet and of VGGs on CIFAR-10 and CIFAR-100. We improved the test accuracy of ResNet50 on CIFAR-100 by 1.56%, on TinyImageNet by 0.86% and on ImageNet by 0.22%. Furthermore, we analyzed the effect of weight compander on the weight distribution. Finally, we presented learnable weight compander where the introduced hyperparameter are learned by SGD.

We have not explored the full range of possibilities of weight compander. Our future work will include analyzing the role of different reparameterization functions and further developing learnable weight compander. We did experiments with arsinh and erf where we could achieve similar performance boosts as for arctan. We will analysis the optimal weight distributions in DNNs to increase the performance gains of weight reparameterization methods. Moreover, we will investigate how weight reparameterization of DNNs can improve other tasks in deep learning. An extension of our introduced method could be to make the reparameterization more complex, i.e to have a correlation between different weights.

References

[1] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2670313

[2] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of Neural Networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org, 2013, pp. 1058–1066. [Online]. Available: http://proceedings.mlr.press/v28/wan13.html

[3] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, p. 022022, feb 2019. [Online]. Available: https://doi.org/10.1088/1742-6596/1168/2/022022

[4] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[5] Y. Yao, L. Rosasco, and A. Caponnetto, "On Early Stopping in Gradient Descent Learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, Aug. 2007. [Online]. Available: https://doi.org/10.1007/s00365-006-0663-2

[6] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: http://www.jstor.org/stable/2346178

[7] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," in *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, J. E. Moody, S. J. Hanson, and R. Lippmann, Eds. Morgan Kaufmann, 1991, pp. 950–957. [Online]. Available: http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization

[8] S. J. Nowlan and G. E. Hinton, "Simplifying Neural Networks by Soft Weight-Sharing," *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992. [Online]. Available: https://doi.org/10.1162/neco.1992.4.4.473

[9] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J. Big Data*, vol. 6, p. 60, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[10] D. W. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *J. Artif. Intell. Res.*, vol. 11, pp. 169–198, 1999. [Online]. Available: https://doi.org/10.1613/jair.614

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[12] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 6105–6114. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[14] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, p.

901. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/ed265bc903a5a097f61d3ec064d96d2e-Abstract.html

[15] J. Schwarz, S. Jayakumar, R. Pascanu, P. E. Latham, and Y. Teh, "Powerpropagation: A sparsity inducing weight reparameterisation," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 28 889–28 903. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/f1e709e6aef16ba2f0cd6c7e4f52b9b6-Paper.pdf

[16] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. Lille, France: JMLR.org, 2015, p. 16131622.

[17] J. Kukaka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," Oct. 2017.

[18] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," Aug. 2017.

[19] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "DropBlock: A regularization method for convolutional networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/7edcfb2d8f6a659ef4cd1e6c9b6d7079-Paper.pdf

[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[21] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450

[22] Y. Wu and K. He, "Group Normalization," *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 742–755, 2020. [Online]. Available: https://doi.org/10.1007/s11263-019-01198-w

[23] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 1135–1143. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html

[24] P. Wimmer, J. Mehnert, and A. Condurache, "FreezeNet: Full Performance by Reduced Storage Costs," in *Computer Vision - ACCV 2020 - 15th Asian Conference on Computer Vision, Kyoto, Japan, November 30 - December 4, 2020, Revised Selected Papers, Part VI*, ser. Lecture Notes in Computer Science, H. Ishikawa, C. Liu, T. Pajdla, and J. Shi, Eds., vol. 12627. Springer, 2020, pp. 685–701. [Online]. Available: https://doi.org/10.1007/978-3-030-69544-6_41

[25] ——, "Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 12 517–12 527. [Online]. Available: https://doi.org/10.1109/CVPR52688.2022.01220

[26] A. Brock, T. Lim, J. Ritchie, and N. Weston, "FreezeOut: Accelerate Training by Progressively Freezing Layers," Dec. 2017, nIPS 2017 Workshop on Optimization : 10th NIPS Workshop on Optimization for Machine Learning, NIPS ; Conference date: 08-12-2017.

[27] A. H. Jiang, D. L. Wong, G. Zhou, D. G. Andersen, J. Dean, G. R. Ganger, G. Joshi, M. Kaminsky, M. Kozuch, Z. C. Lipton, and P. Pillai, "Accelerating Deep Learning by Focusing on the Biggest Losers," *CoRR*, vol. abs/1910.00762, 2019. [Online]. Available: http://arxiv.org/abs/1910.00762

[28] R. S. Raju, K. Daruwalla, and M. H. Lipasti, "Accelerating Deep Learning with Dynamic Data Pruning," *CoRR*, vol. abs/2111.12621, 2021. [Online]. Available: https://arxiv.org/abs/2111.12621

[29] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org, 2013, pp. 1139–1147. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html

[30] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 2682–2690. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Li_Understanding_the_Disharmony_Between_Dropout_and_Batch_Normalization_by_Variance_CVPR_2019_paper.html

[31] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage

[32] R. Ma, J. Miao, L. Niu, and P. Zhang, "Transformed 1 regularization for learning sparse deep neural networks," *Neural Networks*, vol. 119, pp. 286–298, 2019. [Online]. Available: https://doi.org/10.1016/j.neunet.2019.08.015

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034. [Online]. Available: https://doi.org/10.1109/ICCV.2015.123

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[35] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2021068

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[37] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[38] Y. Le and X. S. Yang, "Tiny ImageNet Visual Recognition Challenge," 2015.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[40] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[41] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[43] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf

[44] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7