This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Centered Weight Normalization in Accelerating Training of Deep Neural Networks

Lei Huang[†] Xianglong Liu[†*] Yang Liu[†] Bo Lang[†] Dacheng Tao[‡]
[†]State Key Laboratory of Software Development Environment, Beihang University, P.R.China
[‡]UBTECH Sydney AI Centre, School of IT, FEIT, The University of Sydney, Australia
{huanglei, xlliu, blonster, langbo}@nlsde.buaa.edu.cn, dacheng.tao@sydney.edu.au

Abstract

Training deep neural networks is difficult for the pathological curvature problem. Re-parameterization is an effective way to relieve the problem by learning the curvature approximately or constraining the solutions of weights with good properties for optimization. This paper proposes to re-parameterize the input weight of each neuron in deep neural networks by normalizing it with zero-mean and unit-norm, followed by a learnable scalar parameter to adjust the norm of the weight. This technique effectively stabilizes the distribution implicitly. Besides, it improves the conditioning of the optimization problem and thus accelerates the training of deep neural networks. It can be wrapped as a linear module in practice and plugged in any architecture to replace the standard linear module. We highlight the benefits of our method on both multi-layer perceptrons and convolutional neural networks, and demonstrate its scalability and efficiency on SVHN, CIFAR-10, CIFAR-100 and ImageNet datasets.

1. Introduction

Recently, deep neural networks have achieved grand success across a broad range of applications, *e.g.,* image classification, speech recognition and object detection [33, 35, 14, 36, 7, 25]. Neural networks typically are composed of stacked layers, and the transformation between layers consists of linear mapping with learnable parameters, in which each neuron computes a *weighted sum* over its inputs and adds a *bias* term, and followed by an element-wise nonlinear activation. The stacked structure endows a neural network learning feature hierarchies with features from higher levels formed by the composition of lower level features. Further, deep architectures provide neural networks powerful representation capacity of learning complicated functions that can represent high-level abstractions.

*Corresponding author

While the deep and complex structure enjoys appealing advantages, it also makes learning difficult. Indeed, many explanations for the difficulty of deep learning have been explored, such as the problem of vanishing and exploding gradients [2] , internal covariate shift [16], and the *pathological curvature* [20]. To address these problems, various studies such as finely weight initialization [19, 9, 13, 29, 21], normalization of internal activation [16, 4], and sophistic optimization methods have been proposed accordingly [20, 12, 11].

Our work is dedicated to the problem of *pathological curvature* [20], *i.e.* the condition number of the Hessian matrix of the objective function is low at the optimum regions [28], which makes learning extremely difficult via first order stochastic gradient descent. Several studies [12, 11] recently have attempted to use the pre-conditioning techniques to improve the conditioning of the cost curvature. However, these methods introduce too much overhead and are not convenient to be applied.

An alternative track to facilitate the optimization progress is the transformation of parameterization space of a model [1], which is called re-parameterization. The motivation of re-parameterization is that there may be various equivalent ways to parameterize the same model, some of which are much easier to optimize than others [28]. Therefore, exploring good ways of parameterizing neural networks [6, 28] are essentially important in training deep neural networks.

Inspired by the practical trick that weights are sampled from a distribution with zero mean and a standard deviation for initialization [19, 9, 13], in this paper we propose to constrain the *input weight* of each neuron with zero mean and unit norm by re-parameterization during the course of training, followed by a learnable scalar parameter to adjust the norm of the *input weight*. We use proxy parameters and perform gradient updating on these proxy parameters by back-propagating the gradient information through the normalization process (Figure 1). By introducing this process, the summed input of each neuron is more likely to possess the

properties of zero mean and stable variance. Besides, this technique can effectively improve the conditioning of the optimization problem and thus can accelerate the training of deep neural networks.

We wrap the proposed re-parameterization method into a linear module in practice, which can be plugged in any architecture as a substitute for the standard linear module. The technique we present is generic and can be applied to a broad range of models. Our method is also orthogonal and complementary to recent advances in deep learning, such as *Adam* optimization [17] and batch normalization [16]. We conduct comprehensive experiments on Yale-B, SVHN, CIFAR-10, CIFAR-100 and ImageNet datasets over multi-layer perceptron and convolutional neural network architectures. The results show that centered weight normalization draws its strength in improving the performance of deep neural networks. Our code is available at https://github.com/huangleiBuaa/CenteredWN.

2. Related work

Training deep neural network via first order stochastic gradient descent is difficult in practice, mainly due to the *pathological curvature* problem. Several works have tried the preconditioning techniques to accelerate the training. Martens and Sutskever [20] developed a second-order optimization method based on the *Hessian-free* approach. Other studies [12, 11] explicitly pre-multiply the cost gradient by an approximate inverse of the *Fisher information matrix*, thereby expecting to obtain an approximate *natural gradient*. The approximate inverse can be obtained by using Cholesky factorization [12] or Kronecker-factored approximation [11]. These methods usually introduce much overhead. Besides, their optimization procedures are usually coupled with the preconditioning, and thus can not be easily applied.

Some works addressed the benefits of centered activations [36] and gradients [26]. They show that these transformations make the *Fisher information matrix* approximate block diagonal, and improve the optimization performance. *Batch normalization* [16] further standardizes the activation with centering and scaling based on mini-batch, and includes normalization as a part of the model architecture. Ba *et al.* [4] computed the layer normalization statics over all the hidden units in the same layers, targeting at the scenario that the size of mini-batch is limited. These methods focus on normalizing the activation of the neurons explicitly while our method works by re-parameterizing the model, and is expected to achieve better conditioning and stabilize the activations implicitly.

Re-parameterization is an effective technique to facilitate the optimization progress [6, 28]. Guillaume *et al.* [6] tried to estimate the expected projection matrix, and implicitly whitening the activations. However, the operation of

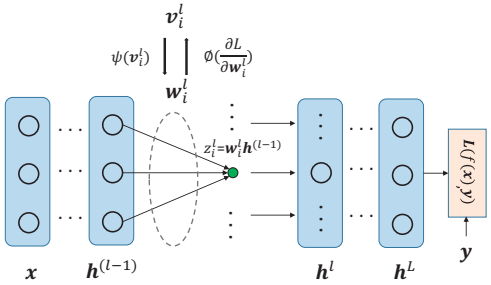


Figure 1. An illustrative example of layered neural networks with re-parameterization (for brevity, we leave out the bias nodes). The proposed re-parameterization method finely constructs a transformation ψ over the proxy parameter $\mathbf{v}$ to ensure that the transformed weight $\mathbf{w}$ has certain beneficial properties for the training of neural network. Gradient updating is executed on the proxy parameter $\mathbf{v}$ by back-propagating the gradient information through the normalization process.

updating the projection matrix is still coupled with the optimization. Salimans and Kingma [28] designed weight normalization [28] as a part of the model architecture. The proposed weight normalization addresses the normalization of the *input weight* for each neuron and decouples the length of those weight vectors from their directions. Our work further powers *weight normalization* by centering the *input weight*, and we argue that it can ulteriorly improve the conditioning and speed up the training for deep neural networks.

There exist studies constructing orthogonal matrix in recurrent neural networks (RNN) [2, 37, 8] by using re-parameterization to avoid the gradient vanish and explosion problem. However, these methods are limited for the hidden to hidden transformation in RNN, because they require the weight matrix to be square matrix. Other alternative methods [5, 38] focus on reducing the storage and computation costs by re-parameterization. Different from them, our work aims to design a general re-parameterized linear module to accelerate training and improve the performance of deep neural networks, and make it an alternative for the standard linear module.

3. Centered weight normalization

We follow the matrix notation that the vector is in column form, except that the derivative is a row vector. Given training data $D = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, ..., M\}$ where $\mathbf{x}$ denotes the input and $\mathbf{y}$ the target. A neural network is a function $f(\mathbf{x}; \theta)$ parameterized by θ that is expected to fit well the training data and have good generalization for the test data. The function $f(\mathbf{x}; \theta)$ adopted by neural network usually consists of stacked layers. The transformation between layers consists of a linear mapping $\mathbf{z}^l = (\mathbf{W}^l)^T \mathbf{h}^{l-1} + \mathbf{b}^l$ with learnable parameters $\mathbf{W}^l \in \mathbb{R}^{d \times n}$ and $\mathbf{b}^l \in \mathbb{R}^n$, and followed by an element-wise nonlinearity: $\mathbf{h}^l = \varphi(s^l)$, where $l \in \{1, 2, ..., L\}$ indexes the layer and L denotes the

total number of layers. By convention, $\mathbf{h}^0$ corresponds to the input $\mathbf{x}$ and $\mathbf{h}^L$ corresponds to the output of the network $f(\mathbf{x};\theta)$. For clarification, we refer to $\mathbf{z}$ and $\mathbf{h}$ as pre-activation and activation respectively. Under the denotation, the learnable parameters are $\theta = \{\mathbf{W}^l, \mathbf{b}^l | l = 1, 2, \ldots, L\}$.

Training the neural networks can be viewed as tuning the parameters to minimize the discrepancy between the desired output $\mathbf{y}$ and the predicted output $f(\mathbf{x};\theta)$. This discrepancy is usually described by a loss function $\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))$, and thus the objective is to optimize the parameters θ by minimizing the loss as follows:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}[\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))]. \tag{1}$$

Stochastic gradient descent (SGD) has proved to be an effective way to train deep networks, in which the gradient of the loss function with respect to the parameters $\frac{\partial \mathcal{L}}{\partial \theta}$ is approximated by the mini-batch $\mathbf{x}_{1\ldots m}$ of size m at each iteration, by computing $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{m}\Sigma_{i=1}^{m}\frac{\partial \mathcal{L}(\mathbf{y}_i, f(\mathbf{x}_i;\theta))}{\partial \theta}$.

3.1. Methodology

Despite the fact that SGD can guarantee a local optimum, it is also well known the practical success of SGD is highly dependent on the curvature of the objective to be optimized. Deep neural network usually exhibits *pathological curvature* problem, which makes the learning difficult. An alternative track to relieve the pathological curvature issue and thus facilitate the optimization progress is the transformation of parameterization space of a network model, which is called re-parameterization.

In the literatures, there exist practical tricks for weight initialization where weights are sampled from a distribution with zero-mean and a standard deviation [19, 9, 13]. These initialization techniques effectively can avoid exponentially reducing/magnifying the magnitudes of input signals or back-propagation signals in the initial phases, by constructing stable and proper variances of the activations among different layers. Motivated by this observation, we propose to constrain the *input weight* of each neuron with zero mean and unit norm during the course of training by re-parameterization, which enjoys stable and fast training of deep neural networks.

For simplicity, we start by considering one certain neuron i in layer l, whose pre-activation $z_i^l = (\mathbf{w}_i^l)^T\mathbf{h}^{(l-1)} + b_i^l$. We denote $\mathbf{w}_i^l$ as the *input weight* of the neuron, as shown in Figure 1. We have left out the superscript l and subscript i for brevity in the following discusses.

Standardize weight We first re-parameterize the *input weight* $\mathbf{w}$ of each neuron and make sure that it has the following properties: (1) *zero-mean*, *i.e.* $\mathbf{w}^T\mathbf{1} = 0$ where $\mathbf{1}$ is a column vector of all ones; (2) *unit-norm*, *i.e.* $\|\mathbf{w}\| = 1$ where $\|\mathbf{w}\|$ denotes the Euclidean norm of $\mathbf{w}$. To achieve the goal, we express the *input weight* $\mathbf{w}$ in terms of the proxy parameter $\mathbf{v}$ (Figure 1) using

$$\mathbf{w} = \frac{\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})}{\|\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})\|} \tag{2}$$

where d is the dimension of the *input weight*, and the stochastic gradient descent or other alternative techniques such as Adam [17] can be directly applied to the optimization with respect to the proxy parameter $\mathbf{v}$. We refer to the proposed re-parametrization as *centered weight normalization*, that is, we center and scale the proxy parameter $\mathbf{v}$ to ensure that the *input weight* $\mathbf{w}$ has the desired *zero-mean* and *unit-norm* properties as discussed before.

In our *centered weight normalization*, the parameter updating is completed based on the proxy parameters, and thus the gradient signal should back-propagate through the normalization process. Specifically, given the derivative of $\partial\mathcal{L}/\partial\mathbf{w}$, we can get the derivative of loss with respect to the proxy parameter $\mathbf{v}$ as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{1}{\|\hat{\mathbf{v}}\|}[\frac{\partial \mathcal{L}}{\partial \mathbf{w}} - (\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\mathbf{w})\mathbf{w}^T - \frac{1}{d}(\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\mathbf{1})\mathbf{1}^T] \tag{3}$$

where $\hat{\mathbf{v}} = \mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})$ is the centered auxiliary parameter.

Adjustable weight scale Our method can be viewed effectively as a solution to the constrained optimization problem over neural networks:

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}[\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))] \\ &s.t. \quad \mathbf{w}^T\mathbf{1} = 0 \; and \; \|\mathbf{w}\| = 1 \end{aligned} \tag{4}$$

where $\mathbf{w}$ indicates the *input weight* for each neuron in each layer. Therefore, we regularize the network with $2n$ constraints in each layer where n is the number of filters in certain layer and the optimization is over the embedded submanifold of the original weight space. While these constraints provide regularization, they also may reduce the

Algorithm 1 Forward pass of linear mapping with centered weight normalization.

1: **Input**: the mini-batch input data $\mathbf{X} \in \mathbb{R}^{d\times m}$ and parameters to be learned: $\mathbf{g} \in \mathbb{R}^{n\times 1}$, $\mathbf{b} \in \mathbb{R}^{n\times 1}$, $\mathbf{V} \in \mathbb{R}^{d\times n}$.
2: **Output**: pre-activation $\mathbf{Z} \in \mathbb{R}^{n\times m}$.
3: compute centered weight: $\hat{\mathbf{V}} = \mathbf{V} - \frac{1}{d}\mathbf{1}_d(\mathbf{1}_d^T\mathbf{V})$.
4: **for** $i = 1$ to n **do**
5: calculate normalized weight with respect to the i-th neuron: $\mathbf{w}_i = \frac{\hat{\mathbf{v}}_i}{\|\hat{\mathbf{v}}_i\|}$
6: **end for**
7: calculate: $\hat{\mathbf{Z}} = \mathbf{W}^T\mathbf{X}$.
8: calculate pre-activation: $\mathbf{Z} = (\mathbf{g}\mathbf{1}_m^T) \odot \hat{\mathbf{Z}} + \mathbf{b}\mathbf{1}_m^T$.

Algorithm 2 Back-propagation pass of linear mapping with centered weight normalization.

Input: pre-activation derivative $\{\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \in \mathbb{R}^{n\times m}\}$. Other auxiliary variables from respective forward pass: $\hat{\mathbf{V}}$, $\mathbf{W}$, $\hat{\mathbf{Z}}$, $\mathbf{X}$, $\mathbf{g}$.
Output: the gradients with respect to the inputs $\{\frac{\partial \mathcal{L}}{\partial \mathbf{X}} \in \mathbb{R}^{d\times m}\}$ and learnable parameters: $\frac{\partial \mathcal{L}}{\partial \mathbf{g}} \in \mathbb{R}^{1\times n}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \in \mathbb{R}^{1\times n}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{V}} \in \mathbb{R}^{d\times n}$.
$\frac{\partial \mathcal{L}}{\partial \mathbf{g}} = \mathbf{1}_m^T(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \odot \hat{\mathbf{Z}})^T$
$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \mathbf{1}_m^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}}^T$
$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \odot (\mathbf{g}\mathbf{1}_m^T)$
$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \mathbf{W}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{X}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}}^T$
for $i = 1$ to n **do**
  $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i} = \frac{1}{\|\hat{\mathbf{v}}_i\|}(\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} - (\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}\mathbf{w}_i)\mathbf{w}_i^T - \frac{1}{d}(\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}\mathbf{1}_d)\mathbf{1}_d^T)$
end for

representation capacity of the networks. To address it, we simply introduce a learnable scalar parameter g to fine tune the norm of $\mathbf{w}$. Similar idea has been introduced in [28], and proved useful in practice. We initialize it to 1, and expect the network learning process can find a proper scalar under the supervision signals. To summarize, we rewrite the pre-activation z of each neuron in the following way

$$z = g(\frac{\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})}{\|\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})\|})^T\mathbf{h} + b. \tag{5}$$

Wrapped module with re-parameterization We can wrap our proposed method as a common module and plug-in it in the neural networks as a substitute for the ordinary linear module. To achieve this goal, the key is to implement the forward and back-propagation passes. Based on Eqn. (5), it is readily to extend for multiple output neurons of size n. We describe the details of the forward pass in Algorithm 1, and the back-propagation pass in Algorithm 2. In the algorithms, the '$\odot$' operator represents element-wise matrix multiplication, $\mathbf{1}_d$ indicates a column vector of all ones of size d, and $\mathbf{X} \in \mathbb{R}^{d\times m}$ is the mini-batch input data feeded into this module, where d is the dimension of the input and m is the size of the mini-batch. $\mathbf{v}_i$ is the i-th column of $\mathbf{V}$, and $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n)$.

Convolutional layer For the convolutional layer, the weight of each feature map is $\mathbf{w}_c \in \mathbb{R}^{d\times F_h\times F_w}$ where F_h and F_w indicate the height and width of the filter, and d is the number of the input feature maps. We unroll $\mathbf{w}_c$ as a F_hF_wd dimension vector $\mathbf{w}$, then the same normalization can be directly executed over the unrolled $\mathbf{w}$.

3.2. Analysis and discussions

Next, we will show our centered weight normalization enjoys certain appealing properties. It can stabilize the distribution of pre-activation z with respect to each neuron. To illustrate this point, we introduce the following proposition where we omit the bias term b to simplify the discussion.

Proposition 1. *Let* $z = \mathbf{w}^T\mathbf{h}$, *where* $\mathbf{w}^T\mathbf{1} = 0$ *and* $\|\mathbf{w}\| = 1$. *Assume* $\mathbf{h}$ *has Gaussian distribution with the mean:* $\mathbb{E}_\mathbf{h}[\mathbf{h}] = \mu\mathbf{1}$, *and covariance matrix:* $cov(\mathbf{h}) = \sigma^2\mathbf{I}$, *where* $\mu \in \mathbb{R}$ *and* $\sigma^2 \in \mathbb{R}$. *We have* $\mathbb{E}_z[z] = 0$, $var(z) = \sigma^2$.

The proof of this proposition is provided in the supplementary materials. Such a proposition tells us that for each neuron the pre-activation z has zero-mean and the same variance as the activations fed in, when the assumption is satisfied. This property does not strictly hold in practice due to the issue of nonlinearities in the hidden layers, however, we still empirically find that our proposed centered weight normalization approximately holds.

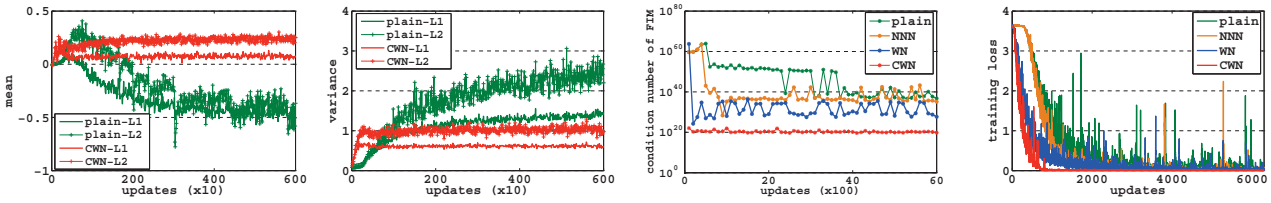
Note that our method is similar to batch normalization [16]. However, batch normalization focuses on normalizing the pre-activation compulsively such that the pre-activation of current mini-batch data is zero-mean and unit-variance, which is a data dependent normalization. Our method normalizes the weights and is expected to have the effect of zero-mean and stable variance implicitly, which is a data independent normalization. Actually, our method can work well by combining batch normalization as described in subsequent experiments.

Besides the good property guaranteed by Proposition 1, another remarkable observation is that the proposed re-parameterization method can make the optimization problem easier, which is supported by the following proposition.

Proposition 2. *Regarding to the proxy parameter* $\mathbf{v}$, *centered weight normalization makes that the gradient* $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$ *has following properties: (1)* zero-mean, i.e. $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} \cdot \mathbf{1} = 0$; *(2) orthogonal to the parameters* $\mathbf{w}$, i.e. $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} \cdot \mathbf{w} = 0$.

The derivative of Proposition 2 is also given in the supplementary materials. The *zero-mean* property of $\partial\mathcal{L}/\partial\mathbf{v}$ usually has an effect that the leading eigenvalue of the Hessian is smaller, and therefore the optimization problem is likely to become better-conditioned [30]. This promises the network to learn at a higher rate, and hence converge much faster. Meanwhile, the gradient is orthogonal to the parameters $\mathbf{w}$, and therefore is orthogonal to $\hat{\mathbf{v}}$. Following the analysis in [28], the gradient $\partial\mathcal{L}/\partial\mathbf{v}$ can self-stabilize its norm, which thus makes optimization of neural networks robust to the value of the learning rate.

Computation complexity Given mini-batch input data $\{\mathbf{X} \in \mathbb{R}^{d\times m}\}$, regarding n neurons, both the forward pass (Algorithm 1) and back-propagation pass (Algorithm 2) of



(a) (b) (c) (d)

Figure 2. A case study on Yale-B dataset ('-Ll' indicates the l-th layer). With respect to the training updates, we analyze (a) the mean over the mini-batch data and all neurons; (b) the variance calculated over the mini-batch data and all neurons; (c) the condition number (log-scale) of relative FIM in the second last layer; and (d) the training loss.

the centered weight normalization have the computational complexity of $O(mnd + nd)$. This means that our centered weight normalization has the same computational complexity as the standard linear module, since the extra cost $O(nd)$ of centered weight normalization is negligible to $O(mnd)$.

For a convolution layer with filters $\mathbf{W} \in \mathbb{R}^{n\times d\times F_h\times F_w}$, given m mini-batch data $\{\mathbf{x}_i \in \mathbb{R}^{d\times h\times w}, i = 1, 2, ..., m\}$, where h and w represent the height and width of the feature map, the computational complexity of the standard convolution layer is $O(nmdhwF_hF_w)$, and while our proposed method is $O(mndhwF_hF_w + ndF_hF_w)$. The extra cost $O(ndF_hF_w)$ of the proposed normalization is also negligible compared to the convolution operation.

4. Experiments

In this section, we begin with a set of ablation study to support the preliminary analysis of the proposed method. Then in the following three subsections, we show the effectiveness of our proposed method on the general network architectures, including (1) Multi-Layer Perceptron (MLP) architecture for face and digit recognition tasks; and (2) Convolutional Neural Network (CNN) models for large scale image classification.

Experimental protocols For all experiments, we adopt ReLUs [22] as the nonlinear activation and the negative log-likelihood as the loss function, where $(\mathbf{x}, y)$ represents the (input image, target class) pair. We choose the random weight initialization by default as described in [19] if we do not specify weight initialization methods. For the experiments on MLP architecture, the input images are resized and transformed to 1,024 dimensional vectors in gray scale, with mean subtracted and variance divided.

4.1. Case study

As discussed in Section 3.2, the proposed Centered Weight Normalization (CWN) method can stabilize the distribution of the pre-activations during training under certain assumptions. A network with the proposed re-parameterization technique is likely to become better-conditioned. In this part, we conduct experiments to validate the conclusions empirically.

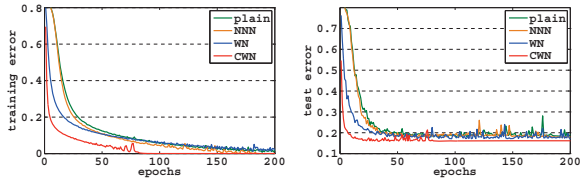
We conduct the experiments on Yale-B dataset[1] for face recognition task. Yale-B dataset has 2,414 images with 38 classes. We randomly sample 380 images (*i.e.*, 10 images per class) as the test set and the others as training set, where all images are resized as 32×32 pixels. We train a 5-layer MLP with $\{128, 64, 48, 48\}$ neurons in the hidden layers. We compare our CWN with the plain network without re-parameterization (named 'plain' for short). In the training, we use *stochastic gradient descent* with batch size of 32. For each method, the best results by tuning different learning rates in $\{0.1, 0.2, 0.5, 1\}$ are reported.

Stabilize activation We investigate the mean and variance of pre-activations in each layer during the course of training. Figure 2 (a) and (b) respectively show the mean and variance from the first and second layers, with respect to different rounds of training updates. The mean and variance are calculated over the mini-batch data and all neurons[2] in the corresponding layers. We find that the distribution of pre-activation in plain network varies remarkably. For instance, the mean decreases and the variance increases gradually, even though the network nearly converges to the minimal loss. The unstable distribution of pre-activation may result in an unstable learning, as shown in Figure 2 (d), where the loss of plain network fluctuates sharply. By comparing the loss curve of CWN and the others, we find that the pre-activation of the network with CWN re-parameterization has a stable mean and variance as shown in Figure 2 (a) and (b). Especially, in the first layer the pre-activation is nearly zero-mean, which empirically supports the fact in Proposition 1. These beneficial properties make the network with CWN re-parameterization converge to an optimum regions in a stable and fast way.

Conditioning analysis The condition number of the Fisher Information Matrix (FIM) is a good indicator to show whether the optimization problem is easy to solve. FIM is defined as $\mathbf{F}_\theta = \mathbf{E}_{\mathbf{x}\sim\pi}\{\mathbf{E}_{y\sim P(y|\mathbf{x},\theta)}[(\frac{\partial \log P(y|\mathbf{x},\theta)}{\partial\theta})(\frac{\partial \log P(y|\mathbf{x},\theta)}{\partial\theta})^T]\}$,

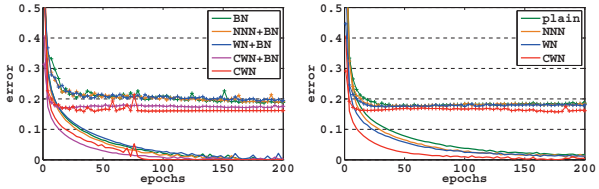
[1] http://vision.ucsd.edu/ leekc/ExtYaleDatabase/ExtYaleB.html

[2] We also observe the mean and variance of the randomly selected neurons in each layer, which also have the similar behaviours.



(a) training error (b) test error

Figure 3. Performances evaluation on MLP architecture over permutation-invariant SVHN.



(a) combining batch normalization (b) using Adam optimization

Figure 4. Comparison of different methods by (a) combining batch normalization and (b) using Adam optimization on permutation-invariant SVHN. We evaluate training error (solid lines) and test error (lines marked with plus) with respect to the training epochs.

and its condition number is $cond(\mathbf{F}_\theta) = \frac{|\lambda(\mathbf{F}_\theta)|_{max}}{|\lambda(\mathbf{F}_\theta)|_{min}}$ where $|\lambda(\mathbf{F}_\theta)|_{max}$ and $|\lambda(\mathbf{F}_\theta)|_{min}$ are the maximum and minimum of the absolute values of $\mathbf{F}_\theta$' eigenvalues. We evaluated the condition number of relative FIM [32] with respect to the last two layers, regarding the feasibility in computation. We compared our methods with 'plain' and the other two re-parameterization methods: Natural Neural Network (NNN) [6] and Weight Normalization (WN) [28].

The results of the second last layer are shown in Figure 2 (c) and the last layer also has similar results, from which we can find that CWN, WN, NNN achieve better conditioning compared to 'plain', and thus converges faster as shown in Figure 2 (d). This indicates that re-parameterization serves as an effective way to make the optimization problem easier if good properties are designed carefully. Compared to WN, our proposed CWN method further significantly improves the conditioning and speed up convergence. This observation is consistent with our analysis in Section 3.2 that the proposed zero-mean property of *input weight* contributes to making the optimization problem easier. At the testing stage, the proposed CWN method achieves the lowest test error as shown in Table 3, which means that CWN not only accelerates and stabilizes training, but also has great potential to improve generalization of the neural network.

4.2. MLP architecture

In this part, we comprehensively evaluate our proposed method on MLP architecture. We investigate both the training and test performances on SVHN [23]. SVHN consists of 32×32 color images of house numbers collected by Google Street View. It has 73,257 train images and 26,032 test images. We train a 6 layers MLP with 128 neurons correspondingly in each hidden layer. Note that here we mainly focus on validating the effectiveness of our proposed methods on MLP architecture, and under this situation, SVHN dataset is permutation invariant.

Table 1. Comparison of test errors (%) averaged over 5 independent runs on Yale-B and permutation-invariant SVHN.

	plain	NNN	WN	CWN
Yale-B	6.16	6.58	4.68	**4.20**
SVHN	18.98	17.99	17.12	**16.16**

We employ the stochastic gradient descent algorithm with mini-batch of size 1024. Hyper-parameters are selected by grid search, based on the test error on validation set (5% samples of the training set) with the following grid specifications: learning rates in $\{0.1, 0.2, 0.5, 1\}$; the natural re-parameterization interval T in $\{20, 50, 100, 200, 500\}$ and the revised term ε within the range of $\{0.01, 0.01, 0.1, 1\}$ for NNN method.

Figure 3 shows the training and test error with respect to the number of epochs. We can find that WN, NNN and CWN converge faster compared to 'plain', which indicates that re-parameterization serves as an effective way to accelerate the training. Among all re-parametrization methods, CWN converges fastest. Besides, we observe that both the training and test error of CWN do not fluctuate after a few number of epochs in the training, which means that CWN is much more stable when it reaches the optimal regions. We can conjecture that the following functions of CWN as discussed in Section 3.2 contributes to this phenomenon: (1) CWN can stabilize the distribution of pre-activation; (2) the gradient $\partial\mathcal{L}/\partial\mathbf{v}$ of CWN can self-stabilize its norm. Table 3 presents the test error for different methods, where we further find that CWN achieves the best performances. These observations together show the great potential of CWN in improving the generalization of the neural network.

Combining batch normalization Batch normalization [16] has shown to be very helpful for speeding up the training of deep networks. Combined with batch normalization, previous re-parameterization methods [6] have shown success in improving the performance. Here we show that over the networks equipped with batch normalization, our proposed CWN still outperforms others in all cases. In this experiment, Batch Normalization (BN) is plugged in before the nonlinearity in neural networks as suggested in [16]. With batch normalization, we build different networks using the re-parameterized linear mapping including WN, NNN and CWN, named 'WN+BN', 'NNN+BN' and 'CWN+BN' respectively.

Figure 4 (a) shows their experimental results on SVHN dataset. We find that 'WN+BN' and 'NNN+BN' have no advantages compared to 'BN', while 'CWN+BN' significantly speeds up the training and achieves better test per-

Table 2. Comparison of test errors (%) averaged over 5 independent runs on Yale-B and permutation-invariant SVHN with Xavier-Init and He-Init.

	Xavier-Init		He-Init	
method	Yale-B	SVHN	Yale-B	SVHN
plain	6.47	17.78	5.47	18.35
NNN	5.47	17.83	5.58	18.14
WN	4.68	17.74	5.58	18.06
CWN	**3.84**	**16.38**	**4.21**	**16.71**

formance. Indeed, batch normalization is invariant to the weights scaling as analyzed in [24] and [4]. However, batch normalization is not re-centering invariant [4]. Therefore, our CWN can further improve the performance of batch normalization by centering the weights.

Amazingly, CWN itself achieves remarkably better performance in terms of both the training speed and the test error than BN, and even better than 'CWN+BN'. This is mainly due to the following reasons. Batch normalization can well stabilize the distribution of the pre-activation/activation, however it also introduces noises stemming from the forceful transformation based on mini-batch. Besides, during test the means and variances are estimated based on the moving averages of mini-batch means and variances [16, 3, 4]. These flaws may have a detrimental effect on models. Our CWN can stabilize the distribution of pre-activation implicitly and is deterministic without introducing stochastic noise. Moreover, the properties described in Proposition 2 can improve the conditioning, self-stabilize the gradient's norm, and thus achieve better performance.

Adam optimization We consider an alternative optimization method, Adam [17], to evaluate the performance. The initial learning rate is set to $\{0.001, 0.002, 0.005, 0.01\}$, and the best results for all methods on SVHN dataset are shown in Figure 4 (b). Again, we find that CWN can reach the lowest errors at both training and test stages.

Different initialization We also have tried different initialization methods: Xavier-Init [9] and He-Init [13]. The experimental setup is the same as in previous experiments. The final test errors both on Yale-B and SVHN datasets are shown in Table 2. We get similar conclusions as the random initialization [19] in previous experiments that CWN achieves the best test performance.

4.3. CNN architectures on CIFAR dataset

In this part, we highlight that the proposed centered weight normalization method also works well on several popular state-of-the-art CNN architectures[3], including VGG [31], GoogLeNet [34], and residual network [14, 15]. We

[3]The details of the used CNN architectures are shown in the supplementary materials.

Table 3. Comparison of test errors (%) averaged over 3 independent runs on 56 layers residual network over CIFAR-10 and CIFAR-100 datasets.

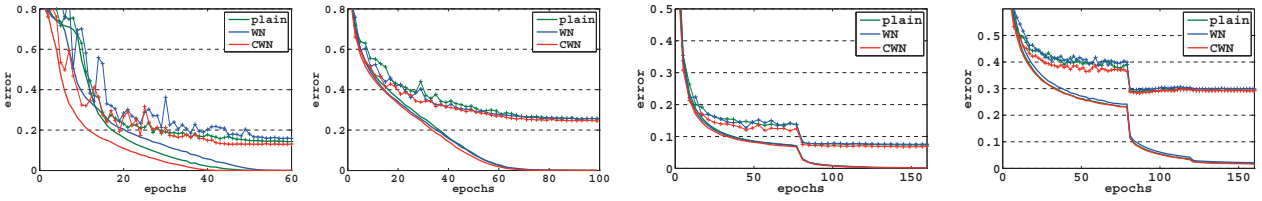
Methods	CIFAR-10	CIFAR-100
plain	7.34 ± 0.52	29.38 ± 0.14
WN	7.58 ± 0.40	29.85 ± 0.66
CWN	$\mathbf{6.85 \pm 0.26}$	$\mathbf{29.23 \pm 0.14}$

evaluate it over CIFAR-10 and CIFAR-100 datasets [18], which consists of 50,000 training images and 10,000 test images from 10 and 100 classes respectively. Each input image consists of 32×32 pixels.

VGG-A We first evaluate the proposed methods on the VGG-A architecture [31] where we set the number of neurons to 512 in the fully connected layer and use batch normalization in the first fully connected layer. The experiments run on CIFAR-10 dataset and we preprocess the dataset by subtracting the mean and dividing the variance.

We employ the stochastic gradient descent as the optimization method with mini-batch size of 256, and use momentum of 0.9, weight decay of 0.0005. The learning rate decays with each K iterations halving the learning rate. We initialize the learning rate $lr = \{0.5, 1, 2, 4\}$, and $K = \{1000, 2000, 4000, 8000\}$. The hyper-parameters are chosen over the validation set of 5,000 examples from the training set by grid search. We report the results of 'plain', WN and CWN in Figure 5 (a). It is easy to see that with the proposed CWN the training converges much faster and promises the lowest test error of 13.06%, compared to WN of 15.89% and 'plain' of 14.33%. Note that here WN obtains even worse performance compared to 'plain', which is mainly because that WN can not ensure the pre-activations nearly zero-mean, and thus degenerate the performance when the network is deep.

GoogleLeNet We conduct the experiments on GoogleLeNet. Here GoogleLeNet is equipped with the batch normalization [16], plugged after the linear mappings. The dataset is preprocessed as described in [10] with global contrast normalization and ZCA whitening. We follow the simple data augmentation that 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. The models are trained with a mini-batch size of 64, momentum of 0.9 and weight decay of 0.0005. The learning rate starts from 0.1 and ends at 0.001, and decays every two epochs with exponentially decaying until the end of the training with 100 epochs. The results on CIFAR-100 are reported in Figure 5 (b), from which we can find that CWN obtains slight speedup compared to WN and 'plain'. Moreover, CWN can attain the lowest test error of 24.45%, compare to 'plain' of 25.52% and WN of 25.39%. We also obtain similar observations on CIFAR-10 (see the supplementary materials).



(a) (b) (c) (d)

Figure 5. Performance comparison on various architecture over CIFAR datasets. We evaluate the training error (solid lines) and test error (lines marked with plus) with respect to the training epochs. (a) VGG-A architecture over CIFAR-10; (b) GoogLeNet architecture over CIFAR-100; (c) residual network architecture over CIFAR-10; (d) residual network architecture over CIFAR-100.

Residual network We also apply our CWN module to the residual network [14] with 56 layers. We follow the same experimental protocol as described in [14] and the publicly available Torch implementation for residual network [4]. Figure 5 (c) and (d) show the training error and test error with respect to epochs on CIFAR-10 and CIFAR-100 dataset respectively, and the test errors after training are shown in Table 3. We can find that CWN converges slightly faster than WN and 'plain' and achieves the best test performance.

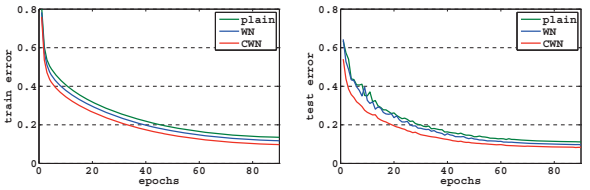
Computational cost We implement our CWN module based on Torch, and the re-parameterization is wrapped in the fastest SpatialConvolution of *cudnn* package [5]. We compared the wall clock time of CWN module with the standard convolution of *cudnn*. We use 3×3 convolution, 32×32 feature map with size 128 and mini-batch size of 64. The results are averaged over 100 runs. CWN costs 18.1ms while the standard one takes 17.3ms.

4.4. Large scale classification on ImageNet dataset

To show the scalability of our proposed method, we apply it to the large-scale ImageNet-2012 dataset [27] using the GoogLeNet architecture. ImageNet-2012 includes images of 1,000 classes, and is split into training sets with 1.2M images, validation set with 50k images, and test set with 100k images. We employ the validation set as the test set, and evaluate the classification performance based on top-1 and top-5 error. We use single scale and single crop test for simplifying discussion. Here we again use stochastic gradient descent with mini-batch size of 64, momentum of 0.9 and weight decay of 0.0001. The learning rate starts from 0.1 and ends at 0.0001, and decays with exponentially decaying until the end of the training with 90 epochs. The top-5 training and test error curves are shown in Figure 6 and the final test errors are shown in Table 4, where we can find that the deep network with CWN can converge to a lower training error faster, and obtain lower test error. We argue that our CWN module draws its strength in improving the performance of deep neural networks.

[4] https://github.com/facebook/fb.resnet.torch

[5] https://developer.nvidia.com/cudnn



(a) training error (b) test error

Figure 6. Top-5 training and test error curves on GoogLeNet architecture over ImageNet-2012 dataset.

Table 4. Comparison of test errors (%) on GoogLeNet over ImageNet-2012 dataset.

Methods	Top-1 error	Top-5 error
plain	30.78	11.14
WN	28.64	9.7
CWN	**26.1**	**8.35**

5. Conclusions

In this paper we introduced a new efficient re-parameterization method named *centered weight normalization*, which improves the conditioning and accelerates the convergence of the deep neural networks training. We validate its effectiveness in image classification tasks over both multi-layer perceptron and convolutional neural network architectures. The re-parameterization method is able to stabilize the distribution and accelerate the training for a number of popular networks. More importantly, it has very low computational overhead and can be wrapped as a linear module suitable for different types of networks. We argue that the centered weight normalization module owns the potential to replace the standard linear module, and contributes to new deep architectures with easier optimization and better generalization.

Acknowledgement

This work was partially supported by NSFC-61370125, NSFC-61402026, SKLSDE-2017ZX-03, FL-170100117, DP-140102164, LP-150100671 and the Innovation Foundation of BUAA for PhD Graduates.

References

[1] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, Feb. 1998. 1

[2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *ICML*, 2016. 1, 2

[3] D. Arpit, Y. Zhou, B. U. Kota, and V. Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *ICML*, 2016. 7

[4] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 1, 2, 7

[5] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. D. Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013. 2

[6] G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu. Natural neural networks. In *NIPS*, 2015. 1, 2, 6

[7] Z. Ding, M. Shao, and Y. Fu. Deep robust encoder through locality preserving low-rank dictionary. In *ECCV*, pages 567–582, 2016. 1

[8] V. Dorobantu, P. A. Stromhaug, and J. Renteria. Dizzyrnn: Reparameterizing recurrent neural networks for norm-preserving backpropagation. *CoRR*, abs/1612.04035, 2016. 2

[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 1, 3, 7

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013. 7

[11] R. B. Grosse and J. Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016. 1, 2

[12] R. B. Grosse and R. Salakhutdinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *ICML*, 2015. 1, 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 1, 3, 7

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 7, 8

[15] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. 7

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 2, 4, 6, 7

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 2, 3, 7

[18] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 7

[19] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade*, 1998. 1, 3, 5, 7

[20] J. Martens and I. Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade (2nd ed.)*. 2012. 1, 2

[21] D. Mishkin and J. Matas. All you need is a good init. In *ICLR*, 2016. 1

[22] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 5

[23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 6

[24] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro. Data-dependent path normalization in neural networks. *CoRR*, abs/1511.06747, 2015. 7

[25] G. Qi. Hierarchically gated deep networks for semantic segmentation. In *CVPR*, pages 2267–2275, 2016. 1

[26] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012. 2

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 8

[28] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 1, 2, 4, 6

[29] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013. 1

[30] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998. 4

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 7

[32] K. Sun and F. Nielsen. Relative natural gradient for learning large complex models. *CoRR*, abs/1606.06069, 2016. 6

[33] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *NIPS*, pages 1988–1996, 2014. 1

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 7

[35] Y. Tian, P. Luo, X. Wang, and X. Tang. Deep learning strong parts for pedestrian detection. In *ICCV*, pages 1904–1912, 2015. 1

[36] S. Wiesler, A. Richard, R. Schlüter, and H. Ney. Mean-normalized stochastic gradient for large-scale deep learning. In *ICASSP*, 2014. 1, 2

[37] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. In *NIPS*, pages 4880–4888. 2016. 2

[38] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang. Deep fried convnets. In *ICCV*, 2015. 2