An Embedded Feature Whitening Approach to Deep Neural Network Optimization

Hongwei Yong and Lei Zhang*

The Hong Kong Polytechnic University
{cshyong,cslzhang}@comp.polyu.edu.hk

Abstract. Compared with the feature normalization methods that are widely used in deep neural network (DNN) training, feature whitening methods take the correlation of features into consideration, which can help to learn more effective features. However, existing feature whitening methods have several limitations, such as the large computation and memory cost, inapplicable to pre-trained DNN models, the introduction of additional parameters, etc., making them impractical to use in optimizing DNNs. To overcome these drawbacks, we propose a novel Embedded Feature Whitening (EFW) approach to DNN optimization. EFW only adjusts the gradient of weight by using the whitening matrix without changing any part of the network so that it can be easily adopted to optimize pre-trained and well-defined DNN architectures. The momentum, adaptive dampening and gradient norm recovery techniques associated with EFW are consequently developed to make its implementation efficient with acceptable extra computation and memory cost. We apply EFW to two commonly used DNN optimizers, *i.e.*, SGDM and Adam (or AdamW), and name the obtained optimizers as W-SGDM and W-Adam. Extensive experimental results on various vision tasks, including image classification, object detection, segmentation and person ReID, demonstrate the superiority of W-SGDM and W-Adam to state-of-the-art DNN optimizers. The code are publicly available at https://github.com/Yonghongwei/W-SGDM-and-W-Adam.

Keywords: DNN Optimization, Feature Whitening, Deep Learning

1 Introduction

The remarkable success of Deep Neural Networks (DNNs) on various vision tasks, including image classification [7], object detection [29, 5], segmentation [5], image retrieval [46, 22], etc., largely owes to the development of DNN optimization techniques. The main goal of DNN optimization is to find a favorable local minimum of the objective function by using the given training data and ensure good generalization performance of the trained model to testing data. Meanwhile, it is anticipated that we can accelerate the converge speed and reduce the training cost. To achieve these goals, a variety of DNN optimization techniques have

*Corresponding author.

been proposed, such as weight initialization strategies [4, 6], efficient active functions (*e.g.*, ReLU [25]), batch normalization (BN) [13], gradient clipping [26, 27], adaptive learning rate optimizers [3, 14, 47], and so on. All these techniques facilitate the training of very deep and effective DNN models.

Among the above techniques, normalization methods have been widely used as a basic module to train a variety of DNN architectures [7, 9]. The most representative method is BN [13]. Similar to BN, instance normalization (IN) [36, 12], layer normalization (LN) [16] and group normalization (GN) [37] have also been proposed to perform Z-score standardization on other dimensions. It has been shown that normalization methods can both speed up the training speed and improve the generalization performance [32, 35, 43, 41]. However, normalization methods do not take the correlation of features into consideration. Therefore, feature whitening or feature decorrelation methods have been developed to solve this problem. For instance, decorrelated batch normalization (DBN) [10] was proposed to perform ZCA-whitening on each mini-batch with a ZCA transformation matrix obtained by eigen-decomposition. IterNorm [11] aims at a more efficient approximation of the ZCA transformation matrix with Newton's iteration. Network deconvolution (ND) [39] extends the ZCA-whitening transformation on a patch of features. The DNN models trained with whitening methods can achieve certain performance gains over normalization methods.

Nevertheless, the existing feature whitening methods have several obvious weaknesses, which make them hard to be widely used in practical applications. The major disadvantage of feature whitening lies in its large computational cost. In each iteration, the ZCA transformation matrix has to be computed by eigen-decomposition, which is computationally expensive when the dimension of features is high. Although some works [11, 39] adopt Newton's iteration to speed up the computation of ZCA transformation, the training cost is still unacceptable compared with BN. Meanwhile, the inference time of the network will increase largely when feature whitening is used. Moreover, feature whitening methods are very memory-consuming in training because more intermediate features need to be stored, especially for the iterative whitening methods. Last but not the least, the existing feature whitening methods cannot be directly applied to optimize pre-trained and well-defined DNN models. One needs to add a feature whitening module into the proper layer and redefine the forward propagation. For instance, if we want to adopt the ResNet50 [7] model pre-trained on ImageNet to downstream tasks, we must redefine the ResNet50 with these whitening methods and train it again on ImageNet. All these drawbacks largely limit the practical usage of feature whitening methods in DNN training.

To address these problems, we propose a novel approach, namely Embedded Feature Whitening (EFW), to DNN optimization by adjusting the gradient of weight with the ZCA transformation matrix. There are several advantages of our proposed approach. First, EFW inherits the advantages of feature whitening, *i.e.*, accelerating the training process and improving the generalization performance. Second, compared with existing feature whitening methods, EFW does not introduce any module into the DNN model to be trained. As a result, it

can be directly adopted to optimize most of the existing DNN models without increasing the inference time. Third, its computation and memory cost is acceptable because EFW only computes the ZCA transformation matrix once for many iterations (*e.g.*, 500) and it does not store any additional intermediate features. In this paper, we adopt EFW into two widely used DNN optimizers: SGD with momentum (SGDM) [28, 14] and Adam (or AdanW) [14, 21], and name the obtained optimizers as W-SGDM and W-Adam. Extensive experiments are conducted to validate the effectiveness of EFW on various vision tasks.

Notation system. In the following development of this paper, we denote by $\boldsymbol{W}$ the weight matrix, whose dimension is $C_{out} \times C_{in}$ for fully connected layers (FC layers) and $C_{out} \times C_{in} \times k_1 \times k_2$ for convolutional layers (Conv layers), where C_{in} is the number of input channels, C_{out} is the number of output channels, and k_1, k_2 are the kernel size of convolutional layers. We denote by $\boldsymbol{A} = [\boldsymbol{A}_n]_{n=1}^N$ and $\boldsymbol{X} = [\boldsymbol{X}_n]_{n=1}^N$ the input and output features of the N samples in one layer. For FC layers, $\boldsymbol{A} \in \mathbb{R}^{C_{out} \times N}$, $\boldsymbol{X} \in \mathbb{R}^{C_{in} \times N}$ and $\boldsymbol{A} = \boldsymbol{W}\boldsymbol{X}$. For Conv layers, $\boldsymbol{A} \in \mathbb{R}^{C_{out} \times h \times w \times N}$, $\boldsymbol{X} \in \mathbb{R}^{C_{in} \times h \times w \times N}$ and $\boldsymbol{A} = \boldsymbol{W} * \boldsymbol{X}$, where h and w are the height and width of a feature map and "$*$" is the convolution operator. Let $\mathcal{L}$ be the objective function, and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ be its gradients on activation and weight, respectively. $\mathcal{U}_1(\cdot)$ denotes the mode 1 unfold operation of a tensor. For example, for a convolution based weight matrix $\boldsymbol{W} \in \mathbb{R}^{C_{out} \times C_{in} \times k_1 \times k_2}$, $\mathcal{U}_1(\boldsymbol{W}) \in \mathbb{R}^{C_{out} \times C_{in} k_1 k_2}$. $vec(\cdot)$ denotes the vectorization function.

2 Related Work

DNN Optimizers. The first-order optimization algorithms have been widely adopted in training a DNN. For example, SGD with Momentum (SGDM) [28] makes use of the momentum of gradient to avoid oscillations and strengthen the relevant gradient direction. Adagrad [3] adapts adaptive learning rates to different parameters, performing larger/smaller gradient steps for infrequent/frequent ones. RMSprop and Adadelta [42] use a similar mechanism to Adagrad, and Adam [14] further introduces the momentum of gradient into adaptive learning rate methods. Based on Adam, Adabelief [47] considers the belief of observed gradient to adjust the adaptive learning rates.

For the second-order optimizers, AdaHessian [38] simplifies the Hessian matrix with the diagonal elements through Hessian-free techniques. Similar to AdaHessian, Apollo [23] simplifies the BFGS algorithm with only diagonal elements. Meanwhile, Kronecker Factored Approximation Curvature (KFAC) [24] uses the Kronecker Factor decomposition to approximate the natural gradient layer-wisely. However, in many computer vision tasks, the generalization performance of these second-order methods does not outperform SGDM.

Feature Whitening. Feature whitening methods remove the linear correlation among different channel features to perform gradient descent more efficiently. Beyond standardization, DBN [10] was proposed to perform ZCA-whitening by eigen-decomposition and backpropagating the transformation. IterNorm [11] aims at a more efficient approximation of the ZCA-whitening matrix in DBN with Newton's iteration. Network deconvolution (ND) [39] adopts decon-

Algorithm 1: Overview of Batch Feature Whitening

Input: Mini-batch input $\boldsymbol{X} \in \mathbb{R}^{C_{out} \times N}$
Output: Output $\boldsymbol{Y} \in \mathbb{R}^{C_{out} \times N}$

```
if Training then
    Centralization: X̂ = Φ1(X|μ_B), μ_B = (1/N) X 1;
    Standardization or decorrelation: Y = Φ2(X̂|Σ_B), Σ_B = (1/N) X̂ X̂^T + εI;
    Update the population statistics μ and Σ;
else
    Calculate output Y = Φ3(X|μ, Σ);
end
Recovery Operation Ŷ = Φ4(Y)
```

volution filters to remove pixel-wise and channel-wise correlations. It has been shown that feature whitening methods can boost both the optimization and the generalization of DNNs [11, 39]. However, they usually need a lot of extra computation and memory, making them impractical in real-world applications.

3 Embedded Feature Whitening

3.1 Overview of Batch Feature Whitening

We briefly summarize the batch whitening process in **Algorithm 1**. In training, batch feature whitening [11, 39] usually involves two main steps, *i.e.*, centralization $\Phi_1(\boldsymbol{X})$ and decorrelation $\Phi_2(\boldsymbol{X})$, which are defined as follows:

$$\begin{aligned} \Phi_1(\boldsymbol{X}|\boldsymbol{\mu}) &= \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{1}^T,\ \boldsymbol{\mu} = \frac{1}{N}\boldsymbol{X}\boldsymbol{1}, \\ \Phi_2(\boldsymbol{X}|\boldsymbol{\Sigma}) &= \boldsymbol{T}\boldsymbol{X},\ \boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^T + \epsilon\boldsymbol{I}, \end{aligned} \tag{1}$$

where $\boldsymbol{T}$ is the whitening matrix, which is related to $\boldsymbol{\Sigma}$. For different whitening methods, $\boldsymbol{T}$ has different formulations [10, 11, 39, 33]. All the whitening matrices should meet that $\frac{1}{N}\Phi_2(\boldsymbol{X})\Phi_2(\boldsymbol{X})^T = \boldsymbol{I}$. Among those whitening transformations, PCA and ZCA whitening are widely used, whose whitening matrices are $\boldsymbol{T} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}^T$ and $\boldsymbol{T} = \boldsymbol{U}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}^T$, respectively, where $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ is the eigen-decomposition of $\boldsymbol{\Sigma} = \boldsymbol{X}\boldsymbol{X}^T/N + \epsilon\boldsymbol{I}$.

In the training step, the batch statistics $\boldsymbol{\mu}_{\mathcal{B}}$ and $\boldsymbol{\Sigma}_{\mathcal{B}}$ are used to perform whitening. Meanwhile, the population statistics $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are updated by exponential moving average [11, 39]. In the inference step, the population statistics are used to replace batch statistics, *i.e.*, $\Phi_3(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \Phi_2(\Phi_1(\boldsymbol{X}|\boldsymbol{\mu}), \boldsymbol{\Sigma})$. After the whitening operation, an additional recovery operation $\Phi_4(\cdot)$ is used to keep the representation capability of the network. The recovery operation is usually a linear operation, such as affine transformation [11] and coloring operation [33], which introduces extra parameters in training.

During the DNN training process, due to the variation of input feature statistics, the whitening matrix also changes. As a consequence, whitening may change the intermediate features acutely, making the following layers hard to learn. It has been shown that ZCA whitening can avoid such a Stochastic Axis Swapping

(SAS) problem, leading to better feature learning performance [10]. Actually, we can show that the solution of the following objective function

$$\min_{\boldsymbol{T}} ||\boldsymbol{X} - \Phi(\boldsymbol{X})||_2^2, \quad s.t. \quad \Phi(\boldsymbol{X}) = \boldsymbol{T}\boldsymbol{X}, \quad \frac{1}{N}\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^T = \boldsymbol{I} \tag{2}$$

is $\boldsymbol{T} = (\boldsymbol{X}\boldsymbol{X}^T/N)^{-\frac{1}{2}}$, which is just the ZCA whitening formulation. (The proof can be found in the **supplementary file**). This ensures that the ZCA whitened feature $\Phi(\boldsymbol{X})$ is close to the original data $\boldsymbol{X}$ and hence dilutes the SAS issue.

3.2 Drawbacks of Feature Whitening

Although many works have shown that feature whitening can both speed up training and gain generalization performance, it has some obvious drawbacks that largely limit its applications to DNN training. First, it needs to perform eigen-decomposition or use Newton's iteration to compute the whitening matrix, both of which will significantly increase the computation and memory cost. Second, existing feature whitening methods cannot be directly adopted to optimize pre-trained DNN models (*e.g.*, ImageNet pre-trained models). We have to redefine the forward propagation of DNNs by introducing a whitening module and retrain the models. Third, the batch feature whitening methods are very sensitive to the training batch size. When batch size is small, the statistics will become inaccurate, leading to a large performance drop. Fourth, most of the current whitening methods will introduce additional parameters into the recovery operation step to keep the representation capability of the DNNs, which increases the number of parameters to be optimized.

Due to the above limitations, though having many attractive properties, feature whitening methods have not been widely used to optimize DNNs yet. To overcome the above drawbacks of feature whitening while inheriting its advantages, we should not change the forward propagation of DNN or introduce new modules (*e.g.*, whitening layer) in DNN, and should reduce its extra computation cost. To achieve these goals, we propose a novel approach to embed the feature whitening operation into the optimization algorithms.

3.3 Removal of Recovery and Centralization Operations

Most batch whitening methods employ a recovery operation to keep the representation capability of DNNs. Actually, the recovery operation may not be necessary. According to their locations in DNN layers, whitening methods can be divided into pre-whitening and post-whitening ones.

When the whitening layer is placed before the convolutional layer, it is a pre-whitening layer, otherwise, it is a post-whitening layer. Traditional normalization layers and whitening layers usually introduce an additional recovery transformation, such as affine transformation [11] or coloring operation [33], to keep the feature representation performance. When post-whitening is adopted, the recovery transformation must be introduced after the whitening operation to keep the performance.

When pre-whitening is adopted, however, the recovery transformation can be removed without harming the representation power of DNNs, because it can be assimilated by the following Conv layer. For instance, supposing that $\boldsymbol{W_r} * \boldsymbol{X}$ is the recovery transformation (affine and coloring transformation can be viewed as a sparse convolutional operation), where $\boldsymbol{W_r}$ is the extra parameters to be learned, $\boldsymbol{W} * \boldsymbol{W_r} * \boldsymbol{X}$ will be the output feature of Conv layer. We can let $\boldsymbol{W'} = \boldsymbol{W} * \boldsymbol{W_r}$ and hence only optimize the Conv layer with parameter $\boldsymbol{W'}$. This property of pre-whitening inspires us to embed the whitening layer into the optimization algorithm without changing any module of the DNN.

Meanwhile, in the traditional whitening methods, there are two main operations: centralization and decorrelation. In forward propagation, we need to introduce these two operations into the whitening layer before optimization. However, for a well-defined DNN, the mean of input feature to a Conv or FC layer is usually not zero since there is no centralization operation before them. A practical way to achieve feature centralization is to introduce an extra bias that is related to the mean of input activation. However, since the normalization layers are usually located after the Conv layer in many popular DNNs (*e.g.*, ResNet), the bias in the Conv layer will have no function. Moreover, since our goal is to optimize a well-defined DNN without changing its forward propagation and introducing any extra parameters, we omit the centralization operation and only take the decorrelation into consideration.

3.4 Formulation of Embedded Feature Whitening

For a FC layer $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X}$, where $\boldsymbol{W}$ denotes the parameters to learn, suppose there is a virtual whitening layer before FC layer, which is $\hat{\boldsymbol{X}} = \boldsymbol{T}\boldsymbol{X}$, where $\boldsymbol{T}$ is defined in Eq. (1). We can reformulate this FC layer with a whitening transformation as $\boldsymbol{Y} = \boldsymbol{W'}\boldsymbol{T}\boldsymbol{X}$, where $\boldsymbol{W'}$ is the new parameters to be optimized. In this way, we can optimize the loss function w.r.t. $\boldsymbol{W'}$, and let $\boldsymbol{W} = \boldsymbol{W'}\boldsymbol{T}$ once the training is finished. According to the backpropagation algorithm, the gradient of $\boldsymbol{W'}$ can be easily obtained by $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W'}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\boldsymbol{T}$. However, the above approach has several serious problems. First, the whitening matrix $\boldsymbol{T}$ will change during the training process because of the update of weights in the previous layers. As a consequence, the relationship between $\boldsymbol{W}$ and $\boldsymbol{W'}$ is not fixed. Second, in the training process, $\boldsymbol{T}$ will contain a certain amount of noise due to the random batch sampling, so it is hard to get an accurate $\boldsymbol{T}$. Therefore, it is difficult to obtain an accurate $\boldsymbol{W}$ from the optimization of $\boldsymbol{W'}$.

To maintain the benefits of batch feature whitening on optimization, we propose to use a modified gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\boldsymbol{T}$ to replace the original weight gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$, and name the method Embedded Feature Whitening (EFW), which embeds the information of feature whitening into the weight gradient. EFW can be introduced into the FC layer, convolutional layer, and Norm layer. Compared with the weight updating formula of SGD $\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^t}$, the updating formula of SGD with EFW is

$$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^t}\boldsymbol{T}^t. \tag{3}$$

Table 1. The updating formulas and whitening matrices of FC, Conv and Norm layers in SGD with the proposed EFW.

Layer	Updating formula	Whitening matrix
FC layer	$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^t} \boldsymbol{T}^t$	$\boldsymbol{T}^t = \left(\boldsymbol{X}^t \boldsymbol{X}^{t^T}\right)^{-\frac{1}{2}}$
Conv layer	$\mathcal{U}_1(\boldsymbol{W}^{t+1}) = \mathcal{U}_1(\boldsymbol{W}^t) - \eta \mathcal{U}_1(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^t}) \boldsymbol{T}^t$	$\boldsymbol{T}^t = \left(\boldsymbol{\mathfrak{X}}^t \boldsymbol{\mathfrak{X}}^{t^T}\right)^{-\frac{1}{2}}$
Norm layer	$\begin{bmatrix} \gamma^{t+1} \\ \beta^{t+1} \end{bmatrix} = \begin{bmatrix} \gamma^t \\ \beta^t \end{bmatrix} - \eta \boldsymbol{T}^t \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \gamma^t} \\ \frac{\partial \mathcal{L}}{\partial \beta^t} \end{bmatrix}$	$\boldsymbol{T}^t = \left(\begin{bmatrix} vec(\boldsymbol{X}^t)^T \\ \boldsymbol{1}^T \end{bmatrix} [vec(\boldsymbol{X}^t), \boldsymbol{1}] \right)^{-\frac{1}{2}}$

The detailed updating formulas are summarized in Table 1. We ignore the factor $1/N$ in the second-order statistic because of the gradient norm recovery operation, which will be explained in Section 3.5.

For the FC layer, we need to calculate the second-order statistic of input activation, *i.e.*, $\boldsymbol{X}^t \boldsymbol{X}^{t^T}$, and the whitening matrix $\boldsymbol{T}^t$, which can be obtained by SVD decomposition of $\boldsymbol{X}^t \boldsymbol{X}^{t^T}$. For the Conv layer, the difference from the FC layer lies in that we need to unfold the convolution operation to matrix multiplication first. The convolution operation can be formulated as a matrix multiplication with the *im2col* operation [39, 44], and then the Conv layer can be viewed as an FC layer. The updating formula of weights for the Conv layer is listed in TABLE 1, where $\mathcal{U}_1(\cdot)$ is the mode 1 unfold operation of a tensor and $\boldsymbol{\mathfrak{X}}$ is the matrix of $\boldsymbol{X}^t$ after *im2col* operation. The normalization layers usually have a channel-wise affine transformation, which is also a linear operation. Suppose that the normalized features are $\boldsymbol{X}^t$ and the parameters of affine transformation are γ and β for one channel, we can obtain the updating rules for γ, β as shown in the bottom row of TABLE 1. If the mean and variance of $\boldsymbol{X}^t$ are zero and one, the second-order statistics will be a diagonal 2×2 matrix. For example, when BN [13] and IN [36, 12] are used, the update rules for (γ, β) degrade to the case of SGD. However, for other normalization methods such as GN [37] and LN [16], the mean and variance of each channel may not be zero and one.

In practice, to avoid that the condition number of the statistic matrix $\boldsymbol{X}^t \boldsymbol{X}^{t^T}$ is too large, we need to add an additional term $\epsilon \boldsymbol{I}$ to the statistic matrix, where $\boldsymbol{I}$ is an identity matrix and ϵ is the dampening parameter. We will discuss how to choose a proper ϵ in the next section.

3.5 Implementation of EFW

Momentum. The estimation of the second-order statistics of $\boldsymbol{X}$ is very important for the whitening methods. The original batch whitening method can only use the current batch statistics for computation, and hence they are very sensitive to the training batch size. When the training batch size is small, the batch statistics will have large noise so that the training will be unstable. In contrast, our proposed EFW method works directly on the final weight updating stage, and it does not change the forward propagation and backward propagation during training. Therefore, EFW can adopt the statistics from more batches to

Algorithm 2: Algorithm of EFW

Input: T_{xx}, T_{svd}, α, $\boldsymbol{M}_{xx}^{t-1}$, $\boldsymbol{T}^{t-1}$, ϵ, input activation $\boldsymbol{X}^t$, gradient $\nabla_{\boldsymbol{W}^t}\mathcal{L}$
Output: $\tilde{\boldsymbol{G}}^t$

$\boldsymbol{G}^t = \nabla_{\boldsymbol{W}^t}\mathcal{L}$;
if $t\%T_{xx} = 0$ **then**
  $\boldsymbol{M}_{xx}^t = \alpha\boldsymbol{M}_{xx}^{t-1} + (1-\alpha)\boldsymbol{X}^t\boldsymbol{X}^{t^T}$ % *Momentum step*
else
  $\boldsymbol{M}_{xx}^t = \boldsymbol{M}_{xx}^{t-1}$
end
if $t\%T_{svd} = 0$ **then**
  $\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T = \boldsymbol{M}_{xx}^t$ % *SVD decomposition*
  $\boldsymbol{T}^t = \boldsymbol{U}(\boldsymbol{D} + \epsilon d_{max}\boldsymbol{I})^{-1/2}\boldsymbol{U}^T$ % *Whitening matrix with dampening*
else
  $\boldsymbol{T}^t = \boldsymbol{T}^{t-1}$
end
$\widehat{\boldsymbol{G}}^t = \boldsymbol{G}^t\boldsymbol{T}^t$ % *Adjust gradient with whitening matrix*
$\tilde{\boldsymbol{G}}^t = \widehat{\boldsymbol{G}}^t \frac{||\boldsymbol{G}^t||_2}{||\widehat{\boldsymbol{G}}^t||_2}$; % *Gradient norm recovery*

achieve a more accurate estimation of feature statistics. Specifically, we compute the momentum of the batch statistics as follows:

$$\boldsymbol{M}_{xx}^t = \alpha\boldsymbol{M}_{xx}^{t-1} + (1-\alpha)\boldsymbol{X}^t\boldsymbol{X}^{t^T}, \tag{4}$$

where $\boldsymbol{M}_{xx}^t$ is the momentum of statistics $\boldsymbol{X}\boldsymbol{X}^T$ in iteration t and α is the momentum parameter. As an approximation to the population of feature statistics, momentum can significantly reduce the noise caused by random batch sampling.

Statistics Computation. Feature whitening methods need to compute the second-order statistics and then compute the whitening matrix for feature learning. The previous batch whitening methods need to perform these computations in each iteration for each batch because the batch statistics and whitening matrix are involved in forward and backward propagations. This however introduces a large amount of computational burden.

Different from the previous batch whitening methods, in our proposed EFW there is no need to compute the second-order statistics and the whitening matrix in each iteration. We only need to compute them once for many iterations. Two hyperparameters, T_{xx} and T_{svd}, are introduced to control the interval for updating the statistics matrix and the whitening matrix, respectively. For the whitening matrix, the updating interval should be set larger because its computation involves SVD decomposition, which is more computationally expensive. In our experiments, we set $T_{xx} = 50$ and $T_{svd} = 500$ and we find that they work effectively to improve the DNN optimization performance without introducing much additional computational cost. Meanwhile, we also implement a cross-GPU synchronization method to facilitate the computation of more reliable feature statistics when using multiple GPUs.

Adaptive dampening. The dimension of $\boldsymbol{M}_{xx}^t$ is very high and it is usually a very singular matrix. When the condition number of $\boldsymbol{M}_{xx}^t$ is too large, it will

be unstable to compute the inverse square root of it. To avoid such a case, in practice we need to add an additional term $\epsilon \boldsymbol{I}$ to the statistic matrix, where $\boldsymbol{I}$ is an identity matrix and ϵ is a dampening parameter.

A too-small dampening may not improve the condition number of $\boldsymbol{M}_{xx}^t$, while a too strong dampening may reduce the accuracy of statistics. Therefore, it is important to choose a proper dampening parameter ϵ. For different layers in a DNN, the statistics $\boldsymbol{M}_{xx}^t$ may have different magnitude. Thus, it is improper to use a uniform dampening scheme for all layers. By taking the magnitude of different features into consideration, we choose an adaptive dampening parameter ϵd_{max}, where d_{max} is the max singular value of $\boldsymbol{M}_{xx}^t$. It is easy to show that the condition number of $\boldsymbol{M}_{xx}^t + \epsilon d_{max}\boldsymbol{I}$ is $\frac{d_{max}+\epsilon d_{max}}{d_{min}+\epsilon d_{max}} < \frac{1+\epsilon}{\epsilon}$. In practice, we can first compute the SVD decomposition of $\boldsymbol{M}_{xx}^t$, *i.e.*, $\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T = \boldsymbol{M}_{xx}^t$, and then obtain the whitening matrix by $\boldsymbol{T}^t = \boldsymbol{U}(\boldsymbol{D} + \epsilon d_{max}\boldsymbol{I})^{-1/2}\boldsymbol{U}^T$. The computation cost of adaptive dampening is the same as fixed dampening.

Gradient Norm Recovery. SGDM and Adam are among the most commonly used optimizers in training DNNs. Their hyperparameters, including learning rate and weight decay, have been well-tuned by researchers on many specific tasks. For example, in objection detection, SGDM with a learning rate 0.02 and weight decay 0.0001 is widely adopted. A natural question is can we hold these well-tuned hyperparameters in the proposed method to ease the tedious work of hyperparameter tuning? If this can be done, EFW can be easily used for solving various vision tasks without further hyperparameter tuning.

In the proposed EFW, the scale of adjusted gradient $\widehat{\boldsymbol{G}}^t = \boldsymbol{G}^t\boldsymbol{T}^t$ might be changed. This implies that the optimal setting of hyperparameters should be changed for the adopted optimizer, limiting the application of the proposed method. Fortunately, this problem of gradient scale changing can be easily addressed by recovering the gradient norm, which is

$$\tilde{\boldsymbol{G}}^t = \widehat{\boldsymbol{G}}^t \frac{||\boldsymbol{G}^t||_2}{||\widehat{\boldsymbol{G}}^t||_2}, \tag{5}$$

It is easy to see that $\tilde{\boldsymbol{G}}^t$ and $\boldsymbol{G}^t$ have the same L_2 norm. With the gradient norm recovery operation, $\tilde{\boldsymbol{G}}^t$ can be readily used in the employed optimizers (*e.g.*, SGDM and Adam) to achieve favorable performance without additional hyperparameter tuning. Of course, one may further improve the performance by tuning fine-grained hyperparameters around their default settings.

Algorithm of EFW. The complexity of EFW is $T(O(\frac{C_{in}^3}{T_{svd}}) + O(\frac{C_{in}^2 N}{T_{xx}}) + O(C_{in}^2 C_{out}))$ for a FC layer, and $T(O(\frac{C_{in}^3 k_1^3 k_2^3}{T_{svd}}) + O(\frac{C_{in}^2 k_1^2 k_2^2 N}{T_{xx}}) + O(C_{in}^2 k_1^2 k_2^2 C_{out}))$ for a Conv layer, where T is the total number of iterations. Since T_{xx} and T_{svd} can be set as large numbers in our implementation (50 and 500, respectively), the complexity is acceptable. The algorithm of EFW is summarized in **Algorithm 2**. In the experiments, we apply EFW to the two commonly used DNN optimizers, *i.e.*, SGDM and Adam (or AdamW), and name the obtained new optimizers as W-SGDM and W-Adam accordingly. We found that EFW only introduces $10\% \sim 20\%$ extra memory consumption in our experiments.

Table 2. Testing accuracies (%) on CIFAR100/CIFAR10. The best and second best results are highlighted in bold and italic fonts, respectively. The numbers in red color indicate the improvement of W-SGDM/W-Adam over SGDM/AdamW, respectively. "-" means that the result is not available due to the problem of "out of memory".

CIFAR100									
Model	SGDM	AdamW [21]	RAdam [19]	Ranger	Adabelief [47]	AdaHessian [38]	Apollo [23]	W-SGDM	W-Adam
R18	77.20±.30	77.23±.10	77.05±.15	76.75±.11	77.43±.36	76.73±.23	77.65±.11	**79.28**±.27 (↑2.08)	*78.75*±.16 (↑1.52)
R50	77.78±.43	78.10±.17	78.20±.15	78.13±.12	79.08±.23	78.48±.22	79.25±.26	**80.90**±.23 (↑3.12)	*80.15*±.22 (↑2.05)
V11	70.80±.29	71.20±.29	71.08±.24	70.58±.14	72.43±.16	67.78±.34	72.35±.33	**73.42**±.28 (↑2.62)	*72.92*±.14 (↑1.72)
D121	79.53±.19	78.05±.26	78.65±.05	78.28±.08	79.88±.08	-	79.83±.16	**81.23**±.10 (↑1.70)	*80.10*±.25 (↑2.05)
MobileNet	68.03±.37	70.07±.19	69.55±.32	69.35±.15	*71.40*±.12	69.45±.30	70.75±.22	70.35±.21 (↑2.32)	**71.92**±.16 (↑1.85)
CIFAR10									
R18	95.10±.07	94.80±.10	94.70±.18	94.75±.18	95.12±.14	94.70±.15	95.20±.12	**95.43**±.08 (↑0.33)	*95.20*±.10(↑0.40)
R50	94.75±.30	94.72±.10	94.72±.10	95.27±.12	95.35±.05	95.35±.11	95.37±.10	**95.80**±.15 (↑1.05)	*95.70*±.07(↑0.98)
V11	92.17±.19	92.02±.08	92.00±.18	92.10±.07	92.45±.18	91.85±.16	92.58±.04	**92.95**±.20 (↑0.78)	*92.88*±.19(↑0.86)
D121	95.37±.17	94.80±.07	95.02±.08	95.45±.11	95.37±.04	-	95.23±.10	**95.72**±.14 (↑0.35)	*95.47*±.12(↑0.67)
MobileNet	90.90±.14	92.08±.10	92.08±.25	92.05±.08	*92.33*±.19	91.25±.12	92.03±16	91.30±.12(↑0.40)	**92.45**±.18(↑0.37)

4 Experiment Results

4.1 Experiment Setup

We evaluate the proposed W-SGDM and W-Adam on various vision tasks, including image classification (on CIFAR100/CIFAR10 [15] and ImageNet [31]), object detection and segmentation (on COCO [18]), and Person Re-identification (Person ReID, on Market1501 [46] and DukeMTMC-ReID [30]). The compared methods include the representative and state-of-the-art DNN optimizers, including SGDM, AdamW [21], RAdam [19], Ranger [19, 45, 40] and Adabelief [47], AdaHessian[1] [38] and Apollo [23]. For the competing methods, we use the default settings for most of their hyper-parameters, and tune their learning rates and weight decays to report their best results.

We first testify W-SGDM and W-Adam with different DNN models on CIFAR100/CIFAR10, including VGG11 [34], ResNet18, ResNet50 [7], DenseNet-121 [9] and MobileNet [8]. Then we perform experiments on ImageNet to validate their performance on the large-scale datasets. After that, we test W-SGDM on COCO for detection and segmentation, and test W-Adam on Market1501 [46] and DukeMTMC-ReID for Person ReID to demonstrate that EFW can be easily adopted to finetune pre-trained models. All experiments are conducted under the Pytorch 1.7 framework with NVIDIA GeForce RTX 2080Ti and eight 3090Ti GPUs. For the hyper-parameters of EFW, we set $\alpha = 0.95$, $T_{xx} = 50$ and $T_{svd} = 500$, $\epsilon = 0.001$ throughout the experiments if not specified. Ablation studies on hyperparameter selection are also provided.

4.2 Image Classification

Results on CIFAR100 and CIFAR10: CIFAR100 and CIFAR10 [15] are two popular datasets to testify DNN optimizers. They include 50K training images and 10K testing images from 100 categories and 10 categories, respectively, and the resolution of the input image is 32×32. We conduct experiments on these two relatively small-scale datasets to illustrate the effectiveness of W-SGDM

[1] Since AdaHessian is very memory expensive, we can only give partial results in the following experiments.

Table 3. Top 1 accuracy (%) on the validation set of ImageNet. The numbers in red color indicate the improvement of W-SGDM/W-Adam over SGDM/AdamW, respectively. "-" means that the result is not available due to the problem of "out of memory".

Model	SGDM	AdamW	RAdam [19]	Ranger	Adabelief [47]	AdaHessian [38]	Apollo [23]	W-SGDM	W-Adam
R18	70.47	70.01	69.92	69.35	70.08	70.08	70.39	*71.43*(↑0.96)	**71.59**(↑1.58)
R50	76.31	76.02	76.12	75.95	76.22	-	76.32	**77.48**(↑1.17)	*76.83*(↑0.81)

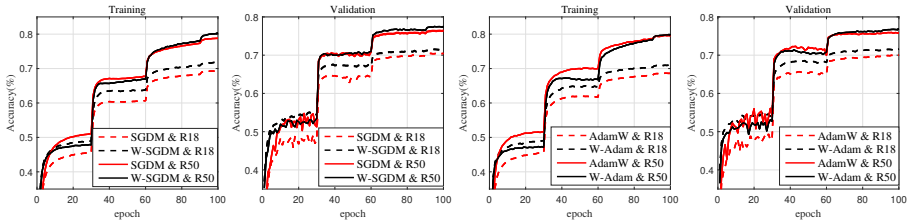

Fig. 1. Training and validation accuracy curves of SGDM, W-SGDM, AdamW and W-Adam on ImageNet with ResNet18 and ResNet50.

and W-Adam with different DNN backbone models, including VGG11 (V11), ResNet18 (R18), ResNet50 (R50), DenseNet121 (D121) and MobileNet[2]. All the DNN models are trained for 200 epochs with batch size 128 on one 2080Ti GPU. The learning rate is multiplied by 0.1 for every 60 epochs. We tune the learning rate in $\{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}, 1e^{-2}, 5e^{-2}, 0.1, 0.15\}$ and weight decay in $\{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}, 1e^{-2}, 5e^{-2}, 0.1, 0.5, 1\}$, and choose the best combination of them for all methods. The detailed settings can be found in the **supplementary material**. We use the default settings for other hyperparameters.

The experiments are repeated 4 times and the results are reported in Table 2 in mean ± std format. We can see that W-SGDM and W-Adam achieve the best and second-best testing accuracies for all the used DNN models. More specifically, W-SGDM improves SGDM from 1.7% to 3.12% on CIFAR100, and from 0.33% to 1.05% on CIFAR10, while W-Adam improves AdamW from 1.52% to 2.05% on CIFAR100, and from 0.37% to 0.98% on CIFAR10. Among the adaptive learning rate methods, Adam, AdamW, RAdam and Ranger perform worse than SGDM. Only Adabelief outperforms SGDM but it is still much worse than W-SGDM and W-Adam. It can be seen that W-SGDM and W-Adam significantly surpass other optimizers in generalization performance, validating the effectiveness of our proposed EFW scheme.

Results on ImageNet: We then evaluate W-SGDM and W-Adam on the large-scale image classification dataset ImageNet [31], which consists of 1.28 million training images and 50K validation images from 1000 categories. ResNet18 and ResNet50 are employed as the backbone models with training batch size 256 on four 2080Ti GPUs. The standard settings in [1] are used, where the models are trained for 100 epochs. We refer to the strategies in [47] to set the learning rate and weight decay. The detailed settings for different optimizers can be found in the **supplementary material**. The top 1 accuracies of competing optimizers on the validation set are reported in Table 3. We can see that W-SGDM and

[2] These models for CIFAR100/10 can be downloaded at the repository https://github.com/weiaicunzai/pytorch-cifar100.

Table 4. Detection results of Faster-RCNN on COCO. Δ means the gain of W-SGDM over SGDM.

Backbone	Algorithm	AP	$AP_{.5}$	$AP_{.75}$	AP_s	AP_m	AP_l
R50	SGDM	37.4	58.1	40.4	21.2	41.0	48.1
	W-SGDM	39.4	60.6	43.1	23.1	42.9	50.7
	Δ	↑2.0	↑2.5	↑2.7	↑1.9	↑1.9	↑2.6
R101	SGDM	39.4	60.1	43.1	22.4	43.7	51.1
	W-SGDM	41.1	61.6	45.1	24.0	45.2	54.3
	Δ	↑1.7	↑1.5	↑2.0	↑1.6	↑1.5	↑3.2

Table 5. Detection and segmentation results of Mask-RCNN on COCO. Δ means the gain of W-SGDM over SGDM.

Backbone	Algorithm	AP^b	$AP^b_{.5}$	$AP^b_{.75}$	AP^m	$AP^m_{.5}$	$AP^m_{.75}$
R50	SGDM	38.2	58.8	41.4	34.7	55.7	37.2
	W-SGDM	39.8	60.8	43.4	36.4	57.6	38.9
	Δ	↑1.6	↑2.0	↑2.0	↑1.7	↑1.9	↑1.7
R101	SGDM	40.0	60.5	44.0	36.1	57.5	38.6
	W-SGDM	41.7	62.5	45.5	37.9	59.4	40.8
	Δ	↑1.7	↑2.0	↑1.5	↑1.8	↑1.9	↑2.2
Swin-T	AdamW	42.7	65.2	46.8	39.3	62.2	42.2
	W-Adam	43.4	65.7	47.5	40.1	63.0	43.2
	Δ	↑0.7	↑0.5	↑0.7	↑0.8	↑0.8	↑1.0

W-Adam are the top 2 performers. Specifically, W-SGDM outperforms SGDM by 0.96% and 1.17%, and W-Adam outperforms AdamW by 1.58% and 0.81% for ResNet18 and ResNet50, respectively. The training and validation accuracy curves of SGDM vs. W-SGDM and AdamW vs. W-Adam are plotted in Fig. 1. For ResNet18, the learning rate and weight decays of W-SGDM and W-Adam are the same as SGDM and AdamW, respectively. While for ResNet50, the weight decays of W-SGDM and W-Adam are set larger than SGDM and AdamW. It can be seen that W-SGDM and W-Adam achieve both higher training accuracy and validation accuracy than SGDM and AdamW. This indicates that EFW can not only boost the generalization performance but also speed up the training process of DNN models on large-scale datasets.

4.3 Object Detection and Segmentation

We then test EFW on COCO [18] detection and segmentation tasks to show that it can be adopted for fine-tuning pre-trained models without changing the well-tuned hyper-parameters of default optimizer, such as learning rate and weight decay. The pre-trained models are downloaded from the PyTorch official websites. They are fine-tuned on COCO *train*2017 (118K images) with four 3090Ti GPUs and 4 images per GPU, and then evaluated on COCO *val*2017 (40K images). The latest version of MMDetection [2] toolbox[3] is used as the framework. We adopt the official implementations and settings for all experiments here. The backbone networks include ResNet50 (R50), ResNet101 (R101) and Swin-T vision transformer [20]. The Feature Pyramid Network (FPN) [17] is also used. The learning rate schedule is 1X for both Faster-RCNN [29] and Mask-RCNN [5].

As we discussed in Section 3.5, with the gradient norm recovery operation in EFW, we can directly adopt the hyperparameters of SGDM into W-SGDM, and the hyperparameters of AdamW into W-Adam. Table 4 lists the Average Precision (AP) of object detection by Faster-RCNN. One can see that the models trained by W-SGDM achieve a clear performance boost of 2.0% for ResNet50 and 1.7% for ResNet101. Table 5 reports the AP^b of detection and AP^m of segmentation by Mask-RCNN. W-SGDM gains AP^b by 1.6% and 1.7% on object detection and 1.7% and 1.8% on segmentation for ResNet50 and ResNet101, respectively. W-Adam achieves 0.7% AP^b gain and 0.8% AP^m gain on Swin-T backbone, showing that EFW can work well on the self-attention layer and

[3] https://github.com/open-mmlab/mmdetection

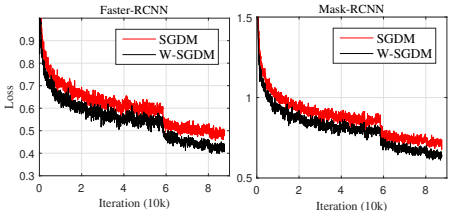

Fig. 2. Training loss curves on COCO by ResNet50.

Table 6. Rank1(%) and mAP(%) on Market1501 and DukeMTMC-reID. Δ means the gain of W-Adam over Adam.

Dataset		Market1501		DukeMTMC	
Backbone	Algorithm	Rank1	mAP	Rank1	mAP
R18	Adam	91.7	77.8	82.5	68.8
	W-Adam	91.8	79.2	83.5	70.4
	Δ	↑0.1	↑1.4	↑1.0	↑1.6
R50	Adam	94.5	85.9	86.4	76.4
	W-Adam	94.5	86.5	87.5	77.2
	Δ	↑0.0	↑0.6	↑1.1	↑0.8
R101	Adam	94.5	87.1	87.6	77.6
	W-Adam	95.0	87.9	88.2	78.3
	Δ	↑0.5	↑0.8	↑0.6	↑0.7

transformer backbones. Fig. 2 shows the training loss curves of Faster-RCNN and Mask-RCNN with ResNet50 backbone. One can see that W-SGDM accelerates the training process and achieves a more favorable local minimum than SGDM. This experiment clearly validates that EFW can be readily embedded into existing optimizers without extra hyper-parameter tuning.

4.4 Person Re-identification

We then use two widely used Person ReID benchmarks, Market1501 [46] and DukeMTMC-ReID [30], to show that W-Adam can also be easily adopted into pre-trained models without extra hyperparameter tuning. In this task, the Adam with L_2 regularization weight decay usually outperforms other optimizers and its hyperparameters have been well-tuned. The person ReID baselines in [22] are used[4]. The default hyperparameters of Adam, such as learning rate and weight decay, are directly applied to W-Adam. The experiments are repeated 4 times, and the average results are reported. Table 6 shows the Rank1 and mAP on Market1501 and DukeMTMC-ReID with ResNet18, ResNet50 and ResNet101 backbones. It is clear that W-Adam outperforms Adam, especially in mAP. This experiment again demonstrates the advantages of EFW as a general DNN optimization technique.

4.5 Ablation Study

Hyper-parameter Tuning: We first tune the dampening parameter ϵ and momentum parameter α. A too small ϵ cannot improve the condition number of the statistic matrix, while a too large ϵ will suppress the useful information in the second order statistics. The results of ResNet18 trained by W-SGDM on CIFAR100 with different ϵ and α are shown in Table 7 and Table 8. We choose the settings with $\epsilon = 1e^{-3}$ and $\alpha = 0.95$ as our default settings. We then tune T_{xx} and T_{svd} to balance the performance and efficiency. Because of the high computational cost of SVD decomposition, T_{svd} should be set larger than T_{xx}. We test six combinations of T_{xx} and T_{svd}, and report their testing accuracies and training time per epoch (sec/epoch) in Table 9. The baseline methods are SGDM

[4] https://github.com/michuanhaohao/reid-strong-baseline

Table 7. Testing accuracies (%) of ResNet18 by W-SGDM on CIFAR100 w.r.t. dampening ϵ.

ϵ	$1e^{-4}$	$5e^{-4}$	$\mathbf{1e^{-3}}$	$5e^{-3}$	$1e^{-2}$
Acc	79.05	79.18	**79.28**	79.12	78.88

Table 8. Testing accuracy (%) of ResNet18 by W-SGDM on CIFAR100 w.r.t. momentum α.

α	0.5	0.8	0.9	**0.95**	0.99	0.999
Acc	79.08	79.15	79.25	**79.28**	78.88	78.50

Table 9. Testing accuracy (%) and training efficiency of ResNet18 by W-SGDM and W-Adam on CIFAR100 w.r.t. T_{xx} and T_{svd}.

Algorithm	T_{xx} / T_{svd}	5 / 50	10 / 100	20 / 200	**50 / 500**	100 / 1000	200 / 2000	baseline
W-SGDM	Acc	79.40	79.33	79.23	**79.28**	79.11	79.02	77.20
	Sec/epoch	85.50	58.32	38.93	**29.78**	26.03	24.25	23.45
W-Adam	Acc	78.84	78.79	78.76	**78.75**	78.67	78.40	77.23
	Sec/epoch	90.17	60.1	40.9	**30.18**	27.14	25.55	24.21

Table 10. Testing accuracy (%) and training efficiency of whitening methods on CIFAR100/CIFAR10.

	Dataset	CIFAR100			CIFAR10		
Method	Model	R18	R50	V11	R18	R50	V11
SGDM-IterNorm	Acc	77.15	79.65	72.30	95.30	95.52	92.45
	sec/epoch	109.27	388.08	98.85	103.12	367.73	96.49
SGDM-ND	Acc	78.65	80.20	72.70	95.37	95.73	93.03
	sec/epoch	65.37	218.34	32.18	64.12	213.45	30.39
W-SGDM	Acc	79.28	80.90	73.42	95.43	95.80	92.95
	sec/epoch	29.78	95.05	13.98	29.02	92.12	13.51

and AdamW. We can see that EFW costs less than 30% additional training time over the original SGDM/AdamW but achieves convincing performance gain over them. The combination of $T_{xx} = 50$ and $T_{svd} = 500$ can balance the performance and efficiency well, and it is chosen as our default setting.

Training Efficiency: We further compare EFW with another two representative whitening methods, *i.e.*, ND [39] and IterNorm [11], with the SGDM optimizer. ND and IterNorm need to redefine the forward propagation of DNN models by replacing the normalization layers with whitening layers. Table 10 shows the testing accuracy (%) and training efficiency (sec/epoch) of different whitening methods on CIFAR100/CIFAR10. One can see that the proposed W-SGDM clearly outperforms ND and IterNorm in both accuracy and efficiency. ND and IterNorm cost more than two times the training time of EFW. Clearly, EFW is much more efficient to perform feature whitening and achieves a more favorable performance boost than conventional whitening methods. It overcomes the major drawbacks of whitening methods and inherits their advantages.

5 Conclusion

In this work, we proposed a novel DNN optimization technique, namely Embedded Feature Whitening (EFW), to address the drawbacks of conventional feature whitening methods, such as large computation cost, extra parameter introduction, inapplicable to pre-trained DNN models, and so on. Different from the existing feature whitening methods, which usually perform a whitening operation on features during forward propagation, EFW only adjusts the gradient of weight with the whitening matrix without changing the forward and backward propagation processes of DNN model training. Meanwhile, we developed the associated momentum, statistics matrix computation, adaptive dampening and gradient norm recovery techniques to make EFW effective and efficient to use. By adopting EFW to the popular SGDM and Adam optimizers, the resulting W-SGDM and W-Adam methods demonstrated their superiority to other leading DNN optimizers in various vision tasks with acceptable extra computation, including image classification, detection, segmentation and person ReID.

References

1. Chen, J., Zhou, D., Tang, Y., Yang, Z., Cao, Y., Gu, Q.: Closing the generalization gap of adaptive gradient methods in training deep neural networks. arXiv preprint arXiv:1806.06763 (2018)
2. Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., et al.: Mmdetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155 (2019)
3. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(Jul), 2121–2159 (2011)
4. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)
5. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017)
6. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
8. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
9. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
10. Huang, L., Yang, D., Lang, B., Deng, J.: Decorrelated batch normalization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 791–800 (2018)
11. Huang, L., Zhou, Y., Zhu, F., Liu, L., Shao, L.: Iterative normalization: Beyond standardization towards efficient whitening. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4874–4883 (2019)
12. Huang, X., Belongie, S.: Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1501–1510 (2017)
13. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
15. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
16. Lei Ba, J., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
17. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2117–2125 (2017)
18. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)

19. Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., Han, J.: On the variance of the adaptive learning rate and beyond. arXiv preprint arXiv:1908.03265 (2019)
20. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10012–10022 (2021)
21. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
22. Luo, H., Gu, Y., Liao, X., Lai, S., Jiang, W.: Bag of tricks and a strong baseline for deep person re-identification. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. pp. 0–0 (2019)
23. Ma, X.: Apollo: An adaptive parameter-wise diagonal quasi-newton method for nonconvex stochastic optimization. arXiv preprint arXiv:2009.13586 (2020)
24. Martens, J., Grosse, R.: Optimizing neural networks with kronecker-factored approximate curvature. In: International conference on machine learning. pp. 2408–2417. PMLR (2015)
25. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
26. Pascanu, R., Mikolov, T., Bengio, Y.: Understanding the exploding gradient problem. CoRR, abs/1211.5063 **2** (2012)
27. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International conference on machine learning. pp. 1310–1318 (2013)
28. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural networks **12**(1), 145–151 (1999)
29. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
30. Ristani, E., Solera, F., Zou, R., Cucchiara, R., Tomasi, C.: Performance measures and a data set for multi-target, multi-camera tracking. In: European conference on computer vision. pp. 17–35. Springer (2016)
31. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)
32. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? Advances in neural information processing systems **31** (2018)
33. Siarohin, A., Sangineto, E., Sebe, N.: Whitening and coloring batch transform for gans. arXiv preprint arXiv:1806.00420 (2018)
34. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
35. Teye, M., Azizpour, H., Smith, K.: Bayesian uncertainty estimation for batch normalized deep networks. arXiv preprint arXiv:1802.06455 (2018)
36. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022 (2016)
37. Wu, Y., He, K.: Group normalization. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 3–19 (2018)
38. Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., Mahoney, M.W.: Adahessian: An adaptive second order optimizer for machine learning. arXiv preprint arXiv:2006.00719 (2020)
39. Ye, C., Evanusa, M., He, H., Mitrokhin, A., Goldstein, T., Yorke, J.A., Fermüller, C., Aloimonos, Y.: Network deconvolution. arXiv preprint arXiv:1905.11926 (2019)

40. Yong, H., Huang, J., Hua, X., Zhang, L.: Gradient centralization: A new optimization technique for deep neural networks. In: European Conference on Computer Vision. pp. 635–652. Springer (2020)
41. Yong, H., Huang, J., Meng, D., Hua, X., Zhang, L.: Momentum batch normalization for deep learning with small batch size. In: European Conference on Computer Vision. pp. 224–240. Springer (2020)
42. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
43. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530 (2016)
44. Zhang, H., Chen, W., Liu, T.Y.: Train feedfoward neural network with layer-wise adaptive rate via approximating back-matching propagation. arXiv preprint arXiv:1802.09750 (2018)
45. Zhang, M.R., Lucas, J., Hinton, G., Ba, J.: Lookahead optimizer: k steps forward, 1 step back. arXiv preprint arXiv:1907.08610 (2019)
46. Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., Tian, Q.: Scalable person re-identification: A benchmark. In: Proceedings of the IEEE international conference on computer vision. pp. 1116–1124 (2015)
47. Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., Duncan, J.S.: Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. arXiv preprint arXiv:2010.07468 (2020)