Learning Deep Architectures via Generalized Whitened Neural Networks

Ping Luo [1 2]

Abstract

Whitened Neural Network (WNN) is a recent advanced deep architecture, which improves convergence and generalization of canonical neural networks by whitening their internal hidden representation. However, the whitening transformation increases computation time. Unlike WNN that reduced runtime by performing whitening every thousand iterations, which degenerates convergence due to the ill conditioning, we present generalized WNN (GWNN), which has three appealing properties. First, GWNN is able to learn compact representation to reduce computations. Second, it enables whitening transformation to be performed in a short period, preserving good conditioning. Third, we propose a data-independent estimation of the covariance matrix to further improve computational efficiency. Extensive experiments on various datasets demonstrate the benefits of GWNN.

1. Introduction

Deep neural networks (DNNs) have improved performances of many applications, as the non-linearity of DNNs provides expressive modeling capacity, but it also makes DNNs difficult to train and easy to overfit the training data.

Whitened neural network (WNN) (Desjardins et al., 2015), a recent advanced deep architecture, is ideally to solve the above difficulties. WNN extends batch normalization (BN) (Ioffe & Szegedy, 2015) by normalizing the internal hidden representation using whitening transformation instead of standardization. Whitening helps regularize each diagonal block of the Fisher Information Matrix (FIM) to be an approximation of the identity matrix. This is an appealing property, as training WNN using stochastic gradient descent (SGD) mimics the fast convergence of natural gradient descent (NGD) (Amari & Nagaoka, 2000). The whitening transformation also improves generalization. As demonstrated in (Desjardins et al., 2015), WNN exhibited superiority when being applied to various network architectures, such as autoencoder and convolutional neural network, outperforming many previous works including SGD, RMSprop (Tieleman & Hinton, 2012), and BN.

Although WNN is able to reduce the number of training iterations and improve generalization, it comes with a price of increasing training time, because eigen-decomposition occupies large computations. The runtime scales up when the number of hidden layers that require whitening transformation increases. We revisit WNN by breaking down its performance and show that its main runtime comes from two aspects, 1) computing full covariance matrix for whitening and 2) solving singular value decomposition (SVD). Previous work (Desjardins et al., 2015) suggests to overcome these problems by a) using a subset of training data to estimate the full covariance matrix and b) solving the SVD every hundreds or thousands of training iterations. Both of them rely on the assumption that the SVD holds in this period, but it is generally not true. When this period becomes large, WNN degenerates to canonical SGD due to ill conditioning of FIM.

We propose generalized WNN (GWNN), which possesses the beneficial properties of WNN, but significantly reduces its runtime and improves its generalization. We introduce two variants of GWNN, including pre-whitening and post-whitening GWNNs. The former one whitens a hidden layer's input values, whilst the latter one whitens the pre-activation values (hidden features). GWNN has three appealing characteristics. First, compared to WNN, GWNN is capable of learning more compact hidden representation, such that the SVD can be approximated by a few top eigenvectors to reduce computation. This compact representation also improves generalization. Second, it enables the whitening transformation to be performed in a short period, maintaining conditioning of FIM. Third, by exploiting knowledge of the distribution of the hidden features, we calculate the covariance matrix in an analytical form to further improve computational efficiency.

[1]Guangdong Provincial Key Laboratory of Computer Vision and Virtual Reality Technology, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China [2]Multimedia Laboratory, The Chinese University of Hong Kong, Hong Kong. Correspondence to: Ping Luo <pluo@ie.cuhk.edu.hk>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

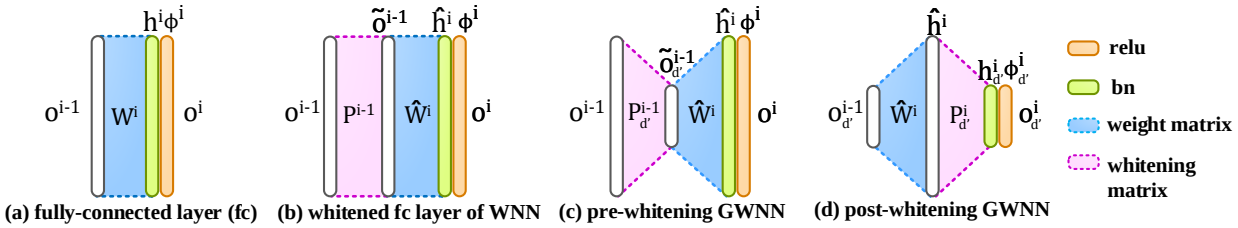

Figure 1. Comparisons of differnet architectures. An ordinary fully-connected (fc) layer can be adapted into (b) a whitened fc layer, (c) a pre-GWNN layer, and (d) a post-GWNN layer. (c) and (d) learn more compact representation than (b) does.

2. Notation and Background

We begin by defining the basic notation for feed-forward neural network. A neural network transforms an input vector o^0 to an output vector o^ℓ through a series of ℓ hidden layers $\{o^i\}_{i=1}^{\ell}$. We assume each layer has identical dimension for the simplicity of notation *i.e.* $\forall o^i \in \mathbb{R}^{d\times 1}$. In this case, all vectors and matrixes in the following should have d rows unless otherwise stated. As shown in Fig.1 (a), each fully-connected (fc) layer consists of a weight matrix, W^i, and a set of hidden neurons, h^i, each of which receives as input a weighted sum of outputs from the previous layer. We have $h^i = W^i o^{i-1}$. In this work, we take fully-connected network as an example. Note that the above computation can be also applied to a convolutional network, where an image patch is vectorized as a column vector and represented by o^{i-1} and each row of W^i represents a filter.

As the recent deep architectures typically stack a batch normalization (BN) layer before the pre-activation values, we do not explicitly include a bias term when computing h^i, because it is normalized in BN, such that $\phi^i = \frac{h^i - \mathbb{E}[h^i]}{\sqrt{\text{Var}[h^i]}}$, where the expectation and variance are computed over a minibatch of samples. LeCun et al. (2002) showed that such normalization speeds up convergence even when the hidden features are not decorrelated. Furthermore, output of each layer is calculated by a nonlinear activation function. A popular choice is the rectified linear unit, $\text{relu}(x) = \max(0, x)$. The precise computation for an output is $o^i = \max(0, \text{diag}(\alpha^i)\phi^i + \beta^i)$, where $\text{diag}(x)$ represents a matrix whose diagonal entries are x. α^i and β^i are two vectors that scale and shift the normalized features, in order to maintain the network's representation capacity.

2.1. Whitened Neural Networks

This section revisits whitened neural networks (WNN). Any neural architecture can be adapted to a WNN by stacking a whitening transformation layer after the layer's input. For example, Fig.1 (b) adapts a fc layer as shown in (a) into a whitened fc layer. Its information flow becomes

$$\widetilde{o}^{i-1} = P^{i-1}(o^{i-1} - \mu^{i-1}), \;\; \hat{h}^i = \hat{W}^i \widetilde{o}^{i-1}, \qquad (1)$$
$$\phi^i = \frac{\hat{h}^i}{\sqrt{\text{Var}[\hat{h}^i]}}, \;\; o^i = \max(0, \text{diag}(\alpha^i)\phi^i + \beta^i),$$

where μ^{i-1} represents a centering variable, $\mu^{i-1} = \mathbb{E}[o^{i-1}]$. P^{i-1} is a whitening matrix whose rows are obtained from eigen-decomposition of Σ^{i-1}, which is the covariance matrix of the input, $\Sigma^{i-1} = \mathbb{E}[(o^{i-1} - \mu^{i-1})(o^{i-1} - \mu^{i-1})^\mathsf{T}]$. The input is decorrelated by P^{i-1} in the sense that its covariance matrix becomes an identity matrix, *i.e.* $\mathbb{E}[\widetilde{o}^{i-1}\widetilde{o}^{i-1^\mathsf{T}}] = I$. To avoid ambiguity, we use '$\hat{}$' to distinguish the variables in WNN and the canonical fc layer whenever necessary. For instance, $\hat{W}^i$ represents a whitened weight matrix. In Eqn.(1), computation of the BN layer has been simplified because we have $\mathbb{E}[\hat{h}^i] = \hat{W}^i P^{i-1}(\mathbb{E}[o^{i-1}] - \mu^{i-1}) = 0$.

We define θ to be a vector consisting of all the whitened weight matrixes concatenated together, $\theta = \{\text{vec}(\hat{W}^1)^\mathsf{T}, \text{vec}(\hat{W}^2)^\mathsf{T}, ..., \text{vec}(\hat{W}^\ell)^\mathsf{T}\}$, where $\text{vec}(\cdot)$ is an operator that vectorizes a matrix by stacking its columns. Let $L(o^\ell, y; \theta)$ denote a loss function of WNN, which measures the disagreement between a prediction o^ℓ made by the network, and a target y. WNN is trained by minimizing the loss function with respect to the parameter vector θ and two constraints

$$\min_{\theta} \;\; L(o^\ell, y; \theta) \qquad (2)$$
$$\text{s.t. } \mathbb{E}[\widetilde{o}^{i-1}\widetilde{o}^{i-1^\mathsf{T}}] = I, \;\; h^i - \mathbb{E}[h^i] = \hat{h}^i, \;\; i = 1...\ell.$$

To satisfy the first constraint, P^{i-1} is obtained by decomposing the covariance matrix, $\Sigma^{i-1} = U^{i-1}S^{i-1}U^{i-1^\mathsf{T}}$. We choose $P^{i-1} = (S^{i-1})^{-\frac{1}{2}}U^{i-1^\mathsf{T}}$, where S^{i-1} is a diagonal matrix whose diagonal elements are eigenvalues and U^{i-1} is an orthogonal matrix of eigenvectors. The first constraint holds under the construction of eigen-decomposition.

The second constraint, $h^i - \mathbb{E}[h^i] = \hat{h}^i$, enforces that the centered hidden features are the same, before and after

adapting a fc layer to WNN, as shown in Fig.1 (a) and (b). In other words, it ensures that their representation powers are identical. By combing the computations in Fig.1 (a) and Eqn.(1), the second constraint implies that $\|(h^i - \mathbb{E}[h^i]) - \hat{h}^i\|_2^2 = \|(W^i o^{i-1} - W^i \mu^{i-1}) - \hat{W}^i \widetilde{o}^{i-1}\|_2^2 = 0$, which has a closed-form solution, $\hat{W}^i = W^i (P^{i-1})^{-1}$. To see this, we have $\hat{h}^i = W^i (P^{i-1})^{-1} P^{i-1}(o^{i-1} - \mu^{i-1}) = W^i(o^{i-1} - \mu^{i-1}) = h^i - \mathbb{E}[h^i]$. The representation capacity can be preserved by mapping the whitened weight matrix from the ordinary weight matrix.

Conditioning of the FIM. Here we show that WNN improves training efficiency by conditioning the Fisher information matrix (FIM) (Amari & Nagaoka, 2000). A FIM, denoted as F, consists of $\ell \times \ell$ blocks. Each block is indexed by F_{ij}, representing the covariance (co-adaptation) between the whitened weight matrixes of the i-th and j-th layers. We have $F_{ij} = \mathbb{E}[\mathrm{vec}(\delta \hat{W}^i)\mathrm{vec}(\delta \hat{W}^j)^\mathsf{T}]$, where $\delta \hat{W}^i$ indicates the gradient of the i-th whitened weight matrix. For example, the gradient of $\hat{W}^i$ is achieved by $\widetilde{o}^{i-1}(\delta \hat{h}^i)^\mathsf{T}$, as illustrated in Eqn.(1). We have $\mathrm{vec}(\delta \hat{W}^i) = \mathrm{vec}(\widetilde{o}^{i-1}(\delta \hat{h}^i)^\mathsf{T}) = \delta \hat{h}^i \otimes \widetilde{o}^{i-1}$, where $\otimes$ denotes the Kronecker product. In this case, F_{ij} can be rewritten as $\mathbb{E}[(\delta \hat{h}^i \otimes \widetilde{o}^{i-1})(\delta \hat{h}^j \otimes \widetilde{o}^{j-1})^\mathsf{T}] = \mathbb{E}[\delta \hat{h}^i (\delta \hat{h}^j)^\mathsf{T} \otimes \widetilde{o}^{i-1}(\widetilde{o}^{j-1})^\mathsf{T}]$. By assuming $\delta \hat{h}$ and $\widetilde{o}$ are independent as demonstrated in (Raiko et al., 2012), F_{ij} can be approximated by $\mathbb{E}[\delta \hat{h}^i (\delta \hat{h}^j)^\mathsf{T}] \otimes \mathbb{E}[\widetilde{o}^{i-1}(\widetilde{o}^{j-1})^\mathsf{T}]$. As a result, when $i = j$, each diagonal block of F, F_{ii}, has a block diagonal structure because we have $\mathbb{E}[\widetilde{o}^{i-1}(\widetilde{o}^{i-1})^\mathsf{T}] = I$ as shown in Eqn.(2), which improves conditioning of FIM and thus speeds up training. In general, WNN regularizes the diagonal blocks of FIM and achieves stronger conditioning than those methods (LeCun et al., 2002; Tieleman & Hinton, 2012) that regularized the diagonal entries.

Training WNN. Alg.1 summarizes training of WNN. At the 1^{st} line, the whitened weight matrix $\hat{W}_0^i$ is initialized by W^i of the ordinary fc layer, which can be pretrained or sampled from a Gaussian distribution. The 4^{th} line shows that $\hat{W}_t^i$ is updated in each iteration t using SGD. The first and second constraints are achieved in the 7^{th} and 8^{th} lines respectively. For example, the 8^{th} line ensures that the hidden features are the same before and after updating the whitening matrix. As the distribution of the hidden representation changes after every update of the whitened weight matrix, to maintain good conditioning of FIM, the whitening matrix, P^{i-1}, needs to be reconstructed frequently by performing eigen-decomposition on Σ^{i-1}, which is estimated using N samples. N is typically 10^4 in experiments. However, this raw strategy increases computation time. Desjardins et al. (2015) performed whitening in every τ iterations as shown in the 5^{th} line of Alg.1 to reduce computations, *e.g.* $\tau = 10^3$.

How good is the conditioning of the FIM by using Alg.1? We measure the similarity of the covariance matrix, $\mathbb{E}[\widetilde{o}^{i-1}(\widetilde{o}^{i-1})^\mathsf{T}]$, with the identity matrix I. This is called the orthogonality. We employ Pearson's correlation[1] as the similarity between two matrixes. Intuitively, this measure has a value between -1 and $+1$, representing negative and positive correlations. Larger values indicate higher orthogonality. Fig.2 visualizes four randomly generated covariance matrixes, where (a,b) are sampled from a uniform distribution between 0 and 1. Fig.2 (c,d) are generated by truncating different numbers of columns of a randomly generated orthogonal matrix. For instance, (a,b) have small similarity with respect to the identity matrix. In contrast, when the correlation equals 0.65 as shown in (c), all entries in the diagonal are larger than 0.9 and more than 80% off-diagonal entries have values smaller than 0.1. Furthermore, Pearson's correlation is insensitive to the size of matrix, such that orthogonality of different layers can be compared together. For example, although matrixes in

Algorithm 1 Training WNN

1: **Init:** initial network parameters θ, α^i, β^i; whitening matrix $P^{i-1} = I$; iteration $t = 0$; $\hat{W}_t^i = W^i$; $\forall i \in \{1...\ell\}$.
2: **repeat**
3: **for** $i = 1$ **to** ℓ **do**
4: update whitened weight matrix $\hat{W}_t^i$ and parameters α_t^i, β_t^i using SGD.
5: **if** $\mathrm{mod}(t, \tau) = 0$ **then**
6: store old whitening matrix $P_o^{i-1} = P^{i-1}$.
7: construct new matrix P^{i-1} by eigen-decomposition on Σ^{i-1}, which is estimated using N samples.
8: transform weight matrix $\hat{W}_t^i = \hat{W}_t^i P_o^{i-1} (P^{i-1})^{-1}$.
9: **end if**
10: **end for**
11: $t = t + 1$.
12: **until** convergence

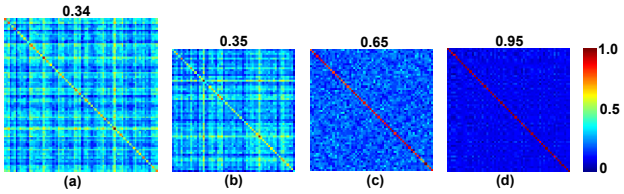

Figure 2. Visualizations of different covariance matrixes (a)-(d), which have different Pearson's correlations (top) with respect to an identity matrix. Larger Pearson's correlation indicates higher orthogonality. (a,b) are sampled from a uniform distribution between 0 and 1. (c,d) are generated by truncating a random orthogonal matrix with different numbers of columns. The colorbar (right) indicates the value of each entry in these matrixes.

[1] Given an identity matrix, I, and a covariance matrix, Σ, the Pearson's correlation between them is defined as $\mathrm{corr}(\Sigma, I) = \frac{\overline{\mathrm{vec}}(\Sigma)^\mathsf{T} \overline{\mathrm{vec}}(I)}{\sqrt{\overline{\mathrm{vec}}(\Sigma)^\mathsf{T} \overline{\mathrm{vec}}(\Sigma) \cdot \overline{\mathrm{vec}}(I)^\mathsf{T} \overline{\mathrm{vec}}(I)}}$, where $\overline{\mathrm{vec}}(\Sigma)$ is a normalized vector by subtracting mean of all entries.

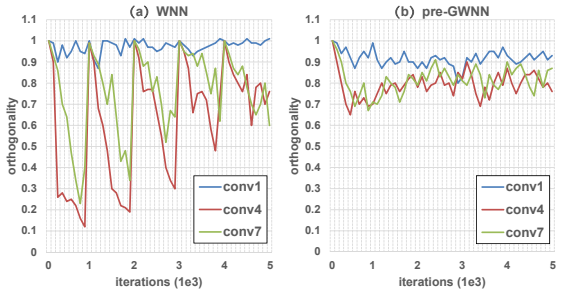

Figure 3. Comparisons of conditioning when training a network-in-network (Lin et al., 2014) on CIFAR-10 (Krizhevsky, 2009) by using (a) WNN and (b) pre-GWNN. Compared to (b), the orthogonalities of three different layers in (a) have large fluctuations due to the ill conditioning of whitening, which is performed in a large period τ. As a result, when τ increases, WNN will degenerate to the canonical SGD.

(a) and (b) have different sizes, they have similar value of orthogonality when they are sampled from the same distribution.

As shown in Fig.3 (a), we adopt network-in-network (NIN) (Lin et al., 2014) that is trained on CIFAR-10 (Krizhevsky, 2009), and plot the orthogonalities of three different convolutional layers, which are whitened every $\tau = 10^3$ iterations by using Alg.1. We see that orthogonality values during training have large fluctuations except those of the first convolutional layer, abbreviated as 'conv1'. This is because the distributions of deeper layers' inputs change after the whitened weight matrixes have been updated, leading to ill-conditions of the whitening matrixes, which are estimated in a large interval. In fact, large τ will degenerate WNN to canonical SGD. However, 'conv1' uses image data as inputs, whose distribution is typically stable during training. Its whitening matrix can be estimated once at the beginning and fixed in the entire training stage.

In the section below, we present generalized whitened neural networks to improve conditioning of FIM while reducing computation time.

3. Generalized Whitened Neural Networks

We present two types of generalized WNN (GWNN), including pre-whitening and post-whitening GWNNs. Both models share beneficial properties of WNN, but have lower computation time.

3.1. Pre-whitening GWNN

This section introduces pre-whitening GWNN, abbreviated as pre-GWNN, which performs whitening transformation before applying the weight matrix (*i.e.* whiten the input), as illustrated in Fig.1 (c). When adapting a fc layer to pre-GWNN, the whitening matrix is truncated by removing those eigenvectors that have small eigenvalues, in order to learn compact representation. This allows the input vector to vary its length, so as to gradually adapt the learned representation to informative patterns with high variations, but not noises. Learning pre-GWNN is formulated analogously to learning WNN in Eqn.(2), but with one additional constraint truncated the rank of the whitening matrix,

$$\min_{\theta} \; L(o^{\ell}, y; \theta) \tag{3}$$
$$\text{s.t. } \operatorname{rank}(P^{i-1}) \leq d', \;\; \mathbb{E}[\widetilde{o}^{i-1}_{d'} \widetilde{o}^{i-1^{\mathsf{T}}}_{d'}] = I,$$
$$h^i - \mathbb{E}[h^i] = \hat{h}^i, \;\; i = 1...\ell.$$

Let d be the dimension of the original fc layer. By combining Eqn.(2), we have $P^{i-1} = (S^{i-1})^{-\frac{1}{2}} U^{i-1^{\mathsf{T}}} \in \mathbb{R}^{d \times d}$, which is truncated by using $P^{i-1}_{d'} = (S^{i-1}_{d'})^{-\frac{1}{2}} U^{i-1^{\mathsf{T}}}_{d'} \in \mathbb{R}^{d' \times d}$, where $S^{i-1}_{d'}$ is achieved by keeping rows and columns associated with the first d' large eigenvalues, whilst $U^{i-1}_{d'}$ contains the corresponding d' eigenvectors. The value of d' can be tuned using a validation set. For simplicity, we choose $d' = \frac{d}{2}$, which works well throughout our experiments. This is inspired by the finding in (Zhang et al., 2015), who disclosed that the first 50% eigenvectors contribute over 95% energy in a deep model.

More specifically, pre-GWNN first projects an input vector to a d' low-dimensional space, $\widetilde{o}^{i-1}_{d'} = P^{i-1}_{d'}(o^{i-1} - \mu^{i-1}) \in \mathbb{R}^{d' \times 1}$. The whitened weight matrix then produces a hidden feature vector of d dimensions, which has the same length as the ordinary fc layer, *i.e.* $\hat{h}^i = \hat{W}^i \widetilde{o}^{i-1}_{d'} \in \mathbb{R}^{d \times 1}$, where $\hat{W}^i = W^i (P^{i-1}_{d'})^{-1} \in \mathbb{R}^{d \times d'}$. The computations of BN and the nonlinear activation are identical to Eqn.(1).

Training pre-GWNN is similar to Alg.1. The main modification is produced at the 7^{th} line in order to reduce runtime. Although Alg.1 decreases number of iterations when training converged, each iteration has additional computation time for eigen-decomposition. For example, in WNN, the required computation of full singular value decomposition (SVD) is typically $\mathcal{O}(Nd^2)$, where N represents the number of samples employed to estimate the covariance matrix. In particular, when we have ℓ whitened layers and T is the number of iterations, all whitening transformations occupy $\mathcal{O}(\frac{Nd^2T\ell}{\tau})$ runtime in the entire training stage. In contrast, pre-GWNN performs the popular online estimation for the top d' eigenvectors in $P^{i-1}_{d'}$ such as online SVD (Shamir, 2015; Povey et al., 2015), instead of using full SVD as WNN did. This difference reduces runtime to $\mathcal{O}(\frac{(N+M)d'T\ell}{\tau'})$, where τ' represents the whitening interval in GWNN and M is the number of samples used to estimate the top eigenvectors. We have $M = N$ as employed in previous works.

For pre-GWNN, reducing runtime and improving conditioning is a tradeoff, since the former requires to increase τ' but the latter requires to decrease it. When $M = N$ and $d' = \frac{d}{2}$, we compare the runtime complexity of pre-GWNN to that of WNN, and obtain a ratio of $\frac{d\tau'}{\tau}$, which tells us that whitening can be performed in a short interval without increasing runtime. For instance, as shown in Fig.3 (b) when $\tau' = 20$, orthogonalities are well preserved and more stable than those in (a). In this case, pre-GWNN reduces computations of WNN by at least $20\times$ when $d > \tau$, which is a typical choice in recent deep architectures (Krizhevsky et al., 2012; Lin et al., 2014) where $d \in \{1024, 2048, 4096\}$.

3.2. Post-whitening GWNN

Another variant we propose is post-whitening GWNN, abbreviated as post-GWNN. Unlike WNN and pre-GWNN, post-GWNN performs whitening transformation after applying the weight matrix (*i.e.* whiten the feature), as illustrated in Fig.1 (d). In general, post-GWNN reduces runtime to $\mathcal{O}(\frac{(N'+M)d'T\ell}{\tau'})$, where $N' \ll N$.

Fig.1 (d) shows how to adapt a fc layer to post-GWNN. Suppose $o^{i-1}_{d'}$ has been whitened by $P^{i-1}_{d'}$ in the previous layer, at the i-th layer we have

$$\hat{h}^i = \hat{W}^i(o^{i-1}_{d'} - \mu^{i-1}_{d'}), \;\; h^i_{d'} = P^i_{d'}\hat{h}^i, \tag{4}$$
$$\phi^i_{d'} = \frac{h^i_{d'}}{\sqrt{\mathrm{Var}[h^i_{d'}]}}, \;\; o^i_{d'} = \max(0, \mathrm{diag}(\alpha^i_{d'})\phi^i_{d'} + \beta^i_{d'}),$$

where $\mu^{i-1}_{d'} = \mathbb{E}[o^{i-1}_{d'}]$. In Eqn.(4), a feature vector $\hat{h}^i \in \mathbb{R}^{d\times 1}$ is first produced by applying a whitened weight matrix on the input, in order to recover the original feature length as the fc layer. A whitening matrix then projects $\hat{h}^i$ to a decorrelated feature vector $h^i_{d'} \in \mathbb{R}^{d'\times 1}$. We have $\hat{W}^i = W^i(P^{i-1}_{d'})^{-1} \in \mathbb{R}^{d\times d'}$, where $P^{i-1}_{d'} = (S^{i-1}_{d'})^{-\frac{1}{2}}U^{i-1^\mathsf{T}}_{d'} \in \mathbb{R}^{d'\times d}$, and U^{i-1} and S^{i-1} contain eigenvectors and eigenvalues of the hidden features at the $i-1$-th layer.

Conditioning. Here we disclose that whitening hidden features also enforces good conditioning of FIM. At this point, we have decorrelated the hidden features by satisfying $\mathbb{E}[h^i_{d'}h^{i^\mathsf{T}}_{d'}] = I$. Then $h^i_{d'}$ follows a standard multivariate Gaussian distribution, $h^i_{d'} \sim \mathcal{N}(0, I)$. As a result, the layer's output follows a rectified Gaussian distribution, which is uncorrelated as presented in remark 1. In post-GWNN, whitening hidden features of the $i-1$-th layer improves conditioning for the i-th layer. To see this, by following the description in Sec.2.1, the diagonal block of FIM associated with the i-th layer can be written as $F_{ii} \approx \mathbb{E}[\delta\hat{h}^i(\delta\hat{h}^i)^\mathsf{T}] \otimes \mathbb{E}[(o^{i-1}_{d'} - \mu^{i-1}_{d'})(o^{i-1}_{d'} - \mu^{i-1}_{d'})^\mathsf{T}]$, where the parameters have low correlations since F_{ii} has a block diagonal structure.

Algorithm 2 Training post-GWNN

Init: initial $\theta, \alpha^i, \beta^i$; and $t = 0, t_w = k, \lambda = \frac{t_w}{k}$; $P^{i-1} = I$, $\hat{W}^i_t = W^i, \forall i \in \{1...\ell\}$.
repeat
  for $i = 1$ **to** ℓ **do**
    update $\hat{W}^i_t$, α^i_t, and β^i_t by SGD.
    if $\mathrm{mod}(t, \tau) = 0$ **then**
      store old $P^{i-1}_o = P^{i-1}_{d'}$.
      estimate mean and variance of $\hat{h}^i$ by a minibatch of N' samples or following remark 2 when $N' = 1$.
      update $P^{i-1}_{d'}$ by online SVD.
      transform $\hat{W}^i_t = \hat{W}^i_t P^{i-1}_o (P^{i-1}_{d'})^{-1}$.
      $t_w = 1$ and $\lambda = \frac{t_w}{k}$.
    end if
  end for
  $t = t + 1$.
  if $t_w < k$ **then** $t_w = t_w + 1$ **end if**
until convergence

Remark 1. *Let $h \sim \mathcal{N}(0, I)$ and $o = \max(0, Ah + b)$. Then $\mathbb{E}[(o_j - E[o_j])(o_k - \mathbb{E}[o_k])] \approx 0$ if A is a diagonal matrix, where j, k index any two entries of o and $j \neq k$.*

For remark 1, we have $A = \mathrm{diag}(\alpha^i_{d'})$ and $b = \beta^i_{d'}$. It tells us three things. First, by using whitening and BN, covariance of any two different entries of $o^i_{d'}$ approaches zero. Second, at the iteration when we construct $P^i_{d'}$, we can estimate the full covariance matrix of $\hat{h}^i$ using the mean and variance of $o^{i-1}_{d'}$, $\mathbb{E}[\hat{h}^i\hat{h}^{i^\mathsf{T}}] = \hat{W}^i\mathbb{E}[(o^{i-1}_{d'} - \mu^{i-1}_{d'})(o^{i-1}_{d'} - \mu^{i-1}_{d'})^\mathsf{T}]\hat{W}^{i^\mathsf{T}}$. The mean and variance can be estimated with a minibatch of samples *i.e.* $N' \ll N$. Third, to the extreme, when $N' = 1$, these statistics can still be computed in analytical forms leveraging remark 2.

Remark 2. *Let a random variable $x \sim \mathcal{N}(0, 1)$ and $y = \max(0, ax + b)$. Then $\mathbb{E}[y] = \frac{a}{\sqrt{2\pi}}e^{-\frac{b^2}{2a^2}} + \frac{b}{2}\Psi(-\frac{b}{\sqrt{2}a})$ and $\mathbb{E}[y^2] = \frac{ab}{\sqrt{2\pi}}e^{-\frac{b^2}{2a^2}} + \frac{a^2+b^2}{2}\Psi(-\frac{b}{\sqrt{2}a})$, where $\Psi(x) = 1 - \mathrm{erf}(x)$ and $\mathrm{erf}(x)$ is the error function.*

The above remark derives the mean and variance of a rectified output unit that has shift and scale parameters. It generalizes (Arpit et al., 2016) that presented a special case when $a = 1$ and $b = 0$. In that case, we have $\mathbb{E}[y] = \frac{1}{\sqrt{2\pi}}$ and $\mathrm{Var}[y] = \mathbb{E}[y^2] - \mathbb{E}[y]^2 = \frac{1}{2} - \frac{1}{2\pi}$, which are consistent with previous work.

Extensions. Remark 1 and 2 can be extended to other nonlinear activation functions, such as leaky rectified unit defined as $\mathrm{leakyrelu}(x) = \max(0, x) + a\min(0, x)$, where the slope of the negative part is controlled by the coefficient a, which is fixed in (Maas et al., 2013) and is learned in (He et al., 2015).

3.3. Training post-GWNN

Similar to pre-GWNN, the learning problem can be formulated as

$$\min_{\theta} \ \lambda L(o^\ell, y; \theta) + (1-\lambda)\sum_{i=1}^{\ell} L^{\text{feat}}(h^i, \hat{h}^i; \theta) \quad (5)$$
$$\text{s.t. } \text{rank}(P^i) \leq d', \ \ \mathbb{E}[h^i_{d'} h^{i\text{T}}_{d'}] = I, \ \ i = 1...\ell.$$

Eqn.(5) has two loss functions. Different from WNN and pre-GWNN where the feature equality constraint can be satisfied in a closed form, this constraint is treated as an auxiliary loss function in post-GWNN, defined as $L^{\text{feat}}(h^i, \hat{h}^i) = \frac{1}{2}\|(h^i - \mathbb{E}[h^i]) - \hat{h}^i\|_2^2$ and minimized in the training stage. It does not have an analytical solution because there is a nonlinear activation function between the weight matrix and the whitening matrix (*i.e.* in the previous layer). In Eqn.(5), λ is a coefficient that balances the contribution of two loss functions, and $1-\lambda$ is linearly decayed as $1-\lambda = \frac{k-t_w}{k}$, where $t_w = 1, 2, ..., k$. At each time after we update the whitening matrix, we start decay by setting $t_w = 1$ and k indicates the iterations at which we stop annealing.

Alg.2 summarizes the training procedure. It preforms online update of the top d' eigenvectors of the whitening matrix similar to pre-GWNN. In comparison, it decreases the runtime of whitening transformation to $\mathcal{O}(\frac{(N'+M)d'T\ell}{\tau'})$, which is $\frac{N+M}{N'+M}$ fold reduction with respect to pre-GWNN. For example, when $N = M$ and $N' = 1$, post-GWNN is capable of reducing computations of pre-GWNN and WNN by $2\times$ and $(2\tau')\times$ respectively, while maintaining better conditioning than these alternatives by choosing small τ'.

4. Empirical Studies

We compare WNN, pre-GWNN, and post-GWNN in the following aspects, including a) number of iterations when training converged, b) computation times for training, and c) generalization capacities on various datasets. We also conduct ablation studies with respect to 1) effect of the number of samples N to estimate the covariance matrix for pre-GWNN and 2) effect of N' for post-GWNN. Finally, we try to tune the value of d'.

Datasets. We employ the following datasets.
a) MNIST (Lecun et al., 1998) has $60,000$ 28×28 images of 10 handwritten digits (0-9) for training and another $10,000$ test images. $5,000$ images from the training set are randomly selected as a validation set.
b) CIFAR-10 (Krizhevsky, 2009) consists of $50,000$ 32×32 color images for training and $10,000$ images for testing. Each image is categorized into one of the 10 object labels. For CIFAR-10, $5,000$ images are chosen for validation.
c) CIFAR-100 (Krizhevsky, 2009) has the same number of images as CIFAR-10, but each image is classified into 100 categories. For CIFAR-100, we select $5,000$ images from training set for validation.
d) SVHN (Netzer et al., 2011) consists of color images of house numbers collected by Google Street View. The task is to predict the center digit (0-9) of each image, which is of size 32×32. There are $73,257$ images in the training set, $26,032$ images for test, and $531,131$ additional examples. We follow (Sermanet et al., 2012) to build a validation set by selecting 400 samples per class from the training set and 200 samples per class from the additional set. We didn't train on validation, which is for tuning hyperparameters.

Experimental Settings. We have two settings, an unsupervised and a supervised learning settings. First, following (Desjardins et al., 2015), we compare the above three approaches on the task of minimizing reconstruction error of an autoencoder on MNIST. The encoder consists of 4 fc sigmoidal layers, which have $1000, 500, 256$, and 30 hidden neurons respectively. The decoder is symmetric and untied with respect to the encoder. Second, for the task of image classification on CIFAR-10, -100, and SVHN, we employ the same network-in-network (NIN) (Lin et al., 2014) architecture, which has 9 convolutional layers and 3 pooling layers defined as[2]: $\text{conv}(192,5)$-$\text{conv}(160,1)$-$\text{maxpool}(3,2)$-$\text{conv}(96,1)$-$\text{conv}(192,5)$-$\text{conv}(192,1)$-$\text{avgpool}(3,2)$-$\text{conv}(192,1)$-$\text{conv}(192,5)$-$\text{conv}(192,1)$-$\text{conv}(l,1)$-$\text{avgpool}(8,8)$, where $l = 10$ for CIFAR-10 and SVHN and $l = 100$ for CIFAR-100. For all models, we use SGD with momentum of 0.9.

4.1. Comparisons of Convergence and Computations

We record the number of epochs and computation time, when training WNN, pre-, and post-GWNN on MNIST and CIFAR-100, respectively. We employ the first setting above for MNIST and the second setting for CIFAR-100. For both settings, hyperparamters are chosen by grid search on the validation sets. The search specifications of minibatch size, learning rate, and whitening interval τ are $\{64, 128, 256\}$, $\{0.1, 0.01, 0.001\}$, and $\{20, 50, 100, 10^3\}$, respectively. In particular, for WNN and pre-GWNN, the number of samples used to estimate the covariance matrix, N, is picked up from $\{10^3, \frac{10^4}{2}, 10^4\}$. For post-GWNN, N' is chosen to be the same as the minibatch size and the decay period $k = 0.1\tau$. For a fair comparison, we report the best performance on validation set for each approach, and didn't employ any data augmentation such as random image cropping and flipping.

[2]The 'conv', 'maxpool', and 'avgpool' represent convolution, max pooling, and average pooling respectively. Each convolutional layer is defined as $\text{conv}(\text{number of filters}, \text{ filter size})$. For each pooling layer, we have $\text{pool}(\text{kernel size}, \text{ stride})$. All convolutions have stride 1.

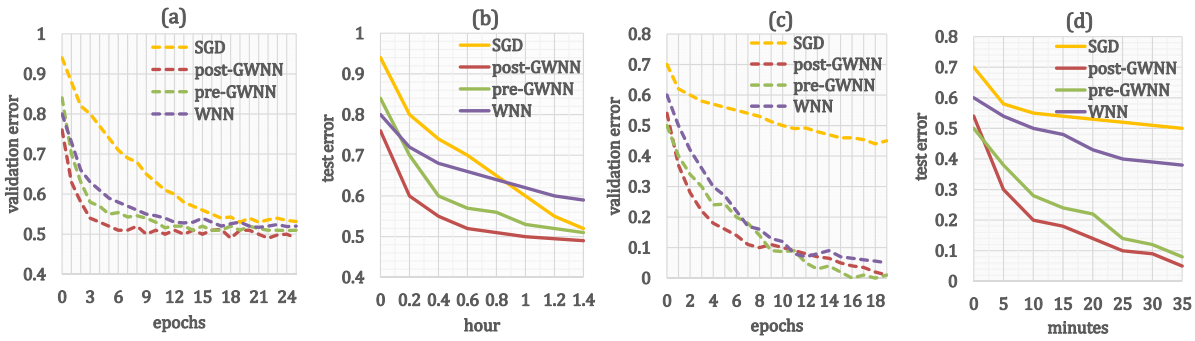

Figure 4. Training of WNN, pre-, post-GWNN on CIFAR-100 (a,b) and MNIST (c,d). (a) and (b) plot the convergence and computation time on the validation and test set of CIFAR-100 respectively. (c) and (d) report corresponding results on MNIST.

The convergence and computation time are reported in Fig.4 (a-d). We have several important observations. First, all three approaches converge much faster than the canonical network trained by SGD. Second, pre- and post-GWNN achieve better convergence than WNN on both datasets as shown in (a) and (c). Moreover, post-GWNN outperforms pre-GWNN. Third, post-GWNN significantly reduces computation time compared to all the other methods, as illustrated in (b) and (d). We see that although WNN reduces the number of epochs, it takes long time to train because its whitening transformation occupies large computations.

4.2. Performances on various Datasets

We evaluate WNN, pre-, and post-GWNN on CIFAR-10, -100, and SVHN datasets, and compare their classification accuracies to existing state-of-the-art methods. For all the datasets and approaches, we utilize the same network structure as mentioned in the second setting above. For two CIFAR datasets, we adopt minibatch size 64 and initial learning rate 0.1, which is reduced by half after every 25 epochs. We train for 250 epochs. As SVHN is a large dataset, we train for 100 epochs with minibatch size 128 and initial learning rate 0.05, which is reduced by half after every 10 epochs. We train on CIFAR-10 and -100 using both without and with data augmentation, which includes random cropping and horizontal flipping. For SVHN, we didn't augment data following (Sermanet et al., 2012). For all the methods, we shuffle samples at the beginning of every epoch. We use $N = \frac{10^4}{2}$ for WNN and pre-GWNN and $N' = 64$ for post-GWNN. For both pre- and post-GWNN, we have $M = N$ and $d' = \frac{d}{2}$. The other experimental settings are similar to Sec.4.1. Table 1 shows the results. We see that pre- and post-GWNN consistently achieve better results than those of WNN, and also outperform previous state-of-the-art approaches.

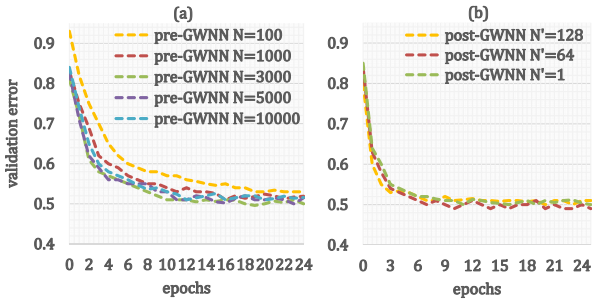

Figure 5. Training of pre- and post-GWNN on CIFAR-100. (a) visualizes the impact of different values of N for pre-GWNN, showing that performance degrades when N is small. (b) plots the impact of N' for post-GWNN, which is insensitive to small values of N'.

4.3. Ablation Studies

The following experiments are conducted on CIFAR-100 using pre- or post-GWNN. The first two experiments follow the setting as mentioned in Sec.4.1. First, we evaluate the effect of the number of samples, N, used to estimate the covariance matrix in pre-GWNN. We compare performances of using different values of N picked up from $\{10^2, 10^3, 3\times10^3, 5\times10^3, 10^4\}$. Fig.5 (a) plots the results. We see that performance can drop because of ill conditioning when N is small *e.g.* $N = 100$. When it is too large *e.g.* $N = 10^4$, we observe slightly overfitting. Second, Fig.5 (b) highlights the effect of N' in post-GWNN. We see that post-GWNN can work reasonably well when N' is small.

Finally, instead of treating $d' = \frac{d}{2}$ as a constant in training, we study the effect of tuning its value on the validation set using a simple heuristic strategy. If the validation error reduces more than 2% over 4 consecutive evaluations, we have $d' = d' - \textit{rate} \times d'$.

Table 1. Comparisons of test errors on various datasets. The top two performances are highlighted for each dataset.

	CIFAR-10	Error(%)
without augmentation	NIN (Lin et al., 2014)	10.47
	NIN+ALP (Agostinelli et al., 2015)	9.59
	Normalization Propagation (Arpit et al., 2016)	**9.11**
	BatchNorm (Ioffe & Szegedy, 2015)	9.41
	DSN (Lee et al., 2015)	9.69
	Maxout (Goodfellow et al., 2013)	11.68
	WNN (Desjardins et al., 2015)	9.87
	pre-GWNN	9.34
	post-GWNN	**8.97**
with augmentation	NIN (Lin et al., 2014)	8.81
	NIN+ALP (Agostinelli et al., 2015)	7.51
	Normalization Propagation (Arpit et al., 2016)	7.47
	BatchNorm (Ioffe & Szegedy, 2015)	**7.25**
	DSN (Lee et al., 2015)	7.97
	Maxout (Goodfellow et al., 2013)	9.38
	WNN (Desjardins et al., 2015)	8.02
	pre-GWNN	7.38
	post-GWNN	**7.10**
	CIFAR-100	Error(%)
without augmentation	NIN (Lin et al., 2014)	35.68
	NIN+ALP (Agostinelli et al., 2015)	34.40
	Normalization Propagation (Arpit et al., 2016)	32.19
	BatchNorm (Ioffe & Szegedy, 2015)	35.32
	DSN (Lee et al., 2015)	34.57
	Maxout (Goodfellow et al., 2013)	38.57
	WNN (Desjardins et al., 2015)	34.78
	pre-GWNN	**32.08**
	post-GWNN	**31.10**
with augmentation	NIN (Lin et al., 2014)	33.37
	NIN+ALP (Agostinelli et al., 2015)	30.83
	Normalization Propagation (Arpit et al., 2016)	29.24
	BatchNorm (Ioffe & Szegedy, 2015)	30.26
	DSN (Lee et al., 2015)	32.16
	WNN (Desjardins et al., 2015)	30.02
	pre-GWNN	**28.78**
	post-GWNN	**28.10**
	SVHN	Error(%)
	NIN (Lin et al., 2014)	2.35
	NIN+ALP (Agostinelli et al., 2015)	2.04
	Normalization Propagation (Arpit et al., 2016)	1.88
	BatchNorm (Ioffe & Szegedy, 2015)	2.25
	DSN (Lee et al., 2015)	1.92
	Maxout (Goodfellow et al., 2013)	2.47
	WNN (Desjardins et al., 2015)	1.93
	pre-GWNN	**1.82**
	post-GWNN	**1.74**

If the error has no reduction over this period, d' is increased by the same rate as above. We use post-GWNN and follow experimental setting in Sec.4.2. We take two different rates $\{0.1, 0.2\}$ as examples. Fig.6 plots the variations of dimensions when $d = 192$ and shows their test errors. We find that keeping d' as a constant generally produces better result than those obtained by the above strategy, but this strategy yields less runtime because more dimensions are pruned.

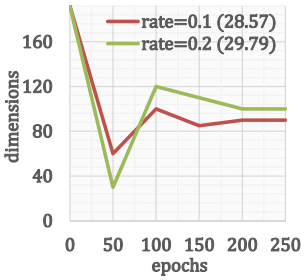

Figure 6. Effect of tuning d'.

5. Conclusion

We presented generalized WNN (GWNN) to reduce runtime and improve generalization of WNN. Different from WNN that reduces computation time by whitening with a large period, leading to ill conditioning of FIM, GWNN learns compact internal representation, such that SVD is approximated by the top eigenvectors in an online manner, making GWNN not only reduces computations but also improves generalization. By exploiting the knowledge of the hidden representation's distribution, we showed that post-GWNN is able to compute the covariance matrix in a closed form, which can be also extended to the other activation function. Extensive experiments demonstrated the effectiveness of GWNN.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China (61503366, 61472410, U1613211), the National Key Research and Development Program of China (No.2016YFC1400700), the External Cooperation Program of BIC, Chinese Academy of Sciences (No.172644KYSB20160033), and the Science and Technology Planning Project of Guangdong Province (2015B010129013, 2014B050505017).

References

Agostinelli, Forest, Hoffman, Matthew, Sadowski, Peter, and Baldi, Pierre. Learning activation functions to improve deep neural networks. In *ICLR*, 2015.

Amari, Shun-ichi and Nagaoka, Hiroshi. Methods of information geometry. In *Tanslations of Mathematical Monographs*, 2000.

Arpit, Devansh, Zhou, Yingbo, Kota, Bhargava U., and Govindaraju, Venu. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *ICML*, 2016.

Desjardins, Guillaume, Simonyan, Karen, Pascanu, Razvan, and Kavukcuoglu, Koray. Natural neural networks. In *NIPS*, 2015.

Goodfellow, Ian J., Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *ICML*, 2013.

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

Ioffe, Sergey and Szegedy, Christian. Batch normalization:

Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. In *Technical Report*, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceeding of IEEE*, 1998.

LeCun, Yann, Bottou, Leon, Orr, Genevieve B., and Mller, Klaus Robert. Efficient backprop. In *Neural Networks: Tricks of the Trade*, 2002.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *AISTATS*, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. In *ICLR*, 2014.

Maas, Andrew L., Hannun, Awni Y., , and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

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

Povey, Daniel, Zhang, Xiaohui, and Khudanpur, Sanjeev. Parallel training of dnns with natural gradient and parameter averaging. In *ICLR workshop*, 2015.

Raiko, Tapani, Valpola, Harri, and LeCun, Yann. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012.

Sermanet, Pierre, Chintala, Soumith, and LeCun, Yann. Convolutional neural networks applied to house numbers digit classification. In *arXiv:1204.3968*, 2012.

Shamir, Ohad. A stochastic pca and svd algorithm with an exponential convergence rate. In *ICML*, 2015.

Tieleman, Tijmen and Hinton, Geoffrey. Rmsprop: Divide the gradient by a running average of its recent magnitude. In *Neural Networks for Machine Learning (Lecture 6.5)*, 2012.

Zhang, Xiangyu, Zou, Jianhua, He, Kaiming, and Sun, Jian. Accelerating very deep convolutional networks for classification and detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.