A. Derivation of Back-Propagation

Given the layer-wise orthogonal weight matrix \( W \), we can perform the forward pass to calculate the loss of the deep neural networks (DNNs). It’s necessary to back-propagate through the orthogonalization transformation, because we aim to update the proxy parameters \( Z \). For illustration, we first describe the proposed orthogonalization by Newton’s iteration (ONI) in Algorithm I. Given the gradient with respect to the orthogonalized weight matrix \( \frac{\partial L}{\partial W} \), we target to compute \( \frac{\partial L}{\partial Z} \). The back-propagation is based on the chain rule. From Line 2 in Algorithm I, we have:

\[
\frac{\partial L}{\partial Z} = \frac{1}{\|Z\|_F} \frac{\partial L}{\partial V} + \text{tr} \left( \frac{\partial L}{\partial Z} \right) \frac{1}{\|Z\|_F} \frac{\partial L}{\partial \|Z\|_F} \frac{\partial \|Z\|_F}{\partial Z} \\
= \frac{1}{\|Z\|_F} \frac{\partial L}{\partial V} - \text{tr} \left( \frac{\partial L}{\partial Z} \right) \frac{1}{\|Z\|_F^2} \left( \|Z\|_F \right) \frac{\partial L}{\partial \|Z\|_F} \\
= \frac{1}{\|Z\|_F} \left( \frac{\partial L}{\partial V} - \text{tr} \left( \frac{\partial L}{\partial Z} \right) \frac{\partial L}{\partial \|Z\|_F} \right),
\]

where \( \text{tr}(\cdot) \) indicates the trace of the corresponding matrix and \( \frac{\partial L}{\partial V} \) can be calculated from Lines 3 and 8 in Algorithm I:

\[
\frac{\partial L}{\partial V} = (B_T)^T \frac{\partial L}{\partial W} + \left( \frac{\partial L}{\partial B_{t-1}} + \frac{\partial L}{\partial S} \right) V.
\]  

We thus need to calculate \( \frac{\partial L}{\partial S} \), which can be computed from Lines 5, 6 and 7 in Algorithm I:

\[
\frac{\partial L}{\partial S} = -\frac{1}{2} \sum_{t=1}^{T} (B_{t-1}^2 S + (B_{t-1}^2) \frac{\partial L}{\partial B_{t}} S^T + B_{t-1} \frac{\partial L}{\partial B_{t}} (B_{t-1} S)^T + \frac{3}{2} \frac{\partial L}{\partial B_{t}}^T).
\]  

where \( \frac{\partial L}{\partial B_{t}} = \frac{\partial L}{\partial W} V^T \) and \( \{ \frac{\partial L}{\partial B_{t-1}}, t = T, ..., 1 \} \) can be iteratively calculated from Line 6 in Algorithm I as follows:

\[
\frac{\partial L}{\partial B_{t-1}} = -\frac{1}{2} \left( \frac{\partial L}{\partial B_{t}} (B_{t-1}^2 S + (B_{t-1}^2) \frac{\partial L}{\partial B_{t}} S^T + B_{t-1} \frac{\partial L}{\partial B_{t}} (B_{t-1} S)^T + \frac{3}{2} \frac{\partial L}{\partial B_{t}}^T) \right) V.
\]  

In summary, the back-propagation of Algorithm I is shown in Algorithm II.

We further derive the back-propagation of the accelerated ONI method with the centering and more compact spectral bounding operation, as described in Section 3.3 of the paper. For illustration, Algorithm III describes the forward pass of the accelerated ONI. Following the calculation in Algorithm II, we can obtain \( \frac{\partial L}{\partial W} \). To simplify the derivation, we represent Line 3 of Algorithm III as the following...
Algorithm III: ONI with acceleration.

1: Input: proxy parameters $Z \in \mathbb{R}^{n \times d}$ and iteration numbers $N$.
2: Centering: $Z_c = Z - \frac{1}{d}Z11^T$.
3: Bounding $Z$’s singular values: $V = \frac{Z}{\sqrt{\|Z\|_F^2}}$.
4: Execute Step 3 to 8 in Algorithm I.
5: Output: orthogonalized weight matrix: $W \in \mathbb{R}^{n \times d}$.

where

\begin{align}
M &= Z_cZ_c^T \quad (5) \\
\delta &= \sqrt{\|M\|_F} \quad (6) \\
V &= \frac{Z_c}{\delta}. \quad (7)
\end{align}

It’s easy to calculate $\frac{\partial C}{\partial Z_c}$ from Eqn.5 and Eqn.7 as follows:

\begin{align}
\frac{\partial C}{\partial Z_c} &= \frac{1}{\delta} \frac{\partial C}{\partial V} + \left( \frac{\partial C}{\partial M} + \frac{\partial C}{\partial M} \right)Z_c, \quad (8)
\end{align}

where $\frac{\partial C}{\partial M}$ can be computed based on Eqn. 6 and Eqn. 7:

\begin{align}
\frac{\partial C}{\partial M} &= \frac{\partial C}{\partial \delta} \frac{\partial \|M\|_F}{\partial \delta} + \frac{\partial C}{\partial M} \\
&= tr\left( \frac{\partial C}{\partial \delta} Z_c \right) - \frac{1}{\delta^2} \frac{1}{2\sqrt{\|M\|_F^2}} \|M\|_F \\
&= \frac{tr\left( \frac{\partial C}{\partial \delta} Z_c \right)}{2\delta^5} M. \quad (9)
\end{align}

Based on Line 2 in Algorithm III, we can achieve $\frac{\partial C}{\partial Z_c}$ as follows:

\begin{align}
\frac{\partial C}{\partial Z_c} &= \frac{\partial C}{\partial Z} - \frac{1}{d} \frac{\partial \|V\|_F^2}{\partial Z} \frac{\partial C}{\partial Z_c} \\
&= -\frac{1}{d} \frac{\partial \|V\|_F^2}{\partial Z} \frac{\partial C}{\partial Z_c}. \quad (10)
\end{align}

In summary, Algorithm IV describes the back-propagation of the Algorithm III.

B. Proof of Convergence Condition for Newton’s Iteration

In Section 3 of the paper, we show that bounding the spectral of the proxy parameters matrix by

\begin{align}
V &= \phi_N(Z) = \frac{Z}{\|Z\|_F} \quad (11)
\end{align}

and

\begin{align}
V &= \phi_N(Z) = \frac{Z}{\sqrt{\|ZZ^T\|_F}} \quad (12)
\end{align}

can satisfy the convergence condition of Newton’s Iterations as follows:

\begin{align}
\|I - S\|_2 < 1, \quad (13)
\end{align}

where $S = VV^T$ and the singular values of $Z$ are nonzero. Here we will prove this conclusion, and we also prove that $\|Z\|_F > \sqrt{\|ZZ^T\|_F}$.

Proof. By definition, $\|Z\|_F$ can be represented as $\|Z\|_F = \sqrt{tr(ZZ^T)}$. Given Eqn.11, we calculate

\begin{align}
S = VV^T = \frac{ZZ^T}{tr(ZZ^T)}. \quad (14)
\end{align}

Let’s denote $M = ZZ^T$ and the eigenvalues of $M$ are $\{\lambda_1, ..., \lambda_n\}$. We have $\lambda_i > 0$, since $M$ is a real symmetric matrix and the singular values of $Z$ are nonzero. We also have $S = \frac{M}{tr(M)}$ and the eigenvalues of $S$ are $\frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$. Furthermore, the eigenvalues of $(I - S)$ are $1 - \frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$, thus satisfying the convergence condition described by Eqn.13.

Similarly, given $V = \phi_N(Z) = \frac{Z}{\|Z\|_F}$, we have $S = \frac{ZZ^T}{\|ZZ^T\|_F} = \frac{M}{\|M\|_F}$ and its corresponding eigenvalues are $\frac{\lambda_i}{\sqrt{\sum_{i=1}^{n} \lambda_i^2}}$. Therefore, the singular values of $(I - S)$ are $1 - \frac{\lambda_i}{\sqrt{\sum_{i=1}^{n} \lambda_i^2}}$, also satisfying the convergence condition described by Eqn.13.

We have $\|Z\|_F = \sqrt{tr(M)} = \sqrt{\sum_{i=1}^{n} \lambda_i}$ and $\sqrt{\|ZZ^T\|_F} = \sqrt{\|M\|_F} = \sqrt{\sum_{i=1}^{n} \lambda_i^2}$. It’s easy to demonstrate that $\|Z\|_F > \sqrt{\|ZZ^T\|_F}$, since $(\sum_{i=1}^{n} \lambda_i)^2 > \sum_{i=1}^{n} \lambda_i^2$.

In Section 3 of the paper, we show that the Newton’s iteration by bounding the spectrum with Eqn. 11 is equivalent to the Newton’s iteration proposed in [11]. Here, we provide the details. In [11], they bound the covariance matrix $M = ZZ^T$ by the trace of $M$ as $\frac{M}{tr(M)}$. It’s clear that $S$ used in Algorithm I is equal to $\frac{M}{\|M\|_F}$, based on Eqn. 14 shown in the previous proof.

C. Orthogonality for Group Based Method

In Section 3.4 of the paper, we argue that group based methods cannot ensure the whole matrix $W \in \mathbb{R}^{n \times d}$ to be either row or column orthogonal, when $n > d$. Here we provide more details.
we sample the entries of proxy matrix \( W \) from the Gaussian distribution \( N(0, 1) \). We evaluate the row orthogonality \( \delta_{\text{Row}} = \| W^T W - I \|_F \) and column orthogonality \( \delta_{\text{Column}} = \| W^T W - I \|_F \). ‘OLM-G32’ indicates the eigen decomposition based orthogonalization method described in [10], with a group size of 32.

We follow the experiments described in Figure 3 of the paper, where we sample the entries of proxy matrix \( Z \in \mathbb{R}^{64 \times 32} \) from the Gaussian distribution \( N(0, 1) \). We apply the eigen decomposition based orthogonalization method [10] with group size \( G \), to obtain the orthogonalized matrix \( W \). We vary the group size \( G \in \{32, 16, 8\} \). We evaluate the corresponding row orthogonality \( \delta_{\text{Row}} = \| W^T W - I \|_F \) and column orthogonality \( \delta_{\text{Column}} = \| W^T W - I \|_F \). The results are shown in Table B. We observe that the group based orthogonalization method cannot ensure the whole matrix \( W \) to be either row or column orthogonal, while our ONI can ensure column orthogonality. We also observe that the group based method has degenerated orthogonality, with decreasing group size.

We also conduct an experiment when \( n = d \), where we sample the entries of proxy matrix \( Z \in \mathbb{R}^{64 \times 64} \) from the Gaussian distribution \( N(0, 1) \). We vary the group size \( G \in \{64, 32, 16, 8\} \). Note that \( G = 64 \) represents full orthogonalization. Figure I shows the distribution of the eigenvalues of \( WW^T \). We again observe that the group based method cannot ensure the whole weight matrix to be row orthogonal. Furthermore, orthogonalization with smaller group size tends to be worse.

### D. Comparison of Wall Clock Times

In Section 3.6 of the paper, we show that, given a convolutional layer with filters \( W \in \mathbb{R}^{m \times d \times F_h \times F_w} \) and \( m \) mini-batch data \( \{ x_i \in \mathbb{R}^{d \times h \times w} \}_{i=1}^m \), the relative computational cost of ONI over the convolutional layer is \( \frac{2n}{m} + \frac{3Nn^2}{mn} \). In this section, we compare the of wall clock time between the convolution wrapping with our ONI and the standard convolution. In this experiment, our ONI is implemented based on Torch [4] and we wrap it to the ‘cudnn’ convolution [3]. The experiments are run on a TITAN Xp.

We fix the input to size \( h = w = 32 \), and vary the kernel size \((F_h, F_w)\), the feature dimensions \((n, d)\) and the batch size \(m\). Table A shows the wall clock time under different configurations. We compare the standard ‘cudnn’ convolution (denoted as ‘cudnn’) and the ‘cudnn’ wrapped with our ONI (denoted as ‘cudnn + ONI’).

We observe that our method introduces negligible computational costs when using a \( 3 \times 3 \) convolution, feature dimension \( n = d = 256 \) and batch size \( m = 256 \). Our method may degenerate in efficiency with a smaller kernel size, larger feature dimension and smaller batch size, based on the computational complexity analysis. However, our method (with iteration of 5) ‘cudnn + ONI-T5’ only costs 1.48\times over the standard convolution ‘cudnn’, under the worst configuration, \( F_h = F_w = 1, n = d = 1024 \) and \( m = 32 \).

<table>
<thead>
<tr>
<th>configurations</th>
<th>cudnn</th>
<th>cudnn + ONI-T1</th>
<th>cudnn + ONI-T3</th>
<th>cudnn + ONI-T5</th>
<th>cudnn + ONI-T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_h = F_w = 3, n=d=256, m=256 )</td>
<td>118.6</td>
<td>122.1</td>
<td>122.9</td>
<td>124.4</td>
<td>125.7</td>
</tr>
<tr>
<td>( F_h = F_w = 3, n=d=256, m=32 )</td>
<td>15.8</td>
<td>18.3</td>
<td>18.9</td>
<td>19.5</td>
<td>20.8</td>
</tr>
<tr>
<td>( F_h = F_w = 3, n=d=1024, m=32 )</td>
<td>71.1</td>
<td>81.7</td>
<td>84.3</td>
<td>89.5</td>
<td>94.2</td>
</tr>
<tr>
<td>( F_h = F_w = 1, n=d=256, m=256 )</td>
<td>28.7</td>
<td>31.5</td>
<td>32.1</td>
<td>33.7</td>
<td>34.6</td>
</tr>
<tr>
<td>( F_h = F_w = 1, n=d=256, m=32 )</td>
<td>10.1</td>
<td>13</td>
<td>13.6</td>
<td>14.2</td>
<td>15.3</td>
</tr>
<tr>
<td>( F_h = F_w = 1, n=d=1024, m=32 )</td>
<td>22.2</td>
<td>27.6</td>
<td>29.7</td>
<td>32.9</td>
<td>37.0</td>
</tr>
</tbody>
</table>

Table A. Comparison of wall-clock time (\( ms \)). We fix the input with size \( h = w = 32 \). We evaluate the total wall-clock time of training for each iteration (forward pass + back-propagation pass). Note that ‘cudnn + ONI-T5’ indicates the ‘cudnn’ convolution wrapped in our ONI method, using an iteration number of 5.

<table>
<thead>
<tr>
<th>( \delta_{\text{Row}} )</th>
<th>( \delta_{\text{Column}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONI-Full</td>
<td>5.66</td>
</tr>
<tr>
<td>OLM-G32</td>
<td>8</td>
</tr>
<tr>
<td>OLM-G16</td>
<td>9.85</td>
</tr>
<tr>
<td>OLM-G8</td>
<td>10.58</td>
</tr>
</tbody>
</table>

Table B. Evaluation for row and column orthogonalization with the group based methods. The entries of proxy matrix \( Z \in \mathbb{R}^{32 \times 64} \) are sampled from the Gaussian distribution \( N(0, 1) \). The entries of proxy matrix \( W \) are sampled from the Gaussian distribution \( N(0, 1) \). We apply the eigen decomposition based orthogonalization method [10] with group size \( G \), to obtain the orthogonalized matrix \( W \). We vary the group size \( G \in \{32, 16, 8\} \). We evaluate the corresponding row orthogonality \( \delta_{\text{Row}} = \| W^T W - I \|_F \) and column orthogonality \( \delta_{\text{Column}} = \| W^T W - I \|_F \). ‘OLM-G32’ indicates the eigen decomposition based orthogonalization method described in [10], with a group size of 32.

![Figure I. The distribution of the eigenvalues of \( WW^T \) with different group size \( G \). The entries of proxy matrix \( Z \in \mathbb{R}^{64 \times 64} \) are sampled from the Gaussian distribution \( N(0, 1) \).](attachment:image.png)
E. Proof of Theorems

Here we prove the two theorems described in Section 3.5 and 3.6 of the paper.

**Theorem 1.** Let \( \hat{h} = Wx \), where \( WW^T = I \) and \( W \in \mathbb{R}^{n \times d} \). Assume: (1) \( E_x(x) = 0 \), \( \text{cov}(x) = \sigma_i^2 I \), and (2) \( E_{\frac{\partial}{\partial h}}(\hat{h}) = 0 \), \( \text{cov}(\hat{h}) = \sigma_i^2 I \). If \( n = d \), we have the following properties: (1) \( \|\hat{h}\| = \|x\| \); (2) \( E_{\hat{h}}(\hat{h}) = 0 \), \( \text{cov}(\hat{h}) = \sigma_i^2 I \); (3) \( \|\frac{\partial \hat{h}}{\partial x}\| = \|\frac{\partial h}{\partial h}\| \); (4) \( E_{\frac{\partial}{\partial h}}(\frac{\partial \hat{h}}{\partial x}) = 0 \), \( \text{cov}(\frac{\partial \hat{h}}{\partial x}) = \sigma_i^2 I \). In particular, if \( n < d \), property (2) and (3) hold; if \( n > d \), property (1) and (4) hold.

**Proof.** Based on \( n = d \) and \( WW^T = I \), we have that \( W \) is a square orthogonal matrix. We thus have \( W^TW = I \). Besides, we have \( \frac{\partial E}{\partial x} = \frac{\partial E}{\partial h} W \).

(1) Therefore, we have

\[
\|\hat{h}\|^2 = \hat{h}^T \hat{h} = x^T W^T W x = x^T x = \|x\|^2. \tag{15}
\]

We thus get \( \|\hat{h}\| = \|x\| \).

(2) It’s easy to calculate:

\[
E_{\hat{h}}(\hat{h}) = E_x(Wx) = W E_x(x) = 0. \tag{16}
\]

The covariance of \( \hat{h} \) is given by

\[
\text{cov}(\hat{h}) = E_{\hat{h}}((\hat{h} - E_{\hat{h}}(\hat{h}))(\hat{h} - E_{\hat{h}}(\hat{h})))^T
\]

\[
= E_x(W(x - E_x(x))(x - E_x(x)))^T
\]

\[
= W E_x(x(x - E_x(x))(x - E_x(x)))^T W^T
\]

\[
= W \text{cov}(x) W^T
\]

\[
= W \sigma_i^2 I W^T
\]

\[
= \sigma_i^2 WW^T
\]

\[
= \sigma_i^2. \tag{17}
\]

(3) Similar to the proof of (1),

\[
\|\frac{\partial \hat{h}}{\partial x}\|^2 = \frac{\partial \hat{h}}{\partial x} \frac{\partial \hat{h}}{\partial x}^T = \frac{\partial \hat{h}}{\partial h} W W^T \frac{\partial \hat{h}}{\partial h}^T
\]

\[
= \frac{\partial \hat{h}}{\partial h} \frac{\partial \hat{h}}{\partial h}^T = \|\frac{\partial \hat{h}}{\partial h}\|^2. \tag{18}
\]

We thus have \( \|\frac{\partial \hat{h}}{\partial x}\| = \|\frac{\partial \hat{h}}{\partial h}\| \).

(4) Similar to the proof of (2), we have

\[
E_{\frac{\partial}{\partial h}}(\frac{\partial \hat{h}}{\partial x}) = E_{\frac{\partial}{\partial x}}(\frac{\partial \hat{h}}{\partial h} W) = E_{\frac{\partial}{\partial x}}(\frac{\partial \hat{h}}{\partial h}) W = 0. \tag{19}
\]

The covariance of \( \frac{\partial E}{\partial x} \) is given by

\[
\text{cov}(\frac{\partial E}{\partial x}) = E_{\frac{\partial}{\partial x}}(\frac{\partial E}{\partial x} - E_{\frac{\partial}{\partial x}}(\frac{\partial E}{\partial x}))^2
\]

\[
= E_{\frac{\partial}{\partial x}}((\frac{\partial E}{\partial x} - E_{\frac{\partial}{\partial x}})(\frac{\partial E}{\partial x} - E_{\frac{\partial}{\partial x}}))^2
\]

\[
= W^T E_{\frac{\partial}{\partial x}}(\frac{\partial E}{\partial x} - E_{\frac{\partial}{\partial x}})(\frac{\partial E}{\partial x} - E_{\frac{\partial}{\partial x}})^T W
\]

\[
= W^T \sigma_i^2 I W
\]

\[
= \sigma_i^2 \sigma_i^2 I W
\]

\[
= \sigma_i^2 W^T W
\]

\[
= \sigma_i^2. \tag{20}
\]

Besides, if \( n < d \), it is easy to show that properties (2) and (3) hold; if \( n > d \), properties (1) and (4) hold.

\( \square \)

**Theorem 2.** Let \( h = \text{max}(0, Wx) \), where \( WW^T = \sigma^2 I \) and \( W \in \mathbb{R}^{n \times d} \). Assume \( x \) is a normal distribution with \( E_x(x) = 0 \), \( \text{cov}(x) = I \). Denote the Jacobian matrix as \( J = \frac{\partial h}{\partial x} \). If \( \sigma^2 = 2 \), we have \( E_x(J J^T) = I \).

**Proof.** For denotation, we use \( A_i \) and \( A_j \) to represent the \( i \)-th row and the \( j \)-th column of \( A \), respectively. Based on \( WW^T = \sigma^2 I \), we obtain \( W_i(W_j)^T = 0 \) for \( i \neq j \) and \( W_i(W_i)^T = \sigma^2 \) otherwise. Let \( h = Wx \). This yields

\[
J_i = \frac{\partial h_i}{\partial x} \frac{\partial h_i}{\partial x}^T \tag{21}
\]

Denote \( M = J J^T \). This yields the following equation from Eqn. 21:

\[
M_{ij} = J_i(J_j)^T = \frac{\partial h_i}{\partial x} \frac{\partial h_j}{\partial x}^T \tag{22}
\]

If \( i \neq j \), we obtain \( M_{ij} = 0 \). For \( i = j \), we have:

\[
M_{ii} = J_i(J_i)^T = (\frac{\partial h_i}{\partial x})^2 \sigma^2 = 1(h_i > 0) \sigma^2, \tag{23}
\]

where \( 1(h_i > 0) \) indicates \( 1 \) for \( h_i > 0 \) and \( 0 \) otherwise. Since \( x \) is a normal distribution with \( E_x(x) = 0 \) and \( \text{cov}(x) = I \), we have that \( h \) is also a normal distribution, with \( E_{\hat{h}}(\hat{h}) = 0 \) and \( \text{cov}(\hat{h}) = I \), based on Theorem 1. We thus obtain

\[
E_x(M_{ii}) = E_{\hat{h}}(1(h_i > 0)) \sigma^2 = \frac{1}{2} \sigma^2. \tag{24}
\]

Therefore, \( E_x(J J^T) = E_x(M) = \sigma^2 I \). \( \square \)
We observe that our ONI can achieve slightly better performance with an increasing group size. The main reason for this is that the activation and gradient during training. Our 'ONI' with a group size of 64, 256, and full orthogonalization can be exponentially vanished. Here we provide the details.

We evaluate the mean absolute value of the activation: \( \sigma_x = \sum_{i=1}^{m} \sum_{j=1}^{d} |x_{ij}| \) for the layer-wise input \( x \in \mathbb{R}^{m \times d} \), and the mean absolute value of the gradient: \( \sigma \frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \sum_{i=1}^{m} \sum_{j=1}^{n} |\frac{\partial \mathcal{L}}{\partial h_{ij}}| \) for the layer-wise gradient \( \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \in \mathbb{R}^{m \times n} \).

Figure II. The magnitude of the activation and gradient for each layer on a 20-layer MLP. (a) The mean absolute value of the activation \( \sigma_x \) for each layer; and (b) the mean absolute value of the gradient \( \sigma \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \) for each layer.

F. Details and More Experimental Results on Discriminative Classification

F.1. MLPs on Fashion-MNIST

Details of Experimental Setup  Fashion-MNIST consists of 60,000 training and 10,000 test images. Each image has a size of 28 \( \times \) 28, and is associated with a label from one of 10 classes. We use the MLP with varying depths and the number of neurons in each layer is 256. We use ReLU \([17]\) as the nonlinearity. The weights in each layer are initialized by random initialization \([13]\) and we use an iteration number of 5 for ONI, unless otherwise stated. We employ stochastic gradient descent (SGD) optimization with a batch size of 5 for ONI, unless otherwise stated. We employ stochastic gradient descent (SGD) optimization with a batch size of 5 for ONI, unless otherwise stated.

We implement a convolutional layer with \( k \) filters, where \( k \) is the varying width based on different configurations. We then sequentially stack three blocks, each of which has \( g \) convolutional layers, and the corresponding convolutional layers have a filter numbers of \( 32k \), \( 64k \), and \( 128k \), respectively, and feature maps sizes of \( 32 \times 32 \), \( 16 \times 16 \), and \( 8 \times 8 \), respectively. We use the first convolution in each block with stride 2 to carry out spatial sub-sampling for feature maps. The network ends with global average pooling and a linear transformation. We vary the depth with \( g \) in \{2, 3, 4\} and the width with \( k \) in \{1, 2, 3\}.

Experimental Setup  We use SGD with a momentum of 0.9 and batch size of 128. The best initial learning rate is chosen from \[\{0.01, 0.02, 0.05\}\] over the validation set of 5,000 samples from the training set, and we divide the learning rate by 5 at 80 and 120 epochs, ending the training at 160 epochs. For ‘OrthReg’, we report the best results using a regularization coefficient \( \lambda \) in \{0.0001, 0.0005\}. For ‘OLM’, we use the group size of \( G = 64 \) and full orthogonalization, and report the best result.

Training Performance  In Section 4.1.2 of the paper, we mention ‘ONI’ and ‘OLM-\( \sqrt{2} \)’ converge faster than other baselines, in terms of training epochs. Figure IV shows the training curves under different configurations (depth and width). It’s clear that ‘ONI’ and ‘OLM-\( \sqrt{2} \)’ converge faster than other baselines under all network configurations, in terms of training epochs. The results support our conclusion that maintaining orthogonality can benefit optimization.
Figure IV. Comparison of training errors on VGG-style networks for CIFAR-10 image classification. From (a) to (i), we vary the depth $3g + 2$ and width $32k$, with $g \in \{2, 3, 4\}$ and $k \in \{1, 2, 3\}$.

F.2.2 Residual Network without Batch Normalization

Here we provide the details of the experimental setups and training performance of the experiments on a 110-layer residual network \[8\] without batch normalization (BN) \[12\], described in Section 4.1.2 of the paper.

**Experimental Setups** We run the experiments on one GPU. We apply SGD with a batch size of 128, a momentum of 0.9 and a weight decay of 0.0001. We set the initial learning rate to 0.1 by default, and divide it by 10 at 80 and 120 epochs, and terminate the training at 160 epochs. For Xavier Init \[5, 1\], we search the initial learning rate from \{0.1, 0.01, 0.001\} and report the best result. For group normalization (GN) \[23\], we search the group size from \{64, 32, 16\} and report the best result. For our ONI, we use the data-dependent initialization methods used in \[21\] to initial the learnable scale parameters.

For small batch size experiments, we train the network with an initial learning rate following the linear learning rate scaling rule \[7\], to adapt the batch size.

**Training Performance** Figure V (a) and (b) show the training curve and test curve respectively. We observe that ‘ONI’ converges significantly faster than ‘BN’ and ‘GN’, in terms of training epochs.

Figure V. Training performance comparison on 110-layer residual network without batch normalization for CIFAR-10 dataset. ‘w/BN’ indicates with BN. We show the (a) training error with respect to the epochs and (b) test error with respect to epochs.

F.3. Details of Experimental Setup on ImageNet

ImageNet-2012 consists of 1.28M images from 1,000 classes \[19\]. We use the official 1.28M labeled images provided for training and evaluate the top-1 and top-5 test classification errors on the validation set, with 50k images.
We keep almost all the experimental settings the same as the publicly available PyTorch implementation [18]: we apply SGD with a momentum of 0.9, and a weight decay of 0.0001; We train over 100 epochs in total and set the initial learning rate to 0.1, lowering it by a factor of 10 at epochs 30, 60 and 90. For ‘WN’ and ‘ONI’, we don’t us weight decay on the learnable scalar parameters.

**VGG Network** We run the experiments on one GPU, with a batch size of 128. Apart from our ‘ONI’, all other methods (‘plain’, ‘WN’, ‘OrthInit’ and ‘OrthReg’) suffer from difficulty in training with a large learning rate of 0.1. We thus run the experiments with initial learning rates of \( \{0.01, 0.05\} \) for these, and report the best result.

**Residual Network** We run the experiments on one GPU for the 18- and 50-layer residual network, and two GPUs for the 101-layer residual network. We use a batch size of 256. Considering that ‘ONI’ can improve the optimization efficiency, as shown in the ablation study on ResNet-18, we run the 50- and 101-layer residual network with a weight decay of \( \{0.0001, 0.0002\} \) and report the best result from these two configurations for each method, for a more fair comparison.

**E.4. Ablation Study on Iteration Number**

We provide the details of the training performance for ONI on Fashion MNIST in Figure VI. We vary \( T \), for a range of 1 to 7, and show the training (Figure VI (a)) and test (Figure VI (b)) errors with respect to the training epochs. We also provide the distribution of the singular values of the orthogonalized weight matrix \( W \), using our ONI with different iteration numbers \( T \). Figure VI (c) shows the results from the first layer, at the 200th iteration. We also obtain similar observations for other layers.

**G. Details on Training GANs**

For completeness, we provide descriptions of the main concepts used in the paper, as follows.

**Inception score (IS)** Inception score (IS) (the higher the better) was introduced by Salimans et al. [20]:

\[
I_D = \exp(E_D[KL[p(y|x)||p(y)]]),
\]

where \( KL[\cdot||\cdot] \) denotes the Kullback-Leibler Divergence, \( p(y) \) is approximated by \( \frac{1}{N} \sum_{i=1}^{N} p(y|x_i) \) and \( p(y|x_i) \) is the trained Inception model [22]. Salimans et al. [20] reported that this score is highly correlated with subjective human judgment of image quality. Following [20] and [16], we calculate the score for 5000 randomly generated examples from each trained generator to evaluate IS. We repeat 10 times and report the average and the standard deviation of IS.

**Fréchet Inception Distance (FID)** [9] Fréchet inception distance (FID) [9] (the lower the better) is another measure for the quality of the generated examples that uses second-order information from the final layer of the inception model applied to the examples. The Fréchet distance itself is a 2-Wasserstein distance between two distributions \( p_1 \) and \( p_2 \), assuming they are both multivariate Gaussian distributions:

\[
F(p_1, p_2) = \|\mu_{p_1} - \mu_{p_2}\|_2^2 + tr(C_{p_1} + C_{p_2} - 2C_{p_1}C_{p_2}^{\frac{1}{2}}),
\]

where \( \{\mu_{p_1}, C_{p_1}\} \), \( \{\mu_{p_2}, C_{p_2}\} \) are the mean and covariance of samples from generated \( p_1 \) and \( p_2 \), respectively, and \( tr(\cdot) \) indicates the trace operation. We calculate the FID between the 10K test examples (true distribution) and the 5K randomly generated samples (generated distribution).

**GAN with Non-saturating Loss** The standard non-saturating function for the adversarial loss is:

\[
L(G, D) = \mathbb{E}_{x \sim q(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[1 - \log D(G(z))],
\]

where \( q(x) \) is the distribution of the real data, \( z \in \mathbb{R}^{d_z} \) is a latent variable, \( p(z) \) is the standard normal distribution \( N(0, I) \), and \( G \) is a deterministic generator function. \( d_z \) is set to 128 for all experiments. Based on the suggestion in [6, 16], we use the alternate cost \( -\mathbb{E}_{z \sim p(z)}[\log D(G(z))] \) to update \( G \), while using the original cost defined in Eqn. 27 for updating \( D \).

**GAN with Hinge Loss** The hinge loss for adversarial learning is:

\[
L_D(\hat{G}, D) = \mathbb{E}_{x \sim q(x)}[\max(0, 1 - D(x))] + \mathbb{E}_{z \sim p(z)}[\max(0, 1 + D(G(z)))]
\]

\[
L_G(G, \hat{D}) = -\mathbb{E}_{z \sim p(z)}[\hat{D}(G(z))]
\]
for the discriminator and the generator, respectively. This type of loss has already been used in [14, 16, 24, 2].

Our code is implemented in PyTorch [18] and the trained Inception model is from the official models in PyTorch [18]. The IS and FID for the real training data are 10.20 ± 0.13 and 3.07 respectively. Note that we do not use the learnable scalar in any the GAN experiment, and set σ = 1 in ONI, for more consistent comparisons with SN.

G.1. Experiments on DCGAN

The DCGAN architecture follows the configuration in [16], and we provide the details in Table D for completeness. The spectral normalization (SN) and our ONI are only applied on the discriminator, following the experimental setup in the SN paper [16].

Figure VII (a) shows the IS of SN and ONI when varying Newton’s iteration number T from 0 to 5. We obtain the same observation as the FID evaluation, shown in Section 4.2 of the paper.

As discussed in Section 4.2 the paper, we conduct experiments to validate the stability of our proposed ONI under different experimental configurations, following [16]. Table C shows the corresponding configurations (denoted by A-F) when varying the learning rate α, first momentum β1, second momentum β2, and the number of updates of the discriminator per update of the generator n_dis. The results evaluated by IS are shown in Figure VII (b). We observe that our ONI is consistently better than SN under the IS evaluation.

G.2. Implementation Details of ResNet-GAN

The ResNet architecture also follows the configuration in [16], and we provide the details in Table E for completeness.

Table C. Hyper-parameter settings in stability experiments on DCGAN, following [16].

<table>
<thead>
<tr>
<th>Setting</th>
<th>α</th>
<th>β1</th>
<th>β2</th>
<th>n_dis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0001</td>
<td>0.5</td>
<td>0.9</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>0.0001</td>
<td>0.5</td>
<td>0.999</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0.0002</td>
<td>0.5</td>
<td>0.999</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0.001</td>
<td>0.5</td>
<td>0.9</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>0.001</td>
<td>0.5</td>
<td>0.999</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0.001</td>
<td>0.9</td>
<td>0.999</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure VIII. Comparison of SN and ONI on ResNet GAN. We show the IS with respect to training epochs using (a) the non-saturating loss and (b) the hinge loss.

The SN and our ONI are only applied on the discriminator, following the experimental setup in the SN paper [16]. We provide the results of SN and ONI in Figure VIII, evaluated by IS.

G.3. Qualitative Results of GAN

We provide the generated images in Figure IX, X and XI. Note that we don’t hand-pick the images, and show all the results at the end of the training.

References

### (a) Generator

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Output Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=1 deconv. BN 512 ReLU</td>
<td>( \rightarrow 4 \times 4 \times 512 )</td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 deconv. BN 256 ReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 deconv. BN 128 ReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 deconv. BN 64 ReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \times 3, stride=1 conv. 3 Tanh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### (b) Discriminator

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Output Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB image ( x \in \mathbb{R}^{32 \times 32 \times 3} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \times 3, stride=1 conv 64 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 conv 64 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \times 3, stride=1 conv 128 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 conv 128 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \times 3, stride=1 conv 256 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \times 4, stride=2 conv 256 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \times 3, stride=1 conv 512 lReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dense ( \rightarrow 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Table D. DCGAN architectures for CIFAR10 dataset in our experiments. ‘lReLu’ indicates the leaky ReLU [15] and its slope is set to 0.1.

---

### (a) Generator

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Output Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dense, 4 \times 4 \times 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock up 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock up 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock up 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BN, ReLU, 3 \times 3 conv, 3 Tanh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### (b) Discriminator

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Output Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB image ( x \in \mathbb{R}^{32 \times 32 \times 3} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock down 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock down 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock down 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResBlock 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global sum pooling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dense ( \rightarrow 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Table E. ResNet architectures for CIFAR10 dataset in our experiments. We use the same ResBlock as the SN paper [16].

---


Figure IX. Generated images for CIFAR-10 by our ONI with different iterations, using DCGAN [16].

(a) $T = 0$
(b) $T = 1$
(c) $T = 2$
(d) $T = 3$
(e) $T = 4$
(f) $T = 5$

Figure X. Generated images for CIFAR-10 by SN and ONI, using DCGAN [16]. We show the results of SN and ONI, with configuration A, B and C.

(a) SN-A
(b) SN-B
(c) SN-C
(d) ONI-A
(e) ONI-B
(f) ONI-C
Figure XI. Generated images for CIFAR-10 by SN and ONI, using ResNet [16]. We show the results of SN and ONI, with the non-saturating and hinge loss.