# Supplementary Materials for an Investigation into the Stochasticity of Batch Whitening

Lei Huang† Lei Zhao Yi Zhou† Fan Zhu† Li Liu† Ling Shao†
†Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE
{lei.huang, yi.zhou, fan.zhu, li.liu, ling.shao}@inceptioniai.org bhneo@126.com

**Algorithm I** A general view of batch whitening algorithms.

1: **Input**: mini-batch inputs $\mathbf{X} \in \mathbb{R}^{d\times m}$.
2: **Output**: $\mathbf{Y} \in \mathbb{R}^{d\times m}$.
3: **if** Training **then**
4: Calculate covariance matrix: $\Sigma = \frac{1}{m}\mathbf{X}\mathbf{X}^T + \epsilon\mathbf{I}$.
5: Calculate whitening matrix: $\mathbf{G} = \phi_1(\Sigma)$.
6: Calculate whitened output: $\widehat{\mathbf{X}} = \mathbf{G}\mathbf{X}$.
7: Update population statistics: $\widehat{\mathbf{G}} = \phi_2(\Sigma/\mathbf{G})$.
8: **else**
9: Calculate whitened output: $\widehat{\mathbf{X}} = \widehat{\mathbf{G}}\mathbf{X}$.
10: **end if**
11: Recover representation: $\mathbf{Y} = \phi_3(\widehat{\mathbf{X}})$.

## A. Back-propagation

Given the Batch Whitening (BW) algorithms described in Algorithm I, we provide the corresponding back-propagation during training in Algorithm II. Note that it is easy to calculate $\frac{\partial\mathcal{L}}{\partial\widehat{\mathbf{X}}}$ in Line. 3 of Algorithm II, given the specific recovery operation shown in the paper. Here, we mainly elaborate on the nontrivial solution of back-propagation through the whitening transformation, *i.e.*, how to calculate $\frac{\partial\mathcal{L}}{\partial\Sigma}$ given $\frac{\partial\mathcal{L}}{\partial\mathbf{G}}$, shown in Line. 5 of Algorithm II.

We consider the following whitening transformations discussed in the paper: Principal Component Analysis (PCA) whitening [4], Zero-phase Component Analysis (ZCA) whitening [4], Cholesky Decomposition (CD) whitening [10] and ItN [5] that approximates ZCA whitening. The back-propagations of all whitening transformations are derived in their original papers [4, 10, 5]. Here, we provide the results for completeness.

### A.1. PCA Whitening

PCA Whitening uses $\mathbf{G}_{PCA} = \Lambda^{-\frac{1}{2}}\mathbf{D}^T$, where $\Lambda = \text{diag}(\sigma_1, \dots, \sigma_d)$ and $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of $\Sigma$, *i.e.* $\Sigma = \mathbf{D}\Lambda\mathbf{D}^T$.

**Algorithm II** Back-propagation of batch whitening algorithms.

1: **Input**: mini-batch gradients $\frac{\partial\mathcal{L}}{\partial\mathbf{Y}} \in \mathbb{R}^{d\times m}$.
2: **Output**: $\frac{\partial\mathcal{L}}{\partial\mathbf{X}} \in \mathbb{R}^{d\times m}$.
3: Calculate: $\frac{\partial\mathcal{L}}{\partial\widehat{\mathbf{X}}} = \frac{\partial\mathcal{L}}{\partial\mathbf{Y}}\frac{\partial\phi_3(\widehat{\mathbf{X}})}{\partial\widehat{\mathbf{X}}}$.
4: Calculate: $\frac{\partial\mathcal{L}}{\partial\mathbf{G}} = \frac{\partial\mathcal{L}}{\partial\widehat{\mathbf{X}}}\mathbf{X}^T$.
5: Calculate: $\frac{\partial\mathcal{L}}{\partial\Sigma} = \frac{\partial\mathcal{L}}{\partial\mathbf{G}}\frac{\partial\phi_1(\Sigma)}{\partial\Sigma}$.
6: Calculate: $\frac{\partial\mathcal{L}}{\partial\mathbf{X}} = \mathbf{G}^T\frac{\partial\mathcal{L}}{\partial\widehat{\mathbf{X}}} + \frac{1}{m}(\frac{\partial\mathcal{L}}{\partial\Sigma} + \frac{\partial\mathcal{L}}{\partial\Sigma}^T)\mathbf{X}$.

The backward pass of PCA whitening is:

$$\frac{\partial\mathcal{L}}{\partial\Lambda} = (\frac{\partial\mathcal{L}}{\partial\mathbf{G}_{PCA}})\mathbf{D}(-\frac{1}{2}\Lambda^{-3/2}) \tag{1}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{D}} = (\frac{\partial\mathcal{L}}{\partial\mathbf{G}_{PCA}})^T\Lambda^{-1/2} \tag{2}$$

$$\frac{\partial\mathcal{L}}{\partial\Sigma} = \mathbf{D}\{(\mathbf{K}^T \odot (\mathbf{D}^T\frac{\partial\mathcal{L}}{\partial\mathbf{D}})) + (\frac{\partial\mathcal{L}}{\partial\Lambda})_{diag}\}\mathbf{D}^T, \tag{3}$$

where $\mathbf{K} \in \mathbb{R}^{d\times d}$ is 0-diagonal with $\mathbf{K}_{ij} = \frac{1}{\sigma_i - \sigma_j}[i \neq j]$, the $\odot$ operator is an element-wise matrix multiplication, and $(\frac{\partial\mathcal{L}}{\partial\Lambda})_{diag}$ sets the off-diagonal elements of $\frac{\partial\mathcal{L}}{\partial\Lambda}$ to zero.

### A.2. ZCA Whitening

ZCA whitening uses $\mathbf{G}_{ZCA} = \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T$, where the PCA whitened input is rotated back by the corresponding rotation matrix $\mathbf{D}$.

The backward pass of ZCA whitening is:

$$\frac{\partial\mathcal{L}}{\partial\Lambda} = \mathbf{D}^T(\frac{\partial\mathcal{L}}{\partial\mathbf{G}_{ZCA}})\mathbf{D}(-\frac{1}{2}\Lambda^{-3/2}) \tag{4}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{D}} = (\frac{\partial\mathcal{L}}{\partial\mathbf{G}_{ZCA}} + (\frac{\partial\mathcal{L}}{\partial\mathbf{G}_{ZCA}})^T)\mathbf{D}\Lambda^{-1/2} \tag{5}$$

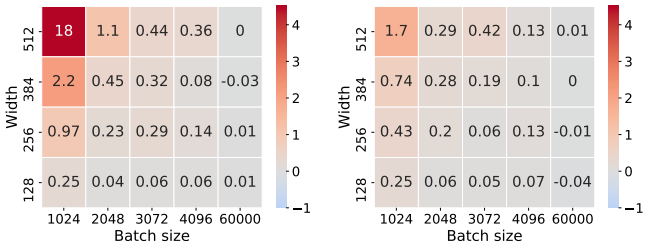
$$\frac{\partial\mathcal{L}}{\partial\Sigma} = \mathbf{D}\{(\mathbf{K}^T \odot (\mathbf{D}^T\frac{\partial\mathcal{L}}{\partial\mathbf{D}})) + (\frac{\partial\mathcal{L}}{\partial\Lambda})_{diag}\}\mathbf{D}^T. \tag{6}$$

### A.3. CD Whitening

CD whitening uses $\mathbf{G}_{CD} = \mathbf{L}^{-1}$, where $\mathbf{L}$ is a lower triangular matrix from the Cholesky decomposition, with $\mathbf{L}\mathbf{L}^T = \Sigma$. The backward pass of CD whitening is:

**Algorithm III** Whitening activations with Newton's iteration.

**Input**: $\Sigma$.
**Output**: $\mathbf{G}_{ItN}$.
$\Sigma_N = \frac{\Sigma}{tr(\Sigma)}$ .
$\mathbf{P}_0 = \mathbf{I}$.
**for** $k = 1$ *to* $T$ **do**
  $\mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}_{k-1}^3\Sigma_N)$
**end for**
$\mathbf{G}_{ItN} = \mathbf{P}_T/\sqrt{tr(\Sigma)}$




(a) ZCA Whitening (b) CD Whitening

Figure I. Comparison of estimating the population statistics of whiten matrix $\widehat{\mathbf{G}}$ using different estimation objects ($\mathbf{G}/\Sigma$). We train MLPs with varying widths (the number of neurons in each layer) and batch sizes, for MNIST classification. We evaluate the difference in test accuracy between $\widehat{\mathbf{G}}_\Sigma$ and $\widehat{\mathbf{G}}_\mathbf{G}$: $AC(\widehat{\mathbf{G}}_\Sigma) - AC(\widehat{\mathbf{G}}_\mathbf{G})$. (a) and (b) show the results using ZCA and CD whitening respectively, under a learning rate of 0.5.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{L}} = -\mathbf{G}_{CD}^T \frac{\partial \mathcal{L}}{\partial \mathbf{G}_{CD}} \mathbf{G}_{CD}^T \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma} = \frac{1}{2}\mathbf{L}^{-T}(P \odot \mathbf{L}^T \frac{\partial \mathcal{L}}{\partial \mathbf{L}} + (P \odot \mathbf{L}^T \frac{\partial \mathcal{L}}{\partial \mathbf{L}})^T)\mathbf{L}^{-1}. \tag{8}$$

### A.4. ItN Whitening

ItN, which approximates the ZCA whitening using Newton's iteration, is shown in Algorithm III, where $T$ is the iteration number and $tr(\cdot)$ denotes the trace of a matrix.

Given $\frac{\partial \mathcal{L}}{\partial \mathbf{G}_{ItN}}$, the back-propagation is:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{P}_T} &= \frac{1}{\sqrt{tr(\Sigma)}} \frac{\partial \mathcal{L}}{\partial \mathbf{G}_{ItN}} \\
\frac{\partial \mathcal{L}}{\partial \Sigma_N} &= -\frac{1}{2}\sum_{k=1}^{T} (\mathbf{P}_{k-1}^3)^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k} \\
\frac{\partial \mathcal{L}}{\partial \Sigma} &= \frac{1}{tr(\Sigma)} \frac{\partial \mathcal{L}}{\partial \Sigma_N} - \frac{1}{(tr(\Sigma))^2} tr(\frac{\partial \mathcal{L}}{\partial \Sigma_N}^T \Sigma)\mathbf{I} \\
&\quad - \frac{1}{2(tr(\Sigma))^{3/2}} tr((\frac{\partial \mathcal{L}}{\partial \mathbf{G}_{ItN}})^T \mathbf{P}_T)\mathbf{I}.
\end{aligned} \tag{9}$$

Here, $\frac{\partial \mathcal{L}}{\partial \mathbf{P}_k}$ can be calculated by the following iterations:

| Method | Step decay | Cosine decay |
|---|---|---|
| Baseline | 76.20 | 76.62 |
| ItN-ARC$_B$ [5] | 77.07 (↑0.87) | 77.47 (↑0.85) |
| ItN-ARC$_C$ [5] | 76.97 (↑0.77) | 77.43 (↑0.81) |
| ItN$_\Sigma$-ARC$_B$ | 77.18 (↑0.98) | 77.68 (↑1.06) |
| ItN$_\Sigma$-ARC$_C$ | **77.28** (↑1.08) | **77.92** (↑1.30) |

Table I. Results using ResNet-50 for ImageNet. We evaluate the top-1 validation accuracy (%, single model and single-crop).

| PARAMETER | VALUE |
|---|---|
| Learning Rate $\alpha$ | $\{0.0001, 0.0002, 0.001\}$ |
| $(\beta_1, \beta_2, n_{dis})$ | $\{(0, 0.9, 1), (0.5, 0.9, 5), (0.5, 0.999, 1), (0.5, 0.999, 5), (0.9, 0.999, 5)\}$ |

Table II. The scope of hyperparameter ranges used in the stability experiments. The Cartesian product of the values suffices to uncover most of the recent results from the literature [7].

| | Scale & Shift | | Coloring | |
|---|---|---|---|---|
| Method | IS | FID | IS | FID |
| BN | $7.141 \pm 0.066$ | 30.26 | $7.264 \pm 0.122$ | 28.35 |
| CD | $7.135 \pm 0.066$ | 30.40 | $7.362 \pm 0.069$ | **25.97** |
| ZCA-64 | $7.128 \pm 0.067$ | 29.24 | $\mathbf{7.445 \pm 0.094}$ | 26.78 |
| ItN | $\mathbf{7.369 \pm 0.104}$ | **29.02** | $7.396 \pm 0.054$ | 28.24 |

Table III. Results on stability experiments with non-saturating loss. We report the best FID and the corresponding IS from all the configurations for different whitening methods.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{P}_{k-1}} &= \frac{3}{2}\frac{\partial L}{\partial \mathbf{P}_k} - \frac{1}{2}\frac{\partial \mathcal{L}}{\partial \mathbf{P}_k}(\mathbf{P}_{k-1}^2\Sigma_N)^T - \frac{1}{2}(\mathbf{P}_{k-1}^2)^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k}\Sigma_N^T \\
&\quad - \frac{1}{2}(\mathbf{P}_{k-1})^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k}(\mathbf{P}_{k-1}\Sigma_N)^T, \quad k = T, ..., 1.
\end{aligned} \tag{10}$$

## B. Stochasticity During Inference

In Section 3.3 of the paper, we mentioned that we try other learning rates in the experiments to compare the estimation methods during inference. Here, we provide the results under a learning rate of 0.5, shown in Figure I.

## C. More Details for Experiments

### C.1. Classification on ImageNet

In Section 4.2.2 of the paper, we mentioned that ItN$_\Sigma$ consistently provides a slight improvement over the original ItN proposed in [5] (using $\mathbf{G}$ as an estimation object). Here, we provide the results, including the original ItN, in Table I. We use an iteration number of 5 for all ItN-related methods. For ItN, we use a group size of 64, as recommended in [5]. For ItN$_\Sigma$, we use a larger group size of 256, since ItN$_\Sigma$ can still provide a good estimation of population statistics in high dimensions during inference, as shown in the paper.

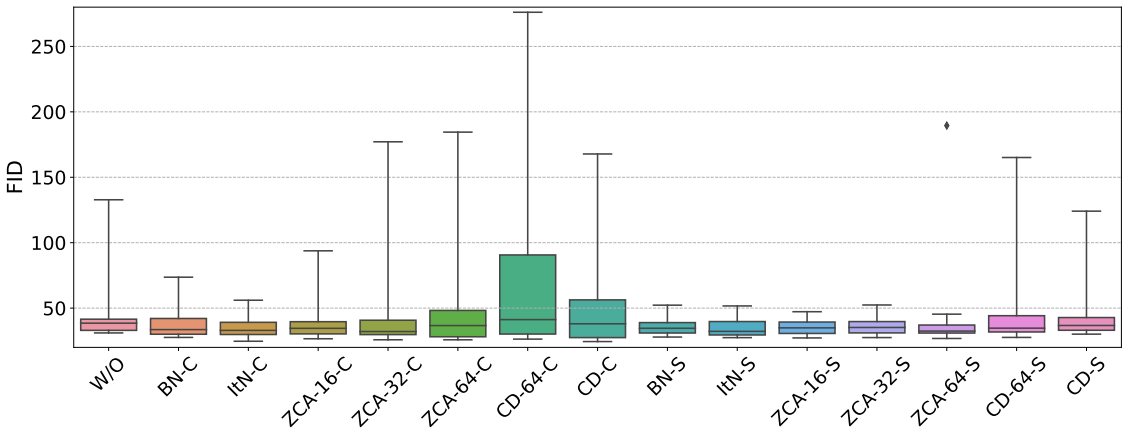



Figure II. Stability experiments on GAN with hinge loss [8, 2] for unconditional image generation. We show the FIDs with full range.

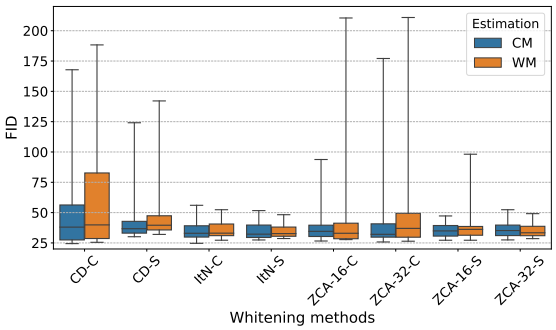



Figure III. Comparison of BW methods when using the Covariance Matrix (CM) and Whitening Matrix (WM) as the estimation object. We show the FIDs with full range.

## C.2. Training GANs

### C.2.1 Stability Experiments

**Experimental Setup** The DCGAN architecture in these experiments is shown in Table VII. Note that spectral normalization [8] is applied on the discriminator, following the setup shown in [10, 8]. We evaluate the FID between the 10k test data and 10K randomly generated examples. We use the Adam optimizer [6] and train for 100 epochs. We consider 15 hyper-parameter settings (Table II) by varying the learning rate $\alpha$, first and second momentum $(\beta_1, \beta_2)$ of Adam, and number of discriminator updates per generator update $n_{dis}$. The learning rate is linearly decreased until it reaches 0.

**Results** In Figures 8 and 9 of the paper, we only show the results with FID ranging in (20, 100) for better representation. Here, we provide the corresponding FIDs with full range in Figure II and III.

We perform experiments on DCGAN with the non-saturating loss [3]. Figure IV shows the results. Here, we obtain similar observations as for the DCGAN with the hinge loss, shown in the paper.

### C.2.2 Validation on Larger Architecture

**Experimental Setup** The architectures in these experiments are shown in Table VIII (DCGAN) and IX (ResNet). The experimental setup is as follows [10]: We evaluate the FID between the 10k test data and 10K randomly generated examples. We use the Adam optimizer [6] with a learning rate $\alpha = 2e^{-4}$. The learning rate is linearly decreased until it reaches 0. We use first momentum $\beta_1 = 0$ and second momentum $\beta_2 = 0.9$. For DCGAN, we use the number of discriminator updates per generator update $n_{dis} = 1$ and train the model for 100 epochs, while for ResNet, we use $n_{dis} = 5$ and train for 50 epochs.

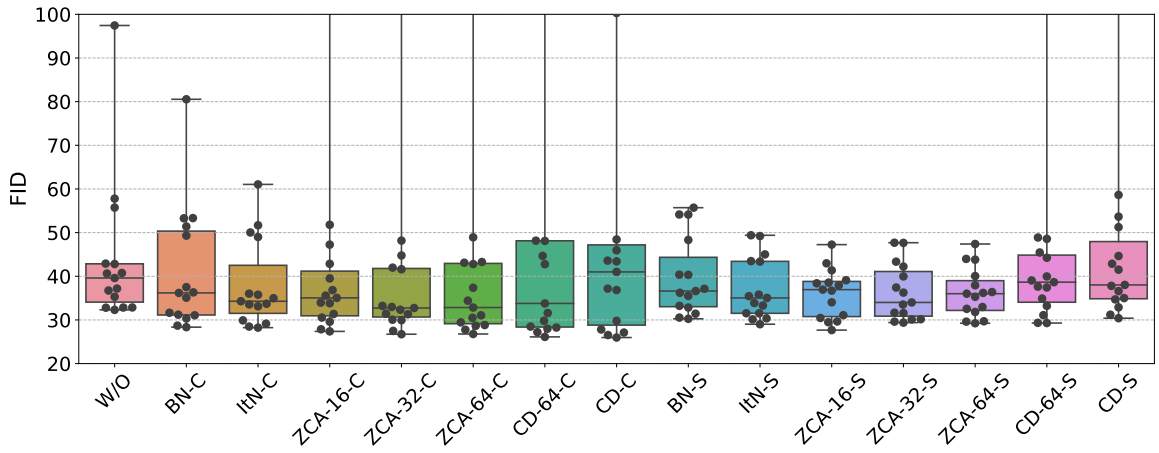
## C.3. Comparison in Running Time

The main disadvantage of whitening methods, compared to standardization, is their expensive computational costs. Here, we provide the time costs of the trained models discussed in the paper.

We note that the computational cost of the Singular Value Decomposition (SVD) used in ZCA/PCA whitening and the Cholesky Decomposition highly depends on the specific deep learning platform (*e.g.*, PyTorch [9] or Tensorflow [1]). We thus conduct experiments both on PyTorch (version number: 1.0.1) and Tensorflow (version number: 1.13), with CUDA (version number: 10.0.130). All implementations of BW are based on the API provided by PyTorch and Tensorflow. We hope that our experiments can provide good guidelines for applying BW in practice.

### C.3.1 Classification Models Using PyTorch

Table IV shows the time costs of the VGG models for CIFAR-10 classification, described in Section 4.2.1 of the paper.

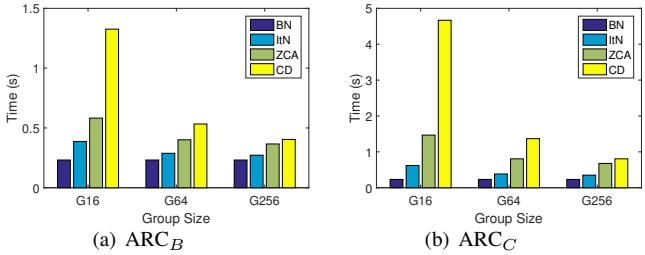



(a) FID ranging in (20,100)

Figure IV. Stability experiments on GAN with non-saturating loss [3] for unconditional image generation.

| Method | Time (Group Size) | | | | | |
|---|---|---|---|---|---|---|
| | 16 | 32 | 64 | 128 | 256 | 512 |
| BN | 0.049 | 0.049 | 0.049 | 0.049 | 0.049 | 0.049 |
| ItN | 0.624 | 0.338 | 0.192 | 0.13 | 0.11 | 0.106 |
| ZCA | 2.245 | 1.536 | 0.906 | 0.498 | 0.505 | 0.512 |
| CD | 9.8 | 4.818 | 2.391 | 1.29 | 0.907 | 0.793 |

Table IV. Time cost (s/iteration) on VGG for CIFAR-10 classification described in Section 4.2.1 of the paper. We compare the BW methods with different group sizes, and the results are averaged over 100 iterations.

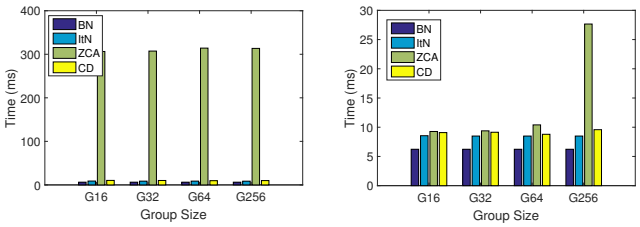



(a) $\text{ARC}_B$ (b) $\text{ARC}_C$

Figure V. Time cost (s/iteration) on ResNet-18 for ImageNet classification described in Section 4.2.2 of the paper. We compare the BW methods with different group sizes under (a) $\text{ARC}_B$ and (b) $\text{ARC}_C$, and the results are averaged over 100 iterations.

Figure V and Table V show the time costs of the ResNet-18 and ResNet-50 described in Section 4.2.2 of the paper, respectively. Note that we run ResNet-18 on one GPU, while ResNet-50 on two GPUs.

Our main observations include: (1) ItN is more computationally efficient. (2) CD whitening is slower than ZCA whitening, which is contrary to the observation in [5] where CD whitening was significantly more efficient than ZCA. The main reason for this is that CD whitening requires the Cholesky decomposition and matrix inverse, which are slower than SVD under the PyTorch platform. We note that under Tensorflow, which was used to implement CD whitening in [10], CD whitening is more efficient than ZCA, as shown in Section C.3.2. (3) Group based whitening with small groups requires more training time than larger groups. The main reason is our unoptimized implementation where we whiten each group sequentially instead of in parallel, because the corresponding platform does not provide an easy way to use linear algebra library of CUDA in parallel, which is also mentioned in [4]. We believe BW would be more widely used if the group based methods could be easily implemented in parallel.




(a) GPU (b) CPU

Figure VI. Time cost (ms/iteration) on GANs for stability experiments described in Section 4.3.1 of the paper. We compare the BW methods with different group sizes running on (a) a GPU and (b) a CPU. We evaluate the time cost of the generator where BW is applied, and the results are averaged over 100 iterations.

### C.3.2 GANs Using Tensorflow

Figure VI shows the time costs of the GAN models described in Section 4.3.1 of the paper. We perform experiments with the BW operation running on both a GPU and a CPU, shown in Figure VI (a) and VI (b), respectively. We observe that ZCA whitening running on GPUs has a significantly more

| Method | Time |
|---|---|
| Baselin (BN) | 0.418 |
| ItN-G256-ARC$_B$ | 0.464 |
| ItN-G256-ARC$_C$ | 0.546 |
| ItN-G64-ARC$_B$ | 0.522 |
| ItN-G64-ARC$_C$ | 0.662 |

Table V. Time cost (s/iteration) on ResNet-50 for ImageNet classification described in Section 4.2.2 of the paper. The results are averaged over 100 iterations.

| Method | DCGAN | | Resnet | |
|---|---|---|---|---|
| | CPU | GPU | CPU | GPU |
| CD | 0.023 | 0.019 | 0.036 | 0.038 |
| ZCA-64 | 0.018 | 0.544 | 0.036 | 0.894 |
| ItN | 0.017 | 0.016 | 0.034 | 0.034 |

Table VI. Time cost (s/iteration) on large GAN architecture described in Section 4.3.2 of the paper. We evaluate the time cost of the generator where BW is applied, and the results are averaged over 100 iterations.

expensive computational cost than other methods, which is consistent with the observation in [10]. We find this can be alleviated when running ZCA whitening on CPUs.

Table VI shows the time cost of the GAN models described in Section 4.3.2 of the paper.

## References


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016. 3

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 3

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*. 2014. 3, 4

[4] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *CVPR*, 2018. 1, 4

[5] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *CVPR*, 2019. 1, 2, 4

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 3

[7] Karol Kurach, Mario Lučić, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. A large-scale study on regularization and normalization in GANs. In *ICML*, 2019. 2

[8] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018. 3

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017. 3

[10] Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening and coloring batch transform for gans. In *ICLR*, 2019. 1, 3, 4, 5, 6

Table VII. DCGAN architectures in stability experiments.

(a) Generator

| $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ |
|---|
| dense, $4 \times 4 \times 256$ |
| $4 \times 4$, stride=2 deconv. BN 256 ReLU |
| $4 \times 4$, stride=2 deconv. BN 128 ReLU |
| $4 \times 4$, stride=2 deconv. BN 64 ReLU |
| $3 \times 3$, stride=1 conv. 3 Tanh |

(b) Discriminator

| RGB image $x \in \mathbb{R}^{32 \times 32 \times 3}$ |
|---|
| $3 \times 3$, stride=1 conv 32 lReLU |
| $3 \times 3$, stride=2 conv 64 lReLU |
| $3 \times 3$, stride=1 conv 64 lReLU |
| $3 \times 3$, stride=2 conv 128 lReLU |
| $3 \times 3$, stride=1 conv 128 lReLU |
| $3 \times 3$, stride=2 conv 256 lReLU |
| $3 \times 3$, stride=1 conv 256 lReLU |
| dense $\rightarrow$ 1 |

Table VIII. Large architecture of DCGAN.

(a) Generator

| $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ |
|---|
| dense, $4 \times 4 \times 512$ |
| $4 \times 4$, stride=2 deconv. BN 512 ReLU |
| $4 \times 4$, stride=2 deconv. BN 256 ReLU |
| $4 \times 4$, stride=2 deconv. BN 128 ReLU |
| $3 \times 3$, stride=1 conv. 3 Tanh |

(b) Discriminator

| RGB image $x \in \mathbb{R}^{32 \times 32 \times 3}$ |
|---|
| $3 \times 3$, stride=1 conv 64 lReLU |
| $4 \times 4$, stride=2 conv 128 lReLU |
| $3 \times 3$, stride=1 conv 128 lReLU |
| $4 \times 4$, stride=2 conv 256 lReLU |
| $3 \times 3$, stride=1 conv 256 lReLU |
| $4 \times 4$, stride=2 conv 512 lReLU |
| $3 \times 3$, stride=1 conv 512 lReLU |
| dense $\rightarrow$ 1 |

Table IX. Large architecture of ResNet. The ResBlock follows [10].

(a) Generator

| $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ |
|---|
| dense, $4 \times 4 \times 256$ |
| ResBlock up 256 |
| ResBlock up 256 |
| ResBlock up 256 |
| BN, ReLU, $3 \times 3$ conv, 3 Tanh |

(b) Discriminator

| RGB image $x \in \mathbb{R}^{32 \times 32 \times 3}$ |
|---|
| ResBlock down 128 |
| ResBlock down 128 |
| ResBlock 128 |
| ResBlock 128 |
| ReLU |
| Global sum pooling |
| dense $\rightarrow$ 1 |