Cross-Iteration Batch Normalization

Zhuliang Yao\(^1,2\) *, Yue Cao\(^2\), Shuxin Zheng\(^2\), Gao Huang\(^1\), Stephen Lin\(^2\)

\(^1\) Tsinghua University \{yzl17@mails., gaohuang\}@tsinghua.edu.cn
\(^2\) Microsoft Research Asia \{yuecao, Shuxin.Zheng, stevelin\}@microsoft.com

Abstract

A well-known issue of Batch Normalization is its significantly reduced effectiveness in the case of small mini-batch sizes. When a mini-batch contains few examples, the statistics upon which the normalization is defined cannot be reliably estimated from it during a training iteration. To address this problem, we present Cross-Iteration Batch Normalization (CBN), in which examples from multiple recent iterations are jointly utilized to enhance estimation quality. A challenge of computing statistics over multiple iterations is that the network activations from different iterations are not comparable to each other due to changes in network weights. We thus compensate for the network weight changes via a proposed technique based on Taylor polynomials, so that the statistics can be accurately estimated and batch normalization can be effectively applied. On object detection and image classification with small mini-batch sizes, CBN is found to outperform the original batch normalization and a direct calculation of statistics over previous iterations without the proposed compensation technique. Code is available at https://aka.ms/cbn.

1. Introduction

Batch Normalization (BN) [10] has played a significant role in the success of deep neural networks. It was introduced to address the issue of internal covariate shift, where the distribution of network activations changes during training iterations due to the updates of network parameters. This shift is commonly believed to be disruptive to network training, and BN alleviates this problem through normalization of the network activations by their mean and variance, computed over the examples within the mini-batch at each iteration. With this normalization, network training can be performed at much higher learning rates and with less sensitivity to weight initialization.

In BN, it is assumed that the distribution statistics for the examples within each mini-batch reflect the statistics over the full training set. While this assumption is generally valid for large batch sizes, it breaks down in the small batch size regime [18, 31, 9], where noisy statistics computed from small sets of examples can lead to a dramatic drop in performance. This problem hinders the application of BN to memory-consuming tasks such as object detection [20, 3], semantic segmentation [15, 2] and action recognition [30], where batch sizes are limited due to memory constraints.

Towards improving estimation of statistics in the small batch size regime, alternative normalizers have been proposed. Several of them, including Layer Normalization (LN) [1], Instance Normalization (IN) [28], and Group Normalization (GN) [31], compute the mean and variance over the channel dimension, independent of batch size. Different channel-wise normalization techniques, however, tend to be suitable for different tasks, depending on the set of channels involved. Although GN is designed for detection task, the slow inference speed limits its practical usage. On the other hand, synchronized BN (SyncBN) [18] yields consistent improvements by processing larger batch sizes across multiple GPUs. These gains in performance come at the cost of additional overhead needed for synchronization across the devices.

A seldom explored direction for estimating better statistics is to compute them over the examples from multiple recent training iterations, instead of from only the current iteration as done in previous techniques. This can substantially enlarge the pool of data from which the mean and variance are obtained. However, there exists an obvious drawback to this approach, in that the activation values from different iterations are not comparable to each other due to the changes in network weights. As shown in Figure 1, directly calculating the statistics over multiple iterations, which we refer to as Naive CBN, results in lower accuracy.

In this paper, we present a method that compensates for the network weight changes among iterations, so that examples from preceding iterations can be effectively used to improve batch normalization. Our method, called Cross-Iteration Batch Normalization (CBN), is motivated by the observation that network weights change gradually, instead of abruptly, between consecutive training iterations,
The importance of normalization in training neural networks has been recognized for decades [11]. In general, normalization can be performed on three components: input data, hidden activations, and network parameters. Among them, input data normalization is used most commonly because of its simplicity and effectiveness [26, 11].

After the introduction of Batch Normalization [10], the normalization of activations has become nearly as prevalent. By normalizing hidden activations by their statistics within each mini-batch, BN effectively alleviates the vanishing gradient problem and significantly speeds up the training of deep networks. To mitigate the mini-batch size dependency of BN, a number of variants have been proposed, including Layer Normalization (LN) [1], Instance Normalization (IN) [28], Group Normalization (GN) [31], and Batch Instance Normalization (BIN) [17]. The motivation of LN is to explore more suitable statistics for sequential models, while IN performs normalization in a manner similar to BN but with statistics only for each instance. GN achieves a balance between IN and LN, by dividing features into multiple groups along the channel dimension and computing the mean and variance within each group for normalization. BIN introduces a learnable method for automatically switching between normalizing and maintaining style information, enjoying the advantages of both BN and IN on style transfer tasks. Cross-GPU Batch Normalization (CGBN or SyncBN) [18] extends BN across multiple GPUs for the purpose of increasing the effective batch size. Though providing higher accuracy, it introduces synchronization overhead to the training process. Kalman Normalization (KN) [29] presents a Kalman filtering procedure for estimating the statistics for a network layer from the layer’s observed statistics and the computed statistics of previous layers.

Batch Renormalization (BRN) [9] is the first attempt to utilize the statistics of recent iterations for normalization. It does not compensate for the statistics from recent iterations, but rather it down-weights the importance of statistics from distant iterations. This down-weighting heuristic, however, does not make the resulting statistics “correct”, as the statistics from recent iterations are not of the current network weights. BRN can be deemed as a special version of our Naive CBN baseline (without Taylor polynomial approximation), where distant iterations are down-weighted.

Recent work have also investigated the normalization of network parameters. In Weight Normalization (WN) [22], the optimization of network weights is improved through a reparameterization of weight vectors into their length and direction. Weight Standardization (WS) [19] instead reparameterizes weights based on their first and second moments for the purpose of smoothing the loss landscape of the optimization problem. To combine the advantages of multiple normalization techniques, Switchable Normalization (SN) [16] and Sparse Switchable Normalization (SSN) [24] make use of differentiable learning to switch among different normalization methods.

The proposed CBN takes an activation normalization approach that aims to mitigate the mini-batch dependency of BN. Different from existing techniques, it provides a way to effectively aggregate statistics across multiple training iter-
lations.

3. Method

3.1. Revisiting Batch Normalization

The original batch normalization (BN) \cite{ioffe2015batch} whitens the activations of each layer by the statistics computed within a mini-batch. Denote $\theta_i$ and $x_{t,i}(\theta_i)$ as the network weights and the feature response of a certain layer for the $i$-th example in the $t$-th mini-batch. With these values, BN conducts the following normalization:

$$\hat{x}_{t,i}(\theta_i) = \frac{x_{t,i}(\theta_i) - \mu_t(\theta_i)}{\sqrt{\sigma_t(\theta_i)^2 + \epsilon}},$$  

where $\hat{x}_{t,i}(\theta_i)$ is the whitened activation with zero mean and unit variance, $\epsilon$ is a small constant added for numerical stability, and $\mu_t(\theta_i)$ and $\sigma_t(\theta_i)$ are the mean and variance computed for all the examples from the current mini-batch, i.e.,

$$\mu_t(\theta_i) = \frac{1}{m} \sum_{i=1}^{m} x_{t,i}(\theta_i),$$  

$$\sigma_t(\theta_i) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_{t,i}(\theta_i) - \mu_t(\theta_i))^2},$$  

where $\nu_t(\theta_i) = \frac{1}{m} \sum_{i=1}^{m} x_{t,i}(\theta_i)^2$, and $m$ denotes the number of examples in the current mini-batch. The whitened activation $\hat{x}_{t,i}(\theta_i)$ further undergoes a linear transform with learnable weights, to increase its expressive power:

$$y_{t,i}(\theta_i) = \gamma \hat{x}_{t,i}(\theta_i) + \beta,$$

where $\gamma$ and $\beta$ are the learnable parameters (initialized to $\gamma = 1$ and $\beta = 0$ in this work).

When the batch size $m$ is small, the statistics $\mu_t(\theta_i)$ and $\sigma_t(\theta_i)$ become noisy estimates of the training set statistics, thus degrading the effects of batch normalization. In the ImageNet classification task for which the BN module was originally designed, a batch size of 32 is typical. However, for other tasks requiring larger models and/or higher image resolution, such as object detection, semantic segmentation and video recognition, the typical batch size may be as small as 1 or 2 due to GPU memory limitations. The original BN becomes considerably less effective in such cases.

3.2. Leveraging Statistics from Previous Iterations

To address the issue of BN with small mini-batches, a naive approach is to compute the mean and variance over the current and previous iterations. However, the statistics $\mu_{t-\tau}(\theta_{t-\tau})$ and $\nu_{t-\tau}(\theta_{t-\tau})$ of the $(t-\tau)$-th iteration are computed under the network weights $\theta_{t-\tau}$, making them obsolete for the current iteration. As a consequence, directly aggregating statistics from multiple iterations produces inaccurate estimates of the mean and variance, leading to significantly worse performance.

We observe that the network weights change smoothly between consecutive iterations, due to the nature of gradient-based training. This allows us to approximate $\mu_{t-\tau}(\theta_i)$ and $\nu_{t-\tau}(\theta_i)$ from the readily available $\mu_{t-\tau}(\theta_{t-\tau})$ and $\nu_{t-\tau}(\theta_{t-\tau})$ via a Taylor polynomial, i.e.,

$$\mu_{t-\tau}(\theta_i) = \mu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \mu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}} (\theta_i - \theta_{t-\tau}) + O(||\theta_i - \theta_{t-\tau}||^2),$$

$$\nu_{t-\tau}(\theta_i) = \nu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \nu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}} (\theta_i - \theta_{t-\tau}) + O(||\theta_i - \theta_{t-\tau}||^2),$$

where $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ are gradients of the statistics with respect to the network weights, and $O(||\theta_i - \theta_{t-\tau}||^2)$ denotes higher-order terms of the Taylor polynomial, which can be omitted since the first-order term dominates when $(\theta_i - \theta_{t-\tau})$ is small.

In Eq. (5) and Eq. (6), the gradients $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ cannot be precisely determined at a negligible cost because the statistics $\mu_{t-\tau}(\theta_{t-\tau})$ and $\nu_{t-\tau}(\theta_{t-\tau})$ for a node at the $l$-th network layer depend on all the network weights prior to the $l$-th layer, i.e., $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau} \neq 0$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau} \neq 0$ for $r \leq l$, where $\theta_{t-\tau}$ denotes the network weights at the $r$-th layer. Only when $r = l$ can these gradients be derived in closed form efficiently.

Empirically, we find that as the layer index $r$ decreases ($r \leq l$), the partial gradients $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ rapidly diminish. These reduced effects of network weight changes at earlier layers on the activation distributions in later layers may perhaps be explained by the reduced internal covariate shift of BN. Motivated by this phenomenon, which is studied in Appendix ??, we propose to truncate these partial gradients at layer $l$.

Thus, we further approximate Eq. (5) and Eq. (6) by

$$\mu_{t-\tau}(\theta_i) \approx \mu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \mu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}} (\theta_i - \theta_{t-\tau}),$$

$$\nu_{t-\tau}(\theta_i) \approx \nu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \nu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}} (\theta_i - \theta_{t-\tau}).$$

A naive implementation of $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ involves computational overhead of $O(C_{out} \times C_{out} \times C_{in} \times K)$, where $C_{out}$ and $C_{in}$ denote the output and input channel dimension of the $l$-th layer, respectively, and $K$ denotes the kernel size of $\theta_{t-\tau}$. Here, we find that the operation can be implemented efficiently in $O(C_{out} \times C_{in} \times K)$, thanks to the averaging over feature responses of $\mu$ and $\nu$. See Appendix ?? for the details.
3.3. Cross-Iteration Batch Normalization

After compensating for network weight changes, we aggregate the statistics of the \( k - 1 \) most recent iterations with those of the current iteration \( t \) to obtain the statistics used in CBN:

\[
\hat{\mu}_{t,k}^l(\theta_t) = \frac{1}{k} \sum_{\tau=0}^{k-1} \mu_{t-\tau}^l(\theta_t),
\]

\[
\tilde{\nu}_{t,k}^l(\theta_t) = \frac{1}{k} \sum_{\tau=0}^{k-1} \max[\nu_{t-\tau}^l(\theta_t), \mu_{t-\tau}^l(\theta_t)]^2,
\]

where \( \mu_{t-\tau}^l(\theta_t) \) and \( \nu_{t-\tau}^l(\theta_t) \) are computed from Eq. (7) and Eq. (8). In Eq. (10), \( \tilde{\nu}_{t,k}^l(\theta_t) \) is determined from the maximum of \( \nu_{t-\tau}^l(\theta_t) \) and \( \mu_{t-\tau}^l(\theta_t)^2 \) in each iteration because \( \nu_{t-\tau}^l(\theta_t) \geq \mu_{t-\tau}^l(\theta_t)^2 \) should hold for valid statistics but may be violated by Taylor polynomial approximations in Eq. (7) and Eq. (8). Finally, \( \hat{\mu}_{t,k}^l(\theta_t) \) and \( \tilde{\nu}_{t,k}^l(\theta_t) \) are applied to normalize the corresponding feature responses \( \{x_{t,i}^l(\theta_t)\}_{i=1}^m \) at the current iteration:

\[
x_{t,i}^l(\theta_t) = \frac{x_{t,i}^l(\theta_t) - \hat{\mu}_{t,k}^l(\theta_t)}{\sqrt{\tilde{\nu}_{t,k}^l(\theta_t)^2 + \epsilon}}.
\]

With CBN, the effective number of examples used to compute the statistics for the current iteration is \( k \) times as large as that for the original BN. In training, the loss gradients are backpropagated to the network weights and activations at the current iteration, i.e., \( \theta_t^l \) and \( x_{t,i}^l(\theta_t) \). Those of the previous iterations are fixed and do not receive gradients. Hence, the computation cost of CBN in backpropagation is the same as that of BN.

Replacing the BN modules in a network by CBN leads to only minor increases in computational overhead and memory footprint. For computation, the additional overhead mainly comes from computing the partial derivatives \( \partial \mu_{t-\tau}^l(\theta_{t-\tau})/\partial \theta_{t-\tau}^l \) and \( \partial \nu_{t-\tau}^l(\theta_{t-\tau})/\partial \theta_{t-\tau}^l \), which is insignificant in relation to the overhead of the whole network. For memory, the module requires access to the statistics \( \{\mu_{t-\tau}^l(\theta_{t-\tau})\}_{\tau=1}^{k-1} \) and \( \{\nu_{t-\tau}^l(\theta_{t-\tau})\}_{\tau=1}^{k-1} \) and the gradients \( \{\partial \mu_{t-\tau}^l(\theta_{t-\tau})/\partial \theta_{t-\tau}^l\}_{\tau=1}^{k-1} \) and \( \{\partial \nu_{t-\tau}^l(\theta_{t-\tau})/\partial \theta_{t-\tau}^l\}_{\tau=1}^{k-1} \) computed for the most recent \( k - 1 \) iterations, which is also minor compared to the rest of the memory consumed in processing the input examples. The additional computation and memory of CBN is reported for our experiments in Table 8.

A key hyper-parameter in the proposed CBN is the temporal window size, \( k \), of recent iterations used for statistics estimation. A broader window enlarges the set of examples, but the example quality becomes increasingly lower for more distant iterations, since the differences in network parameters \( \theta_t \) and \( \theta_{t-\tau} \) become more significant and are compensated less well using a low-order Taylor polynomial. Empirically, we found that CBN is effective with a window size up to \( k = 8 \) in a variety of settings and tasks. The only trick is that the window size should be kept small at the beginning of training, when the network weights change quickly. Thus, we introduce a burn-in period of length \( T_{\text{burn-in}} \) for the window size, where \( k = 1 \) and CBN degenerates to the original BN. In our experiments, the burn-in period is set to 25 epochs on ImageNet image classification and 3 epochs on COCO object detection by default.

Table 1 compares CBN with other feature normalization methods. The key difference among these approaches is the axis along which the statistics are counted and the features are normalized. The previous techniques are all designed to
Table 1. Comparison of different feature normalization methods. #bs/GPU denotes batch size per GPU.

<table>
<thead>
<tr>
<th>IN</th>
<th>LN</th>
<th>GN</th>
<th>CBN</th>
<th>BN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 acc</td>
<td>64.4±0.2</td>
<td>67.9±0.2</td>
<td>68.9±0.1</td>
<td>70.2±0.1</td>
</tr>
</tbody>
</table>

Table 2. Top-1 accuracy of feature normalization methods using ResNet-18 on ImageNet.

<table>
<thead>
<tr>
<th>batch size per GPU</th>
<th>BN</th>
<th>GN</th>
<th>BRN</th>
<th>CBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>70.2</td>
<td>68.4</td>
<td>65.1</td>
<td>55.9</td>
</tr>
<tr>
<td>16</td>
<td>70.2</td>
<td>69.0</td>
<td>69.0</td>
<td>69.1</td>
</tr>
<tr>
<td>8</td>
<td>70.0</td>
<td>69.2</td>
<td>67.9</td>
<td>60.3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Top-1 accuracy of normalization methods with different batch sizes using ResNet-18 as the base model on ImageNet.

4. Experiments

4.1. Image Classification on ImageNet

Experimental settings. ImageNet [21] is a benchmark dataset for image classification, containing 1.28M training images and 50K validation images from 1000 classes. We follow the standard setting in [5] to train deep networks on the training set and report the single-crop top-1 accuracy on the validation set. Our preprocessing and augmentation strategy strictly follows the GN baseline [31]. We use a weight decay of 0.0001 for all weight layers, including γ and β. We train standard ResNet-18 for 100 epochs on 4 GPUs, and decrease the learning rate by the cosine decay strategy [7]. We perform the experiments for five trials, and report their mean and standard deviation (error bar). ResNet-18 with BN is our base model. To compare with other normalization methods, we directly replace BN with IN, LN, GN, BRN, and our proposed CBN.

Comparison of feature normalization methods. In Table 2, we compare the performance of each normalization method with a batch size, 32, sufficient for computing reliable statistics. Under this setting, BN clearly yields the highest top-1 accuracy. Similar to results found in previous works 31], the performance of IN and LN is significantly worse than that of BN. GN works well on image classification but falls short of BN by 1.2%. Among all the methods, our CBN is the only one that is able to achieve accuracy comparable to BN, as it converges to the procedure of BN at larger batch sizes.

Sensitivity to batch size. We compare the behavior of CBN, original BN [10], GN [31], and BRN [9] at the same number of images per GPU on ImageNet classification. For CBN, the recent iterations are utilized so as to ensure that the number of effective examples is no fewer than 16. For BRN, the settings strictly follow the original paper. We adopt a learning rate of 0.1 for the batch size of 32, and linearly scale the learning rate by N/32 for a batch size of N.

The results are shown in Table 3. For the original BN, its accuracy drops noticeably as the number of images per GPU is reduced from 32 to 2. BRN suffers a significant performance drop as well. GN maintains its accuracy by utilizing the channel dimension but not batch dimension. For CBN, its accuracy holds by exploiting the examples of recent iterations. Also, CBN outperforms GN by 0.9% on average top-1 accuracy with different batch sizes. This is reasonable, because the statistics computation of CBN introduces uncertainty caused by the stochastic batch sampling like in BN, but this uncertainty is missing in GN which results in some loss of regularization ability. For the extreme case that the number of images per GPU is 1, BN and BRN fails to produce results, while CBN outperforms GN by 0.4% on top-1 accuracy in this case.

Different network architectures. To verify the generalization ability of CBN, we also compared CBN to BN and GN using different network architectures. The results are shown in Table 4. 1. We choose five types of architectures, i.e., ResNet-50 [6], VGG-16 [25], Inception-v3 [27], DenseNet-121 [8], and MobileNet-v2 [23]. This set
Table 5. Results of feature normalization methods on Faster R-CNN with FPN and ResNet50 on COCO. As the values of standard deviation of all methods are less than 0.1 on COCO, we ignore them here.

Table 6. Results with stronger backbones on COCO object detection and instance segmentation.
Table 5 presents the results. When BN is frozen in the backbone and no normalizer is applied at the head, the \( A^{bbox} \) score is 36.9%. When the original BN is applied at the head only and at both the backbone and the head, the accuracy drops to 36.3% and 35.5%, respectively. For CBN, the accuracy is 37.7% and 37.7% at these two settings, respectively. Without any synchronization across GPUs, CBN can achieve performance on par with syncBN and GN, showing the superiority of the proposed approach.

**Instance segmentation and stronger backbones.** Results of object detection (Faster R-CNN [20]) and instance segmentation (Mask R-CNN [4]) with ResNet-50 and ResNet-101 are presented in Table 6. We can observe that our CBN achieves performance comparable to syncBN and GN with Res50 and R101 as the backbone on both Faster R-CNN and Mask R-CNN, which demonstrates that CBN is robust and versatile to various deep models and tasks.

### 4.3. Ablation Study

**Effect of temporal window size \( k \).** We conduct this ablation on ImageNet image classification and COCO object detection, with each GPU processing 4 images. Figure 3 presents the results. When \( k = 1 \), only the batch from the current iteration is utilized; therefore, CBN degenerates to the original BN. The accuracy suffers due to the noisy statistics on small batch sizes. As the window size \( k \) increases, more examples from recent iterations are utilized for statistics estimation, leading to greater accuracy. Accuracy saturates at \( k = 8 \) and even drops slightly. For more distant iterations, the network weights differ more substantially and Taylor polynomial approximation becomes less accurate.

On the other hand, it is empirically observed that the original BN saturates at a batch size of 16 or 32 for numerous applications [18, 31], indicating that the computed statistics become accurate. Thus, a temporal window size of \( k = \min(\left\lceil \frac{bs}{8} \right\rceil, 8) \) is suggested.

**Effect of compensation.** To study this, we compare CBN to 1) naive CBN, where statistics from recent iterations are directly aggregated without compensation via Taylor polynomial; and 2) the original BN applied with the same effective example number as CBN (i.e., its batch size per GPU is set to the product of batch size per GPU and temporal window size of CBN), which does not require any compensation and serves as an upper performance bound.

The experimental results are also presented in Figure 3. CBN clearly surpasses naive CBN when the previous iterations are included. Actually, naive CBN fails when the temporal window size grows to \( k = 8 \) as shown in Figure 3(a), demonstrating the necessity of compensating for changing network weights over iterations. Compared with the original BN upper bound, CBN achieves similar accuracy at the same effective example number. This result indicates that the compensation using a low-order Taylor polynomial by

---

Figure 3. The effect of **temporal window size (k)** on ImageNet (ResNet-18) and COCO (Faster R-CNN with ResNet-50 and FPN) with \#bs/GPU = 4 for CBN and Naive CBN. Naive CBN directly utilizes statistics from recent iterations, while BN uses the equivalent \#examples as CBN for statistics computation.

---

Figure 4. Training and test curves for CBN, Naive CBN, and BN on ImageNet, with batch size per GPU of 4 and temporal window size \( k = 4 \) for CBN, Naive CBN, and BN-bs4, and batch size per GPU of 16 for BN-bs16. The plot of BN-bs16 is the ideal bound.

---

Figure 5. Results of **different burn-in periods (in epochs)** on CBN, with batch size per iteration of 4, on ImageNet and COCO. CBN is effective.

---

Figure 4 presents the train and test curves of CBN, Naive CBN, BN-bs4, and BN-bs16 on ImageNet, with 4 images per GPU and a temporal window size of 4 for CBN, Naive CBN, and BN-bs4, and 16 images per GPU for BN-bs16. The train curve of CBN is close to BN-bs4 at the beginning, and approaches BN-bs16 at the end. The reason is that we adopt a burn-in period to avoid the disadvantage of rapid statistics change at the beginning of training. The gap between the train curve of Naive CBN and CBN shows that Naive CBN cannot even converge well on the training set. The test curve of CBN is close to BN-bs16 at the end, while Naive CBN exhibits considerable jitter. All these phenomena indicate the effectiveness of our proposed Taylor polynomial compensation.

**Effect of burn-in period length \( T \).** We study the effect of varying the burn-in period length \( T_{burn-in} \) at 4 images per
GPU on both ImageNet image classification (ResNet-18) and COCO object detection (Faster R-CNN with FPN and ResNet-50). Figure 5(a) and 5(b) present the results. When the burn-in period is too short, the accuracy suffers. This is because at the beginning of training, the network weights change rapidly, causing the compensation across iterations to be less effective. When the burn-in period is too long, i.e., CBN is involved too late and the overall performance drops to the BN baseline.

An interesting observation is that the accuracy is stable for a wide range of burn-in periods $T_{\text{burn-in}}$. This leads to a question of whether BN in the small batch-size regime only suffers in terms of generalization performance in later stages of training. For further exploration, we design an experiment to remove other influences: we first train the model on COCO with standard BN and a small batch size, then switch BN to syncBN. We present the experimental results in Table 7. Results show that syncBN works similarly to CBN, which further verifies the high performance of CBN. It also supports our assumption that BN in the small batch-size regime only suffers in terms of generalization performance in later stages of training, which may shed some light on the small batch-size regime.

### 4.4. Analysis

**Computational cost, memory footprint, and training/inference speed.** We examine the computational cost, memory footprint, and the training and inference speed of BN, GN and CBN in a practical COCO object detection task using R50-Mask R-CNN, shown in Table 8. The batch size per GPU and window size of CBN are set to 4.

Compared to BN and GN, CBN consumes about 7% extra memory and 11% more computational cost. The extra memory mainly contains the statistics ($\mu$ and $\nu$), their respective gradients, and the network parameters ($\theta_{t-1} \cdots \theta_{t-(k-1)}$) of previous iterations, while the computational cost comes from calculations of the statistics’ respective gradients, Taylor compensations, and averaging operations.

The overall training speed of CBN is close to both BN and GN. It is worth noting that the inference speed of CBN is equal to BN, which is much faster than GN. The inference stage of CBN is the same as that of BN, where pre-recorded statistics can be used instead of online statistics calculation. From these results, the additional overhead of CBN is seen to be minor. Also, merging BN/CBN into convolution in inference [12] could be utilized for further speedup.

**Using second order statistics for compensation.** Results of CBN with different orders of Taylor expansion (batch size = 4, #iterations for approximation = 3) are shown in Table 9. By directly using the statistics of recent iterations without compensation, Naive CBN outperforms BN with batch size 4 by 1.7% in accuracy. Via compensating the statistics of recent iterations with a first-order Taylor expansion, CBN(1) can further improve the accuracy by 3.2% compared to Naive CBN. However, CBN(2) using a second-order approximation does not achieve better performance than CBN(1). This may be because CBN(1) already achieves performance comparable to BN with large batch size, which serves as the upper bound of our approach, indicating that a first-order approximation is enough for image classification on ImageNet. Therefore, first-order compensation for CBN is adopted by default.

**Using more than one layer for compensation.** We also study the influence of applying compensation over more than one layer. CBN using two layers for compensation achieves 70.1 on ImageNet (batch size per GPU=4, $k=4$), which is comparable to CBN using only one layer. However, the efficient implementation can no longer be used when more than one layer of compensation is employed. As using more layers does not further improve performance but consumes more FLOPs, we adopt one-layer compensation for CBN in practice.

### 5. Conclusion

In the small batch size regime, batch normalization is widely known to drop dramatically in performance. To address this issue, we propose to enhance the quality of statistics via utilizing examples from multiple recent iterations. As the network activations from different iterations are not comparable to each other due to changes in network weights, we compensate for the network weight changes based on Taylor polynomials, so that the statistics can be accurately estimated. In the experiments, the proposed approach is found to outperform original batch normalization and a direct calculation of statistics over previous iterations without compensation. Moreover, it achieves performance on par with SyncBN, which can be regarded as the upper bound, on both ImageNet and COCO object detection.
References

