Neural Networks 93 (2017) 219–229

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Accelerating deep neural network training with inconsistent stochastic gradient descent

Linnan Wang [a,*], Yi Yang [b], Renqiang Min [b], Srimat Chakradhar [b]

[a] Brown University, United States
[b] NEC Laboratories, United States

ARTICLE INFO

Article history:
Received 13 July 2016
Received in revised form 4 May 2017
Accepted 4 June 2017
Available online 16 June 2017

Keywords:
Neural networks
Stochastic gradient descent
Statistical process control

ABSTRACT

Stochastic Gradient Descent (SGD) updates Convolutional Neural Network (CNN) with a noisy gradient computed from a random batch, and each batch evenly updates the network once in an epoch. This model applies the same training effort to each batch, but it overlooks the fact that the gradient variance, induced by Sampling Bias and Intrinsic Image Difference, renders different training dynamics on batches. In this paper, we develop a new training strategy for SGD, referred to as Inconsistent Stochastic Gradient Descent (ISGD) to address this problem. The core concept of ISGD is the inconsistent training, which dynamically adjusts the training effort w.r.t the loss. ISGD models the training as a stochastic process that gradually reduces down the mean of batch's loss, and it utilizes a dynamic upper control limit to identify a large loss batch on the fly. ISGD stays on the identified batch to accelerate the training with additional gradient updates, and it also has a constraint to penalize drastic parameter changes. ISGD is straightforward, computationally efficient and without requiring auxiliary memories. A series of empirical evaluations on real world datasets and networks demonstrate the promising performance of inconsistent training.

1. Introduction

The accessible TFLOPs brought forth by accelerator technologies bolster the booming development in Neural Networks. In particular, large scale neural networks have drastically improved various systems in natural language processing (Gimpel, Das, & Smith, 2010), video motion analysis (Donahue et al., 2015), and recommender systems (Yu, Hsieh, Si, & Dhillon, 2014). However, training a large neural network saturated with nonlinearity is notoriously difficult. For example, it takes 10000 CPU cores up to days to complete the training of a network with 1 billion parameters (Dean et al., 2012). Such computational challenges have manifested the importance of improving the efficiency of gradient based training algorithm.

The network training is an optimization problem that searches for optimal parameters to approximate the intended function defined over a finite training set. A notable aspect of training is the vast solution hyperspace defined by abundant network parameters. The recent ImageNet contests have seen the parameter size of Convolutional Neural Networks increase to $n \sim 10^9$. Solving an optimization problem at this scale is prohibitive to the second order optimization methods, as the required Hessian matrix, of size $10^9 \times 10^9$, is too large to be tackled by modern computer architectures. Therefore, the first order gradient descent is widely used in training the large scale neural networks.

The standard first order full Gradient Descent (GD), which dates back to Cauchy (1847), calculates the gradient with the whole dataset. Despite the appealing linear convergence rate of full gradient descent ($\mathcal{O}(\rho^k)$, $\rho < 1$) (Roux, Schmidt, & Bach, 2012), the computation in an iteration linearly increases with the size of dataset. This makes the method unsuitable for neural networks trained with the sheer volume of labeled data. To address this issue, Stochastic Gradient Descent (Le Cun & Bottou, 2004; Robbins & Monro, 1951) was proposed by observing a large amount of redundancy among training examples. It approximates the dataset with a batch of random samples, and uses the stochastic gradient computed from the batch to update the model. Although the convergence rate of SGD, $\mathcal{O}(1/\sqrt{bk}+1/k)$ (Bottou, 1998) where b is the batch size, is slower than GD, SGD updates the model much faster than GD in a period, i.e. larger k. As a result, the faster convergence is observable on SGD compared to GD in practice. SGD hits a sweet spot between the good system utilization (Wang, Wu, Xu, Xiao, & Yang, 2016) and the fast gradient updates. Therefore, it soon becomes a popular and effective method to train large scale neural networks.

The key operation in SGD is to draw a random batch from the dataset. It is simple in math, while none-trivial to be implemented on a large-scale dataset such as ImageNet (Deng et al., 2009). State of the art engineering approximation is the Fixed Cycle Pseudo

* Corresponding author.
E-mail address: wangnan318@gmail.com (L. Wang).

http://dx.doi.org/10.1016/j.neunet.2017.06.003
0893-6080/© 2017 Elsevier Ltd. All rights reserved.

Random (FCPR) sampling (defined in Section 3.4), which retrieves batches from the pre-permuted dataset like a ring, e.g. $\mathbf{d_0} \rightarrow \mathbf{d_1} \rightarrow \mathbf{d_2} \rightarrow \mathbf{d_0} \rightarrow \mathbf{d_1} \rightarrow \cdots$, where $\mathbf{d_i}$ denotes a batch. In this case, each batch receives the same training iterations as a batch updates the network exactly once in an epoch. Please note this engineering simplification allows batches to repetitively flow into the network, which is different from the random sampling in Statistics. However, it is known that the gradient variances differentiate batches in the training (Johnson & Zhang, 2013), and gradient updates from the large loss batch contribute more than the small loss ones (Simo-Serra et al., 2015). This suggests that rebalancing the training effort across batches is necessary. SGD fails to consider the issue, and we think this is a problem.

In this paper, we propose Inconsistent Stochastic Gradient Descent (ISGD) to rebalance the training effort among batches. The inconsistency is reflected by the uneven gradient updates on batches. ISGD measures the training status of a batch by the associated loss. At any iteration t, ISGD traces the losses in iterations $[t - n_b, t]$, where n_b is the number of distinct batches in a dataset. These losses assist in constructing a dynamic upper threshold to identify a under-trained batch during the training. If a batch's loss exceeds the threshold, ISGD accelerates the training on the batch by solving a new subproblem that minimizes the discrepancy between the loss of current batch and the mean. The subproblem also contains a conservative constraint to avoid overshooting by bounding the parameter change. The key idea of the subproblem is to allow additional gradient updates on a under-trained batch while still remaining the proximity to the current network parameters. Empirical experiments demonstrate ISGD, especially at the final stage, performs much better than the baseline method SGD on various mainstream datasets and networks.

For practical considerations, we also delve into the effect of batch size toward the convergence rate with system factors considered. Enlarging the batch size expedites the convergence (Byrd, Chin, Nocedal, & Wu, 2012), but it linearly adds computations in an iteration. In the scenario of single node training, a small batch is favored to ensure frequent gradient updates. In the scenario of the multi-node training, it entails heavy synchronizations among nodes per iteration. The more gradient updates, the higher synchronization cost is. In this case, a moderate large batch reduces overall communications (Li, Zhang, Chen, & Smola, 2014), and it also improves the system saturation and the available parallelism.

In summary, the novelties of this work are:

- We propose a new training model, referred to as the inconsistent training, to improve the efficiency of SGD.
- We apply the inconsistent training on SGD and its variants.

2. Related work

A variety of approaches have been proposed to improve vanilla SGD for the neural network training. In this section, we demonstrate the concept of inconsistent training is fundamentally different from the existing methods.

The stochastic sampling in SGD introduces the gradient variance, which slows down the convergence rate (Bottou, 1998). The problem motivates researchers to apply various variance reduction techniques on SGD to improve the convergence rate. Stochastic Variance Reduced Gradient (SVRG) (Johnson & Zhang, 2013) keeps network historical parameters and gradients to explicitly reduce the variance of update rule, but the authors indicate SVRG only works well for the fine-tuning of non-convex neural network. Wang, Chen, Smola, and Xing (2013) explore the control variates on SGD, while Zhao and Zhang (2014) explore the importance sampling. These variance reduction techniques, however, are rarely used in the large scale neural networks, as they consume the huge RAM space to store the intermediate variables. ISGD adjusts to the negative effect of gradient variances, and it does not construct auxiliary variables being much more memory efficient and practical than the variance reduction ones.

Momentum (Tseng, 1998) is a widely recognized heuristic to boost SGD. SGD oscillates across the narrow ravine as the gradient always points to the other side instead of along the ravine toward the optimal. As a result, it tends to bounce around leading to the slow convergence. *Momentum* damps oscillations in directions of high curvature by combining gradients with opposite signs, and it builds up speed toward a direction that is consistent with the previously accumulated gradients (Sutskever, Martens, Dahl, & Hinton, 2013). The update rule of *Nesterov's accelerated gradient* is similar to *Momentum* (Sutskever, 2013), but the minor different update mechanism for building the velocity results in important behavior differences. Momentum strikes in the direction of the accumulated gradient plus the current gradient. In contrast, *Nesterov's accelerated gradient* strikes along the previous accumulated gradient, then it measures the gradient before making a correction. This prevents the update from descending fast, thereby increases the responsiveness. ISGD is fundamentally different from these approaches by considering the training dynamics on batches. ISGD rebalances the training effort across batches, while *Momentum* and *Nesterov's accelerated gradient* leverage the curvature tricks. Therefore, the inconsistent training is expected to be compatible with both methods.

Adagrad (Duchi, Hazan, & Singer, 2011) adapts the learning rate to the parameters, performing larger updates for infrequent parameters, and smaller updates for frequent parameters. It accumulates the squared gradients in the denominator, which will drastically shrink the learning rate. Subsequently, *RMSprop* and *Adadelta* have been developed to resolve the issue. These adaptive learning rate approaches adjust the extent of parameter updates w.r.t the parameter's update frequency to increase the robustness of training, while ISGD adjusts the frequency of a batch's gradient updates w.r.t the loss to improve the training efficiency. From this perspective, ISGD is different from the adaptive learning rate approaches.

The core concept of inconsistent training is to spare more training effort on the large loss batches than the small loss ones. The rational behind the scene is that gradient updates from the small loss batches contribute less than the large loss ones. Simo-Serra et al. (2015) adopt a similar idea in training the Siamese network to learn the deep descriptors by intentionally feeding the network with hard training pairs, i.e. pairs yield large losses, and the method is proven to be an effective way to improve the performance. They manually pick the hard pairs to feed the network, while ISGD automatically identifies the hard batch during the training. In addition, the mechanism of ISGD's hard batch acceleration is different from the Simo-Serra's method. ISGD solves a sub-optimization problem on the hard batch to reduce the batch's loss and avoids drastic parameter changes, while the Simo-Serra's method simply feeds the batch more often. Please note it is important to bound the parameter changes, because overshooting a batch leads to the divergence on other batches. In summary, ISGD is the first neural network solver to consider the batch-wise training dynamics, and it has demonstrated promising performance on a variety of real world datasets and models.

3. Problem statement

This section demonstrates the non-uniform batch-wise training dynamics. Theoretically, we prove the contribution of gradient updates varies among batches based on the analysis of SGD's convergence rate. We also hypothesize that Intrinsic Image Differences and Sampling Bias are high level factors to the phenomenon, and the hypothesis is verified by two controlled experiments. Both theories and experiments support our conclusion that the contribution of a batch's gradient update is different.

Then we demonstrate the Fixed Cycle Pseudo Random sampling employed by SGD is inefficient to handle this issue. In particular, the consistent gradient updates on all batches, regardless of their statuses, is wasteful especially at the end of training, and the gradient updates on the small loss batch could have been used to accelerate large loss batches.

3.1. A recap of CNN training

We formulate the CNN training as the following optimization problem. Let ψ be a loss function with weight vector $\mathbf{w}$ as function parameters, which takes a batch of images $\mathbf{d}$ as the input. The objective of CNN training is to find a solution to the following optimization problem:

$$\min_{\mathbf{w}} \quad \psi_{\mathbf{w}}(\mathbf{d}) + \frac{1}{2}\lambda\|\mathbf{w}\|_2^2. \tag{1}$$

The second term is Weight Decay (Moody, Hanson, Krogh, & Hertz, 1995), and λ is a parameter to adjust its contribution (normally around 10^{-4}). The purpose of Weight Decay is to penalize the large parameters so that static noise and irrelevant components of weight vectors get suppressed (Moody et al., 1995).

A typical training iteration of CNN consists of a Forward and Backward pass. Forward pass yields a loss that measures the discrepancy between the current predictions and the truth. Backward pass calculates the gradient, the negative of which points to the steepest descent direction. Gradient Descent updates the $\mathbf{w}$ as follows:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta_t \nabla\psi_{\mathbf{w}}(\mathbf{d}). \tag{2}$$

Whereas evaluating the gradient over the entire dataset is extremely expensive especially for large datasets such as ImageNet. To resolve this issue, mini-batched SGD is proposed to approximate the entire dataset with a small randomly drawn sample $\mathbf{d}_t$. The upside of mini-batched SGD is the efficiency of evaluating a small sample in the gradient calculation, while the downside is the stochastic gradient slowing down the convergence. Let us define a sample space Ω. If $\psi_{\mathbf{w}}(\mathbf{d_t})$ is a random variable defined on a probability space (Ω, Σ, P), the new objective function is

$$\min_{\mathbf{w}} \quad E\{\psi_{\mathbf{w}}(\mathbf{d_t})\} = \int_{\Omega} \psi_{\mathbf{w}}(\mathbf{d_t})dP + \frac{1}{2}\lambda\|\mathbf{w}\|_2^2 \tag{3}$$

the update rule changes to

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta_t \nabla\psi_{\mathbf{w}}(\mathbf{d_{t-1}}) \tag{4}$$

and the following holds,

$$E\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} = \nabla\psi_{\mathbf{w}}(\mathbf{d}). \tag{5}$$

3.2. Measure training status with cross entropy error

We use the loss to reflect the training status of a batch. A convolutional neural network is a function of $R^n \rightarrow R$, the last layer of which is a softmax loss function calculating the cross entropy between the true prediction probabilities $p(x)$ and the estimated prediction probabilities $\hat{p}(x)$. The definition of softmax loss function of a batch at iteration t is

$$\psi_{\mathbf{w_t}}(\mathbf{d_t}) = -\sum_{i}^{n_b}\sum_{x} p(x)\log\hat{p}(x) + \frac{1}{2}\lambda\|\mathbf{w}\|_2^2 \tag{6}$$

where n_b is the number of images in a batch, and λ regulates Weight Decay. Since Weight Decay is applied, the loss of a batch fluctuates around a small number after being fully trained.

The loss produced by the cross entropy is a reliable indicator of a batch's training status. Given a batch $\mathbf{d_t}$, the cross entropy $\psi_{\mathbf{w_t}}(\mathbf{d_t})$ measures the discrepancy between the estimated probabilities and the truth. In the image classification task, the truth $p(x)$ is a normalized possibility vector, containing most zeros with only one scalar set to 1. The index of the vector corresponds to an object category. For example, $p(x) = [0, 0, 1, 0, 0]$ indicates the object belongs the category 2 (index starts from 0). The neural network produces an normalized estimate possibility $\hat{p}(x)$, and the loss function only captures the extent of making the correct prediction as the zeros in $p(x)$ offset the incorrect predictions in $\hat{p}(x)$. If $\hat{p}(x)$ is close to $p(x)$, the loss function yields a small value. If $\hat{p}(x)$ is far from $p(x)$, the loss function yields a large value. Therefore, we use the loss of a batch to assess the model's training status on it. Intuitively a large loss indicates that most predictions made by the network on the batch are false, and the additional training on the batch is necessary.

3.3. Motivation: Non-uniform training dynamics of batches

The gradient variance is the source of batch-wise training variations. The benefit of using a random sample to approximate the population is the significantly less computations in an iteration, while the downside is the noisy gradient. Please note the convergence rate in this section is measured by iterations. To analyze the training dynamics per iteration, we need define the Lyapunov process:

$$h_t = \|\mathbf{w}^t - \mathbf{w}^*\|_2^2. \tag{7}$$

The equation measures the distance between the current solution $\mathbf{w^t}$ and the optimal solution $\mathbf{w}^*$. h_t is a random variable. Hence the convergence rate of SGD can be derived using Eqs. (4) and (7):

$$\begin{aligned} h_{t+1} - h_t &= \|\mathbf{w}^{t+1} - \mathbf{w}^*\|_2^2 - \|\mathbf{w}^t - \mathbf{w}^*\|_2^2 \\ &= (\mathbf{w^{t+1}} + \mathbf{w^t} - 2\mathbf{w}^*)(\mathbf{w^{t+1}} - \mathbf{w^t}) \\ &= (2\mathbf{w^t} - 2\mathbf{w}^* - \eta_t\nabla\psi_{\mathbf{w}}(\mathbf{d_t}))(-\eta_t\nabla\psi_{\mathbf{w}}(\mathbf{d_t})) \\ &= -2\eta_t(\mathbf{w}^t - \mathbf{w}^*)\nabla\psi_{\mathbf{w}}(\mathbf{d_t}) + \eta_t^2(\nabla\psi_{\mathbf{w}}(\mathbf{d_t}))^2 \end{aligned} \tag{8}$$

$\mathbf{d}_t$ is a random sample of $\mathbf{d}$ in the sample space Ω, and $h_{t+1} - h_t$ is a random variable that depends on the drawn sample $\mathbf{d_t}$ and learning rate η_t. It suggests how far an iteration step toward $\mathbf{w}^*$. This equation demonstrates two important insights:

- *Reducing* $\mathbf{VAR}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\}$ *improves the convergence rate*. The expectation of Eq. (8) yields the average convergence rate at the precision of an iteration.

$$\begin{aligned} \mathbf{E}\{h_{t+1} - h_t\} &= -2\eta_t(\mathbf{w}^t - \mathbf{w}^*)\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} \\ &\quad + \eta_t^2\mathbf{E}\{(\nabla\psi_{\mathbf{w}}(\mathbf{d_t}))^2\} \\ &= -2\eta_t(\mathbf{w}^t - \mathbf{w}^*)\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} \\ &\quad + \eta_t^2(\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\})^2 + \mathbf{VAR}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} \end{aligned} . \tag{9}$$

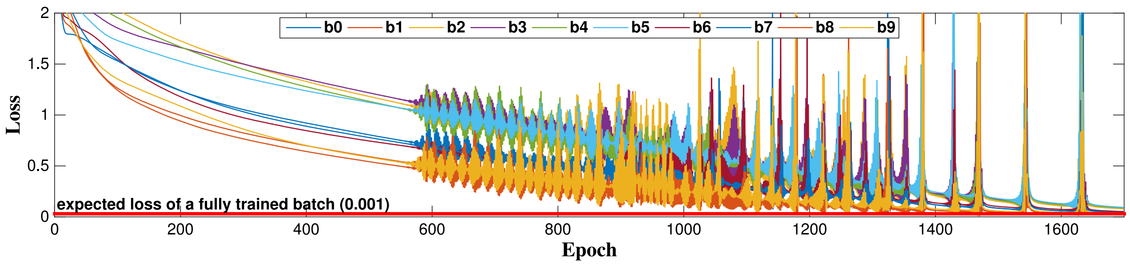
  To simplify the analysis of Eq. (9), let us assume the convexity on $\psi_{\mathbf{w}}(\mathbf{d_t})$ implying that

$$h_{t+1} - h_t < 0 \tag{10}$$

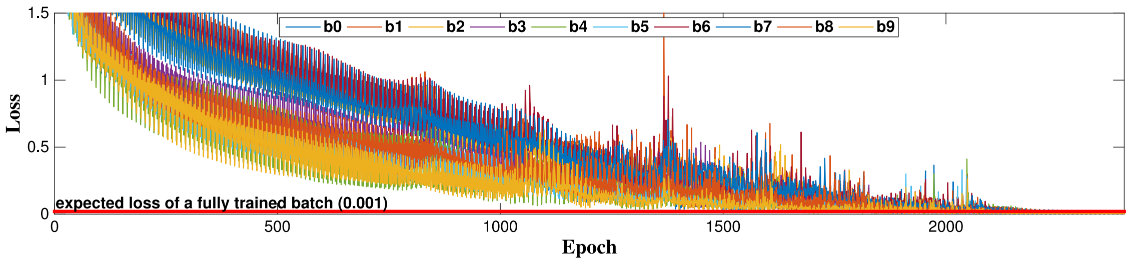
$$-(\mathbf{w}^t - \mathbf{w}^*)\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} < 0. \tag{11}$$

  $\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\}$ is the unbiased estimate of $\mathbf{E}\{\nabla\psi_{\mathbf{w}}(\mathbf{d})\}$. Therefore, maximizing the contribution of an iteration is reduced to the minimization of $\mathbf{VAR}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\}$. This direction has been well addressed (Reddi, Hefny, Sra, Poczos, & Smola, 2015; Reddi, Hefny, Sra, Póczós, & Smola, 2016).
- *The contribution of an iteration,* $h_{t+1} - h_t$, *varies with respect to* $\mathbf{d_t}$. According to Eq. (8), the variance of $h_{t+1} - h_t$ is:

$$\begin{aligned} &\mathbf{VAR}\{h_{t+1} - h_t\} = 4\eta_t^2(\mathbf{w}^t - \mathbf{w}^*)^2\mathbf{VAR}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t})\} \\ &\quad + \eta_t^4\mathbf{VAR}\{(\nabla\psi_{\mathbf{w}}(\mathbf{d_t}))^2\} \\ &\quad - 2\eta_t^3(\mathbf{w}_t - \mathbf{w}_*)\mathbf{COV}\{\nabla\psi_{\mathbf{w}}(\mathbf{d_t}), \nabla\psi_{\mathbf{w}}(\mathbf{d_t})^2\} \end{aligned} . \tag{12}$$



(a) Single-class batches: $\mathbf{b_i}$ randomly draws 1000 images from a category of CIFAR-10.



(b) Independent identically distributed (i.i.d) batches: $\mathbf{b_i}$ randomly draw 100 images from each categories of CIFAR-10 (10 categories in total).

Fig. 1. The loss traces of 10 single class and i.i.d batches in two controlled experiments. We utilize an example network provided in Caffe. There is no cropping on input images and we shuffle the images within a batch. The datasets for each experiment are synthesized from CIFAR-10.

The equation demonstrates $\mathbf{VAR}\{h_{t+1} - h_t\} \neq 0$, which implies the contribution of gradient updates is non-uniform. It is interesting to notice that the determining factors in this equation, $\nabla\psi_{\mathbf{w}}(\mathbf{d_t})^2$ and $\nabla\psi_{\mathbf{w}}(\mathbf{d_t})$, is contingent upon $\mathbf{d_t}$, suggesting a correlation between $h_{t+1} - h_t$ and $\mathbf{d_t}$. This unique insight motivates us to understand what factors in $\mathbf{d_t}$ affect the convergence rate $h_{t+1} - h_t$, and how to address the load balancing problem in the training. Although there are extensive studies toward the variance reduction on $\nabla\psi_{\mathbf{w}}(\mathbf{d_t})$, few explores this direction. Let us use the loss of a batch to measure the model's training status on it (explained in Section 3.2). Fig. 1 demonstrates the loss traces of 10 separate batches during the training. It is observable that the losses of batches degenerate at different rates. Therefore, the empirical observations and Eq. (12) prompt us to conclude that

the contribution of a batch's gradient update is non-uniform .

This also explains the distinctive training dynamics of batches in Fig. 1. Eq. (12) suggests $\mathbf{d}_t$ is critical for the claim. We conduct a set of empirical evaluations to understand how $\mathbf{d}_t$ affect $\mathbf{VAR}\{h_{t+1} - h_t\}$, and we propose two high level factors, Sampling Bias and Intrinsic Image Difference, to explain the phenomenon. The definitions of these two terms are as follows:

Sampling Bias: It is a bias in which a sample is collected in such a way that some members of the intended population are less likely to be included than others.

Intrinsic Image Difference: Intrinsic Image Difference indicates images from the same subpopulation are also different at pixels. For example, the category 'cat' can contain some white cat pictures or black cat pictures. Though black cat and white cat belong to the cat subpopulation, they are still different at pixels.

Sampling Bias is the first factor to explain the training variations on batches. We consider two kinds of Sampling Bias. First, existing datasets, such as Places (Zhou, Lapedriza, Xiao, Torralba, & Oliva, 2014) or ImageNet, contain uneven number of images in each category. As a result, the dominate sub-population is more likely to be selected in a batch than others. Second, the insufficient shuffling on the dataset may lead to clusters of subpopulations. When SGD sequentially draws images from the insufficient permuted dataset to form a randomized batch (explained in Section 3.4), one subpopulation is more likely to be included than others. In both cases, they conform to the definition of Sampling Bias. For example, the chance of sampling 1 from [1, 1, 1, 0, 2, 3] is higher than the rest. To support the claim, we synthesized 10 single-class batches randomly drawn from an exclusive image category in CIFAR-10 (Krizhevsky & Hinton, 2010). Please note CIFAR-10 contains 10 independent image categories. Each batch represents a unique CIFAR-10 category, and they are highly polluted with Sampling Bias as each batch only contains one subpopulation. Fig. 1a demonstrates the loss traces of ten single-class batches. It is obvious to see the losses of ten batches degrade independently. In particular, gradient updates from the yellow batch is more effective than the purple batch. Therefore, these results justify our claim about Sampling Bias and the batch-wise training variation.

Intrinsic Image Difference is the second factor to explain the training variations on batches. To substantiate this point, we conduct a controlled experiment on 10 independent identically distributed batches. A batch includes 1000 images, and each batch contains 100 randomly drawn images from category 0, 100 images from category 1, . . . , 100 images from category 9. This sequence is fixed across batches to eliminate the potential ordering influence. In this case, each batch contains the same number of images from 10 subpopulations in CIFAR-10 and the only difference among them is the pixels. Hence, we consider these batches independent identically distributed. The network is same as the one used in Sampling Bias. Fig. 1b demonstrates the loss traces of 10 i.i.d batches. Although a strong correlation persists through the training, it is still clear the losses of i.i.d batches degrade at separate rates. Particularly, the loss of batch 4 (green) is around 0.5 while batch 3 (purple) is around 1.3 at the epoch 400. Please note these batches are i.i.d, and they are supposed to be approximately identical to the original dataset. However, the training variations still exist indicating the non-uniform contribution of gradient updates from each batches.

3.4. Problems of consistent training in SGD

SGD relies on a key operation, uniformly drawing a batch from the entire dataset. It is simple in math but nontrivial in the system implementation. ImageNet, ILSVRC2012 for example, contains 1431167 256 $\times$ 256 high resolution RGB images accounting for approximately 256 GB in total size. Uniformly drawing a random batch from the 256 GB binary file involves significant overhead such as TLB misses or random Disk I/O operations. In addition, the drastic speed gap between Processor and Disk further deteriorates the issue. Existing deep learning frameworks, such as Caffe (Jia et al., 2014) or Torch (Collobert, Kavukcuoglu, & Farabet, 2011), alleviates the issue by pre-permuting the entire dataset before slicing into batches: $Permute\{\mathbf{d}\} \rightarrow \mathbf{d} = \{\mathbf{d_0}, \mathbf{d_1}, \dots, \mathbf{d_{n-1}}, \mathbf{d_n}\} = \Omega$. During the training, each iteration fetches a batch from the permuted dataset Ω in a sequential manner $\mathbf{d_0} \rightarrow \mathbf{d_1} \rightarrow \cdots \rightarrow \mathbf{d_n}$; and restart fetching from the beginning $\mathbf{d_0}$ after $\mathbf{d_n}$, creating a fixed circle batch retrieval pattern. We refer to this sampling method as Fixed Circle Pseudo Random Sampling. The random reads are subsequently reduced to sequential reads on Disk. Therefore, FCPR Sampling is widely adopted by SGD. Let $n_\mathbf{d}$ to be the size of a dataset and $n_\mathbf{b}$ to be the batch size. The size of sample space is $n_\mathbf{d}/n_\mathbf{b}$, and the batch being assigned to iteration j is $\mathbf{d_t}$, where

$$t = j \bmod * \frac{n_\mathbf{d}}{n_\mathbf{b}}.$$

At any iteration, the model always anticipate a fixed batch, as the batch will flow into the model at iteration $t + 1 \cdot epoch, \dots, t + n \cdot epcoh$. If the training of a batch is dominated by the gradient update on itself, the loss of this batch is predominately reduced at iteration $t, t + 1 * epoch, t + 2 * epoch, \dots, t + n * epoch$. Since the contribution from a batch's gradient update is different, the repetitive batch retrieval pattern fosters the batches' distinctive training speeds. However, the FCPR sampling in SGD treats batches identically.

The problem of FCPR sampling is the consistent gradient updates on batches regardless of the model's training status. It is inefficient to update a small loss batch as frequently as a large loss batch. Fig. 1b demonstrates the yellow batch are fully trained after the epoch 1600, while the blue batch does not until the epoch 2100. During epochs [1600, 2100], the yellow batch stays fully trained most of time indicating unnecessary training iterations on it. Besides, we also verify that the contribution of a batch's gradient update is different. Therefore, regulating the training iterations w.r.t the model's training status on batches will improve the efficiency of SGD.

4. Inconsistent stochastic gradient descent

In this section, we present Inconsistent Stochastic Gradient Descent that rebalances the training effort w.r.t a batch's training status. The inconsistency is reflected by the non-uniform gradient updates on batches. The first question is how to dynamically identify a slow or under-trained batch during the training. We model the training as a stochastic process, and apply the upper control limit to dynamically identify a under-trained batch. The second question is how to accelerate a under-trained batch. We propose a new optimization to be solved on the batch, the objective of which is to accelerate the training without drastic parameters changes. For practical considerations, we also study the effects of ISGD batch size on the convergence rate, system saturations and synchronization cost.

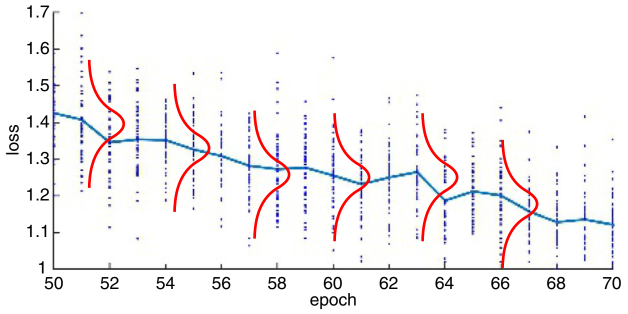


Fig. 2. Each blue dots depicts the loss of a batch produced from the CIFAR network in Caffe on CIFAR-10, and losses are grouped by epochs. The central line is the average loss. From the figure, it is legitimate to assume the losses of batches in an epoch follow the normal distribution, and the training is at reducing the mean of population tile the network converges.

4.1. Identifying under-trained batch

ISGD models the training as a stochastic process that slowly reduces down the average loss of batches. We assume the normal distribution on the batch's loss in an epoch. The reasons are that: (1) SGD demands a small learning rate (lr) (Bottou, 2010) to converge, and lr is usually less than 10^{-1}. lr determines the step length, while the normalized gradient determines the step direction. The small value of lr limits the contribution made by a gradient update, thus the training process is to gradually reduce down the loss toward a local optimal. (2) Each batch represents the original dataset, and there exists a strong correlation among batches in the training. This implies that the loss of a batch will not be drastically different from the average at any iteration t. Fig. 2 demonstrates the loss distribution of training a network on CIFAR-10, in which the losses are arranged by epochs. From the figure, it is valid to assume the normal distribution on the loss in an epoch. Therefore, we conclude that

the training is a stochastic process that slowly decreases the mean of losses tile the network converges.

The 3σ control limit is an effective method to monitor the abnormalities in a statistical process. Since we treat the training as a process that decreases the average loss of batches, ISGD utilizes the upper control limit to dynamically identify abnormal large-loss batches on the fly. To get the limit, ISGD calculates two important descriptive statistics, the running average loss $\bar{\psi}$ and the running standard deviation σ_ψ during the training. ISGD keeps a queue to store the losses produced by iterations in $[t - n_b, t]$, where n_b is the size of sample space (or the number of batches in an epoch). The queue functions as a moving window tracking the loss information in the previous epoch to yield $\bar{\psi}$ and σ_ψ.

$$\bar{\psi} = \frac{1}{n_b} \sum_{i=1}^{n_b} \psi_{\mathbf{w_{t-i}}}(\mathbf{d_{t-i}}) \tag{13}$$

$$\sigma_\psi = \sqrt{\frac{1}{n_b} \sum_{i=1}^{n_b} [\psi_{\mathbf{w_{t-i}}}(\mathbf{d_{t-i}}) - \bar{\psi}]^2}. \tag{14}$$

Since the queue length is fixed at n_b and the loss of a batch is a float number, the calculations of $\bar{\psi}$ and σ_ψ and the memory cost for the queue are O(1) at any iteration t. Therefore, ISGD is much more memory efficient than the variance reduction approaches that require intermediate variables of the same size as the network parameters. With $\bar{\psi}$ and σ_ψ, the upper control limit is

$$limit = \bar{\psi} + 3\sigma_\psi. \tag{15}$$

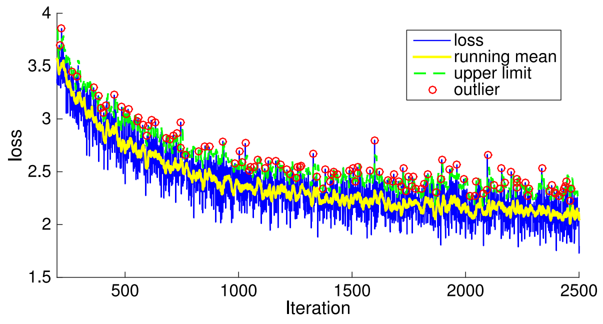


Fig. 3. Supervise the training of AlexNet with the proposed control limit. The red outliers are under-trained batches. ISGD subsequently accelerate the batches by solving Alg. 2. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Algorithm 1: Inconsistent Stochastic Gradient Descent

```
Data: d , w
Result: w'
1  begin
2      iter = 0
3      $n = \frac{n_{training}}{n_{batch}}$
4      $\overline{\psi} = 0$
5      $\sigma_{\psi} = 0$
6      $limit = 0$
7      $loss_{queue} \leftarrow \emptyset$
8      while not converge do
9          $broadcast(\mathbf{w})$
10         $[\psi, \nabla\psi] = ForwardBackward(\mathbf{d_t})$
11         $reduce(\psi)$
12         $reduce(\nabla\psi)$
13         if $iter < n$ then
14             $loss_{queue}.push(\psi)$
15             $\overline{\psi} = \frac{\overline{\psi} \cdot iter + \psi}{iter + 1}$
16         else
17             $l = loss_{queue}.dequeue()$
18             $\sigma_{\psi} = STD(loss_{queue})$
19             $\overline{\psi} = \frac{\overline{\psi} \cdot n - l + \psi}{n}$
20             $limit = \overline{\psi} + 3 * \sigma_{\psi}$
21         $\mathbf{w} = \mathbf{w} - \eta \cdot \nabla\psi$
22         if $\psi > limit$ and $iter > n$ then
23             minimize Eq. 17 with Alg.2
24         iter++
```

In this case, we adopt the 3σ control limit. The multiplier before the σ_{ψ} plays an important role in between the exploration of new batches and the exploitation of the current batch. Please refer to the discussion of Alg. 1 in Section 4.2 for more discussion. If the loss of current iteration t is

$$\psi_{\mathbf{w_{t-1}}}(\mathbf{d_t}) > limit \tag{16}$$

we consider $\mathbf{d_t}$ as a under-trained batch.

Fig. 3 demonstrates an example of the proposed method to identify a under-trained batch on the fly. The blue line is the loss of batch, and the yellow line is the running average $\bar{\psi}$. The green line is the upper control limit, and the red dots are outliers considered as under-trained. The experiment is conducted with AlexNet on ImageNet, and it is clear that ISGD successfully identifies the large-loss batches in the training with the proposed approach.

Algorithm 2: Solving the conservative subproblem to accelerate a under-trained batch

```
Data: $\mathbf{d_t}$, $\mathbf{w}$, $\mathbf{w_{t-1}}$, stop, limit
Result: $\mathbf{w}$
1  begin
2      iter = 0
3      while $iter < stop$ and $\psi > limit$ do
4          $[\psi, \nabla\psi] = ForwardBackward(\mathbf{d_t})$
5          $reduce(\psi)$
6          $reduce(\nabla\psi)$
7          $\mathbf{w} = \mathbf{w} - \zeta\{[\psi_{\mathbf{w}}(\mathbf{d_t}) - limit]\nabla\psi_{\mathbf{w}}(\mathbf{d_t}) + \frac{\epsilon(\mathbf{w} - \mathbf{w_{t-1}})}{n_w}\}$
8          $broadcast(\mathbf{w})$
9          iter++
```

4.2. Inconsistent training

The core concept of our training model is to spend more iterations on the large-loss batches than the small-loss ones. The batch retrieval pattern in ISGD is similar to FCPR sampling but with the following important difference. Once a batch is identified as under-trained, ISGD stays on the batch to solve a new sub-optimization problem to accelerate the training, and the batch receives additional training iterations inside the sub-problem. In this case, ISGD does not compromise the system efficiency of FCPR sampling, while it still regulates the training effort across the batches. The new subproblem is

$$\min_{\mathbf{w}} \quad \phi_{\mathbf{w}}(\mathbf{d_t}) = \frac{1}{2}\|\psi_{\mathbf{w}}(\mathbf{d_t}) - limit\|_2^2 + \frac{\epsilon}{2n_w}\|\mathbf{w} - \mathbf{w_{t-1}}\|_2^2 \tag{17}$$

where n_w is the number of weight parameters in the network and ϵ is a parameter for the second term. The first term minimizes the difference between the loss of current under-trained batch $\mathbf{d_t}$ and the control limit. This is to achieve the acceleration effect. The second term is a conservative constraint to avoid drastic parameter changes. Please note the second term is critical because overshooting a batch negatively affects the training on other batches. The parameter ϵ adjusts the conservative constrain, and it is recommended to be 10^{-1}. The derivative of Eq. (17) is:

$$\nabla\phi_{\mathbf{w}}(\mathbf{d_t}) = [\psi_{\mathbf{w}}(\mathbf{d_t}) - limit]\nabla\psi_{\mathbf{w}}(\mathbf{d_t}) + \frac{\epsilon(\mathbf{w} - \mathbf{w_{t-1}})}{n_w} \tag{18}$$

Please note $limit$, $\mathbf{w_{t-1}}$ and $\mathbf{d_t}$ are constants. Solving Eq. (17) precisely incurs the significant computation and communication overhead, which offsets the benefit of it. In practice, we approximate the solution to the new subproblem, Eq. (17), with the early stopping. This avoids the huge searching time wasted on hovering around the optimal solution. A few iterations, 5 for example, are good enough to achieve the acceleration effects. Therefore, we recommend approximating the solution by the early stopping.

Alg. 1 demonstrates the basic procedures of ISGD. Since the training status of a batch is measured by the loss, ISGD identifies a batch as under-trained if the loss is larger than the control limit $\overline{\psi} + 3 * \sigma_{\psi}$ (Line 20). A stringent limit triggers Eq. (17) more frequently. This increases exploitation of a batch, but it also decreases the exploration of batches to a network in the fixed time. Therefore, a tight limit is also not desired. A soft margin, 2 or 3 σ_{ψ}, is preferred in practice; and this margin is also widely applied in Statistical Process Control to detect abnormalities in a process. We recommend users adjusting the margin according to the specific problem. ISGD adopts a loss queue to dynamically track the losses in an epoch so that the average loss, $\overline{\psi}$, is calculated in O(1) (line 17). The loss queue tracks iterations in the previous epoch; the length of it equals to the length of an epoch. Similarly, calculating

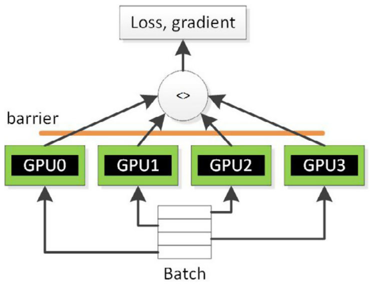


Fig. 4. The parallelization of ISGD with the data parallelism, and ⟨⟩ indicates the *reduce* of all sub-batches' gradients to yield the gradient of original batch.

σ_ψ is also in O(1) time (line 18). We do not initiate Alg. 2 until the first epoch to build up a reliable limit (line 22 the condition of iter $> n$).

We also extend Alg. 1 to work on multi-GPUs using standard MPI communication collectives (Gabriel et al., 2004), such as *broadcast* and *reduce*. *broadcast* copies the target vector to all computing devices, while *reduce* computes as follows: $reduce(\nabla\psi) = \sum_{i=0}^{nGPUs}\nabla\psi_i$. Please refer to Section 4.4 for the details of ISGD parallelization.

Alg. 2 outlines the procedures to solve the conservative subproblem on a under-trained batch. The conservative subproblem intends to accelerate the under-trained batch without drastic weight changes. The update equation in line 7 corresponds to Eq. (18). Specifically, $[\psi_{\mathbf{w}}(\mathbf{d_t}) - limit]\nabla\psi_{\mathbf{w}}(\mathbf{d_t})$ is the gradient of $\frac{1}{2}\|\psi_{\mathbf{w}}(\mathbf{d_t}) - limit\|_2^2$ to accelerate the training of a under-trained batch; the second term, $\frac{\epsilon}{n_w}(\mathbf{w} - \mathbf{w_{t-1}})$, is the gradient of $\frac{\epsilon}{2n_w}\|\mathbf{w} - \mathbf{w_{t-1}}\|_2^2$ that bounds significant weight changes. The *limit* is the same upper control threshold in Alg. 1. The *stop* specifies the maximal approximate iterations to reflect the early stopping. ζ is a constant learning rate.

The neural network training needs gradually decrease the learning rate to ensure the convergence (Bottou, 1998). It is a common tactic to decrease the learning rate w.r.t training iterations. The inconsistent iterations of ISGD requires a new way to guide the learning rate. Instead, ISGD decreases the learning rate w.r.t the average loss of a dataset. The average loss is better than iterations, as it directly reflects the training status of the model, while calculating the average loss of a dataset is expensive. Since the average loss in Eq. (13) is from the latest scan of dataset (or losses in an epoch), it is approximate to the average loss of dataset. Hence, we use the average loss (Alg. 1, line 19) to guide the learning rate.

4.3. Extend to other SGD variants

It is straight forward to extend the inconsistent training to other SGD variants. For example, Momentum (Qian, 1999) updates the weight with the following equations

$$\begin{aligned}\mathbf{v_{t+1}} &= \mu\mathbf{v_t} - \alpha\nabla\psi(\mathbf{w_t})\\ \mathbf{w_{t+1}} &= \mathbf{w_t} + \mathbf{v_{t+1}}\end{aligned} \tag{19}$$

and the Nesterov accelerated gradient follows the update rule of

$$\begin{aligned}\mathbf{v_{t+1}} &= \mu\mathbf{v_t} - \alpha\nabla\psi(\mathbf{w_t} + \mu\mathbf{v_t})\\ \mathbf{w_{t+1}} &= \mathbf{w_t} + \mathbf{v_{t+1}}.\end{aligned} \tag{20}$$

To introduce the inconsistent training to these SGD variants, we only need change the line 21 of Alg. 1 according to Eqs. (19) and (20), respectively. The Alg. 2 remains the same.

4.4. Parallel ISGD

ISGD intends to scale over the distributed or multiGPU system using MPI-style collectives such as *broadcast*, *reduce*, and *allreduce* (Wang, Wu, Bosilca, Vuduc, & Xu, 2016). Alg. 1 and Alg. 2 are already the parallel version manifested by the collectives in them. Fig. 4 demonstrates the data parallelization scheme inside ISGD. Let us assume there are n computing nodes, each of which is a GPU or a server in a cluster. Each node contains a model duplicate. A node fetches an independent segment of the original batch referred to as the sub-batch. Subsequently, all nodes simultaneously calculate sub-gradients and sub-losses with the assigned sub-batches. Once the calculation is done, the algorithm *reduce* sub-gradients and sub-losses (Lines 10–12 in Alg. 1) to a master node so as to acquire a global gradient and loss. Then, the master node updates network weights (line 21 in Alg. 1) and *broadcast* (line 9 in Alg. 1) the latest weights. Please refer to Wang Wu, Bosilca et al., (2016a) for more discussion regarding the MPI-style collectives and the SGD parallelization. Therefore, ISGD separates the algorithm from the system configurations by employing MPI-style collectives (Gabriel et al., 2004). Since MPI is an industrial and academia standard, ISGD is highly portable on various heterogeneous distributed system.

4.5. Batch size and convergence speed

Batch size is the key factor to the parallelism of ISGD. As operations on a batch are independent, scaling ISGD on systems with the massive computing power prefers a sufficiently large batch. An unwieldy large batch size, however, is detrimental to the convergence rate under the limited computing budget. Current convergence rate analysis utilizes iterations as the only performance metric, but it fails to consider the fact that an iteration faster algorithm may cost more time than the slower counterpart. Hence, it is practical to analyze the convergence rate in the time domain.

Let us assume the maximal processing capability of a system is C_1 images per second, and the time spent on synchronizations is C_2 seconds. Network cost is a constant because it only depends on the size of network parameter. A gradient update essentially costs:

$$\begin{aligned}t_{iter} &= t_{comp} + t_{comm}\\ &= \frac{n_b}{C_1} + C_2\end{aligned} \tag{21}$$

where n_b is the batch size. Given fixed time t, the number of gradient updates is

$$T = \frac{t}{t_{iter}}. \tag{22}$$

After T gradient updates, the loss is bounded by Dekel, Gilad-Bachrach, Shamir, and Xiao (2012)

$$\psi \leqslant \frac{1}{\sqrt{n_b T}} + \frac{1}{T}. \tag{23}$$

Let us assume equality in Eq. (23) and substitute Eq. (22). It yields Eq. (24) that governs loss ψ, time t and system configurations C_1 and C_2:

$$\psi t = \sqrt{t}\sqrt{\frac{n_b + C_1C_2}{n_bC_1}} + \frac{n_b}{C_1} + C_2. \tag{24}$$

Fig. 5 presents the predicted training time under two system configurations calculated by Eq. (24) at different batch sizes $n_b \in (0, 3000)$. By fixing ψ, the equation approximates the total training time under different batches. The figure demonstrates the optimal batch size of the first and second system are 500 and 1000 respectively. In this case, a faster system needs a larger batch. The performance of both systems deteriorates afterward. As a result, the optimal batch size is a tradeoff between system configurations and algorithmic convergences.

Table 1
The average top accuracy and time reported from the training with ISGD and SGD on MNIST, CIFAR and ImageNet. IMP stands for the ISGD's improvement over SGD. The data is collected from 10 runs, and ISGD consistently outperforms SGD on 3 datasets.

	Highest reported Top/Top-5 accuracy			Average Top/Top-5 accuracy			Normalized average time to Top/Top-5 accuracy		
	SGD	ISGD	IMP	SGD	ISGD	IMP	SGD	ISGD	IMP
MNIST	99.08%	99.19%	0.11%	99.05%	99.17%	0.12%	1	0.744	25.6%
CIFAR	76.01%	76.55%	0.54%	75.78%	76.42%	0.64%	1	0.772	22.78%
ImageNet	81.10%	82.12%	1.01%	80.56%	81.83%	1.27%	1	0.854	14.53%

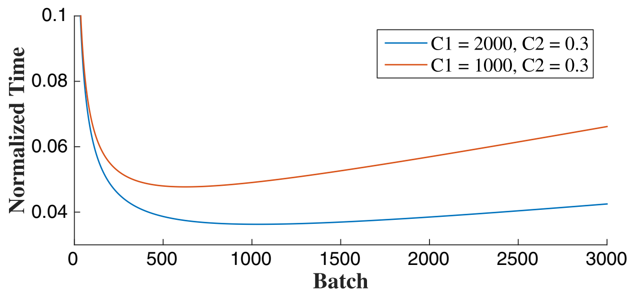


Fig. 5. The predicted training time calculated by Eq. (24) at different batch sizes.

5. Experiments

In this section, we demonstrate the performance of inconsistent training against SGD variants such as Momentum and Nesterov on a variety of widely recognized datasets including MNIST (LeCun, Cortes, & Burges, 1998), CIFAR-10 and ImageNet. MNIST has 60000 handwritten digits ranging from 0 to 9. CIFAR-10 has 60000 32×32 RGB images categorized in 10 classes. ILSVRC 2012 ImageNet has 1431167 256×256 RGB images depicting 1000 object categories. We use LeNet, Caffe CIFAR-10 Quick, and AlexNet to train on MNIST, CIFAR-10, and ImageNet, respectively. The complexity of networks is proportional to the size of datasets. Therefore, our benchmarks cover the small, middle, and large scale CNN training.

We conduct the experiments on a multiGPU system with 4 NVIDIA Maxwell TITAN X. The CUDA version is 7.5, the compiler is GCC 4.8.4. The machine has 64 GB RAM and 1TB SSD. CPU is Xeon E5 4655 v3. Caffe is built with the cuDNN version 4. The GPU machine was exclusively owned by us during the benchmark.

5.1. Qualitative evaluation of inconsistent training

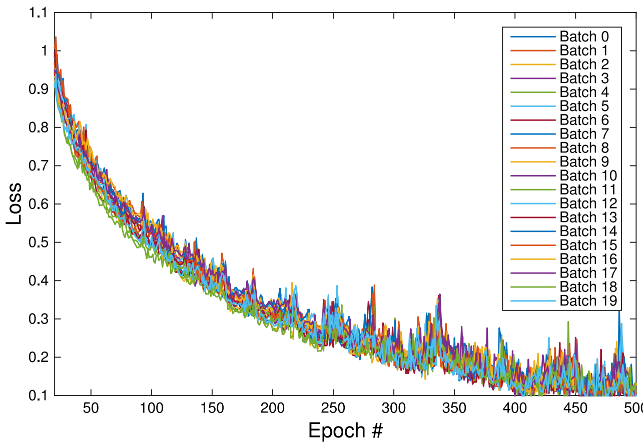
This section intends to qualitatively evaluate the impacts of inconsistent training. The purpose of inconsistent training is to rebalance the training effort across batches so that the large-loss batch receives more training than the small-loss one. To qualitatively evaluate the impacts of inconsistent training, we exam the progression of the loss distribution, the average loss, the standard deviation of batch's loss distribution, as well as the validation accuracy. We setup the training with Caffe CIFAR-10 Quick network on CIFAR-10 dataset. The batch size is set at 2500 yielding 20 independent batches. Figs. 6a and 6b present the loss distribution of 20 batches in the training. We arrange losses in epochs as the solver explores a batch only once in an epoch.

The inconsistent training has the following merits. (1) ISGD converges faster than SGD due to the improvement of training model. We measure the convergence rate by the average loss of batches in a dataset, and the method conforms to the training definition in Eq. (3). The average loss data in Fig. 6d demonstrates that ISGD converges faster than SGD. In contrast with SGD, the lower average loss of ISGD after $iter > 7000$ (Fig. 6d) explains the better accuracy of ISGD after testing 9 (Fig. 6e). The validation accuracy of ISGD in Fig. 6e is also above SGD, that is consistent with data in Fig. 6d that the average loss of ISGD is below the average loss of SGD in the training. These justify the convergence advantage of inconsistent training. (2) ISGD dynamically accelerates the large-loss batch in the progress of training to reduce the training gap with small-loss batches. Therefore, the variation of batch's training status is less than the one trained by SGD. Please note we measure the training status of a batch with its loss, and the variation of batch's training status is measured by the standard deviation of batch's loss distribution. Fig. 6c demonstrates the inconsistent training successfully attenuates the training variations among batches. When $iter \in [1000, 6000]$, the std of batch's loss distribution of ISGD is much lower than SGD. The result is also consistent with the loss distribution in Figs. 6a and 6b, in which the loss distribution of SGD is much wider than ISGD at $epoch \in [50, 300]$.

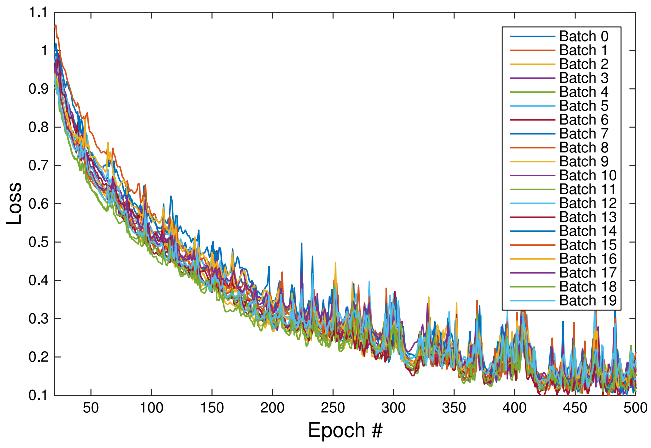
5.2. Performance evaluations

The setup of each comparisons, ISGD V.S. SGD, has been carefully set to be the single factor experiment, i.e. the only difference is the inconsistent training. Some parameters of SGD greatly affect the training performance, setting different values on them jeopardizes the credibility of experiments. Therefore, we ensure the parameters of SGD and ISGD to be same in each comparisons. The first parameter considered is the learning rate. The MNIST tests adopt a constant learning rate of 0.01, and CIFAR tests adopt a constant learning rate of 0.001. Both cases are consistent with the solver defined in Caffe. Caffe fixes the learning rate for these two cases because networks yield the satisfactory accuracies , 75% on CIFAR and 99% on MNIST, without shrinking the learning rate. Since AlexNet has to shrink lr, the learning rate of it has 3 possibilities: $lr = 0.015$ if the average loss $\overline{\psi} \in [2.0, +\infty]$, $lr = 0.0015$ if $\overline{\psi}$ in $[1.2, 2.0)$, and $lr = 0.00015$ if $\overline{\psi}$ in $[0, 1.2)$. The batch size is also same for each comparison in CIFAR, MNIST and ImageNet. We adopt a large batch to fully saturate 4 GPUs. For other parameters such as the weight decay and momentum, they are also same through all the tests.

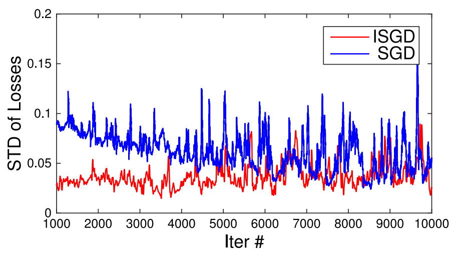
ISGD consistently outperforms SGD in all tests manifesting the effectiveness of inconsistent training. Please note both methods incorporate the momentum term. Since an iteration of ISGD is inconsistent, we test every other 2, 6, 900 s (only count the training time with the test time excluded) for MNIST, CIFAR and ImageNet tests, respectively. The horizontal dashed line represents the target accuracy, and the total training time starts from 0 to the point that the validation accuracy is consistently above the dashed line. In the ImageNet test, ISGD demonstrates the 14.94% faster convergence than SGD. SGD takes 21.4 h to reach the 81% top 5 accuracy, while ISGD takes 18.2 h (Fig. 7c) . In the CIFAR test, ISGD demonstrates 23.57% faster convergence than SGD. The top accuracy for CIFAR-Quick network reported on CIFAR-10 is 75%. After 306 s, the test accuracy of SGD is steadily above 75%, while ISGD only takes 234 s (Fig. 7b). Finally, ISGD demonstrates 28.57% faster convergence than SGD on MNIST dataset. It takes SGD 56 s to reach the 99% top accuracy, while ISGD only takes 40 s. Since the training is essentially a stochastic process, the performance subjects to changes. We repeat each test cases 10 times, and we list the performance data in Table 1. The results also uphold the convergence advantages of inconsistent training.



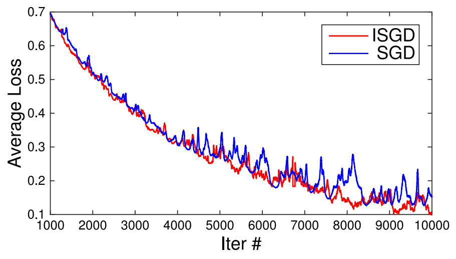
(a) ISGD loss distribution by epochs. The X and Y axes are same as (b).



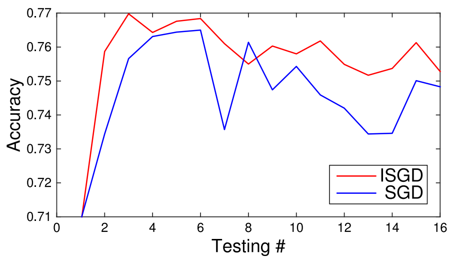
(b) SGD loss distribution by epochs.



(c) STD of the batch's loss distribution.

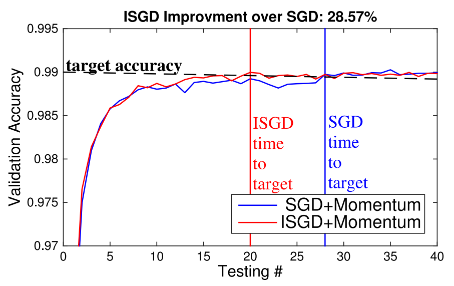


(d) Average loss of 20 batches.

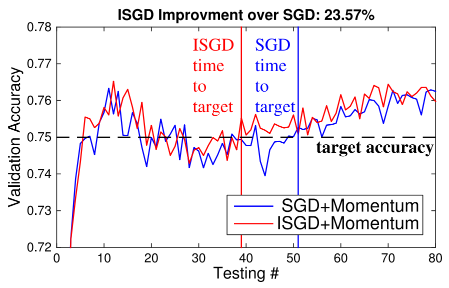


(e) Validation accuracy.

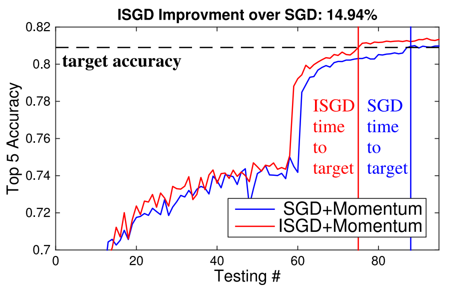
Fig. 6. The loss distribution, the average loss, the standard deviation of loss distribution, as well as the validation accuracy in the training on CIFAR.



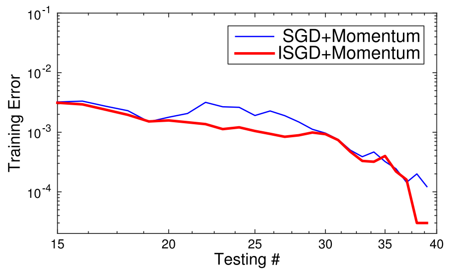
(a) MNIST test accuracy.



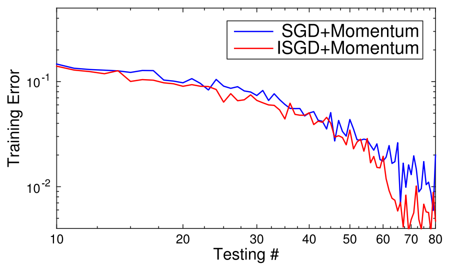
(b) CIFAR test accuracy.



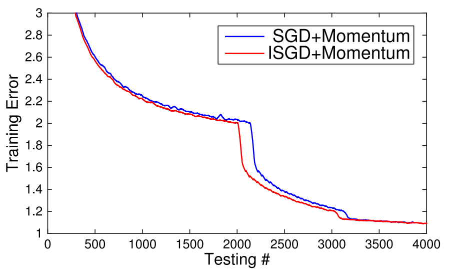
(c) ImageNet top 5 accuracy.



(d) MNIST train error.



(e) CIFAR train error.

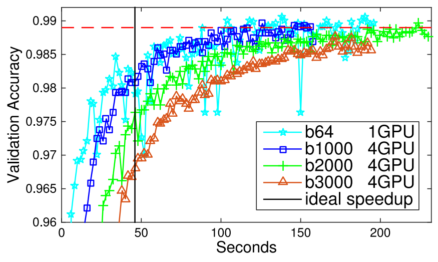


(f) ImageNet train error.

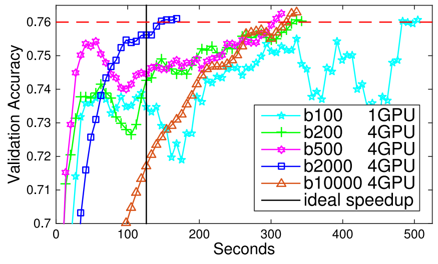
Fig. 7. The validation accuracy and training loss of LeNet, Caffe-Quick, AlexNet on MNIST,CIFAR and ImageNet. ISGD consistently outperforms SGD.

To explain the performance advantages of ISGD, we also use the training dataset to test. Whereas, the training set of ImageNet 256 GB is too large to be tested, we use $\overline{\psi}$ in Alg. 1 to approximate the training error. Figs. 7d–7f demonstrate the training error of ISGD is consistently below the SGD. The results demonstrate the benefit of inconsistent training, and they also explain the good validation accuracy of ISGD in Figs. 7a–7c.

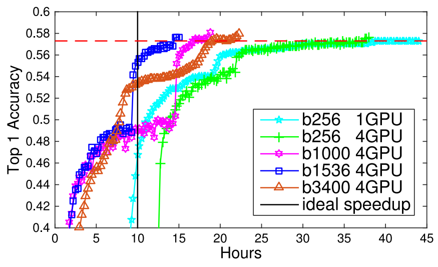
The inconsistent training is also compatible with the Nesterov accelerated gradient. Fig. 9 demonstrates the validation accuracy and the training loss progression on ImageNet trained with the Nesterov accelerated gradient. The inconsistent training beats the regular Nesterov method. If set 58% top 1 accuracy as the threshold, the inconsistent training takes 65 tests to exceed the threshold, while the regular one takes 75 tests. Please note the time interval



(a) MNIST.



(b) CIFAR.



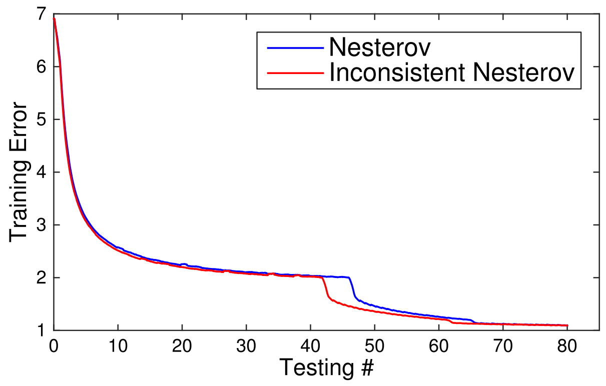
(c) ImageNet.

Fig. 8. The effect of batch size on the total training time. The experiments are conducted with 4 TITAN-X GPUs.

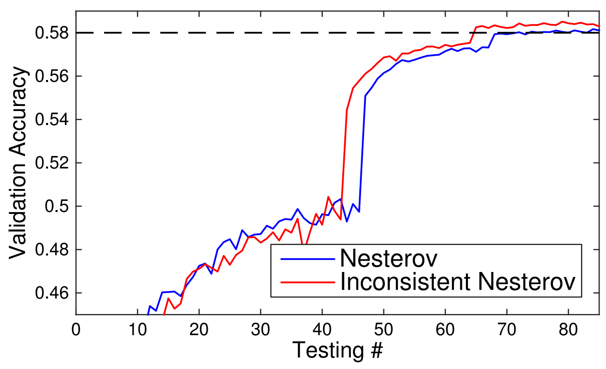
of two consecutive tests is fixed. Therefore, the inconsistent training demonstrates the 13.4% performance gain. The compatibility is under our expectation. The Nesterov method accelerates the convergence by considering the curvature information, while ISGD rebalances the training across batches.

5.3. Time domain convergence rate W.R.T batch size on MultiGPUs

Fig. 8 demonstrates convergence speeds at different batch sizes on MNIST, CIFAR and ImageNet datasets. The figures reflect the following conclusions: (1) A sufficiently large batch is necessary to the multiGPU training. The single GPU only involves computations t_{compt}, while the multiGPU training entails an additional term t_{comm} for synchronizations. A small batch size for the single GPU training is favored to ensure the frequent gradient updates. In the multiGPU training, the cost of synchronizations linearly increases with the number of gradient updates. Increasing batch size improves the convergence rate, thereby fewer iterations and synchronizations. Besides, it also improves system utilizations and saturations. As a consequence, a moderate batch size is favored to the multiGPU training as indicated in Fig. 8. (2) An unwieldy batch size slows down the convergence. Because computations linearly increase with the batch size, which reduces the number of gradient updates in a limited time. The declined convergence speed is observable in Figs. 8a–8c when batch size is set at 3000, 10000, 3400, respectively.



(a) ImageNet train error.



(b) ImageNet top 1 accuracy.

Fig. 9. Train ImageNet with the inconsistent Nesterov accelerated gradient.

6. Summary

ISGD models the training as a stochastic process, and it automatically picks the large loss batch to accelerate the training on them. ISGD accelerates large loss batches through the inconsistent training, the core concept of which allows for additional gradient updates. Extensive experiments have demonstrated our method is effective at acceleration of large loss batches in the network training.

References

Bottou, L. (1998). Online learning and stochastic approximations. *On-Line Learning in Neural Networks*, *17*(9), 142.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186). Springer.

Byrd, R. H., Chin, G. M., Nocedal, J., & Wu, Y. (2012). Sample size selection in optimization methods for machine learning. *Mathematical Programming*, *134*(1), 127–155.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes déquations simultanées. *Comptes Rendus de l'Academie des Sciences*, *25*(1847), 536–538.

Collobert, R., Kavukcuoglu, K., & Farabet, C. (2011). Torch7: A matlab-like environment for machine learning, EPFL-CONF-192376, BigLearn, NIPS Workshop.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, 1223–1231.

Dekel, O., Gilad-Bachrach, R., Shamir, O., & Xiao, L. (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research (JMLR)*, *13*, 165–202.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on* (pp. 248–255). IEEE.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., et al. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2625–2634).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, *12*, 2121–2159.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., et al. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European parallel virtual machine/message passing interface users group meeting* (pp. 97–104). Springer.

Gimpel, K., Das, D., & Smith, N. A. (2010). Distributed asynchronous online learning for natural language processing. In *Proceedings of the fourteenth conference on computational natural language learning* (pp. 213–222). Association for Computational Linguistics.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on multimedia* (pp. 675–678). ACM.

Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*, 315–323.

Krizhevsky, A., & Hinton, G. (2010). Convolutional deep belief networks on cifar-10, Unpublished Manuscript 40.

Le Cun, L. B. Y., & Bottou, L. (2004). Large scale online learning. *Advances in Neural Information Processing Systems*, *16*, 217.

LeCun, Y., Cortes, C., & Burges, C. J. (1998). The MNIST database of handwritten digits, 1998.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 661–670). ACM.

Moody, J., Hanson, S., Krogh, A., & Hertz, J. A. (1995). A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, *4*, 950–957.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151.

Reddi, S. J., Hefny, A., Sra, S., Poczos, B., & Smola, A. J. (2015). On variance reduction in stochastic gradient descent and its asynchronous variants. *Advances in Neural Information Processing Systems*, 2647–2655.

Reddi, S. J., Hefny, A., Sra, S., Póczós, B., & Smola, A. (2016). Stochastic variance reduction for nonconvex optimization. ArXiv preprint arXiv:1603.06160.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 400–407.

Roux, N. L., Schmidt, M., & Bach, F. R. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. *Advances in Neural Information Processing Systems*, 2663–2671.

Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., & Moreno-Noguer, F. (2015). Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the IEEE international conference on computer vision* (pp. 118–126).

Sutskever, I. (2013). *Training recurrent neural networks* (Ph.D. thesis), University of Toronto.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. *ICML (3)*, *28*, 1139–1147.

Tseng, P. (1998). An incremental gradient (-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, *8*(2), 506–531.

Wang, C., Chen, X., Smola, A. J., & Xing, E. P. (2013). Variance reduction for stochastic gradient optimization. *Advances in Neural Information Processing Systems*, 181–189.

Wang, L., Wu, W., Bosilca, G., Vuduc, R., & Xu, Z. (2016a). Efficient communications in training large scale neural networks, ArXiv preprint arXiv:1611.04255.

Wang, L., Wu, W., Xu, Z., Xiao, J., & Yang, Y. (2016b). BLASX: A high performance level-3 BLAS library for heterogeneous multi-GPU computing. In *Proceedings of the 2016 international conference on supercomputing* (p. 20). ACM.

Yu, H.-F., Hsieh, C.-J., Si, S., & Dhillon, I. S. (2014). Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, *41*(3), 793–819.

Zhao, P., & Zhang, T. (2014). Stochastic optimization with importance sampling. ArXiv preprint arXiv:1401.2753.

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., & Oliva, A. (2014). Learning deep features for scene recognition using places database. *Advances in Neural Information Processing Systems*, 487–495.