T2R2 東京科学大学 リサーチリポジトリ
Science Tokyo Research Repository

論文 / 著書情報
Article / Book Information

Title	Asynchronous, data-parallel deep convolutional neural network training with linear prediction model for parameter transition
Author	Ikuro Sato, Ryo Fujisaki, Yosuke Oyama, Akihiro Nomura, Satoshi Matsuoka
Journal/Book name	International Conference on Neural Information Processing, volume 10635, , pp. 305-314
発行日 / Issue date	2017, 11
DOI	https://doi.org/10.1007/978-3-319-70096-0_32
権利情報 / Copyright	The original publication is available at www.springerlink.com.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Powered by T2R2 (Science Tokyo Research Repository)

Asynchronous, Data-Parallel Deep Convolutional Neural Network Training with Linear Prediction Model for Parameter Transition

Ikuro Sato[1], Ryo Fujisaki[1], Yosuke Oyama[2], Akihiro Nomura[2], and Satoshi Matsuoka[2]

[1] Denso IT Laboratory, Inc., Tokyo, Japan
[2] Tokyo Institute of Technology, Tokyo, Japan
{isato,rfujisaki}@d-itlab.co.jp
{oyama.y.aa@m,nomura.a.ac@m,matsu@is}.titech.ac.jp

Abstract. Recent studies have revealed that Convolutional Neural Networks requiring vastly many sum-of-product operations with relatively small numbers of parameters tend to exhibit great model performances. Asynchronous Stochastic Gradient Descent provides a possibility of large-scale distributed computation for training such networks. However, asynchrony introduces stale gradients, which are considered to have negative effects on training speed. In this work, we propose a method to predict future parameters during the training to mitigate the drawback of staleness. We show that the proposed method gives good parameter prediction accuracies that can improve speed of asynchronous training. The experimental results on ImageNet demonstrates that the proposed asynchronous training method, compared to a synchronous training method, reduces the training time to reach a certain model accuracy by a factor of 1.9 with 256 GPUs used in parallel.

1 Introduction

One of the findings in the last few years about Convolutional Neural Network (CNN) is that models requiring a relatively large number of sum-of-product Operations Per Parameter (OPP) in the forward step tend to exhibit high accuracies in recognition tasks [1–3]. One such example can be seen in the ILSVRC classification task [4], where GoogLeNet [2], an example of the computationally intensive deep models with about 221 OPP, scored 6.67% top-5 error rate, whereas AlexNet [5], a parameter-rich model with about 11 OPP, scored 16.4%.

Data-parallel computation in a computing cluster provides possibilities of significant speed-up in training of computationally intensive models [6–9],by which we mean models requiring a large amount of computation to produce gradients with a relatively small number of parameters, like GoogLeNet. In data-parallelism each processor basically repeats two kinds of processes: 1) the gradient-computing process reads a small set of training data, which we refer to as "sub-batch" in this paper, and computes the gradients of the sub-batch

cost; and 2) the parameter-update process updates parameters by adding the gradients from all or a part of the processors utilizing high-speed interconnect communication. Data-parallel training of a computationally intensive model is efficient, compared to that of parameter-rich models, because communication burden of the former is relatively low.

Two strategies mainly exist in data-parallel neural network training: Synchronous Stochastic Gradient Descent (SSGD) [7, 9] and Asynchronous Stochastic Gradient Descent (ASGD) [6, 8, 10, 11]. In SSGD gradient-computing process and parameter-update process run one after the other, whereas in ASGD these two processes run concurrently without waiting the other to be completed. When compared two strategies under the same computational resources, ASGD generally enjoys higher parameter-update frequency for computationally intensive models. This is because ASGD does not suffer from waiting a relatively long period of gradient computation to complete a parameter update. On the other hand, an expected cost or error rate drop per parameter update of ASGD is smaller than that of SSGD in general [8, 10, 11]. In ASGD, gradients are computed based on *stale* parameters, whose timestamp is older than the current timestamp. Due to the staleness, the gradient vector computed in ASGD is no longer parallel to the steepest descent direction at current parameters. One technical challenge is to develop a mechanism that can *predict* future parameters, with which gradients are computed. If this parameter-prediction accuracy can be made high enough so that the computed gradient vector restores the steepest descent direction at current parameters to be updated, ASGD acquires nearly equal expected cost or error rate drop per update as SSGD, and as a consequence, ASGD having a relatively high update frequency outperforms SSGD in speed of training a computationally intensive CNN.

We propose an algorithm for stale parameter updates in ASGD, named PP-ASGD (PP stands for "Parameter Predicted"), aiming to improve the cost or error rate drop per parameter update, compared to a naive ASGD. The contributions of this work are as stated below:

- We propose an ASGD algorithm based on a linear prediction model for parameter transition, depending on parameter staleness and stale momentum.
- We show an experimental evidence that the proposed method provides good prediction accuracies of parameter transitions.
- We show experimental evidences that PP-ASGD reduces training time to reach a certain model accuracy, compared to a naive ASGD with no parameter prediction.
- We show an experimental evidence that PP-ASGD reduces training time to reach a certain model accuracy, compared to SSGD, by a factor of 1.9 for a computationally intensive CNN trained on ImageNet with 256 GPUs.

2 Proposed Method

In this section we discuss the proposed method that works efficiently in a type of computing clusters as stated below. Suppose we have a computing cluster, in

which each compute node contains the same number of GPUs, and any two nodes can communicate through high-speed interconnect. With such computational environment use of collective communication known as MPI-Allreduce [12] is a reasonable choice for parameter update [9, 7]. This routine executes element-wise sum of vectors (gradients, in our case) from every node and places the resultant vector (sum of gradients) to every node. The communication period necessary to run one MPI-Allreduce is typically $O(\log(\#\text{nodes}))$, and this sublinear behavior helps to avoid a communication bottleneck because the communication duration needed for an update grows moderately with respect to the number of nodes. Previous work mostly uses MPI-Allreduce for SSGD [9, 7]; however, it brings drawback of low update frequency for computationally intensive models. To overcome this drawback, we introduce an ASGD algorithm with MPI-Allreduce in Section 2.1. ASGD generally creates a relatively large staleness value in training a computationally intensive model. In Section 2.2, we discuss a parameter prediction model to mitigate this problem.

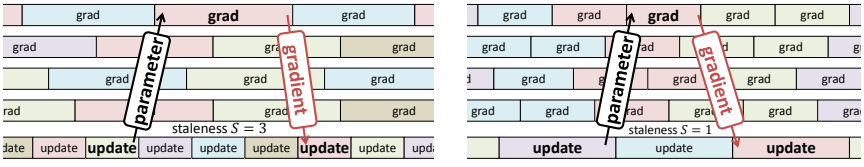
2.1 ASGD with Collective Communication

In Algorithm 1 we give a data-parallel ASGD algorithm that can yield update frequencies independent of the amount of computation needed to produce gradients [13]. [1] The parameter-update thread repeats the update process incessantly. This decouples the parameter-update process from gradient-computing process, thus makes the update frequency F_U independent of the period of gradient computation. The gradient-computation thread uses a GPU to process gradient computation repeatedly and incessantly without any synchronization. Gradient-computation frequency F_G depends on the amount of computation for gradients. For comparison, we give SSGD algorithm with MPI-Allreduce in Algorithm 4, in which update frequency does depend on the load of gradient computation.

A computationally intensive model experiences high staleness compared to a parameter-rich model in ASGD for a given number of nodes. We define time-average staleness, $\overline{S} \in \mathbb{R}$, as

$$\overline{S} = 1 + F_U/F_G. \tag{1}$$

[1] Mutexes need to be implemented in appropriate places to avoid read/write collisions.



(a) computationally intensive model (b) parameter-rich model

Fig. 1. Illustration of the time behavior of 4 `grad` threads and 1 `update` thread in ASGD.

Algorithm 1: ASGD with MPI-Allreduce

```
input  : w_0,                                   /* w_0: initial param. */
       : μ, λ,                  /* μ: momentum rate, λ: learning rate */
       : t_f, G,              /* t_f: max #updates, G: #GPUs in a node */
       : b, 𝒳               /* b: sub-batch size, 𝒳: training dataset */
output : w
1 begin
2   global w, ŵ ← w_0; D_1, D_2, ··· , D_G, M ← 0 · w_0; F ← true /* in bold face */
3   thread update(w_0, μ, λ, t_f, G)      /* See Algorithm 2 for update(). */
4   thread grad(1, b, 𝒳)                   /* See Algorithm 3 for grad(). */
5   thread grad(2, b, 𝒳)                        /* thread lines are       */
6   ···                                         /* executed in parallel. */
7   thread grad(G, b, 𝒳)
8   wait thread                  /* waits until all the thread complete */
9   w ← w
```

Algorithm 2: Function `update`

```
1 Function update(w_0, μ, λ, t_f, G)
2   for t ← 0 to t_f − 1 do
3     D_L ← D_1 + D_2 + ··· + D_G
4     D_1, D_2, ··· , D_G ← 0 · w_0
5     D_A ← MPI-Allreduce(D_L)
6     M ← μM − λD_A
7     w ← w + M
8     ŵ ← f_⌊S̄⌋(w, M) /* Eq.(2) */
9   F ← false /* training done */
```

Algorithm 3: Function `grad`

```
1 Function grad(g, b, 𝒳)
2   do
3     for k ← 1 to b do
4       x_k ← randpick(𝒳)
          /* random sampling */
5     w_ℓ ← ŵ   /* local copy */
6     D_g ← D_g + Σ_k ∇_w J(x_k; w_ℓ)
        /* grad. of cost J() */
7   while F
```

It is an addition of the count of updates in one gradient computation period and offset one, which comes from the fact that the consecutive updates run incessantly. As illustrated in Fig. 1, ASGD training of a computationally intensive model acquires a relatively large staleness value because it has a relatively large F_U/F_G.

There are mainly two approaches to mitigate problems caused by high staleness: $\overline{S}$-reduction and gradient "quality" improvement. The former approach includes use of small sub-batch size [10], and model-parallelism [5, 6, 14]. The latter approach includes our momentum-based prediction mechanism as presented next, and a delay compensation technique based on approximated Hessian [8].

2.2 Linear Prediction Model for Parameter Transition

We discuss the proposed method for predicting future parameters to improve an expected cost or error rate drop per update in ASGD. The basic idea is that right after parameters get updated, the method predicts future parameters, with which gradients are computed, so that the computed gradient vector

Algorithm 4: SSGD with MPI-Allreduce

input : $w_0, \mu, \lambda, t_f, G, b, \mathcal{X}$ /* same input as in Algorithm 1 */
output : w

begin
  global $\hat{\mathbf{w}} \leftarrow w_0$; $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_G \leftarrow 0 \cdot w_0$; $\mathbf{F} \leftarrow$ false /* in bold face */
  $w \leftarrow w_0, M \leftarrow 0 \cdot w_0$
  for $t \leftarrow 0$ **to** $t_f - 1$ **do**
    thread grad$(1, b, \mathcal{X})$ /* thread lines are */
    thread grad$(2, b, \mathcal{X})$ /* executed in parallel. */
    $\cdots$ /* See Algorithm 3 for grad(). */
    thread grad$(G, b, \mathcal{X})$ /* do-while part executed only once */
    wait thread /* waits until all the thread complete */
    $D_L \leftarrow \mathbf{D}_1 + \mathbf{D}_2 + \cdots + \mathbf{D}_G$
    $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_G \leftarrow 0 \cdot w_0$
    $D_A \leftarrow$ MPI-Allreduce(D_L)
    $M \leftarrow \mu M - \lambda D_A$
    $w \leftarrow w + M$
    $\hat{\mathbf{w}} \leftarrow w + \mu M$ /* NAG */

becomes approximately parallel to the steepest-descent direction at the time of update. Suppose we have a parameter vector $w_t \in \mathbb{R}^D$ (D is the dimension of the parameter space) at timestamp t, compute the forward and backward steps, and then use the computed gradients to update w_{t+S} to w_{t+S+1}. Here, S is an integer-valued staleness, with $S = 0$ being SSGD and $S > 0$ being ASGD. The aim of the proposed method is to design a function that can *predict* parameter vector that is $(S+1)$-timestamp ahead; *i.e.*, to design $f_S : \mathbb{R}^D \to \mathbb{R}^D$ so that $f_S(w_t, \cdot) \simeq w_{t+S+1}$.

The explicit form of the parameter prediction function that we use is

$$f_S(w_t, M_t) = w_t + M_t \sum_{S'=1}^{S+1} \mu^{S'}, \tag{2}$$

The function depends on M_t, (stale) momentum vector at timestamp t, and S, an integer-valued staleness given by $S = \lfloor \overline{S} \rfloor$, where the time-average staleness $\overline{S}$ is assumed to be measured during training. The prediction model is a natural extension of Nesterov's Accelerated Gradients (NAG) [15] to stale gradients; *i.e.*, when a staleness value is zero (SSGD), the proposed method becomes equivalent to NAG: $f_0(w, M) = w + \mu M$, as in Algorithm 4.

The proposed method is expected to work well in those cases, which the popular momentum method [16] or its variants, such as NAG, can accelerate convergence, or in other words, the gradients are quite correlated between arbitrary two consecutive iterations. If the parameter prediction accuracy can be made very high, PP-ASGD has a huge advantage to speed-up training of a computationally intensive model as PP-ASGD has a higher update frequency than SSGD.

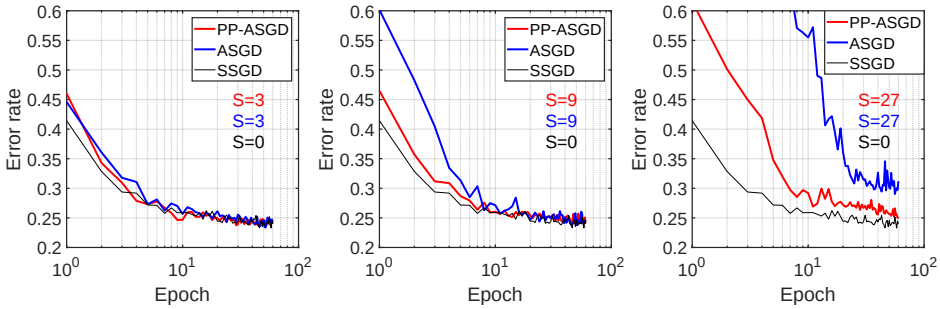


Fig. 2. Classification error rate curves of PP-ASGD, ASGD, and SSGD on the CIFAR-10 validation dataset. Staleness values of PP-ASGD and ASGD are varied: (a) $S = 3$, (b) $S = 9$, and (c) $S = 27$. Horizontal axes are in logarithmic scale.

3 Evaluation

We conducted image classification experiments to compare training times between PP-ASGD and ASGD, and between PP-ASGD and SSGD.

We used three datasets for evaluation: 1) **ImageNet-1000** [4][2] –the 1000-class ILSVRC classification dataset; 2) **ImageNet-32** –a subset of ImageNet-1000, consisting of 32 randomly chosen classes by the authors; and 3) **CIFAR-10** [17][3]. For ImageNet training, on-line data augmentation technique including random scaling, cropping and weak elastic distortion [18] was adopted. For CIFAR-10 training, no data augmentation is used. We used the minimum sub-batch size, *i.e.*, $b = 1$, for all ImageNet training.

We used following computational environments. All ImageNet experiments were conducted in TSUBAME-KFC/DL supercomputer[4]. The program of distributed training is written in C++, CUDA and OpenMPI from scratch. All CIFAR-10 experiments were conducted in a single node with one GPU with a program written in MATLAB. To test (PP-)ASGD on CIFAR-10, nonzero staleness was artificially generated.

We used simple CNN architectures as follows. Convolutional kernels always have 3×3 spatial sizes. Non-overlapping maximum-pooling is adapted. Activation function is given by $\max(a, 0.01a)$, similar to ReLU [19]. Cross entropy loss is used with softmax output. CIFAR-10 CNN has a form of `CCPCPCPFFF`, where '`C`' means convolutional, '`P`' means pooling, and '`F`' means fully-connected layers. Description of the numbers of maps and neurons are omitted. In all experiments the same momentum rate 0.99 is used.

[2] See `http://image-net.org` for details.

[3] See `https://www.cs.toronto.edu/~kriz/cifar.html` for details.

[4] Each compute node of TSUBAME-KFC/DL contains 2 Intel Xeon E5-2620 v2 CPUs and 4 NVIDIA Tesla K80. Since K80 contains 2 GPUs internally, each node has 8 GPUs for total. FDR InfiniBand is equipped for interconnect.

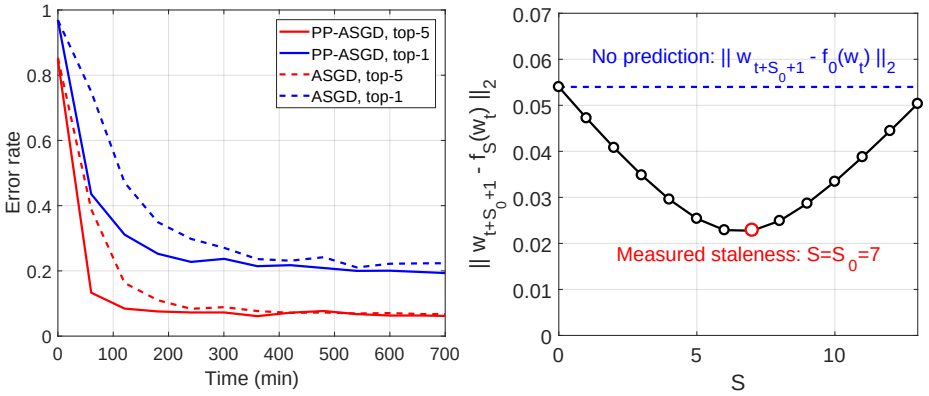


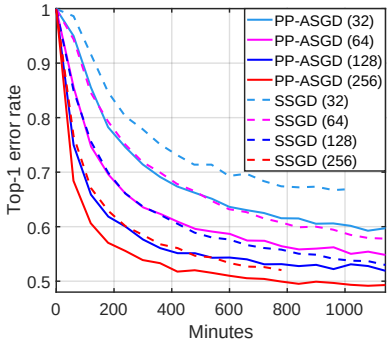
Fig. 3. Left: classification error rate curves of PP-ASGD and ASGD on the ImageNet-32 validation dataset. Right: plot of $\|w_{t+S_0+1} - f_S(w_t)\|_2$ for $S = 0, 1, \cdots, 13$ with measured staleness $S_0 = 7$. Each point is an average of 100 measurements right after 1 epoch. CNN architecture: `CPCPCCPCCPCCPCCF`. We used 32 GPUs to train each model.

3.1 Training Speed: PP-ASGD vs ASGD

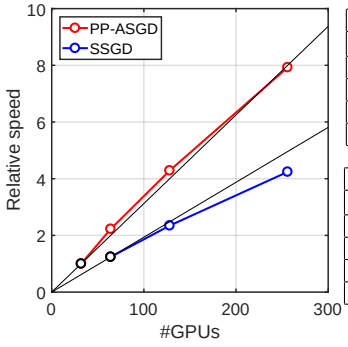
CIFAR-10 Figure 2 shows classification error rate curves of PP-ASGD and ASGD with staleness values $S = 3, 9, 27$. For $S = 3$ the error rate of PP-ASGD at a given point in epoch is lower than or similar to that of ASGD. For $S = 9$ the error rates of PP-ASGD in the interval of first 7 epochs is clearly lower than that of ASGD. The most notable speed-up is observed for $S = 27$ by roughly $5\times$ to reach the same error rate 0.3. As for generalization ability, PP-ASGD produces a much lower error rate than ASGD for $S = 27$. For the case of $S = 3$ or $S = 9$, though the error rate curve fluctuates time-to-time, the model accuracy produced by PP-ASGD is by and large equal to that produced by ASGD.

ImageNet-32 The left side of Fig. 3 shows classification error rate curves of PP-ASGD and ASGD. We used 32 GPUs (4 nodes $\times$ 8 GPUs) in each training. The time-average staleness is about 8.5 for both cases. Note that the computational time for the parameter prediction part is negligible. It is evident from the left side of Fig. 3 that PP-ASGD outperforms ASGD in training speed approximately by a factor of two to reach the same top-5 error rate, say 0.2.

The right side of Fig. 3 shows the parameter prediction error, expressed by $\|w_{t+S_0+1} - f_S(w_t)\|_2$, where S_0 is the measured staleness and S is swept from 0 to 13. From this experiment $\|w_{t+S_0+1} - f_S(w_t)\|_2$ has a minimum at $S = S_0$, indicating that the coefficient in the stale momentum term of the proposed prediction model is indeed appropriate. The horizontal dashed line indicates discrepancy between the stale parameter vector and the future ((S_0+1)-ahead) parameter vector; whereas the red circle indicates discrepancy between the predicted parameter vector by our method and the future ((S_0+1)-ahead) parameter vec-



(a)



(b)

PP-ASGD			
#gpu	$\overline{B}$	$\overline{S}$	F_U
32	20	2.6	13.4
64	43	2.5	12.1
128	99	2.3	9.9
256	278	1.9	8.2

SSGD			
#gpu	$\overline{B}$	$\overline{S}$	F_U
32	32	0.0	4.8
64	64	0.0	4.7
128	128	0.0	4.5
256	256	0.0	3.9

F_U in Hz.

(c)

Fig. 4. (a) Classification error rate curves of PP-ASGD and SSGD on the ImageNet-1000 validation dataset. Numbers in parentheses indicate #GPUs. (b) Relative speeds to reach 0.6 top-1 validation error rate. Black lines indicate ideal linear speed-up lines. (c) Time-average batch size $\overline{B}$ (here, "batch" means a set of sub-batches used for an update), time-average staleness $\overline{S}$, and update frequency F_U. All the experiments use the CNN of the same form, `CCPCCPCPCCPCF`.

tor. The latter discrepancy (by PP-ASGD) is 42% of the former discrepancy (by ASGD). It is considered that this improvement results in the training speed-up.

3.2 Training Speed: PP-ASGD vs SSGD

CIFAR-10 Figure 2 also shows classification error rate curves of SSGD, besides PP-ASGD with staleness values $S = 3, 9, 27$. SSGD has the largest error rate drop in the interval of the first few epochs, but PP-ASGD with $S = 3(9)$ reaches very similar error rates as SSGD after 4(9)-th epoch. As for $S = 27$ case, PP-ASGD clearly produces a degraded generalization performance compared with SSGD.

ImageNet-1000 We conducted large-scale training experiments on ImageNet-1000 by PP-ASGD and SSGD. In Fig. 4 the leftmost figure shows error rate curves, and the middle figure shows the relative training speeds to reach 0.6 top-1 error rate. We ran each training a couple of times with different learning rates, and show the best performing results. The learning rates used for the results shown in Fig. 4 are ranged from 1e-4 to 8e-4.[5] We did not drop learning rate during training. From the figure it is observed that PP-ASGD consistently outperforms SSGD in training speed by a factor of 1.8-1.9 when the same number

[5] In every case the learning rate is varied from 0 to the target value linearly from the beginning of the training until the end of the first epoch for stability. After this period, the learning rate is held fixed at the target value.

of GPUs are used. It is also observed that PP-ASGD exhibits a near-linear speed-up behavior with respect to the number of GPUs up to 256-GPU, while SSGD exhibits a sublinear behavior.

4 Discussion and Conclusion

In this work, we proposed PP-ASGD algorithm that uses a parameter prediction model for asynchronous, data-parallel CNN training. The prediction model is based on a linear function of a stale momentum vector with a coefficient depending on measured staleness value. Experiments showed that our model has good parameter prediction accuracies, that result in reduction of training time to reach a certain model accuracy, compared with a naive ASGD. PP-ASGD also outperforms SSGD in training speed to reach the same model accuracy by a factor of 1.9, when a computationally intensive model is trained on ImageNet using 256 GPUs in parallel.

Lastly, we discuss a possibility of further improvement of gradient quality in asynchronous settings. Zheng, *et al.* [8] proposed a delay compensation technique for asynchronous, distributed deep learning. In their method a compute thread computes gradients and an approximated Hessian matrix using stale parameters, and an update thread corrects the stale gradients by the product of the approximated Hessian and the difference vector between the stale and current parameter vectors. Our method differs in that gradients are computed by *predicted* parameters by stale momentum and that Hessian computation is not necessary. Indeed, it is expected that by *combining* the method of Zheng, *et al.* and ours gradient quality can be further improved. In the combined method, a compute thread computes gradients and approximated Hessian matrix using *predicted* parameters, and an update thread corrects the gradients by the product of the approximated Hessian and the difference vector between the *predicted* and current parameter vectors (that is, the parameter-prediction error). Our method can yield small parameter-prediction error, with which the Hessian correction term would further improve the gradient quality.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas (2016)
2. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition, Boston (2015)
3. Lin, M., Chen, Q., Yan, S.: Network in network. In: International Conference on Learning Representations, Banff (2014)
4. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision 115(3), 211–252 (2015)

5. Krizhevsky, A., Ilya, S., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: The 25th Advances in Neural Information Processing Systems, Lake Tahoe (2012)
6. Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A.Y.: Large scale distributed deep detworks. In: The 25th Advances in Neural Information Processing Systems, Lake Tahoe (2012)
7. Iandola, F.N., Ashraf, K., Moskewicz, M.W., Keutzer, K.: Firecaffe: near-linear acceleration of deep neural network training on compute clusters. ArXiv:1511.00175 (2015)
8. Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z., Liu, T.: Asynchronous stochastic gradient descent with delay compensation for distributed deep learning. ArXiv:1609.08326 (2016)
9. Wu, R., Yan, S., Shan, Y., Dang, Q., Sun, G.: Deep Image: Scaling up image recognition. ArXiv:1501.02876 (2015)
10. Gupta, S., Zhang, W., Wang, F.: Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In: IEEE International Conference on Data Mining, Barcelona (2016)
11. Zhang, W., Gupta, S., Lian, X., Liu, J.: Staleness-aware async-SGD for distributed deep learning. In: The 25th International Joint Conferences on Artificial Intelligence, New York (2016).
12. MPI: A message-passing interface standard, http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
13. Oyama, Y., Nomura, A., Sato, I., Nishimura, H., Tamatsu, Y., Matsuoka, S.: Predicting statistics of asynchronous SGD parameters for a large-scale distributed deep learning system on GPU supercomputers. In: IEEE International Conference on Big Data, Washington DC (2016)
14. Jaderberg, M., Czarnecki, W.M., Osindero, S., Vinyals, O., Graves, A., Kavukcuoglu, K.: Decoupled neural interfaces using synthetic gradients. ArXiv:1608.05343 (2016)
15. Nesterov, Y.: A method of solving a convex programming problem with convergence rate $O(1/k^2)$. Soviet Mathematics Doklady 27, 372–376 (1983)
16. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Networks 12(1), 145–151 (1999)
17. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto (2009)
18. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: The 7th International Conference on Document Analysis and Recognition, Edingburgh (2003)
19. Nair, V., Hinton, G.E.: Rectified Linear Units improve Restricted Boltzmann Machines. In: The 27th International Conference on Machine Learning, Haifa (2010)