

Data Fine-Pruning: A Simple Way to Accelerate Neural Network Training

Junyu Li, Ligang He, Shenyuan Ren, Rui Mao

▶ **To cite this version:**

Junyu Li, Ligang He, Shenyuan Ren, Rui Mao. Data Fine-Pruning: A Simple Way to Accelerate Neural Network Training. 15th IFIP International Conference on Network and Parallel Computing (NPC), Nov 2018, Muroran, Japan. pp.114-125, 10.1007/978-3-030-05677-3_10 . hal-02279554

HAL Id: hal-02279554

https://inria.hal.science/hal-02279554

Submitted on 5 Sep 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Data Fine-pruning: A Simple Way to Accelerate Neural Network Training

Junyu Li[1], Ligang He[1]✉, Shenyuan Ren[1], and Rui Mao[2]

[1] The University of Warwick, Coventry, UK
{j.li.9, liganghe, shenyuanren}@warwick.ac.uk
[2] Shenzhen University, Shenzhen, Guangdong, P.R.China
mao@szu.edu.cn

Abstract. The training process of a neural network is the most time-consuming procedure before being deployed to applications. In this paper, we investigate the loss trend of the training data during the training process. We find that given a fixed set of hyper-parameters, pruning specific types of training data can reduce the time consumption of the training process while maintaining the accuracy of the neural network. We developed a data fine-pruning approach, which can monitor and analyse the loss trend of training instances at real-time, and based on the analysis results, temporarily pruned specific instances during the training process basing on the analysis. Furthermore, we formulate the time consumption reduced by applying our data fine-pruning approach. Extensive experiments with different neural networks are conducted to verify the effectiveness of our method. The experimental results show that applying the data fine-pruning approach can reduce the training time by around 14.29% while maintaining the accuracy of the neural network.

Keywords: Deep Neural Network · Data Pruning · SGD · Acceleration.

1 Introduction

Scaling up layers and parameters in modern neural networks improves the performance dramatically and enables the discovery of sophisticated high-level features. However, it also presents enormous challenges such as the training efficiency of Deep Neural Network (DNN).

Many novel training algorithms and deep neural networks have been designed and achieved good performance with benchmark datasets and even in industrial practices. For instance, Constitutional Neural Networks (CNN) demonstrates impressive performance in areas such as image recognition and classification; Very Deep Constitutional Networks (VGG) uses an architecture with tiny convolution filters and shows a significant improvement in network performance; Deep Residual Network (ResNet) is developed to ease the training of the networks that are substantially deeper than those used previously and gain the accuracy from considerably increased depth of the networks; Wide Residual Networks (WRN)

that advances from ResNet contains a more complex architecture of network and outperforms regular deep ResNets in accuracy and efficiency.

Different configurations of DNN may lead to different level of accuracy and training efficiency (i.e., time consumption). Therefore, much research has been conducted to find the better configuration of the networks. Many endeavours have also been devoted to accelerating the training process by parallel computing. Our work does not focus on the optimization of the network configuration, but takes another approach. We assume that the configuration of the network has been optimized (or is fixed) with given settings of hyper-parameters such as the learning rate and the number of epochs. We propose to simplify the training process by reducing the training time of each epoch (regardless of the network configuration). In this approach, we dig into the training process, monitor and analyse the loss trend of each training instance. A novel method, named with Data Fine-pruning training, is developed to reduce the time consumption of training a model by sensibly and temporarily pruning a fraction of input training data in each training instance. The experimental results show that comparing to regular training, our approach is effective with majority nets and can reduce the training time by about 14.29% while maintaining the accuracy of the network.

The remainder of this paper is organized as follows. Section 2 introduces research backgrounds and motivations of our work. Section 3 reviews recent remarkable works that are related to our work. Section 4 detailed demonstrates our methods of analysing individual data, the way we run data pruning and formulations of time saved applying our approach. Section 5 illustrates the experiment results we did. Finally, a conclusion is addressed in Section 6.

2 Background and Motivation

DNNs have recently led to a series of breakthroughs in many fields such as speech recognition and image classification. Many novel learning algorithms are designed to build an effective model from a set of data and towards a prediction goal, where the model maps each input data to a prediction. Modern DNNs are typically powered by a vital training algorithm: Mini-batch Stochastic Gradient Descent (BSGD). However, there exists a heavy data dependence in the BSGD training which extremely limits the degree of parallelism.

2.1 Mini-batch Stochastic Gradient Descent

BSGD is the most widely used weight updating algorithm in recent notable neural networks. It takes a batch of data instead of using only one example each time as the input data for training. The weights of networks are same for all the instances in a batch during the forward propagation, and the changes in the weights depend on an average loss of a batch data. One core benefit of BSGD is that the changes in weights become much steadier than those in regular Stochastic Gradient Descent (SGD). Moreover, BSGD training can take

the advantage of parallel computing by parallelising the calculations within a batch, so that the processing efficiency can further increase.

$$\omega_{t+1} = \omega_t - \gamma \frac{1}{b} \sum_{i=1}^{b} \nabla_{\omega_t} \ell(f_{\omega_t}(x_i), y_i) \tag{1}$$

where b is the size of a batch data, ω is a weight vector, γ is a learning rate, and $\ell(f_\omega(x), y)$ is a loss function measuring how wrong the model is in terms of its ability to estimate the relationship between data x and corresponding label y.

2.2 Problem Setting

Figure 1 shows the loss trends when training a commonly used network – ResNet with the depth of 18 layers. The sub-figure on the top describes the trend of the values of the loss function over the testing data, which demonstrates that there are four main periods. Such different performance levels are closely related to the changes in the learning rate, where the changing points are at 60, 120, 160. The loss falls sharply from 1.5 to 0.67 in the first interval. However, from the second stage onwards, the decreasing rate slows down in each part. It can be observed from the figure that there exists a plateau in each training period with the corresponding learning rate. Such plateau always occurs in training no matter which network is used. On the contrary, the sub-figure at the bottom reflects the accuracy of the network with the test data. It can be seen that the accuracy increases as the loss decreases and the accuracy curve also contains the plateaus during the training.

In this paper, we aim to reduce the time spent on such training plateaus while achieving similar training results.

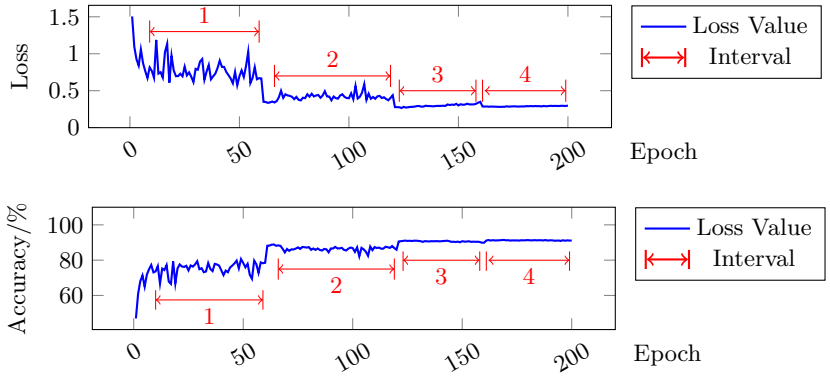


Fig. 1. A Motivation Example

3 Related Work

DNNs can produce much better results than most of other techniques in many fields [12, 19, 5, 3, 23] such as human face recognition. However, training the nets usually take quite a long time and the cost has a significant upward trend [11]. An outstanding amount of effort has been put into extending deep learning models, improving prediction performance and reducing training time consumption [10, 20, 21, 8, 22]. Regarding to the various latest acceleration mechanism for deep learning system, the technique of those could be sorted into two main cases: (i) utilize a set of GPUs to work for deep models and large training sets to take benefits from huge computing capacity facility so as to deal with large scale of models and data, (ii) optimize the training algorithms to enhance training efficiency so that directly reduce the time consumption of training.

3.1 Hardware Accelerating

GPUs are quite suitable for the computations in training a network since SGD and its variants carry high arithmetic density. It is known to all that GPU has advantages in computation capacity, thus applying a set of GPUs to train nets can deliver considerably efficient training of modestly sized Deep Neural Network practical [1, 4, 2, 6]. A common limitation of such strategies is the size of GPU onboard memory. It restricts network model and training data to be small so that the model and data can be fitted into the GPU memory. As a result, the parameters of the network and the number of data used each time are usually reduced in order to utilise the GPU(s) computation. Apart from that, there exists a mismatch in speed between GPU compute and interconnects, which leads the system extremely hard to do data parallelism in real time via a parameter server.

3.2 Algorithm Accelerating

There are a number of works on optimising training algorithm have been done up to now. Momentum [17] and Nesterov Accelerated Gradient [16] are the methods that help accelerate SGD in the relevant direction and dampens oscillations that always happen around local optima. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, nets gain faster convergence and reduced oscillation. Adagrad [7] and its extension (Adadelta) [24] are the algorithms for gradient-based optimisation that mainly do: adapt the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and more significant updates for parameters associated with infrequent features. Adam [9] is another method that computes adaptive learning rates for each parameter. It considers the decaying averages of past and past squared gradients, and update the parameters in a similar way used in Adadelta. Besides, Asynchronous Stochastic Gradient Descent (ASGD) algorithms [15, 13, 14, 25] represented by Hogwild [18] purposes to update the parameter by many workers where they are sharing a parameter server.

3.3 Data Accelerating

According to the studies on previous works on accelerating the training process, we find that there is no method that prunes the training data in a reasonable way so as to train the network with significant data that is a subset of original one. In this way, the time cost of training will be reduced by the number of data that is pruned. Our work is the first to propose an algorithm powered by such an idea. The method does not require making any changes to the original training settings, but real-time analyses performance on each data to make a choice on keeping or ignore. Please note that the decisions are not intended to be permanent, as such decisions are made based on a period of performances.

4 The Data Fine-pruning Approach

Our data fine-pruning approach reduces the training time of some specific epoch and therefore reduce the overall training time. We first investigate the loss trends of individual data in section 4.1. Second, we analyse the type of input data that should be selected for temporary pruing. Next, we introduce the data selection process and present the data fine-pruning approach in detail in subsection 4.2 and 4.3. Finally, section 4.4 formulates the time consumption reduced by our approach.

4.1 Loss trends of individual data

Figure 2 presents the changes in the value of of the loss function over two representative data in two separate training. The two trainings are carried out with the same network. The loss of data 1 manifests a trend of continuous dropping from the beginning to the end in the first training. In contrast, an increasing trend has been observed with data 2. However, things change in the second run, where both data 1 and data 2 experience the decrease in loss. These two data have similar trends as that of overall network performance in the second run. The results indicate that the individual input data may produce varied performance in different runs of training even on the same network.

At the end of the training, data 2 cannot be correctly allocated to the category that it is supposed to be due to the high loss produced in the first run. However, it still costs the time and computing power to make the model adjustments using data 2 in each epoch. Our approach makes use of the fact that some data consistently produce bad results but still cost the time and resources during the training process.

Based on the above analysis, we proposed a pruning method for the training data. It temporarily prunes some data that have poor performance evaluated at real-time during training. Our experiments show that temporarily pruning the data that performed poorly in recent training rounds makes little changes to the final model.

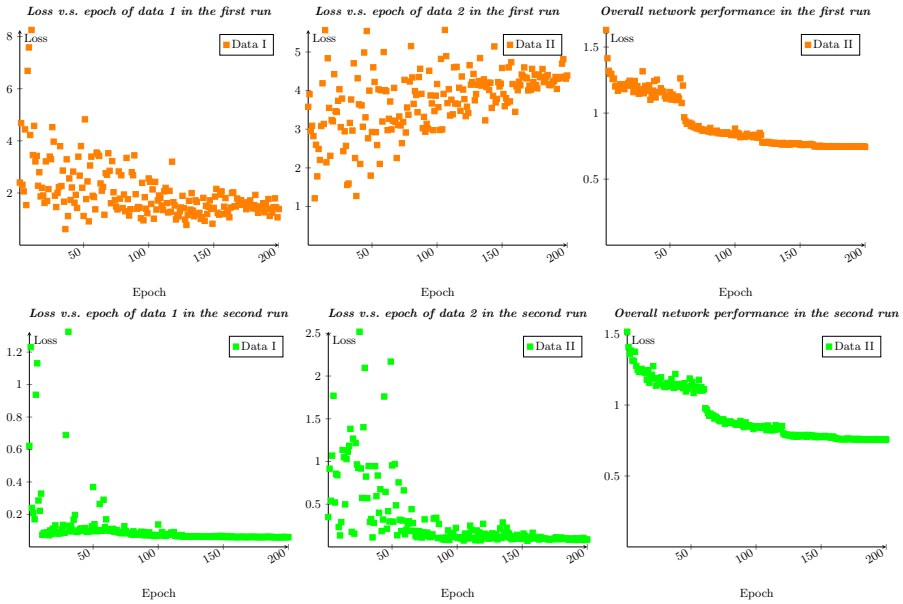


Fig. 2. Performance of Same Data in Same Network but Different Runs

4.2 Loss Monitoring and pruning Selection

Algorithm 1 outlines the loss monitoring procedure and the selection method for data pruning. The loss of each data is monitored during the training process. The *DroppingCount* is defined for each data to measure its training performance and used to decide which data should be selected to prune. A larger *DroppingCount* of data x, denoted by $DroppingCount[x]$ indicates a higher probability for this data item to be to pruned. At the end of training in each epoch, the algorithm examines each loss of the data, denoted by $l(f_{\omega_t}(x), y)$ (where y is the label of data x), within the current batch (line 4). Note that f_{ω_t} denotes the network with parameters of ω at the moment of t. Then the algorithm compares the loss of each data with the loss of the current batch, denoted by $l(f_{\omega_t}(Batch_n))$. The loss value of a batch is the average of all losses in the batch. The algorithm selects the data that perform poorly in this batch and increase their *DroppingCount* values (line 5-7).

Considering that a data item shows the varied behavior through the training stages, the *DroppingCount* of data is held for a window (i.e., a preset number of epochs) and reset at the end of the window (line 1-3). The algorithm judges the behavior of a data item according to the *DroppingCount* value of this data item in the latest window. The size of the window is initialized at the beginning and can be dynamically adjusted during the training. Note that e in the algorithm is the number of current epoch.

Algorithm 1 Loss Monitoring and Pruning Selection

if $e \mod Window == 0$ **then**
  $DroppingCount = EmptyDictionary$
end if
for all (x, y) such that $(x, y) \in BatchData$ **do**
  if $\ell(f_{\omega_t}(x), y) >= \ell(f_{\omega_t}(Batch_n)) * (1 + tolerance)$ **then**
    $DroppingCount[x] += 1$
  end if
end for

4.3 Data fine-pruning

Algorithm 2 Data Fine-pruning during Training

1: **for** $e = 1; e <= Epoch; e++$ **do**
2: **if** $e > PruningWindow$ **then**
3: $StartingPoint = int(StartingPoint/Attenuation)$
4: **end if**
5: **if** $e \mod PruningWindow >= StartingPoint$ **then**
6: **if** $e \mod PruningBlock < PruningCount$ **then**
7: $KeepList = min_{NumData-PruningNum}(DroppingCount)$
8: $(x, y) = (x, y)[KeepList]$
9: **end if**
10: **end if**
11: $\omega_{t+1} = \omega_t - \gamma \frac{1}{b} \sum_{i=1}^{b} \nabla_{\omega_t} \ell(f_{\omega_t}(x_i), y_i)$
12: **end for**

In each window, the algorithm records the data losses and count their corresponding *DroppingCount*. The window size is set according to the changes in learning rate. The windows size is set to a factor of the duration (number of epochs) of the learning rate (e.g., if the duration of the learning rate is 60, the window size is set to be 60 or 30). The analysis is performed for the entire window. However, the data pruning is only performed for the later portion of the window starting from an epoch defined by *StartingPoint*. A fluctuation of the accuracy caused by the adjustment of the learning rate typically lasts for a period and the period becomes shorter as the training progresses. Thus we start to reduce the *StartingPoint* by *Attenuation* after the first window (line 2-4). This measure leads to more pruning rounds so as to further reduce time consumption.

In the later part of each window (line 5), we select some of the epochs to train with the pruned data (line 6), while the original data is still used in other epochs. We only prune the data temporarily because the behaviour of a data varies in different stages of the training process. In the algorithm, *PruningBlock* defines the block of epochs in which the pruned data are used; *PruningCount* is the number of the epochs that performs the data pruning. In each data pruning

epoch, the data are ranked in the decreasing order of *DroppingCount* and the first *PruningNum* number of data in the rank list are pruned in the current epoch (line 7-8). In another word, the data with higher *DroppingCount* values will has more chances to be pruned. *KeepList* stores the indexes of the data that are kept in the training; x is the data index while y is the label of the data. Then the weights of the network are adjusted by BSGD (line 11) at the end of each epoch.

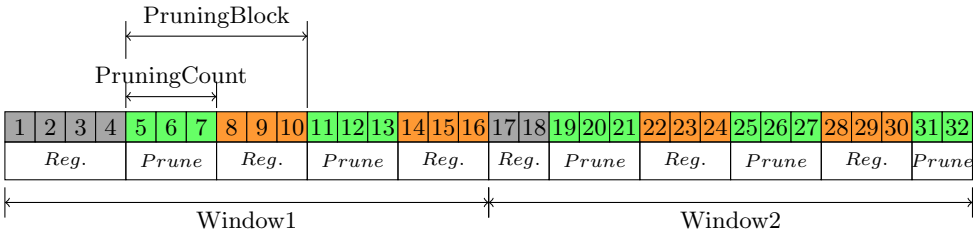


Fig. 3. An example of network training with data fine-pruning method

An exemplar training process using our data fine-pruning method is illustrated in Figure 3. The numbers in the figure is the index of the training epoch. The rounds with gray colour are those running with the regular data before the *StartingPoint*. The two parameters, *PruningBlock* and *PruningCount*, jointly determine the allocation of pruning training and regular training.

4.4 Analysis of performance improvement

The time consumption of the regular training for a network can be formulated by Formula 2. The time of regular running is denoted by $t_{regular}$, the number of epochs by n, the time of forward and backward propagation a batch of data by T, and the number of batches by b. The total time equals to all time consumed over a set of epochs.

$$t_{regular} = \sum_{i=1}^{n} T * b \tag{2}$$

The number of rounds that are trained with the pruned data is denoted by n_{prune}, the size of pruning window by w, the pruning count by c, starting point of pruning by a, the number of iterations using the pruned data by r. According to Algorithm 2, the value of n_{prune} can be obtained by either Formula 3 in the case where the total number of epochs can be divided by the size of the pruning window, or Formula 4 otherwise.

$$n_{prune} = (\left\lfloor \frac{w-a}{r} \right\rfloor * c + (w-a) \bmod r) * \left\lfloor \frac{n}{w} \right\rfloor \tag{3}$$

$$n_{prune} = (\left\lfloor \frac{w-a}{r} \right\rfloor * c + (w-a) \bmod r) * \left\lfloor \frac{n}{w} \right\rfloor + \left\lfloor \frac{n \bmod w - a}{r} \right\rfloor * c + (n \bmod w - a) \bmod r \quad (4)$$

As the number of batches processed in each epoch changes after applying the data-pruning, the average batches over the entire training can be calculated basing on Formula 5.

$$b_{prune} = \frac{1}{n}(\sum_{i=1}^{n_{prune}} T + \sum_{i=1}^{n-n_{prune}} T) \quad (5)$$

t_{prune} denotes the training time with the data pruning approach, t_0 is the computing overhead of the approach, t_{save} is the saved time, which can be obtained by Formula 6 and further by Formula 7.

$$t_{prune} = \sum_{i=1}^{n}(T * b_{prune}) + \sum_{i=1}^{n_{prune}} (t_0) \quad (6)$$

$$t_{save} = t_{regular} - t_{prune} \quad (7)$$

5 Experiments

Our data pruning approach is deployed on several modern neural networks including LeCun network (LeNet), residual network (ResNet), wide residual network (WRN) as well as Vgg network (Vgg). Performance of our method is evaluated with different hyper-parameters and architectures of such networks. Table 1 presents the average value of the best three accuracies of both regular training and data-pruning training as well as the percentage of saved time (*Speedup* in the table). Our experiments are conducted on a workstation with a CPU Intel i7-7700K, a GPU Nvidia GTX 1080 Ti, a hard disk Samsung SSD 970 Pro, four 16GB DDR4 2400Hz memory, Ubuntu 18.04, Cuda 9.0 and cuDNN 7.0.

Table 1 presents the average time consumption of regular training, data fine-pruned training and the percentage of save time (*Speedup*) on four networks: LeNet, VggNet, ResNet and WRN. The data we used in the experiments is a popular benchmark dataset Cifar-10. It can be seen from the table that our data pruning approach can effectively save the training time. Further, higher percentage of time can typically be saved with a larger network. In the best case when WRN-22 is used for training, 14.29% of time is saved. Besides, According to our experiments, the overhead of data pruning approach is very lightweight. It only adds around averaged 4.2 seconds over 200 epochs of the training.

The aim of our data pruning approach is to reduce the training time while maintaining the accuracy. Table 2 compares the accuracy between our approach

Table 1. Time consumption of pruning data training and regular training

	LeNet	VggNet-13	VggNet-16	VggNet-19	ResNet-18
Regular	1220s	2028s	2554s	3110s	1277s
Pruned	1071.73s	1873.67s	2356.74s	2867.7s	1109.92s
Overhead	4.27s	4.33s	4.26s	4.30s	4.08s
Speedup	11.80%	7.40%	7.56%	7.65%	12.76%
	ResNet-34	**ResNet-50**	**WRN-10**	**WRN-16**	**WRN-22**
Regular	2047s	3931s	3h24m	5h8m	6h53m
Pruned	1767.79s	3379.82s	2h56m	4h26m	5h54m
Overhead	4.21s	4.18s	4.12s	4.25s	4.32s
Speedup	13.43%	13.92%	13.73%	13.81%	14.29%

Table 2. Accuracy comparison between pruning data training and regular training

	LeNet	VggNet-13	VggNet-16	VggNet-19	ResNet-18
Regular	75.15%	93.92%	93.79%	93.35%	91.25%
Pruned	74.72%	93.56%	93.44%	93.30%	91.16%
	ResNet-34	**ResNet-50**	**WRN-10**	**WRN-16**	**WRN-22**
Regular	92.86%	93.75%	92.13%	94.22%	95.08%
Pruned	92.85%	93.46%	91.78%	94.20%	94.71%

and regular training. It can be seen from the table that the difference in accuracy is typically less than 0.4% except LeNet with a difference of 0.43%. The reason why LeNet shows the worse accuracy is because of the limitation of the net itself. Comparing LeNet to others, LeNet has a quite small number of layers and parameters, which makes the network more uncertain and unstable. The smallest difference in accuracy observed in our experiments is 0.01% (with ResNet-34).

6 Conclusions and Future Works

Training a deep neural network can be a very time-consuming process. In this paper, we present a data fine-pruning technique, which analyzes the loss of each data at real time and prunes a set of data that performs poorly in recent training epochs. It achieves the noticeable saving of training time while maintaining the accuracy of the results.

There is more work to be done in future. First, our experiments show that some data are commonly identified as *bad* data and are repeatedly selected for pruning. We would like to investigate whether the *bad* data contain the common features. If such common features do exist and can be identified, we can probably make use of the finding and further reduce the training time.

Second, the data that perform poorly may relate to the type of the networks. We would like to conduct more in-depth research regarding the relation between

the *bad* data and the type of networks. If this could be established, we expect to further improve the performance in practice.

Acknowledgement

This work is partially supported by the National Key R&D Program of China 2018YFB1003201 and Guangdong Pre-national Project 2014GKXM054.

References

1. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer (2010)
2. Chilimbi, T.M., Suzue, Y., Apacible, J., Kalyanaraman, K.: Project adam: Building an efficient and scalable deep learning training system. In: OSDI. vol. 14, pp. 571–582 (2014)
3. Cireşan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. arXiv preprint arXiv:1202.2745 (2012)
4. Cui, H., Zhang, H., Ganger, G.R., Gibbons, P.B., Xing, E.P.: Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In: Proceedings of the Eleventh European Conference on Computer Systems. p. 4. ACM (2016)
5. Dahl, G.E., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing **20**(1), 30–42 (2012)
6. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in neural information processing systems. pp. 1223–1231 (2012)
7. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(Jul), 2121–2159 (2011)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)
12. Lei, Y., Scheffer, N., Ferrer, L., McLaren, M.: A novel scheme for speaker recognition using a phonetically-aware deep neural network. In: Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. pp. 1695–1699. IEEE (2014)
13. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: OSDI. vol. 14, pp. 583–598 (2014)
14. Liu, J., Wright, S.J., Ré, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. The Journal of Machine Learning Research **16**(1), 285–322 (2015)

15. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International conference on machine learning. pp. 1928–1937 (2016)
16. Nesterov, Y.: A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In: Doklady AN USSR. vol. 269, pp. 543–547 (1983)
17. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural networks **12**(1), 145–151 (1999)
18. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems. pp. 693–701 (2011)
19. Richardson, F., Reynolds, D., Dehak, N.: Deep neural network approaches to speaker and language recognition. IEEE Signal Processing Letters **22**(10), 1671–1675 (2015)
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
22. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
23. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1701–1708 (2014)
24. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
25. Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.M., Liu, T.Y.: Asynchronous stochastic gradient descent with delay compensation. In: International conference on machine learning (2017)