Simple techniques for improving deep neural network outcomes on commodity hardware

Nicholas Christopher A. Colina, Carlos E. Perez, and Francis N. C. Paraan

Citation: AIP Conference Proceedings **1871**, 040001 (2017); doi: 10.1063/1.4996523
View online: http://dx.doi.org/10.1063/1.4996523
View Table of Contents: http://aip.scitation.org/toc/apc/1871/1
Published by the American Institute of Physics

Simple Techniques for Improving Deep Neural Network Outcomes on Commodity Hardware

Nicholas Christopher A. Colina[1,b)], Carlos E. Perez[c)] and Francis N. C. Paraan[1,a)]

[1]*National Institute of Physics, University of the Philippines Diliman, Quezon City 1101, Philippines*

[a)]Presenting author: fparaan@nip.upd.edu.ph
[b)]ncolina@nip.upd.edu.ph
[c)]ceperez@codeaudit.com

Abstract. We benchmark improvements in the performance of deep neural networks (DNN) on the MNIST data test upon implementing two simple modifications to the algorithm that have little overhead computational cost. First is GPU parallelization on a commodity graphics card, and second is initializing the DNN with random orthogonal weight matrices prior to optimization. Eigenspectra analysis of the weight matrices reveal that the initially orthogonal matrices remain nearly orthogonal after training. The probability distributions from which these orthogonal matrices are drawn are also shown to significantly affect the performance of these deep neural networks.

Introduction

In a deep neural network (DNN) machine learning is achieved by a multidimensional optimization of weight matrices and bias vectors that are constrained by data from a known training set. When applied to a classification task, for example, the optimized DNN maps an input vector into a probability distribution over the set of classes or bins. That is, an input vector is classified as a member of the most likely class.

In this talk, we present typical use cases of two tweaks that can greatly improve the performance of DNNs on a standard handwritten digit classification task. The first modification involves the parallelization of the training algorithm over CPU and GPU cores. In many software packages, DNN optimization under a typically large number of training constraints is often achieved by stochastic gradient descent through backpropagation. This approach is particularly suitable for parallelization because the multidimensional gradients are calculated from local values that come from the current iteration and are typically performed in single-precision. Additionally, only random batches of the training constraints are imposed during the gradient calculations to further minimize memory requirements. Order of magnitude speedups are reported on runs done on consumer-level hardware using open-source GPU-enabled software packages. The benchmarking results for the simple hardware configurations presented here and the continued performance improvements in commodity graphics cards make the case for nascent research groups and teaching laboratories without access to high performance computing facilities to consider investing in them.

The second modification discussed here involves special choices for the (random) initial values for the elements of the weight matrices. Previously, it was demonstrated that using initially orthogonal matrices in DNNs can lead to faster learning, in part, as information is more effectively backpropagated from layer to layer [1]. In the following discussion we present comparative results for linear and nonlinear DNNs that are initialized with four different classes of random matrices: (I) matrices with elements drawn from a uniform distribution, (II) matrices with elements drawn from a Gaussian distribution, (III) random orthogonal matrices generated by QR decomposition, and (IV) random orthogonal matrices generated by singular value decomposition (SVD). We observed that the performance improvements seen for linear DNNs also carry over to the nonlinear case that employs rectified linear unit (ReLU) activation functions. We further analyze the eigenspectra of the optimized weight matrices and find that, although they are no longer orthogonal, their eigenvalues are still concentrated about the unit circle and their Frobenius norms are very close to unity.

Structure, Function and Dynamics from nm to Gm
AIP Conf. Proc. 1871, 040001-1–040001-4; doi: 10.1063/1.4996523
Published by AIP Publishing. 978-0-7354-1551-5/$30.00

Setup

The machine learning software package used in this study is the Python 2.7 Keras framework [2] operating over a Theano backend capable of GPU-parallelization. Runs were performed on two hardware setups corresponding to (1) a typical workstation (Intel E5-2620 @ 2.0 Ghz, 64 GB RAM, Nvidia GTX 1070 GPU card with 1920 CUDA cores @ 1.68 GHz), and (2) a typical desktop (Intel i5-4690 @ 3.5 Ghz, 8 GB RAM, Nvidia GTX 760 GPU card with 1152 CUDA cores @ 980 MHz). Both hardware setups were using the UNIX based Fedora 21 operating system.

Two neural network topologies were used in our tests. The first network consists of six sequential 1000 node hidden layers that constitute five 1000×1000 weight matrices and the second is similar to the first except it makes use of nine hidden layers which comprise eight 1000×1000 matrices. The activation function between the hidden layers of the DNN was varied between an identity (linear) activation function and a nonlinear rectified linear unit (ReLU) activation function. The MNIST data set [3] was used to train the networks for 50 epochs (for the 6 hidden layer models) or 100 epochs (for 9 hidden layer models) using a stochastic gradient descent algorithm with learning rate 10^{-4} and in sample batches of 50 images each. The wall time elapsed in training over 50 epochs in a 6 hidden layer model was used to benchmark the hardware performance during DNN optimization.

Four different sets of initial conditions for the weight matrix elements were used. For class (I) initial conditions, the elements were drawn from a uniform distribution on the interval [0,1). Class (II) initial matrices have elements taken from a Gaussian distribution of zero mean and standard deviation 0.1. For the class (III) random orthogonal Q matrices, a QR decomposition of a class (II) matrix with Gaussian distributed elements was performed (Q is orthogonal and R is upper right triangular). Finally, the class (IV) random orthogonal U matrices were generated by an SVD of a class (II) matrix with Gaussian distributed elements (the SVD of a square matrix $M = UDV^T$ has U and V orthogonal and D diagonal). For all initializations the elements of all bias vectors were initially set to 0.1. The metrics used to evaluate the performance of the differently initialized neural networks are the validation loss (in the form of a cross entropy), the validation accuracy, and the final accuracy of the model. All metrics are measured over the 10,000 images MNIST declares as a test set. Validation loss and validation accuracy are measured during training after each epoch and reflect the ability of the model to generalize the learning to images it has not seen through training. The final accuracy metric is the number of correctly classified images divided by the total number of images used in testing.

Parallelization

Completely parallelizable processes that run solely on a CPU can sometimes be optimized to obtain a nearly linear relationship between speedup and the number of threads and cores. However, speedup due to parallelization on a discrete GPU card is more difficult to model and predict. The interplay between several factors such as data transfer speeds between CPU and GPU memory, GPU threading, complex hardware architecture, code- and driver-level optimizations, and others make it a challenge to predict speedup from knowledge of the number of available GPU cores alone. Thus, many workers have to rely on empirical results from test cases to obtain estimates for the order of magnitude of the speedup one can expect from GPU parallelization. While we cannot test all hardware configurations, we can reasonably expect machines with similar specifications to have improvements of the same order of magnitude.

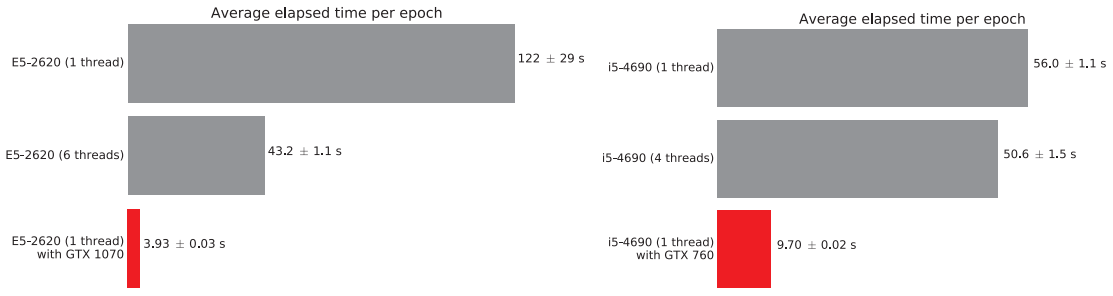

FIGURE 1. Parallelization over CPU threads can improve DNN training times, but this optimization involves repeated operations on matrices of dimensions much larger than the number of CPU cores on a typical computer. Parallelization by commodity graphics cards with thousands of GPU cores yielded significant speedup in the tested workstation ($\sim 30\times$) and desktop ($\sim 6\times$) setups.

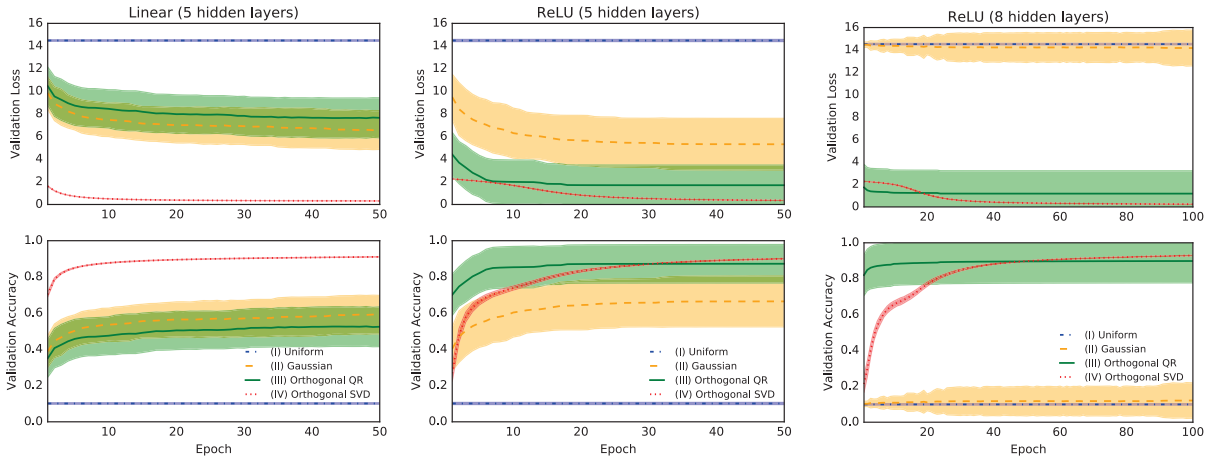

FIGURE 2. DNNs with ReLU activation functions that are initialized with class (IV) random orthogonal weight matrices generated by SVD are more accurate. However, this initialization take more epochs to saturate to a learned state compared to the other initializations considered in this study. Error bands represent one standard deviation in over 30 models with different random initial conditions.

We tested two parallelization schemes in this study. The first approach involves GPU parallelization on a single GPU device and the second uses threading over CPU cores by OpenMP. The average wall times needed for each parallelized algorithm to complete a training epoch are shown in Figure 1. We find that speedups of up to $6\times$ to $30\times$ can be expected in similarly configured training runs using mid-level commodity GPU devices. The order of magnitude of this speedup is significant because current multicore desktop computers only have ~ 10 cores available for CPU parallelization. As observed in our CPU threaded runs, only moderate (sublinear) speedup was achieved by the multithreaded approach. The speedup due to the parallelization over CPU threads is less than ideal due to the fact that the full training algorithm is not completely parallelizable. Furthermore, additional deviations from the ideal linear scaling behavior can be attributed to hardware bottlenecks in threadsafe applications, such as limited sequential memory I/O speeds. If only one GPU device is available, this type of bottleneck can be mitigated by using a single CPU thread along with the GPU card. In this case, parallelizable operations are solely processed by dedicated GPU hardware that has been engineered to perform a large number of identical simple tasks with less device overhead (for example, GPU memory I/O speeds are typically higher than CPU I/O speeds).

Weight matrix initialization

For each model, initialization, and activation function tested, the average validation loss and average validation accuracy versus epoch is plotted in Figure 2. Significantly improved performance was observed for DNNs that have been initialized with class (IV) orthogonal matrices generated by singular value decomposition regardless of model depth or activation function used. The final accuracy of the class (IV) matrix initialization is also significantly higher than the other weight matrix initialization classes for each of the activation functions used in this study (Table 1). Addi-

TABLE 1. Average final test set accuracy over 30 instances of the model with different random initial conditions.

	Linear (5 hidden layers)	ReLU (5 hidden layers)	ReLU (8 hidden layers)
(I) Uniformly distributed elements	(10.0 ± 0.6)%	(10.0 ± 0.5)%	(9.92 ± 0.44) %
(II) Gaussian distributed elements	(59 ± 10)%	(67 ± 16)%	(12.08 ± 9.86)%
(III) Random orthogonal QR matrices	(52 ± 11)%	(86.5 ± 9.5)%	(89.85 ± 12.19) %
(IV) Random orthogonal SVD matrices	(91.1 ± 0.1)%	(90.1 ± 0.17)%	(92.93 ± 0.12) %

TABLE 2. Average Frobenius norm over square weight matrices of 30 trained instances of each topology with different random initial conditions.

	Linear (5 hidden layers)	ReLU (5 hidden layers)	ReLU (8 hidden layers)
(III) Random orthogonal QR matrices	1.004 ± 0.022	1.000 ± 0.022	1.002 ± 0.023
(IV) Random orthogonal SVD matrices	1.000 ± 0.023	1.002 ± 0.024	1.004 ± 0.022

tionally, this type of initialization yielded more reproducible results in the sense that much less variance was observed in validation loss and prediction accuracy throughout training.

Analysis of the eigenspectra of the optimized weight matrices show only small deviations from the eigenspectra of the initial random orthogonal matrices. In order to quantify this deviation, we used the Frobenius norm

$$\|A\|_f = \sqrt{\mathrm{Tr}\,(AA^\dagger)}. \tag{1}$$

For an orthogonal matrix the Frobenius norm is exactly 1 and therefore departures from this value can reveal how far the initially orthogonal matrices deviate from orthogonality after optimization. Table 2 shows the average value of the Frobenius norm over all the square weight matrices across 30 trained models with different random initial conditions. It is observed that even after optimization the weight matrices are nearly orthogonal. This result suggests that a subspace of orthogonal matrices is somehow naturally close to the solution space of the DNN optimization problem.

Another notable observation is the clear difference in performance of the trained DNNs initialized by class (III) and class (IV) matrices, even though both optimizations begin with random orthogonal matrices. The much smaller variance in test set accuracy observed for the SVD orthogonal matrices over the QR orthogonal matrices (Table tab:final-acc) implies more consistent DNN training performance of the former. Thus, in addition to considering the orthogonality of the random initial weight matrices, it is crucial in DNN design to also consider the probability distribution from which these orthogonal weight matrices are taken.

Concluding Remarks

Two modifications to the training of a DNN model, GPU parallelization and orthogonal weight matrix initialization, were presented as tools that can be used to reduce DNN training times and improve final accuracy. These modifications were tested on a 6×1000 dimension layer deep neural network with either linear or ReLU activation functions. Using a commodity GPU device for parallelization can reduce DNN training times by a factor of $\sim 30\times$. Initializing weight matrices using random orthogonal matrices showed improvements in training and final performance over the non-orthogonal initializations tested regardless of the activation function used or model depth. The method by which these orthogonal matrices are generated has also been shown to affect the performance of trained DNNs, with singular value decomposition yielding more accurate DNNs than QR decomposition.

Acknowledgments

NCAC is supported by a DOST-SEI Scholarship. FNCP acknowledges support from the Research Center for Theoretical Physics in Jagna, Bohol and the University of the Philippines Diliman OC and OVCRD through Project No. 161614 PhDIA.

REFERENCES

[1] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *Proceedings of the International Conference on Learning Representations*, edited by Y. Bengio and Y. LeCun (Banff, Canada, 2014), arXiv:1312.6120.

[2] F. Chollet, Keras, Source code available from https://github.com/fchollet/keras. Retrieved March 2017.

[3] Y. LeCun, C. Cortes, and C. J. C. Burges, The MNIST database of handwritten digits, Data set available from http://yann.lecun.com/exdb/mnist/. Retrieved March 2017.