Supplementary Document for "On Optimization Methods for Deep Learning"

Quoc V. Le QUOCLE@CS.STANFORD.EDU
Jiquan Ngiam JNGIAM@CS.STANFORD.EDU
Adam Coates ACOATES@CS.STANFORD.EDU
Abhik Lahiri ALAHIRI@CS.STANFORD.EDU
Bobby Prochnow PROCHNOW@CS.STANFORD.EDU
Andrew Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

1. Introduction

In our ICML paper titled "On optimization methods for deep learning", we discussed the standard and sparse autoencoder model. However, due to space limitations in our paper, we were not able to present further details about the bases learned by the sparse autoencoder model, compare the standard autoencoder with the Hessian Free approach as described in (Martens, 2010) and analyze in detail the effects of GPU on the different optimization methods. We clarify these details in this supplementary document. All the experiments described in this document have been carried out on the MNIST data set.

2. Visualization of the features learned in the hidden layer

In (Lee et al., 2008), the authors present a sparse RBM model that learns features from the MNIST dataset. However, to training the weights for each hidden layer,[1] they update only the bias weights while taking the gradient of the sparsity term at every iteration. According to the authors, they follow this approach because using stochastic gradient descent or minibatches to estimate the sparsity results in biased estimates of the gradient. But, this renders their learning rule incorrect. So, here we investigate the kind of bases learned by our model, when we take the correct gradient of our objective function (including the sparsity term).

In this section, we also present an analysis of the features learned by the model as the number of hidden neurons is varied. We discuss the quality of features extracted with models learning undercomplete, overcomplete and highly overcomplete features.

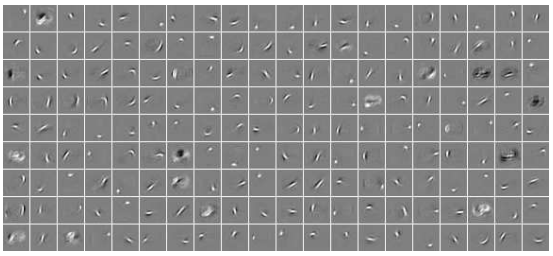
In Figure 1 below, we show the visualization of the features learned by our model in the three cases.[2] We can see that our simple model can learn penstroke features on the MNIST dataset. This is consistent with the results obtained in (Lee et al., 2008). From the figure, we can also observe that as the number of hidden neurons are increased, it becomes more difficult to learn good features. In the case of overcomplete and highly overcomplete models, the learned features tend to be more like complete images from the data set.

3. Comparison of our methods with the Hessian Free approach

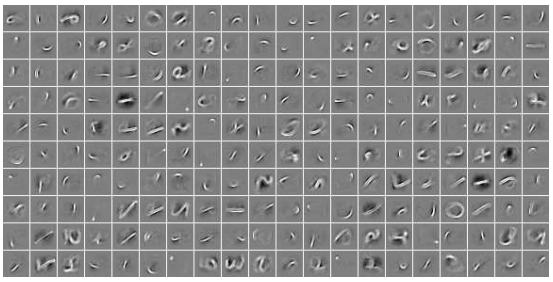
In (Martens, 2010), the author presents a Hessian Free optimization approach that works well for deep autoencoders. It is interesting to know whether this approach works faster than L-BFGS or CG on pretraining shallow networks. So, we performed an experiment to compare our proposed method of using minibatch L-BFGS/CG on GPU against the minibatch Hessian Free method on GPU. We used a standard autoencoder model (i.e., a sparse autoencoder with $\gamma = 0$) with 10000 hidden units, a weight regularization parameter (i.e., λ) value of 0.0001 and a minibatch size of 10000 images. For all three methods (L-BFGS, CG and Hessian Free), we tuned the paramters of the model to yield the best result on the data set. The result we obtained is shown in Figure 2. From the optimization curves in this figure, we notice that on our simple standard autoencoder model, the off-the-

[1] They follow a greedy layerwise training approach

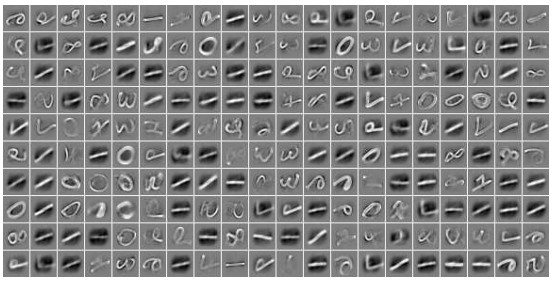
[2] In our experiments, the model with undercomplete features had 196 hidden neurons, the one with overcomplete features had 1000 hidden neurons and the one with highly overcomplete features had 10000 hidden neurons



(a) model with 196 features



(b) model with 1000 features



(c) model with 10000 features

Figure 1. Visualization of the bases learned by the three sparse autoencoder models with undercomplete, overcomplete and highly overcomplete features respectively.

shelf methods discussed in our paper perform better than the Hessian Free approach proposed by James Martens. With the shallow network we considered in our paper, various subroutines for CG backtracking and calculating the Hd vector in the implementation of the Hessian Free optimization algorithm seem to be an overhead.

4. More details on the effects of GPU on different optimization methods

In section 4.5 of our ICML paper, we reported results on an experiment done to analyze the gain in switching from CPUs to GPUs for the different optimization

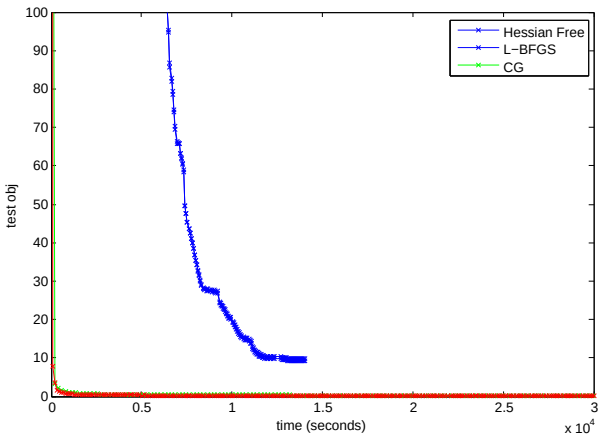


Figure 2. Optimization curves obtained for a standard autoencoder model with 10000 hidden units using L-BFGS large minibatch, CG large minibatch and the Hessian Free method

methods on a standard autoencoder model with one hidden layer. We reported higher gain for L-BFGS and CG compared to SGDs, but in general the speedup obtained was little for all the three optimization methods (i.e., minibatch L-BFGS, CG and SGDs). In this section, we would like to analyze further why this is the case.

Specifically, in the experiment reported in the ICML paper the CPU runs used an 8 core machine as all the experiments performed for the paper were carried out on a cluster. Moreover, the GPU runs involved transfer of minibatch and parameters in and out of the GPU after each epoch. In this section, we include optimization curves for runs performed on a 1 core machine. We also try to see whether we can leverage the GPU better if the GPU runs are optimized further by having the complete training dataset and parameters transferred to the GPU in the beginning of the optimization so as to eliminate data transfer delays. We report the results of our experiment in Figure 3 and Figure 4.[3] For each optimization method, all parameters were set so as to give the best result. From Figure 3 and Figure 4, we can see that an efficient implementation of a GPU run achieves significant gain in speed compared to a CPU run on a 1 core machine for all three optimization methods. However, the gain in switching

[3]Parts (a), (b) and (c) of Figure 4 are the same plots as Figure 3, but zoomed into the regions of the L-BFGS, CG and SGDs curves respectively so that the speedup obtained is visible. Also, the minibatch size used for is 1000 images for L-BFGS, 100 images for CG and 10 images for SGDs.

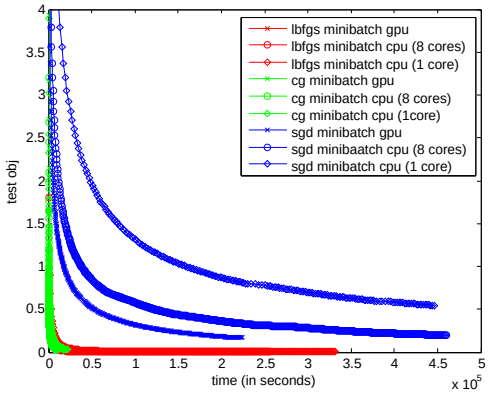


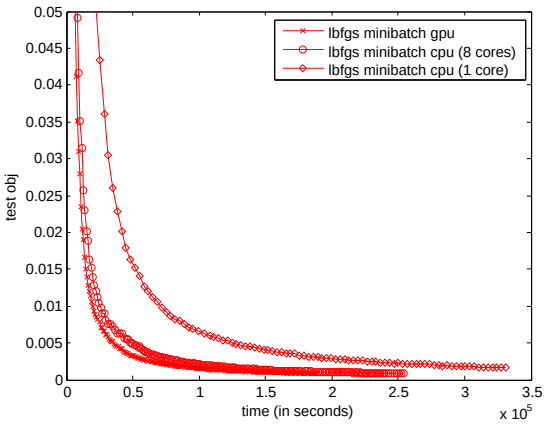
Figure 3. Optimization curves for the experiment to investigate the effect of GPU on the three different methods

from an 8 core CPU to GPU is not much. Also, if the optimization parameters are carefully tuned, then we observe similar gain in switching from CPU to GPU for all three methods.

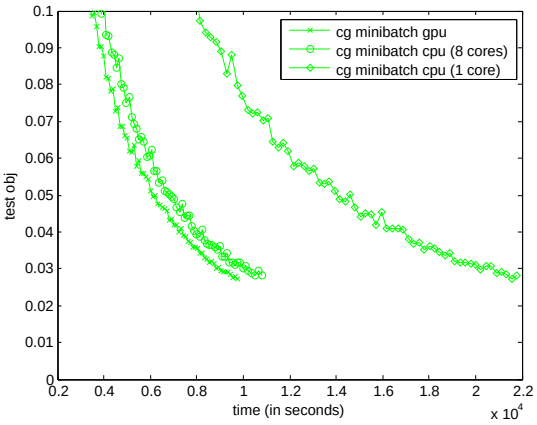
References

Lee, H., Ekanadham, C., and Ng, A.Y. Sparse deep belief net model for visual area v2. In *NIPS*, 2008.

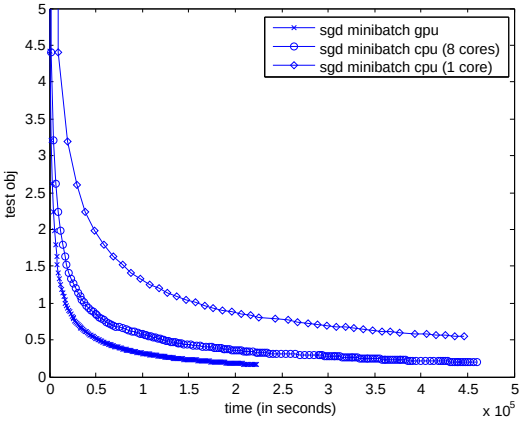
Martens, J. Deep learning via hessian-free optimization. In *ICML*, 2010.



(a) optimization curves for L-BFGS runs



(b) optimization curves for CG runs



(c) optimization curves for SGDs runs

Figure 4. Plots to show the effect of GPU on each optimization method