Investigations on Hessian-Free Optimization for Cross-Entropy Training of Deep Neural Networks

Simon Wiesler[1*], *Jinyu Li*[2], *Jian Xue*[2]

[1]Human Language Technology and Pattern Recognition
Computer Science Department, RWTH Aachen University, 52056 Aachen, Germany
[2]Microsoft Corporation, Redmond, WA, 98052, USA
wiesler@cs.rwth-aachen.de, {jinyli,xue}@microsoft.com

Abstract

Context-dependent deep neural network HMMs have been shown to achieve recognition accuracy superior to Gaussian mixture models in a number of recent works. Typically, neural networks are optimized with stochastic gradient descent. On large datasets, stochastic gradient descent improves quickly during the beginning of the optimization. But since it does not make use of second order information, its asymptotic convergence behavior is slow. In regions with pathological curvature, stochastic gradient descent may almost stagnate and thereby falsely indicate convergence. Another drawback of stochastic gradient descent is that it can only be parallelized within mini-batches. The Hessian-free algorithm is a second order batch optimization algorithm that does not suffer from these problems. In a recent work, Hessian-free optimization has been applied to a training of deep neural networks according to a sequence criterion. In that work, improvements in accuracy and training time have been reported. In this paper, we analyze the properties of the Hessian-free optimization algorithm and investigate whether it is suited for cross-entropy training of deep neural networks as well.

Index Terms: deep learning, optimization, Hessian-free

1. Introduction

Recently, there has been a resurgence in the use of neural networks for acoustic modeling in automatic speech recognition (ASR). In particular, context-dependent deep neural network hidden Markov models (CD-DNN-HMMs), which directly model the tied context-dependent states of HMM speech recognizers have been shown to work very well [1, 2, 3, 4].

The standard training algorithm for deep neural networks (DNNs) is stochastic gradient descent (SGD). Typically, the stochastic gradient is computed on mini-batches. On large and redundant datasets, SGD quickly reaches a region with acceptable performance. Further, the computation within one mini-batch can be parallelized well on GPUs. Still, the use of SGD is not fully satisfactory for two reasons. First, with typical mini-batch sizes, the computation can not be parallelized efficiently across a large number of devices. Second, the training of DNNs is a very difficult and highly non-convex optimization problem. Therefore, the quality of the model strongly depends on the choice of the optimization algorithm. In regions with pathological curvature, SGD converges very slowly or may even fail to improve the training objective further. Therefore, better error rates may be achieved by improving the optimization. Both of these issues are especially important for large-scale learning tasks as speech recognition.

In contrast to SGD, batch algorithms can be parallelized straight-forwardly. Further, most batch algorithms efficiently exploit second-order information, preventing them from stagnating in regions with pathological curvature. The most widely used batch algorithm for large-scale optimization is LBFGS [5], which builds up a low-rank second order model from a limited history of the previous gradients. In a recent publication [6], LBFGS has been applied to a large-scale training of DNNs.

The Hessian-free (HF) algorithm [7] is a second-order batch algorithm which has been specifically designed for the training of deep neural networks. The advantage of HF over LBFGS is that it directly uses a full second-order model. In a recent work [8], Kingsbury et al. applied the HF algorithm to a training of a DNN acoustic model according to a *sequence criterion*. In comparison to SGD, Kingsbury et al. reported a speedup by a factor of 5.5 by parallelizing HF. In addition, they achieved a relative word error rate (WER) reduction by roughly four percent.

It is a natural question, whether these advantages of HF over SGD hold for the conventional cross-entropy (CE) training of DNNs as well. Martens [7] considered only small-scale tasks. Further, he did not compare HF to SGD. In Kingsbury's paper [8], HF has only been applied to sequence training, which differs strongly from CE training, because it is initialized with a very good model that is obtained from CE training. The goal of this work is to study the properties of the HF algorithm and to investigate whether it is beneficial for *large-scale cross-entropy* training of DNNs.

2. The Hessian-Free Algorithm

In this section, we present the HF algorithm and discuss the implications of Martens's specific design choices and its relation to other algorithms in optimization literature.

2.1. Newton-CG method

The starting point for all second order algorithms is *Newton's method*. Let $W \in \mathbb{R}^D$ denote the parameters of the neural network, $F(W)$ the training criterion. Newton's method is based on a quadratic approximation of F around W:

$$m(P) = \frac{1}{2} P^T \mathbf{B} P + \nabla F(W)^T P + F(W) \,. \qquad (1)$$

*The author performed the work at Microsoft.

The matrix $\mathbf{B}$ is chosen as the Hessian matrix $\mathbf{H}(W)$ of F at W. For positive definite $\mathbf{B}$, the Newton step P_N is defined as the optimum of the quadratic approximation

$$P_N := \operatorname*{argmin}_P m(P) \ , \tag{2}$$

which is the solution of the linear system

$$\mathbf{H}(W)P_N = -\nabla F(W) \ . \tag{3}$$

In every iteration of Newton's method, the model is updated by P_N. Newton's method converges rapidly in terms of iterations. For high-dimensional problems as the training of DNNs, Newton's method is prohibitive, because the number of parameters of the Hessian matrix is in the order of $\mathcal{O}(D^2)$.

A well-known modification of Newton's method for high-dimensional problems is the *Newton-CG method*, see *e.g.* [9, Chapter 6]. In the Newton-CG method, the linear system (3) is solved with the iterative conjugate gradient (CG) algorithm [10]. The CG algorithm only requires products of $\mathbf{B}$ and an arbitrary vector, but does not require $\mathbf{B}$ itself. For problems, where an algorithm exists which computes these matrix-vector products efficiently, Newton-CG can be applied.

2.2. Martens's Algorithm

Martens modified the Newton-CG method in several ways in order to apply it to the optimization of DNNs. In the following, we discuss these modifications.

The most important question for the application of the Newton-CG algorithm is how to compute the required matrix-vector products. For neural networks, Hessian-vector products can be computed by a forward-backward pass. In neural network literature, this is known as the *Pearlmutter trick* [11].

Since the Hessian matrix of the CE criterion is not positive definite, Martens decided to use the positive semi-definite *Gauss-Newton* (GN) matrix instead. The GN matrix has originally been defined for the squared error criterion, but can be generalized to the CE criterion [12]. GN-matrix-vector products can be calculated by a forward-backward-pass as well [12].

The per-sample cost of a GN-matrix-vector product is roughly twice the cost of a gradient calculation. For large-scale tasks, calculating GN-matrix-vector products on the complete training data, is highly expensive. A key idea of Martens is to use a *stochastic approximation of the GN matrix*, *i.e.,* to compute matrix-vector products on mini-batches. In contrast, gradients and objective function values are computed exactly.

The quadratic approximation (1) is only reliable in a neighborhood around W. Therefore, the Newton-CG method must be prevented from taking too large steps. This is in particular important when using a stochastic approximation of $\mathbf{B}$. In optimization literature, it is suggested either to perform a *line-search* in the direction P, or, to assume a *trust region* (TR) around W. In TR algorithms, the minimization (2) is replaced by the constrained minimization

$$P_N = \min_P m(P) \quad \text{such that} \quad \|P\| \leq \Delta \ . \tag{4}$$

The trust region radius Δ is dynamically adapted based on the progress of the algorithm. Problem (4) can be solved with matrix factorizations [13], but these are prohibitive for high-dimensional problems. A widely used cheap approximation of (4) is given by Steihaug's method [14], which simply terminates CG when the trust region is left. Martens found the well-understood Steihaug method not to be effective for DNN training and therefore uses a *damping heuristic* instead. This means that the matrix $\mathbf{B}$ in (1) is replaced by $\mathbf{B} + \lambda\mathbf{I}$ with a damping parameter $\lambda > 0$. Here, $\mathbf{I}$ denotes the identity matrix. Similar to TR algorithms, the damping parameter λ is adapted based on the ratio of the actual and the expected improvement

$$\rho = (F(W) - F(W + P))/(m(0) - m(P)) \ . \tag{5}$$

For large values of ρ, the damping parameter is decreased, which results in larger steps. For small ρ, the damping parameter is increased. In principle, a damped GN step is equivalent to a trust region GN step. The heuristic lies in the update of the damping parameter, because it is not related to a trust region radius. Furthermore, the update of the damping parameter is only consistent if CG is run until convergence and ρ is evaluated with the final CG iterate. The advantage of the damping approach is that it is cheaper to solve for the damped update than solving (4), and it avoids the approximation of Steihaug's algorithm.

An important property of HF is that CG is initialized with the solution of the previous CG run multiplied with a constant scalar. This is in contrast to most other Newton-CG implementations, where CG is initialized with zero. Martens reported that this initialization speeds up HF by "an order of magnitude"[7]. Indeed, a suitable initialization of CG may save CG iterations. On the other hand, a non-zero initialization may have a strongly negative effect when a hard-limit on the number of CG iterations is set. In addition, Steihaug's algorithm and related approaches require a zero initialization.

Another possibility for speeding up CG is the use of a preconditioner. Martens suggests the use of a diagonal preconditioner based on the accumulated squared gradient. Accumulating this preconditioner causes additional computations and memory usage, but can save CG iterations.

Finally, Martens introduced a CG backtracking. This means that some intermediate results of the CG run are stored. After the termination of CG, a backtracking along the intermediate results is performed. CG backtracking is helpful if there is a strong mismatch between the quadratic model m and the objective function F. In this case, the intermediate CG results can lead to much better updates than the final CG result. However, the backtracking procedure causes an objective function evaluation for every tested model and is therefore highly expensive. It would be preferable to avoid CG backtracking by improving the damping or using a trust region method.

When using CG backtracking, it is not clear which model should be used for calculating ρ in Equation (5). Martens uses the best model obtained from CG backtracking [7]. However, this counteracts the damping approach. It means that the damping factor may be decreased although the solution of the CG algorithm corresponds to a much too large step. This is undesirable, because it causes many backtrackings in the subsequent steps. Furthermore, the linear system (3) gets ill-conditioned with small damping parameters and therefore CG converges very slowly. In Section 3, we compare Martens's variant with using the final CG iterate for updating ρ.

3. Empirical Analysis of the HF algorithm

In this section, we empirically analyze the consequences of Martens's design choices and the performance of HF in a large-scale setting. In order to be able to compare our results to Martens's results [7], we performed experiments on MNIST, a small-scale handwritten digit recognition task. Additional experiments are performed on a short-message dictation task.

The implementation of the algorithm has been realized in the Microsoft DNN training tool and makes use of GPU accelera-

Algorithm	Preconditioning	Damping	CG initialization	1st epoch with 100% acc	CG iterations
Gradient descent	-	-	zero	(40.2% after 500 ep.)	-
Steihaug	-	-	zero	(99.9% after 500 ep.)	2810
Hessian-free	no	backtracking iterate	zero	237	55532
Hessian-free	no	final iterate	zero	79	2308
Hessian-free	no	backtracking iterate	previous solution	56	1443
Hessian-free	no	final iterate	previous solution	60	1377
Hessian-free	squared gradient	backtracking iterate	previous solution	63	1505
Hessian-free	squared gradient	final iterate	previous solution	56	1458

Table 1: Results on MNIST. The third column lists the model that is used for calculating ρ in (5). The last column lists the total number of CG iterations until 100% training accuracy has been achieved.

tion. As the conventional SGD training [2], all computations are performed on the GPU without synchronization to the main memory. The computation of the gradients and likelihoods can be parallelized to multiple GPUs or a large number of CPUs straight-forwardly, but we simply used one GPU in this work.

3.1. Handwritten Digit Recognition

The task of the MNIST dataset is the classification of images of handwritten digits. The dataset consists of $60,000$ training samples and $10,000$ test samples. The features are the gray values of the images with a resolution of 28×28 pixels. We used DNNs with the same structure as in [7]: four hidden layers with dimensions 1000, 500, 250, and 30.

We trained a baseline system with SGD with a momentum term. It turns out that with this network architecture, zero training error can be achieved after 420 epochs. By using early stopping, we obtained similar results on the test set with all algorithms that can achieve zero classification error on the training data. The goal of this investigation is to improve the optimizer. Therefore, we focus on the accuracy on the training data. As a measure of convergence speed, we compare the number of epochs that are required for separating the training data. For all experiments, we set the maximal number of epochs to 500. For HF, we used the recommended settings of [7], in particular we used a maximum of 250 CG iterations and ten percent of the training data for the computation of matrix-vector products.

In a first test, we compared the batch algorithm gradient descent with a basic HF implementation, in order to verify that the second order information accelerates the optimization. The basic HF implementation does not use preconditioning and always initializes CG with zero. The results in Table 1 show that batch gradient descent is not useful for optimizing deep networks. Even after 500 epochs, the training accuracy is only 40.2 percent. In contrast, the basic HF implementation separates the training data after 237 epochs.

In addition, we tested Steihaug's method. As Martens, we observed that it is not effective for optimizing deep networks. The reason is that the CG iterates move out of the trust region after only very few iterations. Therefore, Steihaug's method does not fully exploit the second order information.

The computation time per epoch of the basic HF implementation is very high, because a large number of CG iterations and CG backtrackings is performed. The performance of HF is greatly improved by using the *final CG iterate* for the damping parameter update. This reduces the number of epochs to 79 and the number of CG iterations from 55532 to 2308. This result clearly shows that the correct update of the damping parameter is crucial and our proposed update performs best.

The initialization of CG with the previous solution has two effects. First, the non-zero initialization reduces the number of CG iterations per epoch. For the variant where the backtracking iterate is used for calculating ρ, the damping parameters are very small. Therefore, the resulting linear problems are poorly conditioned and the number of CG iterations per epoch is strongly decreased by a factor of 10.2. When the final iterate is chosen for updating ρ, the number of CG iterations is only reduced by a factor of 1.3. Secondly, the number of epochs is reduced. The intermediate results which are used for CG backtracking are between the initialization and the exact solution of the linear system. Including the previous solution in the search direction weakens the effect of unrepresentative mini-batches, similar to the momentum term in stochastic algorithms.

Interestingly, we could not observe a decrease in the number of CG iterations by using preconditioning. Furthermore, the preconditioner proposed in [7] requires the accumulation of the squared gradient, which causes additional computations and memory usage. From these results, we can not recommend the use of the squared-gradient preconditioner for HF.

An obvious way for accelerating HF is to compute the likelihoods for CG backtracking and the damping parameter update on a small subset of the training data or as in [8] on the validation set. However, we found that this approach does not work at all when training models from scratch. The reason is that small improvements on the training data do not necessarily carry over to the smaller dataset. This causes the algorithm to increase the damping parameter repeatedly and thereby testing smaller and smaller steps. Thus, the algorithm stagnates although it has not reached an optimum. This problem did not occur in [8], because there a very good initialization has been used.

In comparison to SGD, HF requires less epochs until convergence. Since the computation time of HF per epoch is much higher, the total computation time of SGD and HF is comparable. On large-scale tasks, HF can be accelerated by parallelization. On the other hand, the stochastic approximation of SGD is much more beneficial on large-scale task.

In terms of test set accuracy, HF does not perform better than SGD, because underfitting is not an issue on this task. This situation is different on large-scale tasks.

3.2. Short Message Dictation

In addition to the small-scale experiments on MNIST, we performed experiments on SMD_R3, a short message dictation task. The training set consists of 63.4 hours audio data. The dev and test set have 16028 and 22809 running words.

The baseline acoustic model is a DNN with five hidden layers, each with 1024 nodes. The input to the network is a 572-dimensional vector obtained by concatenating MFCC features with up to third-order derivatives in a window of eleven frames.

Model	Algorithm	Epoch	Acc. (%)	WER (%)
5×1024	SGD	25	66.8	23.8
5×1024	SGD	+25	67.7	**23.4**
5×1024	SGD	+75	69.3	23.7
5×1024	HF	+25	72.9	23.4
5×1024	HF	+45	75.2	**23.3**
5×1024	HF	+75	77.0	23.8
5×512	SGD	10	62.1	26.1
5×512	SGD	+40	63.4	**25.5**
5×512	HF	+10	64.7	25.2
5×512	HF	+40	66.2	**24.3**
4×1024	SGD	10	65.8	24.5
4×1024	SGD	+40	67.3	**24.2**
4×1024	HF	+10	69.5	23.7
4×1024	HF	+30	71.8	**23.4**
4×1024	HF	+40	72.5	23.8

Table 2: Results on SMD_R3. In the third column, a plus denotes the number of additional epochs that are performed on top of the SGD initialization. The fourth column lists the frame accuracy on the training data, and the fifth column the word error rate on the development set.

The DNN is initialized with a layerwise RBM pretraining. The finetuning is performed with SGD with a momentum term. The initial learning rate of SGD is set to 0.08 and later reduced to 0.002. After 25 epochs, SGD seems to stagnate with a frame accuracy of 66.8% on the training data. The WER of the corresponding model on the development set is 23.8%, see the results with a 5×1024-model in Table 2. We used the best configuration of HF from the MNIST experiments, *i.e.,* ρ is updated with the final CG iterate, CG is initialized with the previous solution, and no preconditioning is used. The larger size of this dataset allowed for using only one percent of the training data for computing the matrix-vector products.

In an initial experiment, we tested whether HF is suited for replacing SGD on ASR tasks. It turned out that HF converges orders of magnitude slower than SGD. On large-scale tasks, it did not seem reasonable to train DNNs directly from pretraining with HF, even with the use of parallelization, Therefore, we did not investigate this possibility further.

In the following experiments, we tried to improve the SGD result with HF. Surprisingly, it turned out that the SGD parameters were not in the neighborhood of an optimum at all. After additional 75 HF iterations, the training accuracy increased from 66.8% to 77.0%. Knowing the HF result, a natural question is whether the training accuracy can be improved with SGD as well. After additional 75 SGD epochs, the training accuracy slowly increased to 69.3%, but it was not possible to achieve the training accuracy of HF with limited computation time.

Unfortunately, these large gains on the training set did not carry over to the development set. The best HF result on the development set which we achieved was 23.3% WER, corresponding to an HF model with 75.2% training accuracy. This is only a tiny improvement of 0.1% over the SGD result. With increased accuracy on the training set, the error rate on the development set increases, *i.e.,* the model overfits. The behavior on the test set is the same as on the development set. The SGD result of 22.9% WER is improved by 0.1% WER as well. When comparing the results of the best HF model and the best SGD model, it is noticeable that the training accuracy of the HF model is higher although the word error rate is almost the same.

In the following experiments, we investigated whether improvements can be obtained with HF optimization on smaller models, where overfitting is less an issue. This would be of great practical value, because reducing the decoding costs of hybrid DNN speech recognizers is one of the biggest challenges for their large-scale application. In these experiments, we initialized HF with the result of SGD after ten epochs in order to reduce computation time. Using only half of the nodes per layer, the best SGD result is 25.5% WER, 2.1% worse than with the large model. This gap is reduced to 0.9% when HF is applied. With such a deep but narrow model, overfitting does not occur, but some accuracy is sacrificed. Another experiment is on a DNN with only four layers of 1024 nodes. Here, the best SGD result is 24.2% WER. Interestingly, the best WER with HF is 23.4%. This means, we can achieve the same error rate with a four-layer network as with the five-layer network baseline.

4. Discussion

In this work, we empirically analyzed the properties of the HF algorithm, a second order batch optimization algorithm which has been proposed in [7]. Some techniques that are used in the algorithm are rather heuristic and their effect is not well understood. In particular, we observed in experiments on MNIST that the damping heuristic has a critical impact on the performance of the algorithm. In our opinion, it is highly desirable to replace this damping heuristic with a procedure that is justified by theory. Furthermore, the experiments show that preconditioner proposed in the original HF paper [7] is not effective.

In speech recognition experiments, we observed that HF is not efficient for training DNNs from scratch in large-scale applications. But when starting from a reasonable initialization, making use of second order information is highly beneficial. Since HF can be parallelized well, it seems most attractive to perform only few epochs with SGD, and then continue with HF. We could obtain strong improvements in terms of training accuracy using HF. However, these improvements did not carry over to the test set due to overfitting. This behavior may be different on larger datasets. In future work, we plan to address the overfitting problem more directly by using regularization or by including a sparsity constraint in the optimization algorithm.

Another conclusion that can be taken from the experiments is about local optima in the context of neural network learning. When SGD stagnates, it is often argued that a local optimum has been reached. However, our experiments show that the slow asymptotic convergence behavior of SGD can falsely indicate convergence. The objective function can still be improved strongly by continuing the optimization with a second order algorithm. Note that Martens argued in this direction too when analyzing the role of pretraining [7].

Using HF optimization, we could obtain the same performance with a four-layer model as with a five-layer model. This result is very interesting, because reducing the decoding costs of DNN speech recognizers is currently a big challenge. Also, the result suggests that less deep architectures are possible if a more sophisticated optimization than SGD is used. It certainly requires further research in order to investigate this suggestion.

5. Acknowledgments

The first author would like to thank Frank Seide and Dong Yu for their help with the DNN software and Yifan Gong for the supervision of the internship.

6. References

[1] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[2] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011, pp. 437–440.

[3] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Proceedings of Interspeech 2012*, 2012.

[4] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A.-R. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 30–35.

[5] D. Liu and J. Nocedal, "On the Limited Memory BFGS Method for Large-Scale Optimization," *Math. Program.*, vol. 45, no. 1, pp. 503–528, 1989.

[6] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012.

[7] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, vol. 951, 2010, p. 2010.

[8] B. Kingsbury, "Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization," in *Interspeech*, 2012.

[9] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 1999.

[10] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[11] B. Pearlmutter, "Fast exact multiplication by the hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[12] N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[13] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 3, pp. 553–572, 1983.

[14] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM Journal on Numerical Analysis*, vol. 20, no. 3, pp. 626–637, 1983.