
Improved Preconditioner for Hessian Free Optimization

Olivier Chapelle
Yahoo! Labs
Santa Clara, CA
chap@yahoo-inc.com

Dumitru Erhan
Yahoo! Labs
Sunnyvale, CA
dumitru@yahoo-inc.com

Abstract

We investigate the use of Hessian Free optimization for learning deep autoencoders. One of the critical components in that algorithm is the choice of the preconditioner. We argue in this paper that the *Jacobi* preconditioner leads to faster optimization and we show how it can be accurately and efficiently estimated using a randomized algorithm.

1 Introduction

Deep architectures have gained a lot of attention recently, as they have been shown to extract meaningful non-linear features from the data, which tend to generalize well in a variety of supervised learning settings [Hinton and Salakhutdinov, 2006, Bengio, 2009]. A typical semi-supervised setting is that of *pre-training* a deep architecture using unsupervised building blocks such as Restricted Boltzmann Machines [Hinton and Salakhutdinov, 2006] or non-linear (and possibly denoising) auto-encoders [Bengio et al., 2007, Vincent et al., 2010], followed by optimizing the supervised objective.

Oftentimes, the optimization of the unsupervised learning cost in auto-encoders (shallow and deep) uses stochastic gradient descent (SGD) or non-linear conjugate gradient (NCG) [Hinton and Salakhutdinov, 2006]. Martens [2010] suggests that 2nd order methods, and in particular Hessian-free (HF) methods, are particularly well-suited for optimizing deep auto-encoders. The author argues that the objective functions in such systems exhibit pathological curvature, and that 1st order methods such as SGD are not able to generally overcome this problem because they are blind to the curvature of the objective function.

This paper is a further investigation of Hessian-free methods in the context of deep auto-encoders. Following remarks made in Martens [2010], we argue that a crucial ingredient in this optimization is the preconditioning method used in the Conjugate Gradient subroutine or HF. We propose to use the standard *Jacobi* preconditioner and show how it can be reliably and efficiently estimated with a randomized algorithm. This preconditioner compares favorably with the one proposed by Martens [2010], and makes it possible to optimize the reconstruction error of a deep autoencoder’s objective function faster and better.

2 Hessian-free optimization

Truncated Newton [Nash, 2000] is a powerful optimization technique. There are several variants of it, but they all share in common the same key ingredient. Instead of computing exactly the Newton direction—defined as $H^{-1}g$ where H is the Hessian and g the gradient—a truncated Newton method solves the linear system using conjugate gradient (CG). CG is

an iterative method where each step involves a matrix-vector multiplication Hv . When the number of parameters is large, it is not possible to explicitly compute H , but the matrix-vector product can still be computed [Pearlmutter, 1994].

The Hessian Free (HF) method [Martens, 2010] is a truncated Newton method with some additional tricks designed for the optimization of deep networks, such as the deep autoencoder introduced by Hinton and Salakhutdinov [2006]. We have partly implemented HF in Theano [Bergstra et al., 2010]. The pseudo-code of our implementation is given in algorithm 1.

Algorithm 1 Our implementation (in Theano) of the Hessian Free optimizer

```

 $w \leftarrow 0$  (or some other initialization);  $\lambda \leftarrow 1$ ;  $s_{prev} \leftarrow 0$ 
while Not converged do
   $g = \text{gradient}$ 
   $p = \text{preconditioner}$ 
  Solve  $(A + \lambda I)s = -g$  using CG, where  $A$  is the Gauss-Newton matrix,
  with preconditioner  $p$  and starting from  $s_{prev}$ .
  if  $f(w + s) > f(w)$  then
     $\lambda \leftarrow 4\lambda$ ;  $s_{prev} \leftarrow 0$ .
    Go back to the previous step
  end if
   $\rho \leftarrow \frac{f(w) - f(w + s)}{\frac{1}{2}s^\top A s + s^\top g}$  { $\rho$  should ideally be close to 1.}
   $\lambda \leftarrow 2\lambda$  if  $\rho < 0.25$ ;  $\lambda \leftarrow \frac{\lambda}{2}$  if  $\rho > 0.75$ . {Damping like in Levenberg–Marquardt}
   $w \leftarrow w + s$ ;  $s_{prev} \leftarrow s$ .
end while

```

In order to define the Gauss-Newton matrix used in algorithm 1, let us first make some assumptions on how the objective function f is defined. For a given example x_p , the j -th output unit is denoted $o_j(x_p, w)$. These outputs are then combined to produce a loss. For instance, in a reconstruction task, the cross-entropy loss can be written as $\sum_j \phi_j(o_j, x_p)$, with $\phi_j(o, x) = -x^j \log(o) - (1 - x^j) \log(1 - o)$. Commonly used loss functions can be written in this form with ϕ_j a convex function.

If the objective function f is of the form,

$$f(w) = \sum_p \sum_j \phi_j(o_j(x_p, w), x_p),$$

then the Gauss-Newton matrix A is defined as:

$$A = \sum_p J^p H^p (J^p)^\top, \quad \text{with } J_{ij}^p = \frac{\partial o_j(x_p, w)}{\partial w_i} \quad \text{and } H_{jj}^p = \frac{\partial^2 \phi_j(o_j(x_p, w), x_p)}{\partial o^2}. \quad (1)$$

The advantage of using the Gauss-Newton over the Hessian is that it is positive definite as long as the ϕ_j are convex.

Multiplication between A and a vector has been studied in [Schraudolph, 2002] and uses the \mathcal{R} operator defined in [Pearlmutter, 1994]. This operator has been implemented in Theano and as a result we were able to code the multiplication by the Gauss-Newton matrix in only 3 lines—one line for each the products involved in (1).

One of the key ingredients in the HF method is the choice of the preconditioner. A standard choice is the so-called *Jacobi* preconditioner which consists of the diagonal elements of the matrix of interest. Martens [2010] rightfully observed that there is no efficient way of computing the diagonal elements of the Gauss-Newton matrix. He then decided to use the diagonal elements of the Fisher information matrix as a substitute. But as we will see in the next section, it is possible to efficiently compute an accurate and unbiased *estimate* of the diagonal elements of the Gauss-Newton matrix.

3 Improved preconditioner

We want to compute the diagonal elements of the Gauss-Newton matrix (1):

$$A_{ii} = \sum_p \sum_j (J_{ij}^p)^2 H_{jj}^p.$$

For this purpose, let us define the following random variable

$$a_{ip} := \sum_j J_{ij}^p \sqrt{H_{jj}^p} \epsilon_{jp},$$

where ϵ_{jp} is a $+/-1$ random variable (with equal probability). a_{ip} can be computed with a slightly modified backprop. Indeed, after the forward pass, one simply needs to backpropagate $\sqrt{H_{jj}^p} \epsilon_{jp}$ from the output layer instead of propagating $\phi'_p(o_j(x_p, w))$ as one would normally do for computing the gradient. This can be done easily in Theano using the \mathcal{L} operator.

Theorem 1. $\sum_p a_{ip}^2$ is an unbiased estimate of A_{ii} . Moreover, if the training data is i.i.d., its relative variance decays in $O(1/N)$, where N is the number of training examples.

Proof. Since ϵ_{jp} are independent variables with zero mean and unit standard deviation, we immediately obtain that for all p , $\mathbb{E}a_{ip}^2 = (J_{ij}^p)^2 H_{jj}^p$. This proves the unbiasedness. If the training data is i.i.d., so are the variables a_{ip}^2 . The second part of the theorem simply states the variance of a mean of i.i.d. variables is inversely proportional to the number of variables. \square

4 Experimental setup

We follow closely the experimental setup of [Martens, 2010] and [Hinton and Salakhutdinov, 2006], in that the choice of the model, architecture, and datasets is precisely the same. In this section, we describe summarize this setup and highlight the differences.

4.1 Datasets

We evaluated the various optimization methods on two datasets: **MNIST** and **Curves**:

MNIST is a dataset of 50,000 training examples of size 28x28 representing hand-written digits.

Curves is a dataset of synthetic curves first described by Hinton and Salakhutdinov [2006].

The curves are generated from three randomly chosen points in two dimensions.

The training set contains 20,000 images of size 28x28.

We did not use the test parts of these datasets. No special preprocessing was performed, beyond making sure that the inputs are between 0 and 1.

4.2 Models and architectures

A standard deep auto-encoder as described by [Hinton and Salakhutdinov, 2006] was used. For **Curves**, the sizes of the hidden layers are 400 – 200 – 100 – 50 – 25 – 6, and for **MNIST** the sizes are 1000 – 500 – 250 – 30. The deep auto-encoder was either pre-trained using standard RBM training as described by [Hinton and Salakhutdinov, 2006], or randomly initialized following [Glorot and Bengio, 2010, Section 4.2.1]. The training loss is the cross-entropy.

4.3 Optimization methods and their hyper-parameters

The Hessian-free algorithm was implemented following closely the code by Martens [2010], with a few differences (see also algorithm 1):

- The gradient and objectives functions are computed on the full dataset, but the CG iterations are done on a mini-batch of size 4000.
- The stopping criterion for the linear conjugate gradient is slightly simpler: if $1 - f^{CG}(s_{i-1})/f^{CG}(s_i) < 10^{-4}$, with $f^{CG}(s) = \frac{1}{2}s^\top As + s^\top g$, or if the number of iterations is larger than 50, CG terminates.
- We use the CG search direction found in the last iteration of HF instead of starting from a zero vector (same as [Martens, 2010]). The motivation is that of improving the convergence of CG.
- CG is not guaranteed to decrease the objective function value. In such cases, the damping factor λ is increased and the CG optimization is restarted from a zero vector.

Our comparison is then between *Hessian-free without pre-conditioning* (**HF-None**), with *Fisher matrix pre-conditioning* (**HF-Martens**), as described in [Martens, 2010], and with *Jacobi pre-conditioning* (**HF-Jacobi**), the preconditioning method introduced in Section 3.

We also compare with two standard and widely used optimization techniques:

Stochastic Gradient Descent (SGD) : A learning rate of 0.02 with a mini-batch size of 20 was used.

Non-linear Conjugate Gradient (NCG) : Standard non-linear CG is performed with 50 functions evaluations on each mini-batch of size 4000.

5 Experiments and discussion

The experimental framework was implemented in Theano [Bergstra et al., 2010], a Python library that makes it possible to quickly define and evaluate mathematical expressions involving multi-dimensional arrays. The methods are implemented such that any standard differentiable objective function can be used.¹

We report in this paper the KL divergences between the input and the reconstruction, instead of the cross-entropy, as it allows us to lower-bound the error at zero. We also computed the Mean Squared Error (MSE) between the input and the reconstruction and its trend was very similar to the KL; for this reason we do not report MSE. Since the focus of this paper is on optimization, the KL divergence is computed on the training set.

5.1 Influence of the preconditioner

Figure 1 compares the various choices of the preconditioner on the two datasets described above.

While Martens [2010] does not report results on Hessian-free optimization without preconditioning and simply states that preconditioning accelerates CG, our results show that preconditioning plays an important role in the convergence of the optimization algorithm. Moreover, the type of preconditioning is clearly important as well: results on **Curves** shows that **HF-Jacobi** converges faster than **HF-Martens**.

However, on MNIST with pre-training, **HF-Jacobi** does not converge as fast as **HF-Martens**. This might be because of the mini-batch: it is likely that the Jacobi preconditioner leads to a better minimization of the quadratic approximation, but since this approximation is only computed on a mini-batch, the direction found by CG can be misleading. One of the role of the damping factor is to “regularize” the CG solution. And the damping

¹The source code will be released upon publication

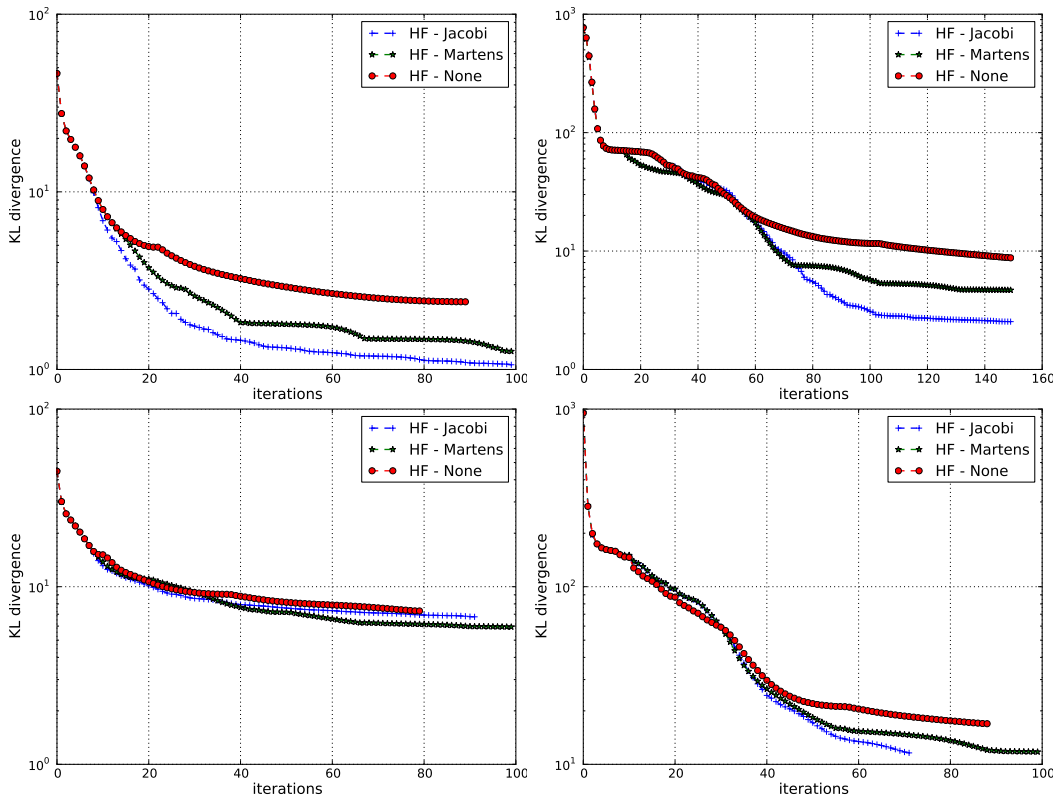


Figure 1: The KL divergence between the input and the reconstruction of the deep auto-encoder. *Left*: after RBM pretraining, *Right*: random initialization, *Top*: Curves, *Bottom*: MNIST

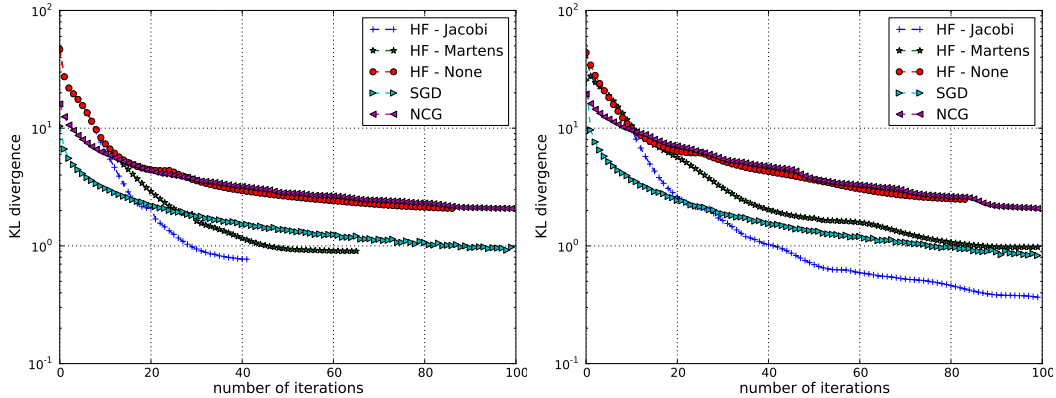


Figure 2: KL divergences on *Curves* (Left) and *MNIST* (Right) after RBM pretraining using a single batch of size 4000 and 7500 respectively.

factor was relatively large for **HF-Jacobi**: its median value during the optimization was 2^{-6} while it was 2^{-15} for **HF-Martens**. It is likely that the CG backtracking method proposed by Martens [2010, Section 4.6] would solve this problem by letting **HF-Jacobi** terminate earlier its CG iterations.

To confirm the above hypothesis, we trained the various methods using a *single* batch of size 7500 on *MNIST*. The plot in figure 2 shows that **HF-Jacobi** converges better in this case.

In other experiments (not shown in this paper) we observed that the size of the mini-batch (in the full version of the HF method) has an influence on the relative order of performance among the preconditioners described above²; a more complete investigation is in order.

5.2 Comparison with other techniques

Even though the primarily focus of the paper was to study the influence of the preconditioned, Figures 2 (single batch), and 3 (mini-batch on full set) compare HF with the other standard optimization techniques, with and without pre-training. Non-linear conjugate gradient (NCG) performed poorly while stochastic gradient descent (SGD) appears to be very competitive. A full comparison between HF and SGD is out of the scope of this paper and would require a more complete implementation of HF with the various improvements proposed by Martens [2010].

In Figure 3, the average time spent for each pass over the data differs by method: it is 7 seconds for SGD, 4.5 seconds for HF and 3.6 seconds for NCG.

6 Conclusion

We evaluated a new preconditioning method for Hessian-free optimization. The analysis was performed in a standard setting of optimizing the reconstruction error of a deep auto-encoder. On two vision datasets, the **HF-Jacobi** preconditioner seems to compare favorably to the previously-used preconditioner for HF methods.

For future work, we plan on studying the effect of the new preconditioner in other settings—both for different models (not just deep auto-encoders) and for different objective functions, including the study of its effects on generalization³. Also we want to test SGD with our Jacobi preconditioner: it is likely that SGD can converge faster after appropriate preconditioning.

²Larger minibatches favored **HF-Jacobi**

³To test the hypothesis postulated by [Martens, 2010], who states that under-fitting might be an issue in training deep autoencoders

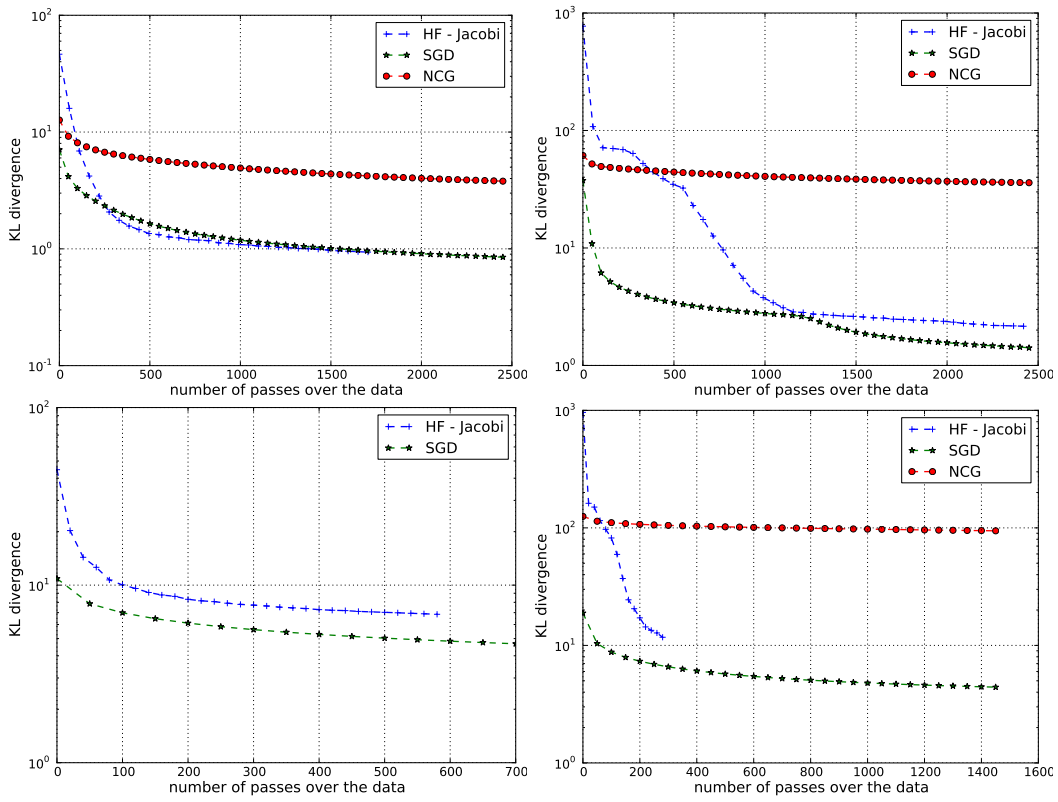


Figure 3: The KL divergence between the input and the reconstruction of the deep auto-encoder. *Left*: after RBM pretraining, *Right*: random initialization, *Top*: Curves, *Bottom*: MNIST

References

- Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.
- Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press, 2007.
- James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral.
- Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May 2010.
- Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.
- Quoc Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.
- James Martens. Deep learning via hessian-free optimization. In Léon Bottou and Michael Littman, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 735–742. ACM, June 2010.
- Stephen G. Nash. A survey of truncated-newton methods. *Journal of Computational and Applied Mathematics*, 124:45–59, 2000.
- Barak Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.
- Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.
- Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(3371–3408), December 2010.