Hessian Free Deep Learning

Subodh Iyengar

December 10, 2010

1 Introduction

Optimization techniques used in Machine Learning play an important role in the training of the Neural Network in regression and classification tasks. Predominantly, first order optimization methods such as Gradient Descent have been used in the training of Neural Networks, since second order methods, such as Newton's method, are computationally infeasible. However, second order methods show much better convergence characteristics than first order methods, because they also take into account the curvature of the error space. Additionally, first order methods require a lot of tuning of the decrease parameter, which is application specific. They also have a tendency to get trapped in local optimum and exhibit slow convergence. Thus Newton's method is absolutely essential to train networks with deep architectures.

The reason for in-feasibility of Newton's method is the computation of the Hessian matrix, which takes prohibitively long. Influential work by Pearlmutter [2] led to development of a method of using the Hessian without actually computing it. Recent work [1] has involved training of a deep network consisting of a number of Restricted Boltzmann Machine using Newton's method without directly computing the Hessian matrix, in a form of "Hessian free" learning. The method had exhibited success on the MNIST handwriting recognition data set when used to train an Restricted Boltzmann Machine using Hinton's [3] method, with a better quality solution for classification tasks.

The proposed work for the CS229 project aims to improve upon the method of "Hessian Free" (HF) learning and apply it to different classification tasks. To do this, the Hessian free learning method will be implemented and results for the experiments using MNIST will be replicated. Through analysis, it is aimed to propose further modifications that will improve the method and also run it on different classification tasks.

2 The Hessian Free Method

Unlike Gradient Descent, in Newton's method the update equation uses a second derivative Hessian to compute the next step:

$$X = X - H^{-1} * \nabla f \tag{1}$$

However, computing the inverse of the Hessian is an $O(n^3)$ operation. For Neural Networks with deep architectures, the number of weights is large, thus the Hessian is a very big matrix. An inversion would take a lot of time and is simply unacceptable for practical applications. Thus, Newton's method has almost been neglected in work on training Neural Networks.

For practical purposes though, it is not necessary to invert the Hessian Matrix. A procedure to calculate it could be formulated as:

$$p = H^{-1} * \nabla f \tag{2}$$

$$Hp = \nabla f \tag{3}$$

The resulting equation is of the form $Ax = b$ and can be solved using some method like Conjugate gradient, provided that H is positive definite. However computing the Hessian itself is an $O(n^2)$ operation, involving perturbing each individual weight and propagating it forward.

The Hessian Free method has existed in literature [2] for a long time. The idea behind the Hessian Free method derives from the equation (3). Instead of physically computing the Hessian which is time consuming, we only need to compute the product, Hp, a matrix-vector product. We can then solve the same equation using Conjugate Gradient for p. This can be done by taking the finite difference of gradient computed in the direction of p as follows:

$$Hp = \lim_{\epsilon \to 0} \frac{\nabla f(\theta + \epsilon d) - \nabla f(\theta)}{\epsilon} \tag{4}$$

However it is found that the Hessian matrix is usually very unstable. Schraudolph adapted Pearlmutter's R-propagation [2] method to use the Gauss-Newton approximation to the Hessian matrix. The Gauss-Newton Matrix G is guaranteed to be positive definite and is quite a good approximation to the Hessian, as well as being cheaper to compute.

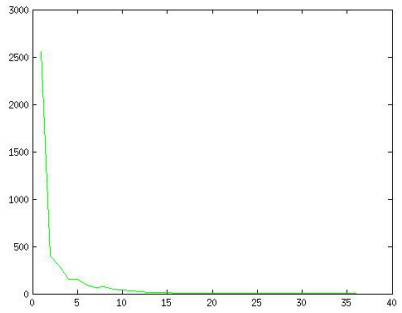
In a recent paper, Martens [1] uses this technique effectively to train a deep network without pre-training. Modifications were proposed for example using mini-batches and modifying the stopping conditions of CG to the Hessian free algorithm to make it more suitable for Machine Learning applications.

His findings indicate that the method performs well without pre-training and beats the simple back-propagation learning procedure in various data sets making it a very interesting method to study.

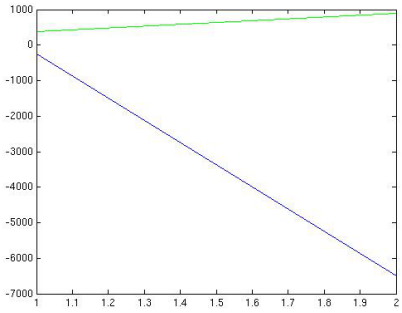
3 The problem with the Hessian

In the optimization framework proposed above, the conjugate gradient method converges to a solution, provided that the Hessian matrix is positive definite. However for practical neural networks that represent non-convex functions, the

Hessian is not guaranteed to be positive definite. This fact had a profound effect on this project on which a lot of time was expended experimenting with the Hessian only to discover that it was indeed negative definite. When optimizing the weights of the deep Network using the Hessian free method, the loss function is found to increase rather than decrease. To ascertain whether the Hessian matrix is indeed positive definite would involve computing its eigenvalues, and examining whether each on is positive, which is a computationally intensive task. Instead, the graphs below that show the behavior of the residue of the Conjugate gradient step(CG) vs. number of iterations of CG confirm that the Hessian is negative definite. The residue (given by the green line) increases instead of decreasing.



(a) Residue using Gauss Newton



(b) Residue using Hessian(Green)

To subvert this dilemma the Hessian matrix is approximated by the Gauss-Newton approximation of the Hessian $H = J^T * J$ where J is the Jacobian matrix of the neural network. So the CG system that must be solved becomes [5]

$$(J^T * J + \lambda I)p = J^T * E$$

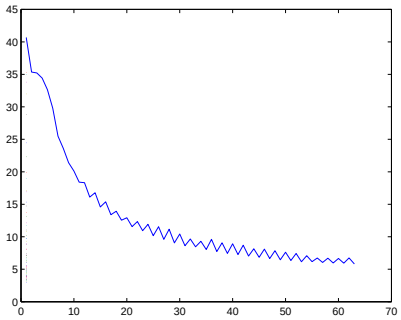
Where E is the loss function output, i.e. the error term, and $J^T * E$ represents the gradient value at that point. The Gauss Newton matrix is guaranteed to be positive definite as shown by the CG minimization step in Fig. 3(a).

4 Implementation and Experiments

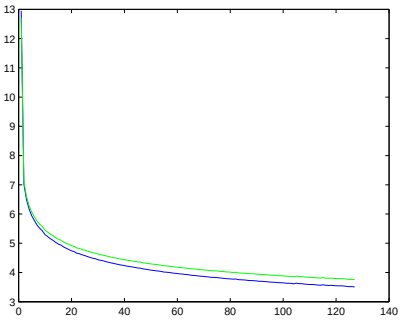
Initially, the Hessian free method was implemented over the code provided by Hinton for his Science paper [4]. However it was found that the stacked RBMs and the other complexities of Hinton's code resulted in a lot of memory leaks and the program used to run out of memory easily. To put things into perspective, the network used in Hinton's case is a 784-1000-250-30-250-1000-784 network. Thus the Jacobian matrix has *2.8 million* x *size of minibatch* entries of doubles and the multiplication of J^T by J thus incurs a huge memory overhead.

Thus Hinton's code was reimplemented from scratch and the code was systematically tested on various sizes of neural networks to prove the correctness if the method. In the smaller networks the solutions converged to zero very fast, within 4 epochs. It was tested against plain gradient descent backpropagation and it did significantly better than the latter method. In bigger networks, the method took time to converge. Hessian Free was tested mainly on 2 neural network architectures, 784-400-784 and 784-250-30-250-784.

To circumvent the issue of memory, and due to project time constraints, the networks were trained on a limited number of minibatches from the MNIST dataset of size 100 each and tested on 10000 patterns on MNIST. The method was run for a particular number of epochs until it was believed that it had converged to a low enough value of reconstruction error. The convergence characteristics of each experiment are shown in figures.

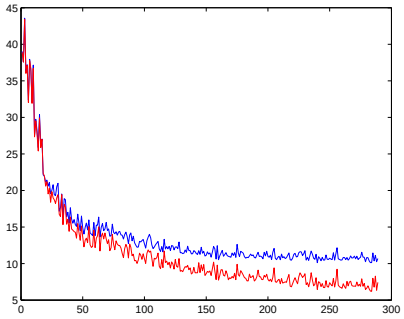


(c) Test Error vs. Epochs (Small Network)

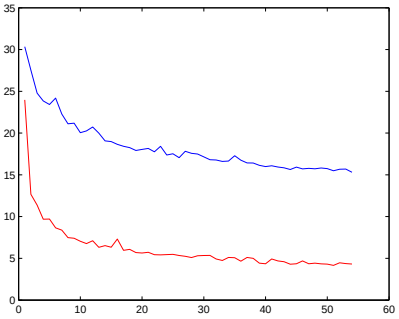


(d) Error vs. Epochs (Hinton)

Figure (c) show the convergence of Hessian Free on the 784-400-784 network with 20 minibatches. Figure (e) and (f) shows the convergence of HF on



(e) Training(R) Testing(B) Error vs. Epochs



(f) Training(R) Testing(B) Error vs. Epochs

the 784-1000-250-30-250-1000-784 network with 20 minibatches and 100 minibatches respectively for training. Figure (d) shows the results of Hinton on his deep architecture. The Table shows the resulting reconstruction errors for various experimental parameters. The graphs are compared with the results obtained from Hinton's experiments and are found to be not quite competitive with the pretraining methods, however the achievement of this project was in getting the reconstruction error to fall consistently, since, in numerous initial experiments the reconstruction error used to decrease initially and then increase later.

Network size	Minibatches	Epochs	Training Error	Test Error
784-400-784	20	60	4.4	5.1
784-250-30-250-784	100	50	4.15	15.48
784-250-30-250-784	20	200	6.2	10.15

5 Discussion

The HF method is found to perform reasonably well for small data set size and for minibatches of smaller size. The method as coded in this report is still not as competitive as it should be against CG back-propagation, however with future work on the improving the method, it should become more competitve. There could be a lot of refinements the algorithm as coded in this report to make it more memory efficient like using Schraudolph's method for calculating the Gauss Newton approximation as proposed by Martens.

6 Conclusion

In the course of the project, Martens' algorithm has been replicated to some extent. The overarching objective of the project, i.e to apply the methods in applications was not met, however encouraging results were obtained by applying the method to MNIST. Instead this shall be kept in the scope of future work. Further effort must be expended to understand the HF algorithm better and implement some of the more finer aspects of it.

7 Acknowledgements

I would like to thank Quoc Le for his help during this project without whom the project would not have reached even this stage.

References

[1] Martens, J. Deep learning via Hessian-free optimization, ICML, 2010.

[2] Pearlmutter, B. A. Fast exact multiplication by the hessian. Neural Computation, 1994.

[3] Hinton, G. E. and Salakhutdinov, R. R Reducing the dimensionality of data with neural networks. Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.

[4] http://www.cs.toronto.edu/ hinton/MatlabForSciencePaper.html

[5] Numerical Optimization, Nocedal and Wright