HESSIAN FREE OPTIMIZATION METHODS FOR MACHINE LEARNING PROBLEMS

AJ FRIEND, ED SCHMERLING, AKSHAY MITTAL

CS229 Class Project

Abstract. In this article, we describe the algorithm and study the performance of a Hessian free optimization technique applied to machine learning problems. We implement the commonly used black box model for optimization and solve a particular challenging recursive neural network learning problem, which exhibits a non-convex and non-differentiable function output. In order to adapt the method to machine learning objective functions, we explore a few modifications to the optimization routine such as damping, mini-batching, modifying termination criteria, etc. The convergence properties are compared against the quasi-Newton L-BFGS method.

Introduction

In most cases, machine learning models essentially solve a single or series of optimization (minimization) steps to find the optimum set of parameters and quantities of interest. The focus of this project is on an efficient iterative routine to use in these optimization steps. The occurrence of large training sets and number of parameters is common in machine learning models, as seen in learning problems such at those described in [1], where recursive neural networks are implemented. The following sections describe a popular Hessian free algorithm [2], which was implemented and tested for a complex Recursive Autoencoder model. We model the objective as an unconditional black box, that is, we only get information about the function value and the gradient (or subgradient) at each iteration step. We also describe the modifications done to the standard Hessian free algorithm to adapt it to machine learning problems, which can exhibit non-smooth and non-convex response behavior, in order to ensure the good convergence properties of the Newton-like steps.

Hessian-Free Optimization

Black Box Model. Our aim is to provide an optimization framework that is applicable to a wide range of problems. In most machine learning problems however, we often run into large data sets and complex code steps to evaluate the objective function and gradient, and it is often impractical to develop intrusive optimization models for these problems. Therefore, the framework we will use requires minimal information about the objective function, i.e. we only require the function value and a derivative at any query point. Thus, we have an unconditional oracle that takes as input a vector x and outputs the function value f and $g(x) = \nabla f(x)$ (or subgradient).

Hessian-Free Minimization. We implement the Hessian-free optimization method advanced by Martens in [2]. We are interested in designing a Newton or Netwon like step at each iteration in the minimization algorithm, which requires curvature i.e second derivative information about the objective function. For large problems, evaluating, storing and inverting the hessian $H(x) = \nabla^2 f(x)$ can be prohibitively expensive due to limited computational resources. Quasi-Netwon methods like L-BFGS tackle some of these challenges by efficiently storing a Hessian approximation. A Hessian-free optimization algorithm, as the name suggests, does not explicitly calculate the Hessian at any point. The algorithm is based on the quadratic minimization model i.e. Newton's method that is described as follows.

Date: Dec 16, 2011.

At each iteration k, we seek a search direction p_k that minimizes the quadratic approximation to the objective function

$$q_k(p) = f(x_k) + p^T g(x_k) + \frac{1}{2} p^T H(x_k) p \approx f(x_k + p). \tag{1}$$

For reasons noted in the damping section below, we actually consider $B(x_k) = H(x_k) + \lambda I$ as our quadratic term, instead of $H(x_k)$. Ideally, we would like to compute

$$p_k = \arg\min_p q_k(p) \Rightarrow B(x_k) p_k = -g(x_k).$$

The above problem is a linear system. Hessian free optimization attempts to solve this system using the conjugate gradient (CG) iterative method, as it only requires matrix-vector products involving $B(x_k)$. The advantage of this idea is that we can form these matrix-vector products without ever explicitly forming the Hessian matrix. Evaluating $B(x_k)$ times a vector d is essentially a directional derivative of the gradient:

$$B(x_k) d = (H(x_k) + \lambda I) d = \lim_{\epsilon \to 0} \frac{g(x_k + \epsilon d) - g(x_k)}{\epsilon} + \lambda d$$

where the limit can be calculated using finite differences (taking $\epsilon = \sqrt{\text{machine precision}}$).

We should note that CG is designed to solve positive definite systems, and that $B(x_k)$ will often be indefinite. We can terminate early or use directions of negative curvature when they are detected to account for this. Also, the system need not be solved exactly. Often we can terminate the algorithm early with an approximate solution to get a good search direction.

The basic outline of the algorithm is as follows.

Algorithm 1 Hessian-Free Optimization

(1) **for** $k = 1, 2, \ldots$ **do**
(2) $g_k \leftarrow \nabla f(x_k)$
(3) compute/update λ
(4) define the function $B_k(d) = H(x_k) d + \lambda d$
(5) $p_k \leftarrow$ CG-solve $(B_k, -g_k)$
(6) $x_{k+1} \leftarrow x_k + p_k$
(7) **end for**

Recursive Autoencoders - Test Problem

The primary test problem for our implementation of Hessian-free optimization comes from the *Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions* paper of Socher et al. [1]. This paper describes a model where, in a very rough sense, a neural network is used to recursively combine word/phrase vectors of a sentence in order to arrive at a single vector representing the whole sentence. This vector might then in turn be used to infer a sentiment for the sentence. We wish to choose the neural network parameters so that the error both in constructing this recursive combination and in deriving sentiments from the output vectors is as small as possible. The objective function to be minimized is a sum of these errors plus a regularization term. From this we may see that evaluating the RAE objective function is approximately linear in the size of its data set (assuming a fixed distribution on the length of training sentences), a fact we use to tune HF optimization for maximum efficiency. This is the only fact specific to the test problem that we use; everything else is consistent with a black box model for optimization.

Adapting HF to Machine Learning Problems

Martens describes a number of modifications to the basic Algorithm 1 that he claims work particularly well for neural network training [2]. We describe our experience with implementing his enhancements for training the RAE model.

Damping. The issue of damping is important to consider in the HF approach because we run CG with implicit access to the true Hessian at each iteration. Thus, unlike with L-BFGS where we maintain a strictly positive definite approximation to the curvature matrix, we may run across directions of extremely low or even negative curvature in HF. If we are not careful, moving along such a direction could result in a very large CG step, possibly taking the algorithm outside of the region where the Newton approximation is even valid. One solution is to add a damping parameter to the Hessian and consider $B = H + \lambda I$. This reconditions H and controls the length of each CG step, mimicking a trust region method. As the condition of B is constantly changing as the HF optimization algorithm searches, we start with $\lambda = 1$ and update λ according to a Levenberg-Marquardt style heuristic:

$$\textbf{if } \rho_k < \frac{1}{4}, \textbf{then } \lambda \leftarrow \alpha\lambda.$$
$$\textbf{elseif } \rho_k > \frac{3}{4}, \textbf{then } \lambda \leftarrow \alpha^{-1}\lambda.$$

where ρ_k is the reduction ratio measuring the accuracy of the Newton approximation and is defined as follows:

$$\rho_k = \frac{f(x_k + p_k) - f(x_k)}{q_k(p_k) - q_k(0)}.$$

Martens suggests $\alpha = \frac{3}{2}$ but we find that this value of α is too large for the RAE objective. When $\rho < \frac{1}{4}$ and the quadratic model is judged to be inaccurate, increasing λ by a factor of $\alpha = \frac{3}{2}$ is too extreme and forces the steplength per iteration to 0, shutting down any optimization. On the other hand when we deem the quadratic model as trustworthy ($\rho > \frac{3}{4}$), taking $\lambda \leftarrow \frac{2}{3}\lambda$ leads to underdamping and the steplengths grow out of control. We find that a slower ratio of $\alpha = \frac{10}{9}$ works well for keeping damping reasonable for the RAE objective.

Mini-batching. The CG minimization problem solved in each step of the HF algorithm is only an approximation to the true RAE objective. Thus it seems reasonable to also approximate the directional derivatives Hd required by CG if doing so is computationally advantageous. To this end we fix "mini-batches" of the training dataset per HFO iteration to which we restrict our computation of the objective function. In the RAE example of [1], the training dataset has 9596 sentences/sentiments. If we choose a mini-batch of only 100 elements, each CG step effectively requires only $\sim \frac{1}{100}$th of a function evaluation. This is the speed improvement that really makes HFO worth considering.

Martens recommends growing the size of the mini-batch as the HF algorithm progresses, but we actually find this not to be the case for the RAE problem. After trying varying schemes of increasing minibatch size we observe that keeping a fixed minibatch size of 100 not only saves on computation but results in a better objective function minimum in the end. More discussion of this phenomenon may be found in the results section.

Directions of Negative Curvature. Even with damping, CG sometimes computes directions of negative curvature for B. This is an unavoidable consequence of the RAE objective being nonconvex. The standard approaches are to exit CG and to either step along the last non-negative curvature direction or to step along the new direction of negative curvature. Our experiments lead us to conclude that there is no clear difference between these two approaches.

In [2], Martens recommends using the Gauss-Newton matrix instead of the actual Hessian matrix, as it is a positive semidefinite approximation to the Hessian. Thus, problems with negative curvature in CG are avoided. We did not use the Gauss-Newton matrix, as our black box abstraction only gave us access to the Hessian. This may have been an important difference between our implementation and [2], and better performance might have been achieved with this approach.

Mini-batched Linesearch. Martens does not mention this addition, but we find that HF performance is improved by using backtracking linesearch on the mini-batched function approximation to take a variable steplength along the direction that CG returns. This is simply another way to take advantage of the relative cheapness of mini-batched function evaluations.

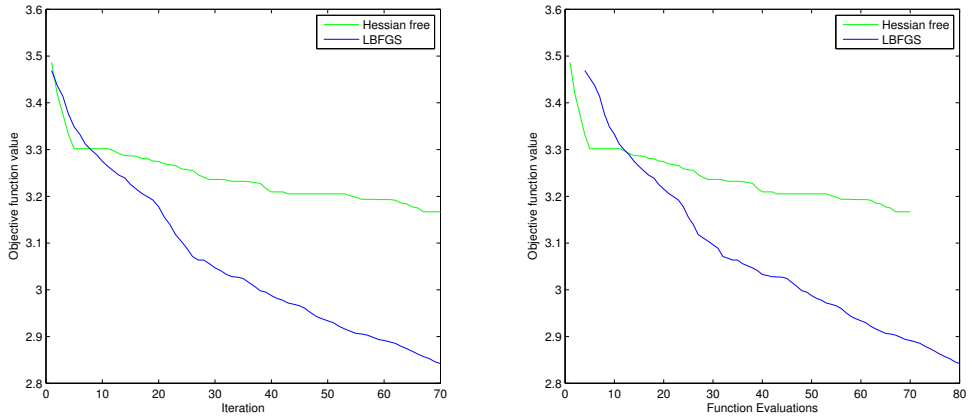

FIGURE 0.1. Comparing convergence properties. HF v.s. L-BFGS. The left plot gives the number of HF iterations. The right plot gives the number of "function evaluations", where a function evaluation can be fractional if it only uses a minibatch instead of the whole data set.

Warm-Starting CG. It is not unreasonable to assume that each CG subproblem is somehow related or similar. Thus, the solution of two successive search direction problems can be similar. As CG is iterative, starting the algorithm at a point which may already be close to the solution can reduce the number of iterations needed to converge. Martens notes, and we agree, that initializing CG at each HF iteration with the direction output by the previous iteration's CG step improves the convergence rate.

Results and Conclusions

We compared our HF implementation to 70 iterations of the L-BFGS algorithm used in [1]. Plots comparing the convergence behavior of the two algorithms are given in the figure.

We observed that Hessian free optimization performs well against L-BFGS at early iterations, but performs worse later on. We expect this is due to the fact that L-BFGS accumulates curvature information at each iteration, improving the model of its problem. These early approximations are based on only a few gradient evaluations, and are thus less accurate.

The Hessian free approach uses only curvature information at the current point, but is potentially advantageous because it essentially has access to the actual hessian. The relative performance of Hessian free optimization appears to decrease as the L-BFGS model "catches up" to finding the true curvature information. However, we should note that there are many parameters to hessian free optimization, including minibatch sizing, damping parameter, back-tracking, and the treatment of direction of negative curvature. Tuning these parameters may provide better results on a per-application basis and gives the user added flexibility in adapting the optimization method to his needs that L-BFGS cannot provide to the same extent as Hessian free optimization does.

We were unable to match the performance of the L-BFGS method. This may be because we did not sufficiently tune our implementation. However, it is also very likely that this is due to our use of the Hessian instead of the Gauss-Newton matrix, which Martens claims often provides better search directions.

Another possibility is that L-BFGS methods may simply be better suited to these types of problems. L-BFGS makes very good use of the gradient information supplied at each step by incorporating approximate curvature information into a Hessian approximation. In this way, the information from each gradient is used many times. In contrast, HF uses a gradient's information just once in each CG iteration. We believe that HF is most promising when minibatching can be employed to make this drawback less severe.

In conclusion, we think that further experimentation with these methods should be done, exploring the trade-offs between HF and L-BFGS, to determine for which problem domains these methods are best suited.

Acknowledgements

We would like to thank Professor Andrew Ng and TA Richard Socher for their guidance and assistance during the course of this project. We also acknowledge the other CS229 teaching staff for their support and help during the quarter towards the completion of this project and its key milestones.

References

1. Socher, R. et al. *Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions.* Conference on Empirical Methods in Natural Language Processing (EMNLP 2011).
2. Martens, J. *Deep learning via Hessian-free optimization.* Proceedings of the 27th International Conference on Machine Learning (2010).