Higher Order Optimization In Deep Learning

Harish Rajagopal
160552, Group 3

Raghav Garg
160534, Group 3

Parv Mor
160479, Group 3

Vishwas Lathi
160808, Group 3

Yash Mahajan
160822, Group 3

1 Problem Description/Variation

Our project [1] is a survey on the use of higher order optimization methods in Deep Learning, particularly quasi-Newton methods.

Various First Order stochastic methods like SGD, ADAGRAD, RMSProp are being used to train Deep Learning models, but all of them suffer from curvature problems as it is very hard to incorporate curvature information in their training.

2 Prelimanaries

2.1 Newton's Method

Newtons method, like gradient descent, is an optimization algorithm which iteratively updates the parameters $\theta \in \mathbb{R}^N$ of an objective function f by computing search directions p and updating θ as $\theta + \alpha p$ for some α.

The central idea behind Newtons method is that f can be locally approximated as:

$$f(\theta + p) \approx q_\theta(p) = f(\theta) + \Delta f(\theta)^\intercal p + \frac{1}{2} p^\intercal B p$$

Here B is the data Hessian matrix which incorporates the curvature information. However, it is too expensive to calculate the Hessian at every step of iteration and thus, although these methods require less number of iterations to converge, the process is usually very slow.

Thus, Quasi-Newton methods are used in place of standard Newton's method. These methods tend to approximate Hessian at each step of iteration in order to make these second order methods faster. Various models that we have studied are Hessian-Free, SFO, AdaQN that uses different Optimization techniques that differ majorly in three major aspects:

- The update rule for the curvature pairs used in the computation of the quasi-Newton matrix
- The frequency of updating
- The applicability to non-convex problems

2.2 L-BFGS update equations

The generic update equations used in L-BFGS methods are

$$s_k = w_{k+1} - w_k$$
$$y_k = \Delta_{B_k} f(w_{k+1}) - \Delta_{B_k} f(w_k)$$

[1] Disclaimer: This report is not reused from any other course project and neither from any internship.

Here B_{k+1} satisfies the secant equation: $B_{k+1}s_k = y_k$

$$H_{k+1}^{-1} = B_{k+1} = B_k + \frac{y_k^T y_k}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \delta I$$

3 Literature Review and description of work on problem

3.1 Hessian-Free Optimization [1]

The loss functions of neural networks are highly non-convex and suffer from the problem of 'pathological curvature'. The pathological curvature is a ravine like region which if difficult for the first order methods like SGD, to traverse and hence reach the minima. If the directions of significant decrease are of low curvature then the optimization via SGD may become too slow and can give false impression of convergence to local minima. HF methods avoid this problem by rescaling the gradient and makes significant progress even along directions of low curvature.

Conjugate Descent is a HF method which makes use of the quadratic nature of optimization problem to generate a set of conjugate directions, which are used along with line search to update the parameters. This method requires computing matrix-vector products which are approximated with high accuracy using finite differences method.

$$Hd = \lim_{x \to 0} \frac{f(\theta + \epsilon d) - f(\theta)}{\epsilon}$$

HF is appealing because it uses linear CG as opposed to non-linear CG. This is because linear CG solves the optimization problem to iteratively generate a set of conjugate directions along which line search is performed exactly, while the directions generated by the non-linear CG don't remain conjugate in the higher iterations and line search is performed inexactly.

Modifications to HF

- **Damping** : HF method identifies directions of low curvature. If such a direction is found then CG will tend to move very far along this direction and may possibly move out of region where 2^{nd} order approximation is weak. The damping parameter i.e $\lambda||d^2||$ controls how conservative the approximation is for a given direction d. One such method is to compute λ using ρ (reduction ratio)

$$\rho = \frac{f(\theta + p) - f(\theta)}{q_\theta(p) - q_\theta(0)}$$

- **Efficient computation of matrix-vector products** : An alternative is to Gaussian-newton matrix G in place of Hessian H to compute products like Hd. An advantage of this method is that the G is guaranteed to be positive semi-definite even when undamped, which guarantees that CG will work for any value of damping parameter. Some variants of HS perform a check for negative curvature directions and if this is the condition then they run a different algorithm. However using Gauss-Newton matrix resulted in better search directions and it's also memory and computationally efficient.
- **Handling large data sets** : The computational costs associated with the computing the matrix-vector products grow linearly with data set. An alternative is to convert HF into an online-learning algorithm, where we use large mini-batches whose size grows as the number of iterations increase. This isn't costly as the the number of iterations taken by HF to converge are far less than the steps taken by gradient descent.
- **Preconditioning CG** : The general idea behind preconditioning procedure for iterative algorithms is to transform the ill-conditioned system $Ax = b$ in such a way that we obtain

an equivalent system $\hat{A}\hat{x} = \hat{b}$ for which the iterative method converges faster. This involves linear transformation of variables $\hat{x} = Cx$ and then optimizing the new objective. This new objective can have better curvature properties as compared to original objective depending on the matrix C.

- **Random Initialization** : HF is not completely unaffected by random initialization but it's much more robust than the 1^{st} order techniques. A technique that was found to work better was sparse initialization, where they limit the number of non-zero incoming weights for each unit. This also partially solves the problem of saturation of gradient in neural network.

3.2 Hessian-Free for RNNs [2]

During the training of RNNs with Hessian-Free optimization, when the damping hyperparameter λ was small, there was a rapid decrease in the accuracy of the quadratic approximation as δ_n was driven by conjugate gradient descent towards its optimum. Thus the Levenburg-Marquardt [3] heuristics would compensate by adjusting λ to be much larger. Unfortunately, it was observed that the parameters seemed to approach a bad local minimum where little-to-no long-term information was being propagated through the hidden states.

One possible explanation is that for large values of λ, any Tikhonov-damped 2nd-order optimization approach will behave similarly to a 1st-order approach and crucial low-curvature directions whose associated curvature is significantly smaller than λ will be effectively masked-out.

Thus, the hypothesis for RNNs is that for certain small changes in a parameter θ, there can be large and highly non-linear changes in the hidden state sequence h, and these arent accurately approximated.

This leads to the usage of a novel damping called "structural damping", that can penalize those directions in parameter space which can lead to large changes in the hidden-state sequence. The damping term $\lambda R_{\theta_n}(\theta)$ is given by:

$$\lambda R_{\theta_n}(\theta) = \lambda \left(\frac{1}{2} ||\delta_n||^2 + \mu S_{\theta_n}(\theta) \right)$$

$$S_{\theta_n}(\theta) \equiv D(h(\theta), h(\theta))$$

where D is a distance function similar to the loss, and $S_{\theta_n}(\theta)$ is a function which quantifies change in the hidden units as a function of the change in parameters.

Calculating the value of S is done by approximating it with a quadratic function (similar to the Hessian-Free Newton's approx. for the loss) at θ_n:

$$S(\theta) = S(\theta_n) + (\theta - \theta_n)^\intercal \frac{dS}{d\theta}\Big|_{\theta=\theta_n} + \frac{1}{2}(\theta - \theta_n)^\intercal G_S (\theta - \theta_n)$$

where G_S is the Gauss-Newton matrix for $S(\theta)$. As D is a distance function, $D(h(\theta), h(\theta_n))$ is minimized at $\theta = \theta_n$ with value θ. This implies that:

$$D(h(\theta), h(\theta_n)) = 0 \text{ and } \frac{dD(h(\theta), h(\theta_n))}{dh(\theta)} = 0 \text{ at } \theta = \theta_n$$

$$\therefore S(\theta) = \frac{1}{2}(\theta - \theta_n)^\intercal G_S (\theta - \theta_n)$$

$$\therefore \lambda R_{\theta_n}(\theta) = \lambda \left(\frac{1}{2} ||\delta_n||^2 + \mu \frac{1}{2}(\theta - \theta_n)^\intercal G_S (\theta - \theta_n) \right)$$

3.3 AdaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs [4]

- The changes proposed by the authors in standard L-BFGS are
  $s_t = \overline{w_t} - \overline{w_{t-1}}$ where $\overline{w_t} = \frac{1}{L} \sum_{i=(t-1)L}^{tL} w_i$
  $y_t = \Delta^2_{H_t} F(\overline{w_t}) s_t$

- s is based on average of iterations over 2L iterations.
- y is not computed using gradients at all. It is computed using a Hessian vector product representing approximate curvature along s.
- Recent results have shown that **Hessian Fischer Information Matrix** gives better estimates of curvature problem.
  $F(w) = E_x[\Delta f_x(w) \Delta f_x(w)^T]$
  $\hat{F}(w) = \frac{1}{|H|} \sum_{i \in H} \Delta_i f(w) \Delta_i f(w)^T$ which is the eFIM.

Initial LBFGS initialisation

- The authors have avoided the use of standard Hessian initialization $H_k^0 = \frac{s_k^T y_k}{y_k^Y y_k} I$ in RNNs for two major reasons namely the problem of vanishing/exploding gradients and due to the fact that s and y are noisy estimates of true iterates.
- The initialization the authors have proposed is $[H_k^0]_{ii} = \frac{1}{\sqrt{\sum_{j=0}^{k} [\nabla f(w_j)]_i^2 + \epsilon}}$

Step Quality Control

While curvature information can be used to improve convergence rates, noisy curvature information may in fact deteriorate performance.The curvature pairs are stored during each cycle but are skipped if the calculated curvature is small.A step is rejected if the function value of the new aggregated point is significantly worse than the previous. In this case,the memory of L-BFGS which allows the algorithm to preclude the deteriorating effect of any stored curvature pairs.

Results
Compared performance of **AdaQN** along with **Adam** and **Adagrad** for two Language Modeling tasks:Word Level LM and character level LM. AdaQN presents a non-trivial improvement over both Adagrad and Adam on all tasks with tanh activation functionOn the character-level task with ReLU activation,adaQN performed better than Adam but worse than Adagrad.

3.4 Sum of functions Optimizer [5]

- The current known methods treat the Hessian of a subfunction on some dataset as a noisy approximation to the full Hessian of a given sum of functions. Now in order to reduce noise we try various methods like regularization and updates on mini-batches.To effectively reduce noise this method requires large mini-batches which require a lot of time since each update step requires evaluation of many different subfunctions or the method gives a model which is overfitted.
- Instead of doing this Dickstein et al. tries to treat the full Hessian of each subfunction as a direct target for estimation, where the sum of all subfunctions yields the original function. This results in separate quadratic approximation of each subfunction which we can effectively use.
- They try to combine the benefits of stochastic and quasi-newton optimization techniques to improve optimization time.
- Suppose we are given some function F(x) whic can be expressed as sum of functions $f_i(x)$ such that,$F(x) = \sum_{i=1}^{N} f_i(x)$. We want to minimize this sum of subfunctions.
- Let $G^t(x) = \sum_{i=1}^{N} g_i^t(x)$, where each $g_i^t(x)$ is a quadratic approximation to $f_i(x)$ at the learning iteration t.
- We know that g_i^{t-1}'s are quadratic,and as we can exactly find the argmin value of a quadratic functions,we can find the exact argmin value of G^{t-1} using the Newton update equation:

$$x^t = x^{t-1} - \eta^t (\mathbf{H}^{t-1})^{-1} \frac{\partial G^{t-1}(x)}{\partial x}$$

- We perform a stochastic update by choosing a random i, and then we update the corresponding subfunction g_i^t using:

$$g_i^t(x) = f_i(x^t) + (x - x^t)^\top f_i^{'}(x^t) + \frac{1}{2}(x - x^t)^\top \mathbf{H}_i^t(x - x^t)$$
$$\mathbf{H}^t = \sum_i \mathbf{H}_i^t$$

- We compute an online approximation of the hessian of the given subfunction g_i^t using L-BFGS. Also the Hessian of the approximated functions can be eexpressed as the sum of hessians of each of the approximated subfunction $g_i^t(x)$.

Reducing the dataset to a smaller subspace

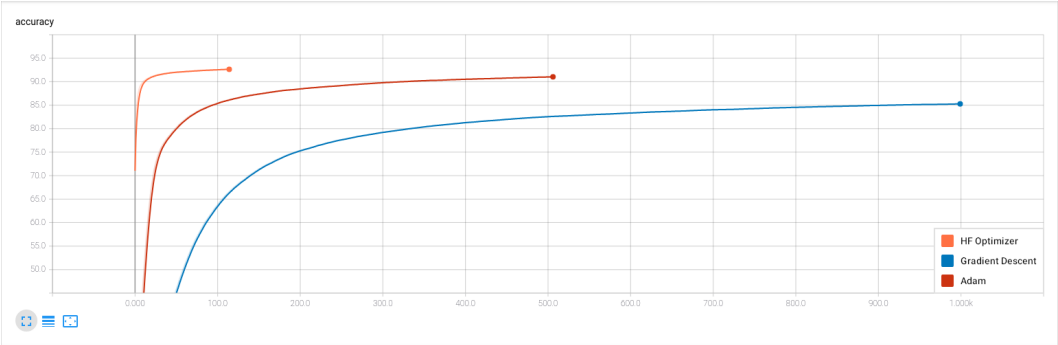
- The dimension of each $\mathbf{H}_i$ is quadratic in terms of dimensions of vector **x** and very large and the calculations take a large amount of time. To make the computation tractable the history is stored & updated in a low dimensional subspace with size between some K_{min} & K_{max} which we can set according to our needs. We can take K_{min} & K_{max} to be 2N and 3N respectively.

- At each step, we find the component of the most recent gradient we have outside of the existing subspace and add it to the current subspace.

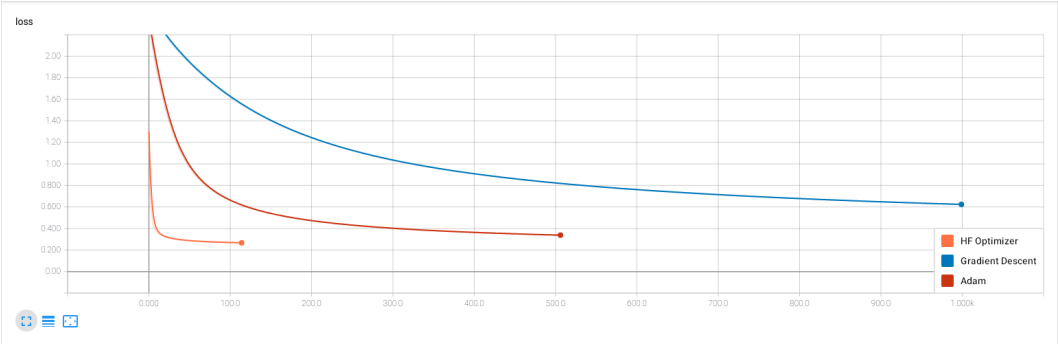
- To prevent the size of the subspace from becoming too large, it is collapsed when the dimension goes beyond K_{max} using QR decomposition on the most recent gradient and position. We then map the vectors in the original subspace into the new ones and continue.

3.5 Experimental Results

The Hessian-Free optimizer was tested against Adam [6] and minibatch gradient descent on a single hidden layer network that recognizes MNIST [7] digits. The model was tested on a laptop running Arch Linux, with an eight-core Intel® Core™i7 CPU and a single NVIDIA GeForce® GTX 960M (4GB) graphics card. Tensorflow v1.12 [8] was used for creating the HFOptimizer class and testing on the MNIST network. Tensorboard [8] was used for visualizing the computation graph and plotting the loss/accuracy curves w.r.t. no. of steps.



(a) Accuracy



(b) Loss

Figure 1: Results of Hessian-Free on a single hidden layer network

On running the code, it was observed that the Hessian-Free optimizer roughly takes thrice the time that Adam takes on a single step, however, this may be due to the fact that our implementation of the HFOptimizer class is not fully parallelized for GPUs.

A computation graph is also provided for the HF optimizer on page 7.

References

[1] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (USA), pp. 735–742, Omnipress, 2010.

[2] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (L. Getoor and T. Scheffer, eds.), ICML '11, (New York, NY, USA), pp. 1033–1040, ACM, June 2011.

[3] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[4] N. Shirish Keskar and A. S. Berahas, "adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs," *ArXiv e-prints*, p. arXiv:1511.01169, Nov. 2015.

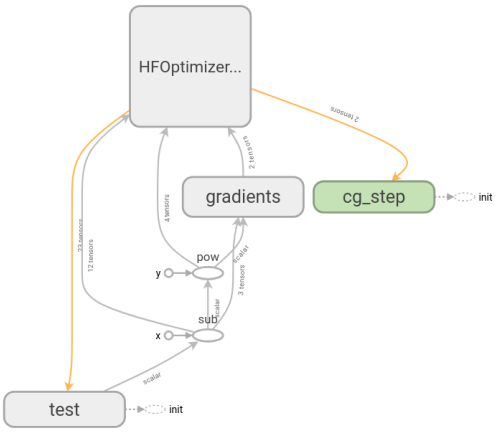
[5] J. Sohl-Dickstein, B. Poole, and S. Ganguli, "An adaptive low dimensional quasi-newton sum of functions optimizer," *CoRR*, vol. abs/1311.2115, 2013.

[6] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv e-prints*, p. arXiv:1412.6980, Dec. 2014.

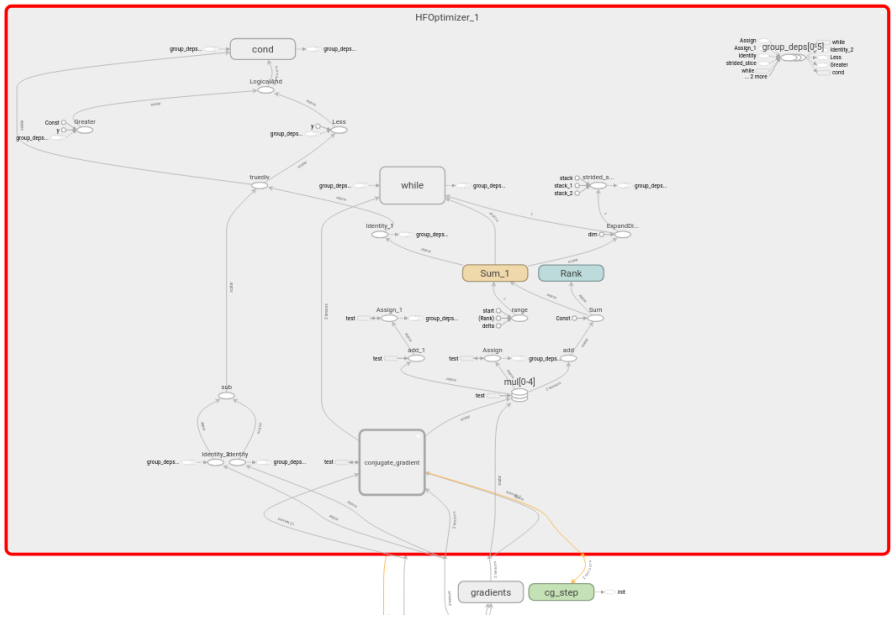
[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah,

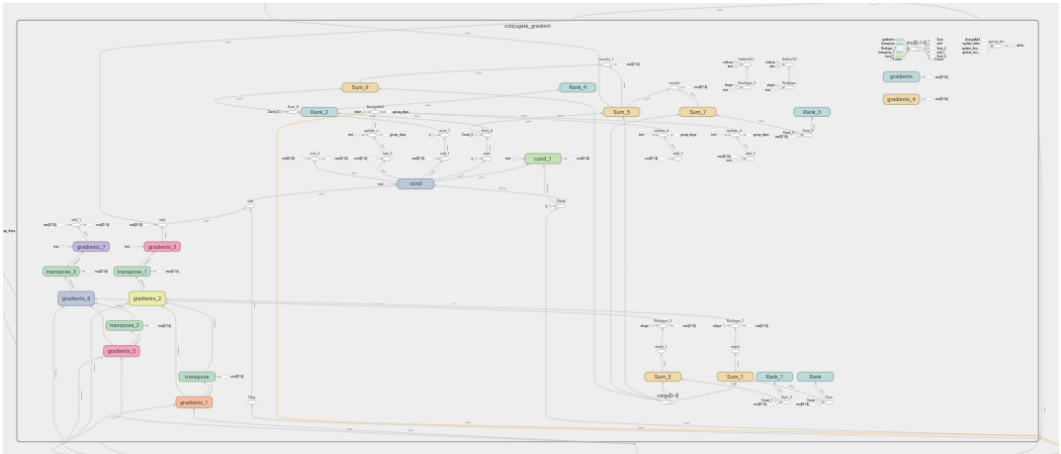
M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.



(a) Overall graph



(b) HF Optimizer



(c) Conjugate Gradient Descent

Figure 2: Computation graphs of Hessian-Free on a simple example