Stochastic Quasi Newton Method

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We implemented stochastic L-BFGS code in pytorch and we followed the algorithm proposed in paper[1]. Changes were made to incorporate stochastic gradient and to handle non-convex functions. After implementation, we analyzed the performance of algorithm with some basic functions and with deep learning applications. We have used the algorithm to optimize the long short term memory network for sentiment analysis application. Numerical results of these experiments are reported to compare the performance of stochastic L-BFGS with state of the art algorithms like SGD and Adam.

1 Motivation

Newton method is best when second order oracle is available and function is convex. But it also incurs computationally heavy operation of matrix inversion. Due to that it requires $O(d^3)$ computation per iteration, where d is dimension of parameter. To take advantage of newton method's quadratic approximation and to get rid of matrix inversion, quasi newton method is used which approximates the inverse of the hessian. BFGS is quite popular quasi newton method.

But BFGS requires $O(d^2)$ computation per iteration and space complexity. In deep learning applications, d in the range of 10^6 is quite normal. It leads to huge memory requirement for storing previous Hessian and computing it. To overcome this, L-BFGS is used which approximates Hessian from beginning in each iteration. It needs to store y and s for last p iteration. It requires $O(2dp)$ storage compared to $O(d^2)$. P is usually in the range of 20 to 100 so there is significant reduction in storage space. In the computational complexity also, L-BFGS takes $O(4dp)$ compared to $O(d^2)$ for BFGS.

Pytorch is extensively used for deep learning applications. To get superior performance on deep learning applications, it is necessary that optimization algorithm has stochastic update. Hence, it would be useful for the pytorch community to incorporate stochastic and non-convex update in pytorch's existing L-BFGS algorithm.

2 Algorithm

2.1 Quasi Newton Methods

In the deterministic optimization setting, Quasi-Newton methods usually employ the following updates:

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k) \quad \text{or} \quad x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k) \tag{1}$$

where B_k is an approximation to the Hessian matrix $\nabla^2 f(x_k)$ at x_k, or H_k is an approximation to $[\nabla^2 f(x_k)]^{-1}$.

The most widely used quasi newton method updates B_k via

$$B_k = B_{k-1} + \frac{y_{k-1}y_{k-1}^T}{s_{k-1}^T y_{k-1}} - \frac{B_{k-1}s_{k-1}s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1}s_{k-1}} \tag{2}$$

where $s_{k-1} = x^{(k)} - x^{(k-1)}, y_{k-1} = \nabla f(x^k) - \nabla f(x^{(k-1)})$.

By using the Sherman-Morrison-Woodbury formula, H_k is

$$H_k = (I - \rho_{k-1}s_{k-1}y_{k-1}^T)H_{k-1}(I - \rho_{k-1}y_{k-1}s_{k-1}^T) + \rho_{k-1}s_{k-1}s_{k-1}^T, \tag{3}$$

where $\rho_{k-1} = 1/(s_{k-1}^T y_{k-1})$.

2.2 Stochastic Quasi Newton

The key issues in designing stochastic quasi newton methods for non-convex problem resides in the difficulty in preserving the positive-definiteness of B_k (and H_k), because of the non-convexity of the problem and the presence of randomness in estimating the gradient. We know that BFGS update 2 preserves the positive-definiteness of B_k as long as the curvature condition

$$s_{k-1}^T y_{k-1} > 0 \tag{4}$$

holds, which is sure for strongly convex functions. For the non-convex problems, the condition4 can be satisfied by performing a line search. However, doing this is no longer feasible in the stochastic setting, because exact function values and gradient information are not available. The important issue is how to preserve the positive definiteness of B_k (or H_k) without line search.

2.2.1 Curvature Condition Update

To address the issue, authors proposed the following update for y_k. The iterate difference is still defined as $s_{k-1} = x_k - x_{k-1}$. We then define

$$\bar{y}_{k-1} = \hat{\theta}_{k-1}y_{k-1} + (1 - \theta_{k-1})B_{k-1}s_{k-1}, \tag{5}$$

where

$$\hat{\theta}_{k-1} = \begin{cases} \frac{0.75 s_{k-1}^T B_{k-1}s_{k-1}}{s_{k-1}^T B_{k-1}s_{k-1} - s_{k-1}^T y_{k-1}} & \text{if } s_{k-1}^T y_{k-1} < 0.25 s_{k-1}^T B_{k-1}s_{k-1}, \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

$$s_{k-1}^T \bar{y}_{k-1} = \hat{\theta}_{k-1}(s_{k-1}^T y_{k-1} - s_{k-1}^T B_{k-1}s_{k-1}) + s_{k-1}^T B_{k-1}s_{k-1} \tag{7}$$

$$s_{k-1}^T \bar{y}_{k-1} = \begin{cases} 0.25 s_{k-1}^T B_{k-1}s_{k-1} & if\ s_{k-1}^T y_{k-1} < 0.25 s_{k-1}^T B_{k-1}s_{k-1}, \\ s_{k-1}^T y_{k-1} & \text{otherwise} \end{cases} \tag{8}$$

which implies $s_{k-1}^T \bar{y}_{k-1} \geq 0.25 s_{k-1}^T B_{k-1}s_{k-1}$. Therefore, if $B_{k-1} \succ 0$, it follows that $\rho_{k-1} > 0$. This implies

$$z^T H_k z = z^T(I - \rho_{k-1}s_{k-1}\bar{y}_{k-1}^T)H_{k-1}(I - \rho_{k-1}\bar{y}_{k-1}s_{k-1}^T)z + \rho_{k-1}(s_{k-1}^T z)^2 > 0 \tag{9}$$

given that $H_{k-1} \succ 0$. It satisfies the positive definiteness of H_k and B_k.

2.2.2 Stochastic Gradient

We generate an auxiliary stochastic gradient at x_k using the sampling from the $(k-1)$-st iteration:

$$\bar{g}_k = \frac{1}{m_{k-1}} \sum_{i=1}^{m_{k-1}} g(x_k, \xi_{k-1,i}) \tag{10}$$

Note that we assume that our SFO can seperate two arguments x_k and ξ_k in the stochastic gradient $g(x_k, \xi_{k-1})$ and generate an output $g(x_k, \xi_{k-1,i})$. The stochastic gradient difference is defined as

$$y_{k-1} := \bar{g}_k - g_{k-1} = \frac{\sum_{i=1}^{m_{k-1}}(g(x_k, \xi_{k-1,i}) - g(x_{k-1}, \xi_{k-1,i}))}{m_{k-1}} \tag{11}$$

2.3 Stochastic damped LBFGS method

In this section, we explain an efficient way to compute $H_k g_k$ without generating H_k explicitly. Before doing this, we first describe a stochastic damped BFGS method as follows.

Computing H_k by stochastic damped BFGS update and computing the step direction $H_k g_k$ requires $O(d^2)$ multiplications. This is costly if d is large. We can adopt the L-BFGS method which is as follows.

Given an initial estimate $H_{k,0} \in \mathbb{R}^{nxn}$ of the inverse Hessian at the current iterate x_k and two sequences $s_j, y_j, j = k-p, .., k-1$, where p is the memory size, the L-BFGS method updates $H_{k,i}$ recursively as

$$H_{k,i} = (I - \rho_j s_j y_j^T) H_{k,i-1} (I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T, j = k - (p - i + 1); i = 1, ..., p, \quad (12)$$

where $\rho_j = (s_j^T y_j)^- 1$.

The output $H_{k,p}$ is then used as the estimate of the inverse Hessian at x_k to compute the search direction at the kth iteration.It is proved in the paper[1] that if the sequence of the pairs s_j, y_j satisfies the curvature condition $s_j^T y_j > 0, j = k-1, ..., k-p$, then $H_{k,p}$ is positive definite provided that $H_{k,0}$ is positive definite.

An efficient choice in the standard L-BFGS method is to set $H_{k,0} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} I$. Since $s_{k-1}^T y_{k-1}$ may not be positive for non convex problems, we set

$$H_{k,0} = \gamma_k^{-1} I, \text{where } \gamma_k = \max\{\frac{y_{k-1}^T y_{k-1}}{s_{k-1}^T y_{k-1}}, \delta\} \geq \delta, \quad (13)$$

where $\delta > 0$ is a given constant.

The algorithm for the step computation in SdLBFGS is given below:

Input: Let x_k be a current iterate. Given the stochastic gradient g_{k-1} at iterate x_{k-1}, the random variable ξ_{k-1}, the batch size m_{k-1},$s_j, \bar{y}_j$ and ρ_j, $j = k-p, ..., k-2$, and $u_0 = g_k$.

Output: $H_k g_k = v_p$

```
1. Set $s_{k-1} = x_k - x_{k-1}$ and calculate $y_{k-1}$ through (11)
2. Calculate $\gamma_k$ through (13)
3. Calculate $\bar{y}_{k-1}$ through (5) and $\rho_{k-1} = (s_{k-1}^T \bar{y}_{k-1})^{-1}$
4. for $i = 0, ...., min\{p, k-1\} - 1$ do
5.     Calculate $\mu_i = \rho_{k-i-1} u_i^T s_{k-i-1}$
6.     Calculate $u_{i+1} = u_i - \mu_i \bar{y}_{k-i-1}$
7. end for
8. Calculate $v_0 = \gamma_k^{-1} u_p$
9. for $i = 0, ...., min\{p, k-1\} - 1$ do
10.    Calculate $l_i = \rho_{k-p+i} v_i^T \bar{y}_{k-p+i}$
11.    Calculate $v_{i+1} = v_i + (\mu_{p-i+1} - l_i) s_{k-p+i}$
12. end for
```

2.4 SQN method for stochastic non convex optimization Algorithm

Input: Given $x_1 \in \mathbb{R}^n$, a positive definite matrix $H_1 \in \mathbb{R}^{nxn}$, batch sizes $\{m_k\}_{k\geq1}$, and step sizes $\{\alpha_k\}_{k\geq1}$

```
1. for $k = 1, 2, ....$ do
2.     Calculate $g_k = \frac{1}{m_k} \sum_{i=1}^{m_k} g(x_k, \xi_{k,i})$
3.     Generate a postive definite Hessian inverse approximation $H_k$
4.     Calculate $x_{k+1} = x_k - \alpha_k H_k g_k$
5. end for
```

2.5 Why scalar implementation of $H_{k,0}$ is fine?

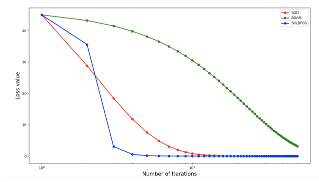
L-BFGS algorithm approximate H_k from initial hessian H_0 unlike BFGS where it updates H_k from H_{k-1} . For L-BFGS, if we take $H_{k,0} = \gamma_k I$ it gives good performance. Where $\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$. Now this $H_{k,0}$ is used only once in multiplying with u_p. So we can do element wise multiplication with u_p and we can get rid of the matrix multiplication.

Actually, that is where the power of L-BFGS lies. It directly gives us the direction $-(H_k * g_k)$ by running the two loop of above algorithm. And that is how we can avoid matrix calculations and just do vector multiplications.

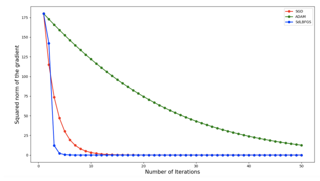
3 Experiments, Results and Observations

3.1 Basic Functions

3.1.1 Quadratic Function



(a) Loss vs iterations



(b) Norm square of Gradient vs iterations

Figure 1: Quadratic function($(x-9)^2 + (y-6)^2$; lr(SGD = 0.1/$\sqrt{t}, LBFGS =$1/$\sqrt{t}$, ADAM = 0.1/$\sqrt{t}$)

For quadratic function, stochastic LBFGS converges to optimal value faster than SGD and ADAM as expected. Because it does the quadratic approximation while other two does the linear approximation.

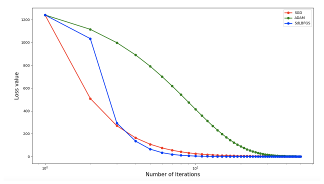
3.1.2 Cubic Function

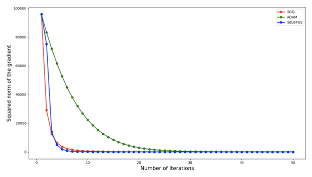
For cubic function(2) also all three algorithms behaves similarly. But when it comes to time, L-BFGS takes maximum time per iteration compared to SGD and ADAM which is expected because per iteration computational complexity is more for L-BFGS than SGD and ADAM.

3.2 Experiments with Deep Learning Models

An LSTM network for sentiment analysis was used for checking the performance of the different optimization algorithms. Along with issues of access to high computing power, we also faced the problems of exploding gradients for the original LBFGS implementation in Pytorch. But, there was no such issue with our implementation, as the checks and updates for the non-convex loss function highlighted in this work, ensured the convergence of our model. For the LSTM implementation, the loss function used is Negative Log Likelihood Loss. The data for our experiments were generated using torch text which is provided by Pytorch for the Sentiment Analysis Task.

In this section, we compare our SdLBFGS model with Adam, SGD and the original LBFGS code from Pytorch. For, SdLBFGS, we took the learning rate of $1/\sqrt{t}$ and a small value of $1e-3$ for


(a) Loss vs iterations


(b) Norm square of gradient vs iterations

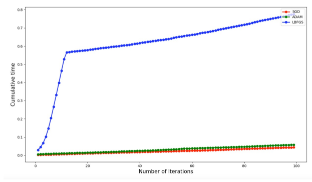

(c) Time vs iterations

Figure 2: Cubic function($(x+3)^3+(y+4)^3$; lr(SGD = $0.01/\sqrt{t}, LBFGS$ =$1/\sqrt{t}$, ADAM = $0.3/\sqrt{t}$))

Adam and SGD. Learning rate of $1/\sqrt{t}$ were found to blow up the gradients of SGD and Adam in the initial iterations and were not found to converge well. Therefore, the small learning rate was used. Figure 3, shows the comparison of average loss computed at each epoch for SdLBFGS, SGD, Adam. The old LBFGS code of Pytorch becomes very high in loss (infinity or nan in our code) after the second epoch. Therefore, we do not report the loss value for it. Adam is performing the best amongst the three. The accuracy of SGD is either lesser or almost as good as SdLBFGS for each epoch, whereas Adam is clearly the better performer. Figure 6 shows the variation of the SdLBFGS loss with varying history size p. The loss went to infinite for $p=5$, but improved for $p=10$ and $p=15$.This is because the larger history size, is good for hessian approximation. Figure 7 shows the time taken by the three algorithms to converge. While, SGD and Adam are faster than SdLBFGS. SdLBFGS is much faster than the LBFGS implementation in pytorch.

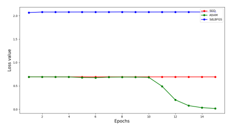


Figure 3: Loss averaged at each epoch

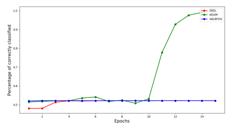


Figure 4: Percentage of correctly classified data samples

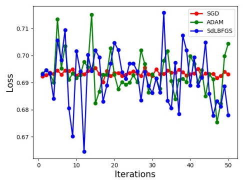


Figure 5: Loss vs iterations

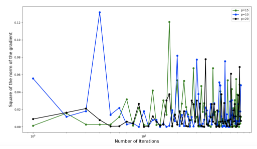


Figure 6: Norm of grad square vs iter for diff. memory size

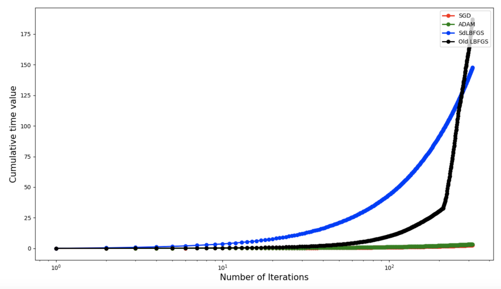


Figure 7: Cumulative Computation time vs iterations

4 Summary

L-BFGS works better when higher epsilon suboptimality is required. It converges to optimum in fewer iterations than SGD and ADAM. But in overall time, SGD and ADAM works far better because per iteration complexity is $O(d^2)$ for L-BFGS while it is $O(d)$ for SGD. So in deep learning application, which is inherently computationally intensive and where higher accuracy is not required, L-BFGS is not desirable algorithm to use. Simple algorithm SGD gives much better performance in deep learning application.

5 Link for Code

Link of github repository containing our code is as follow:

`https://github.com/RGaonkar/Stochastic-LBFGS-Pytorch`

6 References

[1] Wang, Xiao, et al. "Stochastic quasi-Newton methods for nonconvex stochastic optimization." SIAM Journal on Optimization 27.2 (2017): 927-956

[2] Wright, Stephen, and Jorge Nocedal. "Numerical optimization." Springer Science 35.67-68 (1999): 7

[3] http://pytorch.org/tutorials/

[4] http://ruder.io/open