# A Note for L-BFGS Method on Large-Scale Optimization

Jiazheng Liu

April 30, 2018

This is a summary about L-BFGS method on Large-Scale Optimization.Main part of this note is related to arXiv:1401.7020v2, by R.H. Byrd, S.L.Hansen, Jorge Nocedal, Y. Singer.

## 1 Background

### 1.1 A Typical Example

In this section, we will take **binary classification** as a concrete example, to help reader know in which cases a optimal model could be used, why the stochastic optimization shall be used and why the data in such problem must be necessary.

A binary classification is, the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule. There are many ways to implement a classification rule, and most common one is using logistic regression.

Suppose now we have a batch of data. It contains basically two parts: input(may be multiple dimension, in data science ,they are called, features), and **single 0-1 output**, namely the output $z \in \{0,1\}$. The challenging problem is to construct a model, using the data, to predict whether a output is 0 or 1, based on newly coming input.

Pick the logistic regression model,

$$c(w;x_i) = \frac{1}{1 + e^{-x_i^T w}} \tag{1}$$

where the $x_i$ is an input data, $w$ is the parameter. The function has a very good differential property and its range is very ideal.
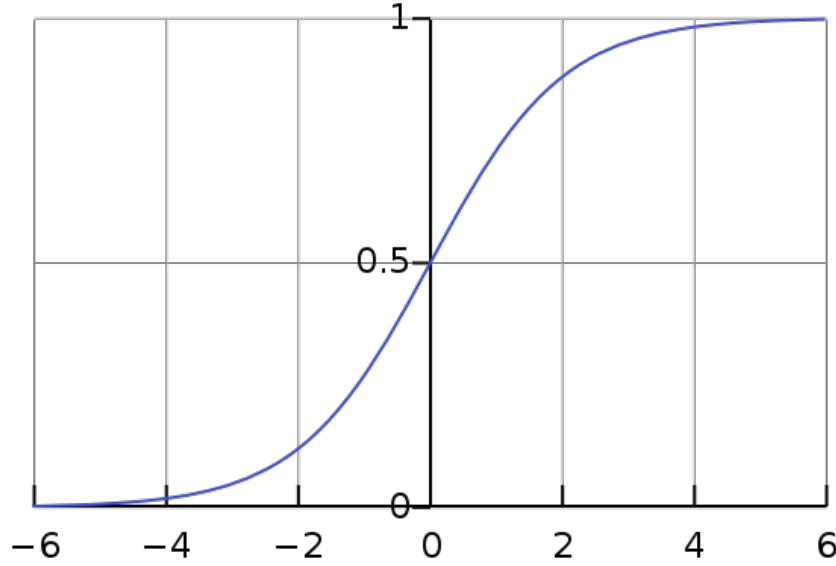
Figure 1: Logistic Curve

A very simple rule is, based on the input vector $x_i$, if the output $c$ greater than 0.5, it will be regarded as 1, otherwise it should be 0. This function, combined with this rule, forms a binary classification rule.

After we have found a good model to conduct predictions, a very natural problem we are facing to is, how to select the parameter? We need to construct some function to evaluate which sets of parameters are good, which are not. Such function are normally called lost function, in tradition optimization, it is called penalty function. The regular cost function associated with logistic regression is cross entropy function:

$$f(w; x_i, z_i) = z_i log(c(w; x_i)) + (1 - z_i) log(1 - c(w; x_i)) \qquad (2)$$

where $x_i$ is still the input vector, and the $z_i \in \{0, 1\}$ is the output. The cost function measures the goodness of parameter. A good parameter should make the value as small as possible, now we can see there is a primal unconstrained optimal model:

$$min_{w \in R^n} f(w; x_i, z_i) \qquad (3)$$

We nearly finish this, except one thing to consider. Now we have a batch of those input-output pairs, we need to find a set of parameter to satisfy conditions in a larger set.

An assumption holds for the data, where the input data must obey the same distribution. (Actually it is a very normal phenomena). In such case, we can transfer our model to a stochastic one:

$$min_{w \in R^n} F(w) = \mathbf{E}[f(w; \zeta)] \qquad (4)$$

In this framework, input-output pairs are "parameter", the real parameters are "variable". And without the data, we can do nothing to predict a good parameter, and the model itself. And that's the reason why data is very important in this kind of problem.

## 1.2 A Following Remark

We provide readers a concrete example. We hope readers can see the framework behind—-such problem are called supervised learning. Imagine there is a strict teacher, each time you get input data, you then make a prediction, and the teacher will tell you whether your answer is correct or not. If it is correct, nothing will happen, if not, well, you may face some penalty. That's reason why it is called supervised learning—-each time you will get someone supervises you.

The basic idea of supervised learning is:

Find a reasonable model – –Define a cost function– Define a optimal objective – Get the input-output data pair – Conduct optimal method – Get the best parameter(minimizer) – Finish the training and ready to make a prediction.

From this process we can observe that, a good method to solve optimization problem is very critical.

In many applications of supervised learning and other relative fields, one constructs very large models from massive amounts of training data. Learning such models imposes high computational and memory demands on the optimization algorithms employed to learn the models. At the same time, it still needs the accuracy and speed. How to make a trade off between speed and accuracy is an important topic in large-scale optimization.

The most famous and most successful optimal algorithm is (first-order) stochastic gradient method.

## 1.3 Why first order

Since the size of data is huge, if a program intends to apply a second-order method, the costing time will be extremely long, which is unacceptable in practice.

## 1.4 Why stochastic gradient descent (SGD)

As we mentioned above, the input data indeed has uncertainty, it is a natural way to regard the objective problem as:

$$min_{w \in R^n} F(w) = \mathbf{E}[f(w; \zeta)] \tag{5}$$

where $\zeta$ is a parameter list, and a random vector at the same time. This expression is very common in machine learning application. $f$ typically takes the form

$$f(w; \zeta) = f(w; x_i, z_i) = l(h(w; x_i), z_i) \tag{6}$$

where $l$ is lost function into $R_+$, and and $h$ is a prediction model parametrized by $w$.

Thus the gradient has the form:

$$F(w) = \frac{1}{N} \sum_1^N f(w; x_i, z_i) \tag{7}$$

In learning applications with very large amounts of training data, it is common to use a mini-batch stochastic gradient based on $b := \|S\| << N$ input-output instances.

$$\bar{\nabla} F(w) = \frac{1}{b} \sum_{i \in S} \nabla f(w; x_i, z_i) \tag{8}$$

And this kind of implement is called batch gradient method.

## 1.5 why L-BFGS

L-BFGS method can use second-order information on local area, which means it has much more possibility to have a faster convergence rate. At same time, it is designed for a limited memory situation, and that makes it suitable for modifying into large-scale computation situation.

## 2 Idea and Algorithm

In Byrd,s paper, the main idea is very simple. If one wants to use Hessian matrix but not hope to make program too slow, what he/she can do? -Just slow down the updating frequency. They choose to decouple the stochastic gradient and curvature estimate calculations.

---

**Algorithm 1** Stochastic Quasi-Newton Method (SQN)

**Input:** initial parameters $w^1$, positive integers $M, L$, and step-length sequence $\alpha^k > 0$

---

1: Set $t = -1$ ▷ Records number of correction pairs currently computed
2: $\bar{w}_t = 0$
3: **for** $k = 1, \ldots,$ **do**
4:    Choose a sample $\mathcal{S} \subset \{1, 2, \cdots, N\}$
5:    Calculate stochastic gradient $\widehat{\nabla} F(w^k)$ as defined in (1.4)
6:    $\bar{w}_t = \bar{w}_t + w^k$
7:    **if** $k \leq 2L$ **then**
8:        $w^{k+1} = w^k - \alpha^k \widehat{\nabla} F(w^k)$ ▷ Stochastic gradient iteration
9:    **else**
10:        $w^{k+1} = w^k - \alpha^k H_t \widehat{\nabla} F(w^k)$, where $H_t$ is defined by Algorithm 2
11:    **end if**
12:    **if** $\mathrm{mod}(k, L) = 0$ **then** ▷ Compute correction pairs every $L$ iterations
13:        $t = t + 1$
14:        $\bar{w}_t = \bar{w}_t / L$
15:        **if** $t > 0$ **then**
16:            Choose a sample $\mathcal{S}_H \subset \{1, \cdots, N\}$ to define $\widehat{\nabla}^2 F(\bar{w}_t)$ by (2.3)
17:            Compute $s_t = (\bar{w}_t - \bar{w}_{t-1})$, $y_t = \widehat{\nabla}^2 F(\bar{w}_t)(\bar{w}_t - \bar{w}_{t-1})$ ▷ correction pairs
18:        **end if**
19:        $\bar{w}_t = 0$
20:    **end if**
21: **end for**

---

Figure 2

In the algorithm, (1.4) is

$$\bar{\nabla} F(w) = \frac{1}{b} \sum_{i \in S} \nabla f(w; x_i, z_i)$$

(2.3) is

$$\bar{\nabla}^2 F(w) := \frac{1}{b_H} \sum_{i \in S_H} \nabla^2 f(w; x_i, z_i)$$

$S$ and $S_H$ are samples, $b$ and $b_H$ are their sizes.

---

**Algorithm 2** Hessian Updating

---

**Input:** Updating counter $t$, memory parameter $M$, and correction pairs $(s_j, y_j)$, $j = t - \tilde{m} + 1, \ldots t$, where $\tilde{m} = \min\{t, M\}$.

**Output:** new matrix $H_t$

1: Set $H = (s_t^T y_t)/(y_t^T y_t)I$, where $s_t$ and $y_t$ are computed in Step 17 of Algorithm 1.
2: **for** $j = t - \tilde{m} + 1, \ldots, t$ **do**
3:     $\rho_j = 1/y_j^T s_j$.
4:     Apply BFGS formula:

$$H \leftarrow (I - \rho_j s_j y_j^T)H(I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T \tag{2.4}$$

5: **end for**
6: **return** $H_t \leftarrow H$

---

Figure 3

# 3 Computation Cost

This part what I get is slightly different from the original paper.
Compare SGD and the new stochastic quasi-Newton method.

$$\text{SGD}: \quad w^{k+1} = w^k - \frac{\beta}{k}\bar{\nabla}F(w^k) \tag{9}$$

$$\text{SQN}: \quad w^{k+1} = w^k - \frac{\beta}{k}H_t \cdot \bar{\nabla}F(w^k) \tag{10}$$

In SGD, the mainly computationally hard part is to generate the each gradient, we have $n$ dimension of variable, then we do a summation (for $b$ different gradients), the main cost is $bn$.

In SQN, the main part is divided into three part:

- summation of gradient $bn$.

- computation of $y_t = \bar{\nabla}^2 F(\bar{w}_t)(\bar{w}_t - w_{t-1}^-)$. Since $\bar{\nabla}^2 F(w) := \frac{1}{b_H}\sum_{i \in S_H}\nabla^2 f(w; x_i, z_i)$, we get it approximately will cost $b_H \cdot n^2$

- generating $H_t$, we pick $m = M$ as iteration time, $M$ denotes the memory limit. Thanks to the symmetry of matrix updating, we can find an algorithm computing $H \leftarrow (I - \rho_j s_j y_j^T)H(I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T$ in $n^2$. This part we have $M \cdot n^2$.

Thanks to the low frequency of updating Hessian matrix, the total cost will be :

$$bn + \frac{b_H n^2 + M n^2}{L} \tag{11}$$

# 4 Convergence Analysis

We assume that the objective function F is strongly convex and twice continuously differentiable. The first assumption may appear to be unduly strong because in certain settings (such as logistic regression) the component functions $f(w; x_i; z_i)$ are

5

convex, but not strongly convex. However, since the lack of strong convexity can lead to very slow convergence, it is common in practice to either add an $l_2$ regularization term, or choose the initial point (or employ some other mechanism) to ensure that the iterates remain in a region where the F is strongly convex. If regularization is used, the objective function takes the form:

$$\frac{1}{2}||w||^2 + \frac{1}{N}\sum_{1}^{N} f(w;x_i,z_i) \tag{12}$$

and sample Hessian is :

$$\sigma I + \frac{1}{b_H}\sum_{i\in S_H} \nabla^2 f(w;x_i,z_i) \tag{13}$$

Under some specific condition:
**Assumptions 1**

- The objective function F is twice continuously differentiable.

- There exist positive constants $\lambda$ and $\Lambda$ such that $\lambda I \prec \bar{\nabla}^2 F(w) \prec \Lambda I$, for all $w \in R^n$ , and for all $S_H \subset 1,...,N$

**Theorem .1.** *If Assumptions 1 hold, there exist constants $0 < \mu_1 \leq \mu_2$ such that the Hessian approximations $\{H_t\}$ generated by Algorithm 1 satisfy:*

$$\mu_1 I \prec H_t \prec \mu_2 I \tag{14}$$

Furthermore, we have assumption 2:
**Assumptions 2**

- The objective function F is twice continuously differentiable.

- There exist positive constants $\lambda$ and $\Lambda$ such that $\lambda I \prec \bar{\nabla}^2 F(w) \prec \Lambda I$, for all $w \in R^n$ , and for all $S_H \subset 1,...,N$

- There is a constant $\gamma$ such that, for all $w \in R^n$, $\mathbf{E}_\zeta[||\nabla f(w^k;\zeta)||]^2 \leq \gamma^2$

**Theorem .2.** *Suppose that Assumptions 2 hold. Let wk be the iterates generated by the Newton-like method, where for $k = 1,2,...,$*

$$\mu_1 I \prec H_k \prec \mu_2 I \tag{15}$$

*and,*

$$\alpha^k = \beta/k, \ with, \ \beta > 1/(2\mu_1\lambda). \tag{16}$$

*Then for all $k > 1$,*

$$\mathbf{E}[F(w^k) - F(w^*)] \leq Q(\beta)/k \tag{17}$$

*where,*

$$Q(\beta) = \max\{\frac{\Lambda\mu_2^2\beta^2\gamma^2}{2(2\mu_1\lambda\beta - 1)}, F(w^1) - F(w^*)\} \tag{18}$$

# 5 Improvement: Enhanced Version

This paper has a very complete work,containing a simple but effective method, a cost approximation and convergence rate analysis.

About the improvement, my naive idea is adjust the "window size" of updating Hessian matrix, just like Armijo line search, making the parameter $L$ adjustable.

We can use a line model to measure a "successful" decent and a "not so good " try. In a good case, we do not need to update the Hessian matrix so frequently, however in a bad case, we need to adjust matrix as soon as possible to make objective function escape from saddle point.

---

**Algorithm 1:** Adjustable Length Stochastic Quasi-Newton Method

**Input** : initial parameters $w_1$, positive integers M,L, step-length sequence $\alpha^k > 0$, reduction rate $0 < \eta < 1$, and size rate $\tau > 1$

1 Set t = -1;
2 $\bar{w}_t = 0$;
3 j = 0;
4 **for** *k = 1,..., do* **do**
5      Choose a sample $S \subset \{1,..,N\}$ ;
6      Calculate stochastic gradient $\bar{\nabla} F(w^k)$ as defined in (1.4) ;
7      $\bar{w}_t = \bar{w}_t + w^k$ ;
8      **if** $k < 2L$ **then**
9          $w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k)$ ;
10      **else**
11          $w^{k+1} = w^k - \alpha^k H_t \bar{\nabla} F(w^k)$ ;
12      **end**
13      $j = j + 1$ ;
14      **if** $j = L$ **then**
15          t = t + 1 ;
16          $\bar{w}_t = \bar{w}_t / L$ ;
17          **if** $t > 0$ **then**
18              Choose a sample $S_H \subset \{1,...,N\}$ to define $\bar{\nabla}^2 F(\bar{w}_t)$ ;
19              Compute $s_t = (\bar{w}_t - w_{t-1}^-)$, $y_t = \bar{\nabla}^2 F(\bar{w}_t)(\bar{w}_t - w_{t-1}^-)$ ;
20          **end**
21          $j = 0$ ;
22          Define line model $G_t(w) = \frac{1}{N} \sum_{i \in S}[f(\bar{w}_t; x_i, z_i) + \nabla f(\bar{w}_t; x_i, z_i)(w - \bar{w}_t)]$ ;
23          **if** $F(w_{t-1}^-) - F(\bar{w}_t) \geq \eta(G_{t-1}(w_{t-1}^-) - G_t(\bar{w}_t))$ **then** L = $\tau$L ;
24          **else** L = L / $\tau$ ;
25      **end**
26 **end**

---

The convergence rate and computation cost are basically as same. However owing to the flexible window size, it becomes a bit hard to analysis.

# 6 Numerical Experiments

The section is a brief report of some numerical experiments.

We choose binary classification task, just like we mentioned in section one. In this problem, we need :

- A batch of input-output pairs to train our model

- An **explicit** loss function ,its gradient and its hessian. They should be written fixedly in a class.

- Some critical parameters, controlling the behavior of gradient method and size of window.

## 6.1   Function Class

Function value:

$$f(w;x_i,z_i) = z_i log(c(w;x_i)) + (1-z_i)log(1-c(w;x_i)) \tag{19}$$

$$c(w;x_i) = \frac{1}{1+e^{-x_i^T w}} \tag{20}$$

Gradient:

$$\nabla f(w;x_i,z_i) \;=\; (c(w;x_i) - z_i)x_i \tag{21}$$

Hessian Matrix(with a vector):

$$\nabla^2 f(w;x_i,z_i)s \;=\; c(w;x_i)(1-c(w;x_i))(x_i^T s)x_i \tag{22}$$

## 6.2   Data Set

Data Set 1 : 16-dimension input(features), 1 output(0-1 discrete value), size is 20000

Data Set 2:

## 6.3   Critical Program Parameter

- Learning rate $\alpha$: $w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k)$. (Right now is a constant and simple function, later on an Armijo line search method may be implemented on)

- Memory limit : $M$, determining the maximal iteration of L-BFGS method

- Window size: $L$, determining the frequency when a Hessian matrix shall be updated.

- (In enhanced version) Adjusted parameter associated with window size: $\eta$ and $\tau$ : $F(w_{t-1}^-) - F(\bar{w}_t) \geq \eta(G_{t-1}(w_{t-1}^-) - G_t(\bar{w}_t))$ and $L = \tau L$ or $L = L/\tau$

## 6.4   Result