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

Jiazheng Liu

May 2, 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.

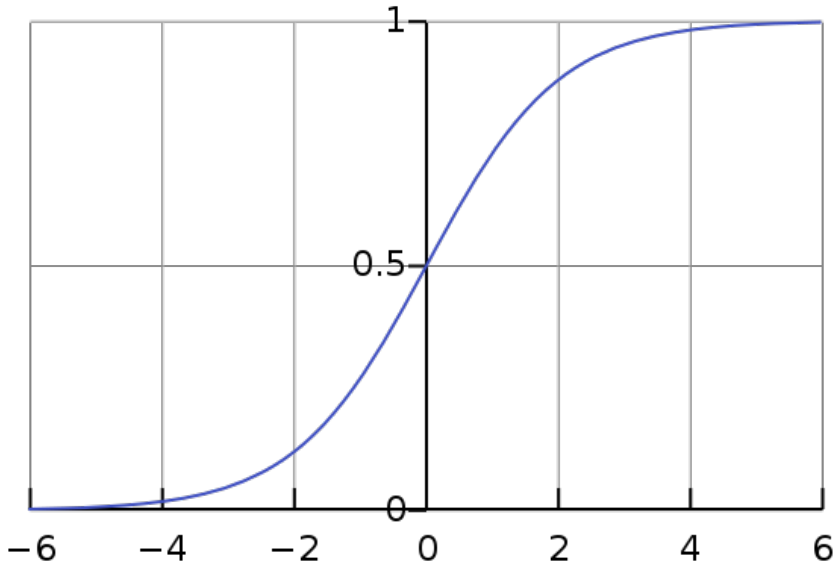

Figure 1: Logistic Curve

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:

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

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:

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

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:

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

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 ζ 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 $\alpha^k > 0$

Set $t = -1$ ▷ Records number of correction pairs currently computed
$\bar{w}_t = 0$
for $k = 1, \ldots,$ **do**
  Choose a sample $\mathcal{S} \subset \{1, 2, \cdots, N\}$
  Calculate stochastic gradient $\widehat{\nabla} F(w^k)$ as defined in (1.4)
  $\bar{w}_t = \bar{w}_t + w^k$
  if $k \leq 2L$ **then**
    $w^{k+1} = w^k - \alpha^k \widehat{\nabla} F(w^k)$ ▷ Stochastic gradient iteration
  else
    $w^{k+1} = w^k - \alpha^k H_t \widehat{\nabla} F(w^k)$, where H_t is defined by Algorithm 2
  end if
  if $\mathrm{mod}(k, L) = 0$ **then** ▷ Compute correction pairs every L iterations
    $t = t + 1$
    $\bar{w}_t = \bar{w}_t / L$
    if $t > 0$ **then**
      Choose a sample $\mathcal{S}_H \subset \{1, \cdots, N\}$ to define $\widehat{\nabla}^2 F(\bar{w}_t)$ by (2.3)
      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
    end if
    $\bar{w}_t = 0$
  end if
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** $j = t - \tilde{m} + 1, ..., t$ **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

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 λ and Λ 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 λ and Λ 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 γ 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,

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

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 $\tau > 1$

Set t = -1;
$\bar{w_t} = 0$;
j = 0;
for k = *1,..., do* **do**
  Choose a sample $S \subset \{1,..,N\}$;
  Calculate stochastic gradient $\bar{\nabla} F(w^k)$ as defined in (1.4) ;
  $\bar{w_t} = \bar{w_t} + w^k$;
  if $k < 2L$ **then**
    $w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k)$;
  else
    $w^{k+1} = w^k - \alpha^k H_t \bar{\nabla} F(w^k)$;
  end
  $j = j + 1$;
  if j = L **then**
    t = t + 1 ;
    $\bar{w_t} = \bar{w_t}/L$;
    if $t > 0$ **then**
      Choose a sample $S_H \subset \{1,...,N\}$ to define $\bar{\nabla}^2 F(\bar{w_t})$;
      Compute $s_t = (\bar{w_t} - \bar{w_{t-1}})$, $y_t = \bar{\nabla}^2 F(\bar{w_t})(\bar{w_t} - \bar{w_{t-1}})$;
    end
    $j = 0$;
    Define line model $G_t(w) = \frac{1}{N}\sum_{i \in S}[f(\bar{w_t}; x_i, z_i) + \nabla f(\bar{w_t}; x_i, z_i)(w - \bar{w_t})]$;
    if $F(\bar{w_{t-1}}) - F(\bar{w_t}) \geq \eta(G_{t-1}(\bar{w_{t-1}}) - G_t(\bar{w_t}))$ **then** L = τL ;
    else L = L / τ ;
  end
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: 3000-dimension input, 1output(0-1 discrete value), size is 50000.

6.3 Critical Program Parameter

- Learning rate α: $w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k)$. (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: η and τ : $F(w_{t-1}^-) - F(\bar{w}_t) \geq \eta(G_{t-1}(w_{t-1}^-) - G_t(\bar{w}_t))$ and $L = \tau L$ or $L = L/\tau$

6.4 Result

We are interested on cost of several parts. For example, the cost of Hessian updating(L-BFGS) is what we are most interested. At the same time, there are two different searching direction updating method:

$$w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k) \tag{23}$$

and

$$w^{k+1} = w^k - \alpha^k H_t \bar{\nabla} F(w^k) \tag{24}$$

We denote them as DirType1 and DirType2.

Table 1: Statistics of Dataset1

	Fixed	Dynamic
Dimension	16	
Initial Guess	AllOnes * 10	
Objective	0.7266	0.7266
Gradient norm	2.51e-7	2.46e-7
Window size	5	5
Iteration times	14	18
L-BFGS implement times	1	4
Typical L-BFGS cost	0.79	1.17
DirType1	0.45	0.53
DirType2	0.87	1.18

On the dataset2, since the dataset's variance is too large, we do the normalization first. Then use our algorithm to optimize the cross entropy function.

Table 2: Statistics of Dataset2

	Fixed	Dynamic
Dimension	3000	
Initial Guess	AllOnes * 10	
Objective	0.0092	0.0092
Gradient norm	0.0036	0.0054
Window size	20	20
Iteration times	101	97
L-BFGS implement times	4	15
Typical L-BFGS cost	28	52
DirType1	0.75	0.78
DirType2	12.18	12.34
Cost Time	85	914

There is one thing need to notice: as the program running, the program will occupy more and more memory, which leads to more cache missing(latency). And mostly, every program will have a very slow behavior in the last phase of running time. Thus, the cost per iteration will definitely increase. In our statistic, we pick the median number as benchmark.

6.5 Remark

What can we learn from the two numerical experiments?

First, there is a very interesting phenomena. The size of window is decreasing all the time, which does not obey what I expect. In my expectation, the window can be adjusted during the function varies. When the objective function decreases as well as we expect, we do not update the Hessian, otherwise, we do a updating. But as we can see, in both cases, the window size is decreasing all the time, from a given number, to 1, and fixed. This indicates us that a better linear model must be found, and more suitable associated parameters τ, η must be given. Otherwise, it will become a simple L-BFGS method, updated each iteration, making the program running slowly.

Second, the step rate α_k among $w^{k+1} = w^k - \alpha^k \bar{\nabla} F(w^k)$, and $w^{k+1} = w^k - \alpha^k H_t \bar{\nabla} F(w^k)$ is very critical. We have a results going as follow:

```
iterations : 2  ;  fx value = 18.4111
iterations : 2  ;  gradient norm = 3.64039


iterations : 4  ;  fx value = 18.4105
iterations : 4  ;  gradient norm = 3.94328


iterations : 6  ;  fx value = 18.4096
iterations : 6  ;  gradient norm = 3.98841


iterations : 8  ;  fx value = 18.4085
iterations : 8  ;  gradient norm = 3.81279


iterations : 10  ;  fx value = 18.4071
iterations : 10  ;  gradient norm = 3.85295


iterations : 12  ;  fx value = 18.4054
iterations : 12  ;  gradient norm = 3.6798


iterations : 14  ;  fx value = 18.4036
iterations : 14  ;  gradient norm = 3.76441


iterations : 16  ;  fx value = 18.4014
iterations : 16  ;  gradient norm = 4.10964


iterations : 18  ;  fx value = 18.3992
iterations : 18  ;  gradient norm = 3.59316


iterations : 20  ;  fx value = 18.3966
iterations : 20  ;  gradient norm = 3.85099


iterations : 22  ;  fx value = 18.3939
iterations : 22  ;  gradient norm = 3.72367


iterations : 24  ;  fx value = 18.391
iterations : 24  ;  gradient norm = 3.81304


iterations : 26  ;  fx value = 18.3878
iterations : 26  ;  gradient norm = 3.91238


iterations : 28  ;  fx value = 18.3845
iterations : 28  ;  gradient norm = 3.81776


iterations : 30  ;  fx value = 18.3811
iterations : 30  ;  gradient norm = 3.81327
```

Figure 4: wired result 1

```
iterations : 32  ;  fx value = 18.3775
iterations : 32  ;  gradient norm = 3.79123

iterations : 34  ;  fx value = 18.3736
iterations : 34  ;  gradient norm = 3.89934

iterations : 36  ;  fx value = 18.3697
iterations : 36  ;  gradient norm = 3.65325

iterations : 38  ;  fx value = 18.3659
iterations : 38  ;  gradient norm = 3.55704

iterations : 40  ;  fx value = 18.3616
iterations : 40  ;  gradient norm = 3.78326

iterations : 42  ;  fx value = 18.3571
iterations : 42  ;  gradient norm = 3.96651

iterations : 44  ;  fx value = 18.3526
iterations : 44  ;  gradient norm = 4.02741

iterations : 46  ;  fx value = 18.348
iterations : 46  ;  gradient norm = 3.74358

iterations : 48  ;  fx value = 18.3433
iterations : 48  ;  gradient norm = 3.74373

iterations : 50  ;  fx value = 18.3384
iterations : 50  ;  gradient norm = 3.68766

iterations : 52  ;  fx value = 18.3335
iterations : 52  ;  gradient norm = 3.58921

iterations : 54  ;  fx value = 18.3282
iterations : 54  ;  gradient norm = 3.88474

iterations : 56  ;  fx value = 18.3231
iterations : 56  ;  gradient norm = 3.6625

iterations : 58  ;  fx value = 18.3175
iterations : 58  ;  gradient norm = 3.94229

iterations : 60  ;  fx value = 18.3121
iterations : 60  ;  gradient norm = 3.69449
```

Figure 5: wired result 2

```
iterations : 62  ;  fx value = 18.3064
iterations : 62  ;  gradient norm = 3.82565


iterations : 64  ;  fx value = 18.3007
iterations : 64  ;  gradient norm = 3.99842


iterations : 66  ;  fx value = 18.2946
iterations : 66  ;  gradient norm = 3.93971


iterations : 68  ;  fx value = 18.2886
iterations : 68  ;  gradient norm = 4.0059


iterations : 70  ;  fx value = 18.2825
iterations : 70  ;  gradient norm = 3.65794


iterations : 72  ;  fx value = 18.2766
iterations : 72  ;  gradient norm = 3.35096


iterations : 74  ;  fx value = 18.2705
iterations : 74  ;  gradient norm = 3.70877


iterations : 76  ;  fx value = 18.264
iterations : 76  ;  gradient norm = 3.81064


iterations : 78  ;  fx value = 18.2575
iterations : 78  ;  gradient norm = 3.71953


iterations : 80  ;  fx value = 18.251
iterations : 80  ;  gradient norm = 3.62768


iterations : 82  ;  fx value = 18.2447
iterations : 82  ;  gradient norm = 3.62258


iterations : 84  ;  fx value = 18.2379
iterations : 84  ;  gradient norm = 3.9369


iterations : 86  ;  fx value = 18.2309
iterations : 86  ;  gradient norm = 3.9237


iterations : 88  ;  fx value = 18.2238
iterations : 88  ;  gradient norm = 3.85911


iterations : 90  ;  fx value = 18.2168
iterations : 90  ;  gradient norm = 3.62762
```

Figure 6: wired result 3

We can observe that the objective value is fluctuating, with a quasi period. That's owing to fixed $alpha$. Right now our strategy is using the function:

$$\alpha(k) = \frac{\alpha_0}{k} \tag{25}$$

If needed, I am considering combing a real Armijo line search in this place.

In conclusion, I must admit, this method is far lower than what I expect. Some make-up methods are under consideration.