sLBFGS+PLS Implementation Notes

Hammaad Adam, Chaitanya Rastogi and Tomas Rube

June 10, 2019

1 Notation, etc.

- f: the learning model we are trying to infer parameters for (objective function); $f(t) = \hat{\mathcal{L}}(x(t))$
- t: the current location; position vector is given by $x(t)$
- y_t, y'_t: noisy function and gradient values of f at t
- σ_f, $\sigma_{f'}$: estimates of noise in function value and gradient
- d: the current line search direction
- $\odot$: element-wise operation, i.e. $x^{\odot 2}$ indicates element-wise squaring of the vector x, while $a \odot b$ indicates element-wise multiplication of the two vectors

The sLBFGS used in testing the code here was copied from my original Java code into Matlab. Testing has shown this version is not as efficient as the original Java code, probably due to some minor 0-1 array indexing issue in `twoLoopRecursion` that makes it slightly unstable. However, I still believe the overall trend of the results holds.

2 Scaling Issues

A major component of the probabilistic line search (PLS) paper discussed the elimination of hyperparameters. Many of these parameters were used to set the intrinsic scale of the gaussian process (GP) surrogate function on which the line search optimization was taking place. Upon closer inspection, the implementation of some of these scaling parameters was not in line with the theoretical description in the paper and/or did not make sense in the context of sLBFGS. Correcting these scaling parameters was critical to the stability of the line search when used with sLBFGS and are discussed below.

2.1 Scale factor β

The scaling factor β in the code and pseudocode is used to eliminate the hyperparameter θ, which "scales the prior variance" according to the paper. By setting $\theta = 1$ and rescaling the objective with $|y_0'|$, we get $y(0) = 0$ and $y'(0) = -1$. In other words,

$$y_i = \frac{y_i - y_0}{|y_0'|} \tag{1}$$

$$y_i' = \frac{y_i'}{|y_0'|} \tag{2}$$

Clearly, $|y_0'|$ refers to the 'norm of the gradient of f at the start of the line search,' based off of the definitions used elsewhere in the paper. However, in the pseudocode for `probLineSearch` at line 15, the scaling factor β is defined as

$$\beta \leftarrow |d' \cdot \Sigma_{df_0}|,$$

where d is the search direction and Σ_{df_0} is the sample variances of the gradient. This is clearly wrong, and the code does not reflect this; rather, the Matlab code uses

```
beta = abs(search_direction'*df0);
```

where `df0` is the function gradient at the origin of the line search. While this statement is congruent with the definition in the text of the paper for SGD, it is *not* for BFGS methods. The reason lies in the definition of the search direction d for both methods:

$$\text{SGD: } d = -\nabla f$$

$$\text{BFGS: } d = -H^{-1}\nabla f$$

where H is the pseduo-hessian matrix computed in the BFGS updates. As such, beta in the two cases becomes

$$\text{SGD: } \beta = |(-\nabla f)' \cdot \nabla f| = |y_0'|$$

$$\text{BFGS: } \beta = |(-H^{-1}\nabla f)' \cdot \nabla f| \neq |y_0'|$$

In the case of BFGS updates, it is possible that the above definition of β can be ≈ 0, as the inverse hessian can rotate the gradient vector to be nearly orthogonal to the gradient. The correct code should be

```
beta = norm(df0);
```

Fixing this rescaling greatly enhances the stability of the GP: before this fix, running sLBFGS + PLS with a batchsize of 20 and epoch period of 50 steps resulted in β values approaching 10^{-9}.

2.2 Rescaling by α_0

α_0 is the initial step size in the line search, in non-dimensional units. In the theoretical discussion of the scaling factor β, we see that (1) and (2) only admit the norm of the function gradient. Similarly, we see that the paper discusses rescaling the noise estimates for σ_f and $\sigma_{f'}$ as follows:

$$\sigma_f = \frac{\sigma_f}{|y'(0)|} \tag{3}$$

$$\sigma_{f'} = \frac{\sigma_{f'}}{|y'(0)|} \tag{4}$$

Curiously, the pseudocode for `probLineSearch` at lines 15 and 16 instead show:

$$\sigma_f \leftarrow \sqrt{\Sigma_{f_0}}/(\alpha_0 \cdot \beta)$$

$$\sigma_{df} \leftarrow \sqrt{(d^{\odot 2})' \cdot \sigma_{df_0}})$$

In addition, the pseudocode for `evaluateObjective` at lines 6 and 7 show:

$$y \leftarrow (y - f_0)/(\alpha_0 \cdot \beta)$$

$$dy \leftarrow (dy' \cdot d)/\beta$$

The original Matlab code follows this convention as well. It is unclear why the initial step size is needed to rescale the function values and gradients, especially as it is not motivated in the text. In fact, given that function values y and noise σ_f are scaled by an additional $1/\alpha_0$ term, we can get improperly scaled function value estimates relative to gradient estimates. In the sLBFGS setting, where a fixed $\alpha_0 = .1$ was used, this translates to an order of magnitude variation. As such, the Matlab code was amended as follows:

```
sigmaf = sqrt(var_f0)/beta; (in probLineSearch)
   y = (y - f0)/beta; (in evaluate_function)
       y_tt = y*beta + f0; (in make_outs)
```

2.3 Step Size Selection

In traditional BFGS methods, the quasi-newton nature of these methods generate 'properly' scaled search directions that allow the initial step size of the line search, α_0, to be set to 1 every iteration. Significantly, this value is independent of the objective function being optimized. In contrast, the original sLBFGS implementation uses a fixed step size η that needs to be optimized to suit the objective at hand. In yet another variation, the original PLS approach removes the need to tune the initial step size by setting it to 1, but relies on ad-hoc schemes that track the size of previous steps to change the step size of the upcoming iteration. Thankfully, these updating schemes are unnecessary in the sLBFGS + PLS setting, where $\alpha_0 = \eta$. While an improvement, it is surprising how the sLBFGS + PLS method cannot deal with objective-independent unitary step sizes without running into convergence issues.

3 Noise Estimation

Another major theoretical component of the PLS paper is the usage of noisy estimates to do a somewhat deterministic optimization of an objective function. As such, determining the noise level of these estimates forms a critical component of the PLS. However, I believe there is a major approximation made in the paper that can significantly impact the performance of the line search, especially within the context of stochastic variance-reduced gradients (SVRG).

The original PLS implementation sets the function and gradient noise levels (σ_f and $\sigma_{f'}$) at the *start* of the line search, i.e. at $t = 0$, and maintains it throughout the search. Surprisingly, faster convergence was observed in the SGD setting when the code was modified to change σ_f and $\sigma_{f'}$ to the largest value encountered during the line search. This test highlighted the impact of the approximation in estimating the variance.

3.1 SEM in the Minibatch Setting

The problem in estimating the SEM of function and gradient values in a mini-batch can be recast into a toy problem:

> Let us begin with a set of points A, which have mean μ_A, variance σ_A^2, and SEM $\sigma_A/\sqrt{|A|}$, where $|A|$ is the cardinality of the set A. Our goal is to estimate μ_A; however, we only have access to sets A' that are generated by sampling points (without replacement) from A such that $|A'| \leq |A|$. If we can only observe sets A', what is the SEM of the means $\mu_{A'}$ that we observe?

Conceptually, this toy problem is key to understanding the error in the process for estimating the function and gradient SEMs in the current approach. When a mini-batch is used to *estimate* function and gradient values, we know the means have some spread (SEM) around the true function and gradient means as computed *exactly* on the entire dataset. As such, in the limit where my mini-batch hits the full size of the dataset, there should be *no* variance in the function and gradient means. The current approach employed in the paper and the code uses a naïve algorithm to compute the variance, using Bessel's correction:

$$\sigma_f^2 = \frac{1}{n}\frac{n}{n-1}\left[\frac{1}{n}\sum_{i\in A'}\ell^2(A_i) - \left(\frac{1}{n}\sum_{i\in A'}\ell(A_i)\right)^2\right] \tag{5}$$

$$\sigma_{f'}^2 = \frac{1}{n}\frac{n}{n-1}d^{\odot 2}\cdot\left[\frac{1}{n}\sum_{i\in A'}\nabla\ell(A_i)^{\odot 2} - \left(\frac{1}{n}\sum_{i\in A'}\nabla\ell(A_i)\right)^{\odot 2}\right] \tag{6}$$

where $N = |A|$ and $n = |A'|$. Unfortunately, this naïve method computes non-zero estimates of σ_f and $\sigma_{f'}$ when $A' = A$. If we have access to the 'true' σ_A of the whole set (i.e. the intrinsic variance), Tomas proposed the following

correction to estimate mini-batch SEM:

$$\mathrm{SEM}_{A'} = \sqrt{\frac{N-n}{Nn}\sigma_A^2}, \tag{7}$$

The derivation is presented below.

Proof. Consider N random variables x_i, where $i \in 1, \ldots, N$ drawn from a distribution with variance σ^2. The average of these variables is then

$$\mu_N = \frac{1}{N}\sum_i^N x_i$$

If you can only use the first n variables to estimate the mean, we get

$$\mu_n = \frac{1}{n}\sum_i^n x_i$$

Let

$$\mu_{N-n} = \frac{1}{N-n}\sum_{i=n+1}^N x_i$$

Then

$$\mu_N = \frac{n\mu_n + (N-n)\mu_{N-n}}{N}$$

We can define the error e in estimating μ_N using μ_n as

$$e(n|N) = \mu_N - \mu_n = \frac{n\mu_n + (N-n)\mu_{N-n}}{N} - \mu_n = \frac{N-n}{N}(\mu_{N-n} - \mu_n)$$

The variance of this error is then

$$\begin{aligned}
\mathrm{Var}[e(n|N)] &= \frac{(N-n)^2}{N^2}\left(\mathrm{Var}[\mu_{N-n}] - \mathrm{Var}[\mu_n]\right) \\
&= \frac{(N-n)^2}{N^2}\left(\frac{1}{N-n} + \frac{1}{n}\right)\sigma^2 \\
&= \frac{N-n}{Nn}\sigma^2
\end{aligned}$$

□

3.2 Other Variance Measures

Tomas suggested an alternative method for measuring the SEM of the gradient:

$$\sigma_{f'}^2 = \frac{1}{n}\frac{n}{n-1}\left[\frac{1}{n}\sum_{i\in A'}\left[d\cdot\nabla\ell(A_i)\right]^{\odot 2} - \left(\frac{1}{n}\sum_{i\in A'} d\cdot\nabla\ell(A_i)\right)^{\odot 2}\right] \tag{8}$$

Conceptually, this modification captures the per-data point gradient variance in the *direction* of descent rather than the projection of the gradient SEM on the descent direction. When tested in place of the original estimator in a SGD context, the PLS was more efficient.

4 PLS in the SVRG Setting

Stochastic Variance Reduced Gradient, or SVRG, is a method used to improve the speed and efficiency of convergence in stochastic optimization methods. Originally designed for use in the SGD context, it was ported to the quasi-newton setting in sLBFGS. SVRG is supposed to reduce the 'variance' of the SGD estimates through a modified update scheme that relies on a gradient μ_k that is computed on the entire dataset once per epoch at some position x_k. The SVRG gradient at every iteration i on mini-batch b is then

$$\nabla_{\text{SVRG}} f(x_i, b) = \underbrace{\nabla f(x_i, b)}_{\substack{\text{stochastic} \\ \text{estimate}}} - \underbrace{(\nabla f(x_k, b) - \mu_k)}_{\text{batch bias}} \tag{9}$$

where $\nabla f(x_k, b)$ indicates the gradient of f at x_k as computed on mini-batch b. In my understanding, this update estimates the batch-induced bias in stochastic estimates by comparing the difference between the gradient computed on the full dataset and the mini-batch b at x_k. This difference represents the bias induced by computing on batch b and can be subtracted from the current estimate of the gradient. As a result, the mean function and gradient values as estimated from the mini-batch should be closer to the true values.

4.1 Scaling to SVRG Space

Every sLBFGS update already computes the SVRG gradient estimate v_t every iteration. In order to properly run the line search in SVRG space, the starting function values need to be adjusted as well:

```
for k = 0:maxEpoch
        ...
        % function and gradient on the full dataset
        [fFull, mu_k] = f(w_k);
        ...
        % iterations within an epoch
        for t = 1:m
                % select batchsize
                sidx = randsample(1:N, batchsize);
                % compute on minibatch
                [f_xt, grad_xt] = f(x_t, sidx);
                [f_wk, grad_wk] = f(w_k, sidx);
                % compute SVRG function and grad
                f_t = (f_xt + fFull) - f_wk;
                v_t = (grad_xt + mu_k) - grad_wk;
                ...
                % compute effective direction
                if (r < 1)
                        % no hessian updates have occured
                        effGrad = v_t;
                else
```

```
        effGrad = twoLoopRecursion(v_t);
end
...
% use line search
if (r < 1)
        step_size = ls(x_t, f_t, v_t, ...
        -effGrad, eta/norm(v_t), ...,
        fFull, mu_k, w_k);
else
        step_size = ls(x_t, f_t, v_t, ...
        -effGrad, eta, fFull, mu_k, ...
        w_k);
end
```

In addition, `evaluate_function` needs to be modified:

```
% select batchsize
sidx = randsample(1:N, batchsize);
% compute on minibatch
[y_t, dy] = f(x0 + tt*alpha0*search_direction, sidx);
[y_k, grad_wk] = f(w_k, sidx);
y = (y_t + fFull) - y_k;
dy = (dy + mu_k) - grad_wk;
```

It is critical that the *same* mini-batch is used to compute function and gradient values at x_t and w_k. Doing so on different batches ruins the impact of bias reduction. When improperly implemented, the improper SVRG updates cause high-frequency, low-amplitude oscillations around the minimum, preventing convergence.

4.2 Variance in the SVRG Context

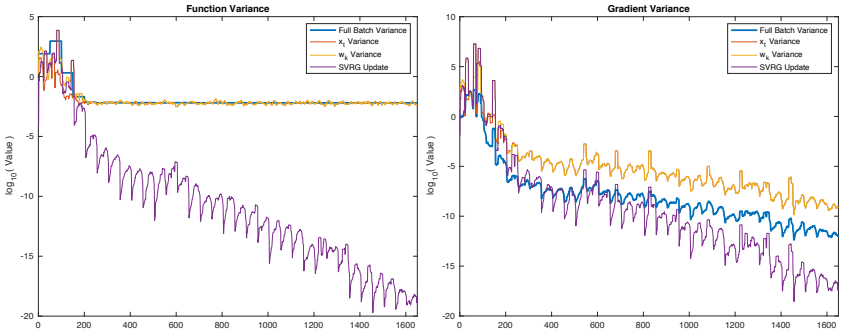
Discuss root 2 inflation, and how we can use variance estimates from the SVRG update to estimate noise. This is more stable (discuss 3 orders of magnitude variation in gradient SD per mini-batch), and also means tomas's estimator, even though it might be more efficient and less biased, will be harder to implement computationally

5 Determining the 'Suitability' of the Descent Direction

5.1 Metrics for GP Precision and Recall

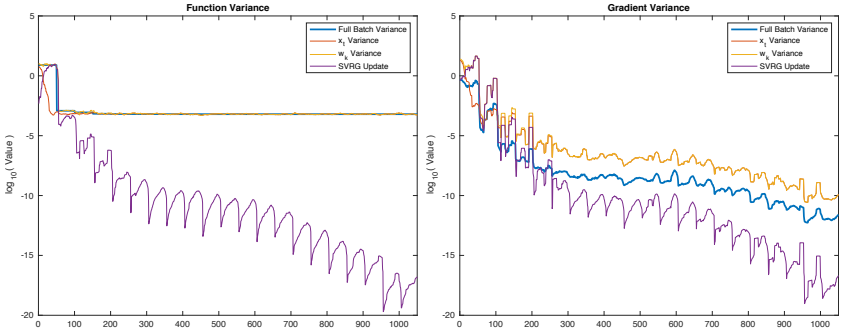
6 Subsampling True Dataset to Approximate SVRG

7 Alternative Line Search Candidate Methods



(a) SVM Function Variance (b) SVM Gradient Variance

Figure 1: Function and Gradient Variances for the SVM test case with a gradient batch size of 20, a hessian batch size of 200, a hessian update period of 10 iterations, and a full gradient computation (an epoch) every 50 iterations. 11 step moving averages are shown for all tracks except for the full batch function variance track. X-axis shows the number of iterations. The full batch variance values use Tomas's SEM correction term.



(a) SVM Function Variance (b) SVM Gradient Variance

Figure 2: Function and Gradient Variances for the SVM test case with a gradient batch size of 200, a hessian batch size of 2000, a hessian update period of 10 iterations, and a full gradient computation (an epoch) every 50 iterations. 11 step moving averages are shown for all tracks except for the full batch function variance track. X-axis shows the number of iterations, and the full batch variance values use Tomas's SEM correction term.