Project: SGD Variants

University of Padova: Optimization For Data Science

Group members:

Massimiliano Conte, Pierpoalo D'Odorico, Eddie Rossi, Luca Solbiati

May 2021

Abstract

Stochastic optimization is a classic approach for solving unconstrained optimization problems under uncertainty. Unconstrained optimization problems can be formulated as optimization problems with uncertain parameters. In this report we will explore some classic approaches and some recent methods, that are widely used in big data community. We will also make some experiments on different datasets.

Contents

1 Stochastic optimization

We start describing the optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x) = \mathbb{E}_\xi[F(x, \xi)]$$

Where $F(x, \xi)$ is function that involves our set of decision variables x and a random variable ξ . The random variable sample space is Ω and its probability is P

1.1 Stochastic gradient approximation

In a classic stochastic gradient method we assume that the function F is continuously differentiable with respect to x for any given ξ. The stochastic gradient method iteration is given by:

$$x_{k+1} = x_k - \alpha_k \nabla F(x_k, \xi_k)$$

where α_k is a suitably chosen stepsize and ξ_k is a sampling realization of ξ. The stochastic gradient is unbiased, i.e. the expected value of the stochastic gradient is equal to $\nabla f(x)$. The algorithm scheme is the following:

Algorithm 1: Stochastic gradient method

Choose a point $x_1 \in \mathbb{R}^n$

for *k = 1,...* **do**

  if x_k *satisfies some specific condition* **then**

    STOP

  end

  Sample ξ_k from ξ random variable

  Set $x_{k+1} = x_k - \alpha_k \nabla F(x_k, \xi_k)$, with $\alpha_k > 0$ suitably chosen stepsize

end

In order to ensure convergence we need a diminishing stepsize α_k, which means that we need a sequence α_k such that $\alpha_k \to 0$ when $k \to +\infty$. To prove the main convergence result of the algorithm we need f to be a σ-strongly convex function with Lipschitz continuous gradient. We notice that in the Stochastic gradient method we need strong convexity to get a sublinear convergence rate of $O(1/k)$, while the classic gradient method only needs Lipschitz continuity of the gradient to get the same rate.

In DS problems we try to minimize the empirical error instead of the true error that is described in the original optimization problem. The empirical error is an approximation of the true error, and it's computed on a training set that contains pairs of input and output values. So the problem will have the following form:

$$\min_{x \in \mathbb{R}^n} f(x) = \mathbb{E}_\xi[F(x, \xi)] = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

We'll mainly focus on this kind of problem. In this setting the classic GD method has an iteration cost of $\mathcal{O}(n)$ and an overall complexity of $\mathcal{O}(n \log(1/\varepsilon))$, while the SGD method has a sublinear rate but with an iteration cost of $\mathcal{O}(1)$, and an overall complexity of $\mathcal{O}(1/\varepsilon)$. It is crucial to note that the SG is independent from the sample set size n. This means that the overall complexity of SG can be larger than the one of GD for small enough n. But the comparison favors SGD with big data i.e. when n is very large, that is because the oracle cost doesn't allow us to take advantage of the better convergence rates of GD. Another reason why SGD is preferable is that in a normal dataset there is a lot of redundancy, and computing the whole gradient does not give much additional information.

1.2 Stochastic Variance reduction Gradient Method

Stochastic Variance reduction Gradient Methods were developed to overcome the issue of high variance in the gradient estimator. Indeed, even though the gradient estimator is unbiased, we may have large variance, and this is the reason why we get a sublinear rate when using the SGD method.

In general, when estimating $\Theta = \mathbf{E}[X]$ we may suppose to have a random variable Y, that is highly correlated with X. So for $\gamma \in [0, 1]$ we may define a point estimator $\hat{\Theta}_\gamma = \gamma(X - Y) + \mathbf{E}[Y]$ that has the following expectation and variance:

$$\mathbf{E}[\hat{\Theta}_\gamma] = \gamma\mathbf{E}[X] + (1 - \gamma)\mathbf{E}[Y]$$
$$var[\hat{\Theta}_\gamma] = \gamma^2(var[X] + var[Y] - 2cov[X, Y])$$

For $\gamma = 1$ we have an unbiased estimator and for $\gamma = 0$ we have a zero variance estimator. The problem for $\gamma = 0$ is that the bias of the estimator is big. For $\gamma \neq 0$ we notice that $var[\hat{\Theta}_\gamma] << var[X]$ when $cov(X, Y)$ is large. So if we have a higly correlated r.v. Y we may use this estimator to significantly reduce variance. The main idea behind SVRG is exactly this one.

1.3 SVRG

Before analyzing SVRG, we set $\Theta = \mathbf{E}[\nabla f_i(x)]$ and $X = \nabla f_i(x)$. At each epoch s SVGR performs t steps in the following way

$$x_{k+1} = x_k - \alpha_k \left[\nabla f_{i_k}(x_k) - \nabla f_{i_k}(\tilde{x}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\tilde{x}) \right]$$

In the formula i_k is a randomly chosen index and $\tilde{x}$ is the last iterate coming from the previous epoch $s - 1$. Putting $\gamma = 1$ and $Y = \nabla f_{i_k}(\tilde{x})$ the expression in the square brackets is the same as $\hat{\Theta}_\gamma = \gamma(X - Y) + \mathbf{E}[Y]$ described before. The algorithm scheme is the following:

Algorithm 2: SVRG variance reduction gradient method

Initialize $\tilde{x}_0$
for *s = 1,...* **do**
  $\tilde{x} = \tilde{x}_{s-1}$
  $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\tilde{x})$
  $x_0 = \tilde{x}$
  for *t = 1, ..., m* **do**
    Pick randomly $i_t \in 1, ..., n$ and update weight
    $x_t = x_{t-1} - \eta(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu})$
  end
  Set $\tilde{x_s} = x_m$
end

Thanks to the variance reduction we're able to achieve a linear convergence rate. Another advantage of SVRG is that unlike other variance reduction methods (like SAG and SAGA) it doesn't need to store the gradients for each iteration.

2 SARAH

2.1 SARAH algorithm

We are interested in solving a problem of the form:

$$\min_{w \in \mathbb{R}^d} \left\{ P(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w) \right\}$$

where each f_i is convex with L-Lipschitz continuous gradient.

SARAH (StochAstic Recursive grAdient algoritHm) [2] is a SGD variation that combines some of the good properties of existing algorithms, such as SAGA and SVRG.
Similarly to SVRG, SARAH's iterations are divided into the outer loop where a full gradient is computed and the inner loop where only stochastic gradient is computed. Unlike SVRG, the steps of the inner loop of SARAH are based on accumulated stochastic information (as in SAGA).
The stochastic gradient estimate v_t is computed as follows

$$v_t = \nabla f_{i_t}(w_t) - \nabla f_{i_t}(w_{t-1}) + v_{t-1}$$

where $\nabla f_{i_t}(w_t)$ is the stochastic gradient of the function computed with respect to the sample i_t and to the parameters at time t, $\nabla f_{i_t}(w_{t-1})$ is the stochastic gradient of the function computed with respect to the same sample i_t but considering parameters at time $t-1$ and v_{t-1} is the stochastic gradient estimate of the previous iteration.
The parameters are then updated using a fixed learning rate η.

$$w_{t+1} = w_t - \eta v_t$$

Unlike SAG/SAGA SARAH does not require a storage of the n past stochastic gradients.

Algorithm 3: SARAH

Initialize $\tilde{w}_0$
for *s = 1,...,τ* **do**
  $w_0 = \tilde{w}_{s-1}$
  $v_0 = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_0)$
  $w_1 = w_0 - \eta v_0$
  for *t = 1, ..., m-1* **do**
    Pick randomly $i_t \in 1, ..., n$
    $v_t = \nabla f_{it}(w_t) - \nabla f_{it}(w_{t-1}) + v_{t-1}$
    $w_{t+1} = w_t - \eta v_t$
  end
  Set $\tilde{w}_s = w_t$ with t chosen uniformly at random from {0, 1, . . . ,m}
end
return $\tilde{w}_\tau$

2.2 Linearly Diminishing Step-Size in a Single Inner Loop

In SVRG, the property of variance reduction of the steps holds if the number of outer iteration grows, but it does not hold, if only the number of inner iterations increases. In SARAH instead, the variance of the steps goes to zero even by running the inner loop for many iterations (without executing additional outer loops).
Under the assumptions of Ł-smoothness and convexity of each $f_i(w)$ and $P(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w)$ μ-strongly convex, the following result can be derived:

$$\mathbb{E}[||v_t||^2] \leq \left[1 - \left(\frac{2}{\eta L} - 1 \right) \mu^2 \eta^2 \right]^t \mathbb{E}[||\nabla P(w_0)||^2]$$

This implies that by choosing $\eta = \mathcal{O}(1/L)$, $||v_t||^2$ converges linearly in expectation with the rate $(1 - 1/\kappa^2)$ where $\kappa := \mu/L$ is the conditioning number.

Furthermore, under the stronger assumptions of Ł-smoothness and μ-strong convexity of each $f_i(w)$ the following can be proved:

$$\mathbb{E}[||v_t||^2] \leq \left(1 - \frac{2\mu L\eta}{\mu + L}\right)^t \mathbb{E}[||\nabla P(w_0)||^2]$$

This means that by setting $\eta = \mathcal{O}(1/L)$, $||v_t||^2$ converges linearly in expectation with the rate $(1 - 1/\kappa)$.
This property makes SARAH theoretically and practically more stable and reliable than SVRG. You might also take advantage of this convergence result to build a stopping criteria for the number of iterations of the inner loop.

2.3 Complexity analysis

Our iteration complexity analysis aims to bound the number of outer iterations τ (or total number of stochastic gradient evaluations) which is needed to guarantee that

$$\mathbb{E}[||\nabla P(w_s)||^2] \leq \epsilon$$

In this case we will say that w_τ is an ϵ-accurate solution.
Please note that since each inner iteration evaluates 2 stochastic gradients, the total work per outer iteration is $\mathcal{O}(n + m)$ in terms of the number of gradient evaluations.

In the general convex case, to get an ϵ-accurate solution, supposing a number of inner iterations $m = \mathcal{O}(1/\epsilon)$, the number of outer iterations required are $s = \mathcal{O}(\log(1/\epsilon))$. Hence the total complexity in terms of the number of gradient evaluations to achieve an ϵ-accurate solution is $\mathcal{O}((n+(1+\epsilon))\log(1/\epsilon))$
To achieve the same result, but with a single outer loop $m = \mathcal{O}(1/\epsilon^2)$ inner iterations are necessary, which means a total comlexity of $\mathcal{O}(n + 1/\epsilon^2)$.

In the strongly convex case, to get an ϵ-accurate solution, supposing a number of inner iterations $m = \mathcal{O}(\kappa)$, the number of outer iterations required are $s = \mathcal{O}(\log(1/\epsilon))$. Hence the total complexity in terms of the number of gradient evaluations to achieve an ϵ-accurate solution is $\mathcal{O}((n+\kappa)\log(1/\epsilon))$.

3 Spiderboost

Spiderboost is a generalization of Spider and SARAH. In fact they use the exact same estimator for the gradient. There are only two differences between Spiderboost and SARAH, the first one is that in Spiderboost they don't reinitialize the value at end of each epoch. In Spiderboost they just continue to use the last value of the previous epoch. The second one is that SARAH is proposed in such a way to process one sample at the time, while Spiderboost has a mini-batch scheme.

Algorithm 4: Spiderboost

Input: $\eta = \frac{1}{2L}, q, K, |S| \in \mathbb{N}$

for $k = 0, ...K - 1$ **do**

  if $mod(k, q) = 0$ **then**

    Compute $v_k = \nabla f(x_k)$

  else

    Draw $|S|$ samples with replacement

    $v_k = \frac{1}{|S|} \sum_{i \in S} [\nabla f_i(x_k) - \nabla f_i(x_{k-1}) + v_{k-1}]$

  end

  $x_{k+1} = x_k - \eta v_k$

end

Output: x_ξ with $\xi \overset{\text{Unif.}}{\sim} \{0, 1, ...K - 1\}$

3.1 Prox-Spiderboost

The authors then proposed an extension for composite optimization. The optimization problem is defined as:

$$\min_{x\in\mathbb{R}^n} \Psi(x) := f(x) + h(x), \quad f(x) := \sum_{i=1}^{n} f_i(x)$$

The point of this optimization problem is that $h(x)$ can be non smooth. For this reason, we could not be able to get the gradient and use the update rule as in the previous methods. So they introduced the proximal mapping, an operator that let us update the variable is a similar way as before, but also taking care of the regularization function $h(x)$.

$$\text{prox}_{\eta h}(x) := arg\min_{u\in\mathcal{X}} \left\{ h(u) + \frac{1}{2\eta}||x-u||^2 \right\}$$

This operator basically optimizes h while it is forced to stay close to the original point x. The parameter $\frac{1}{2\eta}$ acts like a tuner for the penalization of the distance from the original point.
In order to prove the convergence and analize the rates of the algorithm, since we could not have the gradient of $\Psi(x)$, the authors introduced the notion of generalized gradient:

$$G_\eta(x) := \frac{1}{\eta}\left(x - \text{prox}_{\eta h}(x - \eta\nabla f(x))\right)$$

We can see this as the difference quotient where η represents a small value, that in the classic derivative definition goes to 0. In some sense it acts with a reverse order with respect to classical GD methods: first we get a feasible descent direction $x - \text{prox}_{\eta h}(x - \eta\nabla f(x)))$, then we compute a generalized gradient. The generalized gradient is useful to prove convergence because if a given point is optimal then $G_\eta(x) = 0$. In practice we look for $||G_\eta(x)|| < \epsilon$.

The scheme for Prox-Spiderboost is basically the same as Spiderboost but with a different update rule, that uses the proximal operator in order to handle composite (and/or constraint) optimization problems.

Algorithm 5: Prox-Spiderboost

Input: $\eta = \frac{1}{2L}, q, K, |S| \in \mathbb{N}$

for $k = 0, ...K-1$ **do**

  if $mod(k, q) = 0$ **then**

    Compute $v_k = \nabla f(x_k)$

  else

    Draw $|S|$ samples with replacement

    $v_k = \frac{1}{|S|}\sum_{i\in S}[\nabla f_i(x_k) - \nabla f_i(x_{k-1}) + v_{k-1}]$

  end

  $x_{k+1} = \text{prox}_{\eta h}(x_k - \eta v_k)$

end

Output: x_ξ with $\xi \overset{\text{Unif.}}{\sim} \{0, 1, ...K-1\}$

3.2 Prox-Spiderboost-M

After the extension for composite optimization problems, the authors decided to add momentum to Spiderboost, in order to speed up the convergence. The algorithm scheme is the following:

Algorithm 6: Prox-Spiderboost-M

Input: $q, K, |S| \in \mathbb{N}, \{\lambda_k\}_{k=1}^{K-1}, \{\beta_k\}_{k=1}^{K-1} > 0, y_0 = x_0 \in \mathbb{R}^d$

for $k = 0, ...K-1$ **do**

Set $\alpha_{k+1} = \frac{2}{\lceil (k+1)/q \rceil + 1}$

$z_k = (1 - \alpha_{k+1}) y_k + \alpha_{k+1} x_k$

if $mod(k, q) = 0$ **then**

Compute $v_k = \nabla f(x_k)$

else

Draw $|S|$ samples with replacement

$v_k = \frac{1}{|S|} \sum_{i \in S} [\nabla f_i(x_k) - \nabla f_i(x_{k-1}) + v_{k-1}]$

end

$x_{k+1} = \text{prox}_{\lambda_k h}(x_k - \lambda_k v_k)$

$y_{k+1} = z_k + \frac{\beta_k}{\lambda_k}(x_{k+1} - x_k)$

end

Output: z_ξ with $\xi \overset{\text{Unif.}}{\sim} \{0, 1, ...K-1\}$

For each iterate our candidate solution is given by z_k that is a convex combination of the iterates y_k and x_k :

- x_k is the usual Prox-Spiderboost step
- $y_{k+1} = z_k + \frac{\beta_k}{\lambda_k}(x_{k+1} - x_k)$ plays the role of momentum. Basically at each iteration we store the descent step $\frac{\beta_k}{\lambda_k}(x_{k+1} - x_k)$ into y_{k+1} therefore pushing z_{k+1} into the previous descent direction. Observe that the strength of the momentum is directly proportional to β_k and inversely proportional to the stepsize λ_k. Indeed this makes sense, because if λ_k is big then we need less momentum in the descent direction, as we've already took a big step towards it.

Observe that as k grows so does the influence of the momentum. The basic idea behind this is that as we compute more iterates the momentum becomes more stable and we can trust it more as a descent direction.

3.3 Convergence results

Since the assumptions are the same for the three algorithms analysis, we show the assumptions once here:

Assumption 1.
1. The objective function of the optimization problem is bounded from below, i.e.:

$$\Psi^* = \inf_{x \in \mathbb{R}^d} \Psi(x) > +\infty$$

2. Each gradient $\nabla f_i(x)$ is L-Lipschitz continuous, i.e.:

$$\forall x, y \in \mathbb{R}^d, ||\nabla f_i(x) - \nabla f_i(y)|| < L||x - y||$$

3.3.1 Spiderboost results

Theorem 3.1. *Let Assumption 1 hold and apply Spiderboost to solve the non composite optimization problem with parameters* $q = |S| = \sqrt{n}$ *and stepsize* $\eta = \frac{1}{2L}$. *Then, the corresponding output* x_ξ *satisfies* $E||\nabla f(x_\xi)|| \leq \epsilon$ *provided that the total number* K *of iterations satisfies:*

$$k \geq \mathcal{O}\left(\frac{L(f(x_0) - f^*)}{\epsilon^2}\right)$$

Moreover, the overall SFO complexity is $\mathcal{O}(n + \sqrt{n}\epsilon^{-2})$.

3.3.2 Prox-Spiderboost results

Theorem 3.2. *Let Assumption 1 hold and apply Prox-Spiderboost to solve the composite optimization problem with parameters* $\mathcal{X} = \mathbb{R}^d, q = |S| = \sqrt{n}$ *and stepsize* $\eta = \frac{1}{2L}$. *Then, the corresponding output* x_ξ *satisfies* $E||G_\eta(x_\xi)|| \leq \epsilon$ *provided that the total number* K *of iterations satisfies:*

$$k \geq \mathcal{O}\left(\frac{L(\Psi(x_0) - \Psi^*)}{\epsilon^2}\right)$$

Moreover, the overall SFO complexity is $\mathcal{O}(n + \sqrt{n}\epsilon^{-2})$.

3.3.3 Prox-Spiderboost-M results

Theorem 3.3. *Let Assumption 1 hold and apply Prox-Spiderboost-M to solve the composite optimization problem with parameters* $\mathcal{X} = \mathbb{R}^d, q = |S| = \sqrt{n}$ *and stepsize* $\beta_k = \frac{1}{8L}, \lambda_k \in [\beta_k, (1+\alpha_k)\beta_k]$. *Then, the corresponding output* z_ξ *satisfies* $E||G_{\lambda_\xi}(z_\xi)|| \leq \epsilon$ *provided that the total number* K *of iterations satisfies:*

$$k \geq \mathcal{O}\left(\frac{L(\Psi(x_0) - \Psi^*)}{\epsilon^2}\right)$$

Moreover, the overall SFO complexity is $\mathcal{O}(n + \sqrt{n}\epsilon^{-2})$.

4 Katyusha

Katyusha is an accelerated stochastic gradient descent method. While one can naively add a momentum to a full-gradient method to significantly improve performance, the same cannot be said about stochastic methods. Indeed applying a momentum to a stochastic gradient is very dangerous because if the gradient estimator is very innacurate, then adding a momentum to it may progagate an error, therefore hurting performance. To fix this problem, katyusha adds to the "positive" momentum a "negative" momentum, the so called katyusha momentum, as to counteract the accumulation of errors.

4.1 Problem Definition

We have the following minimization problem:

$$\min_{x \in \mathbb{R}^d} F(x) \quad where \quad F(x) := f(x) + \psi(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) + \psi(x)$$

With f_i smooth convex functions and ψ convex and lower semicontinuos (but possibly non differentiable) function called proximal function. We'll usually also assume:

1. f_i to be L-smooth, i.e. each f_i has L-Lipschits continuos gradient
2. ψ to be σ-strongly convex

4.2 Main Idea Behind Katyuhsa

To explain the main idea behind the algorithm let us assume $\psi \equiv 0$.
Katyusha is divided into epochs, each consisting of m iterations (usually $m = 2n$, but anything linear in n works). For each epoch s we compute m iterates: $x_{sm+1}, x_{sm+2}, ..., x_{sm+m}$ where we update x_{k+1} with the following rules:

- $x_{k+1} := \tau_1 z_k + \tau_2 \tilde{x}^s + (1 - \tau_1 - \tau_2) y_k \quad \text{where} \quad \tau_1, \tau_2 \in [0, 1]$
  is a convex combination of z_k, $\tilde{x}^s$ and y_k.
- $\tilde{x}^s$ is the snapshot point defined at the start of each epoch. It's usually a weighted average of $y_{(s-1)m+1}, ..., y_{sm}$. In the convex combination that defines x_{k+1} it behaves like a magnet with weight τ_2 (usually $\tau_2 = 0.5$). This magnet ensures that x_{k+1} is not too far away from $\tilde{x}^s$ so that the gradient estimator remains accurate enough. It can be viewed as a negative momentum that counteracts a fraction of the positive momentum. It's called katyusha momentum.
- $\widetilde{\nabla}_k := \nabla f(\tilde{x}^s) + \nabla f_i(x_k) - \nabla f_i(\tilde{x}^s) \quad where \quad i \in \{1, 2, ..., n\}$ is randomly generated
  is the gradient estimator for each iteration k. Observe that we do a full gradient computation of $\nabla f(\tilde{x}^s)$ at the start of the epoch and then only perform a stochastich update of a random i^{th} gradient evaluated at x_k. It's easy to prove that $\widetilde{\nabla}_k$ is unbiased.
- $y_k := x_k - \frac{1}{3L}\widetilde{\nabla}_k$ is (kind of) the usual SGD step. At the start $y_0 := x_0$
- $z_k := z_{k-1} - \alpha\widetilde{\nabla}_k$ with $\alpha := \frac{1}{3\tau_1 L}$ and $z_0 := x_0$
  z_k plays the role of momentum. It stores all the gradients and inderectly adds a weighted sum of the gradient history into y_{k+1}. With some patience and calculations it is possible to see that basically, thanks to z_k, the influence on y_k of a fixed gradient $\widetilde{\nabla}_t$ increases as the time $s > t$ goes on.

Our candidate solution at each epoch is given by the snapshot $\tilde{x}^s$.

4.3 Katyusha in the σ-strongly convex case

If ψ is σ-strongly convex and f_i are L-smooth then the algorithm is:

Algorithm 7: Katyusha(x_0,S,σ,L)

$m := 2n$
$\tau_2 := 0.5$, $\tau_1 := min(\sqrt{\frac{m\sigma}{3L}}, 0.5)$, $\alpha := \frac{1}{3\tau_1 L}$
$y_0 := z_0 := \tilde{x}^0 := x_0$
for s = 0,...,$S-1$ **do**
  $\mu^s := \nabla f(\tilde{x}^s)$
  for k =sm ,...,$(s+1)m-1$ **do**
    $x_{k+1} := \tau_1 z_k + \tau_2 \tilde{x}^s + (1 - \tau_1 - \tau_2) y_k$
    $i :=$ random number in $\{1, 2, ..., n\}$
    $\widetilde{\nabla}_{k+1} := \mu^s + \nabla f_i(x_{k+1}) - \nabla f_i(\tilde{x}^s)$
    $z_{k+1} := \underset{z \in \mathbb{R}^d}{argmin}\ \frac{1}{2\alpha}||z - z_k||^2 + < \widetilde{\nabla}_{k+1}, z > + \psi(z)$
    Option 1: $y_{k+1} := \underset{y \in \mathbb{R}^d}{argmin}\ \frac{3L}{2}||y - x_{k+1}||^2 + < \widetilde{\nabla}_{k+1}, y > + \psi(y)$
    Option 2: $y_{k+1} := x_{k+1} + \tau_1(z_{k+1} - z_k)$
  end
  $\tilde{x}^{s+1} := (\sum_{j=0}^{m-1}(1 + \alpha\sigma)^j)^{-1} \cdot (\sum_{j=0}^{m-1}(1 + \alpha\sigma)^j \cdot y_{sm+j+1})$
end
return $\tilde{x}^S$

The algorithm has two versions, given by either choosing option 1 or option 2. In the paper they only analyze option 1, but option 2 also works.

The algorithm for $\psi \neq 0$ has a somewhat different update rule for z_k and y_k. Indeed since ψ may not be differentiable we can't use the gradient descent step to optimize it, so we have to resort to other minimization techniques (that depend on ψ itself). When setting :

$$z_{k+1} := \underset{z\in\mathbb{R}^d}{argmin}\ \frac{1}{2\alpha}||z - z_k||^2 + < \widetilde{\nabla}_{k+1}, z > +\psi(z)$$

We're basically asking to perform a gradient step to minimize f (using a stepsize α) while trying, without going too far away from the gradient step, to also minimze ψ. Observe that this minimization problem is strongly convex so it admits a unique solution. While obscure the other parameters are chosen to simplify the proof of the convergence results, or to have better convergence constants. We have the following result:

Theorem 4.1. *If each f_i is convex, L-smooth, and ψ is σ-strongly convex then Katyusha(x_0,S,σ,L) satisfies:*

$$\mathbb{E}[F(\tilde{x}^s) - F(x^*)] \leq \begin{cases} \mathcal{O}\left((1 + \sqrt{\sigma/(3LM)})^{-Sm}\right) \cdot (F(x_0) - F(x^*)) & if\ \frac{m\sigma}{L} \leq \frac{3}{4} \\ \mathcal{O}\left((1.5)^{-S}\right) \cdot (F(x_0) - F(x^*)) & otherwise \end{cases}$$

In particular Katyusha has a linear convergence rate

4.4 Katyusha^{ns} In The Non Strongly Convex Case

When ψ is only convex the algorithm we use is slightly different:

Algorithm 8: katyusha^{ns}(x_0,S,L)

$m := 2n$

$\tau_2 := 0.5$

$y_0 := z_0 := \tilde{x}^0 := x_0$

for $s = 0,...,S-1$ **do**

  $\tau_{1,s} := \frac{2}{s+4}$, $\alpha := \frac{1}{3\tau_{1,s}L}$

  $\mu^s := \nabla f(\tilde{x}^s)$

  for $k = sm, ..., (s+1)m - 1$ **do**

    $x_{k+1} := \tau_{1,s} z_k + \tau_2 \tilde{x}^s + (1 - \tau_{1,s} - \tau_2) y_k$

    $i :=$ random number in $\{1, 2, ..., n\}$

    $\widetilde{\nabla}_{k+1} := \mu^s + \nabla f_i(x_{k+1}) - \nabla f_i(\tilde{x}^s)$

    $z_{k+1} := \underset{z\in\mathbb{R}^d}{argmin}\ \frac{1}{2\alpha}||z - z_k||^2 + < \widetilde{\nabla}_{k+1}, z > +\psi(z)$

    Option 1: $y_{k+1} := \underset{y\in\mathbb{R}^d}{argmin}\ \frac{3L}{2}||y - x_{k+1}||^2 + < \widetilde{\nabla}_{k+1}, y > +\psi(y)$

    Option 2: $y_{k+1} := x_{k+1} + \tau_{1,s}(z_{k+1} - z_k)$

  end

  $\tilde{x}^{s+1} := \frac{1}{m}\sum_{j=1}^{m} y_{sm+j}$

end

return $\tilde{x}^S$

There are only 2 main differences:

1. $\tau_{1,s}$ now depends on the epoch. Indeed as the algorithm goes on we want to reduce the influence of the momentum, as we're approaching the minima and we need smaller steps to reach it.
2. The snapshot $\tilde{x}^s$ is computed as an average of the iterates y_k .

For this algorithm we have the following result:

Theorem 4.2. *If each f_i is convex, L-smooth, and ψ is convex then katyusha^{ns} (x_0,S,L) satisfies:*

$$\mathbb{E}[F(\tilde{x}^s) - F(x^*)] \leq \mathcal{O}\left(\frac{(F(x_0) - F(x^*)}{S^2} + \frac{L||x_0 - x^*||^2}{mS^2}\right)$$

In particular katyusha^{ns} needs $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ iterations to get an error $\leq \epsilon$

4.5 Katyusha^{ns} Minibatch

In the experiments of Wang [3] they use a katyusha^{ns} with minibatch. In the Katyusha paper [1], however, they do not describe a mini-batch version for Katyusha in the non strongly convex case. The SC mini-batch version is quite different from katyusha^{ns}, so, with no other information available, we opted to implement the mini-batch for katyusha^{ns} by using the same algorithm and only changing the gradient estimator to:

$$\widetilde{\nabla}_k := \nabla f(\tilde{x}^s) + \frac{1}{b}\sum_{i \in S_k}(\nabla f_i(x_k) - \nabla f_i(\tilde{x}^s))$$

Where S_k is a list of b indices in $\{1, ..., n\}$ randomly and independently generated.
For this choice of gradient the previous convergence result still holds. Indeed the proof only asks us $\widetilde{\nabla}_k$ to be unbiased and to respect the variance upper bound of [1][lemma 2.4] :

Proposition 4.3 (Variance upper Bound)**.** *For the minibatch gradient estimator the variance upper bound of [1][lemma 2.4] still holds:*

$$\mathbb{E}\left[||\widetilde{\nabla}_{k+1} - \nabla f(x_{k+1})||^2\right] \leq 2L \cdot (f(\tilde{x}^s) - f(x_{k+1}) - < \nabla f_i(x_{k+1}), \tilde{x} - x_{k+1} >)$$

Proof. Recall that the indices in $S_k = \{i_1, ..., i_b\}$ are independent and randomly generated by a uniform distribution on $\{1, ..., n\}$. Let us indicate with $\widetilde{\nabla}^{i_s}_{k+1}$ the gradient estimator of normal Katyusha for a random index i_s. We have that:

$$\begin{aligned}
&\mathbb{E}\left[||\widetilde{\nabla}_{k+1} - \nabla f(x_{k+1})||^2\right] = \mathbb{E}\left[< \widetilde{\nabla}_{k+1} - \nabla f(x_{k+1}), \widetilde{\nabla}_{k+1} - \nabla f(x_{k+1}) >\right] = \\
&= \mathbb{E}\left[< \frac{1}{b}\sum_{s=1}^{b} \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}), \frac{1}{b}\sum_{s=1}^{b} \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}) >\right] = \\
&= \frac{1}{b^2}\sum_{s=1}^{b} \mathbb{E}\left[< \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}), \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}) >\right] + \\
&+ \frac{1}{b^2}\sum_{s \neq j}^{b} \mathbb{E}\left[< \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}), \widetilde{\nabla}^{i_j}_{k+1} - \nabla f(x_{k+1}) >\right] \overset{\text{(independence)}}{=} \\
&= \frac{1}{b^2}\sum_{s=1}^{b} \mathbb{E}\left[< \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}), \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}) >\right] + \\
&+ \frac{1}{b^2}\sum_{s \neq j}^{b} < \mathbb{E}\left[\widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1})\right], \mathbb{E}\left[\widetilde{\nabla}^{i_j}_{k+1} - \nabla f(x_{k+1})\right] > \overset{\text{(unbiased estimator)}}{=} \\
&= \frac{1}{b^2}\sum_{s=1}^{b} \mathbb{E}\left[< \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}), \widetilde{\nabla}^{i_s}_{k+1} - \nabla f(x_{k+1}) >\right] \overset{\text{(lemma 2.4)}}{\leq} \\
&\leq \frac{1}{b^2}\sum_{s=1}^{b} 2L \cdot (f(\tilde{x}^s) - f(x_{k+1}) - < \nabla f_i(x_{k+1}), \tilde{x} - x_{k+1} >) = \frac{1}{b} 2L \cdot (f(\tilde{x}^s) - f(x_{k+1}) - < \nabla f_i(x_{k+1}), \tilde{x} - x_{k+1} >)
\end{aligned}$$

□

As expected the variance upper bound not only is respected, but it's even tighter. Indeed the goal of the minibatch approach is to reduce the variance of the gradient estimator.

4.6 Proximal minimizer

When implementing katyusha one needs to also implement a proximal minimizer, i.e. a function that solves the minimization problem:

$$z_{k+1} := \underset{z\in\mathbb{R}^d}{argmin}\ \frac{1}{2\alpha}||z - z_k||^2 + < \widetilde{\nabla}_{k+1}, z > +\psi(z)$$

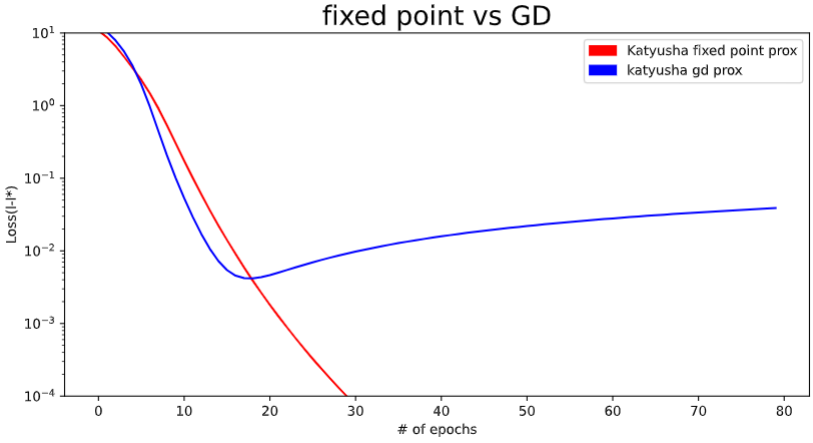
In the case of ψ differentiable one may use a gradient descent method to solve the minimization problem. Since ψ is usually a regularization term this is very efficient, as the gradient computation is quite quick. In our implementation of Katyusha, however, we decided to use a fixed point method to have a more readable and easier code (as we'll see later, having one less hyperparameter to choose is quite convenient). If z_{k+1} is the minimum of:

$$g(z) := \frac{1}{2\alpha}||z - z_k||^2 + < \widetilde{\nabla}_{k+1}, z > +\psi(z)$$

then surely $\nabla g(z_{k+1}) = 0$. We may rewrite this condition as follows:

$$\nabla g(z) = \frac{1}{\alpha}(z - z_k) + \widetilde{\nabla}_{k+1} + \nabla\psi(z) = 0 \iff z = z_k - \alpha\widetilde{\nabla}_{k+1} - \alpha\nabla\psi(z) =: G(z)$$

For α small enough (so for a small enough stepsize) G is locally a contraction. So starting from $z^0 := z_k - \alpha\widetilde{\nabla}_{k+1}$ the sequence $z^{n+1} := G(z^n)$ should converge to z_{k+1}. Even though the fixed point method may have similar performance (or even worse) than a GD method it has the advantage of not having to choose a stepsize to perform the minimization. Since this minimization problem is solved at each Katyusha iteration, and it always has a different form, choosing a proper stepsize for the GD method may be troublesome:



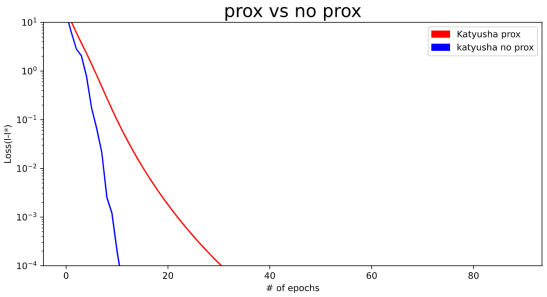
As we can see from the above image, Katyuhsa with a (fixed stepsize) GD proximal minimizer enjoys better performance at first, but then it's not able to optimize properly (due to the wrong stepsize) and it's overtaken by the fixed point method. One may implement a more proper line search or a complex and arbitrary heuristic to make the GD minimizer work, but the added complexity is not worth the effort when the fixed point has a similar performance and none of the complications.

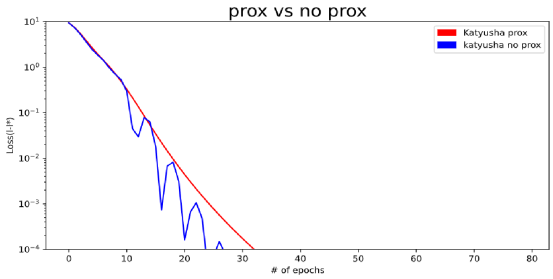
4.7 Proximal Minimizer vs No Proximal Minimizer

If ψ is smooth one may skip the proximal minimization step and perform a simple GD like step by redefining the problem as:

$$\min_{x\in\mathbb{R}^d} \tilde{F}(x) \quad where \quad \tilde{F}(x) := \frac{1}{n}\sum_{i=1}^{n} \tilde{f}_i(x) := \frac{1}{n}\sum_{i=1}^{n}(f_i(x) + \psi(x))$$

In this case, referring to the original formulation, one may write $\tilde{F}(x) := \tilde{f}(x) + \tilde{\psi}(x)$ where $\tilde{\psi} \equiv 0$. Then the proximal minimizer is given by the simple gradient step: $z_{k+1} = z_k - \alpha\widetilde{\nabla}_{k+1}$. From now on when we talk about Katyusha we refer to Katyusha applied to $\tilde{F}$, and when we talk about Katyusha prox we intend Katyusha applied to F of the original formulation.





The version with the (proper) proximal minimizer tends to be more stable, while the one without tends to have better performance, both epoch and time wise.

5 Experiments

5.1 The Datasets

In this section, we compare the performance of our algorithms on 4 datasets:

1. The first dataset is called a9a, also known as "Census Income" dataset. Our objective is predicting, based on census data, whether the income of an adult exceeds \$50K/yr, so the target variable belongs to {-1,1}. In this dataset, continuous features are discretized into quantiles, and are represented by binary features.
2. The second dataset is called w8a and has the same structural characteristics of a9a. In this case we don't have specific informations about the classification task.
3. The third dataset is about binary classification in surgical data for detecting a particular disease.
4. The last one is about wine quality prediction. The target feature is discrete, which makes it suitable for regression tasks.

5.2 Experiment Guidelines

The algorithms we will use are the ones previously described: SVRG, SARAH, Spiderboost Momentum and Katyusha. For each experiment, we initialize all the algorithms at the same point that is generated randomly from the normal distribution. Also, we choose a fixed mini-batch size 256 and set the epoch length q to be $2n/256$ such that all algorithms pass over the entire dataset twice in each epoch.

We first apply these algorithms to solve logistic regression with a smooth non-convex regularizer:

$$\psi(x) := \gamma \cdot \sum_{i=1}^{d} \frac{x^2}{1+x^2}$$

For all the algorithms considered, we set their stepsizes to $\eta = 0.05$ for the first 2 dataset, and $\eta = 0.01$ for the last 2, and a common regularization parameter $\gamma = 0.1$.

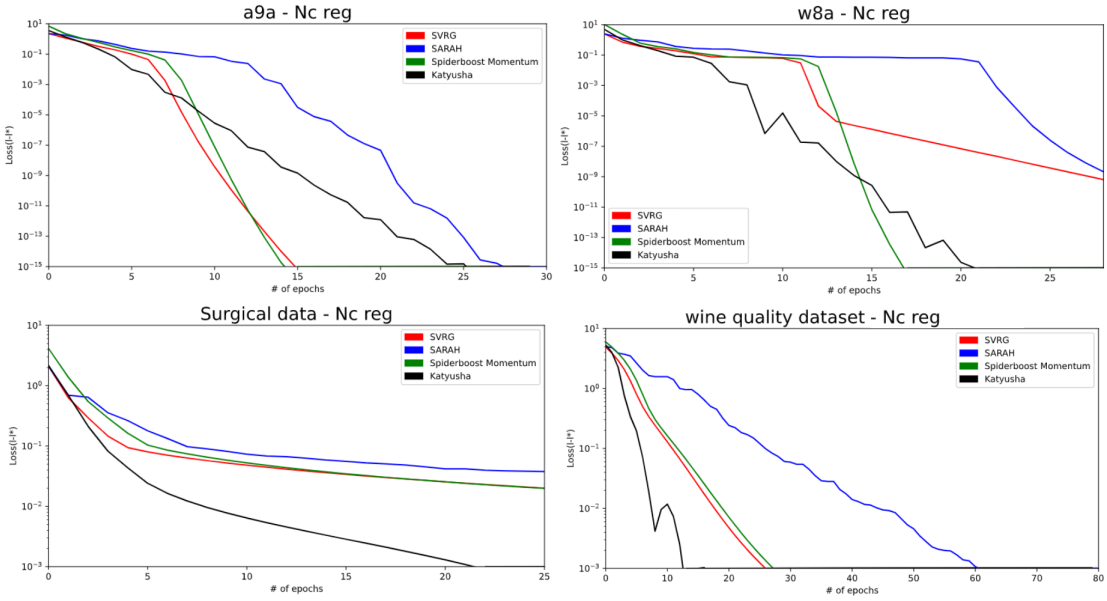
Then we solve a robust linear regression with a l_1-nonsmooth regularizer:

$$\psi(x) := \gamma \cdot ||x||_1$$

of strength $\gamma = 0.1$ and a common stepsize of $\eta = 0.05$ on the first 2 dataset, $\eta = 0.01$ in the third one, and $\eta = 0.1$ in the last one.

5.3 Logistic Regression Non-convex regularization experiments

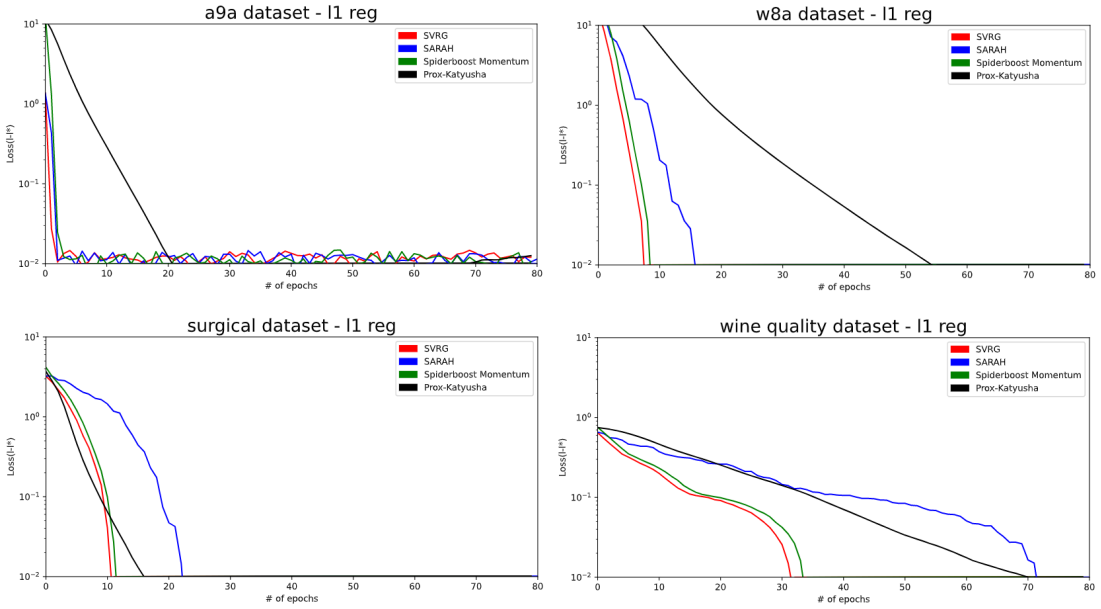
We start by analyzing the algorithms for the logistic regression:



SVRG and Spiderboost consistently have the best performances, occasionally overtaken by Katyusha, while SARAH has the worst performance. We suspect that SARAH poor performance is due to the random re-initialization at the start of each epoch. Unlike the paper [3] where spiderboost momentum significantly outperforms SVRG, our implementation of Spiderboost-M has similar performance to SVRG. We suspect that the main difference is in the momentum parameters that, unfortunately, are not specified in the paper.

5.4 Robust Regression l_1-regularizer

We analyze the algorithms for the robust regression with l_1-regularizer :



In this case, even though the regularization is not smooth, we approximate the gradient of ψ as:

$$\nabla\psi(x) := sign(x)$$

For this choice the algorithms converge anyway. For Katyusha-prox we had to use a different stepsize of η_{kat} because the proximal minimizer wouldn't work with a large stepsize of η (recall that for big stepsizes the fixed point function involved in the proximal minimization may not be a contraction).

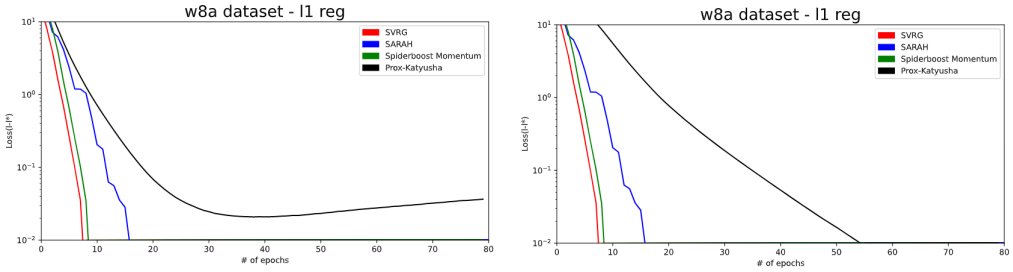
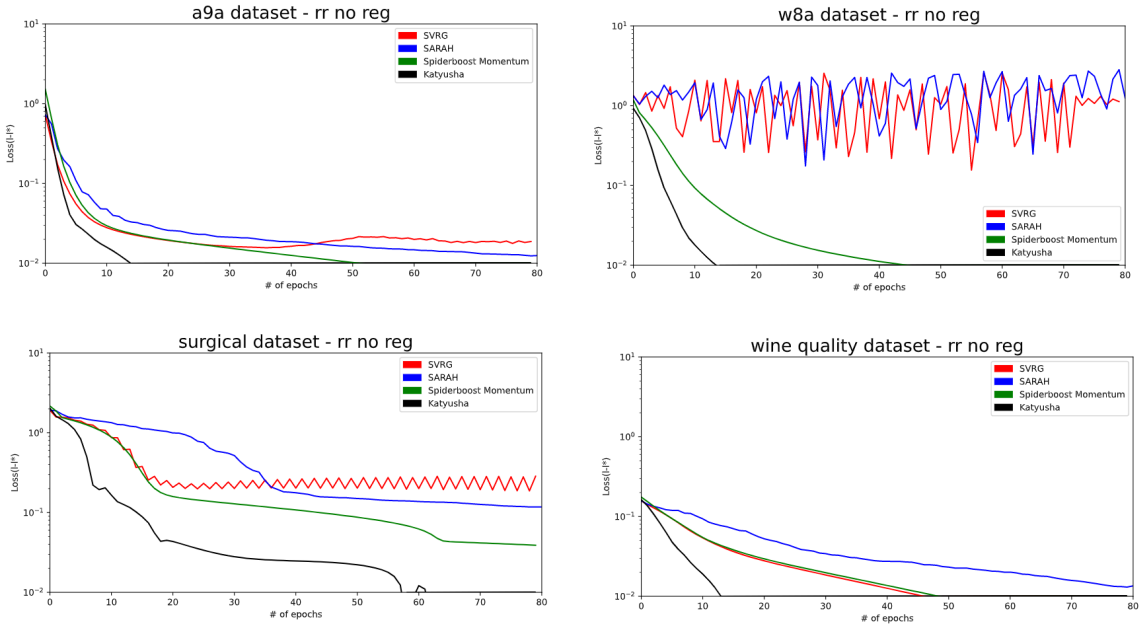


Figure 1: on the left $\eta_{kat} = 0.01$, on the right $\eta_{kat} = 0.001$

We also observed that a more specific tuning of the stepsize significantly improved performances for Katyusha. This may be due to the fact that Katyusha has quite a different structure from the other algorithms. Indeed we think that using a share stepsize for all the 4 algorithms is somewhat unfair to Katyusha.

5.5 Robust Regression with no regularizer

We analyze the algorithms for the robust regression:



In this case Katyusha consistently performs better than other algorithms. SVRG and SARAH have some issues, probably due to a wrong stepsize.

5.6 Conclusions

From our experiments we have concluded that Spiderboost and Katyusha are the most consistent algorithms. SARAH seems to perform worse than SVRG, which most of the time, has performance similar to Spiderboost.
While more stable we also observe that prox-Katyusha has worse performance than Katyusha, both epoch and time-wise. Indeed the proximal minimization step is quite costly and complex to implement. Without a doubt a more proper implementation of the proximal minimizer could solve this issue, but the poor performance improvements may not be worth the effort.

References

[1] Zeyuan Allen-Zhu. "Katyusha: The first direct acceleration of stochastic gradient methods". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 8194–8244.

[2] Lam M Nguyen et al. "SARAH: A novel method for machine learning problems using stochastic recursive gradient". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2613–2621.

[3] Zhe Wang et al. "SpiderBoost and momentum: Faster stochastic variance reduction algorithms". In: *arXiv preprint arXiv:1810.10690* (2018).