Overview of non-convex optimization and adaptive methods for stochastic optimization

Eduard Gorbunov (MIPT ed-gorbunov@yandex.ru)
Gasnikov Alexander (MIPT gasnikov@yandex.ru)

Main references:

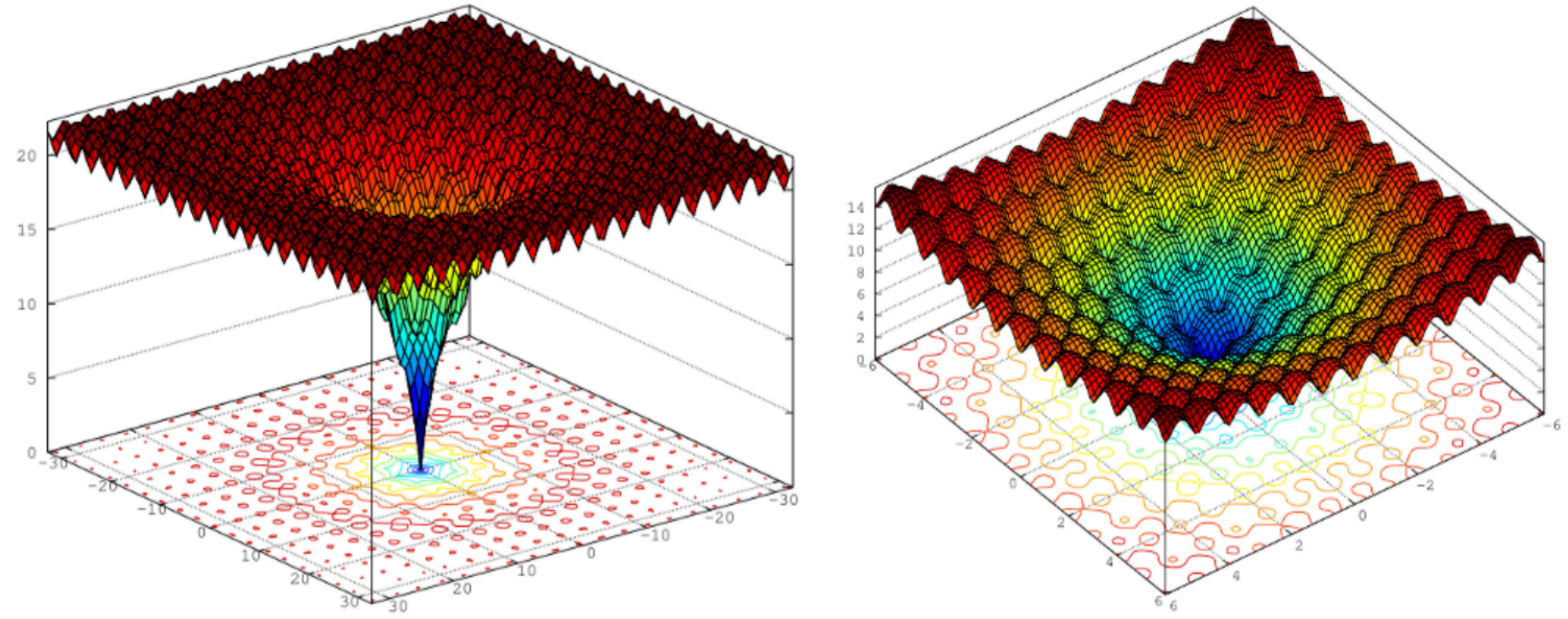
Bottou L., Curtis F.E., Nocedal J. Optimization methods for large-scale machine learning // arXiv.org e-Print archive. 2016. – URL: https://arxiv.org/pdf/1606.04838.pdf
Jain P., Kar P. Non-convex optimization for machine learning // arXiv.org e-Print archive. 2017. – URL: https://arxiv.org/pdf/1712.07897.pdf
Lan G. Lectures on optimization. Methods for Machine Learning // e-print, 2019. – URL: http://pwp.gatech.edu/guanghui-lan/wp-content/uploads/sites/330/2019/02/LectureOPTML.pdf
Ruder A. An overview of gradient descent optimization algorithms // arXiv.org e-Print archive. 2017. – URL: https://arxiv.org/pdf/1609.04747.pdf
Wright S. Optimization algorithms for Data Science // IAS/Park City Mathematics Series. – 2016. URL: http://www.optimization-online.org/DB_FILE/2016/12/5748.pdf

General non convex optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

$$f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

Dimensions:	n
Domain:	$-32.768 \le x_i \le 32.768$
Global Optimum:	$f(x) = 0.0$ at $x = (0.0, 0.0, \ldots, 0.0)$
Operator:	AckleyEvaluator
Charts:	



(a) [-32.768, 32.768] (b) [-6.0, 6.0]

Figure 1: Ackley function plots.

Figure:
`https://dev.heuristiclab.com/trac.fcgi/wiki/Documentation/Reference`

Bad news!

To find ϵ global minimum in general one should takes

$$N(\epsilon) \sim \epsilon^{-n/2}$$

Calculations of smooth f and its derivatives in requested point

Note 1: Below we demonstrate «worth in the world» function philosophy to obtained this bound

Note 2:«Worth in the world» function in our case will be quasi-convex; the problem with this function is that we have only local oracle that return local information (value and (high-order) derivatives at the required point - nothing more).

Lower bound for global optimization

$$f(x) \to \min_{x \in [0,1]^n} .$$

Assume that for $x, y \in [0,1]^n$

$$\left\| \nabla f(y) - \nabla f(x) \right\|_2 \le L \left\| y - x \right\|_2 .$$

Let's divide cube $[0,1]^n$ on small sub-cubes with side length

$$4\sqrt{2\varepsilon/L} .$$

Let's put $f(x) \equiv 0$ everywhere except one sub-cubes that was observed by considered algorithm at the very end.

Lower bound for global optimization

In this sub-cube we can determine $f(x)$ such that:

$$\min_{x\in[0,1]^n} f(x) = -2\varepsilon .$$

So to find such x^N that guarantee (ε-solution)

$$\boxed{f(x^N) - f(x_*) \le \varepsilon}$$

algorithm need to visit each sub-cubes where $f(x) \equiv 0$ at least on time. Therefore the lower bound on N is

$$\boxed{N \sim \left(1/\sqrt{\varepsilon}\right)^n = \varepsilon^{-n/2}.}$$

Sometimes tuned simulated annealing
works good in practice
for significantly non-convex problems!

Simulated annealing

$$f(x) \to \min_{x \in \mathbb{R}^n}.$$

The main idea: consider noisy dynamic $dx/dt = -\nabla f(x)$:

$$dx(t) = -\nabla f(x) + \sqrt{2T}\, dW(t),$$

where $W(t)$ – Wiener process. When $t \to \infty$ one can prove that the distribution of vector $x(t)$ tends to distribution with density function

$$\frac{\exp(-f(x)/T)}{\int \exp(-f(y)/T)dy}.$$

When $T \to 0+$ this distribution concentrate around global minimum x_*.

Simulated annealing (Langevin dynamics)

So we can consider

$$dx(t) = -\nabla f(x) + \sqrt{2T(t)}\,dW(t)$$

with different policy of choosing $T(t)$, such that,

$$T(t) \xrightarrow[t\to\infty]{} 0+.$$

For example, in practice rather popular is the following strategy

$$T(t) = \frac{c}{\ln(2+t)}.$$

Structural parameter c significantly affected on the global convergence.

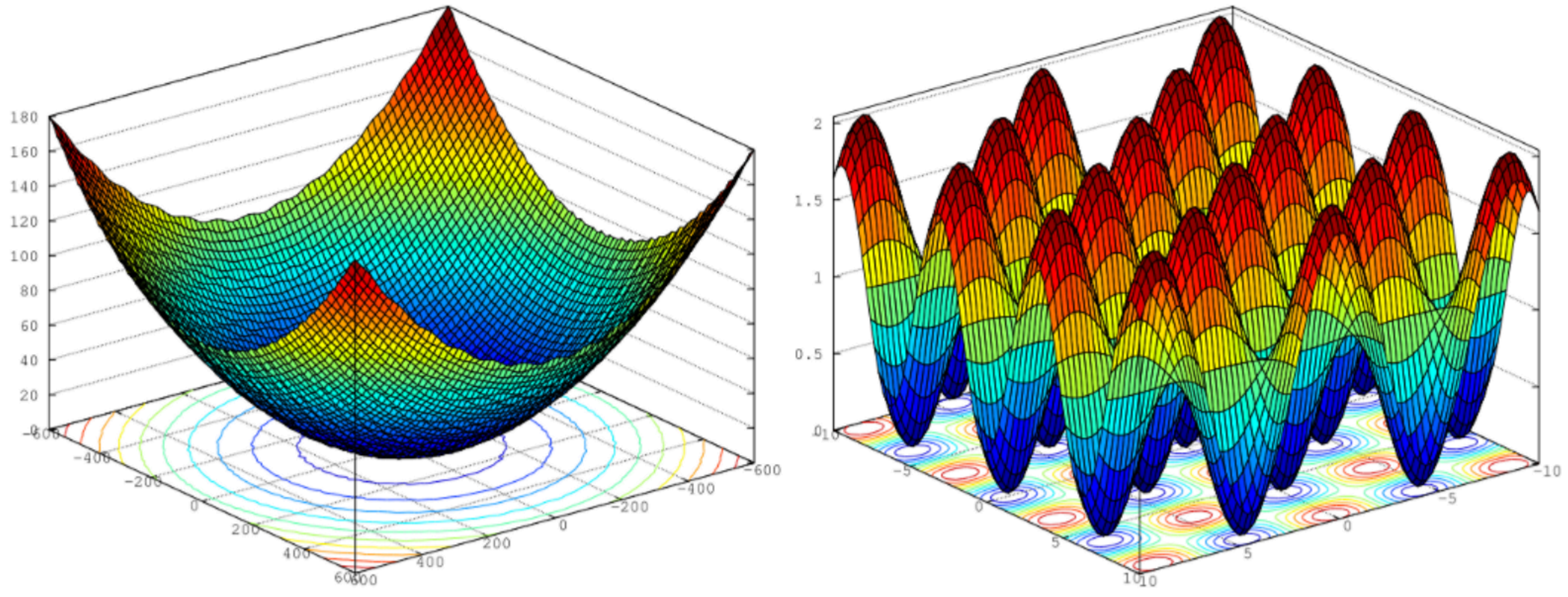
Non asymptotic results see in Xu–Chen–Zou–Gu, NeurIPS, 2018 (7575).

Good news: sometimes it's sufficient to find local minimum

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$$

Matrix completion problem

Dimensions:	n
Domain:	$-600.0 \leq x_i \leq 600.0$
Global Optimum:	$f(x) = 0.0$ at $x = (0.0, 0.0, \ldots, 0.0)$
Operator:	GriewankEvaluator
Charts:	



(a) [-600.0, 600.0] (b) [-10.0, 10.0]

Figure 4: Griewank function plots.

Figure:
https://dev.heuristiclab.com/trac.fcgi/wiki/Documentation/Reference

Let's start with the problem:
To find ϵ-stationary point

Local minimum belongs to the set of stationary points since we consider unconstrained problems

Gradient descent

$$\boxed{f(x) \to \min_{x \in \mathbb{R}^n}.}$$

$$\frac{dx}{dt} = -\nabla f(x).$$

Let's show that $W(x) = f(x)$ is Lyapunov function:

$$\frac{dW(x(t))}{dt} = \left\langle \nabla f(x(t)), \frac{dx(t)}{dt} \right\rangle = ,$$

$$= \left\langle \nabla f(x(t)), -\nabla f(x(t)) \right\rangle = -\left\| \nabla f(x(t)) \right\|_2^2 \leq 0.$$

Gradient descent

$$f(x) \to \min_{x \in \mathbb{R}^n}.$$

Euler discretization of $dx/dt = -\nabla f(x)$ has the form

$$\boxed{x^{k+1} = x^k - h\nabla f\left(x^k\right).}$$

What is about h ? Assume that for $x, y \in \left\{x \in \mathbb{R}^n : f(x) \le f\left(x^0\right)\right\}$

$$\left\|\nabla f(y) - \nabla f(x)\right\|_2 \le L\left\|y - x\right\|_2 .$$

Then (this result can also be obtained from physical dimension considerations)

$$\boxed{h = 1/L.}$$

Gradient descent

Since

$$f\left(x^{k+1}\right) \le f\left(x^{k}\right) - \frac{1}{2L}\left\|\nabla f\left(x^{k}\right)\right\|_2^2$$

we can conclude that (ε-extremum)

$$\boxed{\min_{k=1,\dots,N}\left\|\nabla f\left(x^{k}\right)\right\|_2 \le \varepsilon}$$

when

$$N = \frac{2L\cdot\left(f\left(x^{0}\right) - f\left(x^{extr}\right)\right)}{\varepsilon^2}.$$

If there available inexact gradient and (or) we have weaker assumption on smoothness, see P. Dvurechensky (2017) and references therein.

Gradient descent

This bound

$$N \sim \varepsilon^{-2} \text{ (first-order method, Lipschitz gradient)}$$

is unimprovable up to a constant factor for arbitrary first-order method. But if we assume high-order smoothness of $f(x)$, then lower bound for ε-extremum will have the form (*Carmon Y., Duchi J.C., Hinder O., Sidford A., 2017*)

$$N \sim \varepsilon^{-8/5} \text{ (first-order method, Lipschitz high-order derivatives).}$$

Moreover, if we additionally can use p-order derivatives in algorithm, then optimal bound will have the form (*Birgin E., Gardenghi J., Martinez J. et al., 2016*)

$$N \sim \varepsilon^{-(p+1)/p} \text{ (p-order method, Lipschitz p-order derivatives).}$$

Note: If $n = \dim x$ is small, gradient descent is not optimal, see https://arxiv.org/pdf/2001.02968.pdf for details

Gradient descent

Let's indicate the main drawbacks of these types of results:

1. The results are not global. Moreover, we cannot guarantee that we can find even local minimum. Indeed (Yu. Nesterov, 2004), for

$$f(x_1,x_2)=\frac{1}{2}(x_1)^2+\frac{1}{2}(x_2)^4-\frac{1}{2}(x_2)^2$$

choose $x^0=(1,0)^T$, then $x^k \xrightarrow[k\to\infty]{} (0,0)^T$ – saddle-point.

2. We consider unconstraint optimization problem. It's significant! For example, if we consider $f(x)\to\min\limits_{x\in[0,1]^n}$, the lower bound for ε-extremum has the form $N\sim\varepsilon^{-n/2}$ (Yu. Nesterov, 2012).

Local minimum belongs to the set of stationary points if we consider unconstrained problems.
But unfortunately, typically stationary point (extremum point) is saddle-point, that is far from to be local minimum. So we'd like to have guarantees to converge to local min.
Fortunately, standard method (especially randomized ones) converges to local minimum.
But could vast time in vicinity of saddle-point

Convergence to local minimum

$$f(x) \to \min_{x \in \mathbb{R}^n}.$$

Following by Yu. Nesterov and B. Polyak (2006) let's denote $x^N - (\varepsilon, \delta)$ local minimum if

$$\left\| \nabla f\left(x^N\right) \right\|_2 \le \varepsilon,\ \lambda_{\min}\left(\nabla^2 f\left(x^N\right)\right) \ge -\delta.$$

Assume that for $x, y \in \left\{ x \in \mathbb{R}^n : f(x) \le f\left(x^0\right) \right\}$

$$\left\| \nabla f(y) - \nabla f(x) \right\|_2 \le L_1 \left\| y - x \right\|_2,$$

$$\left\| \nabla^2 f(y) - \nabla^2 f(x) \right\|_2 \le L_2 \left\| y - x \right\|_2.$$

Convergence to local minimum

$$x^{k+1} = x^k - \frac{1}{L_1}\nabla f\left(x^k\right), \text{ if } \left\|\nabla f\left(x^k\right)\right\|_2 > \varepsilon;$$

$$x^{k+1} = x^k + hp^k, \text{ if } \left\|\nabla f\left(x^k\right)\right\|_2 \le \varepsilon \text{ and } \lambda_{\min}^k = \lambda_{\min}\left(\nabla^2 f\left(x^k\right)\right) < -\delta,$$

where $h = 2\delta/L_2$, p^k – eigen vector of $\nabla^2 f\left(x^k\right)$ corresponds to $\lambda_{\min}^k$:

$$\left\langle \nabla f\left(x^k\right), p^k \right\rangle \le 0, \ \left\|p^k\right\|_2 = 1.$$

This method will stop (find (ε, δ) local minimum) after N iterations

$$N \le N(\varepsilon,\delta) = \left(f\left(x^0\right) - f\left(x^{\text{local min}}\right)\right) \cdot \max\left\{\frac{2L_1}{\varepsilon^2}, \frac{3L_2^2}{2\delta^3}\right\}.$$

This is not an optimal method, but it is good for explanation the idea.

Convergence to local minimum

Indeed,

$$f\left(x^{k+1}\right) \le f\left(x^k\right) + h\underbrace{\left\langle \nabla f\left(x^k\right), p^k \right\rangle}_{\le 0} + \frac{h^2}{2!}\underbrace{\left\langle \nabla^2 f\left(x^k\right) p^k, p^k \right\rangle}_{\le -\frac{h^2\delta}{2}} + \frac{h^3}{3!}\underbrace{L_2 \left\| p^k \right\|_2^3}_{= M_2} \le$$

$$\le f\left(x^k\right) - \frac{1}{2}\left(\frac{2\delta}{L_2}\right)^2 \delta + \frac{L_2}{6}\left(\frac{2\delta}{L_2}\right)^3 = f\left(x^k\right) - \frac{2}{3}\frac{\delta^3}{L_2^2}.$$

Note, that it's sufficiently to indicate that $\lambda_{\min}\left(\nabla^2 f\left(x^k\right)\right) \ge -\delta$, or to find such p^k ($\left\|p^k\right\|_2 = 1$), that $\left\langle \nabla^2 f\left(x^k\right) p^k, p^k \right\rangle \le -\delta/2$. One can do it by Lanczos method with $\sim 1/\sqrt{\delta}$ matrix-vector $\nabla^2 f\left(x^k\right) p$ multiplications.

Convergence to local minimum

Note, that by using (Krylov's type methods)

$$\nabla^2 f(x)\mathrm{v} = \nabla \left\langle \nabla f(x), \mathrm{v} \right\rangle,$$

$$\nabla^2 f(x)\mathrm{v} \approx \frac{\nabla f(x+\tau \mathrm{v}) - \nabla f(x)}{\tau}$$

and automatic differentiations one can calculate $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)\mathrm{v}$ for almost the same time (up to a constant 4 – 16) for all reasonable function with known computational tree.

Note: Automatic differentiation is now widespread: *Torch*, *Caffe*, *Theano*, *TensorFlow*. But only the last two libraries support the desired high-order differentiation.

Convergence to local minimum

One can improve the ε-part of the bound on $N(\varepsilon,\delta)$ from $\sim\varepsilon^{-2}$ to $\sim\varepsilon^{-3/2}$ by using cubic regularized Newton method (Yu. Nesterov, B. Polyak, 2006; Grapilia–Netserov, 2019). For that, we need to replace

$$x^{k+1} = x^k - \frac{1}{L_1}\nabla f\left(x^k\right) = \arg\min_{x\in\mathbb{R}^n}\left\{ f\left(x^k\right) + \left\langle \nabla f\left(x^k\right), x - x^k \right\rangle + \frac{L_1}{2!}\left\|x - x^k\right\|_2^2 \right\}$$

by (the complexity of this iteration is comparable with Newton's iteration)

$$x^{k+1} = \arg\min_{x\in\mathbb{R}^n}\left\{ f\left(x^k\right) + \left\langle \nabla f\left(x^k\right), x - x^k \right\rangle + \frac{1}{2!}\left\langle \nabla^2 f\left(x^k\right)\left(x - x^k\right), x - x^k \right\rangle + \frac{L_2}{3!}\left\|x - x^k\right\|_2^3 \right\}.$$

Recall that for second-order methods this bound $\sim\varepsilon^{-3/2}$ is unimrovable. Note, that for cubic regularized Newton method we can skip $x^{k+1} = x^k + hp^k$.

From the results above one may though, that if we have only one stationary point (= local minimum = global minimum), then it's not a problem to find it.

This is true if we mean criteria norm of the gradient, but this is in general false if we consider the criteria discrepancy in function!

The meaning of criteria

For the moment for

$$f(x) \to \min_{x \in \mathbb{R}^n}$$

we've considered two criteria of quality of approximate solution:

$$\min_{k=1,\ldots,N} \left\| \nabla f\left(x^k\right) \right\|_2 \le \varepsilon$$

and

$$f\left(x^N\right) - f\left(x_*\right) \le \varepsilon.$$

It seems that if $f(x)$ has unique extremum = local minimum (that is consequently global minimum), then these criteria are close to each other.

The meaning of criteria

(Yu. Nesterov, V. Skokov, 1980)

$$f(x) = \frac{1}{4}(x_1 - 1)^2 + \sum_{i=1}^{n}\left(x_{i+1} - 2x_i^2 + 1\right)^2 =$$

$$= \frac{1}{4}(x_1 - 1)^2 + \sum_{i=1}^{n}(x_{i+1} - \underbrace{P_2(x_i)}_{\text{Chebyshev polynom}})^2.$$

At point

$$x_* = (1,1,\dots,1)\ (f(x_*) = 0),$$

we have unique extremum = global minimum of $f(x)$.

The meaning of criteria

If we put

$$x^0 = \left(-1, 1, \ldots, 1\right)^T \; \left(f\left(x^0\right) = 1\right),$$

then gradient type methods guarantee small norm of the gradient, but not small discrepancy in function value. In all experiments (with different GD type methods and second-order type methods) under $n = 15$ we have

$$\left\|\nabla f\left(x^N\right)\right\|_2 \approx 10^{-8},$$

but

$$f\left(x^N\right) - f\left(x_*\right) \approx 1/2.$$

For the moment we consider the only first order method - gradient descent. It's optimal for finding stationary points without any additionally assumptions on problem formulations (high-smoothness). But we know from convex optimization that momentum (impulse) schemes accelerate the convergence. What is about non-convex case?

Polyak's heavy ball method

The considered above dynamical system $dx/dt = -\nabla f(x)$ don't have a mechanical intuition. In 1964 B. Polyak propose the following generalization

$$\mu \frac{d^2 x}{dt^2} = -\nabla f(x) - p \frac{dx}{dt}.$$

The discrete variant of this system has a form

$$x^{k+1} = x^k - h\nabla f\left(x^k\right) + \beta\left(x^k - x^{k-1}\right).$$

For (strongly) convex functions this method under proper choice of parameters locally converges with best possible rates (see below). Globally it converges in Chesaro sense like standard GD (E. Ghadimi et al., 2014).

Visualization: https://distill.pub/2017/momentum/

Polyak's heavy ball method (non convex case)

In 1981 A. Grienwank show that if we consider

$$\mu(t)\frac{d^2x}{dt^2} = -\nabla f(x) - p(t)\frac{dx}{dt},$$

where $\mu(t) \sim f(x(t)) - c$ $(c > f(x_*))$ and $p(t) = F\left(\nabla f(x(t))\right)$ with special choice of $F(\ )$, then $x(t)$ when $t \to \infty$ converges to such local minimum x^{loc}, that $f\left(x^{loc}\right) \le c$.

In 2019 E. Diakonikolas and M. Jordan show that for discrete dynamic under special parameters selection one can guarantee

$$\min_{k=1,\dots,N} \left\|\nabla f\left(x^k\right)\right\|_2^2 \le \frac{2L\Delta f}{N}.$$

Nesterov's fast gradient (momentum) method

In 1983 in PhD thesis Yu. Nesterov (supervisor was B. Polyak) proposed the methods of type

$$x^1 = x^0 - h\nabla f\left(x^0\right),$$

$$x^{k+1} = x^k - h\nabla f\left(x^k + \beta_k\left(x^k - x^{k-1}\right)\right) + \beta_k\left(x^k - x^{k-1}\right).$$

For convex function this method converges under

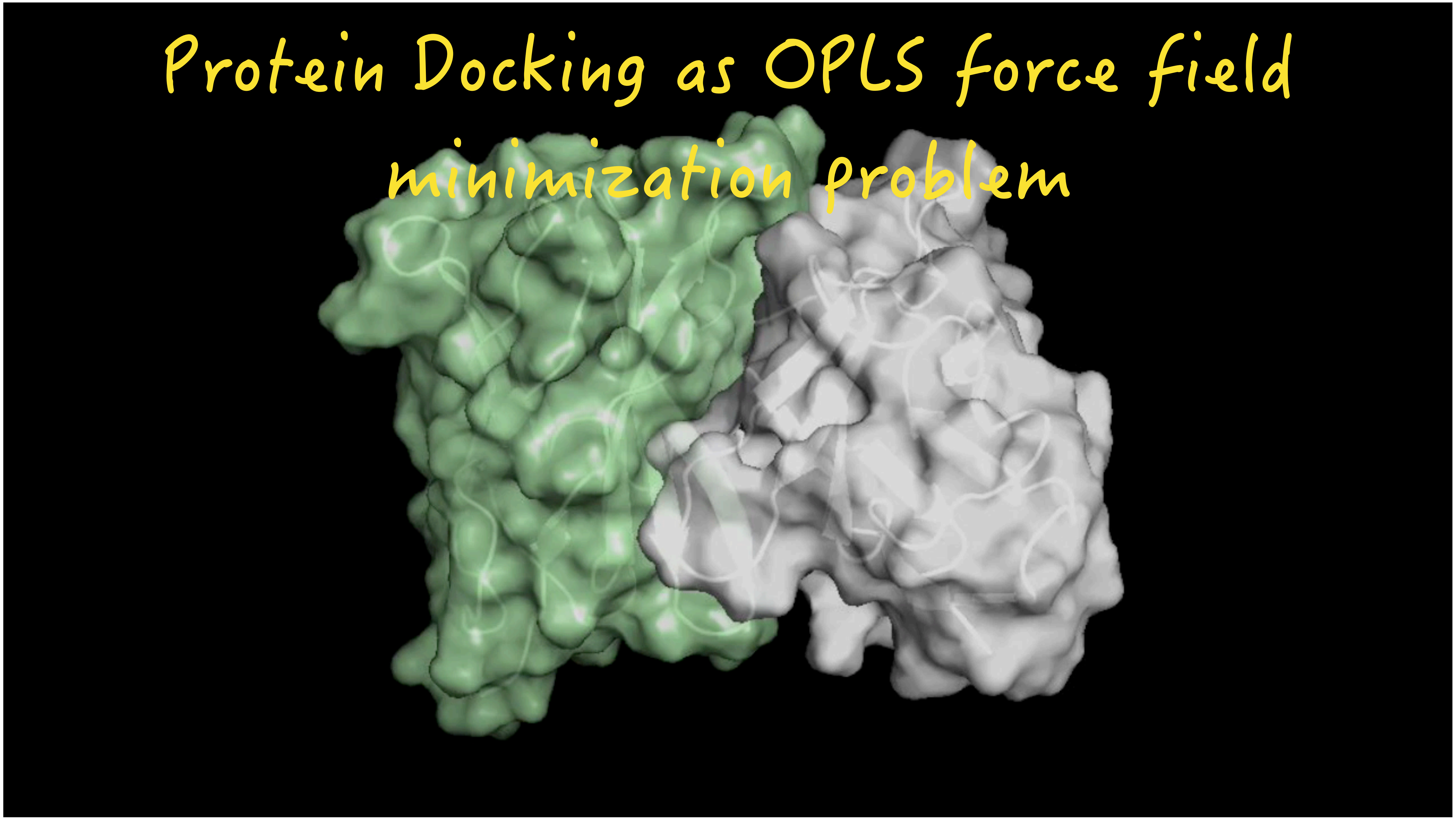
$$h = \frac{1}{L},\ \beta_k = \frac{k-1}{k+2}:$$

$$f\left(x^N\right) - f\left(x_*\right) \le \frac{2LR^2}{N^2}.$$

The same results demonstrate heavy-ball method (with proper parameters) in average

https://arxiv.org/pdf/2002.04664.pdf

No acceleration in theory for non convex problems, but sometimes FGM works!



Data Science applications

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x})$$

Math. Statistic. point of view

Assume the observed x, y related according to the model: $y = F(x, \theta) + \xi$. Where $F(x, \theta)$ – is known function (say, Neural Networks) with unknown vector of parameters θ (weights) and ξ – is a Gaussian noise (unbiased). If we have many observations $y_k = F(x_k, \theta) + \xi_k$, $k = 1, ..., m$, where ξ_k – i.i.d., then the best possible way (Fisher's theorem) to estimate θ is:

$$\frac{1}{m} \sum_{k=1}^{m} |y_k - F(x_k, \theta)|^2 \to \min_{\theta \in \mathbb{R}^n} .$$

Machine Learning point of view

Assume that we want to find θ that explained observed y_k, x_k according to some unknown probabilistic low. We know nothing about this low. Assume that the quality of θ is measured according to the loss function $f(\theta) = \mathbf{E}_{(x,y)}|y - F(x,\theta)|^2$. Since we don't know probabilistic low, we couldn't minimize $f(\theta)$:

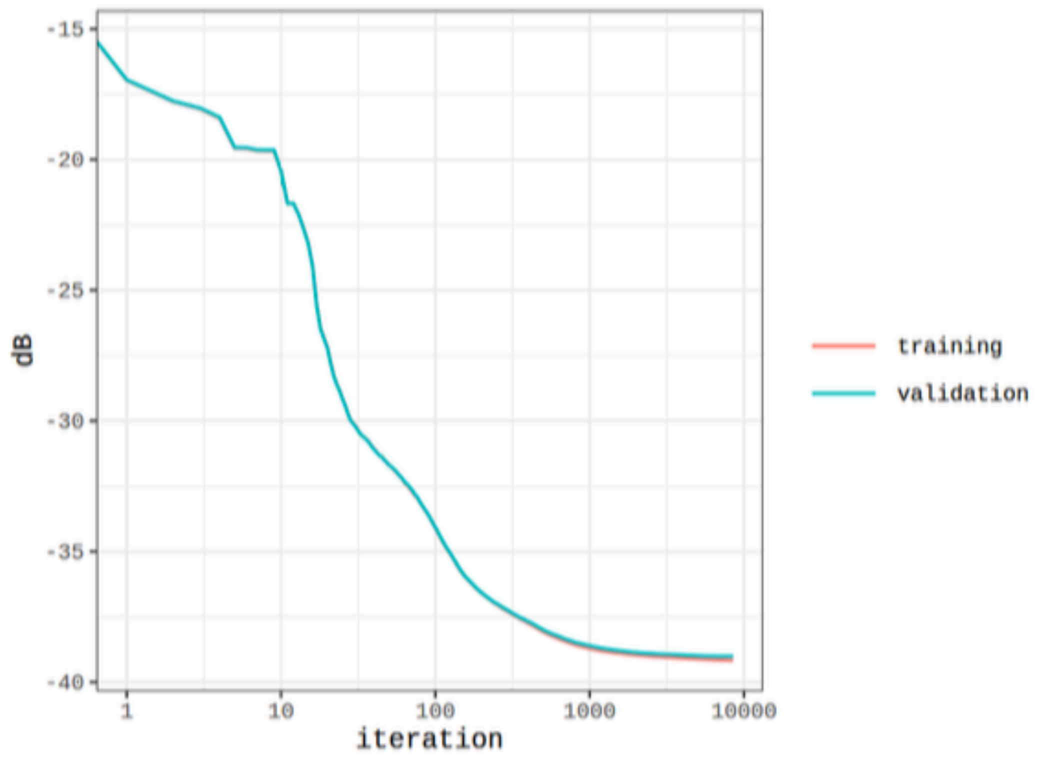
$$f(\theta) = \mathbf{E}_{(x,y)} f((x,y),\theta) \to \min_{\theta \in \mathbb{R}^n} . \tag{2}$$

But we know realizations of random (y, x): $\{y_k, x_k\}_{k=1}^m$. Based on Monte Carlo approach we can build Stochastic Average Approximation (SAA) of (2):

$$\tilde{f}(\theta) = \frac{1}{m}\sum_{k=1}^{m} \tilde{f}((x_k, y_k),\theta) = \frac{1}{m}\sum_{k=1}^{m} |y_k - F(x_k,\theta)|^2 \to \min_{\theta \in \mathbb{R}^n} . \tag{3}$$

How many terms we should take that optimal solution of (3) will be ε-solution (in function) of (2). If $f(\theta)$ is convex, than $m \sim n/\varepsilon^2$. If $f(\theta)$ is μ-strongly convex (in 2-norm) then $m \sim 1/(\mu\varepsilon)$.
SAA Approach. The consequence from that is the following: if we regularize (3) by $\sim \varepsilon\|\theta\|_2^2$, than the exact solution of regularized problem is $O(\varepsilon)$-solution of (2), when $m \sim 1/\varepsilon^2$. That is n-times better!!! So that is why one typically needs regularization.
SA Approach. In Stochastic Averaging approach to (2) we consider (2) to be stochastic convex optimization problem and can apply Stochastic Gradient Descend (SGD) for that. The idea of early stopping is to stop in convex (not strongly convex) case after $N \sim 1/\varepsilon^2$ iterations, rather than $N \sim n/\varepsilon^2$. This will have the same effect.
Note: In general situation we also have that Early stopping of an algorithm can be considered as an convenient (without unknown hyperparameter of regularization) alternative to regularization!

The first observation: Early stopping regularization



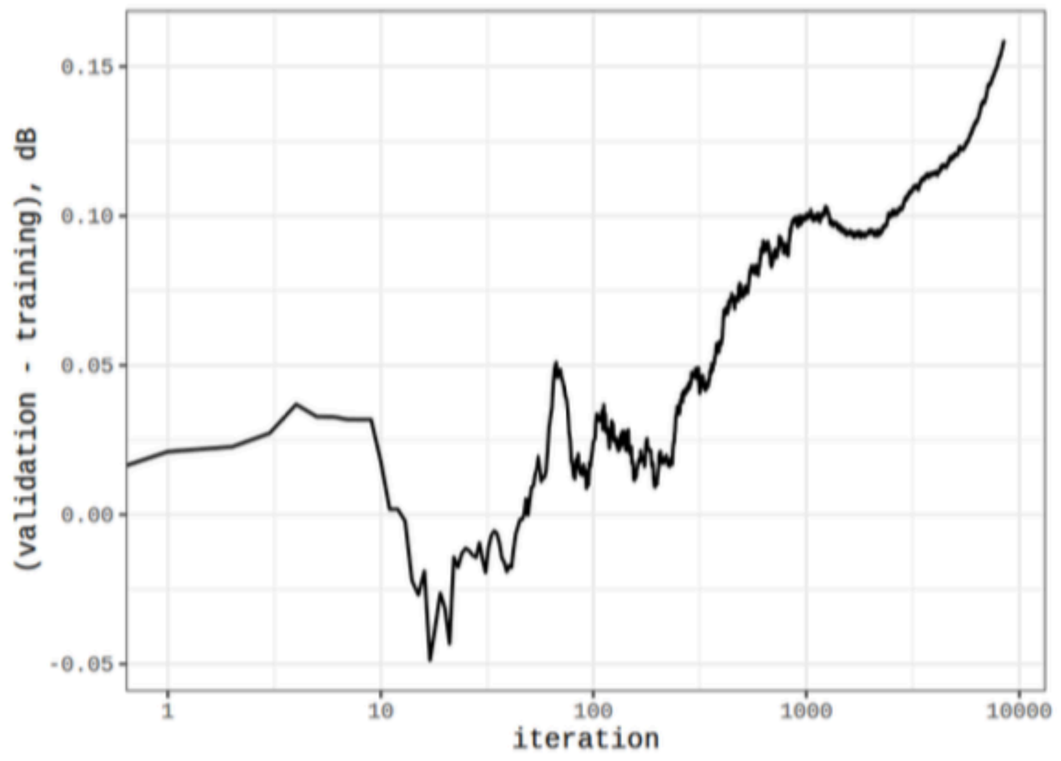


Figure: Residual model overfitting, 50% of original data used as training set

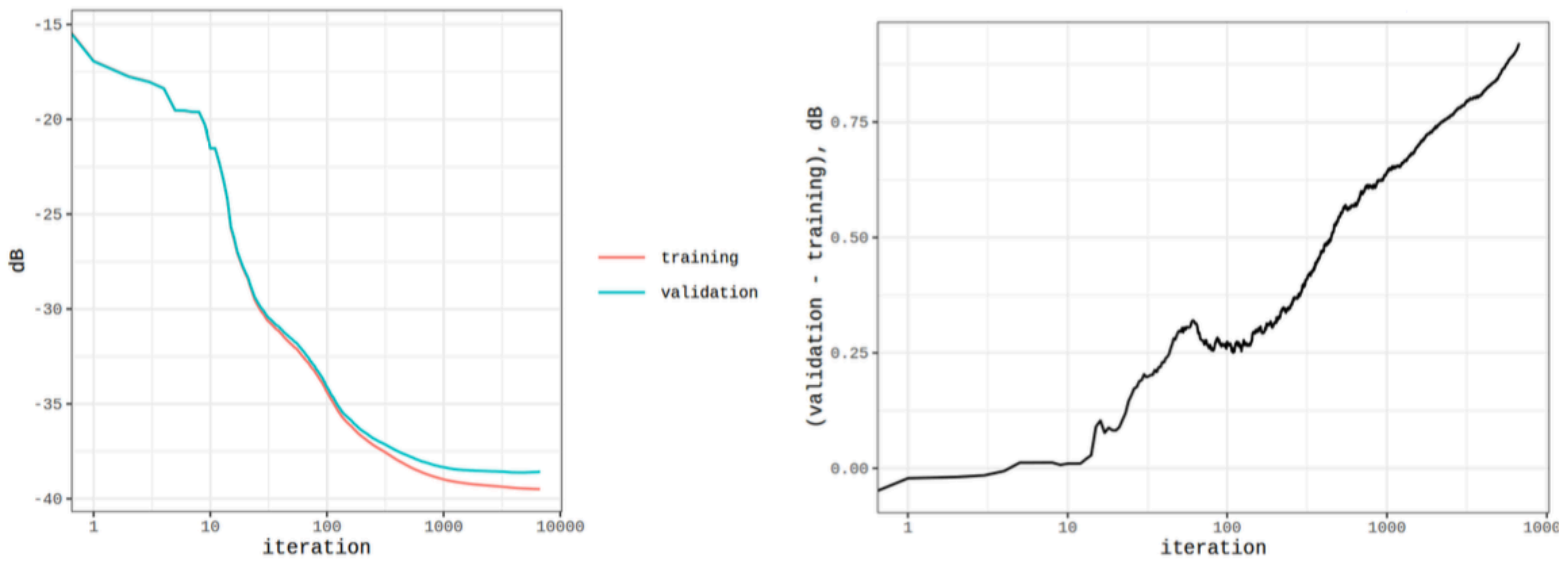


Figure: Residual model overfitting, 10% of original data used as training set

So, it seems, that we can reduce the number of terms in the sum ~ 10 times without significant loss in a quality of test error! This may valuable reduce the complexity of optimization procedure!

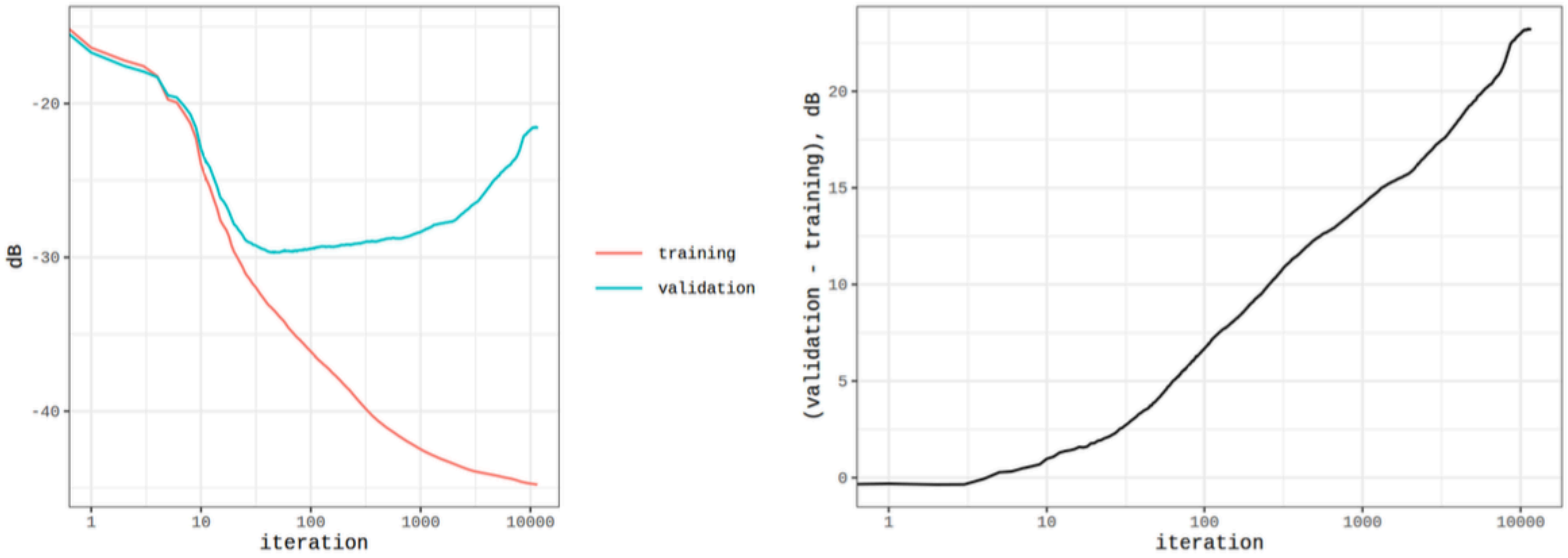


Figure: Residual model overfitting, 1% of original data used as training set

So, if we don't have large enough training set it is worth to use 'Early Stopping' regularization trick. That is to stop iteration procedure much earlier!

Non-convex stochastic optimization

- Consider the problem of searching a local minimum of the following function:

$$f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x),$$

where $f(x)$ is non-convex. For simplicity we assume that each f_i is L-smooth. The natural and most straightforward choice of the stochastic gradient for this problem is $g(x) = \nabla f_\xi(x)$ where ξ is a random variable with uniform distribution on $\{1, \dots, m\}$.

- Another situation that we are interested in is the case when $f(x)$ is L-smooth and can be written as an expectation of non-convex functions:

$$f(x) = \mathbf{E}_\xi[f(x, \xi)].$$

In this situation one can use $g(x) = \nabla f(x, \xi)$ as a stochastic gradient.

In both cases the theory that we discuss is based on the following assumption: the cost of one stochastic gradient computation is $O(1)$ and its scales linearly. For both cases we additionally assume that for some constant σ^2 the variance of $g(x)$ is uniformly upper bounded by σ^2:

$$\mathbf{E}\left[\|g(x) - \nabla f(x)\|_2^2 \mid x\right] \leq \sigma^2.$$

Optimal variance reduced scheme for non convex sum-type smooth problems

SPIDER

In the table below we summarize complexity bounds of finding such $\hat{x}$ that $\mathbf{E}[\|\nabla f(\hat{x})\|_2^2] \leq \varepsilon^2$.

Assumptions	Lower bound
Deterministic oracle, $f(x)$ is L-smooth	$\Omega\left(\frac{L\left(f(x^0)-f_*\right)}{\varepsilon^2}\right)$
$f(x)$ is L-smooth, $\mathbf{E}\left[\|g(x)-\nabla f(x)\|_2^2 \mid x\right] \leq \sigma^2$	$\Omega\left(L\left(f(x^0)-f_*\right)\max\left\{\frac{1}{\varepsilon^2}, \frac{\sigma^2}{\varepsilon^4}\right\}\right)$
$f_i(x)$ is L-smooth, $\mathbf{E}\left[\|g(x)-\nabla f(x)\|_2^2 \mid x\right] \leq \sigma^2$	$\Omega\left(\frac{L(f(x^0)-f(x^*))\min\left\{\sqrt{m}, \max\left\{1, \frac{\sigma}{\varepsilon}\right\}\right\}}{\varepsilon^2}\right)$

General scheme of the optimal first-order method for non-convex optimization

```
Input: learning rates $\{h_k\}_{k\geq 0}$ satisfying $h_k \leq \frac{1}{2L}$, starting point
    $x^0 \in \mathbb{R}^n$, stopping criterion $C$
 1: for $k = 0, 1, 2, \dots$ do
 2:     Get $g^k = g(x^k)$
 3:     if $C$ holds then
 4:         $x^N = x^k$
 5:         break
 6:     else
 7:         $x^{k+1} = x^k - h_k g^k$
 8:     end if
 9: end for
10: return $x^N$
```

Gradient Descent

$$g^k = \nabla f(x^k), \quad h_k = \frac{1}{2L}$$

is optimal in the deterministic case, i.e. in order to reach $\|\nabla f(x^N)\|_2 \leq \varepsilon$ it requires

$$O\left(\frac{L\left(f(x^0) - f_*\right)}{\varepsilon^2}\right) \text{ iterations.}$$

- Yair Carmon et al. "Lower bounds for finding stationary points i". In:Mathematical Programming (2019), pp. 1–50.

Stochastic Gradient Descent

$$g^k = \nabla f(x^k, \{\xi_i^k\}_{i=1}^r) \overset{\text{def}}{=} \frac{1}{r}\sum_{i=1}^r f(x^k, \xi_i),$$

$$r = \max\left\{1, \frac{10\sigma^2}{\varepsilon^2}\right\}, \quad h_k = \frac{1}{2L}$$ Explain further

is optimal in the second case, i.e. in order to reach $\mathbf{E}\left[\|\nabla f(\hat{x}^N)\|_2^2\right] \leq \varepsilon^2$ it requires

$$O\left(\left(L\left(f(x^0) - f_*\right)\max\left\{\frac{1}{\varepsilon^2}, \frac{\sigma^2}{\varepsilon^4}\right\}\right)\right)$$ oracle calls.

- Yossi Arjevani et al. "Lower bounds for non-convex stochastic optimization". In:arXiv preprint arXiv:1912.02365(2019).

SGD with Without Replacement sampling

```
Input: learning rates $\{h_{s,k}\}_{s,k\geq 0}$, starting point $x^0 \in \mathbb{R}^n$, batch size
  $r \geq 1$, number of epochs $S$
  Set $x_0^0 = x^0$
  for $s = 0, 1, 2, \ldots K-1$ do
    Generate random permutation $\{i_{s,1}, \ldots, i_{s,m}\}$ of the set $\{1, \ldots, m\}$
    Set $l = \lceil m/r \rceil$
    for $k = 0, 1, \ldots, l-1$ do
      Set $\hat{r}_s^k = \min\{r, m - kr\}$
      Compute $g_s^k = \frac{1}{r_s^k} \sum_{j=1}^{r_s^k} \nabla f_{i_{s,kr+j}}(x_s^k)$
      $x_s^{k+1} = x_s^k - h_{s,k} g_s^k$
    end for
    $x_{s+1}^0 = x_s^l$
  end for
  return $x_{K-1}^l$
```

For non-convex objectives there is no theory for this method, but in the strongly convex case it was shown recently that it outperforms standard SGD with i.i.d. sampling (see Rajput et al. (2020)): it works with $O\left(\frac{m}{N^2}\right)$ rate instead of $O\left(\frac{1}{N}\right)$.

SPIDER

SAGA type method

SPIDER:

$$
\begin{aligned}
r_k = r &= \max\left\{1, \frac{\sigma^2}{10\varepsilon^2}\right\}, \\
q &= \min\left\{r, m\right\}, \\
g^k &= \begin{cases} \frac{1}{r}\sum\limits_{j=1}^{r} \nabla f_{\xi_{k,j}}(x^k), \text{ if } r < m \text{ and } r \text{ divides } k, \\ \nabla f(x^k), \text{ if } m \leq r \text{ and } m \text{ divides } k, \\ \nabla f_{\xi_k}(x^k) - \nabla f_{\xi_k}(x^{k-1}) + g^{k-1}, \text{ otherwise} \end{cases} \\
h_k = h &= \frac{1}{5L\sqrt{q}}.
\end{aligned}
$$

- Cong Fang et al. “Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator”. In:Advances in Neural Information Processing Systems. 2018, pp. 689–699.

SPIDER is optimal in the third case, i.e. in order to reach $\mathbf{E}\left[\|\nabla f(\hat{x}^N)\|_2^2\right] \leq \varepsilon^2$ it requires

$$O\left(\frac{L(f(x^0) - f(x^*))\min\left\{\sqrt{m}, \max\left\{1, \frac{\sigma}{\varepsilon}\right\}\right\}}{\varepsilon^2}\right) \text{ oracle calls.}$$

SpiderBoost

- SPIDER requires very small stepsizes in theory.
- In the recent works Wang et al. (2018, 2019) authors proposed simpler version of SPIDER called SpiderBoost that works with constant stepsizes.

```
Input: learning rate $h > 0$, epoch length $T$, starting point $x^0 \in \mathbb{R}^n$,
    batch size $r \geq 1$, number of iterations $K$
 1: for $k = 0, 1, 2, \dots$ do
 2:     if $k \mod T = 0$ then
 3:         Compute $g^k = \nabla f(x^k)$
 4:     else
 5:         Uniformly randomly pick set $I_k$ from $\{1, \dots, m\}$ (with
            replacement) such that $|I_k| = r$
 6:         Compute $g^k = \frac{1}{r} \sum_{i \in I_k} \left(\nabla f_i(x^k) - \nabla f_i(x^{k-1})\right) + g^{k-1}$
 7:     end if
 8:     $x^{k+1} = x^k - hg^k$
 9: end for
10: Pick $\xi$ uniformly at random from $\{0, \dots, K-1\}$
11: return $x^\xi$
```

SpiderBoostM

```
Input: learning rate $h > 0$, epoch length $T$, starting point $x^0 \in \mathbb{R}^n$,
    batch size $r \geq 1$, number of iterations $K$, parameters
    $\{\lambda_k\}_{k=1}^{K-1}, \{\beta_k\}_{k=1}^{K-1} > 0$
 1: Set $y^0 = x^0$
 2: for $k = 0, 1, 2, \dots$ do
 3:     $\alpha_{k+1} = \frac{2}{\lceil (k+1)/T \rceil + 1}$
 4:     $z^k = (1 - \alpha_{k+1}) y^k + \alpha_{k+1} x^k$
 5:     if $k \mod T = 0$ then
 6:         Compute $g^k = \nabla f(z^k)$
 7:     else
 8:         Uniformly randomly pick set $I_k$ from $\{1, \dots, n\}$ (with
            replacement) such that $|I_k| = r$
 9:         Compute $g^k = \frac{1}{r} \sum_{i \in I_k} (\nabla f_i(x^k) - \nabla f_i(x^{k-1})) + g^{k-1}$
10:     end if
11:     $x^{k+1} = x^k - \lambda_k g^k$
12:     $y^{k+1} = z^k - \frac{\beta_k}{\lambda_k} x^k + \frac{\beta_k}{\lambda_k}(x^k - \lambda_k g^k)$
13: end for
14: Pick $\xi$ uniformly at random from $\{0, \dots, K-1\}$
15: return $z^\xi$
```

For both methods it is proven that they have complexities

$$O\left(m^{1/2}L(f(x^0) - f^*)\varepsilon^{-2}\right).$$

But is convex case SpiderBoostM behaves like Accelerated method!

Not optimal, but classical VR-method: SVRG

```
Input: learning rate $h > 0$, epoch length $T$, starting point $x^0 \in \mathbb{R}^n$,
    batch size $r \geq 1$
1: $\phi_0 = x_0^0 = x^0$
2: for $s = 0, 1, 2, \dots$ do
3:     for $k = 0, 1, 2, \dots, T-1$ do
4:         Uniformly randomly pick set $I_k$ from $\{1, \dots, n\}$ (with
           replacement) such that $|I_k| = r$
5:         $g^k = \frac{1}{r} \sum_{i \in I_k} \left(\nabla f_i(x_s^k) - \nabla f_i(\phi_s)\right) + \nabla f(\phi_s)$
6:         $x_s^{k+1} = x_s^k - hg^k$
7:     end for
8:     $\phi_{s+1} = x_{s+1}^0 = x_s^k$
9: end for
```

Reddi et al. showed that SVRG with $r = m^{2/3}$ and $T = m^{1/3}$ has the following complexity:

$$O\left(m^{2/3} L(f(x^0) - f^*)\varepsilon^{-2}\right)$$

We've just talked about finding stationary points, but one can show that the developed methods can be modified in such a manner that we will have the same in terms of ϵ, n (up to a logarithmic factors) complexity (SAGA type method, SPIDER):

https://arxiv.org/pdf/1806.08782.pdf

https://arxiv.org/pdf/1807.01695.pdf

Unfortunately, variance reduced schemes in practice works typically worthier then tuned SGD and its (momentum) Adaptive variants!

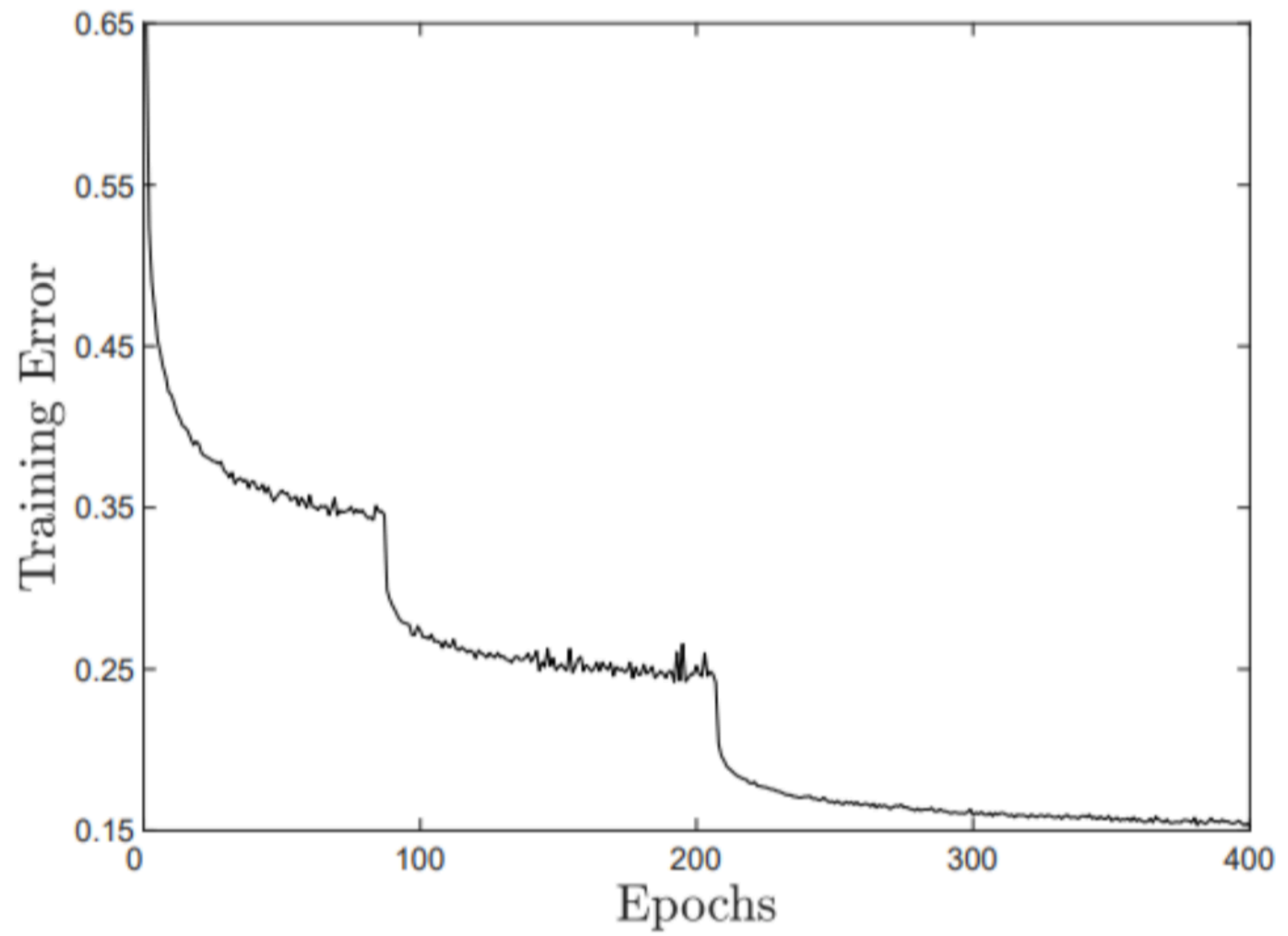


Figure 1: Training error using SGD with mini-batch size 32 to train an 8-layer convolutional neural network on CIFAR-10 [Kri09]. The first 90 epochs use a learning rate of $s = 0.006$, the next 120 epochs use $s = 0.003$, and the final 190 epochs use $s = 0.0005$. Note that the training error decreases as the learning rate s decreases and a smaller s leads to a larger number of epochs for SGD to reach a plateau. See [HZRS16] for further investigation of this phenomenon.

Figure: `https://arxiv.org/pdf/2004.06977.pdf`

Idea of Adaptation

Rough Idea: Sometimes in different algorithms with auxiliary 1D line search (or several such searches) it is better to use rough solution of 1D auxiliary optimization problem. It may significantly accelerate algorithm.
Simple example. In the worse case

$$x^{k+1} = x^k - h_k \nabla f(x^k) = x^k - \tfrac{1}{L} \nabla f(x^k),$$
$$h_k = \mathrm{argmin}_{h \geq 0} f(x^k - h \nabla f(x^k)),$$

where $\|\nabla f(y) - \nabla f(x)\|_2 \leq L \|y - x\|_2$. Hence

$$\frac{1}{2L} \|\nabla f(x^k)\|_2^2 \leq f(x^k) - f(x^{k+1}) \leq f(x^k) - f(x_*),$$

$$L \simeq \tfrac{\|\nabla f(x^k)\|_2^2}{f(x^k) - f(x_*)}, \quad \text{(Polyak–Shor)}$$

$$L \simeq \tfrac{\langle x^k - x^{k-1}, \nabla f(x^k) - \nabla f(x^{k-1}) \rangle}{\|\nabla f(x^k) - \nabla f(x^{k-1})\|_2^2}. \quad \text{(Barzilai-Borwein)}$$

Adaptive Gradient Descent (Malitsky & Mishchenko (2019))

```
1: $x^0 \in \mathbb{R}^d$, $\lambda_0 > 0$, $\theta_0 = +\infty$, $x^1 = x^0 - \lambda_0 \nabla f(x^0)$
2: for $k = 1, 2, \ldots$ do
3:     $\lambda_k = \min\left\{\sqrt{1+\theta_{k-1}}\lambda_{k-1}, \frac{\|x^k - x^{k-1}\|}{2\|\nabla f(x^k) - \nabla f(x^{k-1})\|}\right\}$
4:     $x^{k+1} = x^k - \lambda_k \nabla f(x^k)$
5:     $\theta_k = \frac{\lambda_k}{\lambda_{k-1}}$
6: end for
```

Authors prove $O(\varepsilon^{-1})$ rate in the convex case and $O(\kappa \ln \varepsilon^{-1})$ in the strongly convex case where κ is the condition number of the problem. No convergence guarantees in the non-convex case.

Fast gradient method with 1D-search (Yu. Netserov et al., 2018)

$$x^{k+1} = \tau_{k+1} z^k + \left(1 - \tau_{k+1}\right) y^k,\ \tau_{k+1} \in \operatorname{Arg}\min_{\tau \in [0,1]} f\left(\tau z^k + (1-\tau) y^k\right),$$

$$y^{k+1} = x^{k+1} - h_{k+1} \nabla f\left(x^{k+1}\right),\ h_{k+1} \in \operatorname{Arg}\min_{h \geq 0} f\left(x^{k+1} - h \nabla f\left(x^{k+1}\right)\right),$$

$$z^{k+1} = z^k - \alpha_{k+1} \nabla f\left(x^{k+1}\right),$$

$$\alpha_{k+1} = \frac{1}{2L_{k+1}} + \sqrt{\frac{1}{4L_{k+1}^2} + \alpha_k^2},\ \alpha_0 = 0,$$

$$L_{k+1} = \frac{\left\|\nabla f\left(x^{k+1}\right)\right\|_2^2}{2\left(f\left(x^{k+1}\right) - f\left(y^{k+1}\right)\right)} \leq L,\ f\left(y^{k+1}\right) \leq f\left(x^{k+1}\right) - \frac{1}{2L}\left\|\nabla f\left(x^{k+1}\right)\right\|_2^2.$$

Convergence result of Nesterov's method

For convex function $f(x)$ (there exists also strongly convex variant)

$$f\left(y^N\right) - f\left(x_*\right) \leq \frac{2LR^2}{N^2}.$$

For non convex function $f(x)$ (see also Ghadimi–Lan, 2013; Catalyst, 2018)

$$\min_{k=1,\dots,N} \left\|\nabla f\left(y^k\right)\right\|_2^2 \leq \frac{2L\Delta f}{N}.$$

In both cases these bounds unimprovable up to a constant factors. Note that close approaches proposed A. Nemirovski in the period 1978–1984.

In practice the behavior of this method is close to the family of conjugate gradients methods.

Accelerated alternating minimizations method (S. Guminov et al., 2019)

This method looks the same as Nesterov's FGM with 1D-search, but

$$y^{k+1} = x^{k+1} - h_{k+1}\nabla f\left(x^{k+1}\right),\ z^{k+1} = z^k - \alpha_{k+1}\nabla f\left(x^{k+1}\right),$$

are replaced by

$$y^{k+1} \in \operatorname{Arg}\min_{S_{i(k+1)}\left(x^{k+1}\right)} f\left(x\right),\ z^{k+1} = z^k - \alpha_{k+1}\nabla_{i(k+1)} f\left(x^{k+1}\right),$$

where

$$S_i\left(x^{k+1}\right) = x^{k+1} + \operatorname{span}\left\{e_j,\ j \in I_i\right\},\ i(k+1) \in \operatorname{Arg}\max_{i=1,\dots,m} \left\|\nabla_i f\left(x^{k+1}\right)\right\|_2^2 .$$

The rates of convergence remain the same up to a

$$L \to mL,\ m \text{ is a number of blocks } (i = 1,\dots,m).$$

AdaGrad-Norm (R. Ward et al., 2019)

The problem of parameters selection can be partially solved by using adaptive version of SGD (this line starts in stochastic convex optimization from AdaGrad, 2011). For example (b_0 – large enough),

$$b_{k+1}^2 = b_k^2 + \left\| \overset{r}{\nabla}_x f\left(x^k, \left\{\xi^{k,l}\right\}_{l=1}^r\right) \right\|_2^2, \; x^{k+1} = x^k - \frac{\eta}{b_{k+1}} \overset{r}{\nabla}_x f\left(x^k, \left\{\xi^{k,l}\right\}_{l=1}^r\right),$$

converges as (skip expectation)

$$\min_{k=1,\dots,N} \left\| \nabla f\left(x^k\right) \right\|_2^2 = \tilde{O}\left(\left(\frac{\sigma}{\sqrt{r}} + \frac{\Delta f}{\eta} + L\eta \right) \left(\frac{b_0}{N} + \frac{\sigma}{\sqrt{rN}} \right) \right).$$

This result corresponds to what we've obtained above by proper SGD!

Most popular in practice nowadays are accelerated versions of SGD (see Deep Learning book and http://ruder.io/optimizing-gradient-descent/): Adam, AdaMax, Nadam, AMSGrad, ... Unfortunately, there are lacks of theoretical results for these methods. This direction is actively developed now in the works of: K. Levy, V. Chevher, F. Bach, G. Lan et al.

Adam

$$m_0 = v_0 = 0$$

g^k - batched stochastic gradient

$$\begin{aligned}
\mathbf{m}_i^k &= \beta_1 \mathbf{m}_i^{k-1} + (1-\beta_1)\mathbf{g}_i^k, \quad \hat{\mathbf{m}}_i^k = \frac{\mathbf{m}_i^k}{1-\beta_1^k} \\
\mathbf{v}_i^k &= \beta_2 \mathbf{v}_i^{k-1} + (1-\beta_2)(\mathbf{g}_i^k)^2, \quad \hat{\mathbf{v}}_i^k = \frac{\mathbf{v}_i^k}{1-\beta_2^k} \\
\mathbf{x}_i^{k+1} &= \mathbf{x}_i^k - \frac{\gamma}{\sqrt{\hat{\mathbf{v}}_i^k + \delta}}\hat{\mathbf{m}}_i^k, \quad i = \overline{1,n}, \quad \delta = 10^{-8}.
\end{aligned}$$

$$\beta = 0.9,\ \beta_2 = 0.99 \text{ и } \gamma = 0.001.$$

Sashank J. Reddi, Satyen Kale и Sanjiv Kumar, ON THE CONVERGENCE OF ADAM AND BEYOND, ICLR 2018.

Alexandre Defossez, Leon Bottou, Francis Bach, Nicolas Usunier, On the Convergence of Adam and Adagrad, arXiv preprint, arXiv:2003.02395

AMSGrad

$$
\begin{aligned}
\mathbf{m}_i^k &= \beta_1 \mathbf{m}_i^{k-1} + (1-\beta_1)\mathbf{g}_i^k, \\
\mathbf{v}_i^k &= \beta_2 \mathbf{v}_i^{k-1} + (1-\beta_2)(\mathbf{g}_i^k)^2, \quad \hat{\mathbf{v}}_i^k = \max\{\hat{\mathbf{v}}_i^{k-1}, \mathbf{v}_i^k\} \\
\mathbf{x}_i^{k+1} &= \mathbf{x}_i^k - \frac{\gamma_k}{\sqrt{\hat{\mathbf{v}}_i^k} + \delta}\mathbf{m}_i^k, \quad i = \overline{1, n}, \quad \delta = 10^{-8}, \\
\gamma_k &= \gamma(1-\beta_1)\sqrt{1-\beta_2^k}.
\end{aligned}
$$

Second-order schemes

Hendrikx, H., Xiao, L., Bubeck, S., Bach, F., Massoulie, L. (2020). Statistically Preconditioned Accelerated Gradient Method for Distributed Optimization. arXiv preprint `arXiv:2002.10726`.
Kamzolov, D., Gasnikov, A. (2020). Near-Optimal Hyperfast Second-Order Method for convex optimization and its Sliding. arXiv preprint arXiv:2002.09050.
Lucchi, A., Kohler, J. (2019). A Stochastic Tensor Method for Non-convex Optimization. arXiv preprint arXiv:1911.10367.

Basic idea:

$$f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x) \to \min_{x\in\mathbb{R}^n}.$$

In different second order schemes we have to minimize at each iteration something like

$$f(x^k)+\langle\tilde{\nabla} f(x^k), x-x^k\rangle+\frac{1}{2!}\langle\tilde{\nabla}^2 f(x^k)(x-x^k), x-x^k\rangle+\frac{M}{3!}\|x-x^k\|_2^3.$$

$$f(x^k)+\langle\tilde{\nabla} f(x^k), x-x^k\rangle+\frac{1}{2!}\langle\tilde{\nabla}^2 f(x^k)(x-x^k), x-x^k\rangle+\frac{M}{3!}\|x-x^k\|_2^3.$$

In full gradient/Hessian version:

$$\tilde{\nabla} f(x^k)=\nabla f(x^k), \tilde{\nabla}^2 f(x^k)=\nabla^2 f(x^k).$$

In standard randomized version:

$$\tilde{\nabla} f(x^k)=\frac{1}{r}\sum_{l=1}^{r}\nabla f_{\xi(l)}(x^k), \tilde{\nabla}^2 f(x^k)=\frac{1}{r}\sum_{l=1}^{r}\nabla^2 f_{\xi(l)}(x^k).$$

But the sensitivity of different algorithms to $\tilde{\nabla} f(x^k)$ higher than to $\tilde{\nabla}^2 f(x^k)$. This is not obvious. Newton method break our intuition here. But Newton method is not the method we'd like to apply... So we may use this fact as follows:

$$\tilde{\nabla} f(x^k)=\nabla f(x^k), \tilde{\nabla}^2 f(x^k)=\frac{1}{r}\sum_{l=1}^{r}\nabla^2 f_{\xi(l)}(x^k).$$

How to choose r?

From the general theory we may expect that $r \sim 1/\varepsilon^2$, where ε is the desired accuracy. So $r_{grad} \sim 1/\varepsilon^2$ indeed, that is typically not smaller then m (number of terms in the sum)! But $r_{Hess} \sim 1/\varepsilon$ `arXiv:1810.03763`.
Another argument how to choose r_{Hess} is complexity argument. We need $O(rn^2)$ arithmetic operations (a.o.) to calculate Hessian. At the same time we need $O(n^3)$ a.o. to 'invert' Hessian (solve auxiliary problem). From this we have

$$r \simeq n.$$

Note: There appears very practical version of Strassen's algorithm that improve $O(n^3) \to O(n^{\log_2 7})$.
Huang, J., Smith, T. M., Henry, G. M., van de Geijn, R. A. (2016, November). Strassen's algorithm reloaded. In SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 690-701). IEEE.

For the moment we consider adaptive method for smoothness parameter L. But what is about parameter r?

To answer for this question let's consider the simple SGD in convex case (accelerated and non-convex ones we briefly describe at the end)

Parallelization and minibatch'ing

Assume that

$$E_\xi\left[\nabla_x f(x,\xi)\right] \equiv \nabla f(x),$$

$$E_\xi\left[\left\|\nabla_x f(x,\xi) - \nabla f(x)\right\|_2^2\right] \le D.$$

Let's introduce (here we can use full parallelization!)

$$\overset{r}{\nabla}_x f\left(x,\left\{\xi^l\right\}_{l=1}^r\right) = \frac{1}{r}\sum_{l=1}^r \nabla_x f\left(x,\xi^l\right),$$

where $\left\{\xi^l\right\}_{l=1}^r$ – i.i.d. (with the same distribution as ξ). Note that

$$E_{\left\{\xi^l\right\}_{l=1}^r}\left[\left\|\overset{r}{\nabla}_x f\left(x,\left\{\xi^l\right\}_{l=1}^r\right) - \nabla f(x)\right\|_2^2\right] \le \frac{D}{r},$$

Hence for arbitrary $L > 0$ and all $x^k, \mathrm{v} = x^{k+1} - x^k$

$$\left\langle \nabla f\left(x^k\right) - \overset{r}{\nabla}_x f\left(x^k, \left\{\xi^l\right\}_{l=1}^r\right), \mathrm{v} \right\rangle \le \underbrace{\frac{1}{2L}\left\| \overset{r}{\nabla}_x f\left(x^k, \left\{\xi^l\right\}_{l=1}^r\right) - \nabla f\left(x^k\right) \right\|_2^2}_{\delta^{k+1}} + \frac{L}{2}\|\mathrm{v}\|_2^2.$$

If for all x, y

$$\left\|\nabla f\left(y\right) - \nabla f\left(x\right)\right\|_2 \le L\|y - x\|_2,$$

then for all x^k, x^{k+1}

$$f\left(x^{k+1}\right) \le f\left(x^k\right) + \left\langle \nabla f\left(x^k\right), x^{k+1} - x^k \right\rangle + \frac{L}{2}\left\|x^{k+1} - x^k\right\|_2^2.$$

Hence

$$f\left(x^{k+1}\right) \le f\left(x^k\right) + \left\langle \overset{r}{\nabla}_x f\left(x^k, \left\{\xi^{k+1,l}\right\}_{l=1}^r\right), x^{k+1} - x^k \right\rangle + \frac{2L}{2}\left\|x^{k+1} - x^k\right\|_2^2 + \delta^{k+1}. \ (*)$$

Important fact follows from (*): if $E\left[\delta^{k+1}\right] = \varepsilon/2$ then for batched gradient descent

$$\boxed{x^{k+1} = x^k - \frac{1}{2L}\overset{r}{\nabla}_x f\left(x^k, \left\{\xi^{k+1,l}\right\}_{l=1}^r\right)}$$

converges according to the following estimate

$$E\left[f\left(\bar{x}^N\right)\right] - f_* \le \frac{LR^2}{N} + \frac{\varepsilon}{2}.$$

So it is natural to choose r such that

$$\frac{\varepsilon}{2} = E\left[\delta^{k+1}\right] = E\left[\frac{1}{2L}\left\|\overset{r}{\nabla}_x f\left(x^k, \left\{\xi^l\right\}_{l=1}^r\right) - \nabla f\left(x^k\right)\right\|_2^2\right] = \frac{D}{2Lr},$$

$$\boxed{r = \frac{D}{L\varepsilon}}.$$

All these things can be done for accelerated gradient method!

Drawbacks and the main idea

Even in non accelerated (most simple) variant the method need to know L:

$$x^{k+1} = x^k - \frac{1}{2L}\overset{r}{\nabla}_x f\left(x^k, \left\{\xi^{k+1,l}\right\}_{l=1}^{r}\right)$$

and D:

$$r = \frac{D}{L\varepsilon}.$$

Assume that we know D_0 – the lower bound on D ($D \geq D_0$). The main idea is to choose L^{k+1} from: $L^{k+1} := L^k/2$; $L^{k+1} := 2L^k$, where $r^{k+1} = \left(D_0 / L^{k+1}\varepsilon\right)$, until

$$\boxed{f\left(x^{k+1}\right) \leq f\left(x^k\right) + \left\langle \overset{r^{k+1}}{\nabla}_x f\left(x^k, \left\{\xi^{k+1,l}\right\}_{l=1}^{r^{k+1}}\right), x^{k+1} - x^k \right\rangle + \frac{2L^{k+1}}{2}\left\|x^{k+1} - x^k\right\|_2^2 + \frac{\varepsilon}{2},}(**)$$

holds true! One can show that the number of iterations will be $\mathrm{O}\left(DLR^2/(D_0\varepsilon)\right)$ and oracle calls – $\mathrm{O}\left(DR^2/\varepsilon^2\right)$.

Note 1: If we want to estimate the rate of convergence we should additionally assume: $L^{k+1} \geq L^k$ (V. Spokoiny et al., 2019)

Note 2: We could take $r^{k+1} = 2D_0/(L^k\epsilon)$ // better in practice

Comments

So we've just described simple approach that allows to make fully adaptive stochastic gradient descent from classical gradient descent with

$$\mathrm{O}\left(\frac{LR^2}{\varepsilon}\frac{D}{D_0}\right) \text{ iterations and } \mathrm{O}\left(\frac{DR^2}{\varepsilon^2}\right) \text{ stochastic gradient calculations.}$$

The last bound is unimprovable. Note, that we can fully parallelize calculations at each iteration (problem 3)! Moreover, by using accelerated gradient descent one can improve the number of iterations (problem 2)

$$\mathrm{O}\left(\sqrt{\frac{LR^2}{\varepsilon}\frac{D}{D_0}}\right)$$

This bound is unimprovable up to a factor $\sqrt{D/D_0}$ – payment for the method to be fully adaptive (problem 1).

The main drawback: We have to know the exact values of goal function $f(x)$ at (**). Typically in this case we have real (non stochastic) gradient.

$$f(x) = \mathbb{E} f(x, \xi). \tag{23}$$

In this case we estimate the function as a sample average

$$f(x, \{\xi_l\}_{l=1}^r) = \frac{1}{r} \sum_{l=1}^{r} f(x, \xi_l)$$

and use it in adaptive procedures. In this case we interpret L_k as the worst constant among all Lipschitz constants for $f(x, \xi)$ with different realization of ξ. Indeed, if L_{k+1} satisfies the following

$$\begin{gathered} f(x^{k+1}, \xi^{k+1}) \leq f(x^k, \xi^{k+1}) + \langle \nabla f(x^k, \xi), x^{k+1} - x^k \rangle \\ + L_{k+1} \|x^{k+1} - x^k\|_2^2 + \varepsilon/2. \end{gathered}$$

Then it satisfies

$$\begin{aligned} f(x^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}) \leq{}& f(x^k, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}) \\ &+ \langle \nabla^{r_{k+1}} f(x, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - x^k \rangle \\ &+ L_{k+1} \|x^{k+1} - x^k\|_2^2 + \varepsilon/2. \end{aligned} \tag{24}$$

If, e.g, (23) holds we replace adaptive procedure in the algorithms by (24).

Note that if we don't have an access to stochastic gradient, but have an access to realization (unbiased) $f(x,\xi)$ (see Data Science applications below), then in batched gradient descent and in (24) we may use (this formula works also in non smooth case!)

$$\nabla f(x, \xi := (\xi, e)) :\simeq n\frac{f(x+\tau e, \xi) - f(x-\tau e, \xi)}{2\tau}e$$

where e choses at random (ourselves) from the unit sphere or from coordinate orts. And we assume that $x^{k+1} - x^k$ parallel to e^k (sample of e at k-th step)

arXiv:1707.08486

The rate of convergence will be $\sim n$ times slower.

Another way to choose $r^{k+1} = 2D^{k+1}/(L^k \epsilon)$ adaptively is to estimate in an adaptive manner D^{k+1}:

$$D^{k+1} \simeq \frac{1}{r^{k+1}-1} \sum_{l=1}^{r^{k+1}} \|\nabla f(x^k, \xi^{k+1,l}) - \bar{\nabla} f(x^k)\|_2^2$$

$$\bar{\nabla} f(x^k) = \frac{1}{r^{k+1}} \sum_{l=1}^{r^{k+1}} \nabla f(x^k, \xi^{k+1,l})$$

Since we don't know it advance r^{k+1} we put here $r^{k+1} = r^k/2$

Close approach: https://arxiv.org/pdf/2005.01097.pdf

These all hold true for gradient-free schemes (see above)

REMIND THAT:

Data Science applications

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x})$$

In this case in (24): $L = \max_{i=1,..,n} L_i$

$$\nabla f(x,\xi) = \nabla f_{\xi}(x)$$

Supplementary materials

Adaptive SGD

Dvurechensky Pavel (HSE, IITP, WIAS dvureche@wias-berlin.de)
Gasnikov Alexander (MIPT gasnikov@yandex.ru)
Spokoiny Vladimir (spokoiny@wias-berlin.de)

Let us consider the following minimization problem

$$\min_{y \in Q} f(y), \tag{1}$$

where $f(y)$ is convex function and its gradient is Lipschitz continuous w.r.t. $\| \cdot \|_2$ with the constant L_f:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L_f \|x - y\|_2.$$

We assume that the access to the objective f is given through stochastic oracle $\nabla f(x, \xi)$, where ξ is a random variable. The main assumptions on the stochastic oracle are standard for stochastic approximation literature Nemirovski et al. (2009)

$$\mathbb{E} \nabla f(x, \xi) = \nabla f(x), \quad \mathbb{E}\left(\|\nabla f(x, \xi) - \nabla f(x)\|_2^2\right) \leq D. \tag{2}$$

We start with stochastic gradient descent with general stepsize h

$$x^{k+1} = x^k - h\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r). \tag{3}$$

where $\nabla^r f(x, \{\xi_l\}_{l=1}^r)$ is a stochastic approximation for the gradient $\nabla f(x)$ with mini-batch of size r

$$\nabla^r f(x, \{\xi_l\}_{l=1}^r) = \frac{1}{r}\sum_{l=1}^r \nabla f(x, \xi_l),$$

where each stochastic gradient $\nabla f(x, \xi_l)$ satisfies (2).

Lemma 1. With step size $h = \frac{1}{2L}$ in stochastic gradient descent (3) the following holds

$$f(x^{k+1}) - f(x^k) \leq -\frac{1}{4L}\|\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r)\|_2^2 + \frac{1}{2L}\delta_{k+1}^2,$$

where $\delta_{k+1}^2 = \|\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r) - \nabla f(x^k)\|_2^2$.

Algorithm 1 Stochastic Gradient Descent

```
Require: Number of iterations $N$, variance $D$, Lipschitz constant
    $L$, accuracy $\varepsilon$.
 1: Calculate batch size
        $$r = \max\{D/(L\varepsilon),\ 1\}$$
 2: for $k = 0, \dots, N-1$ do
 3:     $$x^{k+1} = x^k - \frac{1}{2L}\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r)$$
 4: end for
Ensure: $\bar{x}^N = \frac{1}{N}\sum_{k=1}^N x^k$.
```

Theorem 1. Algorithm 1 with stochastic gradient oracle calls $T = O\left(\frac{DR^2}{\varepsilon^2}\right)$, batch size $r = \max\{D/(L\varepsilon), 1\}$, number of iterations $N = O\left(\frac{LR^2}{\varepsilon}\right)$ outputs a point $\bar{x}^N$ satisfying

$$\mathbb{E}f(\bar{x}^N) - f(x^*) \leq \varepsilon. \tag{6}$$

Acceleration (see Algorithm 3 below)

Algorithm 1 Stochastic Gradient Descent

```
Require: Number of iterations $N$, variance $D$, Lipschitz constant $L$, accuracy $\varepsilon$.
 1: Calculate batch size
        $$r = \max\{D/(L\varepsilon),\ 1\}$$
 2: for $k = 0, \dots, N-1$ do
 3:
        $$x^{k+1} = x^k - \frac{1}{2L}\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r)$$
 4: end for
Ensure: $\bar{x}^N = \frac{1}{N}\sum_{k=1}^N x^k$.
```

Theorem 1. Algorithm 1 with stochastic gradient oracle calls $T = O\left(\frac{DR^2}{\varepsilon^2}\right)$, batch size $r = \max\{D/(L\varepsilon), 1\}$, number of iterations $O\left(\sqrt{LR^2/\varepsilon}\right)$ outputs a point $\bar{x}^N$ satisfying

$$\mathbb{E}f(\bar{x}^N) - f(x^*) \leq \varepsilon. \tag{6}$$

Algorithm 2 Adaptive Stochastic Gradient Descent

```
Require: Number of iterations $N$, accuracy $\varepsilon$, $D_0$, initial guess $L_0$.
1: for $k = 0, \dots, N-1$ do
2:     $L_{k+1} := L_k/4$
3:     repeat
4:         $L_{k+1} := 2L_{k+1}$
5:         $r_{k+1} = \max\{D_0/(L_{k+1}\varepsilon),\ 1\}$
6:         $x^{k+1} = x^k - \frac{1}{2L_k}\nabla^{r_k} f(x^k, \{\xi_l^{k+1}\}_{l=1}^{r_k})$
7:     until
           $f(x^{k+1}) \le f(x^k) + \langle \nabla^{r_{k+1}} f(x, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - x^k \rangle + L_{k+1}\|x^{k+1} - x^k\|_2^2 + \varepsilon/2$   (11)
8: end for
Ensure: $\bar{x}_N = \frac{1}{N}\sum_{k=1}^{N} x^k$.
```

$$D_0 \ \geq \ D$$

```
Algorithm 2 Adaptive Stochastic Gradient Descent
Require: Number of iterations $N$, accuracy $\varepsilon$, $D_0$, initial guess $L_0$.
1: for $k = 0, \dots, N-1$ do
2:     $L_{k+1} := L_k/2$
3:     repeat
4:         $L_{k+1} := 2L_{k+1}$
5:             $r_{k+1} = \max\{D_0/(L_{k+1}\varepsilon),\ 1\}$
6:             $x^{k+1} = x^k - \frac{1}{2L_k}\nabla^{r_k} f(x^k, \{\xi_l^{k+1}\}_{l=1}^{r_k})$
7:     until
           $f(x^{k+1}) \le f(x^k) + \langle \nabla^{r_{k+1}} f(x, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - x^k\rangle + L_{k+1}\|x^{k+1} - x^k\|_2^2 + \varepsilon/2 \quad (11)$
8: end for
Ensure: $\bar{x}_N = \frac{1}{N}\sum_{k=1}^{N} x^k$.
```

$$D_0 \ \ge\ D$$

Algorithm 3 Adaptive Stochastic Accelerated Gradient Method

```
Require: Number of iterations $N$, $D_0$ accuracy $\varepsilon$, $\Omega \geq 1$, $A_0 = 0$,
    initial guess $L_0$.
 1: for $k = 0, \dots, N-1$ do
 2:     $L_{k+1} := L_k/4$.
 3:     repeat
 4:         $L_{k+1} := 2L_{k+1}$
 5:         $\alpha_{k+1} = (1 + \sqrt{1 + 8A_k L_{k+1}})/(4L_{k+1})\ ;\ A_{k+1} = A_k + \alpha_{k+1}$
 6:         $r_{k+1} = \max\{\Omega \alpha_{k+1} D_0/\varepsilon,\ 1\}$
 7:         $y^{k+1} = (\alpha_{k+1} u^k + A_k x^k)/A_{k+1}$
 8:         $u^{k+1} = u^k - \alpha_{k+1} \nabla^{r_{k+1}} f(y^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}})$
 9:         $x^{k+1} = (\alpha_{k+1} u^{k+1} + A_k x^k)/A_{k+1}$
10:     until
            $f(x^{k+1}) \leq f(y^{k+1}) +$
                $\langle \nabla^{r_{k+1}} f(y^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - y^{k+1} \rangle +$
                $L_{k+1} \|x^{k+1} - y^{k+1}\|_2^2 + \Omega D_0/(L_{k+1} r_{k+1})$    (15)
11: end for
Ensure: $x^N$
```

Algorithm 3 Adaptive Stochastic Accelerated Gradient Method

```
Require: Number of iterations $N$, $D_0$ accuracy $\varepsilon$, $\Omega \geq 1$, $A_0 = 0$,
    initial guess $L_0$.
 1: for $k = 0, \ldots, N-1$ do
 2:     $L_{k+1} := L_k/2$
 3:     repeat
 4:         $L_{k+1} := 2L_{k+1}$
 5:
            $\alpha_{k+1} = (1 + \sqrt{1 + 8A_k L_{k+1}})/(4L_{k+1})\,;\; A_{k+1} = A_k + \alpha_{k+1}$
 6:
            $r_{k+1} = \max\{\Omega \alpha_{k+1} D_0/\varepsilon,\ 1\}$
 7:
            $y^{k+1} = (\alpha_{k+1} u^k + A_k x^k)/A_{k+1}$
 8:
            $u^{k+1} = u^k - \alpha_{k+1} \nabla^{r_{k+1}} f(y^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}})$
 9:
            $x^{k+1} = (\alpha_{k+1} u^{k+1} + A_k x^k)/A_{k+1}$
10:     until
            $f(x^{k+1}) \leq f(y^{k+1}) +$
                $\langle \nabla^{r_{k+1}} f(y^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - y^{k+1} \rangle +$
                $L_{k+1} \|x^{k+1} - y^{k+1}\|_2^2 + \Omega D_0/(L_{k+1} r_{k+1})$    (15)
11: end for
Ensure: $x^N$
```

Practical aspects

$$f(x) = \mathbb{E} f(x, \xi). \tag{16}$$

In this case we estimate the function as a sample average

$$f(x, \{\xi_l\}_{l=1}^{r}) = \frac{1}{r} \sum_{l=1}^{r} f(x, \xi_l)$$

and use it in adaptive procedures. In this case we interpret L_k as the worst constant among all Lipschitz constants for $f(x, \xi)$ with different realization of ξ. Indeed, if L_{k+1} satisfies the following

$$f(x^{k+1}, \xi^{k+1}) \le f(x^k, \xi^{k+1}) + \langle \nabla f(x^k, \xi), x^{k+1} - x^k \rangle + L_{k+1} \|x^{k+1} - x^k\|_2^2 + \varepsilon/2.$$

Then it satisfies

$$f(x^{k+1}, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}) \le f(x^k, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}) + \langle \nabla^{r_{k+1}} f(x, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - x^k \rangle + L_{k+1} \|x^{k+1} - x^k\|_2^2 + \varepsilon/2. \tag{17}$$

If, e.g, (16) holds we replace adaptive procedure in the algorithms by (17).

Algorithm 4 Non-convex Stochastic Gradient Descent

```
Require: Number of iterations $N$, variance $D$, Lipschitz constant $L$, accuracy $\varepsilon$
1: Calculate
       $r = \max\{12D/(\varepsilon^2),\ 1\}$
2: for $k = 0, \dots, N-1$ do
3:     $x^{k+1} = x^k - \frac{1}{2L}\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r)$
4: end for
Ensure: $\hat{x} = \arg\min_{k=1,..N} \|\nabla f(x^k)\|_2$.
```

Theorem 2. Algorithm 4 with the total number of stochastic gradient oracle calls [2] $T = O\left(\frac{DL(f(x^0)-f(x^*))}{\varepsilon^4}\right)$ and number of iterations $N = O\left(\frac{L(f(x^0)-f(x^*))}{\varepsilon^2}\right)$ outputs a point [3] $\hat{x}^N$ which satisfies

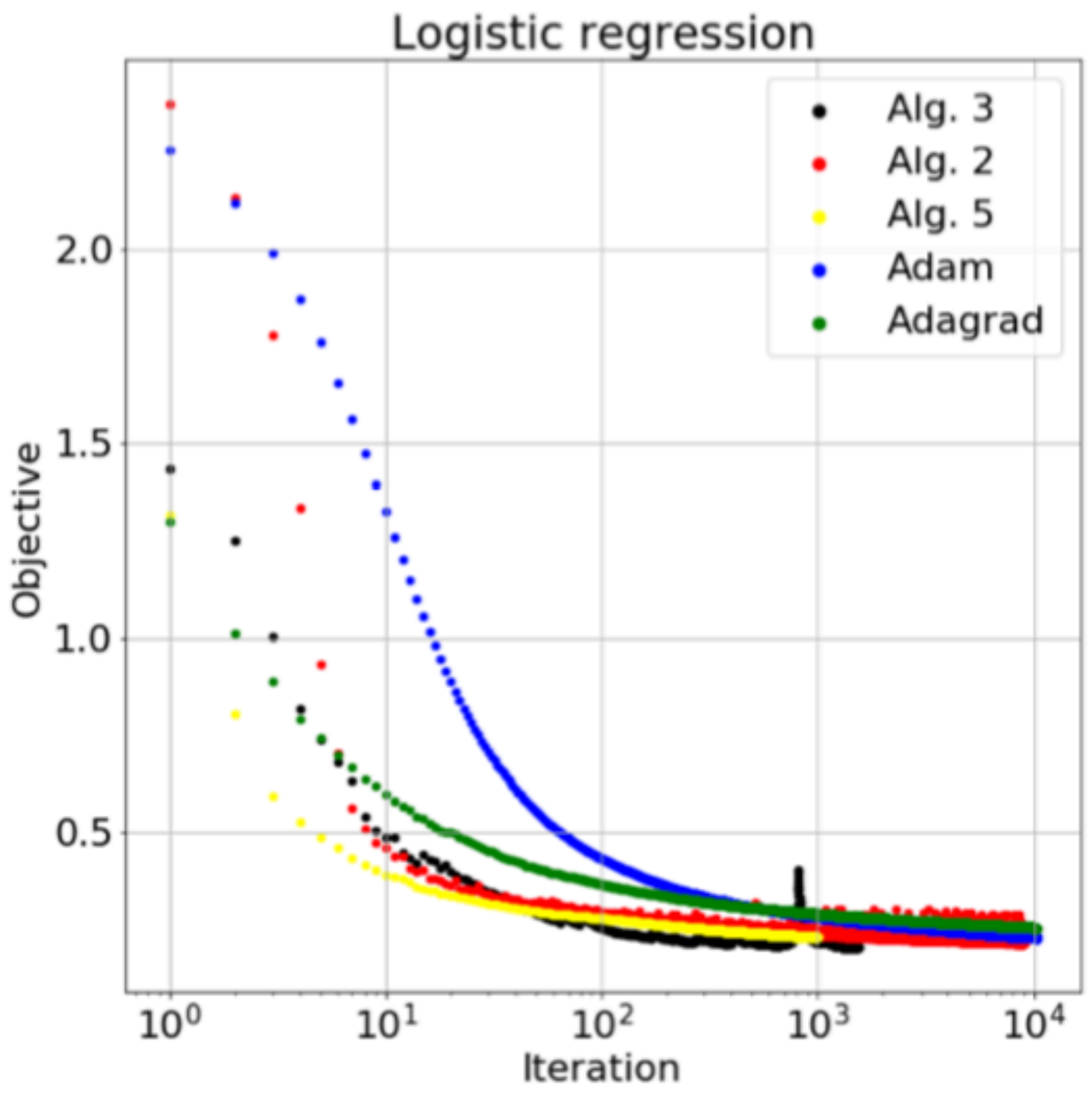
$$\|\nabla f(\hat{x}^N)\|_2 \le \varepsilon. \tag{19}$$

Algorithm 5 Adaptive Non-convex Stochastic Gradient Descent

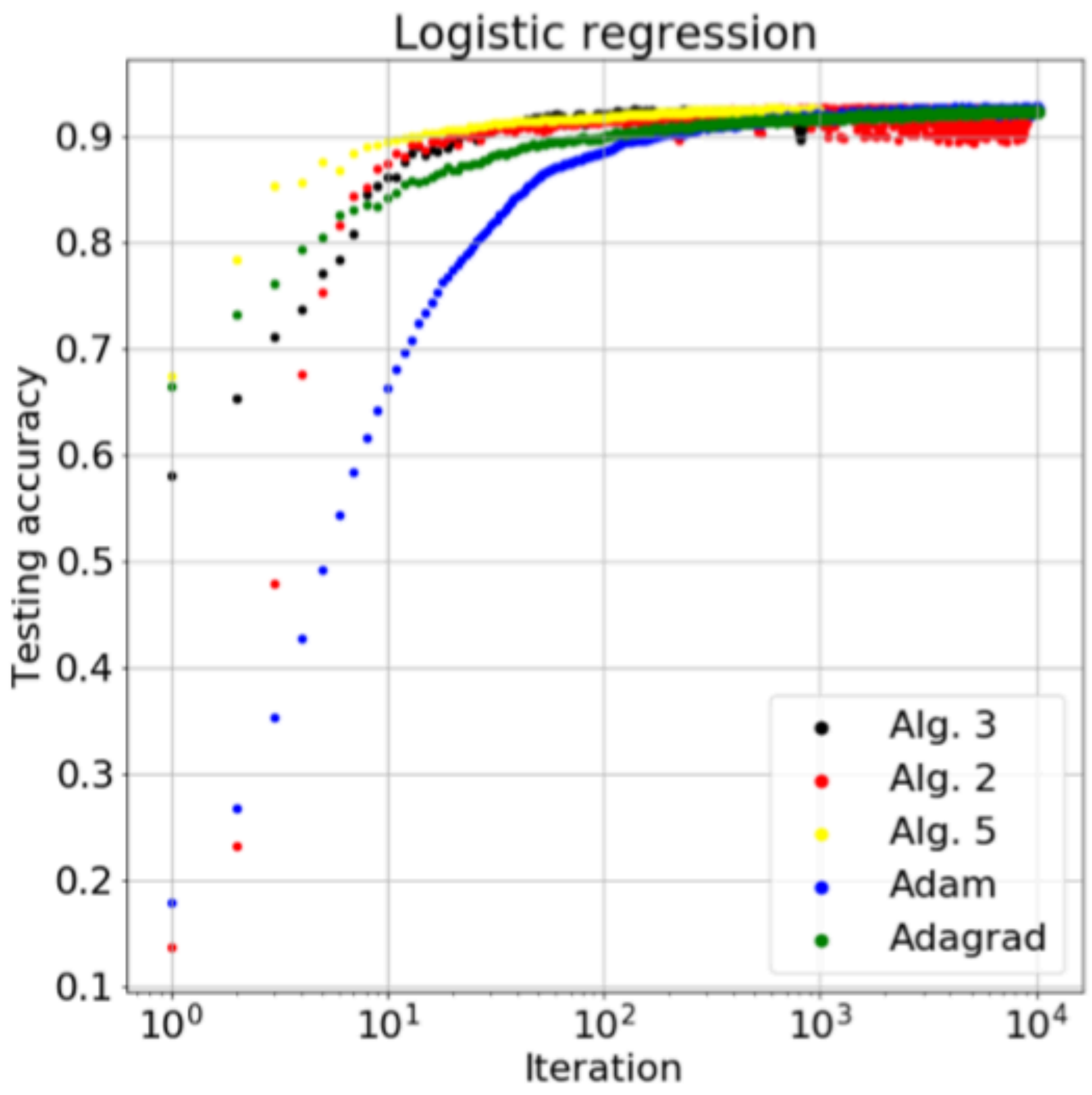
```
Require: Number of iterations $N$, $D_0$, accuracy $\varepsilon$, initial guess $L_0$
1: Calculate
       $r = \max\{8D_0/(\varepsilon^2),\ 1\}$
2: for $k = 0, \dots, N-1$ do
3:     $L_{k+1} := L_k/4$.
4:     repeat
5:         $L_{k+1} := 2L_{k+1}$
6:
           $x^{k+1} = x^k - \frac{1}{2L_k}\nabla^r f(x^k, \{\xi_l^{k+1}\}_{l=1}^r)$
7:     until
           $f(x^{k+1}) \le f(x^k) + \langle \nabla^{r_{k+1}} f(x^k, \{\xi_l^{k+1}\}_{l=1}^{r_{k+1}}), x^{k+1} - x^k\rangle + L_{k+1}\|x^{k+1} - x^k\|_2^2 + \frac{\varepsilon^2}{32L_{k+1}}$
8: end for
Ensure: $\hat{x} = \arg\min_{k=1,..N} \|\nabla f(x^k)\|_2$.
```

Theorem 3. Algorithm 5 with expected number of stochastic gradient oracle calls $\tilde{T} = O\left(\frac{D_0 L(f(x^0)-f(x^N))}{\varepsilon^4}\right)$ and expected number of iterations $\tilde{N} = O\left(\frac{L(f(x^0)-f(x^N))}{\varepsilon^2}\right)$ outputs a point $\hat{x}$ satisfying

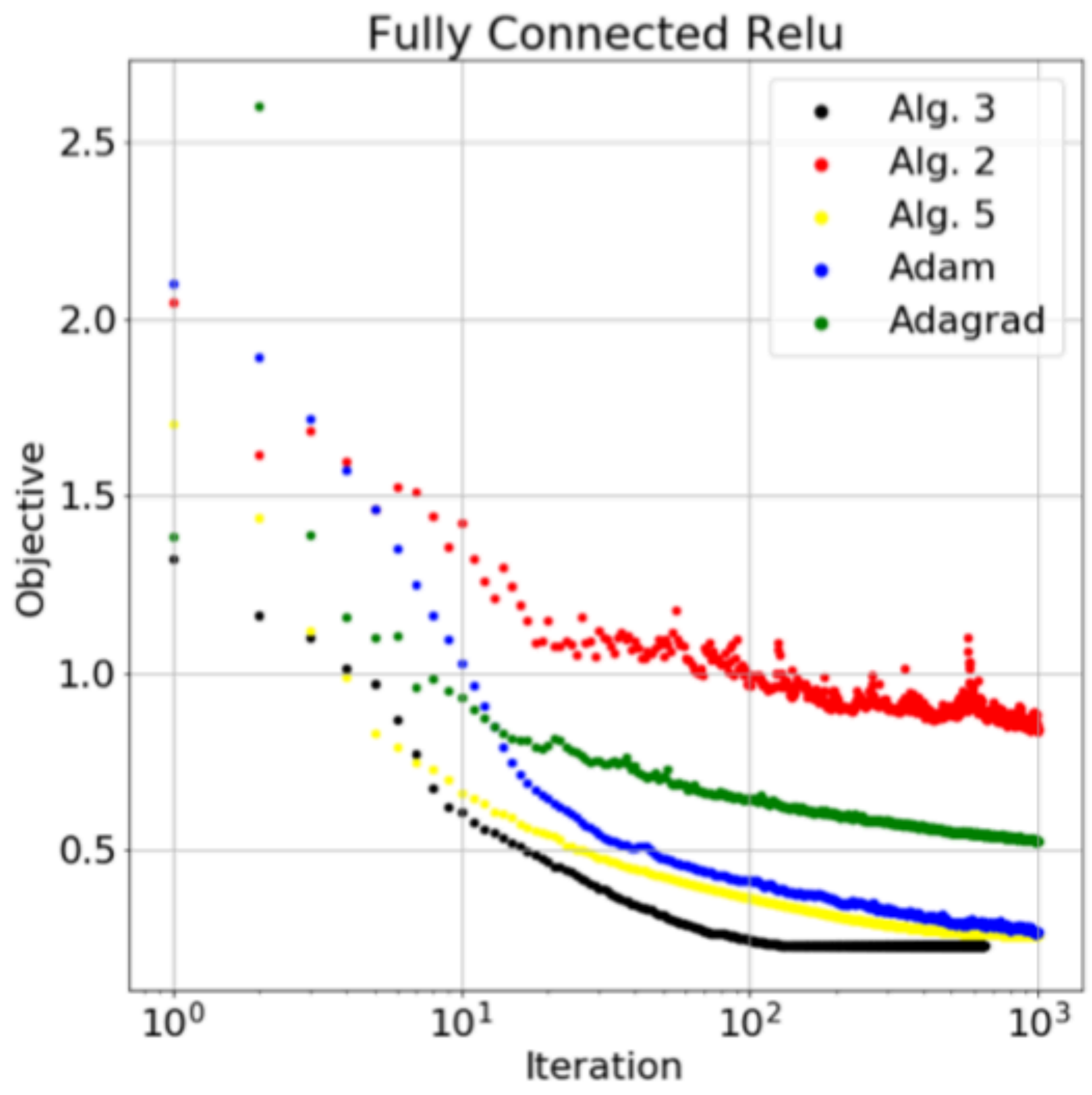
$$\|\nabla f(\hat{x})\|_2 \le \varepsilon.$$



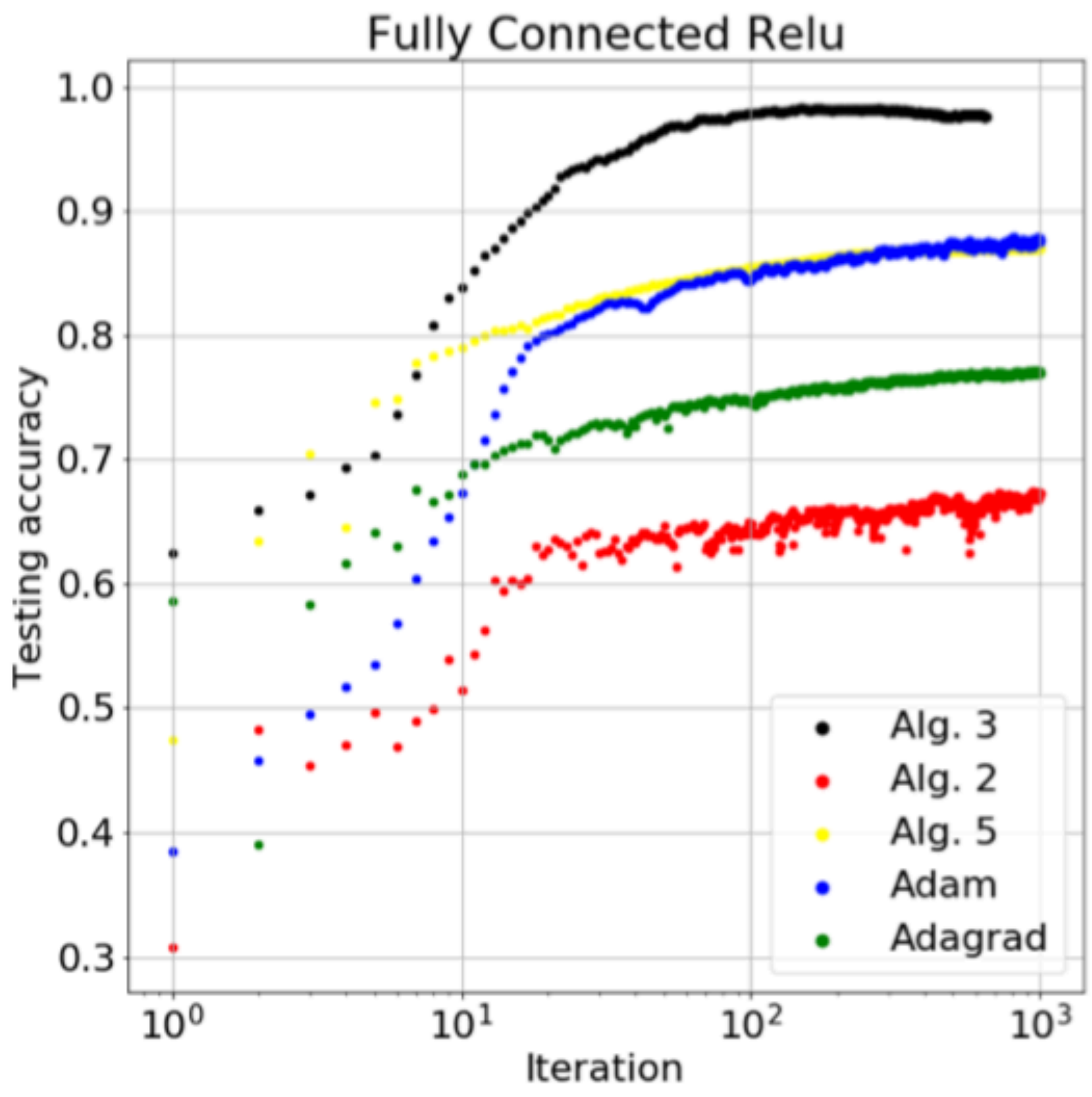
(a) Objective



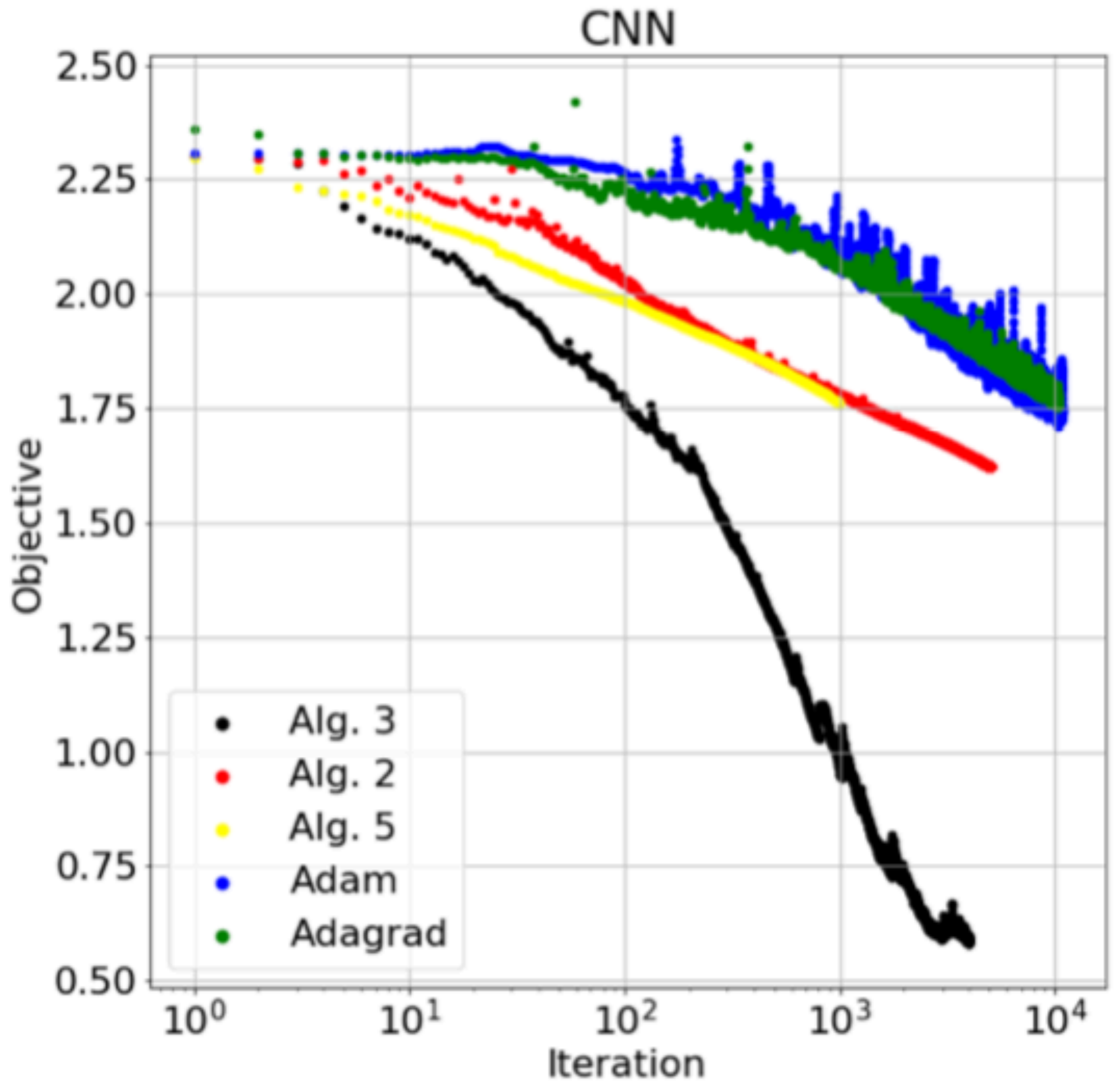
(b) Testing accuracy



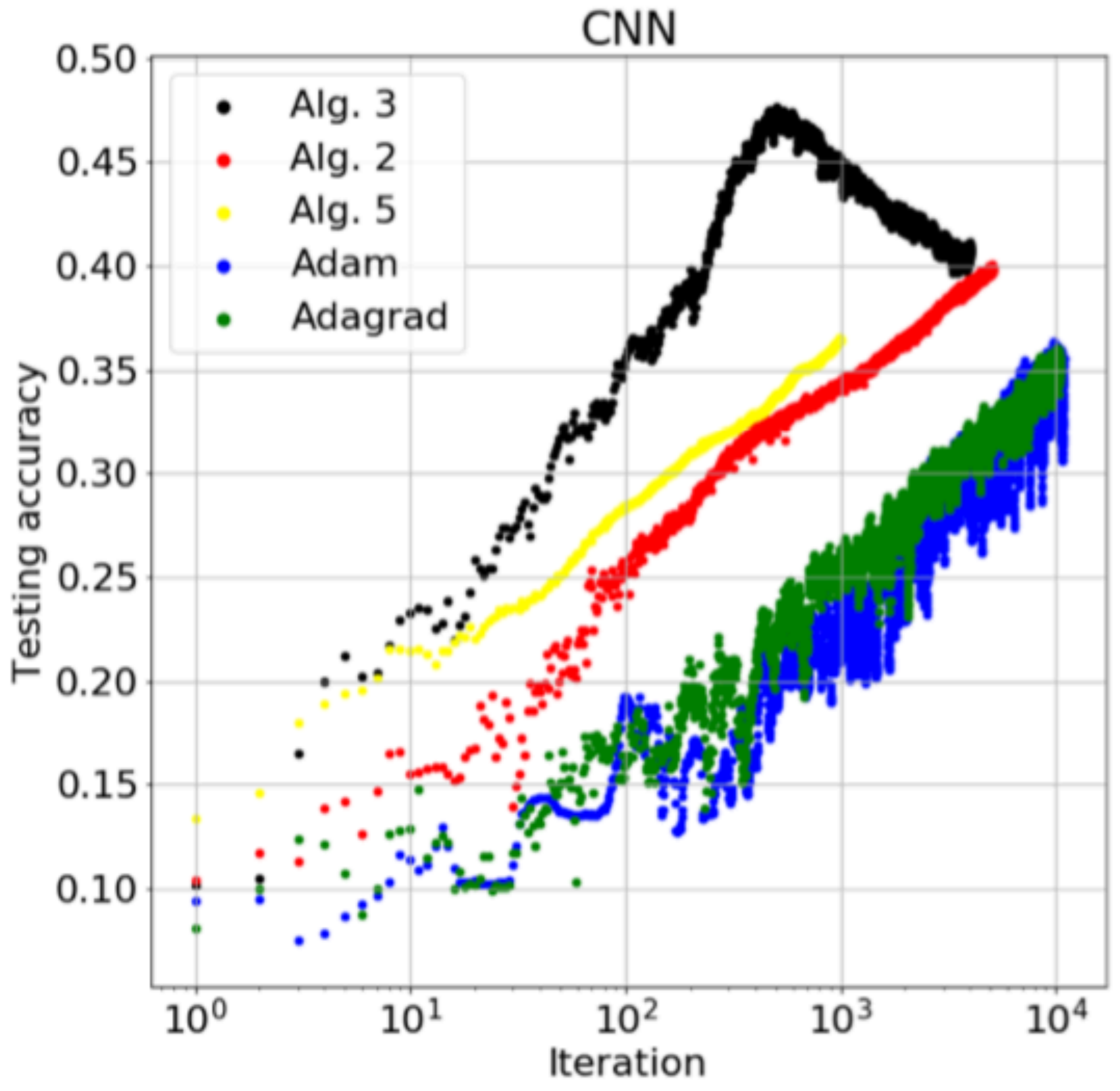
(e) Objective



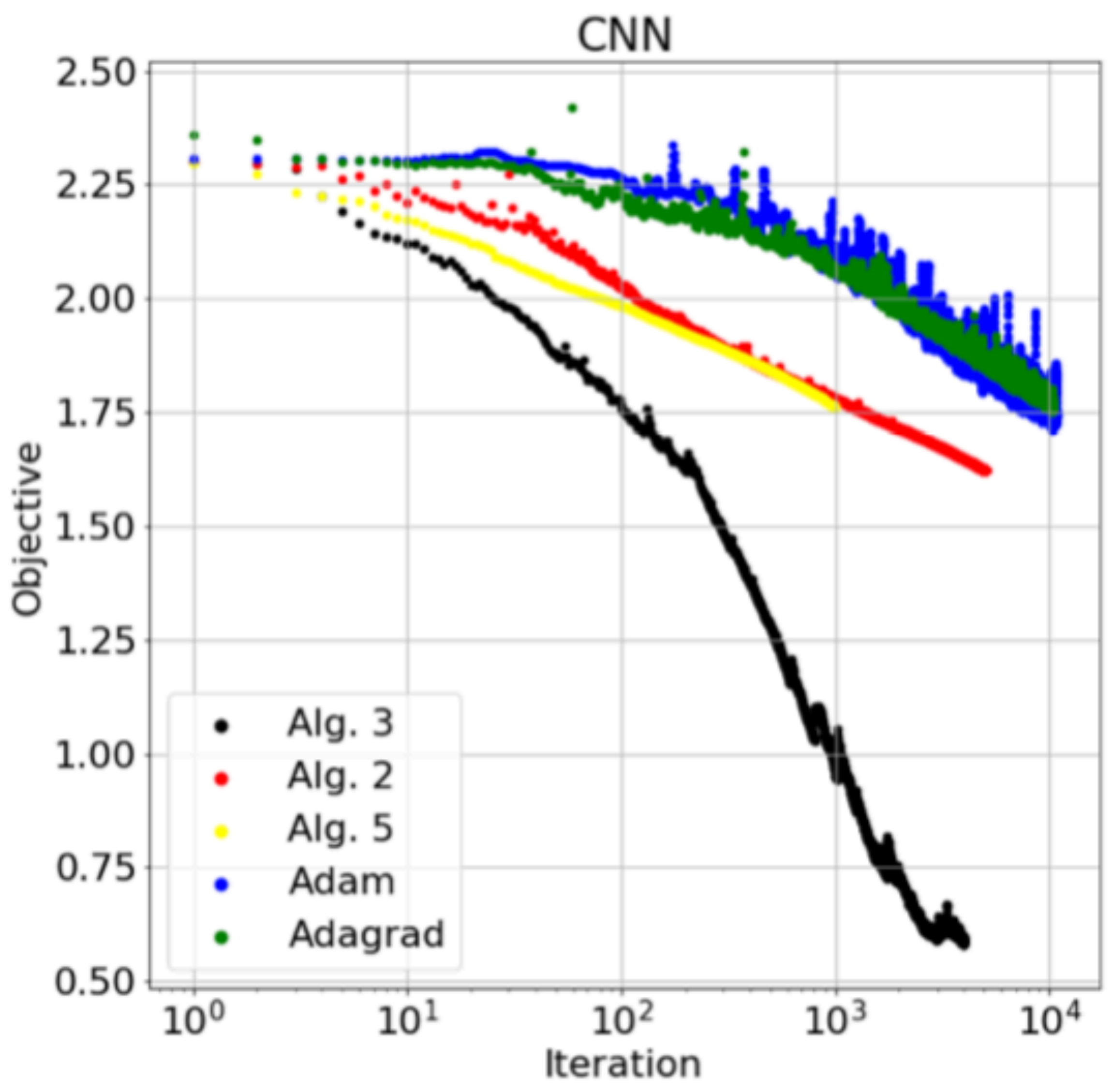
(f) Testing accuracy



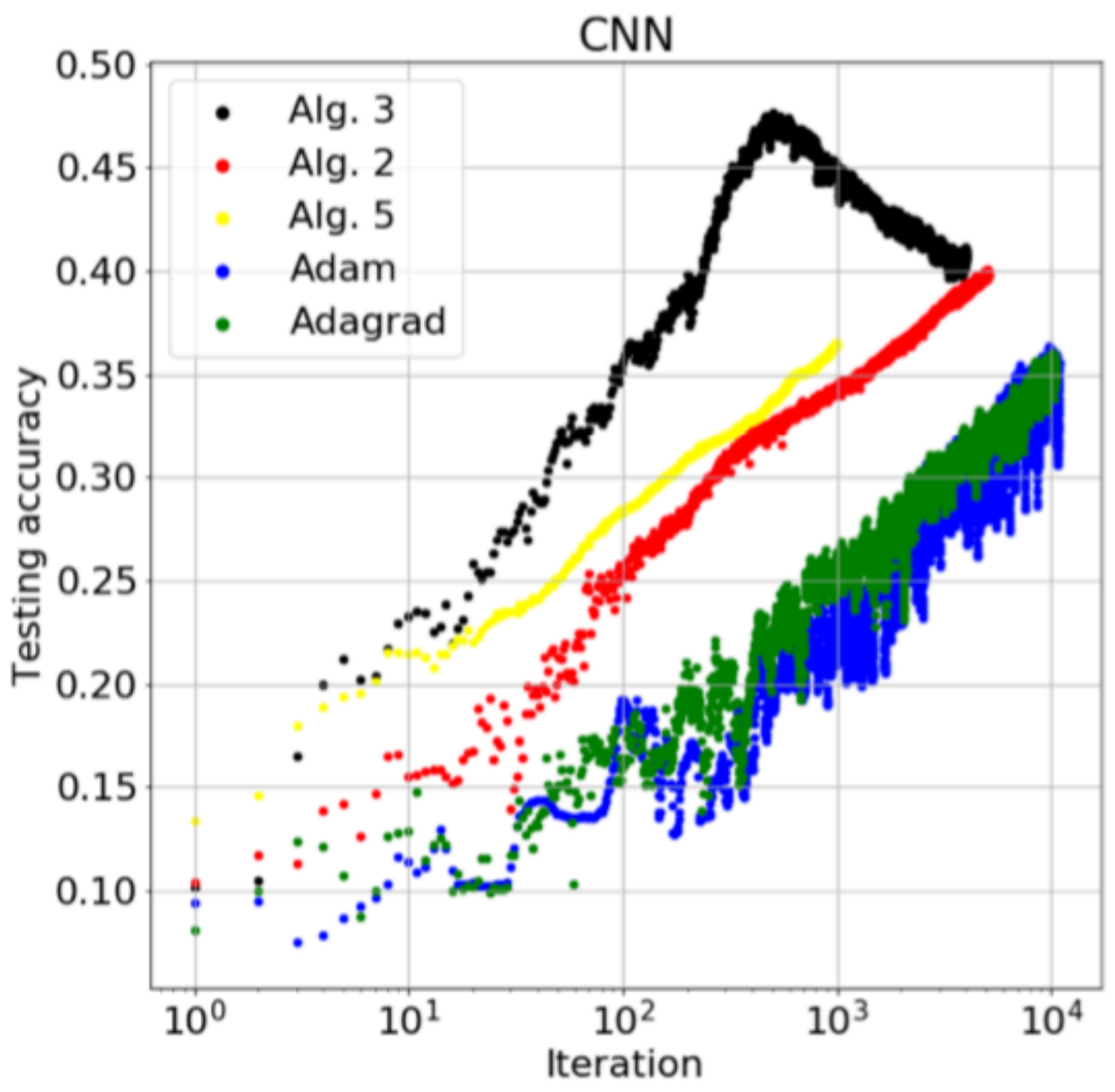
(g) Objective



(h) Testing accuracy



(g) Objective



(h) Testing accuracy

Adaptive Catalyst

Gasnikov Alexander (MIPT gasnikov@yandex.ru)
Ivanova Anastasia (HSE, MIPT anastasiya.s.ivanova@phystech.edu)

$$\min_{x \in \mathbb{R}^n} f(x)$$

To propose the main scheme of the algorithm we need to define the following functions:

$$F_{L,x}(y) = f(y) + \frac{L}{2}||y - x||_2^2$$

Algorithm 1 Monteiro–Svaiter algorithm

```
1: Parameters: $z^0, y^0, A_0 = 0$
2: for $k = 0, 1, \dots, N-1$ do
3:     Choose $L_{k+1}$ and $y^{k+1}$ such that
           $\|\nabla F_{L_{k+1}, x^{k+1}}(y^{k+1})\|_2 \le \frac{L_{k+1}}{2}\|y^{k+1} - x^{k+1}\|_2,$
       where
           $a_{k+1} = \frac{1/L_{k+1} + \sqrt{1/L_{k+1}^2 + 4A_k/L_{k+1}}}{2},$
           $A_{k+1} = A_k + a_{k+1},$
           $x^{k+1} = \frac{A_k}{A_{k+1}} y^k + \frac{a_{k+1}}{A_{k+1}} z^k$
4:     $z^{k+1} = z^k - a_{k+1} \nabla f\left(y^{k+1}\right)$
5: end for
```

Theorem 1. (Monteiro and Svaiter, 2013, Theorem 3.6) Let sequence (x^k, y^k, z^k), $k \geq 0$ be generated by Algorithm 1 and define $R := \left\|y^0 - x_\star\right\|_2$. Then, for all $N \geq 0$,

$$\frac{1}{2}\left\|z^N - x_\star\right\|_2^2 + A_N \cdot \left(f\left(y^N\right) - f\left(x_\star\right)\right) + \frac{1}{4}\sum_{k=1}^{N} A_k L_k \left\|y^k - x^k\right\|_2^2 \leq \frac{R^2}{2}, \qquad (4)$$

$$f\left(y^N\right) - f\left(x^\star\right) \leq \frac{R^2}{2A_N}, \quad \left\|z^N - x_\star\right\|_2 \leq R, \qquad (5)$$

$$\sum_{k=1}^{N} A_k L_k \left\|y^k - x^k\right\|_2^2 \leq 2R^2. \qquad (6)$$

Lemma 2. (Monteiro and Svaiter, 2013, Lemma 3.7 a)) Let sequences $\{A_k, L_k\}$, $k \geq 0$ be generated by Algorithm 1. Then, for all $N \geq 0$,

$$A_N \geq \frac{1}{4}\left(\sum_{k=1}^{N} \frac{1}{\sqrt{L_k}}\right)^2. \qquad (7)$$

Algorithm 2 Adaptive Catalyst

```
Require: Starting point $x^0$ initial guess $L_0 > 0$; parameters $\alpha > \beta > \gamma > 0$; optimization method $\mathcal{M}$.
 1: for $k = 0, 1, \dots, N-1$ do
 2:     $L_{k+1} = \beta \cdot \min\{\alpha L_k, L_u\}$
 3:     $t = 0$
 4:     repeat
 5:         $t := t + 1$
 6:         $L_{k+1} := \max\{L_{k+1}/\beta, L_d\}$
 7:         Compute
              $$a_{k+1} = \frac{1/L_{k+1} + \sqrt{1/L_{k+1}^2 + 4A_k/L_{k+1}}}{2},$$
              $$A_{k+1} = A_k + a_{k+1},$$
              $$x^{k+1} = \frac{A_k}{A_{k+1}} y^k + \frac{a_{k+1}}{A_{k+1}} z^k.$$
 8:         Compute an approximate solution of the following problem with auxiliary non-accelerated method $\mathcal{M}$
              $$y^{k+1} \approx \underset{y}{\operatorname{argmin}} F_{L_{k+1}, x^{k+1}}(y)$$
 9:         By running $\mathcal{M}$ with starting point $x^{k+1}$ and output point $y^{k+1}$ we wait $N_t$ iterations to fulfill adaptive stopping criteria
              $$\|\nabla F_{L_{k+1}, x^{k+1}}(y^{k+1})\|_2 \le \frac{L_{k+1}}{2} \|y^{k+1} - x^{k+1}\|_2.$$
10:     until $t > 1$ and $N_t \ge \gamma \cdot N_{t-1}$ or $L_{k+1} = L_d$
11:     $z^{k+1} = z^k - a_{k+1} \nabla f\left(y^{k+1}\right)$
12: end for
13: Output: $y^{k+1}$
```

Proposition 3. The convergence rate for the method $\mathcal{M}$ for problem

$$\min_{y \in \mathbb{R}^n} F(y)$$

can be written in the general form as follows (for randomized algorithms, like Algorithm 4, this estimates holds true with high probability)

$$F(y^N) - F(y_\star) = O\left(L_F R^2\right) \min\left\{\frac{C_n}{N}, \exp\left(-\frac{\mu_F N}{C_n L_F}\right)\right\},$$

where $y_\star$ is the solution of the problem, $R = ||y^0 - y_\star||_2$, function F is μ_F–strongly convex and L_F is a constant which characterized smoothness of function F.

Typically $C_n = O(1)$ for the standard full gradient first order methods, $C_n = O(p)$, where p is a number of blocks, for alternating minimization with p blocks and $C_n = O(n)$ for gradient free or coordinate descent methods, where n is dimension of y. See the references in next Remark for details.

Theorem 4. Consider Algorithm 2 with $L_d < L_u$ for solving problem (1), where $Q = \mathbb{R}^n$, with auxiliary (inner) non-accelerated algorithm (method) $\mathcal{M}$ that satisfy Proposition 3 with constants C_n and L_f such that $L_d \leq L_f \leq L_u$.

Then the total complexity of the proposed Algorithm 2 with inner method $\mathcal{M}$ is

$$\tilde{O}\left(C_n \cdot \max\left\{\sqrt{\frac{L_u}{L_f}}, \sqrt{\frac{L_f}{L_d}}\right\} \cdot \sqrt{\frac{L_f R^2}{\varepsilon}}\right).$$

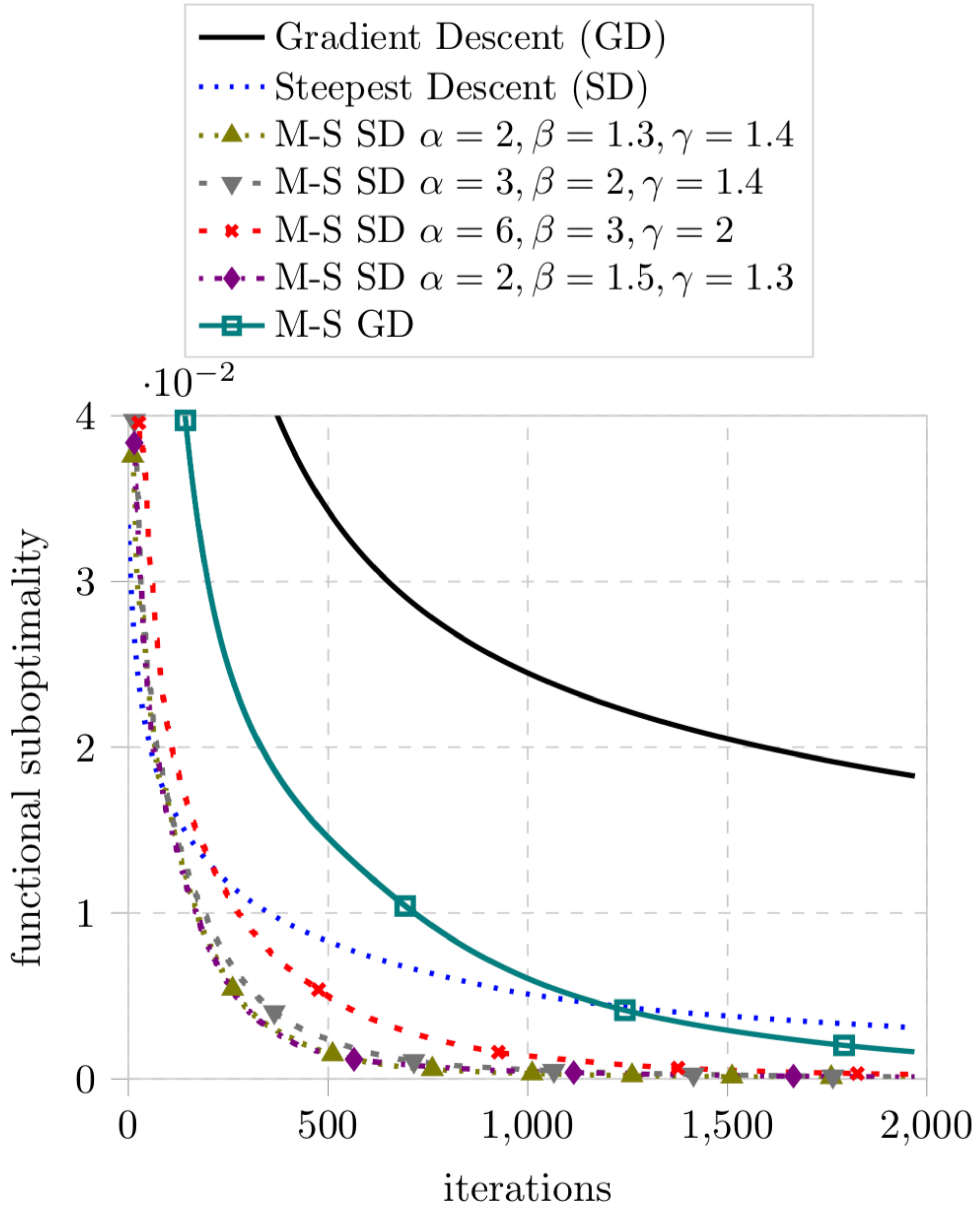


Fig. 2. Logistic regression (12) with a1a dataset from LIBSVM repository.

To apply M-S envelop for non-convex case we need one trick

$$\min_{x,y} \quad F(x,y) = \sum_{u,i} c_{ui}\left(p_{ui} - x_u^\top y_i\right)^2 + \lambda\left(\sum_u \|x_u\|_2^2 + \sum_i \|y_i\|_2^2\right)$$

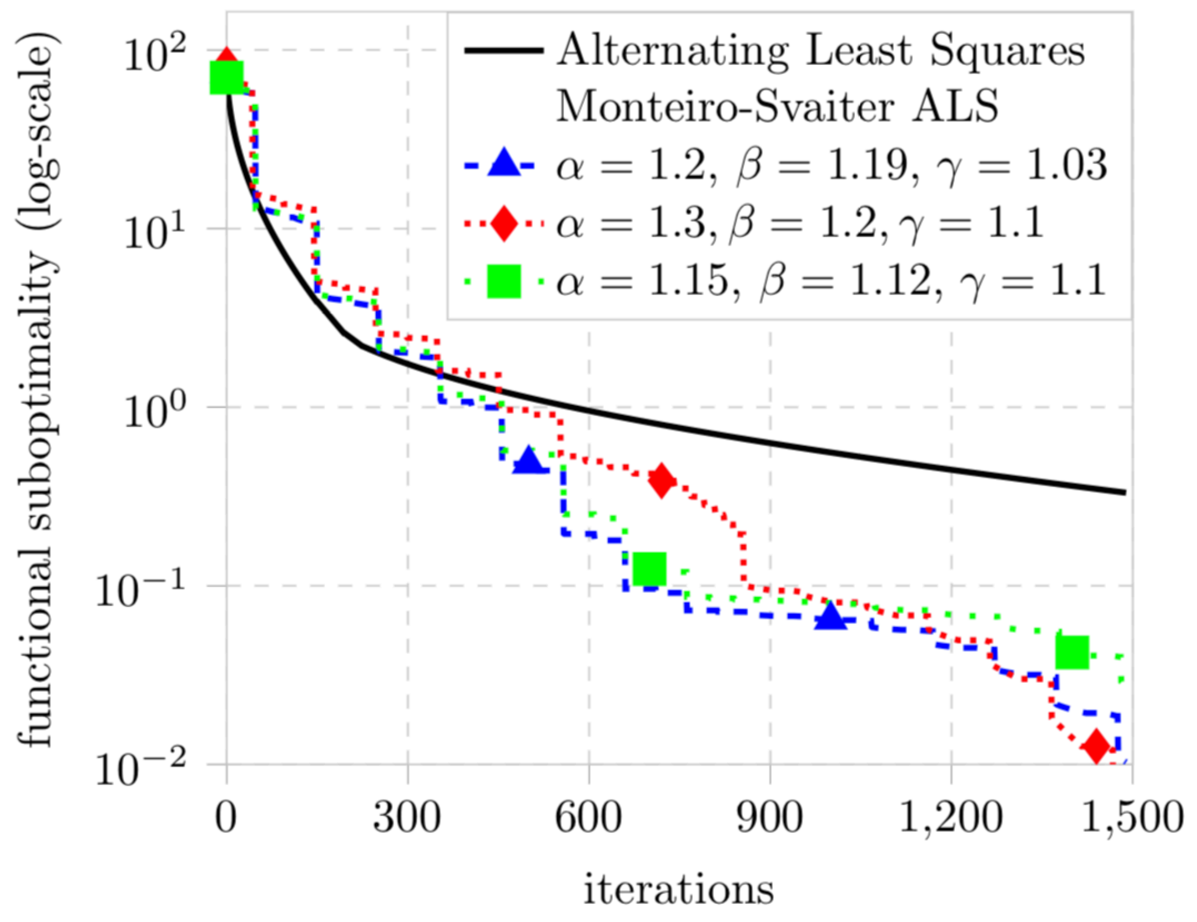


Figure 2: Matrix completion problem (13) with different (α, β, γ).

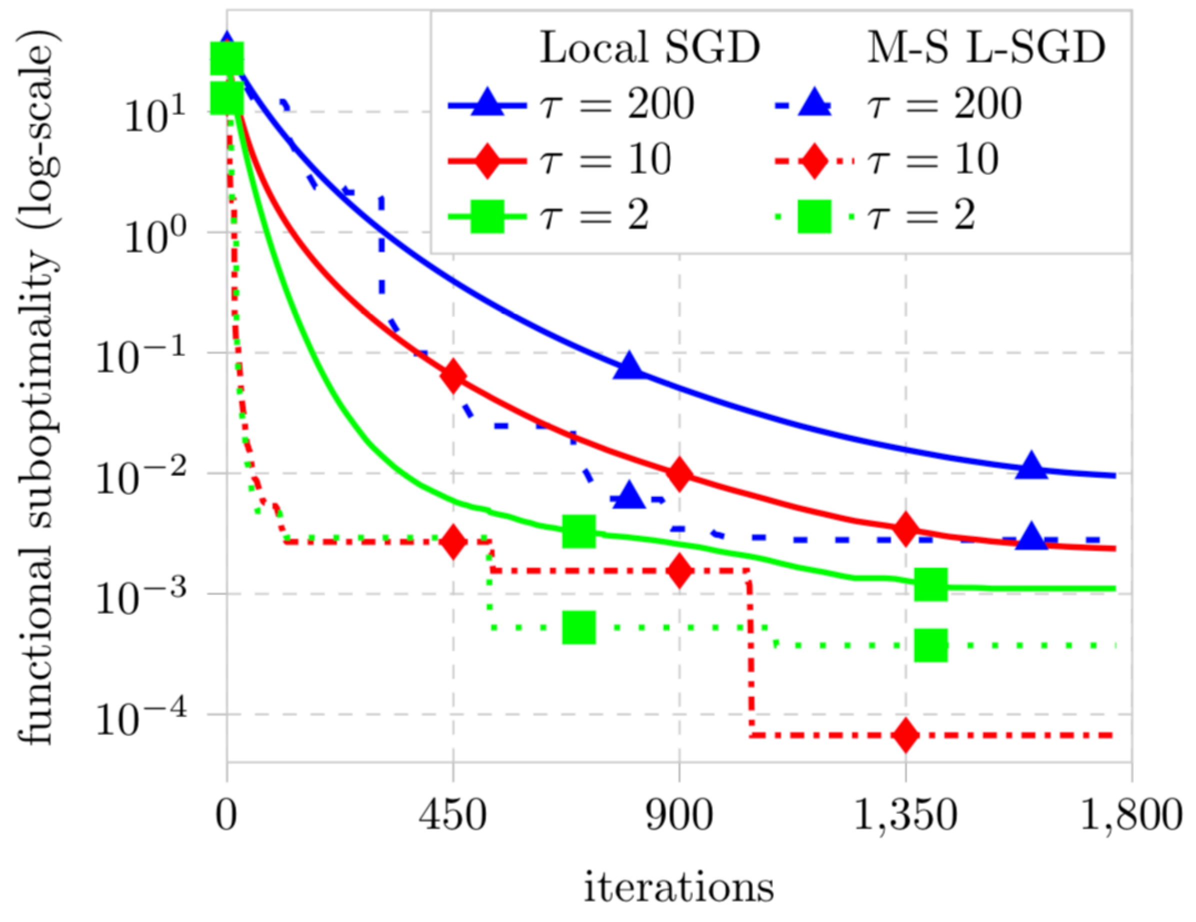


Figure 3: Regularized logistic loss (14) for different synchronization intervals τ.

$$\tilde{y}_{k+1} \approx \operatorname*{argmin}_y F_{L,x}^{k+1}(y)$$

$$y_{k+1} = \arg\min\{f(y) \mid y \in \{y_k, \tilde{y}_{k+1}\}\}$$

Popular nowadays Federated Learning