Stochastic Methods for Optimization and Sampling

M1, TSE

Convergence Rates for Stochastic Algorithms

Course organization

The course is divided into two parts

1. **Stochastic optimization** ($\sim$ 15 hours): A continuation of the S1's course on convex optimization, with a focus on variations of *stochastic gradient descent*.

2. **Sampling methods** ($\sim$ 15 hours): Focuses on methods to generate random variables following a given distribution.

A complete set of lecture notes is available on Moodle.

Feel free to ask for adjustments, feedback is appreciated!

Contact: julien.chhor@tse-fr.eu

Grading

The final grade will be calculated as follows.

$$\text{Final Grade} = \max\left(\text{Exam},\ \frac{1}{4}\text{Homework} + \frac{3}{4}\text{Exam}\right) + \text{Bonus}.$$

- **Exam:** Focused on theoretical understanding and mathematical proofs (2 hours).
- **Homework:** A set of coding exercises to implement the algorithms seen in class. Must be completed in Python.
- **Bonus:** For any typo/error found in the material (lecture notes, slides, exercise sheets, solutions...), the first person to point it out will receive 0.5 points on their final grade.

Syllabus: Stochastic optimization

We will cover the following algorithms and study their theoretical properties

- Stochastic gradient descent
- Minibatch stochastic gradient descent
- Proximal stochastic gradient descent
- Variance reduction techniques (e.g., SVRG)
- Acceleration methods (Momentum, Nesterov acceleration)
- Algorithms with adaptive step sizes (Adagrad, RMSprop, Adam)

Syllabus: Sampling methods

We will cover the following algorithms to obtain simulations of random variables following a given distribution

- Simulation of random variables with classical probability distributions (Bernoulli, binomial, multinomial, exponential...)
- Inversion of the CDF method
- Rejection method
- Simulation of normally distributed random variables
- Monte Carlo methods and confidence intervals
- Importance sampling (variance reduction, rare event simulation)
- Markov Chain Monte Carlo (MCMC) methods (Metropolis-Hastings algorithm, Gibbs sampling).

Stochastic algorithms: Motivations

Motivations: Classical optimization methods

Let $F : \mathbb{R}^d \to \mathbb{R}$ be a function to minimize over a set $\Theta \subset \mathbb{R}^d$:

$$\inf_{\theta \in \Theta} F(\theta).$$

Classical strategies: We can solve the problem

- **analytically**: KKT, Lagrange duality for example
- **numerically**: Gradient descent, Newtonian methods etc.

Quick reminder on Gradient Descent

Given a function $F : \mathbb{R}^d \to \mathbb{R}$, we want to solve

$$\min_{\theta \in \mathbb{R}^d} F(\theta)$$

Gradient Descent

1. Start from an initial guess $\theta_0 \in \mathbb{R}^d$
2. Until termination condition, iterate:

$$\theta_{k+1} = \theta_k - \gamma_k \nabla f(\theta_k).$$

where $\gamma_k > 0, \forall k$.

Here, γ_k is called the *step size*.

Advantages and drawbacks of GD

Advantages of GD

If F is convex or strongly convex, GD converges "fast" toward the minimizer θ^*:

- If $\begin{cases} F \text{ is convex} \\ \nabla F \text{ is } L\text{-Lipschitz,} \end{cases}$ then $F(\theta_k) - F(\theta^*) = O(1/k)$.
- If $\begin{cases} F \text{ is } \mu\text{-strongly convex} \\ \nabla F \text{ is } L\text{-Lipschitz,} \end{cases}$ then $F(\theta_k) - F(\theta^*) = O(e^{-\mu k/L})$.

In both cases, we may take $\gamma_k = 1/L$.

Advantages and drawbacks of GD

Drawbacks of GD

1. It can get stuck in local minima for non-convex functions.

2. For certain functions F to optimize, one GD iteration can be extremely slow to compute. Such functions appear frequently in modern machine learning.

A motivating example

A difficult case for gradient descent

Suppose we apply gradient descent to minimize

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta), \qquad \text{for some "very large" } n.$$

Each iteration of GD requires computing n gradients.

$$\theta_{k+1} = \theta_k - \frac{\gamma_k}{n} \underbrace{\sum_{i=1}^{n} \nabla f_i(\theta_k)}_{n \text{ gradients to compute!}}$$

This can be prohibitive if n is too large.

How to circumvent this issue?

A solution: The SGD algorithm

Idea: At each step k, select *one* index

$$i_k \sim \text{Unif}(\{1, \dots, n\})$$

and replace $\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_k)$ with $\nabla f_{i_k}(\theta_k)$.

Stochastic Gradient Descent (SGD)

1. Start from an initial guess $\theta_0 \in \mathbb{R}^d$
2. Until termination, iterate:

$$i_k \sim \text{Unif}(\{1, \dots, n\})$$

$$\theta_{k+1} = \theta_k - \gamma_k \nabla f_{i_k}(\theta_k).$$

where $\gamma_k > 0, \forall k$.

Warning: The step sizes γ_k's are different for SGD and GD.

Advantages and drawbacks of SGD

Advantages of SGD over GD:

- Each step is n times faster to compute (though less precise). This can lead to considerable runtime improvement.
- SGD can sometimes escape from local minima (but not always).

Drawbacks of SGD:

- Strictly speaking, it is NOT a descent algorithm: $F(\theta_k)$ can go up and down, but "tends to decrease".
- Can still get stuck in local minima for non-convex objectives
- Slower dependence in k than GD, but possibly better dependence on other problem parameters (typically n). (Proof later in the course)

How usual is this issue?

In practice, it is very common to encounter the structure

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta), \qquad \text{where } n \text{ is "very large"} ,$$

especially in the context of *empirical risk minimization*.

Example: Linear regression

Observations: $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ where

$$\underbrace{y_i}_{\text{observed}} = \underbrace{x_i}_{\text{observed}}^{\top} \underbrace{\theta^*}_{\text{to learn}} + \xi_i \qquad \text{for some } \theta^* \in \mathbb{R}^d.$$

Goal: Estimate θ^*, or learn a predictor of the form $\varphi_\theta(x) = x^\top \theta$.

Here ξ_i's are iid, centered noises, mutually independent of the $x_i's$.

Example: Linear regression

Classical estimator: OLS (for small dimension, i.e. $d < n$)

The Ordinary Least Squares (OLS) estimator is given by

$$\begin{aligned}\hat{\theta}^{OLS} &= \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^\top \theta\right)^2 \\ &= \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)\end{aligned}$$

where $f_i(\theta) = (y_i - x_i^\top \theta)^2$.

It exactly solves $\arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$ where n is possibly very large.

Empirical risk minimization

It turns out that θ^{OLS} has a closed-form expression that can be computed directly, without using GD or SGD.

But in general, empirical risk minimization requires optimization algorithms (e.g. for training neural networks, etc).

General setting of supervised learning

Observations: n iid data points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$.

Goal: Find a predictor $\varphi : \mathcal{X} \to \mathcal{Y}$ such that $\varphi(x)$ "predicts well" y.

Define a loss function $\ell : \mathcal{Y}^2 \to \mathbb{R}_+$, typically convex. The quantity $\ell\big(y_i, \varphi(x_i)\big)$ represents the error for predicting y_i by $\varphi(x_i)$.

Empirical risk minimization

For a parametrization $(\varphi_\theta)_{\theta\in\Theta}$, we select θ as the minimizer of the empirical risk on the dataset:

$$\hat{\theta} = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \varphi_\theta(x_i)).$$

One of the most commonly used predictors in machine learning.

It is a minimization problem of the form

$$\arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} f_i(\theta), \qquad \text{where} \qquad f_i(\theta) = \ell(y_i, \varphi_\theta(x_i)).$$

Claim: $\hat{\theta}$ will have good generalization properties under suitable conditions.

Summing up

Goal: Given functions $f_i : \Theta \subset \mathbb{R}^d \to \mathbb{R}$, we want to solve

$$\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta), \qquad \text{for some "very large" } n.$$

This problem often arises for *empirical risk minimization* in the context of supervised ML.

Problem with Gradient Descent: Each step requires computing n gradients:

$$\theta_{k+1} = \theta_k - \frac{\gamma_k}{n} \underbrace{\sum_{i=1}^{n} \nabla f_i(\theta_k)}_{n \text{ gradients to compute!}}.$$

Prohibitive if n is too large.

The SGD algorithm

Idea: At each step k, select *one* index

$$i_k \sim \text{Unif}(\{1, \dots, n\})$$

and replace $\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_k)$ with $\nabla f_{i_k}(\theta_k)$.

Stochastic Gradient Descent

1. Start from an initial guess $\theta_0 \in \mathbb{R}^d$
2. Until termination, iterate:

$$i_k \sim \text{Unif}(\{1, \dots, n\}) \text{ independent of the past}$$

$$\theta_{k+1} = \theta_k - \gamma_k \nabla f_{i_k}(\theta_k).$$

where $\gamma_k > 0, \forall k$.

Warning: The γ_k's are different for SGD and GD.

Comparison between GD and SGD

Rates of GD and SGD

Here we consider

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$$

We will prove that, when running GD or SGD for k steps, we will get an error decaying as follows

$k =$ number of iterations	F cvx, ∇F is L-Lip +technical conditions	F μ-stgly cvx, ∇F L-Lip +technical conditions
GD	$\frac{C}{k}$	$C\exp\big(-\frac{\mu k}{L}\big)$
SGD	$\frac{C\log(k)}{\sqrt{k}}$	$\frac{C}{k}$

Table 1: Error obtained after k iterations

Warning: The value of C may change in each appearance!

Rates of SGD for convex objectives

General stochastic gradient descent algorithm

Recall: For some open set $\Theta \subseteq \mathbb{R}^d$, we aim to minimize

$$\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta).$$

More general formulation

It is a special case of

$$\min_{\theta \in \Theta} \mathbb{E}[f(\theta, \xi)],$$

where the expectation is taken with respect to ξ.

(Here, we have a measurable function

$$f : \Theta \times \Xi \to \mathbb{R}$$
$$(x, t) \mapsto f(x, t)$$

and a random variable ξ on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with values in Ξ. We suppose that $\mathbb{E}(|f(x, \xi)|) < +\infty$.)

Link between the two formulations

Indeed, suppose $\Xi = \{1, \dots, n\}$ and $\xi \sim \text{Unif}(\{1, \dots, n\})$. Then

$$\begin{aligned}
\mathbb{E}\left[f(\theta, \xi)\right] &= \sum_{i=1}^{n} \mathbb{E}\big[f(\theta, \xi) \,\big|\, \xi = i\big] \underbrace{\mathbb{P}(\xi = i)}_{=1/n} \\
&= \sum_{i=1}^{n} f(\theta, i) \cdot \frac{1}{n} \\
&=: \frac{1}{n} \sum_{i=1}^{n} f_i(\theta), \qquad\qquad \forall \theta \in \mathbb{R}^d.
\end{aligned}$$

Here, we have defined $f_i = f(\cdot, i),\ \forall i = 1, \dots, n$.

From now on, we will write

$$F(\theta) := \mathbb{E}\left[f(\theta, \xi)\right].$$

General SGD

Assumption 1: The law $\mathbb{P}_\xi$ of ξ may be known or unknown, but we assume that we can easily *generate* $\xi_1, \ldots, \xi_k \overset{iid}{\sim} \mathbb{P}_\xi$.

Assumption 2: We assume that $E\left[\nabla f(\theta, \xi)\right] = \nabla F(\theta)$ for any θ.

General SGD Algorithm

1. Start from an initial guess θ_0
2. Until termination condition, iterate:

$$\text{Draw } \xi_{k+1} \sim \mathbb{P}_\xi, \text{ independent of } (\xi_1, \ldots, \xi_k)$$

$$\theta_{k+1} = \theta_k - \gamma_k \nabla_\theta f(\theta_k, \xi_{k+1})$$

where $\gamma_k > 0, \forall k$.

Here $\nabla_\theta f\left(\theta_k, \xi_{k+1}\right)$ denotes the gradient of $\theta \mapsto f\left(\theta, \xi_{k+1}\right)$ at θ_k.

Rates for convex objectives

Theorem (Rates of SGD for convex functions)

Suppose that:

1. $\theta \mapsto f(\theta, \xi)$ is convex and differentiable for all ξ,
2. $\exists C > 0, \forall \theta \in \mathbb{R}^d : \mathbb{E}\left(\left\|\nabla_\theta f(\theta, \xi)\right\|^2\right) \leq C$,
3. The minimizer $\theta^* \in \arg\min F$ is attained
4. For any $k \in \mathbb{N} : \gamma_k = \frac{\gamma^*}{\sqrt{k+1}}$ for some $\gamma^* > 0$.

The iterates $(\theta_k)_k$ of SGD satisfy the convergence guarantee

$$\mathbb{E}\left[F\left(\bar{\theta}_k^\gamma\right) - F\left(\theta^*\right)\right] \leq \frac{\|\theta_0 - \theta^*\|^2 + C{\gamma^*}^2(1 + \log(k+1))}{4\gamma^*(\sqrt{k+2} - 1)}$$

where $\bar{\theta}_k^\gamma = \frac{\sum_{l=0}^k \gamma_l \theta_l}{\sum_{j=0}^k \gamma_j}$ is a convex combination of all previous iterates.

Remarks on the Theorem

Remarks

1. Need convexity of $\theta \mapsto f(\theta, \xi)$ for *each* $\xi \in \Xi$.
   It *implies* that $F(\theta) = \mathbb{E}\big[f(\theta, \xi)\big]$ is convex in θ.

2. The condition $\exists C > 0, \forall \theta \in \mathbb{R}^d : \mathbb{E}\big(\big\|\nabla f(\theta, \xi)\big\|^2\big) \leq C$ implies that F must be $\sqrt{C}$-Lipschitz (see proof in lecture notes).

3. For convex functions, the convergence rate of SGD is $O\big(\frac{\log(k)}{\sqrt{k}}\big)$ if the step sizes satisfy $\gamma_k = \frac{1}{\sqrt{k+1}}$.

4. This rate is NOT a guarantee on $F(\theta_k) - F(\theta^*)$, but rather on the *expectation* $\mathbb{E}\left[F(\bar{\theta}_k^\gamma) - F(\theta^*)\right]$.

5. If F is just convex, controlling $\mathbb{E}\left[F(\bar{\theta}_k^\gamma) - F(\theta^*)\right]$ does not give any guarantee on $\|\bar{\theta}_k^\gamma - \theta^*\|^2$.

Reminders on convexity

Proposition

If $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, then we have for any $a, b \in \mathbb{R}^d$:

$$f(b) \geq f(a) + \langle \nabla f(a), b - a \rangle$$

Important remarks on runtime

Rates of GD and SGD

Here we consider

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$$

We will prove that, when running GD or SGD for k steps, we will get an error decaying as follows

$k =$ number of iterations	F cvx, ∇F is L-Lip +technical conditions	F μ-stgly cvx, ∇F L-Lip +technical conditions
GD	$\frac{C}{k}$	$C\exp\big(-\frac{\mu k}{L}\big)$
SGD	$\frac{C\log(k)}{\sqrt{k}}$	$\frac{C}{k}$

Table 2: Error obtained after k iterations

Warning: The value of C may change in each appearance!

Difference runtime/number of iterations

Warning: Runtime is DIFFERENT from the number of iterations!

Suppose computing one gradient takes 1 second.

- Each SGD step involves computing 1 gradient.
  Therefore, k iterations are completed in $t = k$ seconds.
- Each GD step involves computing n gradients.
  Therefore, k iterations are completed in $t = nk$ seconds.

Rates of GD and SGD

We now rephrase the earlier table in terms of runtime:

$t = $ runtime	F cvx, ∇F is L-Lip +technical conditions	F μ-stgly cvx, ∇F L-Lip +technical conditions
GD	$\frac{Cn}{t}$	$C\exp\big(-\frac{\mu t}{Ln}\big)$
SGD	$\frac{C\log(t)}{\sqrt{t}}$	$\frac{C}{t}$

Table 3: Error obtained after t units of time

Rates of SGD for strongly convex objectives

Reminder: strongly convex functions

Let $\varphi : \Theta \to \mathbb{R}$ where Θ is convex and $\mu > 0$, and denote by $\|\cdot\|$ the Euclidean norm.

Definition

We say that f is μ-strongly convex if, for any $t \in [0,1]$ and any $x, y \in \Theta$, we have

$$f\big(tx + (1-t)y\big) \leq tf(x) + (1-t)f(y) - \frac{\mu}{2}t(1-t)\big\|x-y\big\|^2.$$

Proposition

If $f : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex and differentiable, then we have for any $a, b \in \mathbb{R}^d$:

$$f(b) \geq f(a) + \big\langle \nabla f(a), b-a \big\rangle + \frac{\mu}{2}\|b-a\|^2.$$

Rates for strongly convex objectives

Theorem (Rates of SGD for strongly convex functions)

Suppose that:

1. $\theta \mapsto f(\theta, \xi)$ is convex and differentiable for all ξ,
2. F is μ-strongly convex and its gradient is L-Lipschitz,
3. $\exists C > 0, \forall \theta \in \mathbb{R}^d : \mathbb{E}\left(\left\|\nabla_\theta f(\theta, \xi)\right\|^2\right) \leq C$,
4. $\theta^* \in \arg\min F$ is attained,
5. For any $k \in \mathbb{N}$: $\gamma_k = \frac{1}{\mu(k+1)}$.

The iterates $(\theta_k)_k$ of the SGD algorithm satisfy, for $k \geq 2$

$$\mathbb{E}\left[\left\|\theta_k - \theta^*\right\|^2\right] \leq \frac{C}{k\mu^2},$$

$$\mathbb{E}\left[F\left(\theta_k\right) - F\left(\theta^*\right)\right] \leq \frac{CL}{2k\mu^2}.$$

Proof: See lecture notes.

A slightly modified version of the theorem

In this lecture, we will prove a slightly modified version of the previous theorem.

Theorem (Rates of SGD for strongly convex functions)

Under the same assumptions as above and with

$$\gamma_k = \frac{2}{(k+1)\mu},$$

the iterates $(\theta_k)_k$ of the SGD algorithm satisfy, for $k \geq 2$

$$\mathbb{E}\left[\|\theta_k - \theta^*\|^2\right] \leq \frac{4C}{k\mu^2},$$

$$\mathbb{E}\left[F(\theta_k) - F(\theta^*)\right] \leq \frac{2CL}{k\mu^2}.$$

This theorem is slightly weaker than the previous one (although the rates differ only by a constant factor), but its proof is simpler.

Remarks on the theorem

1. Compared to the purely convex case, this theorem requires strong convexity of F (not of each of the $\theta \mapsto f(\theta, \xi)$ for $\xi \in \Xi$), and that ∇F be L-Lipschitz.

2. The step-size sequence now decays as $\gamma_k = O(\frac{1}{\mu k})$.
   (In the convex case, the decay was $\frac{1}{\sqrt{k}}$).

3. Convergence rate: $O(\frac{C}{\mu^2 k})$. (In the convex case it was $O(\frac{\log(k)}{\sqrt{k}})$).

4. We obtain guarantees on both $\mathbb{E}\left[F(\theta_k) - F(\theta^*)\right]$ and $\|\theta_k - \theta^*\|^2$ due to the strong convexity of F and Lipschitzness of ∇F.

5. The initial value $\|\theta_0 - \theta^*\|^2$ does not appear! Why?

Where is $\|\theta_0 - \theta^*\|^2$ hidden?

The term $\|\theta_0 - \theta^*\|^2$ is hidden in the term C/μ^2, as we *must* have $\|\theta_0 - \theta^*\|^2 \leq 16C/\mu^2$.

This comes from the fact that F is μ-strongly convex, but also $\sqrt{C}$-Lipschitz due to the assumption $\forall \theta \in \mathbb{R}^d : \mathbb{E}\left(\left\|\nabla_\theta f(\theta, \xi)\right\|^2\right) \leq C$.

Proposition

Let $F : \Theta \to \mathbb{R}$ be $\sqrt{C}$-Lipschitz and μ-strongly convex. Then the diameter of Θ must satisfy

$$\begin{aligned} \operatorname{diam}(\Theta) &= \max\{\|x - y\| : x, y \in \Theta\} \\ &\leq 4\sqrt{C}/\mu. \end{aligned}$$

Proof

Suppose for a contradiction that $\exists x, y \in \Theta$ such that $\|x - y\| > 4\sqrt{C}/\mu$. Then by strong convexity of F, we would have

$$\begin{aligned} F(x) - F(y) &\geq \langle \nabla F(y), x - y \rangle + \frac{\mu}{2}\|x - y\|^2 \\ &> -\big\|\nabla F(y)\big\|\big\|x - y\big\| + \frac{\mu}{2}\|x - y\|\frac{4\sqrt{C}}{\mu} \\ &\geq -\sqrt{C}\|x - y\| + 2\sqrt{C}\|x - y\| \\ &= \sqrt{C}\|x - y\|, \end{aligned}$$

which contradicts the $\sqrt{C}$-Lipschitzness of F.

Useful results for the proof

In the proof, we will use the following lemmas.

Lemma

Under the assumptions of the theorem, it holds that

$$\mathbb{E}\left[\nabla f(\theta_k, \xi_{k+1}) \,\middle|\, \theta_k\right] = \nabla F(\theta_k).$$

Lemma

For any $k \in \mathbb{N}$ it holds that

$$\left\langle \theta^* - \theta_k, \nabla F(\theta_k) \right\rangle \leq -\frac{\mu}{2} \left\| \theta_k - \theta^* \right\|^2.$$

Mini-batch SGD

Mini-batch SGD

Mini-batch SGD is a generalization of SGD and GD. Consider

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta), \qquad \text{where } n \text{ is "large".}$$

Idea

Fix an integer m (where $1 \leq m \leq n$), called the mini-batch size.

At each iteration, draw a *subset* $v_k \subseteq \{1, \ldots, n\}$ of size m, u.a.r.

The random subset v_k is called the mini-batch.

Update θ_k by averaging the gradients in the mini-batch v_k

$$\theta_{k+1} = \theta_k - \frac{\gamma_k}{m} \sum_{i \in v_k} \nabla f_i(\theta_k).$$

Algorithm

For some chosen mini-batch size $m \in \{1, \ldots, n\}$, the algorithm is

Mini-batch SGD

Start from $\theta_0 \in \mathbb{R}^d$.

Until termination condition, iterate:

Draw a subset $v_k \subseteq \{1, \ldots, n\}$ of size m, u.a.r.

Update $\theta_{k+1} = \theta_k - \dfrac{\gamma_k}{m} \displaystyle\sum_{i \in v_k} \nabla f_i(\theta_k)$.

Remarks: Minibatch SGD interpolates between GD and SGD.

- GD is equivalent to mini-batch SGD with $m = n$.
- SGD is equivalent to mini-batch SGD with $m = 1$.

Influence of the parameter m

Each stochastic gradient is now given by

$$g_k = \frac{1}{m} \underbrace{\sum_{i \in v_k} \nabla f_i(\theta_k)}_{m \text{ gradients}} .$$

Influence of m

When increasing m, each stochastic gradient g_k

- becomes a more accurate estimator of $\nabla F(\theta_k)$,
- but takes more time to compute.

The parameter m controls a trade-off between precision and speed!

This property, as well as the convergence rate of mini-batch SGD, will be studied in the exercise session.

A variant: Epoch-based mini-batch SGD

A widely used variant operates with a modified mini-batch selection*.

1. **Shuffle the set of functions $f_1, \ldots, f_n$ into $f_{\sigma(1)}, \ldots, f_{\sigma(n)}$.**
   (Here, σ is a random permutation of the set $\{1, \ldots, n\}$).

2. **Select the mini-batches as consecutive blocks of size m.**

   1st mini-batch: $f_{\sigma(1)}, \ldots, f_{\sigma(m)}$ (first block)
   2nd mini-batch: $f_{\sigma(m+1)}, \ldots, f_{\sigma(2m)}$ (second block)
   $\vdots$
   Last mini-batch: $f_{\sigma(n-m+1)}, \ldots, f_{\sigma(n)}$ (last block).

3. **Reshuffle the set of functions and repeat.**

*For simplicity of exposition, we assume that n/m is an integer here.

Algorithm

Formally, the algorithm is given as follows.

Epoch-based mini-batch SGD

1. Start from $\theta_0 \in \mathbb{R}^d$
2. Until termination, iterate:

   Draw a random permutation σ of $\{1, \dots, n\}$.

   For $\ell = 1, \dots, n/m$:

$$\theta_{k+1} = \theta_k - \frac{\gamma_k}{m} \left(\nabla f_{\sigma((\ell-1)m+1)}(\theta_k) + \cdots + \nabla f_{\sigma(\ell m)}(\theta_k) \right).$$

A single pass through the entire set of indices is called an epoch.

Within an epoch, all batches are disjoint (i.e., no index will be used twice in a row), which is not guaranteed in the initial version of mini-batch SGD.

Proximal Stochastic Gradient Descent

Proximal SGD

We now consider the *penalized* problem

$$\min_{\theta \in \mathbb{R}^d} \mathbb{E}(f(\theta, \xi)) + g(\theta)$$

and we assume that g has a simple, easily computable proximal operator:

$$\text{prox}_g(x) = \arg\min_{y \in \mathbb{R}^d} \; g(y) + \frac{1}{2}\|x - y\|^2$$

For instance,

- $g(\theta) = \|\theta\|_1$ (LASSO penalization),
- $g(\theta) = \|\theta\|_2^2$ (Ridge penalization),
- $g(\theta) = \begin{cases} 0 & \text{if } \theta \in \Theta \\ +\infty & \text{otherwise} \end{cases}$ (characteristic function), equivalent to projecting onto Θ.

Important example: Characteristic function

Important special case: Suppose that g is the characteristic function χ_Θ of a closed convex set $\Theta \subseteq \mathbb{R}^d$:

$$g(\theta) = \chi_\Theta(\Theta) = \begin{cases} 0 & \text{if } \theta \in \Theta \\ +\infty & \text{otherwise.} \end{cases}$$

Then the proximal operator simplifies as follows

$$\begin{aligned} \forall x \in \mathbb{R}^d : \qquad \operatorname{prox}_g(x) &= \arg\min_{y \in \mathbb{R}^d} \; g(y) + \frac{1}{2}\|x-y\|^2 \\ &= \arg\min_{y \in \Theta} \; 0 + \frac{1}{2}\|x-y\|^2 \\ &= \arg\min_{y \in \Theta} \; \|x-y\|^2 \\ &\stackrel{\text{def}}{=} \operatorname{Proj}_\Theta(x). \end{aligned}$$

This is called the projection of x onto Θ (closest point to x in Θ).

Proximal SGD

Proximal SGD

1. Start from $\theta_0 \in \Theta$
2. Until termination condition, iterate:

$$\text{Draw } \xi_{k+1} \sim \mathbb{P}_\xi, \text{ independent of } (\xi_1, \ldots, \xi_k)$$

$$\theta_{k+1} = \text{prox}_{\gamma_k g}(\theta_k - \gamma_k \nabla_\theta f(\theta_k, \xi_{k+1}))$$

$$(= \text{Proj}_\Theta(\theta_k - \gamma_k \nabla_\theta f(\theta_k, \xi_{k+1})) \quad \text{if } g = \chi_\Theta)$$

where $\gamma_k > 0, \forall k \in \mathbb{N}$.

Reminder:

$$\text{prox}_g(x) = \arg\min_y g(y) + \frac{1}{2}\|x - y\|^2$$

Proper functions

We will assume that g is *proper*.

Definition

A function $g : \mathbb{R}^d \to [-\infty, +\infty]$ is said to be proper if

1. g never takes the value $-\infty$ $(\forall x \in \mathbb{R}^d : g(x) > -\infty)$
2. g is not identically equal to $+\infty$ $(\exists x_0 \in \mathbb{R}^d : g(x_0) < +\infty)$.

We denote the *domain* of g by

$$\mathrm{dom}(g) = \{x \in \mathbb{R}^d : g(x) < +\infty\}.$$

Rates for Proximal SGD

Theorem (Rates for Proximal SGD)

Suppose that:

1. $\theta \mapsto f(\theta, \xi)$ is convex and differentiable for all ξ,
2. g is proper and convex,
3. F is μ-strongly convex and has an L-Lipschitz gradient,
4. $\exists C > 0, \forall \theta \in \operatorname{dom} g : \mathbb{E}\left(\left\|\nabla_\theta f(\theta, \xi) - \nabla F(\theta_k)\right\|^2\right) \leq C$
5. there exists $x^* \in \arg\min F + g$,
6. $\forall k \in \mathbb{N} : \gamma_k = \frac{1}{\mu(k+1)}$.

The iterates $(\theta_k)_k$ of the proximal SGD satisfy, for $k \geq 2$

$$\mathbb{E}\left[\left\|\theta_k - \theta^*\right\|^2\right] \leq \frac{8C}{\mu^2 k}.$$

Remarks on the theorem

1. Compared to the previous SGD algorithm, the condition $\mathbb{E}\left(\|\nabla_\theta f(\theta,\xi)\|^2\right) \leq C$ is replaced with the weaker condition $\mathbb{E}\left(\|\nabla_\theta f(\theta,\xi) - \nabla F(\theta_k)\|^2\right) \leq C$.

2. Formally, we can apply this theorem with $g = 0$. We then recover the rate of *non*-proximal SGD algorithm for strongly convex functions, with the *weaker* condition $\mathbb{E}\left(\|\nabla_\theta f(\theta,\xi) - \nabla F(\theta_k)\|^2\right) \leq C$.

Variance Reduction

Setting

Setting: We come back to minimizing

$$\min_{x\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} f_i(x) \overset{\text{def}}{=} \min_{x\in\mathbb{R}^d} F(x)$$

where F is μ-strongly convex and ∇F is L-Lipschitz.

- GD attains the rate $O\big(\exp(-k\mu/L)\big)$ at step k, but computes n gradients at each step.
- SGD attains the rate $O\big(\frac{1}{\mu k}\big)$ at step k, but computes 1 gradient at each step.

Goal: Can we obtain exponential rates while keeping a low per-iteration cost?

Intuition

"Mix" GD and SGD to combine their advantages!

1. **Every T steps:** Compute one *full gradient* $\nabla F(w_k)$
2. **In between:** Use a *corrected SGD*:
   - $i_t \sim \text{Unif}(\{1, \dots, n\})$
   - $x_{t+1} = x_t - \gamma g_t$
   - where $g_t = \underbrace{\nabla f_{i_t}(x_t)}_{\text{usual term}} - \big(\underbrace{\nabla f_{i_t}(w_k) - \nabla F(w_k)}_{\text{correction term}} \big)$

Key: g_t is correctly centered at $\nabla F(x_t)$, but has smaller variance!

$$\begin{aligned} \mathbb{E}_{i_t}\left[g_t\right] &= \mathbb{E}_{i_t} \nabla f_{i_t}(x_t) - \mathbb{E}_{i_t}\left[\nabla f_{i_t}(w_k) - \nabla F(w_k)\right] \\ &= \nabla F(x_t) \; - \; \big(\cancel{\nabla F(w_k)} \; - \; \cancel{\nabla F(w_k)} \big) \end{aligned}$$

where $\mathbb{E}_{i_t} = \mathbb{E}\left[\, \cdot \,|\, x_t, w_k\right]$. (Variance: cf Lemmas below).

SVRG used in practice

Initial value: w_0

Outer loop: At step k:

$x_0 = w_k$

Compute $\nabla f_i(w_k)$ for all $i = 1 \dots, n$

Inner loop: For $t = 0, \dots, T-1$:

Draw $i_t \sim \text{Unif}(\{1, \dots, n\})$ independent of the past

$g_t = \nabla f_{i_t}(x_t) - \big(\nabla f_{i_t}(w_k) - \nabla F(w_k)\big)$

$x_{t+1} = x_t - \gamma g_t$

$w_k = \frac{1}{T} \sum_{t=1}^{T} x_t$

Stochastic Variance-Reduced Gradient (SVRG)

SVRG admits another formulation, easier to analyze mathematically

Stochastic Variance-Reduced Gradient (SVRG)

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $w_0 = x_0$.
2. Until termination condition, iterate

$$i_{k+1} \sim \text{Unif}(\{1, \dots, n\})$$

$$g_{k+1} = \nabla F(w_k) + \nabla f_{i_{k+1}}(x_k) - \nabla f_{i_{k+1}}(w_k)$$

$$x_{k+1} = x_k - \gamma g_{k+1}$$

$$w_{k+1} = \begin{cases} x_k & \text{with probability } p \\ w_k & \text{with probability } 1 - p. \end{cases}$$

If we updated $w_{k+1} \leftarrow x_k$, compute and store $\nabla F(w_{k+1})$

Equivalent to updating w_k approximately every $T = 1/p$ steps.

In this class, we focus on this formulation.

Convergence rate of SVRG

Theorem (Convergence rate for SVRG)

Suppose that

1. $\forall i \in \{1, \dots, n\}$, f_i is cvx, differentiable, and ∇f_i is L Lipschitz
2. F is μ-strongly convex.
3. $\gamma \leq \frac{1}{15L}$.

Let $x^* = \arg\min F$. The iterates of SVRG satisfy

$$\mathbb{E}\left[\|x_k - x^*\|^2\right] \leq c^k \Delta_0,$$

$$\text{where} \quad \begin{cases} c = \max(1 - \gamma\mu, 1 - p/2) \\ \Delta_0 = \|x_0 - x^*\|^2 + \frac{24\gamma^2 L}{p}\Big(F(x_0) - F(x^*)\Big). \end{cases}$$

Moreover, the expected cost of an iteration is $2 + pn$ stochastic gradients.

Comments on the Theorem

1. Now the rate has been improved to

$$\Delta_0 \big[\max(1-\gamma\mu, 1-p/2) \big]^k \leq \Delta_0 \exp\left(-k \min\left(\gamma\mu, \frac{p}{2} \right) \right).$$

2. Rate exponentially fast in k (for SGD with strongly convex functions, it was $O(\frac{1}{\mu^2 k})$), but the initial condition Δ_0 depends on the function F to optimize.

3. We can now take γ as a constant (for the classical SGD with strongly convex functions, it was $\gamma_k = \frac{\gamma^*}{k+1}$).

   $\Longrightarrow$ Due to the variance reduction, we can take larger steps!

4. This is due to the fact that we *store* $\nabla F(w_k)$.

Proof: Technical lemmas

Lemma (Taylor-Lagrange)

Let f be convex, with L-Lipschitz gradient, then

$$\left|f(u) - f(v) - \langle \nabla f(v), u - v\rangle\right| \leq \frac{L}{2}\|u - v\|^2.$$

Lemma

Let f be convex with an L-Lipschitz gradient. For all a and b,

$$f(a) \geq f(b) + \langle \nabla f(b), a - b\rangle + \frac{1}{2L}\|\nabla f(a) - \nabla f(b)\|^2.$$

(**Just technical**: only needed in the proof of the previous theorem, especially through the following corollary.)

Technical result

Corollary (Toward variance reduction)

Suppose

1. $F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$
2. $\forall i : f_i$ is convex, and ∇f_i is L-Lipschitz
3. $\exists x^* \in \mathbb{R}^d : \nabla F(x^*) = 0$.
4. $i_k \sim \text{Unif}(\{1, \ldots, n\})$

Then, for any $y \in \mathbb{R}^d$, we have

$$\mathbb{E}\|\nabla f_{i_k}(x^*) - \nabla f_{i_k}(y)\|^2 = \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(x^*) - \nabla f_i(y)\|^2$$
$$\leq 2L\big[F(y) - F(x^*)\big].$$

Corollary

Corollary (Variance reduction lemma)

Suppose

1. $F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$
2. $\forall i : f_i$ is convex, and ∇f_i is L-Lipschitz
3. $\exists x^* \in \mathbb{R}^d : \nabla F(x^*) = 0.$
4. $i_k \sim \text{Unif}(\{1, \ldots, n\})$

Then

$$\mathbb{E}\Big[\|g_{k+1}\|^2 \Big| x_k, w_k\Big] \leq 6L\Big(F(x_k) - F(x^*)\Big) + 12L\Big(F(w_k) - F(x^*)\Big)$$

$$\longrightarrow 0 \quad \text{as } w_k, x_k \to x^*.$$

Variance reduction! Since $\mathbb{V}(g_{k+1}|w_k, x_k) \leq \mathbb{E}(\|g_{k+1}\|^2 \big| x_k, w_k)$, this implies $\mathbb{V}(g_{k+1}|w_k, x_k) \underset{w_k, x_k \to x^*}{\longrightarrow} 0$. We can then take larger step sizes.

Acceleration methods

Momentum algorithm

Momentum algorithm

Inputs: $\alpha > 0,\ \beta \in [0, 1],\ \epsilon > 0$

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $m_0 = 0$.
2. Until termination condition, iterate:

   Generate $\xi_{k+1} \sim \mathbb{P}_\xi$ independent of the past

   $$m_{k+1} = \beta m_k + (1 - \beta) \nabla f(x_k, \xi_{k+1})$$

   $$\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta^{k+1}} \text{ (optional, we can also keep } \hat{m}_{k+1} = m_{k+1})$$

   $$x_{k+1} = x_k - \alpha_k \hat{m}_{k+1}$$

Momentum smooths out zigzag trajectories of GD or SGD.

It can also escape from local minima (not always).

Momentum: illustration

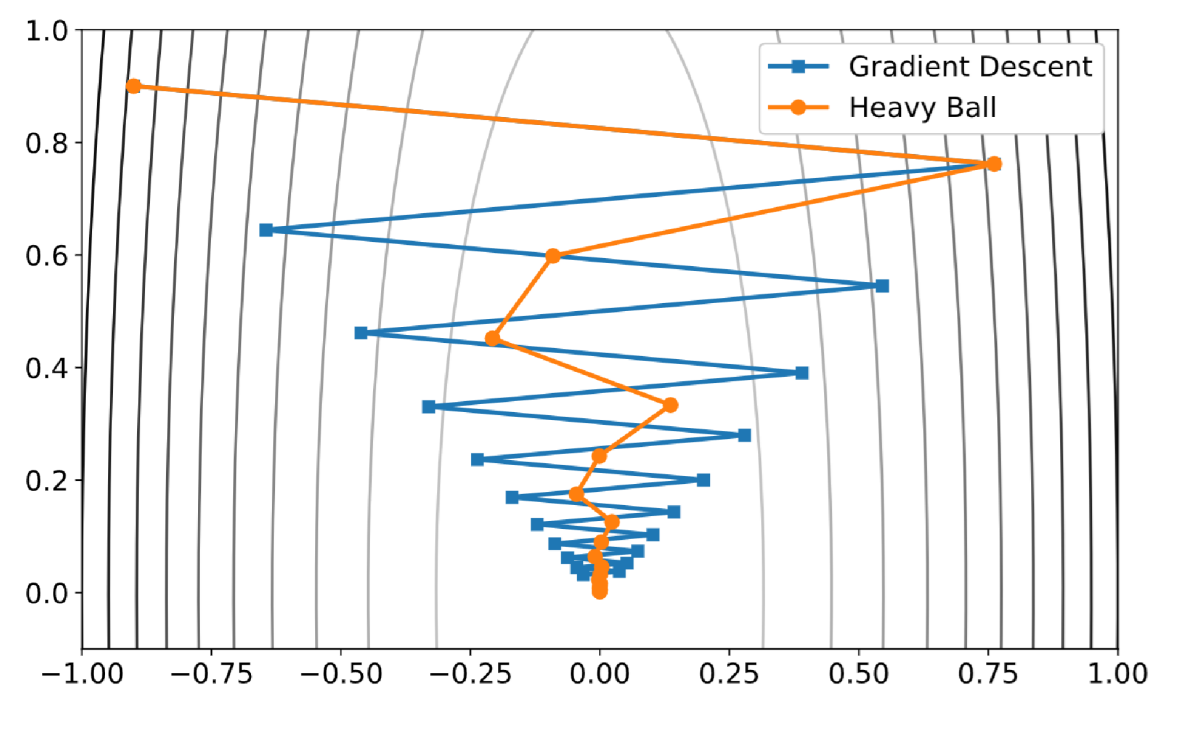

Figure 1: Gradient descent versus heavy ball (momentum) method.

An improvement: Nesterov Accelerated Gradient (NAG)

Drawback of momentum: it can overshoot, especially when approaching convergence.

Alternative: Nesterov Accelerated Gradient (NAG) evaluates the gradient at a *look-ahead point* rather than the current point:

$$m_t = \beta m_{t-1} + \nabla f(\underbrace{x_{t-1} - \alpha m_{t-1}}_{\text{look-ahead point}})$$
$$x_t = x_{t-1} - \alpha m_t.$$

Here, $x_{t-1} - \alpha m_{t-1}$ is an estimator of the position of the next iterate, and m_t stands for the velocity (or momentum) at time t.

NAG pseudo-code

Nesterov Accelerated Gradient (NAG) Algorithm

Inputs: $\alpha > 0,\ \beta \in [0, 1],\ \epsilon > 0$

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $m_0 = 0$.
2. Until termination condition, iterate:

   Generate $\xi_{k+1} \sim \mathbb{P}_\xi$ independent of the past

   $\tilde{x}_k = x_k - \beta m_k$ (look-ahead point)

   $m_{k+1} = \beta m_k + \nabla f\left(\tilde{x}_k, \xi_{k+1}\right)$

   $x_{k+1} = x_k - \alpha m_{k+1}$

Some remarks on Nesterov acceleration

Interpretation of the update

$$m_t = \beta m_{t-1} + \nabla f(x_{t-1} - \beta m_{t-1})$$

If we are supposed to overshoot at the next step, then $\nabla f(x_{t-1} - \beta m_{t-1})$ will point in the opposite direction to m_{t-1}.

The update therefore *decreases the velocity* before we overshoot. This happen especially when approaching convergence.

We then update $x_t = x_{t-1} - \alpha m_t$ with a lower velocity m_t. We slow down to prevent unwanted oscillations.

Adaptive step-sizes

Motivation

Suppose $F : \mathbb{R}^d \longrightarrow \mathbb{R}$, is "more convex" along some directions.

Example: $F(x, y) = x^2 + 1000y^2$.

The learning rate γ_k should vary along different directions as well.

- Along directions j where $\partial_j F$ is "large": reduce the learning rate (avoid huge jumps)
- Along directions j where $\partial_j F$ is "small": increase the learning rate (accelerate convergence)

But F is unknown: We need a method that *adapts* the learning rate as we go.

Notation

For any three vectors $a, a', b \in \mathbb{R}^d$, we define

$$a \odot b = \begin{pmatrix} a_1 b_1 \\ \vdots \\ a_d b_d \end{pmatrix}, \qquad a \oslash b \text{ or } \frac{a}{b} = \begin{pmatrix} a_1/b_1 \\ \vdots \\ a_d/b_d \end{pmatrix},$$

$$\|a\|_b^2 = \sum_{i=1}^{n} a_i^2 b_i, \qquad \langle a, a' \rangle_b = \sum_{i=1}^{d} a_i a_i' b_i$$

$$a^{\odot 2} = a \odot a,$$

if these quantities exist.

Adagrad (Adaptive Gradient Algorithm)

Adagrad

1. Start from an initial guess $x_0 \in \mathbb{R}^d$.
2. Until termination condition, iterate:

Generate ξ_{k+1} independent of the past

$$g_{k+1} = \begin{pmatrix} g_{k+1}(1) \\ \vdots \\ g_{k+1}(d) \end{pmatrix} = \nabla f_x\left(x_k, \xi_{k+1}\right)$$

$$\gamma_{k+1}(j) = \frac{\alpha}{\sqrt{\sum_{s=0}^{k} g_{s+1}(j)^2}}, \ \forall j = 1, \ldots, d$$

$$x_{k+1} = x_k - \gamma_{k+1} \odot g_{k+1}$$

Here, we recall that $\gamma_k \odot g_k = \begin{pmatrix} \gamma_k(1) g_k(1) \\ \vdots \\ \gamma_k(d) g_k(d) \end{pmatrix}$

Adagrad's rate of convergence

Theorem

Suppose that

1. $f(\cdot, \xi)$ is convex for all ξ
2. There exists $x^* \in \arg\min F$
3. $\forall k \geq 0, \forall i \in \{1, \ldots, d\} : |x_{k,i} - x_i^*| \leq D$
4. For all $x \in \mathbb{R}^d$, $\forall \xi \in \Xi : \|\nabla f(x, \xi)\| \leq G$.

Then the iterates of Adagrad satisfy

$$\mathbb{E}\left[F\left(\bar{x}_K\right) - F\left(x^*\right)\right] \leq \frac{dG}{\sqrt{K}}\left(\frac{D^2}{\alpha} + 2\alpha\right)$$

where $\bar{x}_K = \frac{1}{K}\sum_{k=0}^{K-1} x_k$.

Comments on the theorem

Remarks

1. Adagrad adapts the learning rate γ_k along every direction. Namely, $\gamma_k = (\gamma_k(1), \dots \gamma_k(d))^\top$ is a *vector*, no longer a scalar.

2. For each coordinate j, $\gamma_{k+1}(j) = \alpha \left(\sum_{s=0}^{k} g_{s+1}(j)^2 \right)^{-1/2}$, is *smaller* if the partial derivatives $|g_s(j)|$ are *larger* along coordinate j.

3. However, $\gamma_k(j)$ *always decreases* as $k \to \infty$, which can cause computational issues and prevent convergence in practice.

4. This is why other algorithms (e.g. RMSprop or Adam) were introduced. They propose updates that gradually *forget* older gradients $g_s(j)^2$.

An alternative: RMSProp

RMSprop stands for Root Mean Squared Propagation.

It is an improvement over Adagrad.

RMSProp algorithm

Inputs: $\alpha > 0, \beta \in [0,1],\ \epsilon > 0$

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $v_0 = 0$.
2. Until termination condition, iterate:

   Generate $\xi_{k+1} \sim \mathbb{P}_\xi$ independent of the past

   $$v_{k+1} = \beta v_k + (1-\beta)\, \nabla f\left(x_k, \xi_{k+1}\right)^{\odot 2}$$

   $$\hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta^{k+1}} \text{ (optional, we can also keep } \hat{v}_{k+1} = v_{k+1}).$$

   $$x_{k+1} = x_k - \frac{\alpha}{\epsilon + \sqrt{\hat{v}_{k+1}}} \odot \nabla f(x_k, \xi_{k+1})$$

RMSProp vs. Adagrad

1. **Adagrad - Adaptive Step Size:**

$$\gamma_{k+1} = \frac{\alpha}{\sqrt{\sum_{s=0}^{k} g_{s+1}^{\odot 2}}} \in \mathbb{R}^d, \quad (g_k = \nabla f(x_k, \xi_{k+1})).$$

Advantage: Tailors γ_k along each dimension; lower $\gamma_k(j)$ for steeper directions.

Drawback: Accumulates all past gradients, leading to perpetually decreasing γ_k and slower updates.

2. **RMSProp - Forgetting Rule:**

$$\gamma_{k+1} \propto \frac{\alpha}{\sqrt{\sum_{s=0}^{k} \beta^{k-s} g_{s+1}^{\odot 2}}},$$

Emphasizes recent gradients, and forgets older ones. Adapts dynamically to the *local* geometry of the function.

Adam

1. Adam stands for stochastic gradient with ADAptive Moment estimation.
2. Frequently used for training Deep Neural Networks.
3. Also an adaptive algorithm, but with a forgetting rule that discounts older gradients.

Adam

Adam algorithm

Inputs: $\alpha > 0,\ \beta_1, \beta_2 \in [0, 1],\ \epsilon > 0$

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $m_0 = v_0 = 0$.
2. Until termination condition, iterate:

Generate $\xi_{k+1} \sim \mathbb{P}_\xi$ independent of the past

$$m_{k+1} = \beta_1 m_k + (1 - \beta_1) \nabla f(x_k, \xi_{k+1})$$

$$\hat{m}_{k+1} = \frac{m_{k+1}}{1 - \beta_1^{k+1}}$$

$$v_{k+1} = \beta_2 v_k + (1 - \beta_2) \nabla f(x_k, \xi_{k+1})^2$$

$$\hat{v}_{k+1} = \frac{v_{k+1}}{1 - \beta_2^{k+1}}$$

$$x_{k+1} = x_k - \frac{\alpha_k}{\epsilon + \sqrt{\hat{v}_{k+1}}} \hat{m}_{k+1}$$

Recall: Momentum algorithm

Momentum algorithm

Inputs: $\alpha > 0,\ \beta \in [0, 1],\ \epsilon > 0$

1. Start from an initial guess $x_0 \in \mathbb{R}^d$, and let $m_0 = 0$.
2. Until termination condition, iterate:

   Generate $\xi_{k+1} \sim \mathbb{P}_\xi$ independent of the past

   $$m_{k+1} = \beta m_k + (1 - \beta) \nabla f(x_k, \xi_{k+1})$$

   $$\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta^{k+1}} \text{ (optional, we can also keep } \hat{m}_{k+1} = m_{k+1})$$

   $$x_{k+1} = x_k - \alpha \hat{m}_{k+1}$$

Momentum smooths out zigzag trajectories of GD or SGD.

It can also escape from local minima (not always).

Adam

Adam combines the advantages of Momentum and RMSProp.

1. Smooths out possible oscillations (zigzags) coming from the discreteness of a GD or SGD trajectory.

2. Can escape from a local minimum if it reaches it with "sufficient momentum".

3. Uses an adaptive step size $\gamma_k \in \mathbb{R}^d$ (smaller $\gamma_k(j)$ along directions j where the function is more convex).

4. Step size is adapted to the *local* geometry of the function.

5. In practice, one generally takes $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Concluding remarks on convergence rates

Convergence rates allow comparing algorithms, even before running them.

Example: For μ-strongly convex functions, we know that

$$F(x_k^{GD}) - F(x^*) \leq C_1 \exp\left(-\frac{k\mu}{L}\right)$$
$$\mathbb{E}\Big(F(x_k^{SGD}) - F(x^*)\Big) \leq \frac{C_2}{k\mu^2}.$$

Suppose we want to solve this problem up to $\epsilon > 0$, i.e. find a point x_k such that $F(x_k) - F(x^*) \leq \epsilon$ or $\mathbb{E}[F(x_k) - F(x^*)] \leq \epsilon$.

For GD: We need $C_1 \exp\left(-\frac{k\mu}{L}\right) \leq \epsilon \iff k \geq \frac{L}{\mu}\log(C_1/\epsilon)$.

For SGD: We need $\frac{C_2}{k\mu^2} \leq \epsilon \iff k \geq \frac{C_2}{\mu^2\epsilon}$.

Concluding remarks on convergence rates

What about runtime? Suppose we come back to

$$F(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x)$$

How long does it take to obtain an ϵ-solution?

Suppose computing 1 gradient takes 1 second.

For GD: We need to wait $nk = \frac{nL}{\mu} \log(C_1/\epsilon)$ seconds.

For SGD: We need to wait $k = \frac{C_2}{\mu^2 \epsilon}$ seconds.

$\implies$ If $\frac{C_2}{\mu^2 \epsilon} < \frac{nL}{\mu} \log(C_1/\epsilon)$, we should choose SGD.

This case arises when ϵ is not too small or n is very large.