Cours 3

Stochastic optimization for large scale problems

Elisa Riccietti and Théo Mary

The reference for this part is *Optimization Methods for Large-Scale Machine Learning*, by Leon Bottou, Frank E. Curtis, Jorge Nocedal, SIAM Review, Vol. 60, No. 2, pp. 223-311.

LIP-ENS Lyon

The context

The problem: large scale optimization problems

$$\min_{x \in \mathbb{R}^n} f(x), \quad \text{with } n \text{ large}$$

Typically

$$f(x) = \sum_{i=1}^{m} f_i(x) \quad m \text{ large}$$

with $f, f_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \ldots, m$

Example 1: Image restoration

$$\min_x \|Ax - b\|^2 + \lambda \|Dx\|^2$$

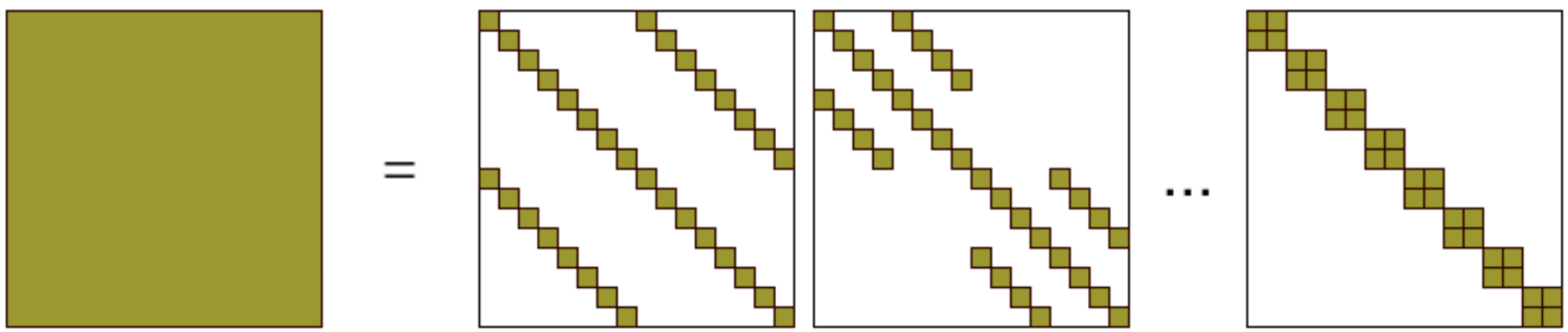
Original

Noisy image

Denoised image

Example 2: Matrix factorization

$$\min_{X_1,\ldots,X_L} \|A - X_1 \ldots X_L\|_F^2$$



Example 3: Neural networks training

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \ell(NN(\theta; x_i), y^i)$$

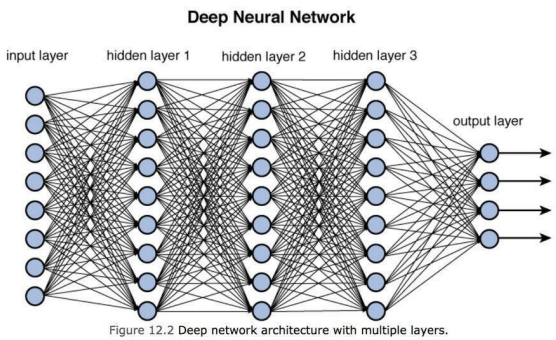
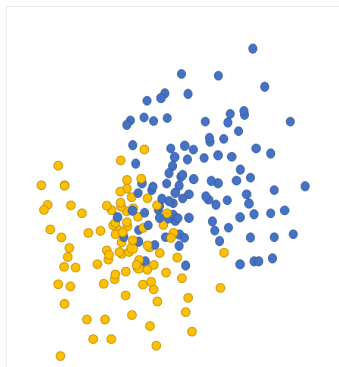
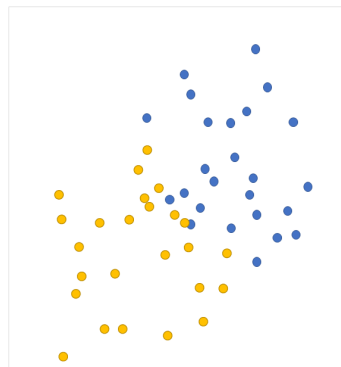


Figure 12.2 Deep network architecture with multiple layers.


TRAINING DATA


TEST DATA

Outline

Classical optimization methods

Iterative method

- Given a starting guess $x_0 \in \mathbb{R}^n$
- Define a descent direction $p_k \in \mathbb{R}^n$: $\nabla f(x_k)^T p_k < 0$
- Build a sequence $\{x_k\}_{k \in \mathbb{N}}$ such that $x_{k+1} = x_k + \alpha_k p_k$, where α_k is called the step length or the learning rate.
- Goal: find a stationary point x^* of f: $\nabla f(x^*) = 0$

Classical optimization methods

$$x_{k+1} = x_k + \alpha_k p_k$$

If α_k is small enough, from Taylor's theorem:

$$f(x_{k+1}) \sim f(x_k) + \alpha_k \nabla f(x_k)^T p_k < f(x_k).$$

This ensures x^* not to be a maximum point, but in unfortunate cases it may be a saddle point, there is usually no guarantee of convergence to a minimum.

First and second order methods

- **First order** just use first order derivatives (gradient)
- **Second order** also use second order derivatives (Jacobian, Hessian matrices)

$$\nabla f(\mathrm{x}) = \begin{pmatrix} \frac{\partial f(\mathrm{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathrm{x})}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n$$

$$\nabla^2 f(\mathrm{x}) = \begin{pmatrix} \frac{\partial^2 f(\mathrm{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(\mathrm{x})}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(\mathrm{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\mathrm{x})}{\partial x_n \partial x_n} \end{pmatrix} = \begin{pmatrix} \left(\nabla \frac{\partial f(\mathrm{x})}{\partial x_1}\right)^T \\ \vdots \\ \left(\nabla \frac{\partial f(\mathrm{x})}{\partial x_n}\right)^T \end{pmatrix} \in \mathbb{R}^{n \times n}$$

If $f \in C^2(\mathrm{x})$ then $\nabla^2 f(x)$ is a symmetric matrix.

First order: gradient method

$$p_k = -\nabla f(x_k).$$

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k).$$

- ☺ quite cheap: it requires just the computation of the gradient of f at each iteration
- ☹ slow convergence: it requires a large number of iterations to reach a stationary point

Gradient method

Given x_0, $\mathrm{toll} > 0$, set $k = 0$.

While $\|\nabla f(x_k)\| > \mathrm{toll}$

1. Compute the search direction $p_k = -\nabla f(x_k)$.
2. Choose α_k.
3. Set $x_{k+1} = x_k + \alpha_k p_k$
4. Set $k = k + 1$

Second order: Newton's method

$$p_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k).$$

- ☺ fast convergence: usually requires few iterations to reach a stationary point
- ☹ expensive: it requires the computation of first and second order derivatives and the solution of a linear system at each iteration $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$.

Newton's method

Given x_0, $\text{toll} > 0$, set $k = 0$.

While $\|\nabla f(x_k)\| > \text{toll}$

1. Compute the search direction by solving $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$.
2. Choose α_k.
3. Set $x_{k+1} = x_k + \alpha_k p_k$
4. Set $k = k + 1$

Outline

Large scale setting

Typical problem

$$\min_{x \in \mathbb{R}^n} f(x) = \sum_{i=1}^{m} f_i(x) \quad n, m \text{ large}$$

Typically, $n, m = 10^6, 10^9$

Gradient step in large scale setting

$$f(x) = \sum_{i=1}^{m} f_i(x), \quad f_i(x) \in \mathbb{R};$$

$$\nabla f(x) = \sum_{i=1}^{m} \nabla f_i(x), \quad \nabla f_i(x) \in \mathbb{R}^n$$

Cost of computing $\nabla f(x)$: $O(nm)$.

Newton's step in large scale setting

$$f(x) = \sum_{i=1}^{m} f_i(x), \quad f_i(x) \in \mathbb{R}$$

$$\nabla f(x) = \sum_{i=1}^{m} \nabla f_i(x), \quad \nabla f_i(x) \in \mathbb{R}^n$$

$$\nabla^2 f(x) = \sum_{i=1}^{m} \nabla^2 f_i(x), \quad \nabla^2 f_i(x) \in \mathbb{R}^{n \times n}$$

Cost of computing $\nabla^2 f(x)$: $O(n^2 m)$.
Cost of solving $\nabla^2 f(x) p = -\nabla f(x)$: $O(n^3)$

What changes from
classical optimization
to machine learning setting?

Expected risk minimization

Given a family of input-output pairs $(\mathrm{x}^i, y^i) \sim P(x, y)$, we want to build a parametric model $F_{\boldsymbol{\theta}}$ such that $F_{\boldsymbol{\theta}}(\mathrm{x}^i) \sim y^i$
We wish to minimize the expected risk:

$$R(\theta) = \int \ell(F_{\boldsymbol{\theta}}(\mathrm{x}), \mathrm{y}) dP(\mathrm{x}, \mathrm{y}) = \mathbb{E}[\ell(F_{\boldsymbol{\theta}}(\mathrm{x}), \mathrm{y})]$$

where ℓ is a loss function.
Problem: $P(x, y)$ is not known !
Approximation: estimate the expected risk by the empirical risk

Warning!: change of notations, the variables become $\boldsymbol{\theta}$

Training procedure

Given a model $F_{\boldsymbol{\theta}}$ and a training set $\{(\mathrm{x}^1, y^1), \ldots, (\mathrm{x}^m, y^m)\}$, find the best parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ to fit the data and to generalize well.

Empirical risk minimisation (ERM) principle

Given a loss function ℓ,

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(X,Y)}(\ell(Y, F_{\boldsymbol{\theta}}(X))) \sim \min_{\boldsymbol{\theta}} \underbrace{R_m(\boldsymbol{\theta})}_{f(\boldsymbol{\theta})} := \frac{1}{m} \sum_{i=1}^{m} \underbrace{\ell(y^i, F_{\boldsymbol{\theta}}(\mathrm{x}^i))}_{f_i(\theta)}$$

Example

$F_{\boldsymbol{\theta}}$: a NN, $\boldsymbol{\theta}$ weights and biases
ℓ: Logistic loss $\ell(\theta) = -y \log(F_{\boldsymbol{\theta}}(x)) - (1-y) \log(1 - F_{\boldsymbol{\theta}}(x))$
Quadratic loss $\ell(\theta) = (y - F_{\boldsymbol{\theta}}(x))^2$

Early stopping

Finding an exact minimizer should be avoided because the risk is to do overtraining: the model is adapted too well to the training set and is not well suited to generalize to unseen samples. Solution: early stopping (Classically $\|\nabla f(x)\| < \text{toll}$ with $\text{toll} \sim 10^{-6}, 10^{-8}$).

Outline

Towards stochastic optimization

Large data sets: redundancy
We don't need to take into account all the samples at each iteration

$$R_m(\boldsymbol{\theta}) := \frac{1}{m} \sum_{i=1}^{m} \underbrace{\ell(y^i, F_{\boldsymbol{\theta}}(\mathrm{x}^i))}_{f_i(\theta)}$$

$\rightarrow$ Stochastic optimization with randomly chosen subsets

Stochastic gradient descent for NN training

GD

For all $k \geq 1$ set

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}(y^i, F_{\boldsymbol{\theta}}(\mathrm{x}^i))$$

SGD

For all $k \geq 1$, choose randomly $\bar{i}_k \in \{1, \ldots, m\}$ and set

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} \mathcal{L}(y^{\bar{i}_k}, F_{\boldsymbol{\theta}}(\mathrm{x}^{\bar{i}_k}))$$

Stochastic gradient descent

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

Stochastic Gradient (SG) Method

1. Given an initial iterate x_0.
2. For $k = 0, 1, 2, \ldots$ do
   - 2.1 choose randomly i_k in $\{1, \ldots, m\}$.
   - 2.2 compute $p_k = -\nabla f_{i_k}(x_k)$
   - 2.3 choose a stepsize $\alpha_k > 0$
   - 2.4 set $x_{k+1} = x_k + \alpha_k p_k$

Stochastic algorithm (non deterministic): the sequence is not uniquely determined from x_0, but depends on the random sequence $\{i_k\}$.

1 epoch = all the samples have been seen (Iteration: a sample, epoch: a full batch)

SGD vs GD

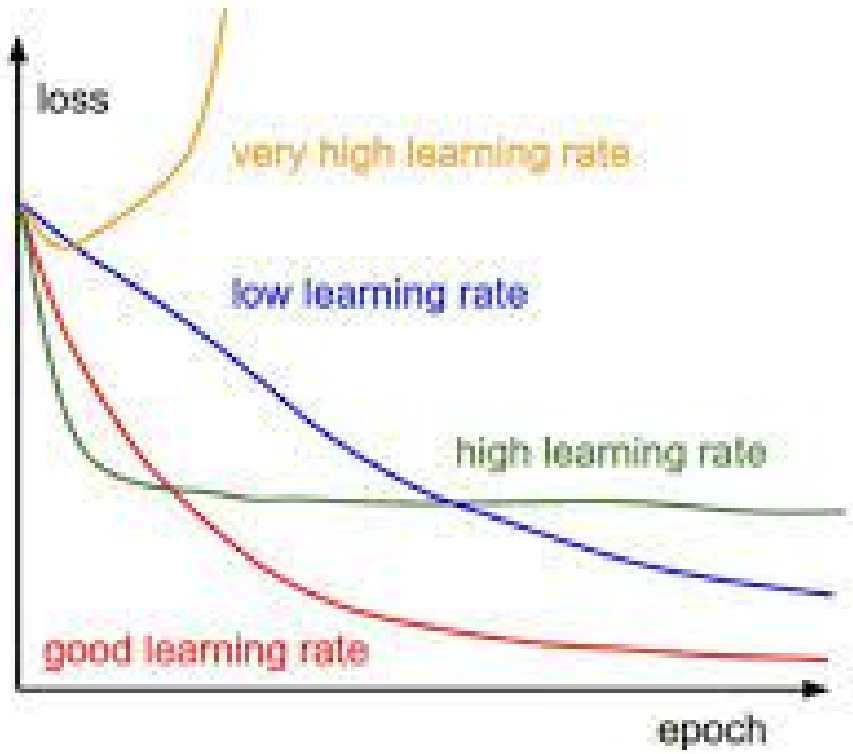
Advantages: much cheaper

- Uses just one sample for the gradient: $\nabla f_{i_k}(\boldsymbol{\theta}_k)$ rather than $\sum_{i=1}^{m} \nabla f_i(\boldsymbol{\theta}_k)$
- Heuristic strategies to update α_k or fixed (usually small) α
- Suitable for very large scale problems
- Fast convergence at the beginning of the procedure: good if high accuracy is not required

Disadvantages: slow convergence

- Uses just partial information
- Needs $\lim_{k\to\infty} \alpha_k = 0$ to converge
- Requires tuning of α_k
- More suitable for convex problems, degrades for ill-posed problems

Choose the learning rate



SGD vs GD

- Standard gradient method is more expensive but gives a better direction (more information is used)
- **Stochastic and full approaches offer different trade-offs in terms of per-iteration costs and expected per-iteration improvement** in minimizing the empirical risk.

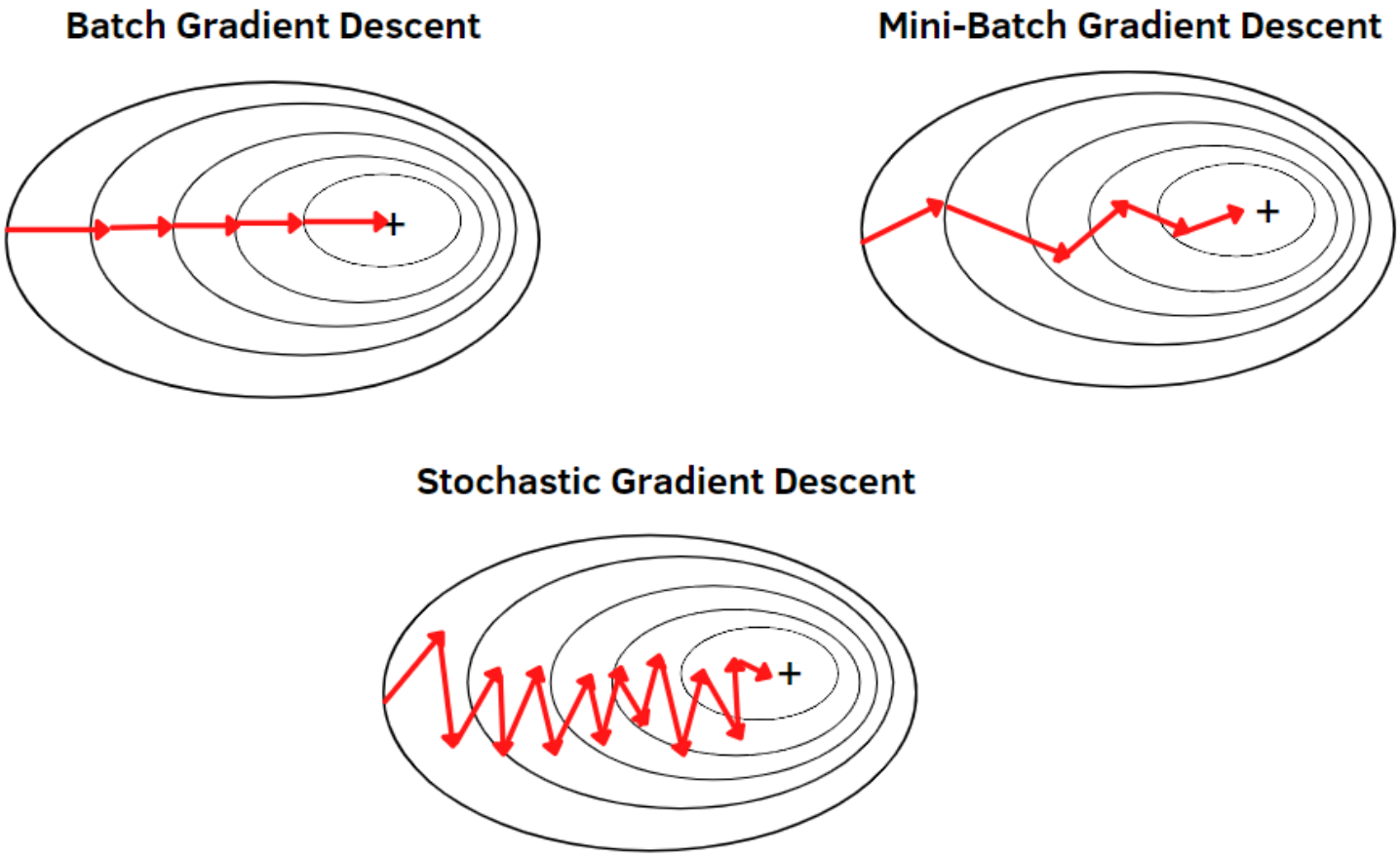
Mini-batch gradient

- Between full gradient method and stochastic gradient method
- Take a small subset $\mathcal{I}_k \subset \{1, \ldots, m\}$ of the samples at each iteration:

$$x_{k+1} = x_k - \frac{\alpha_k}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(x_k),$$

- This allows some degree of parallelization
- Reduces variance of the stochastic gradient estimates, the method is easier to tune in terms of choosing the stepsizes.

SGD vs GD


Batch Gradient Descent
Mini-Batch Gradient Descent
Stochastic Gradient Descent

Convergence results - convex case

Assume f strongly convex, $\boldsymbol{\theta} \in \mathbb{R}^n$

- GD converges linearly:

$$f(\boldsymbol{\theta}_k) - f^* \leq O(\rho^k), \quad \rho \in (0,1),$$

  $\rightarrow$ number of iterations is proportional to $\log(1/\epsilon)$.
- SGD with $\alpha_k = O(1/k)$ converges sublinearly in expectation [Theorem 4.7 `https://epubs.siam.org/doi/epdf/10.1137/16M1080173`]:

$$\mathbb{E}(f(\boldsymbol{\theta}_k) - f^*) = O(1/k).$$

- GD cost : $m \log(1/\epsilon)$ (each iteration costs m)
- SGD cost: $1/\epsilon$ does not depend on m!
- In the big data regime where m is large, $m \log(1/\epsilon) \gg 1/\epsilon$.

Convergence results - general case

What kind of assumptions do we need?

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

with $g(x_k, \xi_k)$ stochastic estimate of $\nabla f(x)$

1. Lipschitz continuity of the gradient
2. The norm of $g(x_k, \xi_k)$ is comparable to the norm of the gradient.
3. In expectation, the vector $-g(x_k, \xi_k)$ is a direction of sufficient descent for f from x_k
4. The variance of $g(x_k, \xi_k)$ is not too large

Assumptions translated in mathematical terms

1. $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all $x, y \in \mathbb{R}^n$
2. $\|\mathbb{E}_{\xi_k}(g(x_k, \xi_k))\| \leq \mu_G \|\nabla f(x_k)\|$
3. $\nabla f(x_k)^T \mathbb{E}_{\xi_k}(g(x_k, \xi_k)) \geq \mu \|\nabla f(x_k)\|^2$
4. $\mathrm{var}(g(x_k, \xi_k)) \leq M + M_V \|\nabla f(x_k)\|^2$

Example

These properties hold with $\mu_G = \mu = 1$ if $g(x_k, \xi_k)$ is an unbiased estimate of $\nabla f(x_k)$

Consequence:

$$\mathbb{E}_{\xi_k}(\|g(x_k, \xi_k)\|^2) \leq M + M_G \|\nabla f(x_k)\|^2, \quad M_G = M_V + \mu_G^2 \geq \mu^2 > 0$$

Notations

We use $\mathbb{E}[\cdot]$ to denote an expected value taken with respect to the joint distribution of all random variables. For example, since x_k is completely determined by the realizations of the independent random variables $\{\xi_1, \xi_2, \ldots, \xi_{k-1}\}$, the total expectation of $f(x_k)$ for any $k \in \mathbb{N}$ can be taken as

$$\mathbb{E}[f(x_k)] = \mathbb{E}_{\xi_1}\mathbb{E}_{\xi_2}\ldots\mathbb{E}_{\xi_{k-1}}[f(x_k)]$$

Theorem (GD, Nonconvex Objective, Fixed Stepsize)

Under the assumption, suppose that the GD method is run with a fixed positive stepsize, $\alpha_k = \alpha \leq \frac{2}{L}$. Then, the sum-of-squares and average-squared gradients of f corresponding to the GD iterates satisfy the following inequalities for all $k \in \mathbb{N}$:

$$\sum_{k=1}^{K} \|\nabla f(x_k)\|^2 \leq f(x_1) - f_{\text{inf}}$$

and therefore

$$\sum_{k=1}^{\infty} \|\nabla f(x_k)\|^2 \leq f(x_1) - f_{\text{inf}}$$

and

$$\lim_{k \to \infty} \|\nabla f(x_k)\| = 0.$$

Proof (I)

From the Lipschitz continuity of the gradient

$$\begin{aligned} f(x_{k+1}) - f(x_k) &\leq \nabla f(x_k)^T (x_{k+1} - x_k) + \frac{L}{2}\|x_{k+1} - x_k\|^2 \\ &= -\alpha \nabla f(x_k)^T \nabla f(x_k) + \frac{\alpha_k^2 L}{2}\|\nabla f(x_k)\|^2 \\ &= \alpha(-1 + \frac{L}{2}\alpha)\|\nabla f(x_k)\|^2 < -\|\nabla f(x_k)\|^2 \end{aligned}$$

Summing both sides of this inequality for $k \in \{1, \ldots, K\}$ and recalling the assumption gives

$$f_{\text{inf}} - f(x_1) \leq f(x_{K+1}) - f(x_1) \leq -\sum_{k=1}^{K} \|\nabla f(x_k)\|^2$$

Theorem (SGD, Nonconvex Objective, Fixed Stepsize)

Under the assumption, suppose that the SG method is run with a fixed positive stepsize, $\alpha_k = \alpha \leq \frac{\mu}{LM_G}$. Then, the expected sum-of-squares and average-squared gradients of f corresponding to the SG iterates satisfy the following inequalities for all $k \in \mathbb{N}$:

$$\mathbb{E}\left[\sum_{k=1}^{K} \|\nabla f(x_k)\|^2\right] \leq \frac{K\alpha LM}{\mu} + 2\frac{(f(x_1) - f_{\text{inf}})}{\mu\alpha}$$

and therefore

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \|\nabla f(x_k)\|^2\right] \leq \frac{\alpha LM}{\mu} + 2\frac{(f(x_1) - f_{\text{inf}})}{\mu K\alpha} \overset{K\to\infty}{\to} \frac{\alpha LM}{\mu}.$$

Proof (I)

From the Lipschitz continuity of the gradient

$$f(x_{k+1}) - f(x_k) \leq \nabla f(x_k)^T (x_{k+1} - x_k) + \frac{L}{2}\|x_{k+1} - x_k\|^2$$
$$= -\alpha \nabla f(x_k)^T g(x_k, \xi_k) + \frac{\alpha^2 L}{2}\|g(x_k, \xi_k)\|^2$$

Passing to the expected value

$$\mathbb{E}_{\xi_k}[f(x_{k+1})] - f(x_k) \leq -\alpha \nabla f(x_k)^T \mathbb{E}_{\xi_k}[g(x_k, \xi_k)]$$
$$+ \frac{\alpha^2 L}{2}\mathbb{E}_{\xi_k}[\|g(x_k, \xi_k)\|^2]$$

Proof (II)

Taking the total expectation and from the assumption on the step:

$$\mathbb{E}[f(x_{k+1})] - \mathbb{E}(f(x_k)) \leq -(\mu - \frac{1}{2}\alpha L M_G)\alpha \mathbb{E}[\|\nabla f(x_k)\|^2] + \frac{1}{2}\alpha^2 L M$$
$$\leq -\frac{1}{2}\mu\alpha \mathbb{E}[\|\nabla f(x_k)\|^2] + \frac{1}{2}\alpha^2 L M.$$

Summing both sides of this inequality for $k \in \{1, \ldots, K\}$ and recalling the assumption gives

$$f_{\text{inf}} - f(x_1) \leq \mathbb{E}[f(x_{K+1})] - f(x_1) \leq -\frac{1}{2}\mu\alpha \sum_{k=1}^{K} \mathbb{E}[\|\nabla f(x_k)\|^2] + \frac{1}{2} K \alpha^2 L M.$$

Rearranging and dividing by K yields the thesis.

Outline

Limitations of SGD

- difficult to set the stepsize
- may diverge with fixed stepsizes
- slow, sublinear rate of convergence whit diminishing stepsizes

Several techniques to try to accelerate the convergence:

- variance reduction methods
- accelerated methods with momentum
- adaptive stepsize

Variance reduction methods

Variance reduction methods: improve rate of convergence by incorporating new gradient information in order to construct a more reliable step with smaller variance

Can achieve linear rate with fixed stepsize for strongly convex objectives if the variance of the stochastic vectors $\nabla f_{i_k}(x_k)$ decreases geometrically, i.e. it exist $M > 0$ and $\xi \in (0, 1)$ such that

$$\text{var} = \mathbb{E}[(\|\nabla f_{i_k}(x_k)\| - \mathbb{E}(\|\nabla f_{i_k}(x_k)\|))^2] \leq M\xi^k,$$

Two classes of methods:

- dynamic sample size
- gradient aggregation methods

Dynamic sampling methods

Gradually increasing the mini-batch size: increasingly accurate gradient estimates

$$x_{k+1} = x_k - \frac{\alpha}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(x_k), \quad |\mathcal{I}_k| = n_k = \lceil \tau^{k-1} \rceil$$

for some $\tau > 1$. It exists $M > 0$ such that $\mathrm{var} \leq \frac{M}{n_k}$.

- In practice : good value of the parameter τ (may jeopardize per-iteration costs)
- Adaptive sampling algorithm: choosing the sample sizes not according to a prescribed sequence, but adaptively : $\mathrm{sampled\ var} \leq C \|\nabla f_{\mathcal{I}_k}(x_k)\|^2$ for a positive constant $C > 0$. If this is not satisfied the size of the set is increased.

Gradient aggregation methods

Achieve a lower variance by using full gradient information in a parsimonious way: either increase the cost or the memory Among them:

- SVRG - stochastic variance reduced gradient
- SAGA - stochastic average gradient

SVRG: stochastic variance reduced gradient

It operates in cycles.
At the beginning of each cycle, an iterate x_k is available at which the algorithm computes a full (or batch) gradient

$$\nabla f_m(x_k) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x_k).$$

Then, after initializing $\tilde{x}_1 = x_k$, a set of N inner iterations indexed by j with an update $\tilde{x}_{j+1} = \tilde{x}_j - \alpha \tilde{g}_j$ are performed, where

$$\begin{aligned}\tilde{g}_j &= \nabla f_{i_j}(\tilde{x}_j) - (\nabla f_{i_j}(x_k) - \nabla f_m(x_k)) \\ &= \begin{pmatrix} \frac{\partial f_{i_j}(\tilde{x}_j)}{\partial x_1} \\ \vdots \\ \frac{\partial f_{i_j}(\tilde{x}_j)}{\partial x_n} \end{pmatrix} - \begin{pmatrix} \frac{\partial f_{i_j}(x_k)}{\partial x_1} \\ \vdots \\ \frac{\partial f_{i_j}(x_k)}{\partial x_n} \end{pmatrix} + \frac{1}{m} \begin{pmatrix} \frac{\partial f_1(x_k)}{\partial x_1} + \cdots + \frac{\partial f_m(x_k)}{\partial x_1} \\ \vdots \\ \frac{\partial f_1(x_k)}{\partial x_n} + \cdots + \frac{\partial f_m(x_k)}{\partial x_n} \end{pmatrix}\end{aligned}$$

and $i_j \in \{1, \ldots, m\}$ is chosen at random.

SVRG method

1. Given an initial iterate x_0, stepsize $\alpha > 0$ and positive integer N.
2. For k = 0,1,2,... do
   - 2.1 Compute the batch gradient $\nabla f_m(x_k)$
   - 2.2 Initialize $\tilde{x}_1 = x_k$
   - 2.3 for $j = 1, \ldots, N$ do
     - 2.3.1 Chose i_j uniformly from $\{1, \ldots, m\}$.
     - 2.3.2 Set $\tilde{g}_j = \nabla f_{i_j}(\tilde{x}_j) - (\nabla f_{i_j}(x_k) - \nabla f_m(x_k))$
     - 2.3.3 Set $\tilde{x}_{j+1} = \tilde{x}_j - \alpha \tilde{g}_j$.
   - 2.4 Update x_{k+1}
     - 2.4.1 Option 1: Set $x_{k+1} = \tilde{x}_{N+1}$.
     - 2.4.2 Option 2: Set $x_{k+1} = \frac{1}{N} \sum_{j=1}^{N} \tilde{x}_{j+1}$
     - 2.4.3 Option 3: Choose j uniformly from $\{1, ..., N\}$ and set $x_{k+1} = \tilde{x}_{j+1}$.

SVRG as a variance reduction technique

- SVRG update: application of a variance reduction technique to the stochastic gradient.
- Variance reduction is a statistical technique that is used to reduce the variance of a random variable X by using another random variable Y, which is positively correlated with X. A new variable Z_α is defined as

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}(Y),$$

  which has reduced variance.
- Since $\mathbb{E}[\nabla f_{i_j}(\mathrm{x}_k)] = \nabla f_m(\mathrm{x}_k)$, one can view $\tilde{\mathrm{g}}_j$ as the result of the application of this technique to the stochastic gradient $\nabla f_{i_j}(\tilde{\mathrm{x}}_j)$, choosing $\alpha = 1$.
- $\tilde{\mathrm{g}}_j$ represents an unbiased estimator of $\nabla f_m(\tilde{\mathrm{x}}_j)$, but with a smaller variance

SAGA

Does not operate in cycles, and computes batch gradients only at the initial point.

- Compute $\nabla f_i(x_0)$ for $i = 1, \ldots, m$ and store it in a $n \times m$ table.
- At iteration k choose randomly i_k, compute $\nabla f_{i_k}(\mathrm{x}_k)$ and update the i_k-column of the table
- Compute the new step. Choose $i_k \in 1, ..., m$ randomly and set

$$\mathrm{g}_k = \nabla f_{i_k}(\mathrm{x}_k) - \nabla f_{i_k}(\mathrm{x}_{[i_k]}) + \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\mathrm{x}_{[i]})$$

  where $\mathrm{x}_{[i]}$ represents the latest iterate at which ∇f_i was evaluated (from the table).
- $\mathbb{E}[\mathrm{g}_k] = \nabla f_m(\mathrm{x}_k)$
- As opposed to SVRG, SAGA needs to store all the gradients but does not have additional cost

SAGA algorithm

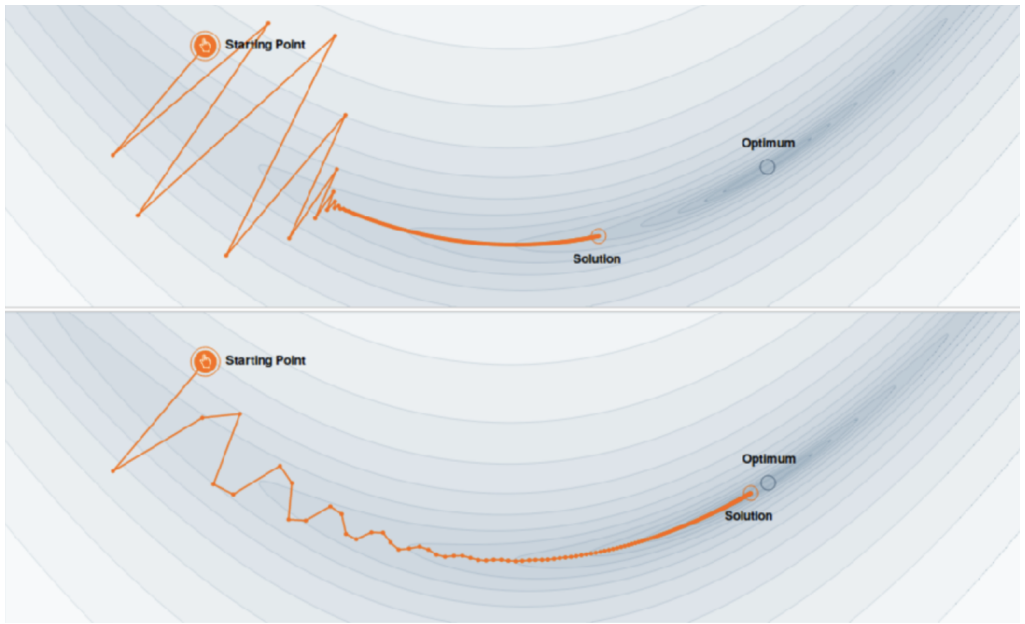
SAGA method

1. Given an initial iterate x_0, stepsize $\alpha > 0$.
2. For $i = 1, \ldots, m$ do
   - 2.1 Compute $\nabla f_i(x_1)$.
   - 2.2 Store $\nabla f_i(x_{[i]}) = \nabla f_i(x_1)$ (create the table).
3. For $k = 1, 2, \ldots$ do
   - 3.1 Choose j uniformly in $\{1, \ldots, m\}$.
   - 3.2 Compute $\nabla f_j(x_k)$.
   - 3.3 Set $g_k = \nabla f_j(x_k) - \nabla f_j(x_{[j]}) + \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(x_{[i]})$
   - 3.4 Store $\nabla f_j(x_{[j]}) = \nabla f_j(x_k)$ (update the table).
   - 3.5 Set $x_{k+1} = x_k - \alpha g_k$.

Outline

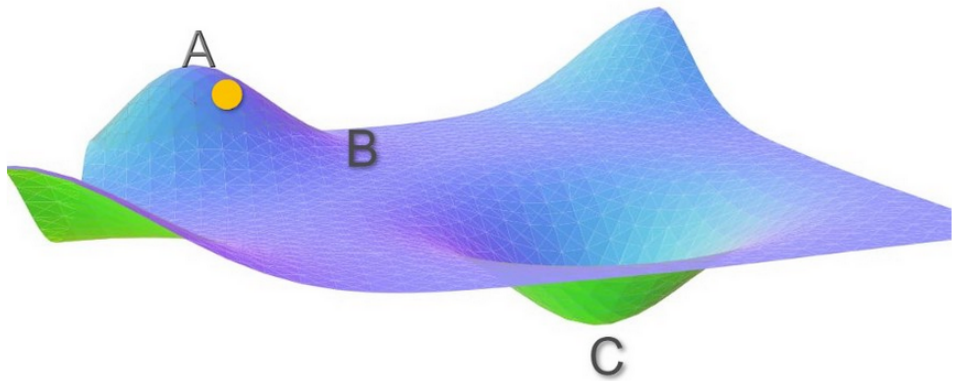
Accelerated gradient method: momentum

1. The gradient in a plateau is negligible or zero $\rightarrow$ very small steps.
2. The path followed by gradient descent is very jittery, also when operating with mini-batch mode.
3. Could never reach the optimum

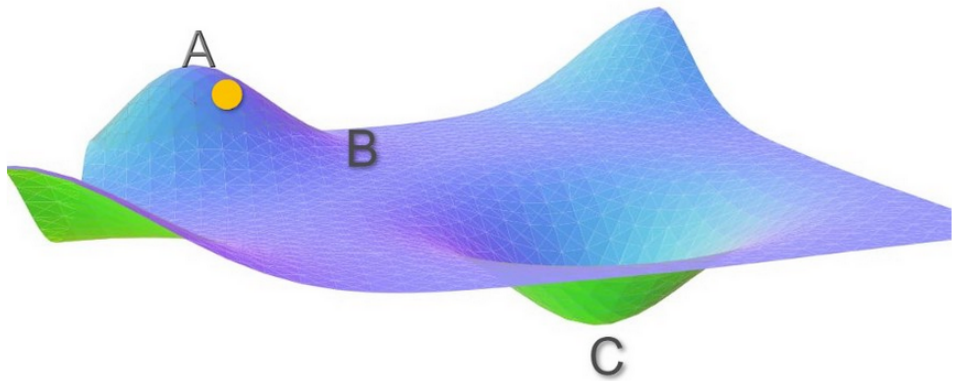


How to fix this? $\rightarrow$ gradient descent with momentum

Momentum: keep memory of the past



Momentum: keep memory of the past



- Define the step as the average of past gradients instead of the gradient at the current iteration.
- Cannot consider all the gradients with equal weightage.
- Need to use some sort of weighted average

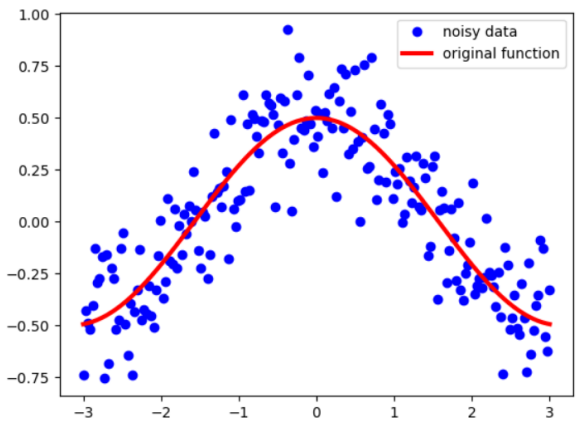
Gradient method with momentum

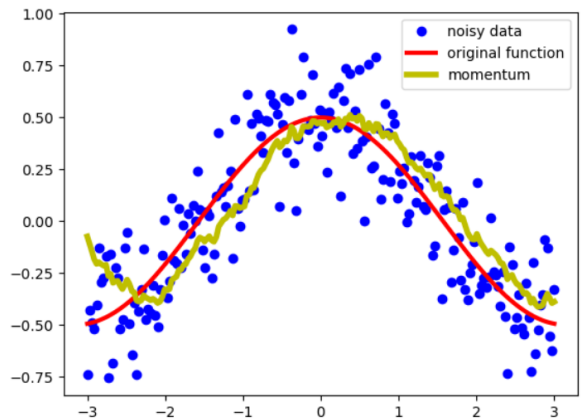
Exponential Moving Average (EMA)

Consider a noisy sequence $y(t)$. The EMA $s(t)$ for a series $y(t)$ may be calculated recursively as:

$$s(t) = \begin{cases} y(1) & \text{if } t = 1, \\ \beta s(t-1) + (1-\beta) y(t) & \text{if } t > 1 \end{cases}$$

where $\beta \in [0, 1]$ represents the degree of weighting increase. A lower β discounts older observations faster.





EMA for gradients

- New weight update:

$$\mathsf{s}_k = \beta \mathsf{s}_{k-1} + (1-\beta)\mathsf{g}_k$$

  where $\mathsf{s}_k = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$, g_k is the gradient approximation (full gradient, or a mini-batch or stochastic gradient).

- Often $\beta \to \beta_k$ and $(1-\beta) \to \alpha_k$

$$\boldsymbol{\theta}_{k+1} = \underbrace{\boldsymbol{\theta}_k - \alpha_k g_k}_{\text{standard gradient step}} + \underbrace{\beta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})}_{\text{momentum term}},$$

- Special cases:
  - $\beta_k = 0$ for all $k \in \mathbb{N}$: classical GD/SGD
  - $\alpha_k = \alpha$ and $\beta_k = \beta$: *heavy ball method*.

Heavy ball momentum

- By expanding the update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{j=1}^{k} \beta^{k-j} g_k$$

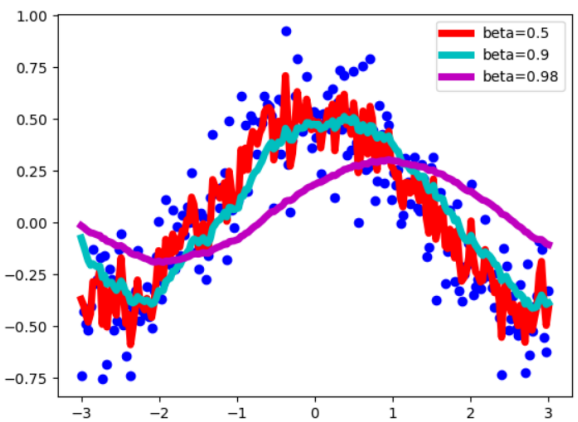
each step is an exponentially decaying average of past gradients.

Heavy ball momentum

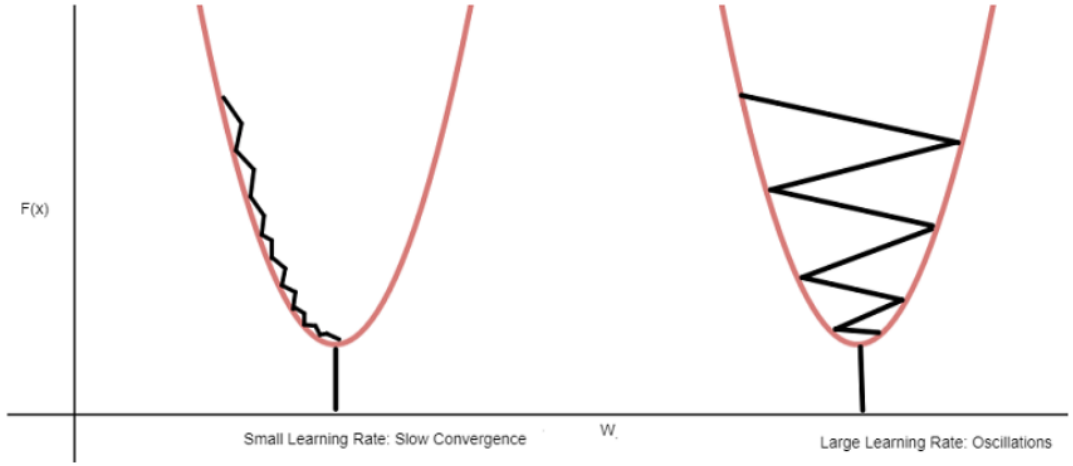
- By expanding the update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{j=1}^{k} \beta^{k-j} g_k$$

  each step is an exponentially decaying average of past gradients.
- Let's analyse the contribution of β. Assume $\alpha = 1$. At $k = 3$; g_3 will contribute 100% of its value
  - $\beta = 0.1$: g_2 10% and g_1 1%:
  - $\beta = 0.9$: g_2 90% and g_1 81%.
  - Higher β: contribution from earlier gradients decreases slowly, accommodates more gradients from the past. Usually $\beta \sim 0.9$



How does momentum help?



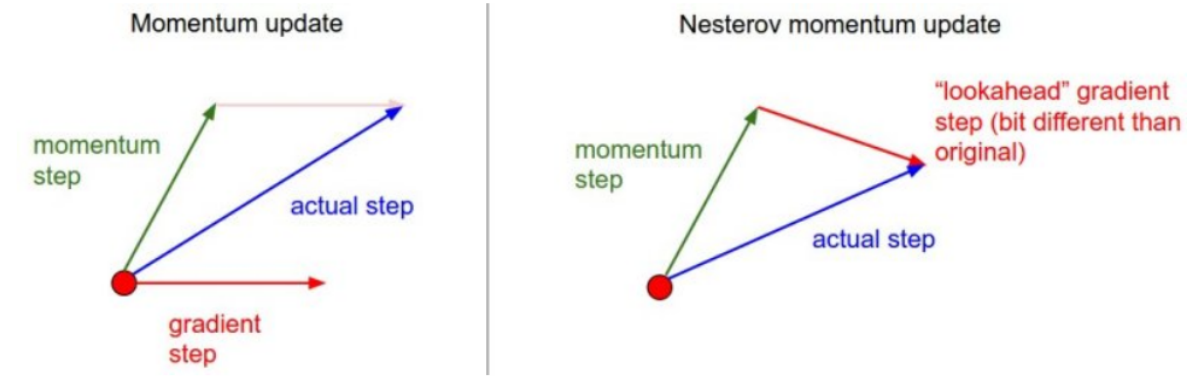
- LR too small: small steps, convergence takes a lot of time even when the gradient is high.
- LR too high: the sequence oscillates around the minima

How does momentum fix this?

1. *All the past gradients have the same sign*: the summation term will become large and we will take large steps
2. *Different signs*: the summation term will become small and the steps will be small, damping the oscillations.

Nesterov accelerated gradient

- Treats the future approximate position $\tilde{\boldsymbol{\theta}}_k = \boldsymbol{\theta}_k + \beta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$ as a "lookahead"
- Computes the gradient at $\tilde{\boldsymbol{\theta}}_k$ instead of the old $\boldsymbol{\theta}_k$

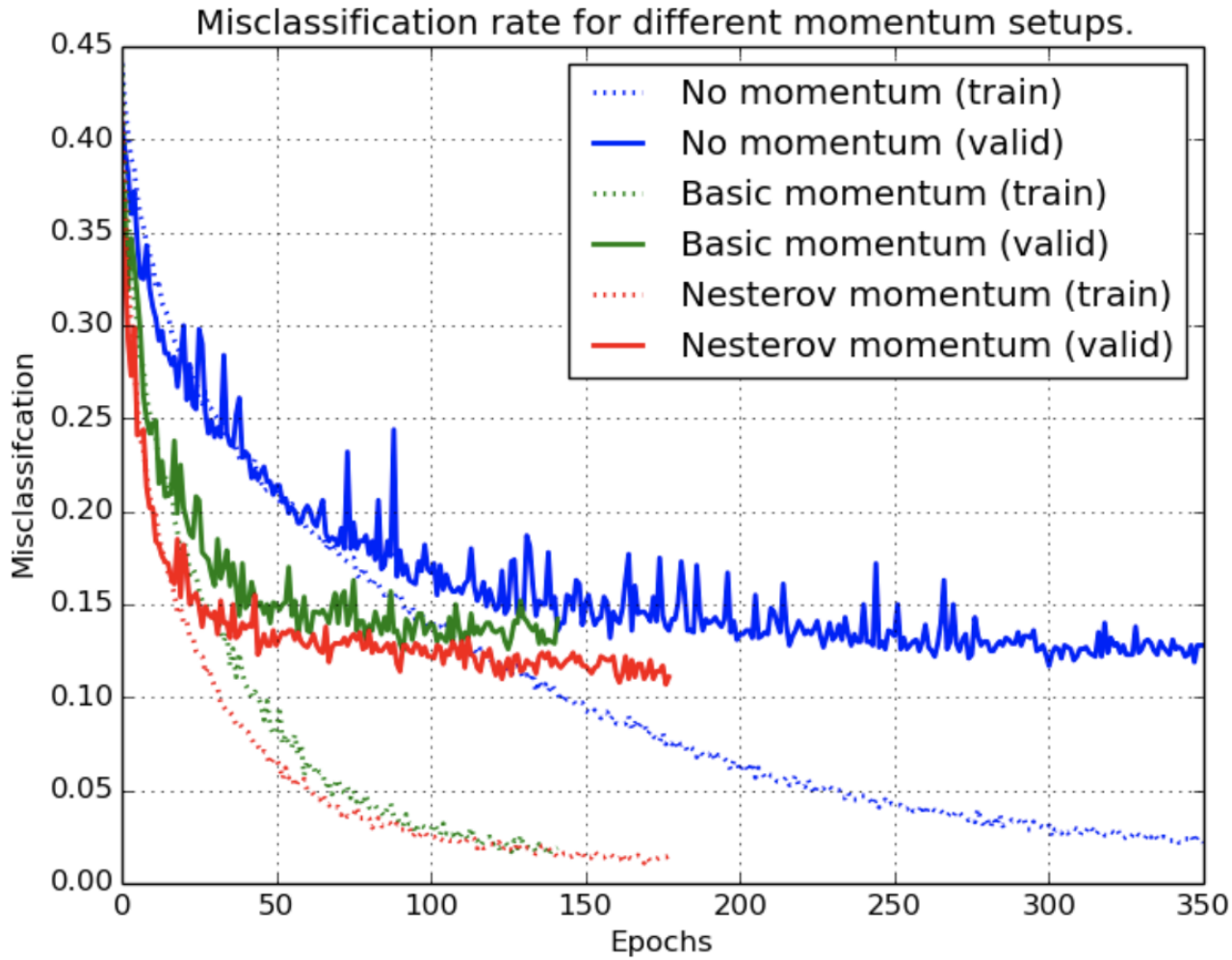


Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

- The Nesterov update is then $\boldsymbol{\theta}_{k+1} = \tilde{\boldsymbol{\theta}}_k - \alpha_k g(\tilde{\boldsymbol{\theta}}_k)$
- In momentum: first gradient descent step and then momentum term, in Nesterov: first momentum then gradient descent (with the gradient evaluated at $\tilde{\boldsymbol{\theta}}_k$, not at $\boldsymbol{\theta}_k$).
- Better convergence: distance to the optimal value decaying with a rate $O(1/k)$ for momentum and $O(1/k^2)$ for Nesterov

Numerical test

CNN on a classification problem



Outline

AdaGrad (Adaptive Gradient, 2011)

$$v_0 = 0$$
$$v_{k+1} = v_k + g_k^2$$
$$x_{k+1} = x_k + \alpha g_k / \sqrt{v_k}$$

- Diagonal scaling on the coordinates of g_k: adaptive componentwise stepsizes, good for coordinates that can vary by orders of magnitude.
- Useful for sparse gradients:
  - Frequent updates (large accumulated gradient) smaller learning rate : prevents the parameter from changing too drastically and stabilizes learning.
  - Infrequent updates (small accumulated gradient) larger learning rate : more significant updates when necessary.
- Stepsizes: $\frac{\alpha}{\sqrt{v_k}} \rightarrow$ Stepsizes go to zero!

RMSProp: Root Mean Square Propagation (2012)

Componentwise:

$$
\begin{aligned}
v_0 &= 0 \\
v_{k+1} &= \beta v_k + (1-\beta) g_k^2 \\
x_{k+1} &= x_k + \alpha g_k / \sqrt{v_k}
\end{aligned}
$$

- Improves ADAGRAD by considering the exponential moving average of the squared gradient
- Slows down the decrease of the stepsize

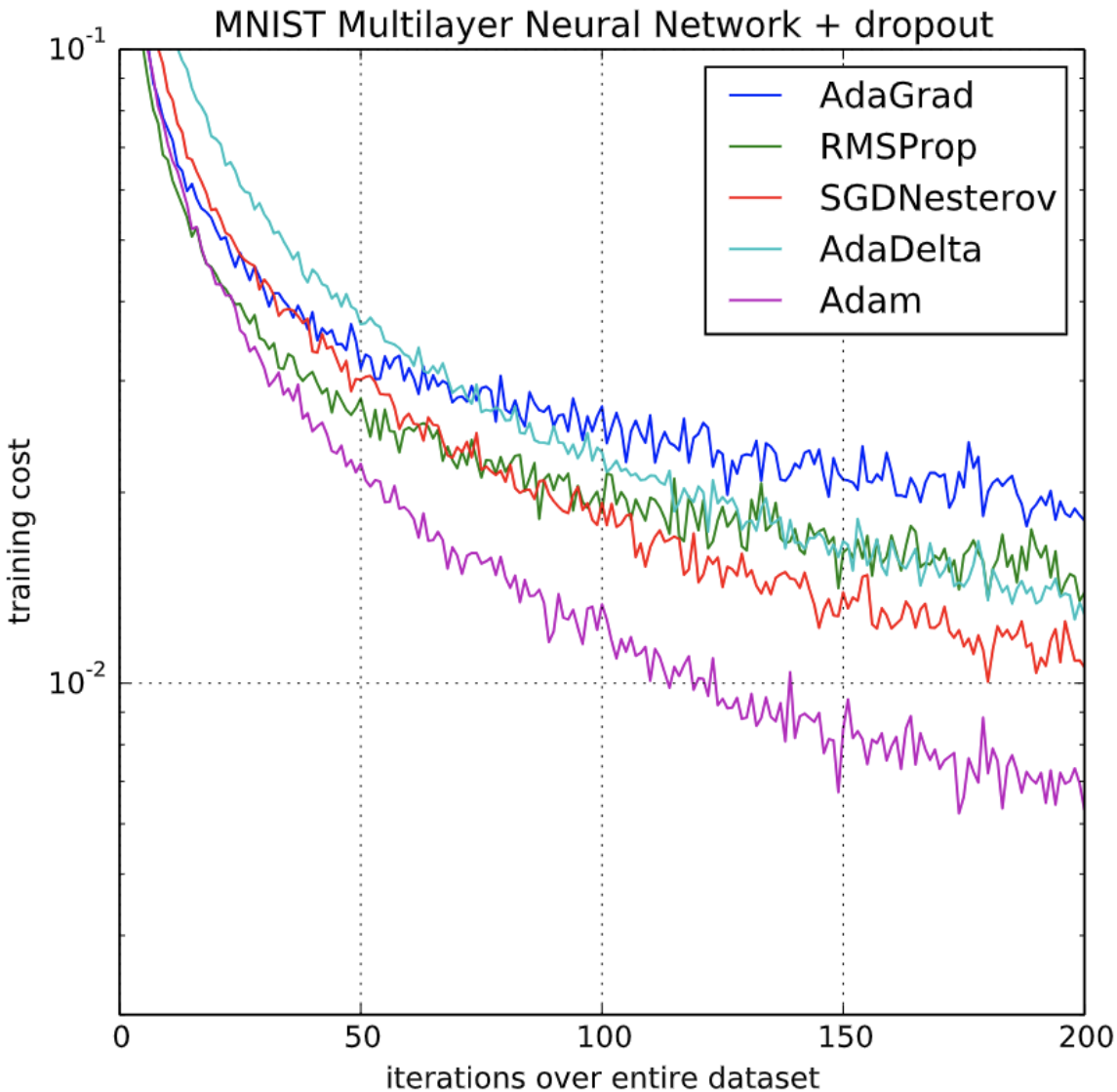
Adam (2015)

Componentwise:

$$
\begin{aligned}
v_0 &= 0 \\
m_{k+1} &= \beta_1 m_k + (1-\beta_1) g_k \\
v_{k+1} &= \beta_2 v_k + (1-\beta_2) g_k^2 \\
\hat{m}_k &= \frac{m_k}{1-\beta_1^k} \\
\hat{v}_k &= \frac{v_k}{1-\beta_2^k} \\
x_{k+1} &= x_k + \alpha \hat{m}_k / \sqrt{\hat{v}_k}
\end{aligned}
$$

The algorithm updates exponential moving averages of the gradient (m_k) and the squared gradient (v_k) These moving averages are initialized as (vectors of) 0's, leading to moment estimates that are biased towards zero

Numerical test



Outline

Inexact second order methods

Newton's step:

$$x_{k+1} = x_k + \alpha_k p_k, \quad H_m(x_k)p_k = -\nabla f_m(x_k), \quad H_m := \sum_{i=1}^{m} H_i \in \mathbb{R}^{n\times n}$$

- Classically: accurately solve the Newton system through matrix factorization techniques (LU, Cholesky..)
- Instead: use an iterative approach for inexact solution:

$$\|H_m(x_k)p_k + \nabla f_m(x_k)\| \leq \mathrm{tol}$$

Example: conjugate gradient (CG) method.

Inexact second order methods

- If linear solves are accurate enough, Newton-CG method can enjoy a superlinear rate of convergence
- No access to the Hessian itself, only Hessian-vector products: Hessian-free. This is ideal when such products can be coded directly without having to form an explicit Hessian
- Each product is at least as expensive as a gradient evaluation, but as long as the number of products (one per CG iteration) is not too large, the improved rate of convergence can compensate for the extra per-iteration work required over a simple full gradient method.

Example

Let $f(\mathrm{x}) = e^{x_1 x_2}$ and for $d \in \mathbb{R}^2$
$\Phi(\mathrm{x}; \mathrm{d}) = \nabla f(\mathrm{x})^T \mathrm{d} = x_2 e^{x_1 x_2} d_1 + x_1 e^{x_1 x_2} d_2$.

$$\nabla \Phi_{\mathrm{x}}(\mathrm{x}, \mathrm{d}) = H_f(\mathrm{x})\mathrm{d} = \begin{bmatrix} x_2^2 e^{x_1 x_2} d_1 + (e^{x_1 x_2} + x_1 x_2 e^{x_1 x_2}) d_2 \\ (e^{x_1 x_2} + x_1 x_2 e^{x_1 x_2}) d_1 + x_1^2 e^{x_1 x_2} d_2 \end{bmatrix}$$

with $H_f(\mathrm{x})$ the Hessian matrix of f. Can compute $H_f(\mathrm{x})\mathrm{d}$ without computing $H_f(\mathrm{x})$ explicitly.
Storing the scalars x_1, x_2 and $e^{x_1 x_2}$ from the evaluation of f, the additional costs of computing the gradient-vector and Hessian-vector products are small.
In general, one can compute $H_f(\mathrm{x})\mathrm{d}$ at a cost that is a small multiple of the cost of evaluating $\nabla f(\mathrm{x})$, and without forming the Hessian, which would require $O(n^2)$ storage.

Subsampled Hessian-Free Newton Methods

Idea: the Hessian matrix need not be as accurate as the gradient to yield an effective iteration. Given $\mathcal{I}_k \subset \{1, \ldots, m\}$ and $\mathcal{I}_k^H \subset \{1, \ldots, m\}$, the stochastic gradient estimate is

$$\nabla f_{\mathcal{I}_k}(x_k) = \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(x_k),$$

and the stochastic Hessian estimate is

$$H_{\mathcal{I}_k}(x_k) = \frac{1}{|\mathcal{I}_k^H|} \sum_{i \in \mathcal{I}_k^H} H_{f_i}(x_k),$$

where $\mathcal{I}_k^H$ is uncorrelated with $\mathcal{I}_k$ and $|\mathcal{I}_k^H| < |\mathcal{I}_k|$.

- $|\mathcal{I}_k^H|$ small to reduce the cost of product involving the Hessian and thus the cost of CG iterations.
- $|\mathcal{I}_k|$ large enough so that the curvature information captured through the Hessian-vector products is productive.

Subsampled Hessian-Free Inexact Newton Method

1. Given an initial iterate x_0, $\rho \in (0,1)$, and $\max_{CG} \in \mathbb{N}$.
2. For $k = 0, 1, 2, \ldots$ do
   - 2.1 Choose randomly $\mathcal{I}_k$ and $\mathcal{I}_k^H$.
   - 2.2 Compute p_k applying Hessian-free CG to solve $H_{\mathcal{I}_k}(x_k)p_k = -\nabla f_{\mathcal{I}_k}(x_k)$ until $\max_{CG}$ iterations have been performed or a trial solution yields

$$\|r_k\| := \|H_{\mathcal{I}_k}(x_k)p_k + \nabla f_{\mathcal{I}_k}(x_k)\| \leq \rho \|\nabla f_{\mathcal{I}_k}(x_k)\|$$

   - 2.3 Set $x_{k+1} = x_k + \alpha_k p_k$, where α_k satisfies (A)+(W)

Cost of step computation in Newton-CG

- Let g_{cost} be the cost of computing $\nabla f_{\mathcal{I}_k}(x_k)$
- $\text{factor} \times g_{cost}$ denote the cost of one Hessian-vector product.
- $\max_{CG}$ maximum number of CG iterations

The step computation cost is

$$\max_{CG} \times \text{factor} \times g_{cost} + g_{cost}.$$

- If $|\mathcal{I}_k^H| = |\mathcal{I}_k| = n$ for all $k \in \mathbb{N}$, the factor is at least 1 and $\max_{CG} \sim 5, 20$: cost is many times the cost of an SG iteration.
- Stochastic subsampled framework: the factor can be chosen to be sufficiently small such that $\max_{CG} \times \text{factor} \sim 1$, leading to a per-iteration cost proportional to that of SG.

Rate of convergence

Defining $r_k := H_{\mathcal{I}_k}(x_k)p_k + \nabla f_{\mathcal{I}_k}(x_k)$ for all $k \in \mathbb{N}$, the iteration can enjoy a linear, superlinear, or quadratic rate of convergence by controlling $\|r_k\|$, where for the superlinear rates one must have

$$\frac{\|r_k\|}{\|\nabla f(x_k)\|} \to 0.$$

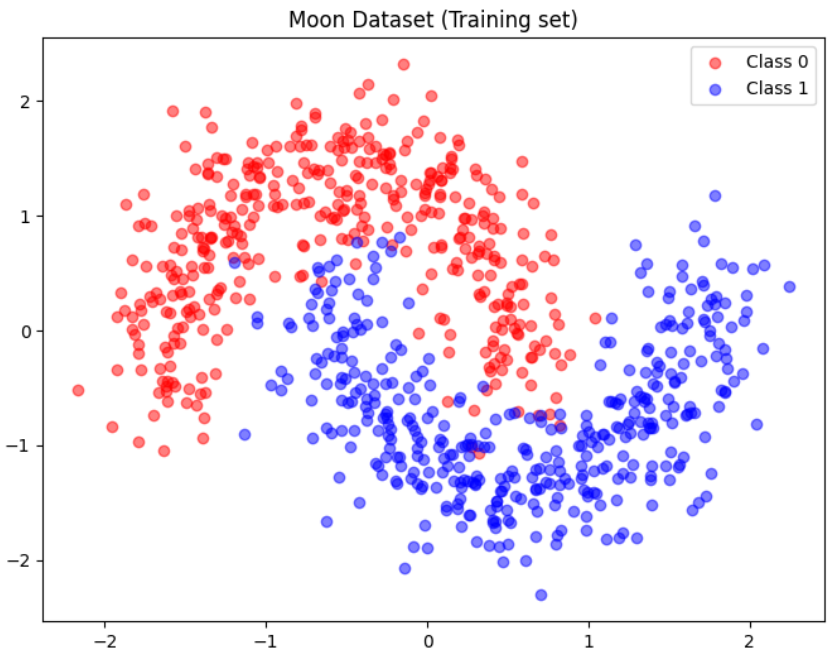
1

[1] R. S. Dembo, Eisenstat S. C., and T. Steihaug. Inexact Newton Methods. SIAM Journal on Numerical Analysis, 19(2):400-408, 1982.

Exercise

Consider the moon dataset

$$X, y = make\_moons(n\_samples = N, noise = \delta, random\_state = 42)$$



- Split the dataset into training and test sets
- Normalize the data for better performance
- Build a simple neural network model
- Train it with different optimizers (GD, SGD, ADAM..)

Exercise

Test the effect of:

- size N
- noise δ
- learning rate
- momentum