Stochastic Gradient Method

V. Leclère (ENPC)

May 28th, 2021

Why should I bother to learn this stuff ?

- Main algorithm principle for training machine learning model, and in particular deep neural network
- $\Longrightarrow$ useful for
  - understanding how the library train ML models
  - specialization in optimization
  - specialization in machine learning

Contents

The optimization problem

We consider the following optimization problem

$$\underset{x \in \mathbb{R}^p}{\text{Min}} \qquad F(x) := \mathbb{E}\Big[f(x, \xi)\Big]$$

where ξ is a random variable.

Contents

Computing the gradient

$$F(x) := \mathbb{E}\Big[f(x, \xi)\Big]$$

Under some regularity conditions (e.g. $f(\cdot, \xi)$ differentiable, $\dfrac{\partial f(x, \cdot)}{\partial x}$ Lipschitz, and ξ integrable) we have

$$\nabla F(x) = \mathbb{E}\left[\frac{\partial f}{\partial x}(x, \xi)\right]$$

This is obvious if ξ is finitely supported : $\mathrm{supp}(\xi) = \{\xi_i\}_{i \in [N]}$, and $p_i := \mathbb{P}(\xi = \xi_i)$,

$$\nabla F(x) = \frac{\partial}{\partial x}\left(\sum_{i \in [N]} p_i f(x, \zeta)\right) = \sum_{i \in [N]} p_i \frac{\partial}{\partial x} f(x, \zeta)$$

Computing the gradient

$$F(x) := \mathbb{E}\Big[f(x, \boldsymbol{\xi})\Big]$$

Under some regularity conditions (e.g. $f(\cdot, \xi)$ differentiable, $\dfrac{\partial f(x, \cdot)}{\partial x}$ Lipschitz, and $\boldsymbol{\xi}$ integrable) we have

$$\nabla F(x) = \mathbb{E}\left[\frac{\partial f}{\partial x}(x, \boldsymbol{\xi})\right]$$

This is obvious if $\boldsymbol{\xi}$ is finitely supported : $\mathrm{supp}(\boldsymbol{\xi}) = \{\xi_i\}_{i\in[N]}$, and $p_i := \mathbb{P}(\boldsymbol{\xi} = \xi_i)$,

$$\nabla F(x) = \frac{\partial}{\partial x}\left(\sum_{i\in[N]} p_i f(x, \zeta)\right) = \sum_{i\in[N]} p_i \frac{\partial}{\partial x} f(x, \zeta)$$

Standard continuous optimization method

Thus, we are looking at

$$\underset{x \in \mathbb{R}^p}{\text{Min}} \; F(x)$$

where F is a (strongly) convex differentiable function if $f(\cdot, \xi)$ is, and we know how to compute its gradient.

Thus, we should be able to solve our problem through the method presented in earlier courses:

- gradient algorithm
- conjugate gradient
- Newton / Quasi-Newton

Why bother with another (class of) algorithm ?

Standard continuous optimization method

Thus, we are looking at

$$\underset{x \in \mathbb{R}^p}{\text{Min}} \; F(x)$$

where F is a (strongly) convex differentiable function if $f(\cdot, \xi)$ is, and we know how to compute its gradient.

Thus, we should be able to solve our problem through the method presented in earlier courses:

- gradient algorithm
- conjugate gradient
- Newton / Quasi-Newton

Why bother with another (class of) algorithm ?

Standard continuous optimization method

Thus, we are looking at

$$\underset{x \in \mathbb{R}^p}{\text{Min}} \; F(x)$$

where F is a (strongly) convex differentiable function if $f(\cdot, \xi)$ is, and we know how to compute its gradient.

Thus, we should be able to solve our problem through the method presented in earlier courses:

- gradient algorithm
- conjugate gradient
- Newton / Quasi-Newton

Why bother with another (class of) algorithm ?

Contents

Computing the gradient is costly

For a given solution $x \in \mathbb{R}^p$ computing the gradient

$$\nabla F(x) = \mathbb{E}\left[\frac{\partial f(x, \xi)}{\partial x}\right]$$

is costly as it requires to compute a multidimensionnal integral (if ξ admits a density), or a large sum.

Indeed, in most machine learning application, we consider that ξ is uniformly distributed over the data (*empirical risk minimization*), thus computing the gradient require a pass over every sample in the dataset.

Dataset of size $N > 10^6$ are common.

Computing the gradient is costly

For a given solution $x \in \mathbb{R}^p$ computing the gradient

$$\nabla F(x) = \mathbb{E}\left[\frac{\partial f(x, \xi)}{\partial x}\right]$$

is costly as it requires to compute a multidimensionnal integral (if ξ admits a density), or a large sum.

Indeed, in most machine learning application, we consider that ξ is uniformly distributed over the data (*empirical risk minimization*), thus computing the gradient require a pass over every sample in the dataset.

Dataset of size $N > 10^6$ are common.

Computing the gradient is costly

For a given solution $x \in \mathbb{R}^p$ computing the gradient

$$\nabla F(x) = \mathbb{E}\left[\frac{\partial f(x, \xi)}{\partial x}\right]$$

is costly as it requires to compute a multidimensionnal integral (if ξ admits a density), or a large sum.

Indeed, in most machine learning application, we consider that ξ is uniformly distributed over the data (*empirical risk minimization*), thus computing the gradient require a pass over every sample in the dataset.

Dataset of size $N > 10^6$ are common.

Estimating the gradient

Instead of using a true gradient

$$g^{(k)} = \nabla F(x^{(k)})$$

we can use a *statistical estimator* of the gradient

$$\hat{g}^{(k)} \rightsquigarrow \nabla F(x^{(k)}) = \mathbb{E}\Big[\frac{\partial f(x^{(k)}, \boldsymbol{\xi})}{\partial x}\Big]$$

The most standard estimator being

$$\hat{g}^{(k)} = \frac{\partial f(x^{(k)}, \xi^{(k)})}{\partial x}$$

where $\xi^{(k)}$ is drawn randomly according to the law of $\boldsymbol{\xi}$ (i.e. it is a random datapoint).

Estimating the gradient

Instead of using a true gradient

$$g^{(k)} = \nabla F(x^{(k)})$$

we can use a *statistical estimator* of the gradient

$$\hat{g}^{(k)} \rightsquigarrow \nabla F(x^{(k)}) = \mathbb{E}\Big[\frac{\partial f(x^{(k)}, \boldsymbol{\xi})}{\partial x}\Big]$$

The most standard estimator being

$$\hat{g}^{(k)} = \frac{\partial f(x^{(k)}, \xi^{(k)})}{\partial x}$$

where $\xi^{(k)}$ is drawn randomly according to the law of $\boldsymbol{\xi}$ (i.e. it is a random datapoint).

Pros and Cons

Pros:

- computing $\hat{g}^{(k)} = \frac{\partial f(x^{(k)}, \xi^{(k)})}{\partial x}$ is really easy
- we do not need to spend lots of time early on to get a precise gradient
- we can stop whenever we want (do not need a full pass on the data)

Cons:

- $\hat{g}^{(k)}$ is a noisy estimator of the gradient
- requires a new convergence theory
- $x^{(k+1)} := x^{(k+1)} - \alpha \hat{g}^{(k)}$ generally does not converges almost surely to the optimal solution as this make a noisy trajectory

Noisy trajectory

- At optimality we should have $\nabla F(x^\sharp) = 0$
- It doesnot mean that $\frac{\partial f(x^\sharp, \xi^{(k)})}{\partial x}$ equals 0 !
- In particular there is no reasong for $\hat{g}^{(k)}$ to be eventually small, only its expectation should be small !

- $\rightsquigarrow$ we generally use either:
  - decreasing step e.g. $\alpha^{(k)} = \frac{\alpha^{(0)}}{k}$
  - average points $\bar{x}^{(k)} = \frac{1}{k} \sum_{\kappa \leq k} x^{(\kappa)}$

Noisy trajectory

- At optimality we should have $\nabla F(x^\sharp) = 0$
- It doesnot mean that $\frac{\partial f(x^\sharp, \xi^{(k)})}{\partial x}$ equals 0 !
- In particular there is no reasong for $\hat{g}^{(k)}$ to be eventually small, only its expectation should be small !

- $\rightsquigarrow$ we generally use either:
  - decreasing step e.g. $\alpha^{(k)} = \dfrac{\alpha^{(0)}}{k}$
  - average points $\bar{x}^{(k)} = \dfrac{1}{k} \sum_{\kappa \leq k} x^{(\kappa)}$

Mini-batch version

- $\hat{g}^{(k)} = \frac{\partial f(x^{(k)},\xi^{(k)})}{\partial x}$ is an easy to compute but noisy estimator of the gradient
- $\hat{g}^{(k)} = \frac{1}{N}\sum_{i\in[N]} \frac{\partial f(x^{(k)},\xi_i)}{\partial x}$ is a long (full batch) to compute but perfect estimator
- minibatch aims at a middle ground : randomly draw a sample S of realizations of $\boldsymbol{\xi}$, and use $\hat{g}^{(k)} = \frac{1}{|S|}\sum_{\xi\in S} \frac{\partial f(x^{(k)},\xi)}{\partial x}$

Video explanation

Videos by Andrew Ng (Former Standford professor)

- https://www.youtube.com/watch?v=W9iWNJNFzQI&list=PLWbSa0uhIdsa6wpq9s_cKOP-PjeU0aIIu&index=24 (13')
- https://www.youtube.com/watch?v=l4lSUAcvHFs&list=PLWbSa0uhIdsa6wpq9s_cKOP-PjeU0aIIu&index=25(6')

Another video with numericals tricks to improve the convergence

- https://www.youtube.com/watch?v=kK8-jCCR4is&list=PLWbSa0uhIdsa6wpq9s_cKOP-PjeU0aIIu&index=23(10')

Contents

What you have to know

- That for a stochastic problem gradient step requires to compute an expectation
- That stochastic gradient do not compute the true gradient, but only an estimator of the gradient

What you really should know

- gradient algorithm (or more advanced version) is faster in term of number of iterations
- stochastic gradient needs more iteration, but each is faster

What you have to be able to do

- dive in the scientific litterature on the subject if you need to implement this type of algorithm