Lecture 3: Optimization of Neural Networks

3 October 2022

Alessandro Betti alessandro.betti@inria.fr

Office F319, Fermat Building Inria

A plan for this

- GD: convergence analysis
- Initialization, normalization, cross-validation
- SGD in details: convergence analysis
- SGD vs Online GD
- Important Variants of GD
- Activity: experimental comparison

Notation

WARNING! In this lecture R_n will be denoted simply as f and the vector of parameters w will be denote as x. (Why? Because I can and I like it better!)

$$\begin{cases} R_n \leftrightarrow f \\ w \leftrightarrow x \end{cases}$$

- We will denote with f_i the value of the loss on the $i-th$ example of the training set as a function of the model parameters.

Convergence of GD

- If $f: \mathbb{R}^N \to \mathbb{R}$ is a $C^1(\mathbb{R}^N, \mathbb{R})$ and ∇f is L-Lipshitz, then for $0 < \tau < 2/L$ the explicit GD method

$$x^{k+1} = \begin{cases} x^0 & \text{if } k = -1 \\ x^k - \tau \nabla f(x^k) & \text{if } k \geq 0 \end{cases}$$

converges, in the sense that

$$\lim_{k \to +\infty} \|\nabla f(x^k)\| = 0$$

Stochastic Gradient Descent

- It is a method to optimize functions of the form

$$f = \frac{1}{n}\sum_{i=1}^{n} f_i,$$

  when n is large.

- The algorithm can be written as follow:
  1. Start from $x^0 \in \mathbb{R}^N$;
  2. For each $k \geq 1$ select $I_k = \{i_k^1, \ldots, i_k^m\} \subset \{1, 2, \ldots, n\}$ by sampling each i_k^j from $\{1, 2, \ldots, n\}$ independently with probability $1/n$. Then compute

$$x^{k+1} = x^k - \tau_k \sum_{i \in I_k} \nabla f_i(x^k)$$

Stochastic Gradient Descent (cont.)

- The set I_k is called a minibatch and typically $m \ll n$. An example could be $n = 10^6$ (size of ImagNet), $m = 10^3$.
- In what follows we will assume

$$I_k = \{i_k\}$$

- Usually in standard libraries I_k is chosen in a different way, by random reshuffling.

Stochastic Gradient Descent: convergence analysis

Let $f_i: \mathbb{R}^N \to \mathbb{R}$ be such that for all $i = 1, \dots, m$ the following assumptions holds

1. $f_i \in C^1(\mathbb{R}^N; \mathbb{R})$;
2. ∇f_i is L_i-Lipshitz; (smoothnes)
3. $\inf f > -\infty$;
4. $\forall x \in \mathbb{R}^N$: $\sum_{i=1}^m |\nabla f_i(x) - \nabla f(x)|^2 \leq m\sigma^2$, (bounded variance of the gradients)

where we let $f := (1/m)\sum_{i=1}^m f_i$. If $\tau_k \leq 1/\overline{L}$ for all $k \geq 0$, $\sum_{k=0}^{\infty} \tau_k = +\infty$ and $\sum_{k=1}^{+\infty} \tau_k^2 < +\infty$, then there exists a subsequence X^{i_n} such that $\nabla f(X^{i_n}) \to 0$ a. s. for large n. If in addition f is coercive X^{i_n} converges a. s. to a stationary point.

SGD vs Online Gradient

- SGD

$$x^{k+1} = x^k - \tau_k \nabla f_{i_k}(x^k),$$

with i_k chosen at random from a **fixed** training set $\{1, 2, \ldots, n\}$

- Online Gradient

$$x^{k+1} = x^k - \tau_k \nabla f_{i_k}(x^k),$$

where f_{i_k} is provided at each step by an external process without having any control on it (for instance it may be the result of a measurement made by a sensor).

Variants of GD and SGD

- We will now discuss some important modifications of the gradient rule that apply in the case of GD, SGD and Online Gradient.
- They are modification of the update rule

$$x^{k+1} = x^k - \tau g^k$$

where

$$g^k = \begin{cases} \nabla f(x^k) & \text{for GD} \\ \nabla f_{i_k}(x^k) & \text{for SGD or OGD} \end{cases}$$

Exponential Moving Average

- Consider a sequence $(a_n)_{n\geq 0}$, then an exponential moving average with discount factor α of $(a_n)_{n\geq 0}$ is the sequence $(\langle a\rangle_n^\alpha)_{n\geq 0}$ defined by the recurrence relation

$$\begin{cases} \langle a\rangle_0^\alpha = (1-\alpha)a_0 \\ \langle a\rangle_{n+1}^\alpha = \alpha\langle a\rangle_n^\alpha + (1-\alpha)a_{n+1} \end{cases}$$

- Solving the recursion we find

$$\langle a\rangle_n^\alpha = (1-\alpha)\sum_{k=0}^{n}\alpha^{k-n}a_k$$

Normalized Exponential Moving Average

- The Exponential Moving Average has the property that for α close to 1 it multiplies the first terms of the original sequence by a very small factor.
- To avoid this problem sometimes it is useful to consider a *normalized* version which we will denote as $\widehat{\langle a \rangle}_n^\alpha$, defined as

$$\widehat{\langle a \rangle}_n^\alpha := \frac{\langle a \rangle_n^\alpha}{1 - \alpha^{n+1}}, \qquad n \geq 0.$$

- Notice that indeed

$$\widehat{\langle a \rangle}_0^\alpha = a_0.$$

Heavy Ball Method: gradient descent with momentum

- Let $x^0 \in \mathbb{R}^N$ be an initial point, then set

$$\begin{cases} x^k = x^0 & \text{for } k < 0 \\ x^{k+1} \in \arg\min_{s \in \mathbb{R}^N} \left(\begin{array}{l} f(x^k) + g^k \cdot (s - x^k) \\ \quad + \dfrac{1}{2\tau}\|s - x^k\|^2 + \dfrac{1}{2\mu}\|s - 2x^k + x^{k-1}\|^2 \end{array} \right) & \text{for } k > 0 \end{cases}$$

- Can be interpreted as the explicit Euler method for solving the ODE

$$mx''(t) + \gamma x'(t) + \nabla f(x(t)) = 0,$$

  hence the name heavy ball method.

Heavy Ball Method (cont.)

- The method can be explicitly written as

$$x^{k+1} = x^k - \alpha \nabla f(x^k) + \beta(x^k - x^{k-1})$$

  with

$$\alpha = \frac{\mu\tau}{\mu + \tau}, \qquad \beta = \frac{\tau}{\mu + \tau}.$$

- The momentum term, can also be explicitly written, solving the recursion to get

$$x^{k+1} = x^k - \tau \langle g \rangle_k^\beta,$$

- $\tau = \alpha/(1-\beta)$ in this case is usually called "effective learning rate" and it is related (up to a constant) to the *terminal velocity* $1/\gamma$.

Heavy Ball Method (cont.)

- Hence the heavy ball method can obviously be described by

$$x^{k+1} \in \operatorname*{arg\,min}_{s \in \mathbb{R}^N} f(x^k) + \langle g \rangle_k^\beta \cdot (s - x^k) + \frac{1}{2\tau} \|s - x^k\|^2$$

  thus using the averages gradient $\langle g \rangle_k^\beta$ instead of the true gradient g^k to approximate f around x^k.

AdaGrad

- Take $x^0 \in \mathbb{R}^N$ as an initial point and then

$$x^{k+1} \in \arg\min_{s \in \mathbb{R}^N} f(x^k) + g^k \cdot (s - x^k) + \frac{1}{2\tau}(s - x^k) \cdot H_k^{1/2}(\beta)(s - x^k),$$

with

$$(H_k^{1/2}(\beta))_{ij} = \left(\varepsilon + \sqrt{\sum_{n=0}^{k} (g_i^k)^2}\right)\delta_{ij}, \quad i, j \in 1, \ldots, N$$

AdaGrad (cont.)

- Written explicitly the update rule is

$$x_i^{k+1} = x_i^k - \frac{\tau}{\varepsilon + \sqrt{\sum_{n=0}^{k} (g_i^k)^2}} g_i^k, \quad i = 1, \dots, N$$

- It is an adaptive learning rate method that normalize each component of the gradient
- It has been devised with the intent of penalizing the development of weights that has already changed during learning.
- In practice it also introduce ad decaying factor $1/\sqrt{k}$ of the learning rate that in online learning can be important for convergence but in general could bias the training.

RMSprop

- For every component of the gradient g_i^k, $i = 1, \ldots, N$ define the sequence of the square of this component as$(g_{i\,n}^2)_{n\geq 0}$ so that $g_{i\,n}^2 := (g_i^n)^2$
- Take $x^0 \in \mathbb{R}^N$ as an initial point and then

$$x^{k+1} \in \arg\min_{s\in\mathbb{R}^N} f(x^k) + g^k \cdot (s - x^k) + \frac{1}{2\tau}(s - x^k) \cdot \overline{H}_k^{1/2}(\beta)(s - x^k),$$

with

$$(\overline{H}_k^{1/2}(\beta))_{ij} = \left(\varepsilon + \sqrt{\sum_{n=0}^{k} \langle g_i^2 \rangle_n^\beta}\right)\delta_{ij}, \quad i, j \in 1, \ldots, N$$

RMSprop (cont.)

- Written explicitly the update rule is

$$x_i^{k+1} = x_i^k - \frac{\tau}{\varepsilon + \sqrt{\sum_{n=0}^{k} \langle g_i^2 \rangle_n^{\beta}}} g_i^k, \quad i = 1, \ldots, N$$

- It also provides a component-wise normalization of the step boosting the development of weights which have small gradient
- The use of an average instead of a sum helps to avoid the decaying behaviour that AdaGrad has.

ADAM

- Can be obtained as a specific combination of a variant of RMSprop and the heavy-ball method, namely once $x^0 \in \mathbb{R}^N$ is chosen we do

$$x^{k+1} \in \arg\min_{s\in\mathbb{R}^N} f(x^k) + \widehat{\langle g\rangle}_k^{\beta_1} \cdot (s - x^k) + \frac{1}{2\tau}(s - x^k) \cdot \widehat{\overline{H}}_k^{1/2}(\beta_2)(s - x^k),$$

  where

$$(\widehat{\overline{H}}_k^{1/2}(\beta_2))_{ij} = \left(\varepsilon + \sqrt{\sum_{n=0}^{k} \widehat{\langle g_i^2\rangle}_n^{\beta_2}}\right)\delta_{ij}, \quad i,j \in 1,\dots,N$$

- Written explicitly the update rule is

$$x_i^{k+1} = x_i^k - \frac{\tau}{\varepsilon + \sqrt{\sum_{n=0}^{k} \widehat{\langle g_i^2\rangle}_n^{\beta_2}}} \widehat{\langle g_i\rangle}_k^{\beta_1}, \quad i = 1,\dots,N$$