Statistical Machine Learning
Finding Minima Algorithms

Horacio Gómez-Acevedo
Department of Biomedical Informatics
University of Arkansas for Medical Sciences

March 8, 2023

Case $n = 1$

Let's suppose we have a real valued function which is smooth $f : \mathbb{R} \to \mathbb{R}$. Then, we can approximate the function in a vicinity of $x = c$ by the so-called Taylor expansion

$$f(x) = f(c) + \frac{df}{dx}(c)(x-c) + \frac{1}{2!}\frac{d^2 f}{dx^2}(c)(x-c)^2 + \frac{1}{3!}\frac{d^3 f}{dx^3}(c)(x-c)^3 + \cdots$$

Notice that when x is very close to c, then $(x - c)^p$ for $p \geq 3$ starts getting exceedingly small. Then, we write

$$f(x) \approx f(c) + \frac{df}{dx}(c)(x - c) + \frac{1}{2!}\frac{d^2 f}{dx^2}(c)(x - c)^2$$

If at the point $x = c$ the function reaches a (local) minimum, then $f'(c) = 0$.

Case $n \geq 2$

Let's suppose we have a real valued function which is smooth $f : \mathbb{R}^n \to \mathbb{R}$. Then, we can approximate the function in a vicinity of $x = c$ by the so-called Taylor expansion

$$\begin{aligned}
f(\mathbf{x}) =& f(\mathbf{c}) + \sum_i \frac{\partial f}{\partial x_i}(c)(x_i - c_i) + \frac{1}{2!} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j}(c)(x_i - c_i)(x_j - c_j) + \\
&+ \text{ higher order terms} \\
\approx& f(\mathbf{c}) + (\mathbf{x} - \mathbf{c})^T \nabla f(\mathbf{c}) + \frac{1}{2}(\mathbf{x} - \mathbf{c})^T H_f(\mathbf{c})(\mathbf{x} - \mathbf{c})
\end{aligned}$$

Where $H_f(\mathbf{c})$ is the **Hessian matrix** defined as

$$H_f(\mathbf{c}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(\mathbf{c}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{c}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{c}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{c}) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(\mathbf{c}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\mathbf{c}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{c}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\mathbf{c}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(\mathbf{c}) \end{pmatrix}$$

Minimum Condition

For the case $n = 1$, the point $x = c$ if $f'(c) = 0$ and $f''(c) < 0$, then the function f has a **local minimum** at c.
For the case when $n \geq 2$, if $\nabla f(\mathbf{c}) = 0$ and that H_f satisfies for points close to **c**

$$v^T H_f v > 0 \; \forall v \in \mathbb{R}^n - \{0\}$$

Then, **c** is a **local minimum** of f.

Directional Derivatives

The measure of the instantaneously rate of change of a function at a point in a given direction. Let $f : \mathbb{R}^2 \to \mathbb{R}$ a differentiable function and v a unitary vector in $\mathbb{R}^n$ (i.e., $\|v\| = 1$).

$$\begin{aligned} \frac{\partial f}{\partial v}(x^0) &= \lim_{t \to 0^+} \frac{f(x^0 + tv) - f(x^0)}{t} \\ &= v^T \nabla f(x^0) = \langle \nabla f(x^0), v \rangle \end{aligned}$$

We can write our approximation as

$$f(x^0 + \eta v) - f(x^0) = \eta \langle \nabla f(x^0), v \rangle + \text{ higher order terms}$$

Gradient Descent Algorithm

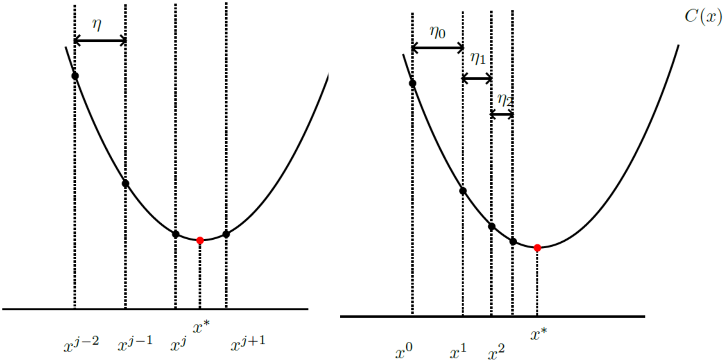
It can be proven that the largest change in the function f is approximately equal to

$$f(x^0 + \eta v) - f(x^0) = -\eta \|\nabla f(x^0)\|$$

where η is the **learning rate**
The algorithm build iteratively a sequence (x^n) of vectors that will approach to the (hopefully) global minimum x^* as follows :

- Choose an initial point x^0 in the basin of attraction of x^*
- Construct
$$x^{n+1} = x^n - \eta \frac{\nabla f(x^n)}{\|\nabla f(x^n)\|}$$

Changing the learning rate



AdaGrad

In 2011, Duchi et al. described a modified version of the gradient descent called **Adaptive Gradient**.
If $C(x)$ denotes the cost function ($x \in \mathbb{R}^N$), then the gradient vector $g_t = \nabla C(x(t))$. Let G_t denote the $N \times N$ matrix

$$G_t = \sum_{i=1}^{t} g_i g_i^T$$

and consider the update

$$x(t+1) = x(t) - \eta G_t^{-1/2} g_t$$

Since $G_t^{-1/2}$ is computationally impractical in high dimension, the update can be done using only the diagonal elements of the matrix

$$x(t+1) = x(t) - \eta \mathrm{diag}(G_t)^{-1/2} g_t$$

AdaGrad (cont)

The diagonal elements of G_t can be calculated by

$$(G_t)_{jj} = \sum_{k=1}^{t} (g_{kj})^2$$

where

$$g_j g_j^T = \begin{pmatrix} g_{j1} \\ \vdots \\ g_{jN} \end{pmatrix} (g_{j1}, \dots, g_{jN}) = \begin{pmatrix} (g_{j1})^2 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & (g_{jN})^2 \end{pmatrix}$$

RMSProp

The **Root Mean Square Propagation** or RMSProp is a variant of the gradient descent method with adaptive learning rate, which is obtained when the gradient is divided by a running average of its magnitude.
Let $C(x)$ denote the cost function, and $g_t = \nabla C(x(t))$ is the gradient evaluated at time step t. Then, the running average is defined recursively by

$$v(t) = \gamma v(t-1) + (1-\gamma) g_{t-1}^2,$$

where $\gamma \in (0,1)$ is called the **forgetting factor**, and the vector g_{t-1}^2 denotes the element-wise square of the gradient g_{t-1}.
It can be shown that $v(t)$ can be expressed as

$$v(t) = \gamma^t v(0) + (1-\gamma) \sum_{j=1}^{t} \gamma^{t-j} g_j^2$$

RMSProp

The minimum of the cost function $x^* = \arg\min_x C(x)$ is obtained by the approximation sequence $(x(t))_{t\geq 1}$ defined as

$$x(t+1) = x(t) - \eta \frac{g_t}{\sqrt{|v(t)|}}$$

where η is the learning rate.
One interpretation of v is that it represents an estimation of the (uncentered) variance of the gradient.

Adam

The **Adaptive Moment Estimation** or Adam is another adaptive learning method.

Fact

Recall that for a random variable X the first and second moments are defined as $\mathbb{E}(X)$ and $\mathbb{E}(X^2)$ respectively.

The objective is to minimize the expectation of the cost function. Namely, we look for the minimum value x^* that satisfies

$$x^* = \arg\min_x \mathbb{E}(C(x))$$

In this case, we consider two exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$, and the moments updates

$$\begin{aligned} m(t) &= \beta_1 m(t-1) + (1-\beta_1) g_t \\ v(t) &= \beta_2 v(t-1) + (1-\beta_2)(g_t)^2 \end{aligned}$$

where $m(0) = v(0) = 0$.

Adam (cont)

We can write

$$m(t) = (1 - \beta_1) \sum_{j=1}^{t} \beta_1^{t-j} g_j$$

$$v(t) = (1 - \beta_2) \sum_{j=1}^{t} \beta_2^{t-j} (g_j)^2$$

Then, if we assume that the first and second moments are stationary we get

$$\mathbb{E}(m(t)) = (1 - \beta_1^t)\mathbb{E}(g_t)$$

$$\mathbb{E}(v(t)) = (1 - \beta_2^t)\mathbb{E}(g_t)^2$$

Therefore, the bias-corrected moments are

$$\widehat{m}(t) = \frac{m(t)}{1 - \beta_1^t}$$

$$\widehat{v}(t) = \frac{v(t)}{1 - \beta_2^t}$$

Adam (cont)

Finally, the recursive formula becomes

$$x(t+1) = x(t) - \eta \frac{\widehat{m}(t)}{\sqrt{|\widehat{v}(t)|} + \varepsilon}$$

the $\varepsilon > 0$ is a small scalar used to prevent division by zero.

References

Materials and some of the pictures are from (Calin, 2019).

Calin, O. (2019). *Deep Learning Architectures*. Springer Series in the Data Sciences. Springer. ISBN: 978-3-030-36723-7.

I have used some of the graphs by hacking TiKz code from StakExchange, Inkscape for more aesthetic plots and other old tricks of TEX