WIKIPEDIA

Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as **gradient ascent**.

Gradient descent is generally attributed to Cauchy, who first suggested it in 1847.[1] Hadamard independently proposed a similar method in 1907.[2][3] Its convergence properties for non-linear optimization problems were first studied by Haskell Curry in 1944,[4] with the method becoming increasingly well-studied and used in the following decades,[5][6] also often called **steepest descent.**

Contents

Description

Gradient descent is based on the observation that if the multi-variable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(\mathbf{x})$ decreases *fastest* if one goes from $\mathbf{a}$ in the direction of the negative gradient of F at $\mathbf{a}, -\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

for a $\gamma \in \mathbb{R}_+$ small enough, then $F(\mathbf{a_n}) \geq F(\mathbf{a_{n+1}})$. In other words, the term $\gamma \nabla F(\mathbf{a})$ is subtracted from $\mathbf{a}$ because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0.$$

We have a monotonic sequence

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

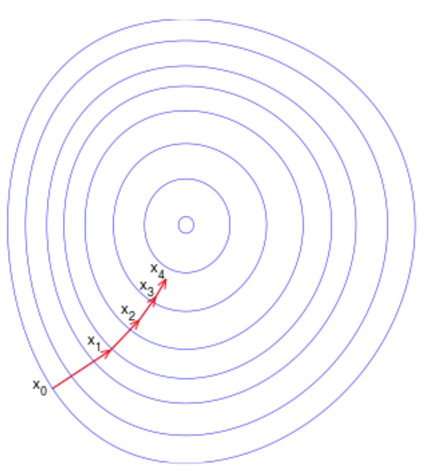


Illustration of gradient descent on a series of level sets

so hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum. Note that the value of the *step size* γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen either via a line search that satisfies the Wolfe conditions, or the Barzilai–Borwein method[7][8] shown as following),

$$\gamma_n = \frac{\left|(\mathbf{x}_n - \mathbf{x}_{n-1})^T \left[\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\right]\right|}{\|\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\|^2}$$

convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in the adjacent picture. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

An analogy for understanding gradient descent

The basic intuition behind gradient descent can be illustrated by a hypothetical scenario. A person is stuck in the mountains and is trying to get down (i.e., trying to find the global minimum). There is heavy fog such that visibility is extremely low. Therefore, the path down the mountain is not visible, so they must use local information to find the minimum. They can use the method of gradient descent, which involves looking at the steepness of the hill at their current position, then proceeding in the direction with the steepest descent (i.e., downhill). If they were trying to find the top of the mountain (i.e., the maximum), then they would proceed in the direction of steepest ascent (i.e., uphill). Using this method, they would eventually find their way down the mountain or possibly get stuck in some hole (i.e., local minimum or saddle point), like a mountain lake. However, assume also that the steepness of the hill is not immediately obvious with simple observation, but rather it requires a sophisticated instrument to measure, which the person happens to have at the moment. It takes quite some time to measure the steepness of the hill with the instrument, thus they should minimize their use of the instrument if they wanted to get down the mountain before sunset. The difficulty then is choosing the frequency at which they should measure the steepness of the hill so not to go off track.

In this analogy, the person represents the algorithm, and the path taken down the mountain represents the sequence of parameter settings that the algorithm will explore. The steepness of the hill represents the slope of the error surface at that point. The instrument used to measure steepness is differentiation (the slope of the error surface can be calculated by taking the derivative of the squared error function at that point). The direction they choose to travel in aligns with the gradient of the error surface at that point. The amount of time they travel before taking another measurement is the step size.

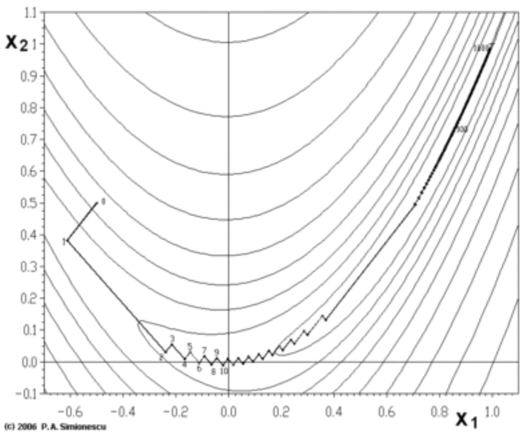
Fog in the mountains

Examples

Gradient descent has problems with pathological functions such as the Rosenbrock function shown here.

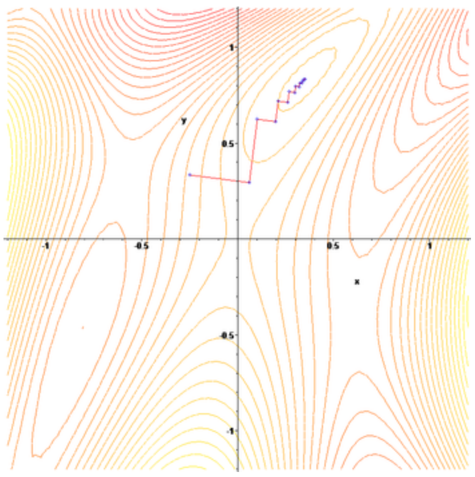
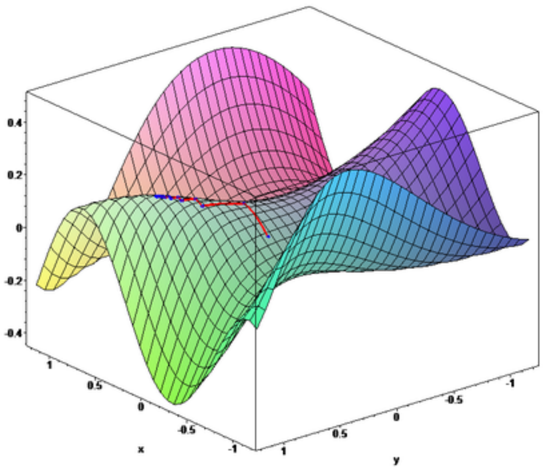
$$f(x_1,x_2)=(1-x_1)^2+100(x_2-x_1^2)^2.$$

The Rosenbrock function has a narrow curved valley which contains the minimum. The bottom of the valley is very flat. Because of the curved flat valley the optimization is zigzagging slowly with small step sizes towards the minimum.



The zigzagging nature of the method is also evident below, where the gradient descent method is applied to

$$F(x,y)=\sin\left(\frac{1}{2}x^2-\frac{1}{4}y^2+3\right)\cos(2x+1-e^y).$$

Choosing the step size and descent direction

Since using a step size γ that is too small would slow convergence, and a γ too large would lead to divergence, finding a good setting of γ is an important practical problem. Philip Wolfe also advocated for using "clever choices of the [descent] direction" in practice.[9] Whilst using a direction that deviates from the steepest descent direction may seem counter-intuitive, the idea is that the smaller slope may be compensated for by being sustained over a much longer distance.

To reason about this mathematically, let's use a direction $\mathbf{p}_n$ and step size γ_n and consider the more general update:

$$\mathbf{a}_{n+1}=\mathbf{a}_n-\gamma_n\,\mathbf{p}_n.$$

Finding good settings of $\mathbf{p}_n$ and γ_n requires a little thought. First of all, we would like the update direction to point downhill. Mathematically, letting θ_n denote the angle between $\nabla F(\mathbf{a_n})$ and $\mathbf{p}_n$, this requires that $\cos\theta_n>0.$ To say more, we need more information about the objective function that we are optimising. Under the fairly weak assumption that F is continuously differentiable, we may prove that:[10]

$$F(\mathbf{a}_{n+1})\leq F(\mathbf{a}_n)-\gamma_n\|\nabla F(\mathbf{a}_n)\|_2\|\mathbf{p}_n\|_2\left[\cos\theta_n-\max_{t\in[0,1]}\frac{\|\nabla F(\mathbf{a}_n-t\gamma_n\mathbf{p}_n)-\nabla F(\mathbf{a}_n)\|_2}{\|\nabla F(\mathbf{a}_n)\|_2}\right]$$

This inequality implies that the amount by which we can be sure the function F is decreased depends on a trade off between the two terms in square brackets. The first term in square brackets measures the angle between the descent direction and the negative gradient. The second term measures how quickly the gradient changes along the descent direction.

In principle inequality ($\mathbf{1}$) could be optimized over $\mathbf{p}_n$ and γ_n to choose an optimal step size and direction. The problem is that evaluating the second term in square brackets requires evaluating $\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n)$, and extra gradient evaluations are generally expensive and undesirable. Some ways around this problem are:

- Forgo the benefits of a clever descent direction by setting $\mathbf{p}_n = \nabla F(\mathbf{a_n})$, and use line search to find a suitable step-size γ_n, such as one that satisfies the Wolfe conditions.
- Assuming that F is twice-differentiable, use its Hessian $\nabla^2 F$ to estimate $\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2 \approx \|t\gamma_n \nabla^2 F(\mathbf{a}_n)\mathbf{p}_n\|$.Then choose $\mathbf{p}_n$ and γ_n by optimising inequality ($\mathbf{1}$).
- Assuming that ∇F is Lipschitz, use its Lipschitz constant L to bound $\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2 \leq Lt\gamma_n \|\mathbf{p}_n\|$. Then choose $\mathbf{p}_n$ and γ_n by optimising inequality ($\mathbf{1}$).
- Build a custom model of $\max\limits_{t\in[0,1]} \dfrac{\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2}{\|\nabla F(\mathbf{a}_n)\|_2}$ for F. Then choose $\mathbf{p}_n$ and γ_n by optimising inequality ($\mathbf{1}$).
- Under stronger assumptions on the function F such as convexity, more advanced techniques may be possible.

Usually by following one of the recipes above, convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

Solution of a linear system

Gradient descent can be used to solve a system of linear equations

$$A\mathbf{x} - \mathbf{b} = 0$$

reformulated as a quadratic minimization problem. If the system matrix A is real symmetric and positive-definite, the quadratic function to minimize commonly is

$$F(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} - 2\mathbf{x}^T \mathbf{b},$$

so that

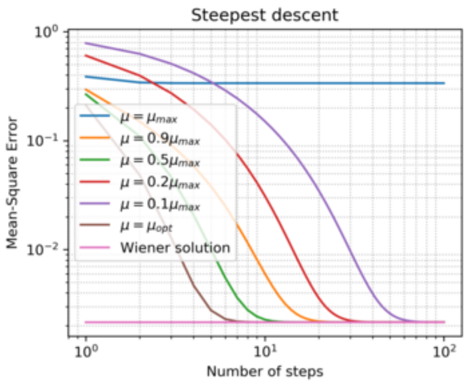
$$\nabla F(\mathbf{x}) = 2(A\mathbf{x} - \mathbf{b}).$$

For a general real matrix A, linear least squares define

$$F(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2.$$

In traditional linear least squares for real A and $\mathbf{b}$ the Euclidean norm is used, in which case

$$\nabla F(\mathbf{x}) = 2A^T(A\mathbf{x} - \mathbf{b}).$$



The steepest descent algorithm applied to the Wiener filter[11]

The line search minimization, finding the locally optimal step size γ on every iteration, can be performed analytically for quadratic functions, and explicit formulas for the locally optimal γ are known.[5][12]

For example, for real symmetric and positive-definite matrix A, a simple algorithm can be as follows,[5]

```
repeat in the loop:
    $\mathbf{r} := \mathbf{b} - \mathbf{Ax}$
    $\gamma := \mathbf{r}^\mathsf{T}\mathbf{r}/\mathbf{r}^\mathsf{T}\mathbf{Ar}$
    $\mathbf{x} := \mathbf{x} + \gamma\mathbf{r}$
    if $\mathbf{r}^\mathsf{T}\mathbf{r}$ is sufficiently small, then exit loop
end repeat loop
return $\mathbf{x}$ as the result
```

To avoid multiplying by A twice per iteration, we note that $\mathbf{x} := \mathbf{x} + \gamma\mathbf{r}$ implies $\mathbf{r} := \mathbf{r} - \gamma\mathbf{Ar}$, which gives the traditional algorithm,[13]

```
$\mathbf{r} := \mathbf{b} - \mathbf{Ax}$
repeat in the loop:
    $\gamma := \mathbf{r}^\mathsf{T}\mathbf{r}/\mathbf{r}^\mathsf{T}\mathbf{Ar}$
    $\mathbf{x} := \mathbf{x} + \gamma\mathbf{r}$
    if $\mathbf{r}^\mathsf{T}\mathbf{r}$ is sufficiently small, then exit loop
    $\mathbf{r} := \mathbf{r} - \gamma\mathbf{Ar}$
end repeat loop
return $\mathbf{x}$ as the result
```

The method is rarely used for solving linear equations, with the conjugate gradient method being one of the most popular alternatives. The number of gradient descent iterations is commonly proportional to the spectral condition number $\kappa(A)$ of the system matrix A (the ratio of the maximum to minimum eigenvalues of $A^T A$), while the convergence of conjugate gradient method is typically determined by a square root of the condition number, i.e., is much faster. Both methods can benefit from preconditioning, where gradient descent may require less assumptions on the preconditioner.[13]

Solution of a non-linear system

Gradient descent can also be used to solve a system of nonlinear equations. Below is an example that shows how to use the gradient descent to solve for three unknown variables, x_1, x_2, and x_3. This example shows one iteration of the gradient descent.

Consider the nonlinear system of equations

$$\begin{cases} 3x_1 - \cos(x_2x_3) - \tfrac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1x_2) + 20x_3 + \tfrac{10\pi-3}{3} = 0 \end{cases}$$

Let us introduce the associated function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2x_3) - \tfrac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1x_2) + 20x_3 + \tfrac{10\pi-3}{3} \end{bmatrix},$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

One might now define the objective function

$$F(\mathbf{x}) = \frac{1}{2}G^{\mathrm{T}}(\mathbf{x})G(\mathbf{x}) = \frac{1}{2}\left[\left(3x_1 - \cos(x_2x_3) - \frac{3}{2}\right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1\right)^2 + \left(\exp(-x_1x_2) + 20x_3 + \frac{10\pi-3}{3}\right)^2\right],$$

which we will attempt to minimize. As an initial guess, let us use

$$\mathbf{x}^{(0)} = \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{0} - \gamma_0 \nabla F(\mathbf{0}) = \mathbf{0} - \gamma_0 J_G(\mathbf{0})^{\mathrm{T}} G(\mathbf{0}),$$

where the Jacobian matrix J_G is given by

$$J_G(\mathbf{x}) = \begin{bmatrix} 3 & \sin(x_2x_3)x_3 & \sin(x_2x_3)x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2\exp(-x_1x_2) & -x_1\exp(-x_1x_2) & 20 \end{bmatrix}.$$

We calculate:

$$J_G(\mathbf{0}) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}, \qquad G(\mathbf{0}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}.$$

Thus

$$\mathbf{x}^{(1)} = \mathbf{0} - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix},$$

and

$$F(\mathbf{0}) = 0.5\left((-2.5)^2 + (-1)^2 + (10.472)^2\right) = 58.456.$$

Now, a suitable γ_0 must be found such that

$$F\left(\mathbf{x}^{(1)}\right) \le F\left(\mathbf{x}^{(0)}\right) = F(\mathbf{0}).$$

This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001,$ which gives

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}.$$

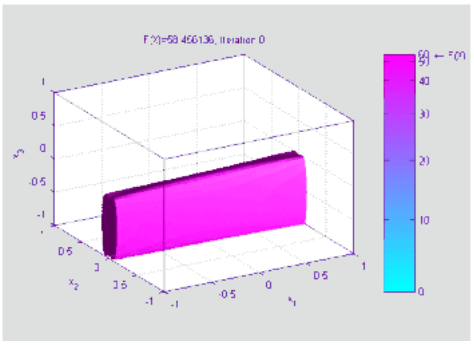
Evaluating the objective function at this value, yields

$$F\left(\mathbf{x}^{(1)}\right) = 0.5\left((-2.48)^2 + (-1.00)^2 + (6.28)^2\right) = 23.306.$$

The decrease from $F(\mathbf{0}) = 58.456$ to the next step's value of

$$F\left(\mathbf{x}^{(1)}\right) = 23.306$$

is a sizable decrease in the objective function. Further steps would reduce its value further, until an approximate solution to the system was found.



An animation showing the first 83 iterations of gradient descent applied to this example. Surfaces are isosurfaces of $F(\mathbf{x}^{(n)})$ at current guess $\mathbf{x}^{(n)}$, and arrows show the direction of descent. Due to a small and constant step size, the convergence is slow.

Comments

Gradient descent works in spaces of any number of dimensions, even in infinite-dimensional ones. In the latter case, the search space is typically a function space, and one calculates the Fréchet derivative of the functional to be minimized to determine the descent direction.[6]

That gradient descent works in any number of dimensions (finite number at least) can be seen as a consequence of the Cauchy-Schwarz inequality. That article proves that the magnitude of the inner (dot) product of two vectors of any dimension is maximized when they are colinear. In the case of gradient descent, that would be when the vector of independent variable adjustments is proportional to the gradient vector of partial derivatives.

The gradient descent can take many iterations to compute a local minimum with a required accuracy, if the curvature in different directions is very different for the given function. For such functions, preconditioning, which changes the geometry of the space to shape the function level sets like concentric circles, cures the slow convergence. Constructing and applying preconditioning can be computationally expensive, however.

The gradient descent can be combined with a line search, finding the locally optimal step size γ on every iteration. Performing the line search can be time-consuming. Conversely, using a fixed small γ can yield poor convergence.

Methods based on Newton's method and inversion of the Hessian using conjugate gradient techniques can be better alternatives.[14][15] Generally, such methods converge in fewer iterations, but the cost of each iteration is higher. An example is the BFGS method which consists in calculating on every step a matrix by which the gradient vector is multiplied to go into a "better" direction, combined with a more sophisticated line search algorithm, to find the "best" value of γ. For extremely large problems, where the computer-memory issues dominate, a limited-memory method such as L-BFGS should be used instead of BFGS or the steepest descent.

Gradient descent can be viewed as applying Euler's method for solving ordinary differential equations $x'(t) = -\nabla f(x(t))$ to a gradient flow. In turn, this equation may be derived as an optimal controller[16] for the control system $x'(t) = u(t)$ with $u(t)$ given in feedback form $u(t) = -\nabla f(x(t))$.

Modifications

Gradient descent can converge to a local minimum and slow down in a neighborhood of a saddle point. Even for unconstrained quadratic minimization, gradient descent develops a zig-zag pattern of subsequent iterates as iterations progress, resulting in slow convergence. Multiple modifications of gradient descent have been proposed to address these deficiencies.

Fast gradient methods

Yurii Nesterov has proposed[17] a simple modification that enables faster convergence for convex problems and has been since further generalized. For unconstrained smooth problems the method is called the fast gradient method (FGM) or the accelerated gradient method (AGM). Specifically, if the differentiable function F is convex and ∇F is Lipschitz, and it is not assumed that F is strongly convex, then the error in the objective value generated at each step k by the gradient descent method will be bounded by $\mathcal{O}\left(\frac{1}{k}\right)$. Using the Nesterov acceleration technique, the error decreases at $\mathcal{O}\left(\frac{1}{k^2}\right)$.[18] It is known that the rate $\mathcal{O}\left(k^{-2}\right)$ for the decrease of the cost function is optimal for first-order optimization methods. Nevertheless, there is the opportunity to improve the algorithm by reducing the constant factor. The optimized gradient method (OGM)[19] reduces that constant by a factor of two and is an optimal first-order method for large-scale problems.[20]

For constrained or non-smooth problems, Nesterov's FGM is called the fast proximal gradient method (FPGM), an acceleration of the proximal gradient method.

Momentum or *heavy ball* method

Trying to break the zig-zag pattern of gradient descent, the *momentum or heavy ball method* uses a momentum term in analogy to a heavy ball sliding on the surface of values of the function being minimized,[5] or to mass movement in Newtonian dynamics through a viscous medium in a conservative force field.[21] Gradient descent with momentum remembers the solution update at each iteration, and determines the next update as a linear combination of the gradient and the previous update. For unconstrained quadratic minimization, a theoretical convergence rate bound of the heavy ball method is asymptotically the same as that for the optimal conjugate gradient method.[5]

This technique is used in Stochastic gradient descent#Momentum and as an extension to the backpropagation algorithms used to train artificial neural networks.[22][23]

Extensions

Gradient descent can be extended to handle constraints by including a projection onto the set of constraints. This method is only feasible when the projection is efficiently computable on a computer. Under suitable assumptions, this method converges. This method is a specific case of the forward-backward algorithm for monotone inclusions (which includes convex programming and variational inequalities).[24]

See also

- Backtracking line search
- Conjugate gradient method
- Stochastic gradient descent
- Rprop
- Delta rule
- Wolfe conditions
- Preconditioning
- Broyden–Fletcher–Goldfarb–Shanno algorithm
- Davidon–Fletcher–Powell formula
- Nelder–Mead method
- Gauss–Newton algorithm
- Hill climbing
- Quantum annealing

References

1. Lemaréchal, C. (2012). "Cauchy and the Gradient Method" (https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf) (PDF). *Doc Math Extra*: 251–254.
2. Hadamard, Jacques (1908). "Mémoire sur le problème d'analyse relatif à Véquilibre des plaques élastiques encastrées". *Mémoires présentés par divers savants éstrangers à l'Académie des Sciences de l'Institut de France*. **33**.
3. Courant, Richard (January 1943). "Variational methods for the solution of problems of equilibrium and vibrations" (https://projecteuclid.org/journals/bulletin-of-the-american-mathematical-society-new-series/volume-49/issue-1/Variational-methods-for-the-solution-of-problems-of-equilibrium-and/bams/1183504922.full). *Bull. Amer. Math. Soc.* **49**: 1–23. doi:10.1090/S0002-9904-1943-07818-4 (https://doi.org/10.1090%2FS0002-9904-1943-07818-4) – via Project Euclid.

4. Curry, Haskell B. (1944). "The Method of Steepest Descent for Non-linear Minimization Problems" (https://doi.org/10.1090%2Fqam%2F10667). *Quart. Appl. Math*. **2** (3): 258–261. doi:10.1090/qam/10667 (https://doi.org/10.1090%2Fqam%2F10667).
5. Polyak, Boris (1987). *Introduction to Optimization* (https://www.researchgate.net/publication/342978480).
6. Akilov, G. P.; Kantorovich, L. V. (1982). *Functional Analysis* (2nd ed.). Pergamon Press. ISBN 0-08-023036-9.
7. Barzilai, Jonathan; Borwein, Jonathan M. (1988). "Two-Point Step Size Gradient Methods". *IMA Journal of Numerical Analysis*. **8** (1): 141–148. doi:10.1093/imanum/8.1.141 (https://doi.org/10.1093%2Fimanum%2F8.1.141).
8. Fletcher, R. (2005). "On the Barzilai–Borwein Method". In Qi, L.; Teo, K.; Yang, X. (eds.). *Optimization and Control with Applications*. Applied Optimization. **96**. Boston: Springer. pp. 235–256. ISBN 0-387-24254-6.
9. Wolfe, Philip (1969-04-01). "Convergence Conditions for Ascent Methods" (https://doi.org/10.1137/1011036). *SIAM Review*. **11** (2): 226–235. doi:10.1137/1011036 (https://doi.org/10.1137%2F1011036). ISSN 0036-1445 (https://www.worldcat.org/issn/0036-1445).
10. Bernstein, Jeremy; Vahdat, Arash; Yue, Yisong; Liu, Ming-Yu (2020-06-12). "On the distance between two neural networks and the stability of learning". arXiv:2002.03432 (https://arxiv.org/abs/2002.03432) [cs.LG (https://arxiv.org/archive/cs.LG)].
11. Haykin, Simon S. Adaptive filter theory. Pearson Education India, 2008. - p. 108-142, 217-242
12. Saad, Yousef (2003). *Iterative methods for sparse linear systems* (https://archive.org/details/iterativemethods0000saad/page/195) (2nd ed.). Philadelphia, Pa.: Society for Industrial and Applied Mathematics. pp. 195 (https://archive.org/details/iterativemethods0000saad/page/195). ISBN 978-0-89871-534-7.
13. Bouwmeester, Henricus; Dougherty, Andrew; Knyazev, Andrew V. (2015). "Nonsymmetric Preconditioning for Conjugate Gradient and Steepest Descent Methods" (https://doi.org/10.1016%2Fj.procs.2015.05.241). *Procedia Computer Science*. **51**: 276–285. doi:10.1016/j.procs.2015.05.241 (https://doi.org/10.1016%2Fj.procs.2015.05.241).
14. Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (https://archive.org/details/numericalrecipes00pres_0) (2nd ed.). New York: Cambridge University Press. ISBN 0-521-43108-5.
15. Strutz, T. (2016). *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond* (2nd ed.). Springer Vieweg. ISBN 978-3-658-11455-8.
16. Ross, I. M. (2019-07-01). "An optimal control theory for nonlinear optimization" (http://www.sciencedirect.com/science/article/pii/S0377042719300019). *Journal of Computational and Applied Mathematics*. **354**: 39–51. doi:10.1016/j.cam.2018.12.044 (https://doi.org/10.1016%2Fj.cam.2018.12.044). ISSN 0377-0427 (https://www.worldcat.org/issn/0377-0427).
17. Nesterov, Yu. (2004). *Introductory Lectures on Convex Optimization : A Basic Course*. Springer. ISBN 1-4020-7553-7.
18. Vandenberghe, Lieven (2019). "Fast Gradient Methods" (https://www.seas.ucla.edu/~vandenbe/236C/lectures/fgrad.pdf) (PDF). *Lecture notes for EE236C at UCLA*.
19. Kim, D.; Fessler, J. A. (2016). "Optimized First-order Methods for Smooth Convex Minimization" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5067109). *Math. Prog.* **151** (1–2): 81–107. arXiv:1406.5468 (https://arxiv.org/abs/1406.5468). doi:10.1007/s10107-015-0949-3 (https://doi.org/10.1007%2Fs10107-015-0949-3). PMC 5067109 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5067109). PMID 27765996 (https://pubmed.ncbi.nlm.nih.gov/27765996). S2CID 207055414 (https://api.semanticscholar.org/CorpusID:207055414).
20. Drori, Yoel (2017). "The Exact Information-based Complexity of Smooth Convex Minimization". *Journal of Complexity*. **39**: 1–16. arXiv:1606.01424 (https://arxiv.org/abs/1606.01424). doi:10.1016/j.jco.2016.11.001 (https://doi.org/10.1016%2Fj.jco.2016.11.001). S2CID 205861966 (https://api.semanticscholar.org/CorpusID:205861966).
21. Qian, Ning (January 1999). "On the momentum term in gradient descent learning algorithms" (https://web.archive.org/web/20140508200053/http://brahms.cpmc.columbia.edu/publications/momentum.pdf) (PDF). *Neural Networks*. **12** (1): 145–151. CiteSeerX 10.1.1.57.5612 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5612). doi:10.1016/S0893-6080(98)00116-6 (https://doi.org/10.1016%2FS0893-6080%2898%2900116-6). PMID 12662723 (https://pubmed.ncbi.nlm.nih.gov/12662723). Archived from the original (http://brahms.cpmc.columbia.edu/publications/momentum.pdf) (PDF) on 8 May 2014. Retrieved 17 October 2014.
22. "Momentum and Learning Rate Adaptation" (http://www.willamette.edu/~gorr/classes/cs449/momrate.html). Willamette University. Retrieved 17 October 2014.
23. Geoffrey Hinton; Nitish Srivastava; Kevin Swersky. "The momentum method" (https://www.coursera.org/lecture/neural-networks/the-momentum-method-Oya9a). *Coursera*. Retrieved 2 October 2018. Part of a lecture series for the Coursera online course Neural Networks for Machine Learning (https://www.coursera.org/learn/neural-networks) Archived (https://web.archive.org/web/20161231174321/https://www.coursera.org/learn/neural-networks) 2016-12-31 at the Wayback Machine.
24. Combettes, P. L.; Pesquet, J.-C. (2011). "Proximal splitting methods in signal processing". In Bauschke, H. H.; Burachik, R. S.; Combettes, P. L.; Elser, V.; Luke, D. R.; Wolkowicz, H. (eds.). *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. New York: Springer. pp. 185–212. arXiv:0912.3522 (https://arxiv.org/abs/0912.3522). ISBN 978-1-4419-9568-1.

Further reading

- Boyd, Stephen; Vandenberghe, Lieven (2004). "Unconstrained Minimization" (https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf#page=471) (PDF). *Convex Optimization*. New York: Cambridge University Press. pp. 457–520. ISBN 0-521-83378-7.
- Chong, Edwin K. P.; Żak, Stanislaw H. (2013). "Gradient Methods" (https://www.google.com/books/edition/An_Introduction_to_Optimization/iD5s0iKXHP8C?hl=en&gbpv=1&pg=PA131). *An Introduction to Optimization* (Fourth ed.). Hoboken: Wiley. pp. 131–160. ISBN 978-1-118-27901-4.
- Himmelblau, David M. (1972). "Unconstrained Minimization Procedures Using Derivatives". *Applied Nonlinear Programming*. New York: McGraw-Hill. pp. 63–132. ISBN 0-07-028921-2.

External links

- Using gradient descent in C++, Boost, Ublas for linear regression (http://codingplayground.blogspot.it/2013/05/learning-linear-regression-with.html)
- Series of Khan Academy videos discusses gradient ascent (https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/gradient-and-directional-derivatives/v/gradient)
- Online book teaching gradient descent in deep neural network context (http://neuralnetworksanddeeplearning.com/chap1.html#learning_with_gradient_descent)
- "Gradient Descent, How Neural Networks Learn" (https://www.youtube.com/watch?v=IHZwWFHWa-w&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=2). *3Blue1Brown*. October 16, 2017 – via YouTube.