Gradient descent

From Wikipedia, the free encyclopedia

Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.

Gradient descent is also known as **steepest descent**, or the **method of steepest descent**. Gradient descent should not be confused with the method of steepest descent for approximating integrals.

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

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

for γ small enough, then $F(\mathbf{a}) \geq F(\mathbf{b})$. In other words, the term $\gamma \nabla F(\mathbf{a})$ is subtracted from $\mathbf{a}$ because we want to move against the gradient, namely down toward the 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

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

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 via a line search that satisfies the Wolfe conditions), 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.

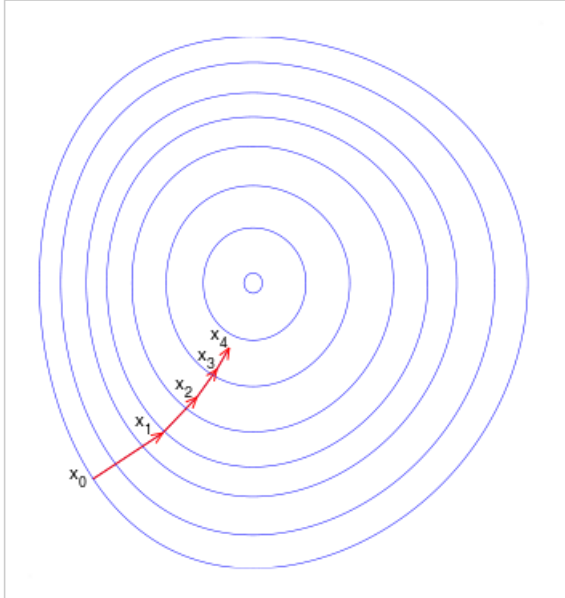


Illustration of gradient descent on a series of level sets.

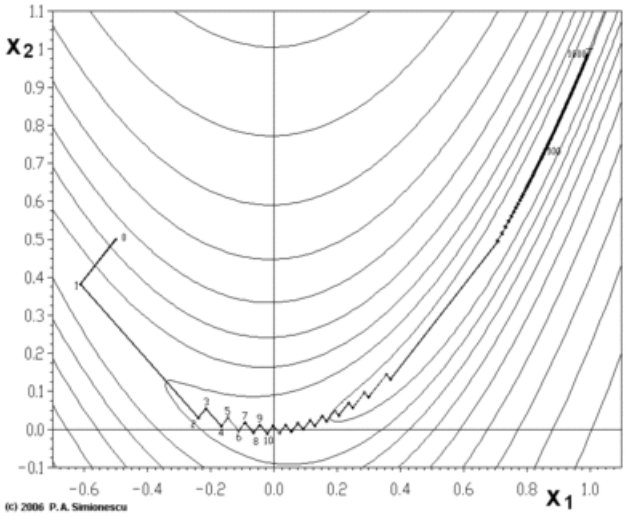
This process is illustrated in the picture to the right. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative 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.

Examples

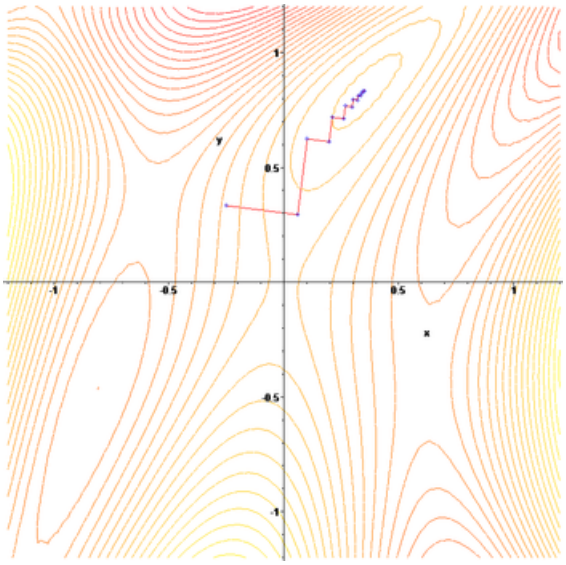
Gradient descent has problems with pathological functions such as the Rosenbrock function shown here.

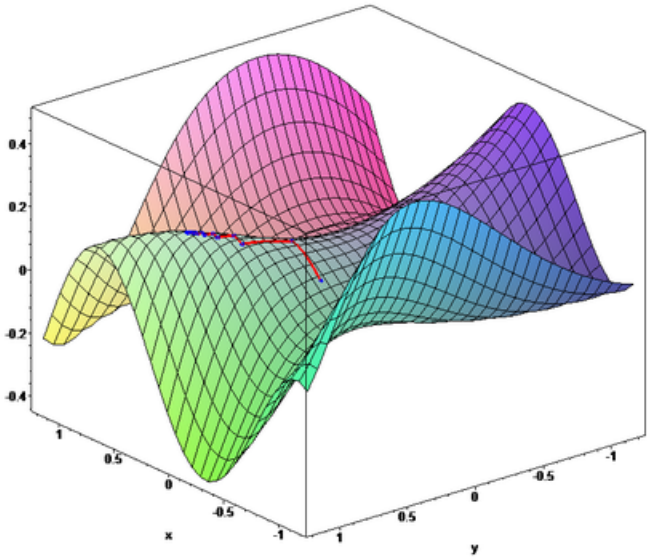
$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

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 zig-zagging slowly with small stepsizes towards the minimum.



The "Zig-Zagging" nature of the method is also evident below, where the gradient descent method is applied to
$F(x, y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)\cos(2x + 1 - e^y)$.





Limitations

For some of the above examples, gradient descent is relatively slow close to the minimum: technically, its asymptotic rate of convergence is inferior to many other methods. For poorly conditioned convex problems, gradient descent increasingly 'zigzags' as the gradients point nearly orthogonally to the shortest direction to a minimum point. For more details, see the comments below.

For non-differentiable functions, gradient methods are ill-defined. For locally Lipschitz problems and especially for convex minimization problems, bundle methods of descent are well-defined. Non-descent methods, like subgradient projection methods, may also be used.[1] These methods are typically slower than gradient descent. Another alternative for non-differentiable functions is to "smooth" the function, or bound the function by a smooth function. In this approach, the smooth problem is solved in the hope that the answer is close to the answer for the non-smooth problem (occasionally, this can be made rigorous).

Solution of a linear system

Gradient descent can be used to solve a system of linear equations, reformulated as a quadratic minimization problem, e.g., using linear least squares. The solution of

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

in the sense of linear least squares is defined as minimizing the function

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

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

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

In this case, the line search minimization, finding the locally optimal step size γ on every iteration, can be performed analytically, and explicit formulas for the locally optimal γ are known.[2]

For solving linear equations, gradient descent is rarely used, with the conjugate gradient method being one of the most popular alternatives. The speed of convergence of gradient descent depends on the maximal and minimal eigenvalues of A, while the speed of convergence of conjugate gradients has a more complex dependence on the eigenvalues, and can benefit from preconditioning. Gradient descent also benefits from preconditioning, but this is not done as commonly.

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 a nonlinear system of equations:

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

suppose we have the function

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

where

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

and the objective function

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

With initial guess

$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

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

The Jacobian matrix $J_G(\mathbf{x}^{(0)})$

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

Then evaluating these terms at $\mathbf{x}^{(0)}$

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

So that

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

and

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

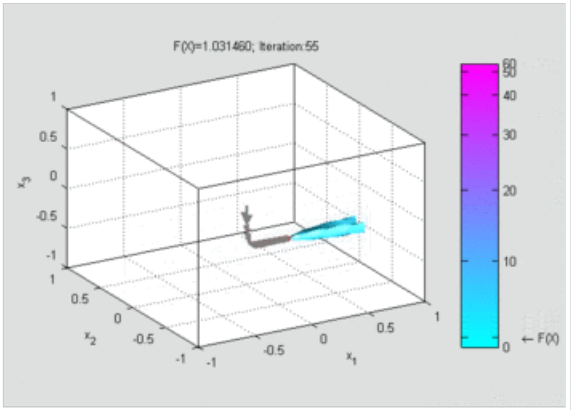
Now a suitable γ_0 must be found such that $F(\mathbf{x}^{(1)}) \leq F(\mathbf{x}^{(0)})$. This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$ which gives

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

Evaluating at this value,

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

The decrease from $F(\mathbf{x}^{(0)}) = 58.456$ to the next step's value of $F(\mathbf{x}^{(1)}) = 23.306$ is a sizable decrease in the objective function. Further steps would reduce its value until a 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 Gâteaux derivative of the functional to be minimized to determine the descent direction.[3]

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.[4][5] 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 Euler's method for solving ordinary differential equations $\boldsymbol{x}'(t) = -\nabla f(\boldsymbol{x}(t))$ of a gradient flow.

Computational examples

Python

The gradient descent algorithm is applied to find a local minimum of the function $f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$. Here is an implementation in the Python programming language.

```
# From calculation, it is expected that the local minimum occurs at x=9/4

x_old = 0 # The value does not matter as long as abs(x_new - x_old) > precision
x_new = 6 # The algorithm starts at x=6
gamma = 0.01 # step size
precision = 0.00001

def df(x):
    y = 4 * x**3 - 9 * x**2
    return y

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new += -gamma * df(x_old)

print("The local minimum occurs at ", +x_new)
```

The above piece of code has to be modified with regard to step size according to the system at hand and convergence can be made faster by using an adaptive step size. In the above case the step size is not adaptive. It stays at 0.01 in all the directions which can sometimes cause the method to fail by diverging from the minimum.

MATLAB

The following MATLAB code demonstrates a concrete solution for solving the non-linear system of equations presented in the previous section:

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

```
% Multi-variate vector-valued function G(x)
G = @(x) [
    3*x(1) - cos(x(2)*x(3)) - 3/2           ;
    4*x(1)^2 - 625*x(2)^2 + 2*x(2) - 1      ;
    exp(-x(1)*x(2)) + 20*x(3) + (10*pi-3)/3];

% Jacobian of G
JG = @(x) [
    3,                   sin(x(2)*x(3))*x(3),   sin(x(2)*x(3))*x(2) ;
    8*x(1),              -1250*x(2)+2,          0                   ;
    -x(2)*exp(-x(1)*x(2)), -x(1)*exp(-x(1)*x(2)), 20                 ];

% Objective function F(x) to minimize in order to solve G(x)=0
F = @(x) 0.5 * sum(G(x).^2);

% Gradient of F (partial derivatives)
dF = @(x) JG(x).' * G(x);

% Parameters
GAMMA = 0.001;     % step size (learning rate)
MAX_ITER = 1000;   % maximum number of iterations
FUNC_TOL = 0.1;    % termination tolerance for F(x)

fvals = [];        % store F(x) values across iterations
progress = @(iter,x) fprintf('iter = %3d: x = %-32s, F(x) = %f\n', ...
    iter, mat2str(x,6), F(x));

% Iterate
iter = 1;          % iterations counter
x = [0; 0; 0];     % initial guess
fvals(iter) = F(x);
progress(iter, x);
while iter < MAX_ITER && fvals(end) > FUNC_TOL
    iter = iter + 1;
    x = x - GAMMA * dF(x);   % gradient descent
    fvals(iter) = F(x);      % evaluate objective function
    progress(iter, x);       % show progress
end

% Plot
plot(1:iter, fvals, 'LineWidth',2); grid on;
```

```
title('Objective Function'); xlabel('Iteration'); ylabel('F(x)');

% Evaluate final solution of system of equations G(x)=0
disp('G(x) = '); disp(G(x))

% Output:
%
% iter =   1: x = [0;0;0]                        , F(x) = 58.456136
% iter =   2: x = [0.0075;0.002;-0.20944]        , F(x) = 23.306394
% iter =   3: x = [0.015005;0.0015482;-0.335103] , F(x) = 10.617030
% ...
% iter = 187: x = [0.683335;0.0388258;-0.52231]  , F(x) = 0.101161
% iter = 188: x = [0.684666;0.0389831;-0.522302] , F(x) = 0.099372
%
% (converged in 188 iterations after exceeding termination tolerance for F(x))
```

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).[6]

Fast gradient methods

Another extension of gradient descent is due to Yurii Nesterov from 1983,[7] and has been subsequently generalized. He provides a simple modification of the algorithm that enables faster convergence for convex problems. 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 $\mathcal{O}(1/k)$. Using the Nesterov acceleration technique, the error decreases at $\mathcal{O}(1/k^2)$.[8] It is known that the rate $\mathcal{O}(1/k^2)$ 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) [9] reduces that constant by a factor of two and is an optimal first-order method for large-scale problems. [10]

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

The momentum method

Yet another extension, that reduces the risk of getting stuck in a local minimum, as well as speeds up the convergence considerably in cases where the process would otherwise zig-zag heavily, is the *momentum method*, which uses a momentum term in analogy to "the mass of Newtonian particles that move through a viscous medium in a conservative force field".[11] This method is often used as an extension to the backpropagation algorithms used to train artificial neural networks.[12][13]

See also

- Conjugate gradient method
- Stochastic gradient descent
- Rprop
- Delta rule
- Wolfe conditions
- Preconditioning
- BFGS method
- Nelder–Mead method

References

- Mordecai Avriel (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Jan A. Snyman (2005). *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing. ISBN 0-387-24348-8
- Raad Z. Homod, K. S. M. Sahari, H. A.F. Almurib, F. H. Nagi, *Gradient auto-tuned Takagi-Sugeno fuzzy forward control of a HVAC system using predicted mean vote index* Energy and Buildings, 49 (6) (2012) 254-267 (http://www.sciencedirect.com/science/article/pii/S0378778812000904)
- Cauchy, Augustin (1847). *Méthode générale pour la résolution des systèmes d'équations simultanées*. pp. 536–538.

1. Kiwiel, Krzysztof C. (2001). "Convergence and efficiency of subgradient methods for quasiconvex minimization". *Mathematical Programming (Series A)*. **90** (1). Berlin, Heidelberg: Springer. pp. 1–25. doi:10.1007/PL00011414. ISSN 0025-5610. MR 1819784.
2. Yuan, Ya-xiang (1999). "Step-sizes for the gradient method" (PDF). *AMS/IP Studies in Advanced Mathematics*. Providence, RI: American Mathematical Society. **42** (2): 785.
3. G. P. Akilov, L. V. Kantorovich, Functional Analysis, Pergamon Pr; 2 Sub edition,ISBN 0-08-023036-9, 1982
4. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 2nd Ed., Cambridge University Press, New York, 1992
5. T. Strutz: *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)*. 2nd edition, Springer Vieweg, 2016, ISBN 978-3-658-11455-8.
6. P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing" (https://arxiv.org/pdf/0912.3522v4.pdf), in: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, (H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, Editors), pp. 185-212. Springer, New York, 2011.
7. Yu. Nesterov, "Introductory Lectures on Convex Optimization. A Basic Course" (Springer, 2004, ISBN 1-4020-7553-7)
8. Fast Gradient Methods (http://www.ee.ucla.edu/~vandenbe/236C/lectures/fgrad.pdf), lecture notes by Prof. Lieven Vandenberghe for EE236C at UCLA
9. D. Kim, J A Fessler, "Optimized first-order methods for smooth convex minimization" Math. Prog. 151:8-107, Sep. 2016 http://dx.doi.org/10.1007/s10107-015-0949-3
10. Yoel Drori, "The exact information-based complexity of smooth convex minimization." http://arxiv.org/abs/1606.01424
11. Qian, Ning (January 1999). "On the momentum term in gradient descent learning algorithms" (PDF). *Neural Networks*. **12** (1): 145–151. Retrieved 17 October 2014.
12. "Momentum and Learning Rate Adaptation". Willamette University. Retrieved 17 October 2014.
13. Geoffrey Hinton; Nitish Srivastava; Kevin Swersky. "6 - 3 - The momentum method". *YouTube*. Retrieved 18 October 2014. Part of a lecture series for the Coursera online course Neural Networks for Machine Learning (https://www.coursera.org/course/neuralnets).

External links

- Using gradient descent in C++, Boost, Ublas for linear regression (http://codingplayground.blogspot.it/2013/05/learning-linear-regression-with.html)

Categories: First order methods | Optimization algorithms and methods | Gradient methods

---