Foundations of Data Science, Fall 2020

9. Optimisation II

Prof. Dan Olteanu

DaST Data•(Systems+Theory)

University of Zurich[UZH]

Oct 20, 2020

https://lms.uzh.ch/url/RepositoryEntry/16830890400

https://uzh.zoom.us/j/96690150974?pwd=cnZmMTduWUtCeWoxYW85Z3RMYnpTZz09

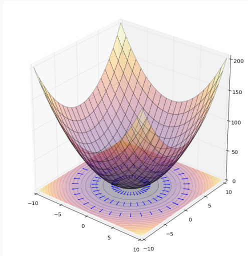
Calculus Background: Gradient Vectors are Orthogonal to Contour Curves

Theorem: If a function f is differentiable, the gradient of f at a point is either zero or perpendicular to the contour line of f at that point.

Analogy: Two hikers at the same location on a mountain.

1. Choose the direction where the slope is steepest
2. Choose a path that keeps the same height

The theorem says that they depart in directions perpendicular to each other.



Calculus Background: Gradient Points in the Direction of Steepest Increase

Each component of the gradient says how fast the function changes wrt the standard basis:

$$\frac{\partial f}{\partial x}(\mathbf{a}) = \lim_{h \to 0} \frac{f(\mathbf{a} + h\,\hat{\mathbf{i}}) - \mathbf{f}(\mathbf{a})}{h}$$

where $\hat{\mathbf{i}}$ is the unit vector in the direction of x (captures information in which direction we move)

What about changing wrt the direction of some arbitrary vector $\mathbf{v}$?

Directional Derivative $\nabla_{\mathbf{v}}$: derivative in direction of $\mathbf{v}$

$$\nabla_{\mathbf{v}} f(\mathbf{a}) = \lim_{h \to 0} \frac{f(\mathbf{a} + h\mathbf{v}) - f(\mathbf{a})}{h} = \nabla f(\mathbf{a}) \cdot \mathbf{v}$$

Geometric interpretation: Multiply $\|\nabla f(\mathbf{a})\|$ by the scalar projection of $\mathbf{v}$

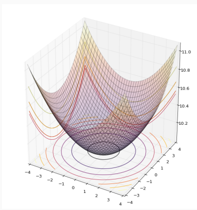
$$\nabla f(\mathbf{a}) \cdot \mathbf{v} = \|\nabla f(\mathbf{a})\| \|\mathbf{v}\| \cos\theta, \quad \cos\theta \text{ is the angle between } \nabla f(\mathbf{a}) \text{ and } \mathbf{v}$$

Maximal value is for $\cos\theta = 1$ or $\theta = 0$, so $\nabla f(\mathbf{a})$ and $\mathbf{v}$ have the same direction

Calculus Background: The Hessian Matrix

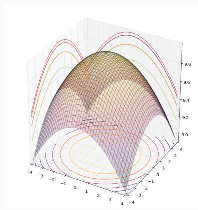
Hessian $\mathbf{H}$: The matrix of all second-order partial derivatives of f

- Symmetric as long as all second derivatives exist
- Captures the curvature of the surface



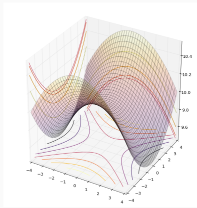
$\mathbf{H}$ has positive eigenvalues

local minimum



$\mathbf{H}$ has negative eigenvalues

local maximum



$\mathbf{H}$ has mixed eigenvalues

saddle point

Degenerate case: Eigenvalue = 0. No inverse, the gradient is locally unchanging

Gradient and Hessian: Example

$$z = f(w_1, w_2) = \frac{w_1^2}{a^2} + \frac{w_2^2}{b^2}$$

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{2w_1}{a^2} \\ \frac{2w_2}{b^2} \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} \end{bmatrix} = \begin{bmatrix} \frac{2}{a^2} & 0 \\ 0 & \frac{2}{b^2} \end{bmatrix}$$

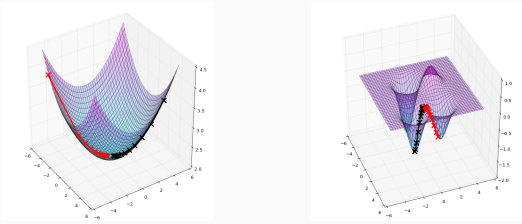
Gradient Descent

Gradient Descent Algorithm

- Gradient descent is one of the simplest, but very general optimisation algorithm for finding a local minimum of a differentiable function
- It is iterative, it produces a new vector $\mathbf{w}_{t+1}$ at each iteration t:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$

- At each iteration, it moves in the direction of the steepest descent
- $\eta_t > 0$ is the learning rate or step size



Gradient Descent for Least Squares Regression

$$\mathcal{L}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - y)^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) = \sum_{i=1}^{N} (\mathbf{x}_i^{\mathsf{T}}\mathbf{w} - y_i)^2$$

We can compute the gradient of $\mathcal{L}$ with respect to $\mathbf{w}$

$$\nabla_{\mathbf{w}} \mathcal{L} = 2\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w} - \mathbf{X}^{\mathsf{T}}\mathbf{y}\right)$$

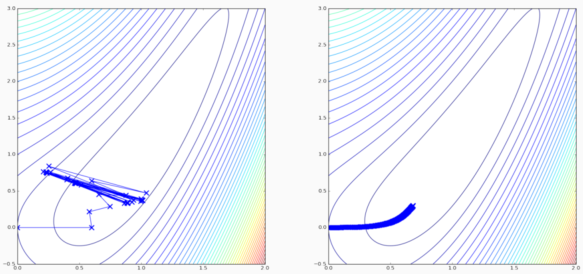
Gradient descent vs closed-form solution for very large (N) and wide (D) datasets

- Computational complexity of inverting matrices in closed-form solution
- In contrast: each gradient calculation linear in N and D

Choosing a Step Size

Choosing a good step-size is key and we may want a time-varying step size

- It step size is too large, algorithm may never converge
- If step size is too small, convergence may be very slow



Choosing a Step Size

- Constant step size: $\eta_t = \mathtt{c}$
- Decaying step size: $\eta_t = \mathtt{c}/t$. Different rates of decay common, e.g., $\frac{1}{\sqrt{t}}$
- Backtracking line search
  - Start with $\mathtt{c}/t$ (usually a large value)
  - Check for a decrease: Is $f(\mathbf{w}_t - \eta_t \nabla f(\mathbf{w})) < f(\mathbf{w})$?
  - If decrease condition not met, multiply η_t by a decaying factor, e.g., 0.5
  - Repeat until the decrease condition is met
  - What we use in our research prototypes: Armijo line search condition and the Barzilai-Borwein step size adjustment

A field guide to forward-backward splitting with a FASTA implementation. Goldstein, Studer, Baraniuk. 2014. `https://arxiv.org/abs/1411.3406`

When to Stop? Test for Convergence

Fixed number of iterations:	Terminate if	$t \geq \mathrm{T}$
Small increase:	Terminate if	$f(\mathbf{w}_{t+1}) - f(\mathbf{w}_t) \leq \epsilon$
Small change:	Terminate if	$\lVert \mathbf{w}_{t+1} - \mathbf{w}_t \rVert \leq \epsilon$

Stochastic Gradient Descent

Optimisation Algorithms for Machine Learning

We minimise the objective function over data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; \mathbf{x}_i, y_i) + \underbrace{\lambda \mathcal{R}(\mathbf{w})}_{\text{Regularisation Term}}$$

The gradient of the objective function is

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}}\ell(\mathbf{w}; \mathbf{x}_i, y_i) + \lambda \nabla_{\mathbf{w}}\mathcal{R}(\mathbf{w})$$

For Ridge Regression we have

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - y_i)^2 + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w}$$

$$\nabla_{\mathbf{w}}\mathcal{L}_{\text{ridge}} = \frac{1}{N}\sum_{i=1}^{N} 2(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - y_i)\mathbf{x}_i + 2\lambda \mathbf{w}$$

Stochastic Gradient Descent

As part of the learning algorithm, we calculate the following gradient:

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}}\ell(\mathbf{w}; \mathbf{x}_i, y_i) + \lambda \nabla_{\mathbf{w}}\mathcal{R}(\mathbf{w})$$

Suppose we pick a random datapoint $(\mathbf{x}_i, y_i)$ and evaluate $\mathbf{g}_i = \nabla_{\mathbf{w}}\ell(\mathbf{w}; \mathbf{x}_i, y_i)$

What is $\mathbb{E}[\mathbf{g}_i]$?

$$\mathbb{E}[\mathbf{g}_i] = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}}\ell(\mathbf{w}; \mathbf{x}_i, y_i)$$

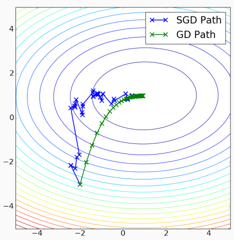
In expectation $\mathbf{g}_i$ points in the same direction as the entire gradient
(except for the regularisation term)

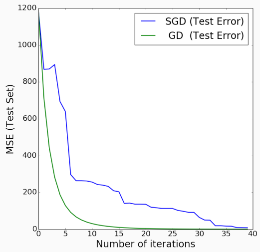
We compute the gradient at one data point instead of at all data points!

- Online learning
- Cheap to compute one gradient

Stochastic Gradient Descent vs (Batch) Gradient Descent

- 1000 data points for training and 1000 data points for test
- 2 features $x_1 \sim \mathcal{N}(0, 5)$ and $x_2 \sim \mathcal{N}(0, 8)$; centred labels
- Least-squares linear regression model $f_{\mathbf{w}}(\mathbf{x}) = x_1 w_1 + x_2 w_2$
- Parameters (w_1, w_2): initial $(-2, -3)$ and final $(1, 1)$





In practice: mini-batch gradient descent significantly improves the performance

- reduces the variance in the gradients and hence it is more stable than SGD

Sub-Gradient Descent

Minimising the Lasso Objective

Linear model trained with least squares loss and ℓ_1-regularisation:

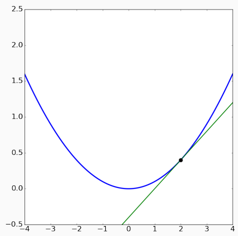
$$\mathcal{L}_{\text{lasso}}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - y_i)^2 + \lambda \sum_{i=1}^{D} |w_i|$$

- Quadratic part of the loss function can't be framed as linear programming
- Lasso regularisation does not allow for closed-form solutions
- Must resort to general optimisation methods
- We still have the problem that the objective function is not differentiable!

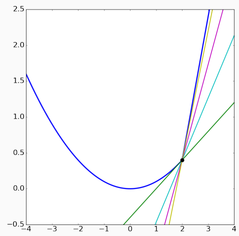
Sub-gradient Descent

Focus on the case when f is convex:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \quad \text{for all } x, y \text{ and for } \alpha \in [0, 1]$$



$f_1(x) = 0.1x^2$



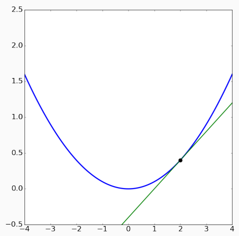
$$f_2(x) = \begin{cases} f_1(x) & \text{if } x < 2 \\ 2x - 3.6 & \text{otherwise} \end{cases}$$

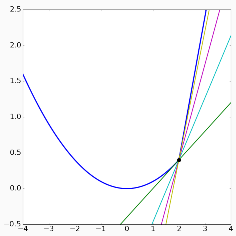
Tangent lines at $x = 2$

Sub-gradient Descent

Focus on the case when f is convex:

$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y) \quad \text{for all } x, y \text{ and for } \alpha \in [0,1]$$





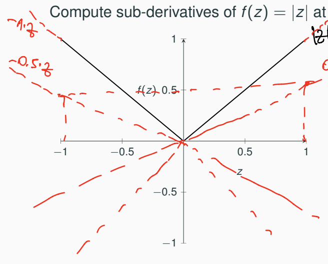
$$f(x) \ge f(x_0) + g(x - x_0) \quad \text{where } g \text{ is a sub-derivative}$$
$$f(\mathbf{x}) \ge f(\mathbf{x}_0) + \mathbf{g}^\top(\mathbf{x} - \mathbf{x}_0) \quad \text{where } \mathbf{g} \text{ is a sub-gradient}$$

Any $\mathbf{g}$ satisfying the above inequality is called a sub-gradient at $\mathbf{x}_0$

14

Sub-gradient Descent: Example 1

Compute sub-derivatives of $f(z) = |z|$ at points $x_0 = 1$, $x_1 = -3$, and $x_3 = 0$.



$f(z) \ge f(x) + g(z - x)$

$x_2 = 0$:

$f(z) \ge f(0) + g(z)$

$f(z) \ge g \cdot z$

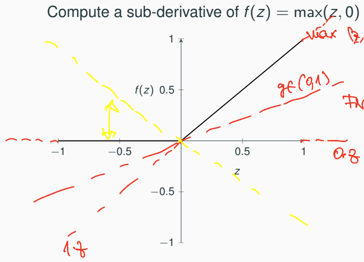
For which values of g would this hold?

$g \in [-1, 1]$.

15

Sub-gradient Descent: Example 2

Compute a sub-derivative of $f(z) = \max(z, 0)$ at point $x_0 = 0$.



$f(z) \ge f(x_0) + g \cdot (z - x_0)$

For $x_0 = 0$:

$f(z) \ge 0 + g \cdot z = g \cdot z$

$g \in [0, 1]$.

16

Sub-gradient Descent: Example 3

Compute a sub-gradient of $f([w_1, w_2, w_3, w_4]^\top) = \sum_{i=1}^{4} |w_i|$ at point $\mathbf{w}_0 = [2, -3, 0, 1]^\top$

17

Constrained Convex Optimisation

Constrained Convex Optimisation

Gradient descent

- Minimises $f(\mathbf{x})$ by moving in the negative gradient direction at each step:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$

- There is no constraint on the parameters

Projected gradient descent

- Minimises $f(\mathbf{x})$ subject to additional constraints $\mathbf{w}_C \in C$:

$$\mathbf{z}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} = \underset{\mathbf{w}_C \in C}{\operatorname{argmin}} \|\mathbf{z}_{t+1} - \mathbf{w}_C\|$$

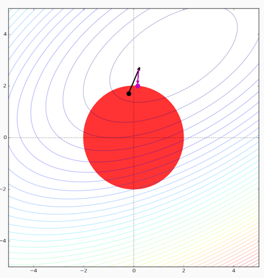
- Gradient step is followed by a projection step

18

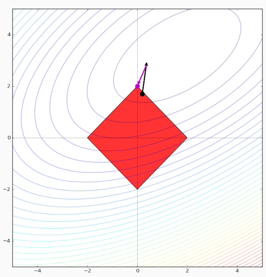
Constrained Convex Optimisation: Examples

Minimise $(\mathbf{X}\mathbf{w} - \mathbf{y})^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$ subject to the ridge and lasso constraints

$\mathbf{w}^\top\mathbf{w} < R$



$\sum_{i=1}^{D} |w_i| < R$



Second Order Methods

Newton's Method

In calculus: Finds roots of a differentiable function f, i.e., solutions to $f(\mathbf{x}) = 0$

In optimisation: Finds roots of f', i.e., solutions to $f'(\mathbf{x}) = 0$

- Function f needs to be twice-differentiable
- The roots of f' are stationary points of f, i.e., minima/maxima/saddle points

Newton's Method in One Dimension

- Construct a sequence of points $x_1, \dots, x_n$ starting with an initial guess x_0
- Sequence converges towards a minimiser x^* of f using sequence of second-order Taylor approximations of f around the iterates:

$$f(x) \approx f(x_k) + (x - x_k)f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(x_k)$$

- $x_{k+1} = x^*$ defined as the minimiser of this quadratic approximation
- If f'' is positive, then the quadratic approximation is convex, and a minimiser is obtained by setting the derivative to zero:

$$0 = \frac{d}{dx}\left(f(x_k) + (x - x_k)f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(x_k)\right) = f'(x_k) + (x^* - x_k)f''(x_k)$$

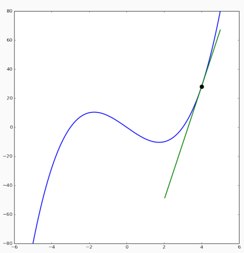
Then,

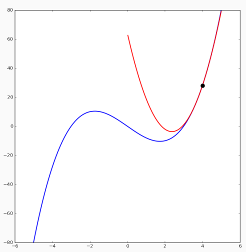
$$x_{k+1} = x^* = x_k - f'(x_k)[f''(x_k)]^{-1}$$

Geometric Interpretation of Newton's Method

At iteration k, we fit a paraboloid to the surface of f at x_k with the same slopes and curvature as the surface at x_k and go for the extremum of that paraboloid

$f(x) = x^3 - 9x$ $\quad g(x) = f(x_0) + (x - x_0)f'(x_0)$ $\quad r(x) = f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$





- Gradient descent: First derivative
- Local linear approximation

- Newton: Second derivative
- Degree 2 Taylor approximation around current point

Newton's Method in High Dimensions

First derivative $\rightarrow$ gradient

Second derivative $\rightarrow$ Hessian

- Approximate f around $\mathbf{x}_k$ using second order Taylor approximation

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^\top(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\top \mathbf{H}_k(\mathbf{x} - \mathbf{x}_k)$$

- The gradient of f is given by:

$$\nabla_{\mathbf{x}} f = \mathbf{g}_k + \mathbf{H}_k(\mathbf{x} - \mathbf{x}_k)$$

- Setting $\nabla_{\mathbf{x}} f = 0$, we obtain $\mathbf{x}^* = \mathbf{x}_k - \mathbf{H}_k^{-1}\mathbf{g}_k$
- We move directly to the (unique) stationary point $\mathbf{x}^*$ of f
- We repeat the above iteration with $\mathbf{x}_{k+1} = \mathbf{x}^*$

Newton's Method: Computation and Convergence

Newton's method

- Computational requirements at each Newton step
  - $D + \binom{D}{2}$ partial derivatives and inverse of the Hessian
  - Instead: Compute $\mathbf{x}$ as the solution of the system of linear equations

    $$\mathbf{H}_k(\mathbf{x} - \mathbf{x}_k) = -\mathbf{g}_k$$

    using factorisations (e.g., Cholesky) of $\mathbf{H}_k$
- For convex f
  - It converges to stationary points of the quadratic approximation
  - All stationary points are global minima
  - Converges to unique minimiser quadratically fast *if*$^+$ *f is strongly convex*

Newton's Method: Computation and Convergence

Newton's method

- Computational requirements at each Newton step
  - $D + \binom{D}{2}$ partial derivatives and inverse of the Hessian
  - Instead: Compute $\mathbf{x}$ as the solution of the system of linear equations

    $$\mathbf{H}_k(\mathbf{x} - \mathbf{x}_k) = -\mathbf{g}_k$$

    using factorisations (e.g., Cholesky) of $\mathbf{H}_k$
- For convex f
  - It converges to stationary points of the quadratic approximation
  - All stationary points are global minima
  - Converges to unique minimiser quadratically fast *if*$^+$ *f is strongly convex*
- For non-convex f
  - Stationary points may not be minima nor in the decreasing direction of f
  - Not successful for training deep neural networks:
    abundance of saddle points for their non-convex objective functions

Summary

Convex Optimization

- Convex Optimization is 'efficient' (*i.e.,* polynomial time)
- Try to cast learning problem as a convex optimization problem
- Many, many extensions of standard gradient descent exist: Adagrad, Momentum-based, BGFS, L-BGFS, Adam, *etc.*
- Books: Boyd and Vandenberghe, Nesterov's Book

Non-Convex Optimization

- Encountered frequently in deep learning
- (Stochastic) Gradient Descent gives local minima
- Nonlinear Programming - Dimitri Bertsekas