Machine Learning

Chapter 8

Optimization

Saeed Ebadollahi

Outline

8.1 Introduction

- Local vs global optimization
- Constrained vs unconstrained optimization
- Convex vs nonconvex optimization
- Smooth vs non smooth optimization

8.2 First-order methods

- Descent direction
- Step size (learning rate)
- Convergence rates
- Momentum methods

8.3 Second-order methods

- Newton's method
- BFGS and other quasi-Newton methods
- Trust region methods

8.4 Stochastic gradient descent

- Application to finite sum problems
- Example: SGD for fitting linear regression
- Choosing the step size (learning rate)
- Iterate averaging
- Variance reduction
- Preconditioned SGD

8.1 Introduction

The core problem in machine learning → parameter estimation

- This requires solving an **optimization problem**, where we try to find the values for a set of variables $\theta \in \Theta$, that minimize a scalar-valued **loss function** or **cost function** $\mathcal{L} : \Theta \to \mathbb{R}$:

$$\theta^* \in \underset{\theta \in \Theta}{\operatorname{argmin}}\, \mathcal{L}(\theta)$$

→ focusing on **continuous optimization**, rather than **discrete optimization**.

- **Parameter space** is given by $\Theta \subseteq \mathbb{R}^D$, ($D$ is the number of variables being optimized over)

minimize $\boldsymbol{\mathcal{L}}(\theta) = -R(\theta) \equiv$ *maximize* a **score function** or **reward function** $R(\theta)$

✓ **objective function:** a function we want to maximize or minimize.
✓ **Solver:** An algorithm that can find an optimum of an objective function.

8.1 Introduction

Local vs global optimization

global optimum: A point that satisfies Equation $\theta^* \in \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(\theta)$

global optimization: Finding such a point satisfies above Equation

local minimum:

$$\exists \delta > 0, \forall \theta \in \Theta \quad s.t. \quad \|\theta - \theta^*\| < \delta, \mathcal{L}(\theta^*) \leq \mathcal{L}(\theta)$$

strict local minimum:

$$\exists \delta > 0, \forall \theta \in \Theta \quad s.t. \quad \|\theta - \theta^*\| < \delta, \mathcal{L}(\theta^*) < \mathcal{L}(\theta)$$

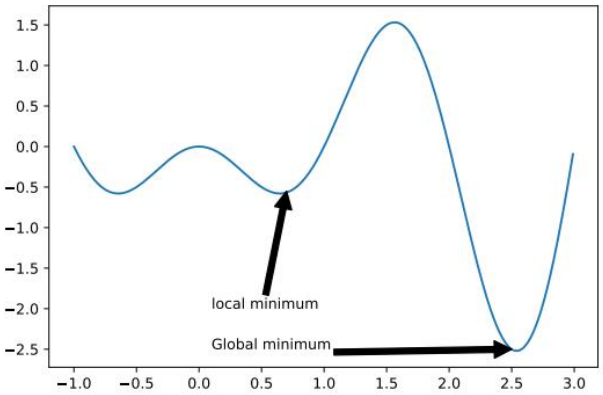
saddle point: a point where some directions point downhill, and some uphill. More precisely, at a saddle point, the eigenvalues of the Hessian will be both positive and negative.

8.1 Introduction

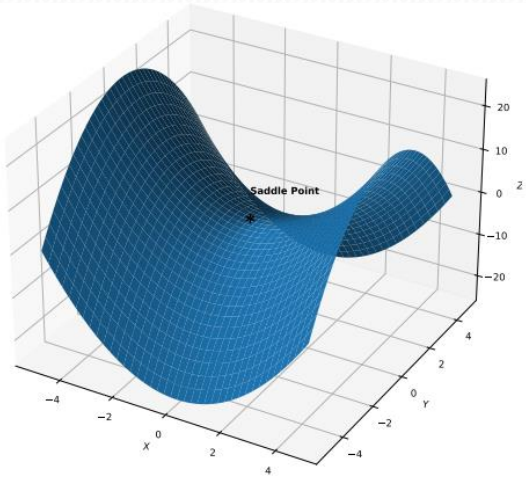
Local vs global optimization

flat local minimum: A local minimum surrounded by other local minima with the same objective value.

globally convergent: if an algorithm is guaranteed to converge to a stationary point from any starting point.



(a)



(b)

Figure 8.1: (a) Illustration of local and global minimum in 1d. Generated by code at figures.probml.ai/book1/8.1. (b) Illustration of a saddle point in 2d. Generated by code at figures.probml.ai/book1/8.1.

8.1 Introduction

Optimality conditions for local vs global optima

Consider a point $\theta^* \in \mathbb{R}^D$, and let $g^* = \nabla \mathcal{L}(\theta)|_{\theta^*}$,$(g(\theta) = \nabla \mathcal{L}(\theta))$ be the gradient at that point, and $H^* = \nabla^2 \mathcal{L}(\theta)|_{\theta^*}$, $(H(\theta) = \nabla^2 \mathcal{L}(\theta))$ be the corresponding Hessian.

Conditions characterize every local minimum:

- Necessary condition: If $\boldsymbol{\theta}^*$ is a local minimum, then we must have $\boldsymbol{g}^* = \mathbf{0}$ (i.e., $\boldsymbol{\theta}^*$ must be a **stationary point**), and $\boldsymbol{H}^*$ must be positive semi-definite.
- In optimization, a stationary point is a point where the gradient of the objective function is zero.
- Sufficient condition: If $\boldsymbol{g}^* = \mathbf{0}$ and $\boldsymbol{H}^*$ is positive definite, then θ^* is a local minimum.

Question: Why a zero gradient is not sufficient?

- ✓ note that the stationary point could be a local minimum, maximum or **saddle point**
- ✓ If the Hessian at a point is positive semi-definite, then some directions may point uphill, while others are flat.
- ✓ if the Hessian is strictly positive definite, then we are at the bottom of a "bowl", and all directions point uphill.

8.1 Introduction

Constrained vs unconstrained optimization

partition the set of **constraints** $\mathcal{C}$

- **inequality constraints**: $g_j(\theta) \leq 0 \; for \; j \in I$
  Example: $g_i(\theta) = -\theta_i \leq 0$
- **equality constraints**: $h_k(\theta) = 0 \; for \; k \in \mathcal{E}$
  Example: $h(\theta) = (1 - \sum_{i=1}^{D} \theta_i) = 0$

We define the feasible set as the subset of the parameter space that satisfies the constraints:

$$\mathcal{C} = \{\theta : g_i(\theta) \leq 0 : j \in I, h_k(\theta) = 0 \; for \; k \in \mathcal{E}\} \subseteq \mathbb{R}^D$$

Our constrained optimization problem now becomes

$$\theta^* \in \underset{\theta \in \mathcal{C}}{\operatorname{argmin}} \, \mathcal{L}(\theta)$$

If $\mathcal{C} = \mathbb{R}^D$, it is called **unconstrained optimization**.

8.1 Introduction

Constrained vs unconstrained optimization

feasibility problem: The task of finding any point (regardless of its cost) in the feasible set.

common strategy for solving constrained problems:

1. Create penalty terms that measure how much we violate each constraint.
2. Add these terms to the objective and solve an unconstrained optimization problem.

The **Lagrangian** is a special case of such a combined objective.

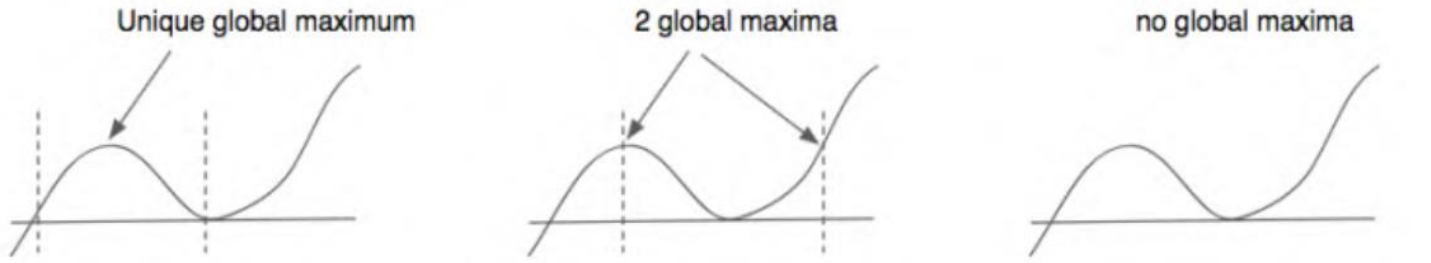


Figure 8.2: Illustration of constrained maximization of a nonconvex 1d function. The area between the dotted vertical lines represents the feasible set. (a) There is a unique global maximum since the function is concave within the support of the feasible set. (b) There are two global maxima, both occuring at the boundary of the feasible set. (c) In the unconstrained case, this function has no global maximum, since it is unbounded.

8.1 Introduction

Convex vs nonconvex optimization

In **convex optimization**, we require the objective to be a convex function defined over a convex set. In such problems, every local minimum is also a global minimum.

- **Convex sets**

We say S is a **convex set** if, for any $\boldsymbol{x}, \boldsymbol{x}' \in S$, we have

$$\lambda x + (1-\lambda)x' \in S, \qquad \forall \lambda \in [0,1]$$

That is, if we draw a line from $\boldsymbol{x}$ *to* $\boldsymbol{x}'$, all points on the line lie inside the set.

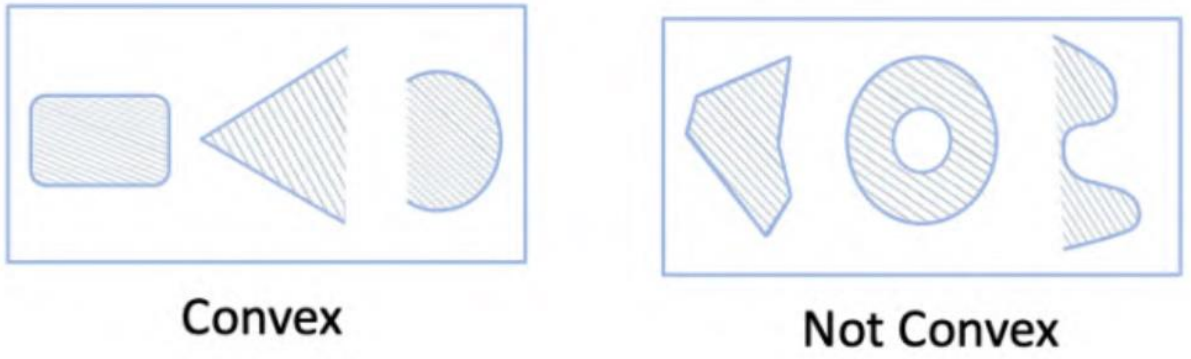


Figure 8.3: Illustration of some convex and non-convex sets.

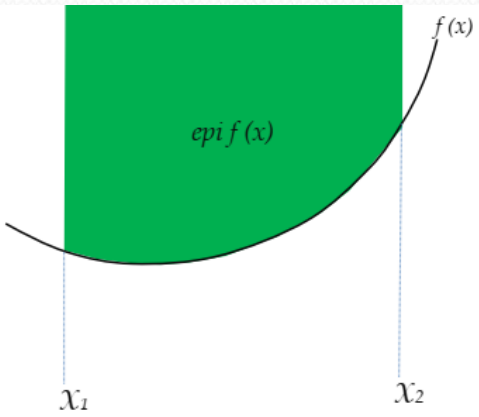
8.1 Introduction

Convex vs nonconvex optimization

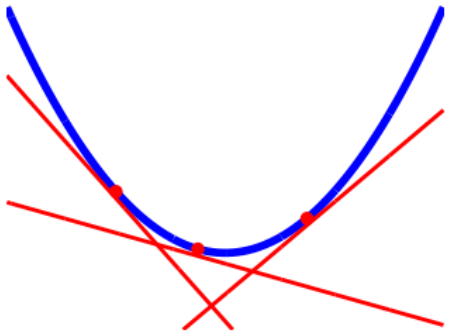
- **Convex functions**

We say f is a **convex function** if its **epigraph** defines a convex set. Equivalently, a function $f(x)$ is called convex if it is defined on a convex set and if, for any $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{S}$, and for any $0 \leq \lambda \leq 1$, we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$



(a)



(b)

Figure 8.4: (a) Illustration of the epigraph of a function. (b) For a convex function $f(x)$, its epipgraph can be represented as the intersection of half-spaces defined by linear lower bounds derived from the **conjugate function** $f^*(\lambda) = \max_x \lambda x - f(x)$.

8.1 Introduction

Convex vs nonconvex optimization

A function is called **strictly convex** if the inequality is strict.

If $-f(x)$ is convex $\longrightarrow$ $f(x)$ is **concave**

If $-f(x)$ is strictly convex $\longrightarrow$ $f(x)$ is **strictly concave**

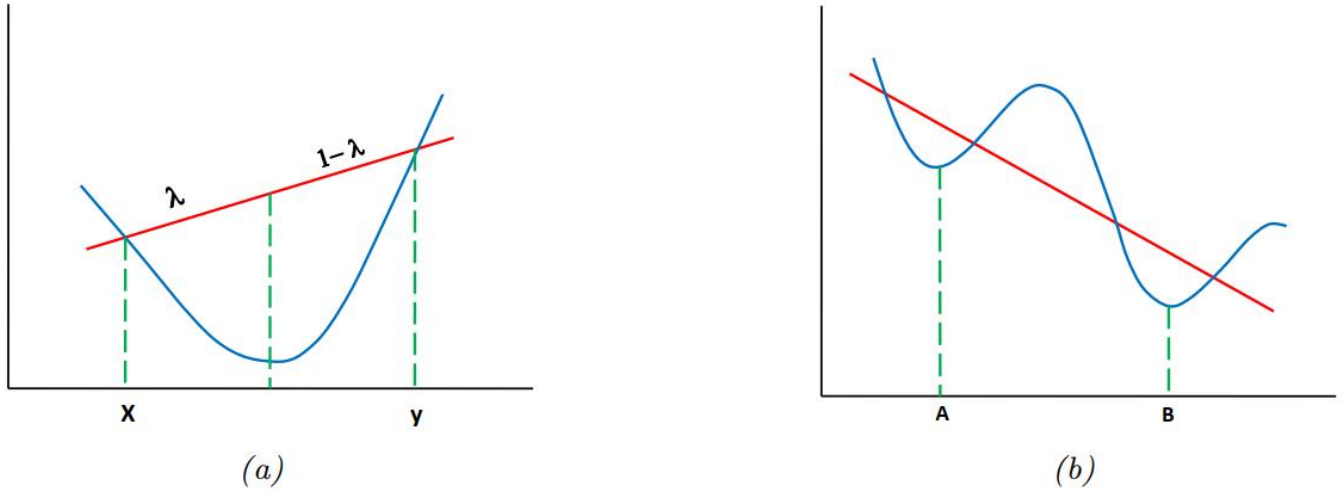


*Figure 8.5: (a) Illustration of a convex function. We see that the chord joining $(x, f(x))$ to $(y, f(y))$ lies above the function. (b) A function that is neither convex nor concave. **A** is a local minimum, **B** is a global minimum.*

8.1 Introduction

Characterization of convex functions

Theorem 8.1.1. *Suppose $f: \mathbb{R}^n \to \mathbb{R}$ is twice differentiable over its domain. Then f is convex if $H = \nabla^2 f(x)$ is positive semi definite for all $\boldsymbol{x} \in dom(f)$. Furthermore, f is strictly convex if $\boldsymbol{H}$ is positive definite.*

$$f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T} \mathbf{A} \boldsymbol{x}$$

This is convex if **A** is positive semi definite, and is strictly convex if **A** is positive definite. It is neither convex nor concave if **A** has eigenvalues of mixed sign.

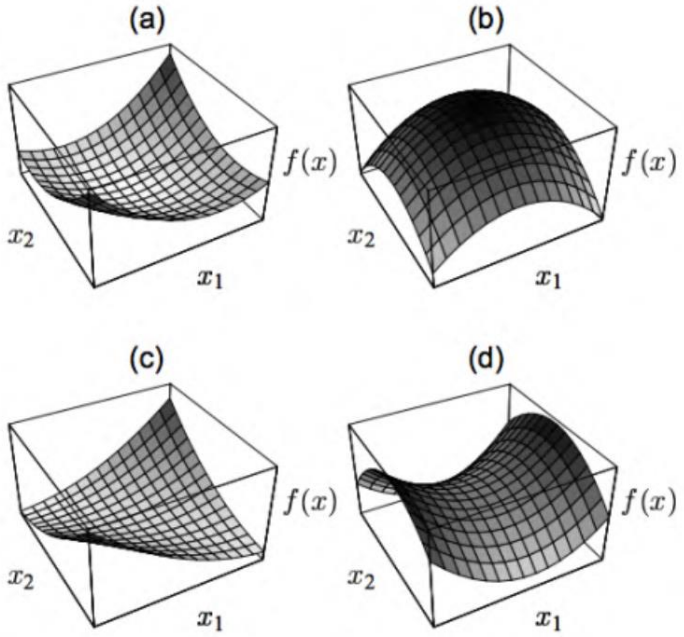


*Figure 8.6: The quadratic form $f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T} \mathbf{A} \boldsymbol{x}$ in 2d. (a) **A** is positive definite, so f is convex. (b) **A** is negative definite, so f is concave. (c) **A** is positive semidefinite but singular, so f is convex, but not strictly. Notice the valley of constant height in the middle. (d) **A** is indefinite, so f is neither convex nor concave. The stationary point in the middle of the surface is a saddle point. From Figure 5 of [She94].*

8.1 Introduction

Strongly convex functions

We say a function f is **strongly convex** with parameter m > 0 if the following holds for all $\boldsymbol{x}, \boldsymbol{y}$ in f's domain:

$$\big(\nabla f(x) - \nabla f(y)\big)^T (x - y) \geq m\|x - y\|_2^2 \qquad \nabla^2 f(\boldsymbol{x}) \succeq m\mathbf{I}$$

A strongly convex function is also strictly convex, but not vice versa.

Differences between convex, strictly convex, and strongly convex:

(consider the case where f is twice continuously differentiable and the domain is the real line.)

- f is convex if and only if $f''(x) \geq 0$ for all x.
- f is strictly convex if $f''(x) > 0$ for all x (note: this is sufficient, but not necessary).
- f is strongly convex if and only if $f''(x) \geq m > 0$ for all x.

Note that it can be shown that a function f is strongly convex with parameter m if the bellow function is convex.

$$J(\boldsymbol{x}) = f(\boldsymbol{x}) - \frac{m}{2}\|x\|^2$$

8.1 Introduction

Smooth vs non smooth optimization

In **smooth optimization**, the objective and constraints are continuously differentiable functions. For smooth functions, we can quantify the degree of smoothness using the **Lipschitz constant**. In the 1d case, this is defined as any constant L ≥ 0 such that, for all real x1 and x2, we have

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|$$

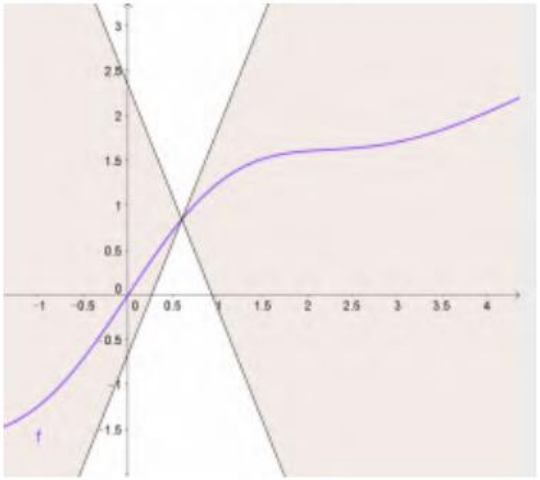


Figure 8.8: For a Lipschitz continuous function f, there exists a double cone (white) whose origin can be moved along the graph of f so that the whole graph always stays outside the double cone. From https://en.wikipedia.org/wiki/Lipschitz_continuity. Used with kind permission of Wikipedia author Taschee.

8.1 Introduction

Smooth vs non smooth optimization

In **non smooth optimization**, there are at least some points where the gradient of the objective function or the constraints is not well-defined.

composite objective:

$$\mathcal{L}(\theta) = \mathcal{L}_s(\theta) + \mathcal{L}_r(\theta)$$

smooth (differentiable) ← $\mathcal{L}_s$, $\mathcal{L}_r$ → Non smooth ("rough")

In machine learning applications, $\mathcal{L}_s$ is usually the training set loss, and $\mathcal{L}_r$ is a regularizer, such as the ℓ_1 norm of θ.

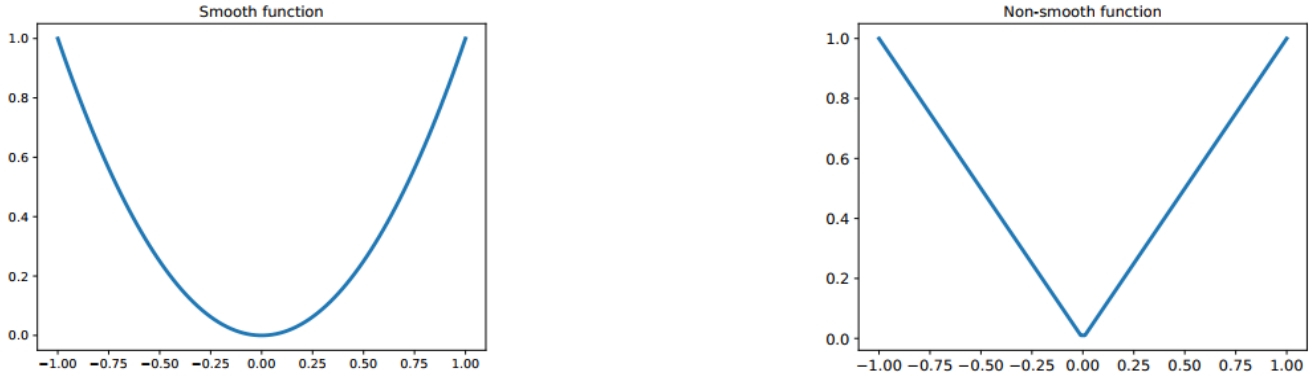


Figure 8.7: (a) Smooth 1d function. (b) Non-smooth 1d function. (There is a discontinuity at the origin.) Generated by code at figures.probml.ai/book1/8.7.

8.1 Introduction

Smooth vs non smooth optimization

- **Subgradients**

for a convex function of several variables, $f: \mathbb{R}^n \to \mathbb{R}$, we say that $g \in \mathbb{R}^n$ is a **subgradient** of f at $\boldsymbol{x} \in dom(f)$ if for all vectors $\boldsymbol{z} \in dom(f)$,

$$f(\boldsymbol{z}) \geq f(\boldsymbol{x}) + g^T(\boldsymbol{z} - \boldsymbol{x})$$

Note that a subgradient can exist even when f is not differentiable at a point.

A function f is called **subdifferentiable** at $\boldsymbol{x}$ if there is at least one subgradient at $\boldsymbol{x}$.

The set of such subgradients is called the **subdifferential** of f at $\boldsymbol{x}$, and is denoted $\partial f(x)$.

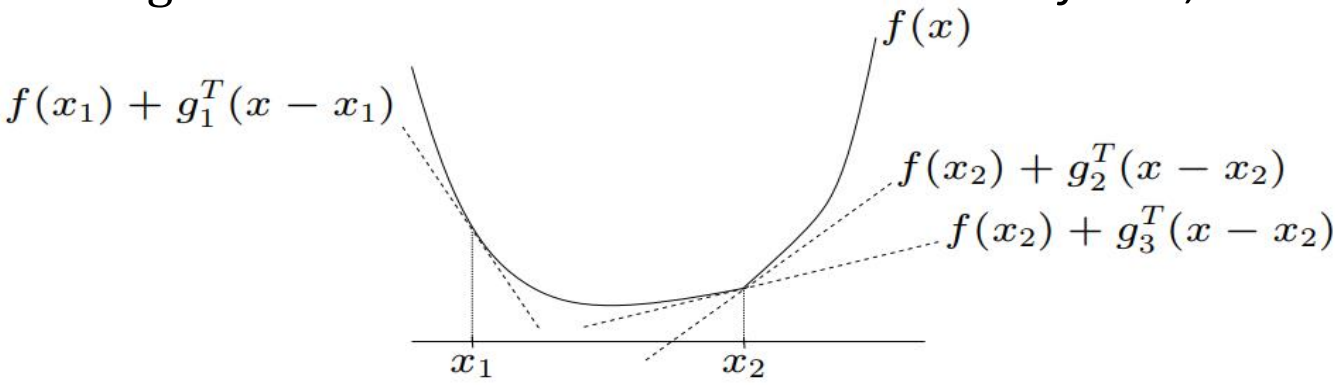


Figure 8.9: Illustration of subgradients. At $\boldsymbol{x}_1$, the convex function f is differentiable, and $\boldsymbol{g}_1$ (which is the derivative of f at $\boldsymbol{x}_1$) is the unique subgradient at $\boldsymbol{x}_1$. At the point $\boldsymbol{x}_2$, f is not differentiable, because of the "kink". However, there are many subgradients at this point, of which two are shown. From https://web.stanford.edu/class/ee364b/lectures/subgradients_slides.pdf. Used with kind permission of Stephen Boyd.

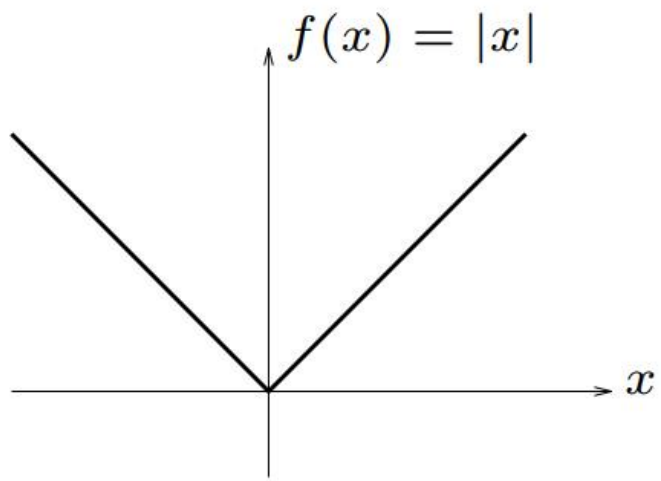
8.1 Introduction

Smooth vs non smooth optimization

- **Subgradients**

Example: $f(x) = |x|$. Its subdifferential is given by

$$\partial f(x) = \begin{cases} \{-1\} & if\ x < 0 \\ [-1, 1] & if\ x = 0 \\ \{+1\} & if\ x > 0 \end{cases}$$



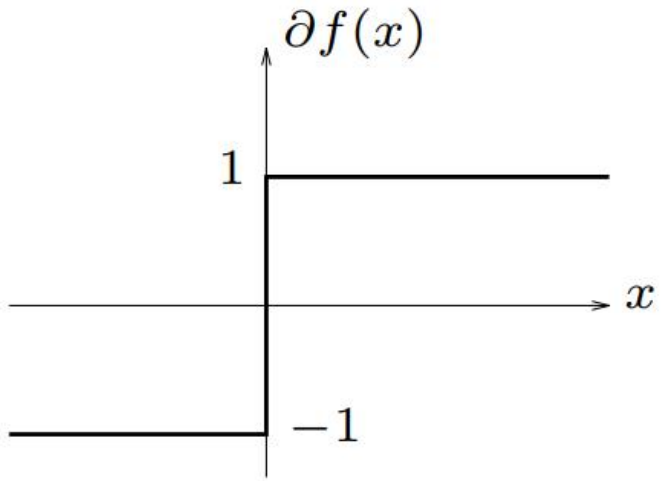


Figure 8.10: The absolute value function (left) and its subdifferential (right). From https://web.stanford.edu/class/ee364b/lectures/subgradients_slides.pdf. Used with kind permission of Stephen Boyd.

8.2 First-order methods

In this section, we consider iterative optimization methods that leverage first order derivatives of the objective function, i.e., they compute which directions point "downhill", but they ignore curvature information. All of these algorithms require that the user specify a starting point $\boldsymbol{\theta}_0$. Then at each iteration t, they perform an update of the following form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \, \boldsymbol{d}_t$$

where η_t is known as the **step size** or **learning rate**, and $\boldsymbol{d}_t$ is a **descent direction**, such as the negative of the **gradient**, given by $\boldsymbol{g}_t = \boldsymbol{\nabla}_\theta \mathcal{L}(\theta)|_{\boldsymbol{\theta}_t}$. These update steps are continued until the method reaches a stationary point, where the gradient is zero.

8.2 First-order methods

Descent direction

We say that a direction $\boldsymbol{d}$ is a **descent direction** if there is a small enough (but nonzero) amount η we can move in direction $\boldsymbol{d}$ and be guaranteed to decrease the function value. Formally, we require that there exists an $\eta_{max} > 0$ such that

$$\mathcal{L}(\theta + \eta d) < \mathcal{L}(\theta)$$

for all $0 < \eta < \eta_{max}$. The gradient at the current iterate,

$$g_t \triangleq \nabla \mathcal{L}(\theta)|_{\theta_t} = \nabla \mathcal{L}(\theta_t) = g(\theta_t)$$

points in the direction of maximal increase in f, so the negative gradient is a descent direction. It can be shown that any direction $\boldsymbol{d}$ is also a descent direction if the angle θ between $\boldsymbol{d}$ and $-\boldsymbol{g_t}$ is less than 90 degrees and satisfies

$$d^T g_t = \|d\| \|g_t\| \cos(\theta) < 0$$

It seems that the best choice would be to pick $d_t = -g_t$. This is known as the direction of **steepest descent**. However, this can be quite slow.

8.2 First-order methods

Step size (learning rate)

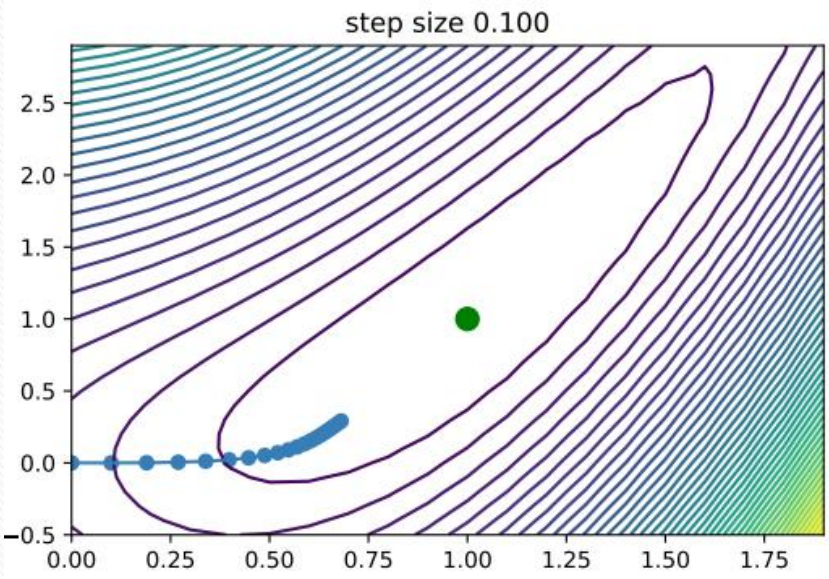
In machine learning, the sequence of step sizes $\{\eta_t\}$ is called the **learning rate schedule**.

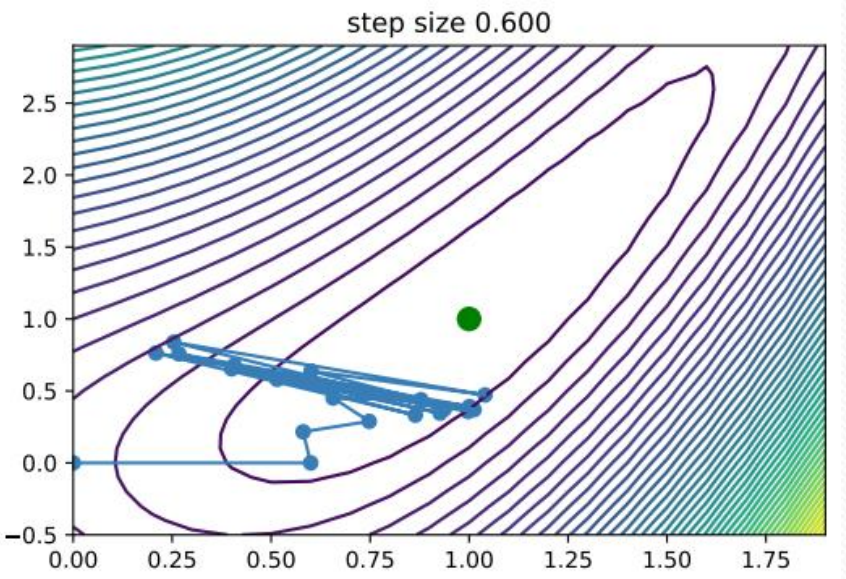
Constant step size

The simplest method is to use a constant step size, $\eta_t = \eta$. However, if it is too large, the method may fail to converge, and if it is too small, the method will converge but very slowly.

Example: consider the bellow convex function, (descent direction: $d_t = -g_t$)

$$\mathcal{L}(\theta) = 0.5(\theta_1{}^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$$





8.2 First-order methods

Constant step size

In some cases, we can derive a theoretical upper bound on the maximum step size we can use.

Example: Consider a quadratic objective, $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T \mathbf{A}\boldsymbol{\theta} + \boldsymbol{b}^T\boldsymbol{\theta} + c$ with $\mathbf{A} \succcurlyeq 0$. One can show that steepest descent will have global convergence if the step size satisfies

$$\eta < \frac{2}{\lambda_{max}(\mathbf{A})}$$

the largest eigenvalue of A

More generally, setting $\eta < 2/L$, where L is the Lipschitz constant of the gradient, ensures convergence.

8.2 First-order methods

Line search

The optimal step size can be found by finding the value that maximally decreases the objective along the chosen direction by solving the 1d minimization problem

$$\eta_t = \underset{\eta>0}{\operatorname{argmin}}\, \phi_t(\eta) = \underset{\eta>0}{\operatorname{argmin}}\, \mathcal{L}(\theta_t + \eta d_t)$$

This is known as **line search**, since we are searching along the line defined by d_t.

If the loss is convex, this subproblem is also convex, because $\phi_t(\eta) = \mathcal{L}(\theta_t + \eta d_t)$ is a convex function of an affine function of η, for fixed θ_t and d_t.

Example: consider the quadratic loss $\mathcal{L}(\theta) = \frac{1}{2}\theta^T A\theta + b^T\theta + c$

Computing the derivative of ϕ gives

$$\frac{d\phi(\eta)}{d\eta} = \frac{d}{d\eta}\left[\frac{1}{2}(\theta+\eta d)^T A(\theta+\eta d) + b^T(\theta+\eta d) + c\right] = d^T A(\theta+\eta d) + d^T b$$

$$= d^T(A\theta + b) + \eta d^T A d$$

Solving for $\frac{d\phi(\eta)}{d\eta} = 0$ gives $\eta = -\frac{d^T(A\theta+b)}{d^T A d}$

exact line search: Using the optimal step size.

8.2 First-order methods

Convergence rates

For certain convex problems, with a gradient with bounded Lipschitz constant, one can show that gradient descent converges at a **linear rate**. This means that there exists a number $0 < \mu < 1$ such that

$$|\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_*)| \leq \mu |\mathcal{L}(\theta_t) - \mathcal{L}(\theta_*)|$$

rate of convergence: μ

- quadratic objective: $\mathcal{L}(\theta) = \frac{1}{2}\theta^T A\theta + b^T\theta + c$ with $A \succ 0$.

Suppose we use steepest descent with exact line search. One can show that the convergence rate is given by

$$\mu = \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2$$

the largest eigenvalue of A ← λ_{max} , λ_{min} → the smallest eigenvalue of A

Rewrite:

$$\mu = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2, \qquad \left(\kappa = \frac{\lambda_{max}}{\lambda_{min}}: \quad \text{the condition number of A}\right)$$

Intuitively, the condition number measures how "skewed" the space is, in the sense of being far from a symmetrical "bowl".

8.2 First-order methods

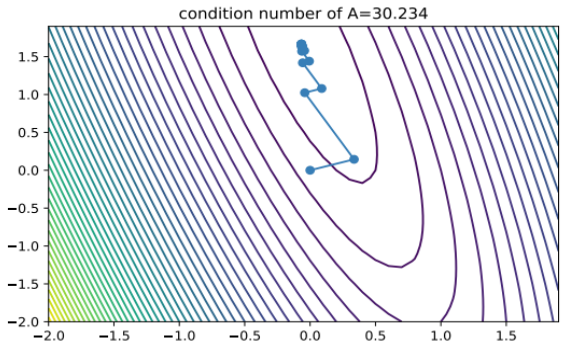
Convergence rates

- non-quadratic functions:

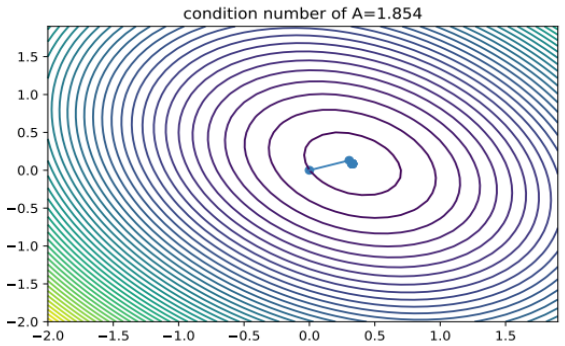
The objective will often be locally quadratic around a local optimum. Hence the convergence rate depends on the condition number of the Hessian, $\kappa(H)$, at that point. We can often improve the convergence speed by optimizing a surrogate objective (or model) at each step which has a Hessian that is close to the Hessian of the objective function.

conjugate gradient descent: Overcome the inefficient **zig-zag** behavior of the path of steepest descent with an exact line-search

$A = [20, 5; 5, 2]$
$b = [-14; -6]$
$c = 10$
$\kappa(A) = 30.234$



(a)



(b)

$A = [20, 5; 5, 16]$
$b = [-14; -6]$
$c = 10$
$\kappa(A) = 1.8541$

Figure 8.12: Illustration of the effect of condition number κ on the convergence speed of steepest descent with exact line searches. (a) Large κ. (b) Small κ. Generated by code at figures.probml.ai/book1/8.12.

8.2 First-order methods

Momentum methods

Heavy ball (**momentum method):** Move faster along directions that were previously good, and slow down along directions where the gradient has suddenly changed, just like a ball rolling downhill.

$$m_t = \beta m_{t-1} + g_{t-1}$$
$$\theta_t = \theta_{t-1} - \eta_t m_t$$

m_t: momentum (mass times velocity)
$0 < \beta < 1$, typical value = 0.9
If β=0 ⟶ the method reduces to gradient descent

We see that m_t is like an exponentially weighted moving average of the past gradients

$$m_t = \beta m_{t-1} + g_{t-1} = \beta^2 m_{t-2} + \beta g_{t-2} + g_{t-1} = \cdots = \sum_{\tau=0}^{t-1} \beta^\tau g_{t-\tau-1}$$

If all the past gradients are a constant, say $\boldsymbol{g}$, this simplifies to

$$m_t = g \sum_{\tau=0}^{t-1} \beta^\tau$$

The scaling factor is a geometric series, whose infinite sum is given by

$$1 + \beta + \beta^2 + \cdots = \sum_{i=0}^{\infty} \beta^i = \frac{1}{1-\beta}$$

8.2 First-order methods

Nesterov momentum

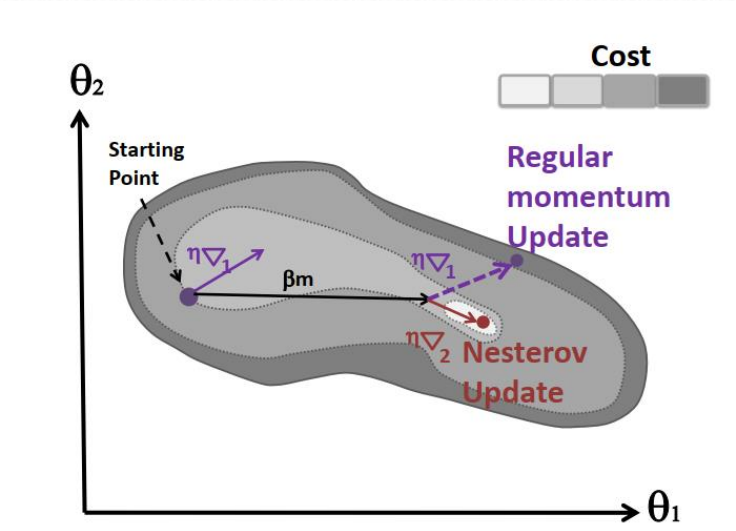
The **Nesterov accelerated gradient** method modifies the gradient descent to include an extrapolation step, as follows:

$$\tilde{\theta}_{t+1} = \theta_t + \beta_t(\theta_t - \theta_{t-1})$$
$$\theta_{t+1} = \tilde{\theta}_{t+1} - \eta_t \nabla \mathcal{L}(\tilde{\theta}_{t+1})$$

This is essentially a form of one-step "look ahead", that can reduce the amount of oscillation.

Rewritten in the same format as standard momentum:

$$m_{t+1} = \beta m_t - \eta_t \nabla \mathcal{L}(\theta_t + \beta m_t)$$
$$\theta_{t+1} = \theta_t + m_{t+1}$$



8.2 First-order methods

Nesterov momentum

- ✓ This method can be faster than standard momentum:

the momentum vector is already roughly pointing in the right direction, so measuring the gradient at the new location, $\theta_t + \beta m_t$, rather than the current location, θ_t, can be more accurate.

- ✓ The Nesterov accelerated gradient method is provably faster than steepest descent for convex functions when β and η_t are chosen appropriately.
- ✓ It is called "accelerated" because of this improved convergence rate, which is optimal for gradient-based methods using only first-order information when the objective function is convex and has Lipschitz-continuous gradients.
- ✓ In practice, however, using Nesterov momentum can be slower than steepest descent, and can even unstable if β or η_t are misspecified.

8.3 Second-order methods

first-order methods: Optimization algorithms that only uses the gradient.
Advantage: the gradient is cheap to compute and to store
Disadvantage: they do not model the curvature of the space, and hence they can be slow to converge.

Second-order methods: incorporates curvature in various ways (e.g., via the Hessian).
Advantage: faster convergence

8.3 Second-order methods

Newton's method

The classic second-order method is **Newton's method**. This consists of updates of the form

$$\theta_{t+1} = \theta_t - \eta_t H_t^{-1} g_t$$

where

$$H_t \triangleq \nabla^2 \mathcal{L}(\theta)|_{\theta_t} = \nabla^2 \mathcal{L}(\theta_t) = \mathrm{H}(\theta_t)$$

is assumed to be positive-definite to ensure the update is well-defined.

The intuition for why this is faster than gradient descent is that the matrix inverse H^{-1} "undoes" any skew in the local curvature.

Algorithm 1: Newton's method for minimizing a function

1. Initialize $\boldsymbol{\theta}_0$;
2. **for** $t = 1, 2, \ldots$ *until convergence* **do**
3. Evaluate $\boldsymbol{g}_t = \nabla \mathcal{L}(\boldsymbol{\theta}_t)$;
4. Evaluate $\mathbf{H}_t = \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t)$;
5. Solve $\mathbf{H}_t \boldsymbol{d}_t = -\boldsymbol{g}_t$ for $\boldsymbol{d}_t$;
6. Use line search to find stepsize η_t along $\boldsymbol{d}_t$;
7. $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \boldsymbol{d}_t$;

8.3 Second-order methods

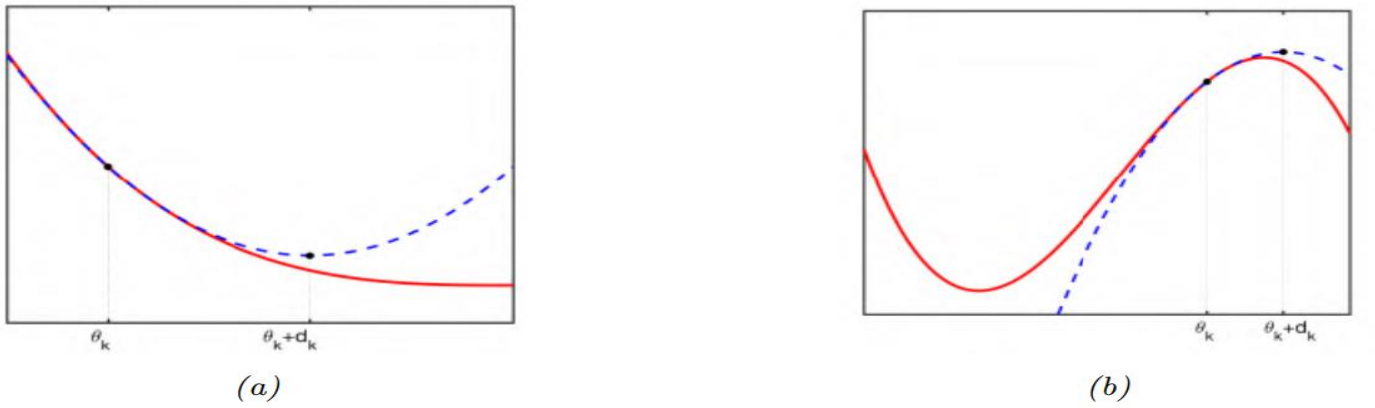
Newton's method

Consider making a second-order Taylor series approximation of $\mathcal{L}(\theta)$ around θ_t:

$$\mathcal{L}_{quad}(\theta) = \mathcal{L}(\theta_t) + g_t^T(\theta - \theta_t) + \frac{1}{2}(\theta - \theta_t)^T H_t(\theta - \theta_t)$$

The minimum of $\mathcal{L}_{quad}$ is at

$$\boldsymbol{\theta = \theta_t - H_t^{-1} g_t}$$



(a) (b)

Figure 8.14: Illustration of Newton's method for minimizing a 1d function. (a) The solid curve is the function $\mathcal{L}(x)$. The dotted line $\mathcal{L}_{\text{quad}}(\theta)$ is its second order approximation at θ_t. The Newton step d_t is what must be added to θ_t to get to the minimum of $\mathcal{L}_{\text{quad}}(\theta)$. Adapted from Figure 13.4 of [Van06]. Generated by code at figures.probml.ai/book1/8.14. (b) Illustration of Newton's method applied to a nonconvex function. We fit a quadratic function around the current point θ_t and move to its stationary point, $\theta_{t+1} = \theta_t + d_t$. Unfortunately, this takes us near a local maximum of f, not minimum. This means we need to be careful about the extent of our quadratic approximation. Adapted from Figure 13.11 of [Van06]. Generated by code at figures.probml.ai/book1/8.14.

8.3 Second-order methods

Newton's method

✓ if the quadratic approximation is a good one ⟶ $d_t = -H_t^{-1} g_t$

✓ in a "pure" Newton method ⟶ $\eta_t = 1$

However, we can also use line search to find the best step size; this tends to be more robust as using $\eta_t = 1$ may not always converge globally.

➢ If we apply this method to linear regression, we get to the optimum in one step, since we have $H = X^T X$ and $g = X^T X w - X^T y$, so the Newton update becomes

$$w_1 = w_0 - H^{-1} g = w_0 - (X^T X)^{-1}(X^T X w_0 - X^T y) = w_0 - w_0 + (X^T X)^{-1} X^T y$$

which is the OLS estimate.

➢ when we apply this method to logistic regression, it may take multiple iterations to converge to the global optimum

8.3 Second-order methods

BFGS and other quasi-Newton methods

Quasi-Newton (**variable metric**) methods: iteratively build up an approximation to the Hessian using information gleaned from the gradient vector at each step.

BFGS: updates the approximation to the Hessian $B_t \approx H_t$ as follows:

$$B_{t+1} = B_t + \frac{y_t y_t^T}{y_t^T s_t} - \frac{(B_t s_t)(B_t s_t)^T}{s_t^T B_t s_t}$$

$$s_t = \theta_t - \theta_{t-1}$$

$$y_t = g_t - g_{t-1}$$

If B_0 is positive-definite, and the step size η is chosen via line search satisfying both the Armijo condition and the following curvature condition :

$$\nabla \mathcal{L}(\theta_t + \eta d_t) \geq c_2 \eta d_t^T \nabla \mathcal{L}(\theta_t)$$

Then B_{t+1} will remain positive definite. The constant c_2 is chosen within $(c, 1)$ where c is the tunable parameter. The two step size conditions are together known as the **Wolfe conditions**. We typically start with a diagonal approximation, $\boldsymbol{B_0 = I}$. Thus BFGS can be thought of as a "diagonal plus low-rank" approximation to the Hessian.

8.3 Second-order methods

BFGS and other quasi-Newton methods

Alternatively, BFGS can iteratively update an approximation to the inverse Hessian, $C_t \approx H_t^{-1}$, as follows:

$$C_{t+1} = \left(I - \frac{s_t y_t^T}{y_t^T s_t}\right) C_t \left(I - \frac{y_t s_t^T}{y_t^T s_t}\right) + \frac{s_t s_t^T}{y_t^T s_t}$$

Since storing the Hessian approximation still takes $O(D^2)$ space, for very large problems:

limited memory BFGS (L-BFGS):

- ✓ Control the rank of the approximation by only using the M most recent (s_t, y_t) pairs.
- ✓ Ignoring older information.

Rather than storing B_t explicitly, we just store these vectors in memory, and then approximate $H_t^{-1} g_t$ by performing a sequence of inner products with the stored s_t and y_t vectors.

The storage requirements are therefore $O(MD)$.

Typically choosing M to be between 5–20 suffices for good performance.

8.4 Stochastic gradient descent

- **stochastic optimization**: The goal is to minimize the average value of a function:

$$\mathcal{L}(\theta) = \mathbb{E}_{q(z)}[\mathcal{L}(\theta, z)]$$

z: a random input to the objective
- a "noise" term, coming from the environment
- a training example drawn randomly from the training set

At each iteration, we assume we observe $\mathcal{L}_t(\theta) = \mathcal{L}(\theta, z_t)$ where $z_t \sim q$.

- **stochastic gradient descent (SGD):**

If the distribution $q(\boldsymbol{z})$ is independent of the parameters we are optimizing, we can use $g_t = \nabla_\theta \mathcal{L}_t(\theta_t)$. In this case, The resulting algorithm can be written as follows:

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta_t, z_t) = \theta_t - \eta_t g_t$$

As long as the gradient estimate is unbiased, then this method will converge to a stationary point, providing we decay the step size η_t at a certain rate.

8.4 Stochastic gradient descent

Application to finite sum problems

finite sum problem:

many model fitting procedures are based on empirical risk minimization, which involve minimizing the following loss:

$$\mathcal{L}(\theta_t) = \frac{1}{N}\sum_{n=1}^{N} \ell\big(y_n, f(x_n;\theta_t)\big) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(\theta_t)$$

The gradient of this objective:

$$g_t = \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \mathcal{L}_n(\theta_t) = \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \ell\big(y_n, f(x_n;\theta_t)\big)$$

approximate this by sampling a **minibatch** of $B \ll N$ samples:

$$g_t \approx \frac{1}{|B_t|}\sum_{n\in B_t} \nabla_\theta \mathcal{L}_n(\theta_t) = \frac{1}{|B_t|}\sum_{n\in B_t} \nabla_\theta \ell\big(y_n, f(x_n;\theta_t)\big)$$

B_t: a set of randomly chosen examples to use at iteration t.

✓ the theoretical rate of convergence of SGD is slower than batch GD, but in practice SGD is often faster, since the per-step time is much lower.

8.4 Stochastic gradient descent

Example: SGD for fitting linear regression

least mean squares (LMS) algorithm (**delta rule, Widrow-Hoff rule**):

Objective:

$$\mathcal{L}(\theta) = \frac{1}{2N}\sum_{n=1}^{N}(x_n^T\theta - y_n)^2 = \frac{1}{2N}\|X\theta - y\|_2^2$$

The gradient:

$$g_t = \frac{1}{N}\sum_{n=1}^{N}(\theta_t^T x_n - y_n)x_n$$

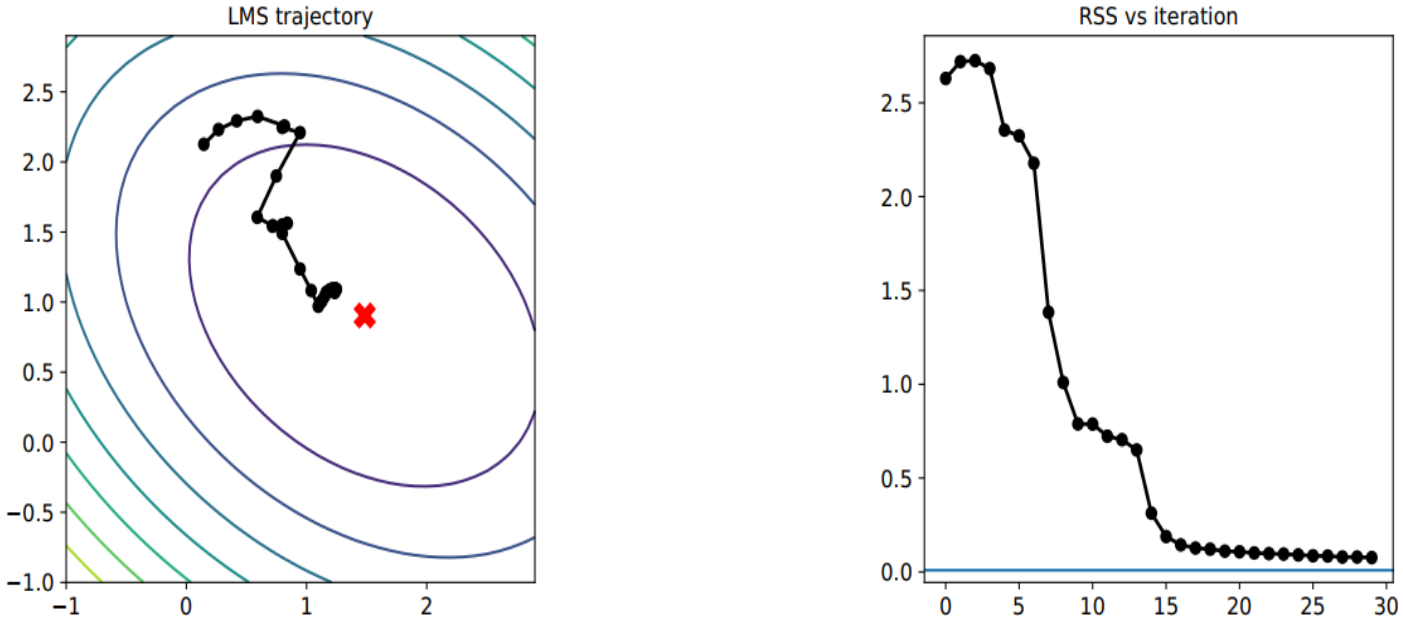
Now consider using SGD with a minibatch size of $B = 1$. The update becomes

$$\theta_{t+1} = \theta_t - \eta_t(\theta_t^T x_n - y_n)x_n$$

$n = n(t)$: the index of the example chosen at iteration t.

8.4 Stochastic gradient descent

Example: SGD for fitting linear regression



(a) *(b)*

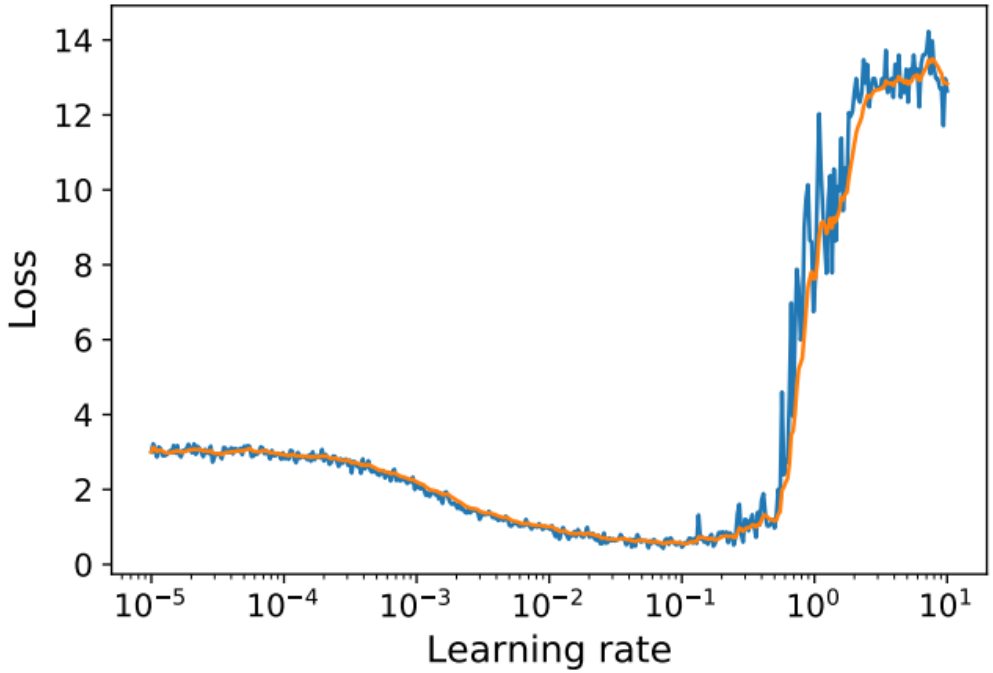
Figure 8.16: Illustration of the LMS algorithm. Left: we start from $\boldsymbol{\theta} = (-0.5, 2)$ *and slowly converging to the least squares solution of* $\hat{\boldsymbol{\theta}} = (1.45, 0.93)$ *(red cross). Right: plot of objective function over time. Note that it does not decrease monotonically. Generated by code at figures.probml.ai/book1/8.16.*

8.4 Stochastic gradient descent

Choosing the step size (learning rate)

overly small learning rate → underfitting
overly large learning rate → instability of the model

} fail to converge to a local optimum



8.4 Stochastic gradient descent

Choosing the step size (learning rate)

Choosing a good learning rate:

1. Start with a small learning rate and gradually increase it, evaluating performance using a small number of minibatches.
2. then make a plot and pick the learning rate with the lowest loss.

a sufficient condition for SGD to achieve convergence is if the learning rate schedule satisfies the **Robbins-Monro conditions**:

$$\eta_t \to 0, \qquad \frac{\sum_{t=1}^{\infty} {\eta_t}^2}{\sum_{t=1}^{\infty} \eta_t} \to 0$$

Examples of learning rate schedules:

piecewise constant: $\eta_t = \eta_i \;\; if \;\; t_i \leq t \leq t_{i+1}$

t_i: a set of time points at which we adjust the learning rate to a specified value.

exponential decay: $\eta_t = \eta_0 e^{-\lambda t}$

polynomial decay: $\eta_t = \eta_0 (\beta t + 1)^{-\alpha}$

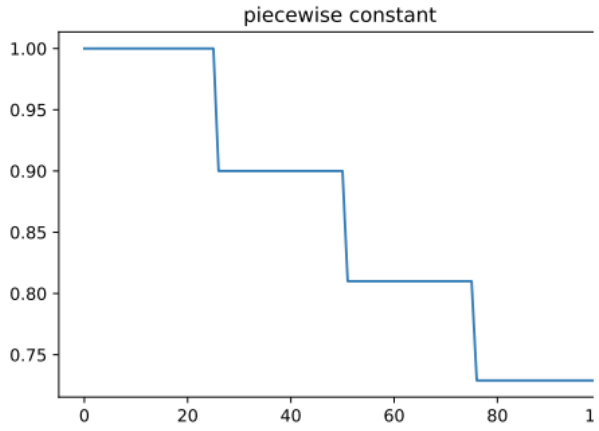
8.4 Stochastic gradient descent

Choosing the step size (learning rate)

Examples of learning rate schedules:

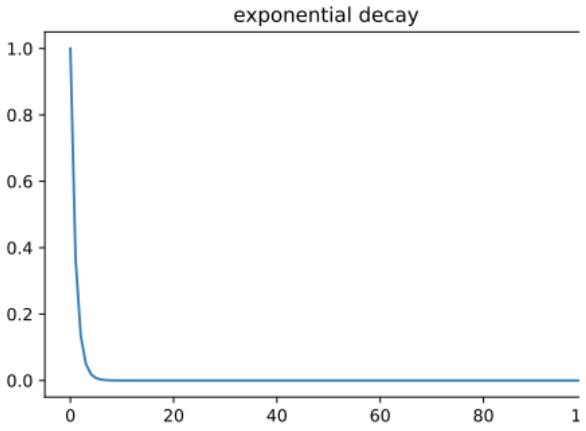
step decay: $\eta_i = \eta_0 \gamma^i$ reduces the initial learning rate by a factor of γ for each threshold.

$$\eta_0 = 1, \gamma = 0.9$$



(a)

reduce-on-plateau: the thresholds times are computed adaptively, by estimating when the train or validation loss has plateaued.

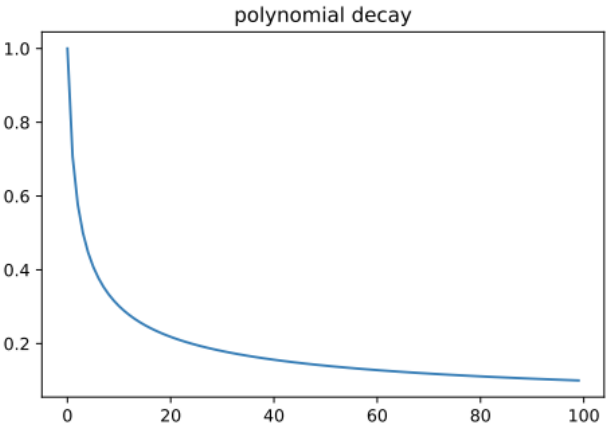


(b)

square-root schedule:

$$\eta_t = \eta_0 \frac{1}{\sqrt{t+1}}$$

$$\alpha = 0.5, \beta = 1$$



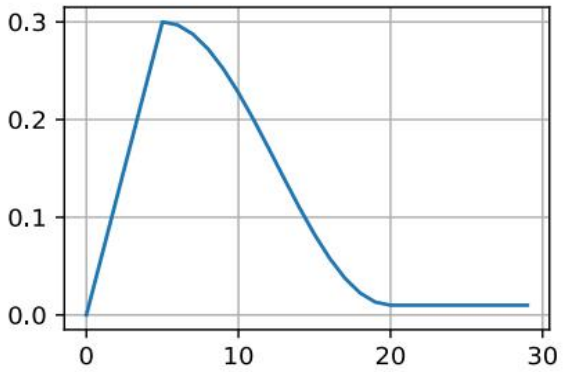
(c)

8.4 Stochastic gradient descent

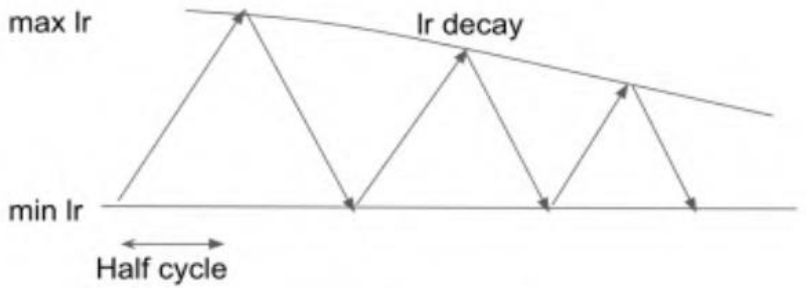
Choosing the step size (learning rate)

learning rate warmup (**one-cycle learning rate schedule**): In the deep learning community, another common schedule is to quickly increase the learning rate and then gradually decrease it again.

cyclical learning rate: It is also possible to increase and decrease the learning rate multiple times, in a cyclical fashion.



(a)



(b)

Figure 8.19: (a) Linear warm-up followed by cosine cool-down. (b) Cyclical learning rate schedule.

8.4 Stochastic gradient descent

Iterate averaging

iterate averaging (**Polyak-Ruppert averaging**): To reduce the variance of the estimate produced by SGD, we can compute the average using:

$$\bar{\theta}_t = \frac{1}{t}\sum_{i=1}^{t}\theta_i = \frac{1}{t}\theta_t + \frac{t-1}{t}\bar{\theta}_{t-1}$$

θ_t: usual SGD iterates

- The estimate $\bar{\theta}_t$ achieves the best possible asymptotic convergence rate among SGD algorithms, matching that of variants using second-order information, such as Hessians.

8.4 Stochastic gradient descent

Variance reduction

various ways to reduce the variance in SGD:

1. **stochastic variance reduced gradient** (**SVRG**): The basic idea is to use a control variate, in which we estimate a baseline value of the gradient based on the full batch, which we then use to compare the stochastic gradients to.

2. **stochastic averaged gradient accelerated** (**SAGA**): Unlike SVRG, it only requires one full batch gradient computation, at the start of the algorithm. However, it "pays" for this saving in time by using more memory. In particular, it must store *N* gradient vectors.

These methods reduce the variance of the gradients, rather than the parameters themselves and are designed to work for finite sum problems.

8.4 Stochastic gradient descent

Preconditioned SGD

Preconditioned SGD involves the following update:

$$\theta_{t+1} = \theta_t - \eta_t M_t^{-1} g_t$$

M_t: a preconditioning matrix (preconditioner), typically chosen to be positive-definite.

- ✓ The noise in the gradient estimates make it difficult to reliably estimate the Hessian.
- ✓ It is expensive to solve for the update direction with a full preconditioning matrix. Therefore most practitioners use a diagonal preconditioner M_t.
- ✓ Such preconditioners do not necessarily use second-order information, but often result in speedups compared to vanilla SGD.

8.4 Stochastic gradient descent

AdaGrad

The ADAGRAD (adaptive gradient) method was originally designed for optimizing convex objectives where many elements of the gradient vector are zero; these might correspond to features that are rarely present in the input, such as rare words. The update has the following form

$$\theta_{t+1,d} = \theta_{t,d} - \eta_t \frac{1}{\sqrt{s_{t,d}+\epsilon}} g_{t,d}, \qquad s_{t,d} = \sum_{t=1}^{t} {g_{t,d}}^2$$

The update in vector form:

$$\Delta\theta_t = -\eta_t \frac{1}{\sqrt{s_t + \epsilon}} g_t$$

- Viewed as preconditioned SGD, this is equivalent to taking $M_t = diag(s_t + \epsilon)^{1/2}$.
- This is an example of an adaptive learning rate; the overall step size η_t still needs to be chosen, but the results are less sensitive to it compared to vanilla GD.
- In particular, we usually fix $\eta_t = \eta_0$.

8.4 Stochastic gradient descent

RMSProp

A defining feature of AdaGrad is that the term in the denominator gets larger over time, so the effective learning rate drops. While it is necessary to ensure convergence, it might hurt performance as the denominator gets large too fast.

An alternative is to use an exponentially weighted moving average(EWMA) of the past squared gradients, rather than their sum:

$$s_{t+1,d} = \beta s_{t,d} + (1-\beta) {g_{t,d}}^2$$

In practice we usually use $\beta \sim 0.9$, which puts more weight on recent examples. In this case,

$$\sqrt{s_{t,d}} \approx RMS(g_{1:t,d}) = \sqrt{\frac{1}{t}\sum_{\tau=1}^{t} {g_{\tau,d}}^2}$$

The overall update of RMSProp is

$$\Delta\theta_t = -\eta_t \frac{1}{\sqrt{s_t + \epsilon}} g_t$$

8.4 Stochastic gradient descent

AdaDelta

The AdaDelta is similar to RMSprop. However, in addition to accumulating an EWMA of the gradients in $\hat{s}$, it also keeps an EWMA of the updates δ_t to obtain an update of the form

$$\Delta\theta_t = -\eta_t \frac{\sqrt{\delta_{t-1}+\epsilon}}{\sqrt{s_t+\epsilon}} g_t, \quad \delta_t = \beta\delta_{t-1} + (1-\beta)(\Delta\theta_t)^2$$

- This has the advantage that the "units" of the numerator and denominator cancel, so we are just elementwise-multiplying the gradient by a scalar.
- This eliminates the need to tune the learning rate η_t, although popular implementations of AdaDelta still keep η_t as a tunable hyperparameter.
- Since these adaptive learning rates need not decrease with time, these methods are not guaranteed to converge to a solution.

8.4 Stochastic gradient descent

Adam (adaptive moment estimation)

It is possible to combine RMSProp with momentum. In particular, let us compute an EWMA of the gradients and squared gradients

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$s_t = \beta_2 s_{t-1} + (1-\beta_2) {g_t}^2$$

update:

$$\Delta\theta_t = -\eta_t \frac{1}{\sqrt{s_t} + \epsilon} m_t$$

- ✓ The standard values for the various constants:

$$\beta_1 = 0.9, \beta_2 = 0.999 \text{ and } \epsilon = 10^{-6}$$

- ✓ For the overall learning rate, it is common to use a fixed value such as η_t=0.001.
- ✓ As the adaptive learning rate may not decrease over time, convergence is not guaranteed.
- ✓ If we initialize with $m_0 = s_0 = 0$, then initial estimates will be biased towards small values. The authors therefore recommend using the bias-corrected moments, which increase the values early in the optimization process. These estimates are given by

$$\hat{m}_t = m_t / (1-\beta_1^t)$$
$$\hat{s}_t = s_t / (1-\beta_2^t)$$