Optimization and Logistic regression

Qingyun Yu, Xueyi He

School of Mathematics
Sun Yat-sen University

August 26, 2020

Table of Contents

Table of Contents

Introduction

- What is Optimization? $\min_{x \in D} f(x)$
  Usually, in most machine learning case:

$$\arg \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x})$$

$$\text{s.t.} \begin{cases} g_i(\mathbf{x}) \geq 0, i = 1, \ldots, m \\ h_j(\mathbf{x}) = 0, j = 1, \ldots, r \end{cases}$$

  where $f(x)$, $g_i(\cdot), i = 1, \ldots, m$ and $h_j(\cdot), j = 1, \ldots, r$ are both real functions:$\mathbb{R}^d \to \mathbb{R}$,$x = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$
- We call $f(x)$ an objective function.

Local minima

- An $x \in \mathbb{R}^d$ is called a feasible point in D, if

$$D = \left\{ \mathbf{x} \in \mathbb{R}^d : g_i(\mathbf{x}) \geq 0, i = 1, \ldots, m; h_j(\mathbf{x}) = 0, j = 1, \ldots, r \right\},$$

  We can rewrite the optimization problem as $\min_{x \in D} f(x)$.
- If $x* \in D$ such that,

$$f(\mathbf{x}) \geq f\left(\mathbf{x}^*\right), \forall \mathbf{x} \in D,$$

  then x^* is a global solution to the opt problem
- If $x^* \in D$ such that,

$$f(x) > f\left(x^*\right), \forall x \in D \backslash \left\{x^*\right\},$$

  then x^* is a strict global solution to the constrained opt problem.

Introduction

Usually, optimization is used in many machine learning algorithms:

- Linear regression:

$$\text{minimize}_w \|Xw - y\|^2$$

- Classification (logistic regresion or SVM):

$$\begin{gathered}\text{minimize}_w \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i x_i^T w\right)\right) \\ \text{or } \|w\|^2 + C\sum_{i=1}^{n} \xi_i \text{ s.t. } \xi_i \geq 1 - y_i x_i^T w, \xi_i \geq 0\end{gathered}$$

- Maximum likelihood estimation:

$$\underset{\theta}{\text{maximize}} \sum_{i=1}^{n} \log p_\theta\left(x_i\right)$$

linear or non-linear least squares, minimizing Bayesian risks, regularized regressions, etc.

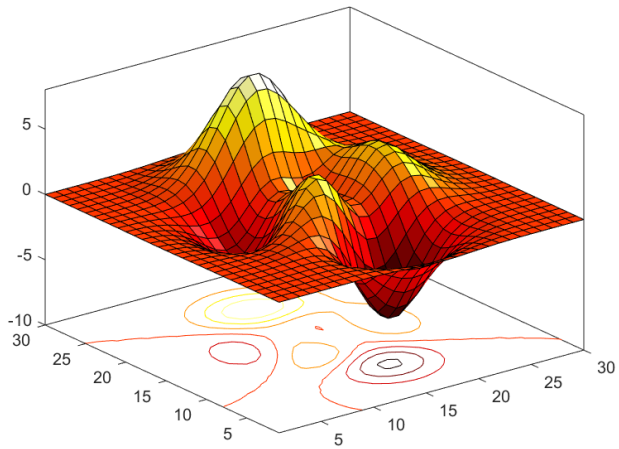
Local minima

- Suppose D is the domain of all feasible points, If there is a neighborhood of x^* ,

$$U(\mathbf{x}^*) = \left\{ \mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{x}^*\| < \delta \right\} (\delta > 0)$$

, such that $f(x) \geq f(x^*), \forall x \in U(x^*) \cap D$, then x^* is a local minimum of $f(\cdot)$

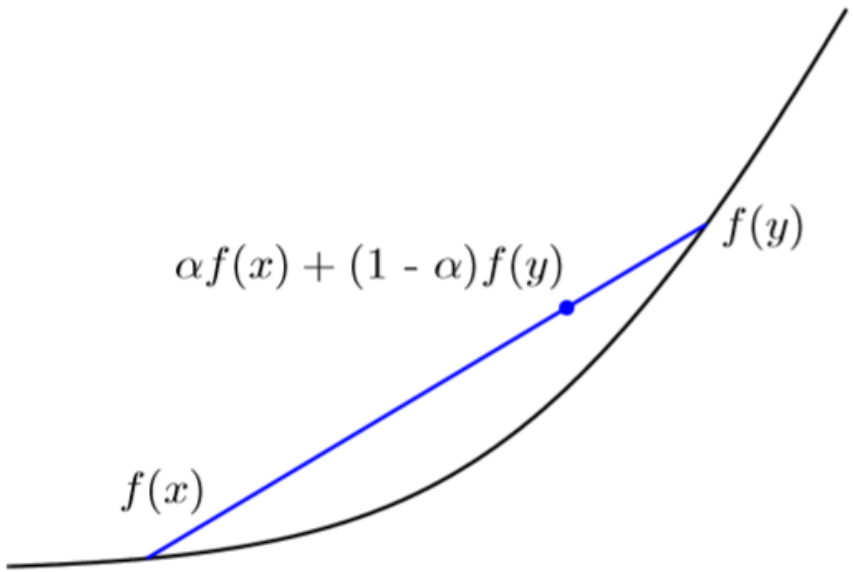
- Local minima may or may not be global minima.



Convex functions: definition

- A function $f : \mathbf{R}^n \to \mathbf{R}$ is convex if dom f is a convex set and if for all x, $y \in$ dom f, and θ with $0 \leq \theta \leq 1$, we have

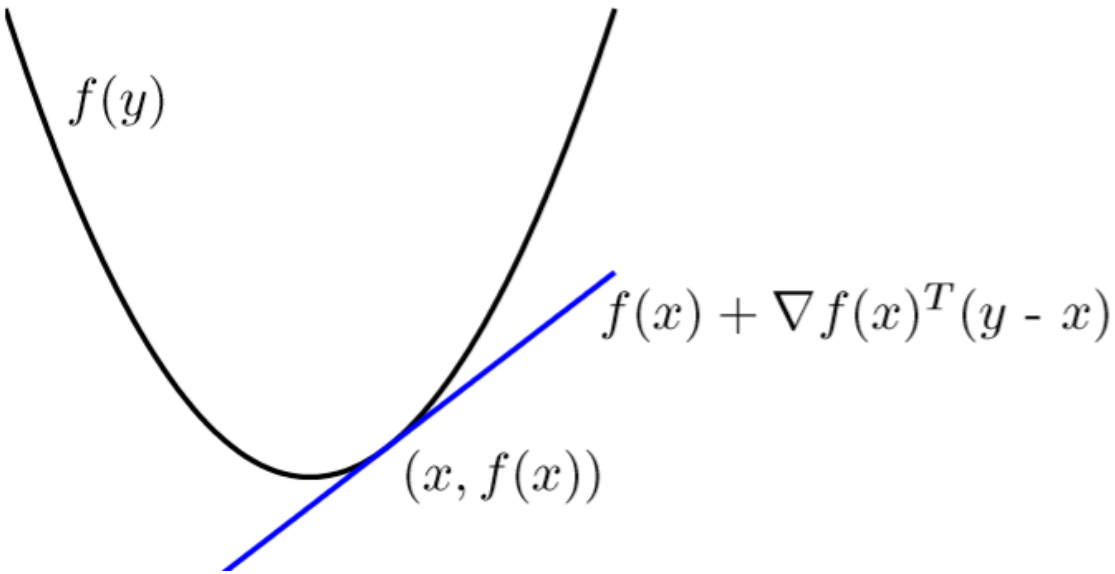
$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$$



Convex functions: First order convexity conditions

- Suppose f is differentiable (*i.e.*, its gradient ∇f exists at each point in dom f, which is open). Then f is convex if and only if

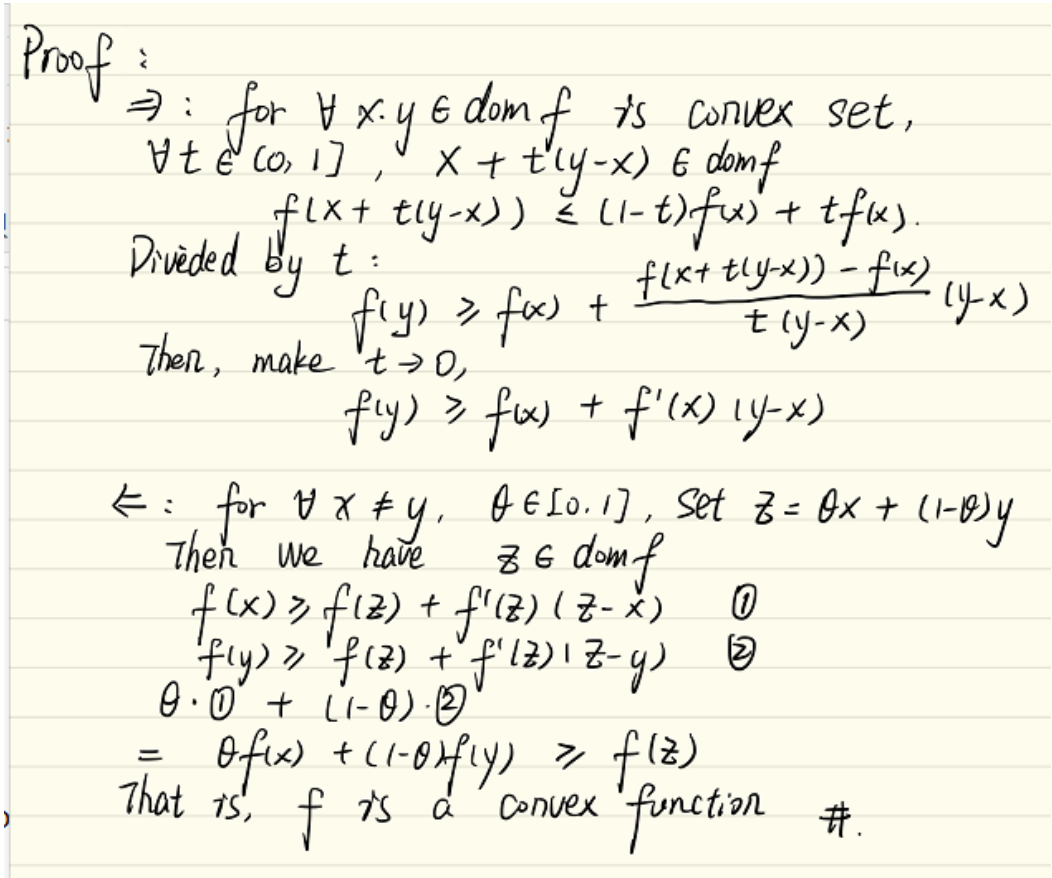
$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$



Convex functions: First order convexity conditions

- Suppose f is differentiable at $x^\star$(*i.e.*, its gradient ∇f exists at each point in dom f, which is open). Then f is convex if and only if dom f is convex and

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$



Convex functions: First order convexity conditions

- Suppose f is differentiable, if $x^\star$ is a local minima, $f'(x^\star) = 0$

Proof:

$\Rightarrow$: Suppose $x^* = \arg\min_{x \in D} f(x)$, we get $x^* \pm \Delta x$

$\Delta x \in \partial B(0,1)$,

Then $f(x^* - \Delta x) \geq f(x^*)$

$f(x^* + \Delta x) \geq f(x^*)$

So we have $\lim_{\Delta x \to 0} \frac{f(x^* + \Delta x) - f(x^*)}{\Delta x} \geq 0$

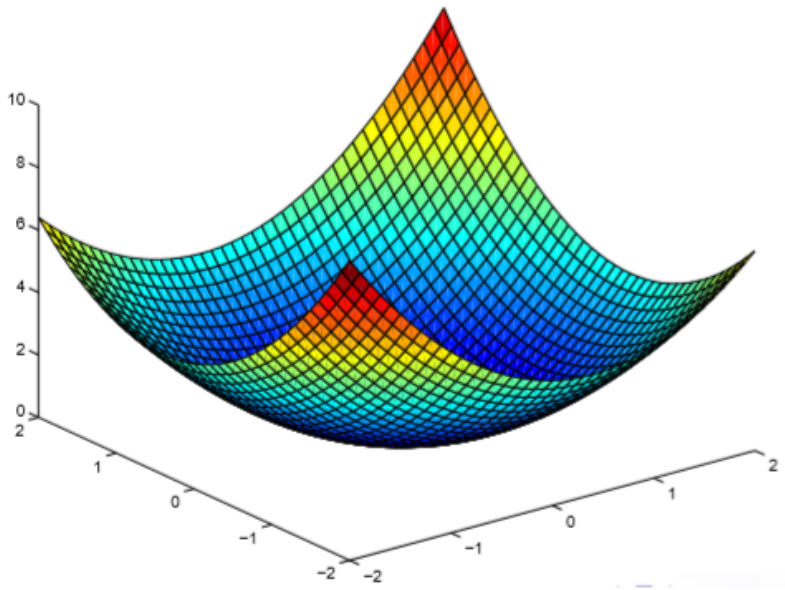
$\lim_{\Delta x \to 0} \frac{f(x^*) - f(x^* - \Delta x)}{\Delta x} \leq 0$

As for $f'(x^*)$ exist, so we have $f'(x^*) = 0$

Convex functions: Second order convexity conditions

- We now assume that f is twice differentiable, that is, its Hessian or second derivative $\nabla^2 f$ exists at each point in $\operatorname{dom} f$, which is open. Then f is convex if and only if $\operatorname{dom} f$ is convex and its Hessian is positive semidefinite: for all $x \in \operatorname{dom} f$,

$$\nabla^2 f(x) \succeq 0$$



Convex functions: Second order convexity conditions

- We now assume that f is twice differentiable, that is, its Hessian or second derivative $\nabla^2 f$ exists at each point in $\operatorname{dom} f$, which is open. Then f is convex if and only if $\operatorname{dom} f$ is convex and its Hessian is positive semidefinite: for all $x \in \operatorname{dom} f$,

$$\nabla^2 f(x) \succeq 0$$

Proof:

$\Rightarrow$: $\forall\, p \in \mathbb{R}^n$, let $\lambda > 0$, $\forall\, x \in \operatorname{dom} f$

$$f(x+\lambda p) = f(x) + \lambda \nabla f^T(x)\cdot p + \tfrac{1}{2}\lambda^2 p^T \nabla^2 f(x) p + o(\lambda^2\|p\|^2)$$

$$\geq f(x) + \lambda \nabla f^T(x) p \quad \text{(first order conditions)}$$

$$\Rightarrow \quad p^T \nabla^2 f(x) p + o(\lambda^2\|p\|^2)/\tfrac{1}{2}\lambda^2 \geq 0$$

let $\lambda \to 0$

We have $p^T \nabla^2 f(x) p \geq 0$

$\therefore\ \nabla^2 f(x) \succeq 0$

$\Leftarrow$: $\forall\, x, y \in \operatorname{dom} f$, by intermediate value theom. There is $x_m \in (x, y)$ s.t.

$$f(y) = f(x) + \nabla^T f(x)(y-x) + \tfrac{1}{2}(y-x)^T \nabla^2 f(x_m)(y-x)$$

$$\geq f(x) + \nabla^2 f(x)(y-x)$$

According to first order conditions, f is a convex function. #

Convex functions: Proofs

- Suppose f is a convex function, $x^\star$ is the local minima, then $x^\star$is the global minima.

Proof:

Assume x^* is a local minima for f

There is a $U(x^*)$ s.t.

$$\forall x \in U(x^*),\ f(x) \geq f(x^*)$$

Then, for $\forall y \in \text{dom} f$

we could find $t > 0$ small enough to make

$$(1-t)x^* + ty \in U(x^*)$$

That's

$$f((1-t)x^* + t(y)) \geq f(x^*)$$

As for f is a convex function,

$$f(x^*) \leq f((1-t)x^* + ty) \leq (1-t)f(x^*) + t(y)$$

$$\Rightarrow f(x^*) \leq f(y) \quad \text{for } \forall y \in \text{dom} f$$

So, x^* is the global minima. #

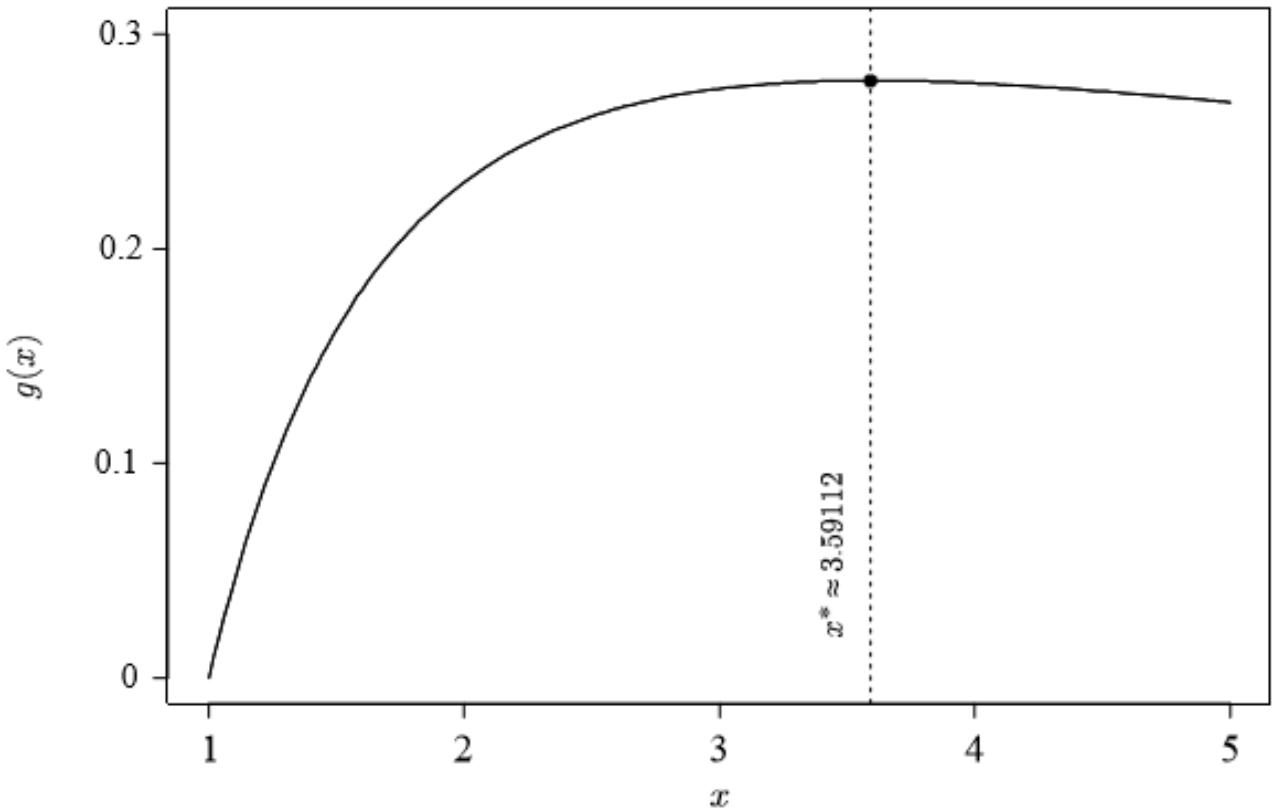
Gradient and Hessian Matrix

- Gradient:
  if a function $f(x)$ is differentiable, its gradient vector is
  $$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_d} \right)^T.$$
- Hessian Matrix($H_f(x)$ or $H(x)$):
  if f (x) has second derivatives,
  $$\nabla^2 f(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{\substack{i=1,\ldots,d \\ j=1,\ldots,d}}$$
  .
- If all the second derivatives are continuous, the Hessian matrix is symmetrix.

Table of Contents

A simple example

- maximize $g(x) = \frac{\log x}{1+x}, x \in \mathbb{R}_{++}$, no analytic solution can be found, so we resort to iterative method to approximate a solution.



- Starting Value(an initial guess): $x^{(0)} = 3.0$.

Bisection method

- The update may be based on an attempt to find the root of

$$g'(x) = \frac{1 + 1/x - \log x}{(1+x)^2}$$

.

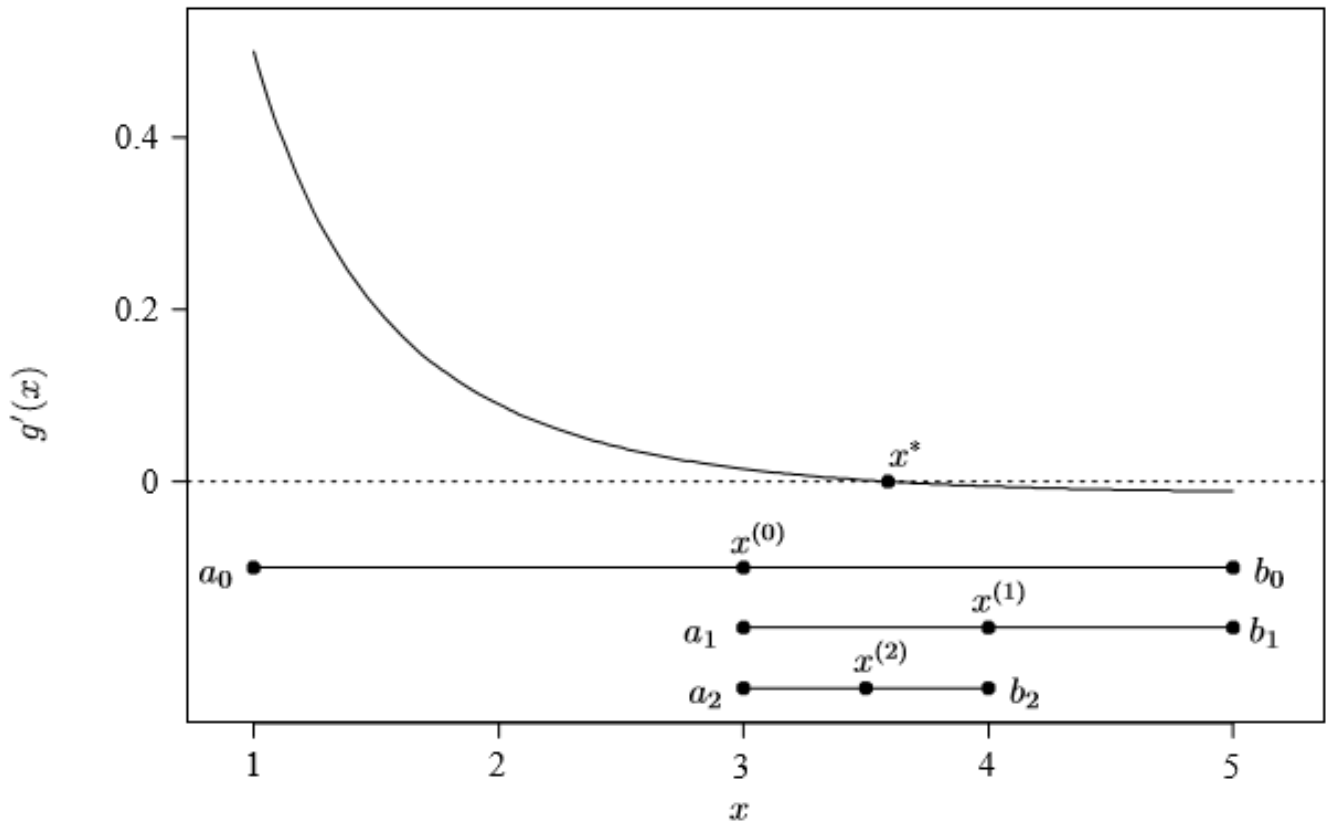
- Bisection Method: if $\nabla g(x)$ is continuous in interval $[a_0, b_0]$ and $\nabla g(a_0) \nabla g(b_0) \leq 0$, then by the intermediate value theorem, there exists $x^* \in [a_0, b_0]$ such that $\nabla g(x^*) = 0$.
- Let $x^{(0)} = (a_0 + b_0)/2$ be the starting value. The updating equations are

$$[a_{t+1}, b_{t+1}] = \begin{cases} \left[a_t, x^{(t)}\right] & \text{if } g'(a_t) g'\left(x^{(t)}\right) \leq 0 \\ \left[x^{(t)}, b_t\right] & \text{if } g'(a_t) g'\left(x^{(t)}\right) > 0 \end{cases},$$

and

$$x^{(t+1)} = \frac{1}{2}(a_{t+1} + b_{t+1}).$$

Bisection method



- stopping rule based on some convergence criteria: monitor $x^{(t+1)}$ or $g'(x^{(t+1)})$?

Stopping Rule

- absolute convergence criterion

$$\left| x^{(t+1)} - x^{(t)} \right| < \epsilon,$$

- relative convergence criterion

$$\frac{\left| x^{(t+1)} - x^{(t)} \right|}{\left| x^{(t)} \right|} < \epsilon,$$

- Preference between above

$$\frac{\left| x^{(t+1)} - x^{(t)} \right|}{\left| x^{(t)} \right| + \epsilon} < \epsilon,$$

  ϵ is predetermined.

- Other Rule to kill the convergence.

Convergence Order

- consider the speed of the root-finding approach.
  Define $\epsilon^{(t)} = x^{(t)} - x^*$ A method has comvergence of order β, if $\lim_{t\to\infty} \epsilon^{(t)} = 0$ and,

$$\lim_{t\to\infty} \frac{\left|\epsilon^{(t+1)}\right|}{\left|\epsilon^{(t)}\right|^{\beta}} = c,$$

  for some constants $c \neq 0$ and $\beta > 0$.
- For bisection, $\beta = 1$ is quite slow.
- Methods for generating a reasonable starting value include graphing, preliminary estimates, educated guesses, and trial and error.

Table of Contents

Gradient Descent

The simplest algorithm in the world (almost).

- Goal:

$$\arg\min_{\mathbf{x}\in D} f(\mathbf{x})$$

- we simply iterate,

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha^{(t)} \nabla f\left(\mathbf{x}^{(t)}\right).$$

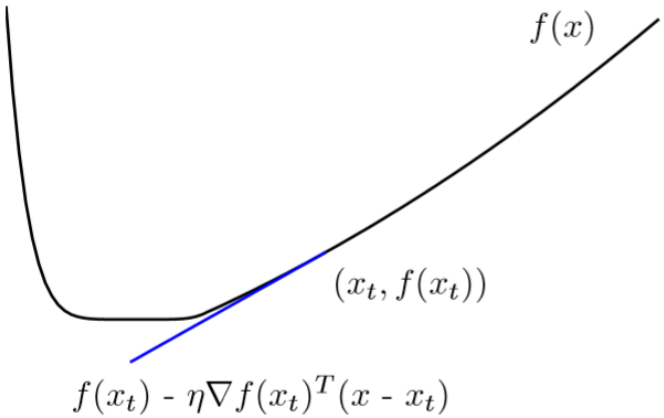
  where $\alpha^{(t)}$ is the stepsize.

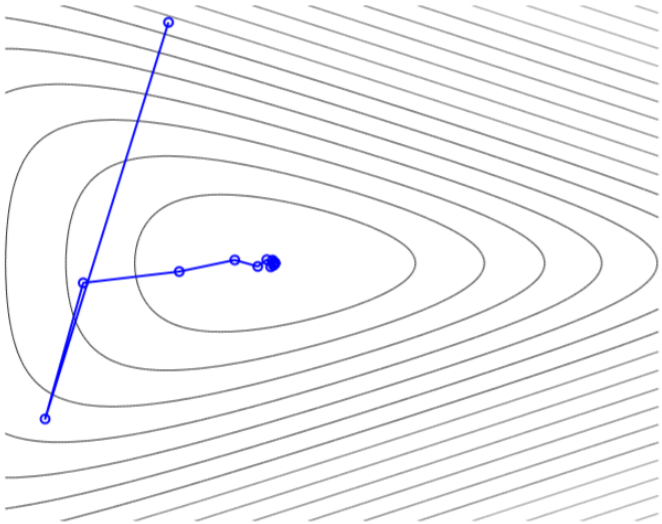
Gradient Descent

- also known as steepest descent. by Taylor expansion of $f(x^{(t)} + \alpha d)$ at the last update $x^{(t)}$

$$f(x^{(t)} + \alpha d) = f(x^{(t)}) + \alpha \nabla f(x^{(t)}) d + 0(\alpha)$$

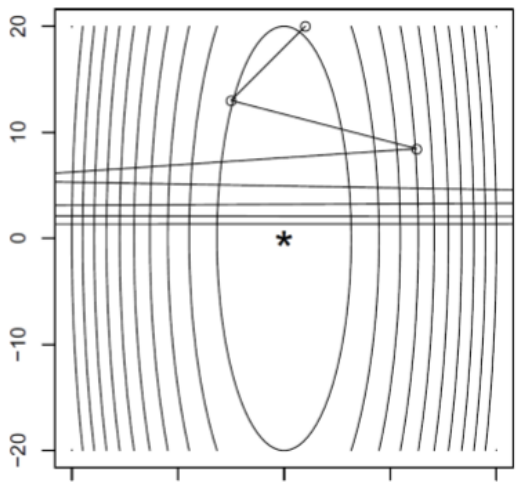
where $d \in \partial B(0,1)$is the direction vector, $\alpha \in \mathbb{R}$.
According to the Cauchy-inequality:$|\nabla f(x^{(t)}) d| \leq |\nabla f(x^{(t)})| \cdot |d|$.
to maximize $|\nabla f(x^{(t)}) d|$ is equal to make $d = \nabla f(x^{(t)})$

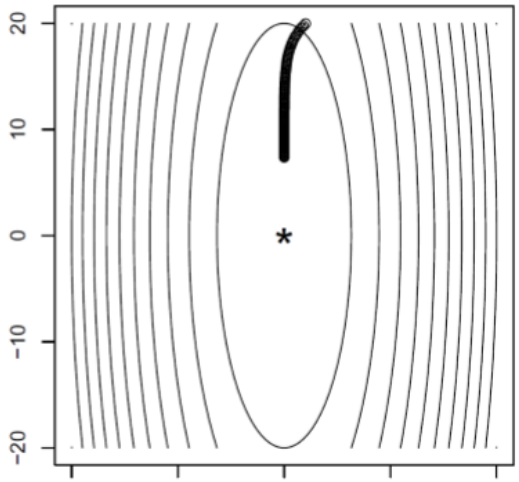




Stepsize Matters

- if $\alpha^{(t)} > 0$ is too big, the algorithm may divergence.
- if $\alpha^{(t)} > 0$ is too small, the algorithm can be too slow.
- consider $f(x) = \left(10x_1^2 + x_2^2\right)/2$,



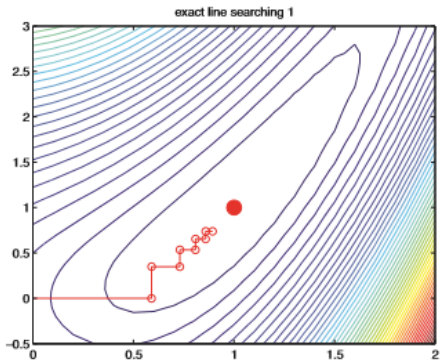


Stepsize Matters

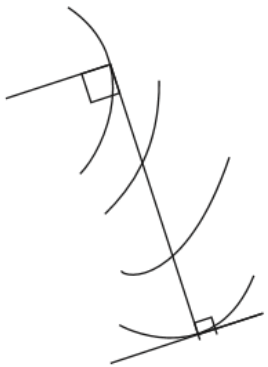
- Step having:
  1)Start with initial $\alpha^{(0)}$.
  2)if this $\alpha^{(0)}$ can not make the objective function $f(x)$ downhill, then we set $\alpha^{(0)} = \alpha^{(0)}/2$.
  3)repeat until the $\alpha^{(0)}$ make the objective function downhill.
- line search methods:
  we want to pick α to minimize $f(\mathbf{x} - \alpha \nabla f(\mathbf{x}))$.

$$f(\mathbf{x} + \alpha d) \approx f(\mathbf{x}) + \alpha \mathbf{g}^T d,$$

where d is our descent direction, $\mathbf{g} = f'(\mathbf{x} + \alpha d)$.



(a)

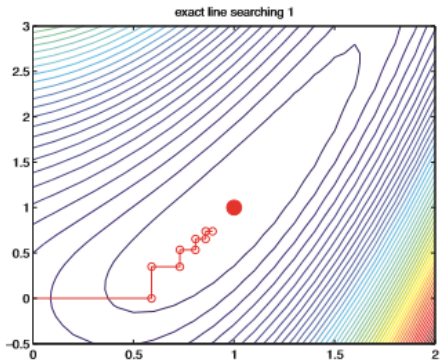


(b)

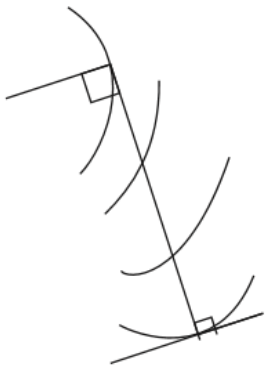
Stepsize Matters

- line search methods:
  we want to pick α to minimize $\varphi(\alpha) = f(\mathbf{x} - \alpha \nabla f(\mathbf{x}))$.
  A necessary condition for the optimum is $\varphi'(\alpha) = 0$
  That is,

$$d^T g = 0 \Leftrightarrow g = 0 \text{ or } g \perp d$$



(a)



(b)

The subgradient method

- The subgradient method is a simple algorithm for minimizing a non differentiable convex function.
- Suppose $f : \mathbf{R}^n \to \mathbf{R}$ is convex. To minimize f, the subgradient method uses the iteration

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

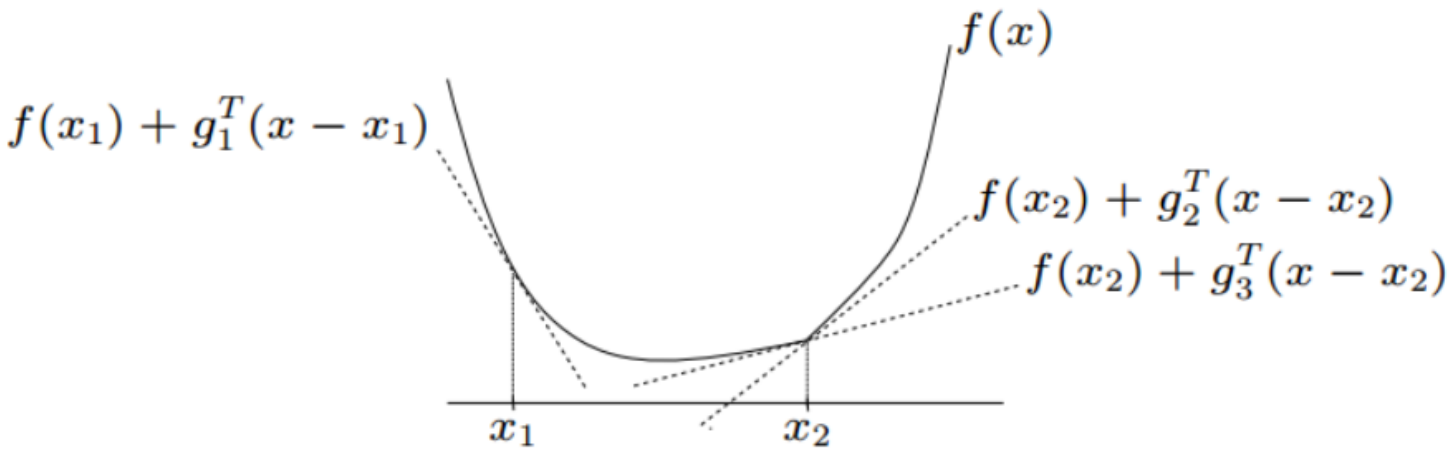
Here $x^{(k)}$ is the k th iterate, $g^{(k)}$ is any subgradient of f at $x^{(k)}$, and $\alpha_k > 0$ is the k th step size.

The subgradient method

- The subgradient set, or subdifferential set, $\partial f(x)$ of f at x is

$$\partial f(x) = \{g : f(y) \geq f(x) + g^T(y - x) \text{ for all } y\},$$

When f is differentiable, the only possible choice for g is $\nabla f(x)$,



The subgradient method

- subgradient method is not a descent method
- Keep track of the best point found. at each step, we set

$$f_{\text{best}}^{(k)} = \min\left\{f_{\text{best}}^{(k-1)}, f\left(x^{(k)}\right)\right\}$$

and $i_{\text{best}}^{(k)} = k$ if $f(x^k) = f_{\text{best}}^{(k)}$.
Then we have:

$$f_{\text{best}}^{(k)} = \min\left\{f\left(x^{(1)}\right), f\left(x^{(2)}\right), \ldots, f\left(x^{(k)}\right)\right\}$$

- (Some step size rules and convergence rules can be found in supplement.)

Proof of convergence for subgradient descent

Idea: bound $\|x_{t+1} - x^*\|$ using subgradient inequality. Assume that $\|g_t\| \leq G$

$$\begin{aligned}\|x_{t+1} - x^*\|^2 &= \|x_t - \eta g_t - x^*\|^2 \\ &= \|x_t - x^*\|^2 - 2\eta g_t^T (x_t - x^*) + \eta^2 \|g_t\|^2\end{aligned}$$

Recall that

$$f(x^*) \geq f(x_t) + g_t^T (x^* - x_t) \Rightarrow \quad -g_t^T (x_t - x^*) \leq f(x^*) - f(x_t)$$

so

$$\|x_{t+1} - x^*\|^2 \leq \|x_t - x^*\|^2 + 2\eta [f(x^*) - f(x_t)] + \eta^2 G^2$$

Then

$$f(x_t) - f(x^*) \leq \frac{\|x_t - x^*\|^2 - \|x_{t+1} - x^*\|^2}{2\eta} + \frac{\eta}{2} G^2$$

Proof of convergence for subgradient descent

Sum from $t = 1$ to T :

$$\sum_{t=1}^{T} f(x_t) - f(x^*) \leq \frac{1}{2\eta} \sum_{t=1}^{T} \left[\|x_t - x^*\|^2 - \|x_{t+1} - x^*\|^2 \right] + \frac{T\eta}{2} G^2$$

$$= \frac{1}{2\eta} \|x_1 - x^*\|^2 - \frac{1}{2\eta} \|x_{T+1} - x^*\|^2 + \frac{T\eta}{2} G^2$$

Now let $D = \|x_1 - x^*\|$, and keep track of min along run,

$$f(x_{\text{best}}) - f(x^*) \leq \frac{1}{2\eta T} D^2 + \frac{\eta}{2} G^2$$

Set $\eta = \frac{D}{G\sqrt{T}}$ and

$$f(x_{\text{best}}) - f(x^*) \leq \frac{DG}{\sqrt{T}}$$

Table of Contents

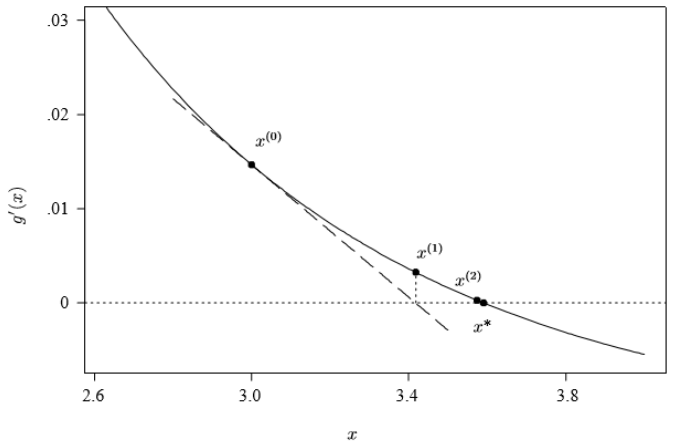
Newton Raphson iteration

- Suppose the objective function is $g(x), \nabla g(x)$ exists and continuous and $\nabla^2 g(x^*) \neq 0$.
  Then by Taylor expansion of $\nabla g(x)$ at the last update $x^{(t)}$,

$$0 = g'(x^*) \approx g'\left(x^{(t)}\right) + \left(x^* - x^{(t)}\right) g''\left(x^{(t)}\right).$$

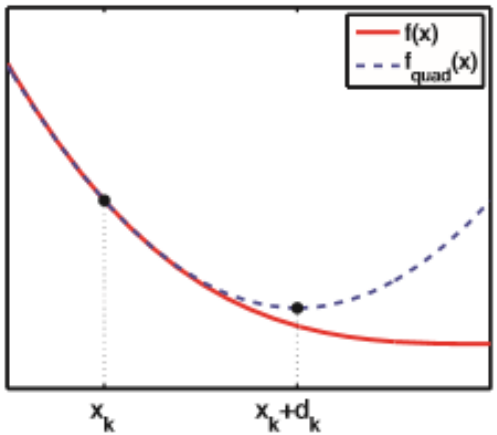
  One can update x by

$$x^{(t+1)} = x^{(t)} - \frac{g'\left(x^{(t)}\right)}{g''\left(x^{(t)}\right)} = x^{(t)} + h^{(t)}.$$

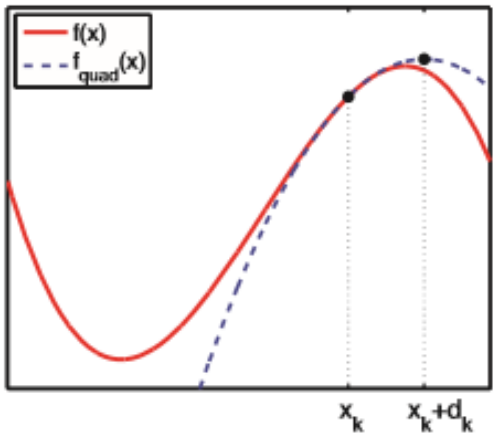


Newton Raphson iteration

- Suppose the objective function is $g(x), \nabla g(x)$ exists and continuous and $\nabla^2 g\left(x^*\right) \neq 0$
- Another optimize view: by Taylor expansion of $g(x)$ at the last update $x^{(t)}$,
  $g_{quad}(x) = g(x^t) + g'\left(x - x^{(t)}\right) + \frac{1}{2}\left(x - x^{(t)}\right)^2 g''\left(x^{(t)}\right)$.



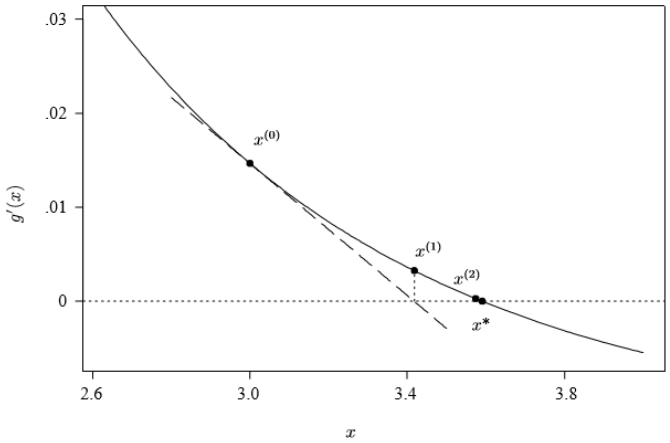
(a)

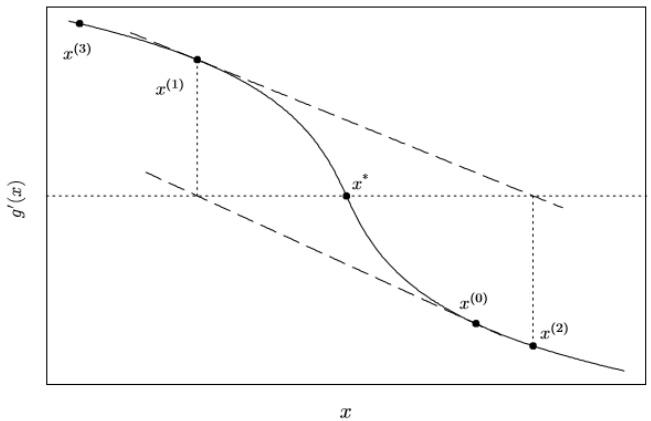


(b)

Convergences

- Whether Newton Raphson method converges depends on the shape of f and the starting value.





- To better understand what ensures convergence, we must carefully analyze the errors at successive steps.(Appendix.)
- If $\nabla f(x)$ is twice continuously differentiable, it is convex, and has a root, then Newton's method converges to the root from any starting point.

Convergences

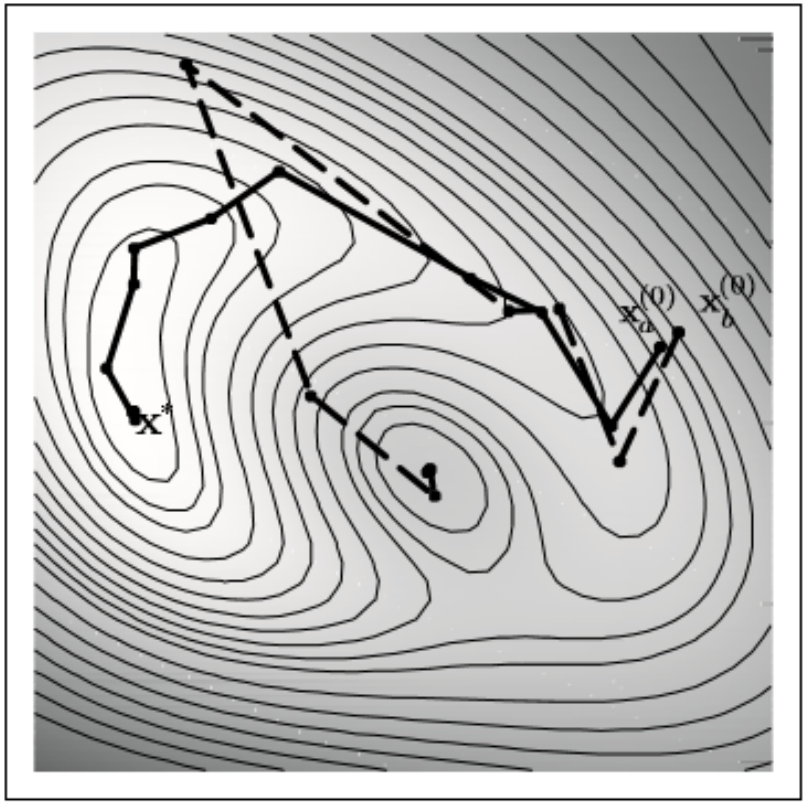


FIGURE 2.7 Application of Newton's method for maximizing a complicated bivariate function, as discussed in Example 2.4. The surface of the function is indicated by shading and contours, with light shading corresponding to high values. Two runs starting from $\mathbf{x}_a^{(0)}$ and $\mathbf{x}_b^{(0)}$ are shown. These converge to the true maximum and to a local minimum, respectively.

Convergence Order

define $\epsilon^{(t)} = x^{(t)} - x^*$, A Taylor expansion yields:

$$0 = g'(x^*) = g'\left(x^{(t)}\right) + \left(x^* - x^{(t)}\right) g''\left(x^{(t)}\right) + \frac{1}{2}\left(x^* - x^{(t)}\right)^2 g'''(q),$$

for some q between $x^{(t)}$ and x^*. Rearranging terms, we find

$$x^{(t)} + h^{(t)} - x^* = \left(x^* - x^{(t)}\right)^2 \frac{g'''(q)}{2g''\left(x^{(t)}\right)}.$$

Then,

$$\epsilon^{(t+1)} = \left(\epsilon^{(t)}\right)^2 \frac{g'''(q)}{2g''\left(x^{(t)}\right)}.$$

that's,

$$\frac{\epsilon^{(t+1)}}{\left(\epsilon^{(t)}\right)^2} = \frac{g'''(q)}{2g''\left(x^{(t)}\right)}.$$

- convergence order is $\beta = 2$.

Fisher Scoring

- In an MLE problem, $\theta^{(t+1)} = \theta^{(t)} - \frac{l'\left(\theta^{(t)}\right)}{l''\left(\theta^{(t)}\right)}$.

  the negative Hessian matrix $l''\left(\theta^{(t)}\right)$ can be replaced by the expected Fisher's Information $-I(\theta)$,

$$\theta^{(t+1)} = \theta^{(t)} + l'\left(\theta^{(t)}\right) I\left(\theta^{(t)}\right)^{-1}.$$

- Generally, Fisher scoring works better in the beginning to make rapid improvements, while Newton's method works better for refinement near the end.

Secant Method

- The second derivative may be hard to calculate, it can be replaced by the numerical differentiation:

$$x^{(t+1)} = x^{(t)} - g'\left(x^{(t)}\right) \frac{x^{(t)} - x^{(t-1)}}{g'\left(x^{(t)}\right) - g'\left(x^{(t-1)}\right)}.$$

- This approach requires two starting points.
- Convergence order is around $\beta = 1.62$, slower than Newton method.
- The converging conditions akin to those for Newton method.

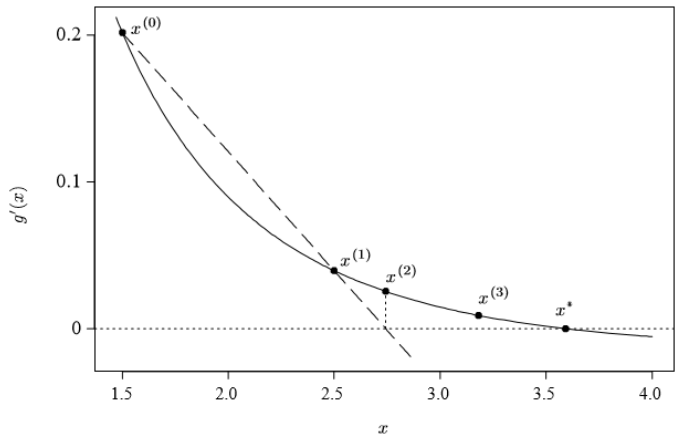


Table of Contents

Newton-Like Methods

- Newton methods are based on the quadratic Taylor expansion of the objective function $g(x)$.
- The hard part is how to calculate the Hessian matrix. It may be computationally expensive to evaluate the Hessian.
- The steps taken by Newton's method are not necessarily always downhill: at each iteration, we cannot guarantee that $g\left(x^{(t+1)}\right) < g\left(x^{(t)}\right)$.
- Some very effective methods rely on updating equations of the form:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \left(\mathbf{M}^{(t)}\right)^{-1} \mathbf{g}'\left(\mathbf{x}^{(t)}\right),$$

where $\mathbf{M}^{(t)}$ is a $p \times p$ matrix approximating the Hessian, $\mathbf{g}''\left(\mathbf{x}^{(t)}\right)$.

Descent Algorithm

- Denote $\mathrm{h}(t)$ the increment in the $(t+1)$ th step of Newton method, i.e.

$$\mathbf{h}(t) = -\alpha^{(t)} \left[M_t\right]^{-1} \nabla g\left(\mathbf{x}^{(t)}\right),$$

  where $\alpha^{(t)} > 0$ is the stepsize, M_t is the Hessian or its replacement.
- Note that: For any fixed $\mathbf{x}^{(t)}$ and positive definite $\mathbf{M}^{(t)}$, note that as $\alpha^{(t)} \to 0$ we have

$$\begin{aligned} g\left(\mathbf{x}^{(t+1)}\right) - g\left(\mathbf{x}^{(t)}\right) &= g\left(\mathbf{x}^{(t)} + \mathbf{h}^{(t)}\right) - g\left(\mathbf{x}^{(t)}\right) \\ &= -\alpha^{(t)} \mathbf{g}'\left(\mathbf{x}^{(t)}\right)^{\mathrm{T}} \left(\mathbf{M}^{(t)}\right)^{-1} \mathbf{g}'\left(\mathbf{x}^{(t)}\right) + o\left(\alpha^{(t)}\right) \end{aligned}$$

  where the second equality follows from the linear Taylor expansion $g\left(\mathbf{x}^{(t)} + \mathbf{h}^{(t)}\right) = g\left(\mathbf{x}^{(t)}\right) + \mathbf{g}'\left(\mathbf{x}^{(t)}\right)^{\mathrm{T}} \mathbf{h}^{(t)} + o\left(\alpha^{(t)}\right)$.
- If M_t is PD, descency can be assured by choosing $\alpha^{(t)}$ properly, yielding $g\left(x^{(t+1)}\right) - g\left(x^{(t)}\right) < 0$.

Quasi Newton Method

- The calculating of the Hessian matrix may be hard.
- The strategy for numerically approximating the Hessian by $\mathbf{M^{(t)}}$ is a computationally burdensome one.
  the (i, j) th element of $\mathbf{M}^{(t)}$ to equal

$$\mathbf{M}_{ij}^{(t)} = \frac{g_i'\left(\mathbf{x}^{(t)} + h_{ij}^{(t)}\mathbf{e}_j\right) - g_i'\left(\mathbf{x}^{(t)}\right)}{h_{ij}^{(t)}}.$$

  for some constants $h_{ij}^{(t)}$.
- How about update the $\mathbf{M}^{(t)}$ by incorporate the information about the curvature of $\mathbf{g}$ in the the direction of $h^{(t)}$ near x^t?

Quasi Newton Method

(1) M_t is PD so that $\mathbf{d}^{(t)} \triangleq -M_t^{-1}\nabla f\left(\mathbf{x}^{(t)}\right)$ is downhill.
(2) M_{t+1} is obtained by M_t iteratively,

$$M_{t+1} = M_t + \Delta M_t,$$

where ΔM_t is some correction.
(3) M_{t+1} must satisfy,

$$M_{t+1}\delta^{(t)} = \zeta^{(t)},$$

where $\delta^{(t)} \triangleq \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}, \zeta^{(t)} \triangleq \nabla f\left(\mathbf{x}^{(t+1)}\right) - \nabla f\left(\mathbf{x}^{(t)}\right).$

Quasi Newton Method

Suppose now we have M_t and $x^{(t+1)}$ after several iterations, we need to get M_{t+1}. Consider the quadratic Taylor expansion of $f(x)$ at $x^{(t+1)}$,

$$\begin{aligned} f(\mathrm{x}) \approx & f\left(\mathrm{x}^{(t+1)}\right)+\nabla f\left(\mathrm{x}^{(t+1)}\right)^{T}\left(\mathrm{x}-\mathrm{x}^{(t+1)}\right) \\ & +\frac{1}{2}\left(\mathrm{x}-\mathrm{x}^{(t+1)}\right)^{T} \nabla^{2} f\left(\mathrm{x}^{(t+1)}\right)\left(\mathrm{x}-\mathrm{x}^{(t+1)}\right). \end{aligned}$$

which means that,

$$\nabla f(x) \approx \nabla f\left(x^{(t+1)}\right)+\nabla^{2} f\left(x^{(t+1)}\right)\left(x-x^{(t+1)}\right).$$

Now let $x=x^{(t)}$,

$$\nabla f\left(\mathbf{x}^{(t)}\right) \approx \nabla f\left(\mathbf{x}^{(t+1)}\right)+\nabla^{2} f\left(\mathbf{x}^{(t+1)}\right)\left(\mathbf{x}^{(t)}-\mathbf{x}^{(t+1)}\right).$$

Then, $\nabla^{2} f\left(\mathbf{x}^{(t+1)}\right)\left(\mathbf{x}^{(t+1)}-\mathbf{x}^{(t)}\right) \approx \nabla f\left(\mathbf{x}^{(t+1)}\right)-\nabla f\left(\mathbf{x}^{(t)}\right)=\zeta^{(t)}$.

BFGS Method

- In the BFGS,

$$M_{t+1} = M_t + \frac{\zeta^{(t)} \left(\zeta^{(t)}\right)^T}{\left(\zeta^{(t)}\right)^T \delta^{(t)}} - \frac{\left(M_t \delta^{(t)}\right) \left(M_t \delta^{(t)}\right)^T}{\left(\delta^{(t)}\right)^T M_t \delta^{(t)}}.$$

- Usually we use $M_0 = I$ as an initial value, it can be shown that $\{M_t\}$ is always PD and the direction of $-H_t^{-1} \nabla f\left(\mathbf{x}^{(t)}\right)$ is always downhill.

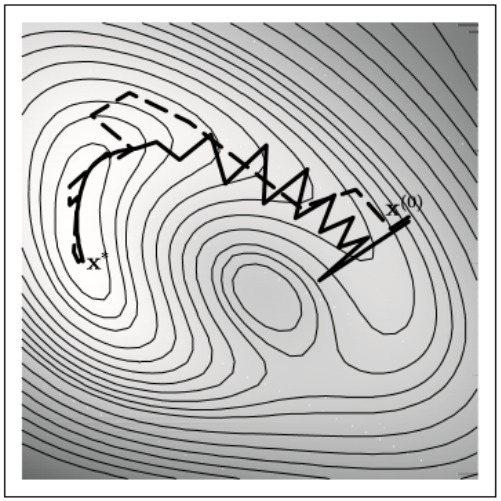


FIGURE 2.8 Applications of two optimization methods for maximizing a complex bivariate function. The surface of the function is indicated by shading and contours, with light shading corresponding to high values. The two methods start at a point $\mathbf{x}^{(0)}$ and find the true maximum, $\mathbf{x}^*$. The solid line corresponds to the method of steepest ascent (Example 2.6). The dashed line corresponds to a quasi-Newton method with the BFGS update (Example 2.7). Both algorithms employed backtracking, with the initial value of $\alpha^{(t)}$ at each step set to 0.25 and 0.05, respectively.

Nonlinear Gauss Seidel Iteration

- Alternatively referred to as backfitting or cyclic coordinate descent.
- For $j = 1, ..., p$, Gauss Seidel iteration proceeds by viewing the j-th component of $\mathrm{g}^{'}$ as a univariate real function of x_j only.
- All p components are cycled through solving for the one-dimensional root of $g_j^{'}(x_j^{t+1})$.

$$x_j^{k+1} = argmin_{y \in \mathbb{R}} g(x_1^{(k+1)}, \ldots x_{j-1}^{(k+1)}, y, x_{j+1}^{(k)}, \ldots, x_p^{(k)})$$

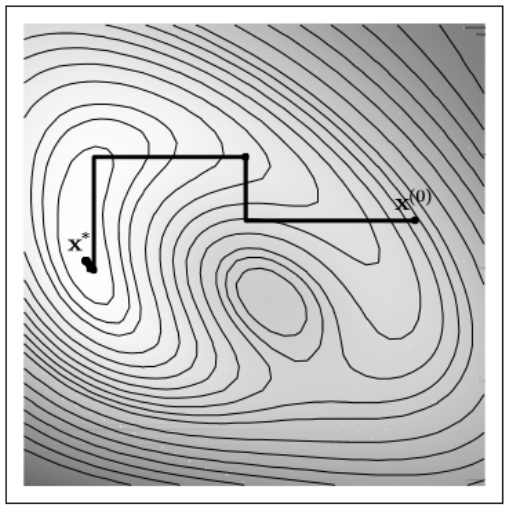


FIGURE 2.14 Application of Gauss–Seidel iteration for maximizing a complex bivariate function, as discussed in Example 2.10. The surface of the function is indicated by shading and contours. From the starting point of $\mathbf{x}^{(0)}$, several steps are required to approach the true maximum, $\mathbf{x}^*$. Each line segment represents a change of a single coordinate in the current solution, so complete steps from $\mathbf{x}^{(t)}$ to $\mathbf{x}^{(t+1)}$ correspond to pairs of adjacent segments.

Table of Contents

Discriminative approach

We can generalize linear regression to the (binary) classification setting by making two changes.

- First we replace the Gaussian distribution for y with a Bernoulli distribution, which is more appropriate for the case when the response is binary, $y \in \{0, 1\}$. That is, we use

$$p(y \mid \mathbf{x}, \mathbf{w}) = \text{Ber}(y \mid \mu(\mathbf{x})),$$

  where $\mu(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}] = p(y = 1 \mid \mathbf{x})$.
- Second, we compute a linear combination of the inputs, as before, but then we pass this through a function that ensures $0 \leq \mu(\mathbf{x}) \leq 1$ by defining

$$\mu(\mathbf{x}) = \text{sigm}\left(\mathbf{w}^T \mathbf{x}\right).$$

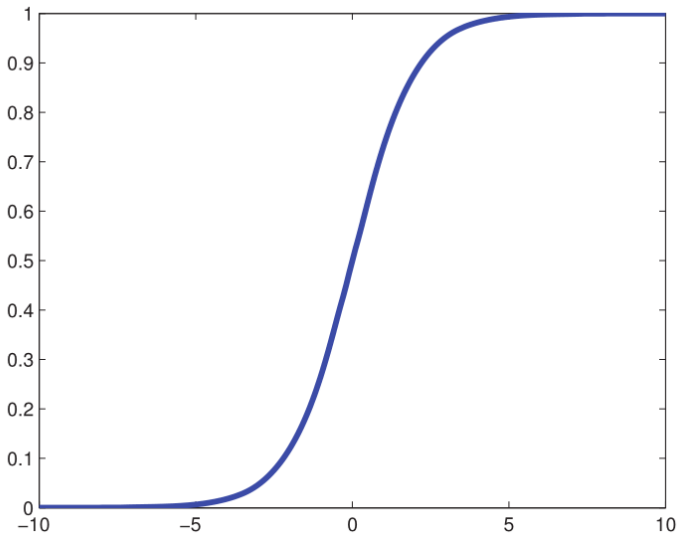
  where $\text{sigm}(\eta)$ refers to the sigmoid function, also known as the logistic or logit function.

Discriminative approach

Sigmoid function is defined as

$$\text{sigm}(\eta) \triangleq \frac{1}{1+\exp(-\eta)} = \frac{e^{\eta}}{e^{\eta}+1}.$$

The term "sigmoid" means S-shaped:



It is also known as a squashing function, since it maps the whole real line to $[0, 1]$, which is necessary for the output to be interpreted as a probability.

Discriminative approach

Putting these two steps together we get

$$p(y \mid \mathbf{x}, \mathbf{w}) = \text{Ber}\left(y \mid \text{sigm}\left(\mathbf{w}^T \mathbf{x}\right)\right).$$

This is called logistic regression due to its similarity to linear regression (although it is a form of classification, not regression!).

$$p(y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}^T \mathbf{x})}{1 + \exp(\mathbf{w}^T \mathbf{x})},$$

$$p(y = 0 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}.$$

Discriminative approach

The negative log-likelihood(NLL) for logistic regression is given by

$$\begin{aligned}\mathrm{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N} \log\left[\mu_i^{\mathbb{I}(y_i=1)}\left(1-\mu_i\right)^{\mathbb{I}(y_i=0)}\right] \\ &= -\sum_{i=1}^{N}\left[y_i \log \mu_i + (1-y_i)\log\left(1-\mu_i\right)\right],\end{aligned}$$

where $\mu_i \triangleq p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w})$. This is also called the cross-entropy error function.

Discriminative approach

$$\begin{aligned}\mathrm{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N}\left[y_i \log \mu_i + (1-y_i)\log\left(1-\mu_i\right)\right] \\ &= -\sum_{i=1}^{N}\left[y_i \log \frac{\exp(\mathbf{w}^T\mathbf{x})}{1+\exp(\mathbf{w}^T\mathbf{x_i})} + (1-y_i)\log\frac{1}{1+\exp(\mathbf{w}^T\mathbf{x})}\right].\end{aligned}$$

MLE

Unlike linear regression, we can no longer write down the MLE in closed form. Instead, we need to use an optimization algorithm to compute it. For this, we need to derive the gradient and Hessian.

$$\mu_i \triangleq p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \frac{\exp(\mathbf{w}^T \mathbf{x}_i)}{1 + \exp(\mathbf{w}^T \mathbf{x}_i)} = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_i)},$$

$$\begin{aligned}
\frac{\partial \mu_i}{\partial \mathbf{w}} &= \frac{-exp(-\mathbf{w}^T \mathbf{x}_i)(-\mathbf{x}_i)}{(1 + exp(-\mathbf{w}^T \mathbf{x}_i)^2)} \\
&= \frac{-exp(-\mathbf{w}^T \mathbf{x}_i)}{1 + exp(-\mathbf{w}^T \mathbf{x}_i)} \frac{1}{1 + exp(-\mathbf{w}^T \mathbf{x}_i)} \mathbf{x}_i \\
&= (1 - \mu_i)\mu_i \mathbf{x}_i.
\end{aligned}$$

MLE

$$\mathrm{NLL}(\mathbf{w}) = -\sum_{i=1}^{N} \left[y_i \log \mu_i + (1 - y_i) \log (1 - \mu_i) \right].$$

The gradient of the NLL:

$$\begin{aligned}
\mathbf{g}(\mathbf{w}) &= \frac{\partial NLL}{\partial \mathbf{w}} \\
&= \sum_i (y_i \frac{1}{\mu_i} \mu_i (1 - \mu_i) \mathbf{x}_i + (1 - y_i) \frac{-1}{1 - \mu_i} \mu_i (1 - \mu_i) \mathbf{x}_i) \\
&= \sum_i (y_i \mathbf{x}_i (1 - \mu_i) + (y_i - 1) \mu_i \mathbf{x}_i) \\
&= \sum_i (y_i \mathbf{x}_i - y_i \mathbf{x}_i \mu_i + y_i \mathbf{x}_i \mu_i - \mu_i \mathbf{x}_i) \\
&= \sum_i (\mu_i - y_i) \mathbf{x}_i = \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y}).
\end{aligned}$$

MLE

The gradient of the NLL:

$$\mathbf{g}(\mathbf{w}) = \sum_i (\mu_i - y_i)\,\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}).$$

The Hessian of the NLL:

$$\begin{aligned}\mathbf{H}(\mathbf{w}) &= \frac{\partial \mathbf{g}(\mathbf{w})^T}{\partial \mathbf{w}} = \sum_i \frac{\partial \mu_i}{\partial \mathbf{w}} \mathbf{x}_i^T = \sum_i \mu_i\,(1-\mu_i)\,\mathbf{x}_i \mathbf{x}_i^T \\ &= \mathbf{X}^T \mathbf{S} \mathbf{X},\end{aligned}$$

where $\mathbf{S} \triangleq \operatorname{diag}(\mu_i\,(1-\mu_i))$. We can show that $\mathbf{H}$ is positive definite. Hence the NLL is convex and has a unique global minimum. Later we will discuss IRLS for finding this minimum.

Table of Contents

Iteratively reweighted least squares (IRLS)

Let us now apply Newton's algorithm to find the MLE for binary logistic regression. The Newton update at iteration $k+1$ for this model is as follows (using $\eta_k = 1$, since the Hessian is exact):

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k - \mathbf{H}^{-1}\mathbf{g}_k \\
&= \mathbf{w}_k + \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\left(\mathbf{y} - \boldsymbol{\mu}_k\right) \\
&= \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\left[\left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)\mathbf{w}_k + \mathbf{X}^T\left(\mathbf{y} - \boldsymbol{\mu}_k\right)\right] \\
&= \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\left[\mathbf{S}_k\mathbf{X}\mathbf{w}_k + \mathbf{y} - \boldsymbol{\mu}_k\right] \\
&= \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{S}_k\mathbf{z}_k
\end{aligned} \tag{1}$$

where we have defined the working response as

$$\mathbf{z}_k \triangleq \mathbf{X}\mathbf{w}_k + \mathbf{S}_k^{-1}\left(\mathbf{y} - \boldsymbol{\mu}_k\right).$$

Iteratively reweighted least squares (IRLS)

Equation (1) is an example of a weighted least squares problem, which is a minimizer of

$$
\begin{aligned}
Q &= (\mathbf{z_k} - \mathbf{X}\mathbf{w})^T \mathbf{S}_k (\mathbf{z_k} - \mathbf{X}\mathbf{w}). \\
Q &= \mathbf{z_k}^T \mathbf{S}_k \mathbf{z_k} - \mathbf{z_k^T} \mathbf{S}_k \mathbf{X}\mathbf{w} - \mathbf{w^T} \mathbf{X^T} \mathbf{S}_k \mathbf{z_k} + \mathbf{w^T} \mathbf{X^T} \mathbf{S}_k \mathbf{X}\mathbf{w} \\
&= \mathbf{z_k}^T \mathbf{S}_k \mathbf{z_k} - 2\mathbf{z_k^T} \mathbf{S}_k \mathbf{X}\mathbf{w} + \mathbf{w^T} \mathbf{X^T} \mathbf{S}_k \mathbf{X}\mathbf{w}, \\
\frac{\partial Q}{\partial \mathbf{w}} &= -2\mathbf{X}^T \mathbf{S}_k \mathbf{z_k} + 2\mathbf{X}^T \mathbf{S}_k \mathbf{X}\mathbf{w}, \\
\frac{\partial Q}{\partial \mathbf{w}} &= 0 \Rightarrow \mathbf{w}_{k+1} = \left(\mathbf{X}^T \mathbf{S}_k \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{S}_k \mathbf{z}_k.
\end{aligned}
$$

Iteratively reweighted least squares (IRLS)

since $\mathbf{S}_k$ is a diagonal matrix, we can rewrite the targets in component form (for each case $i = 1 : N$) as

$$z_{ki} = \mathbf{w}_k^T \mathbf{x}_i + \frac{y_i - \mu_{ki}}{\mu_{ki}(1 - \mu_{ki})}.$$

This algorithm is known as iteratively reweighted least squares or IRLS for short, since at each iteration, we solve a weighted least squares problem, where the weight matrix $\mathbf{S}_k$ changes at each iteration.

Iteratively reweighted least squares (IRLS)

Algorithm 8.2: Iteratively reweighted least squares (IRLS)

$\mathbf{w} = \mathbf{0}_D$;
$w_0 = \log(\overline{y}/(1-\overline{y}))$;
repeat
  $\eta_i = w_0 + \mathbf{w}^T \mathbf{x}_i$;
  $\mu_i = \text{sigm}(\eta_i)$;
  $s_i = \mu_i(1-\mu_i)$;
  $z_i = \eta_i + \frac{y_i - \mu_i}{s_i}$;
  $\mathbf{S} = \text{diag}(s_{1:N})$;
  $\mathbf{w} = (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\mathbf{z}$;
until *converged*;

LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Logistic Regression:

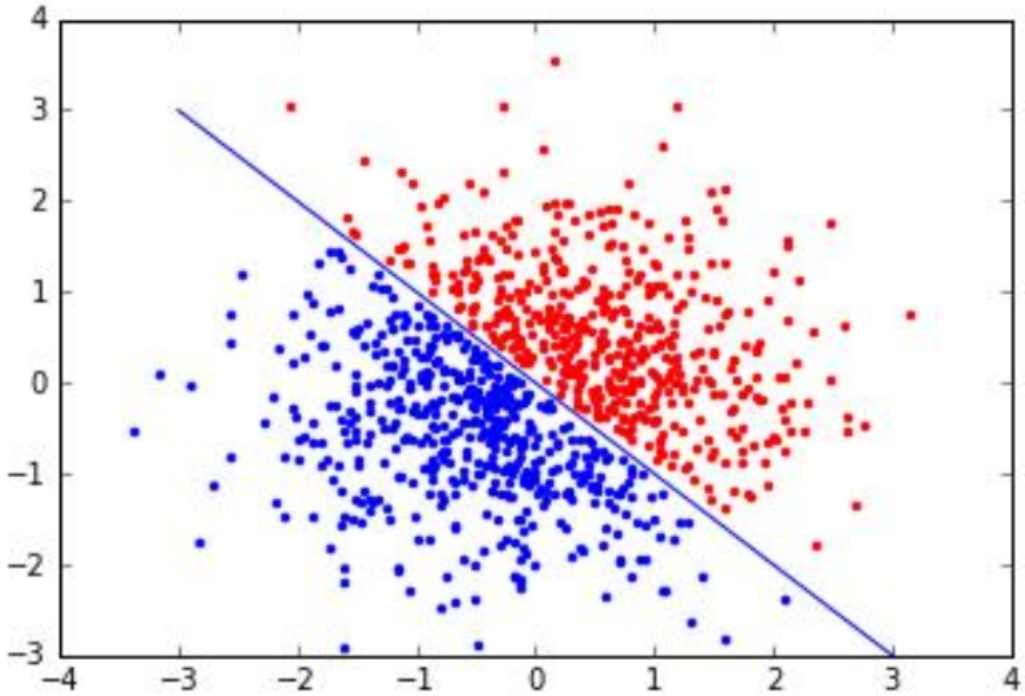
$$p(y=1 \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}^T\mathbf{x})}{1+\exp(\mathbf{w}^T\mathbf{x})}, p(y=0 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(\mathbf{w}^T\mathbf{x})}.$$

$$\Rightarrow L(\mathbf{x}) = log_e\{\frac{p(y=1 \mid \mathbf{x}, \mathbf{w})}{p(y=0 \mid \mathbf{x}, \mathbf{w})}\} = \mathbf{w}^T\mathbf{x}.$$

$$L(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0 \Leftrightarrow \frac{p(y=1 \mid \mathbf{x}, \mathbf{w})}{p(y=0 \mid \mathbf{x}, \mathbf{w})} > 1 \Leftrightarrow p(y=1 \mid \mathbf{x}, \mathbf{w}) > p(y=0 \mid \mathbf{x}, \mathbf{w})$$

LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Logistic Regression:



LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Linear Discriminative Analysis:

$$
\begin{aligned}
f_1(\cdot) &\sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \\
f_2(\cdot) &\sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \\
\boldsymbol{\Sigma}_1 &= \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}_{XX}
\end{aligned}
$$

$$
\frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} = \frac{\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^\tau \boldsymbol{\Sigma}_{XX}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right\}}{\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^\tau \boldsymbol{\Sigma}_{XX}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)\right\}}
$$

$$
\log_e \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} = (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\tau \boldsymbol{\Sigma}_{XX}^{-1}\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\tau \boldsymbol{\Sigma}_{XX}^{-1}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)
$$

LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Linear Discriminative Analysis:

$$L(\mathbf{x}) = \log_e \left\{ \frac{p(y=1 \mid x)}{p(y=0 \mid x)} \right\} = \log_e \left\{ \frac{f_1(\mathbf{x})\pi_1}{f_2(\mathbf{x})\pi_2} \right\} = b_0 + \mathbf{b}^\tau \mathbf{x},$$

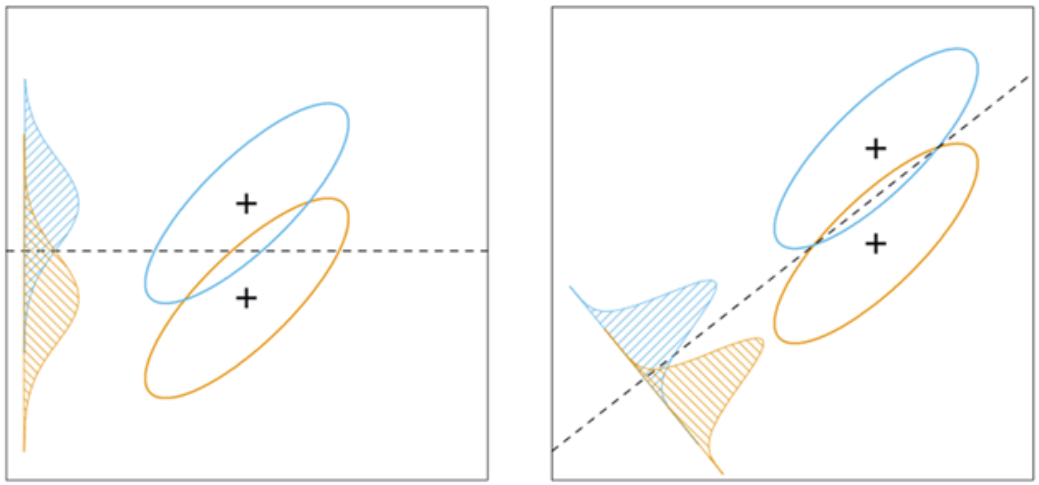
where

$$\mathbf{b} = \boldsymbol{\Sigma}_{XX}^{-1} (\mu_1 - \mu_2),$$
$$b_0 = -\tfrac{1}{2} (\mu_1 - \mu_2)^\tau \boldsymbol{\Sigma}_{XX}^{-1} (\mu_1 + \mu_2) + \log_e \left(\frac{\pi_1}{\pi_2} \right).$$

$y = 1$ if $L(\mathbf{x}) > 0$, other $y = 0$.

LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Linear Discriminative Analysis:



LR(Logistic Regression) and LDA(Linear Discriminative Analysis)

Comparison:

- Their discriminant functions are all linear function.
- Logistic regression has no limit on the distribution of $\mathbf{x}$, but Linear Discriminative Analysis limits $\mathbf{x}$ to a normal distribution.

Table of Contents

Stochastic Gradient Descent

Assume that the parameter to be learned is $\theta \in \mathbb{D}$, cost function is $J(\theta)$, then our goal is:

$$\arg\min_{\theta \in \mathbb{D}} J(\theta) = \arg\min_{\theta \in \mathbb{D}} \sum_{i=1}^{N} J_i(\theta, X^{(i)}, Y^{(i)}).$$

We set that the gradient of the cost function is $\nabla J(\theta)$, learning rate is η .

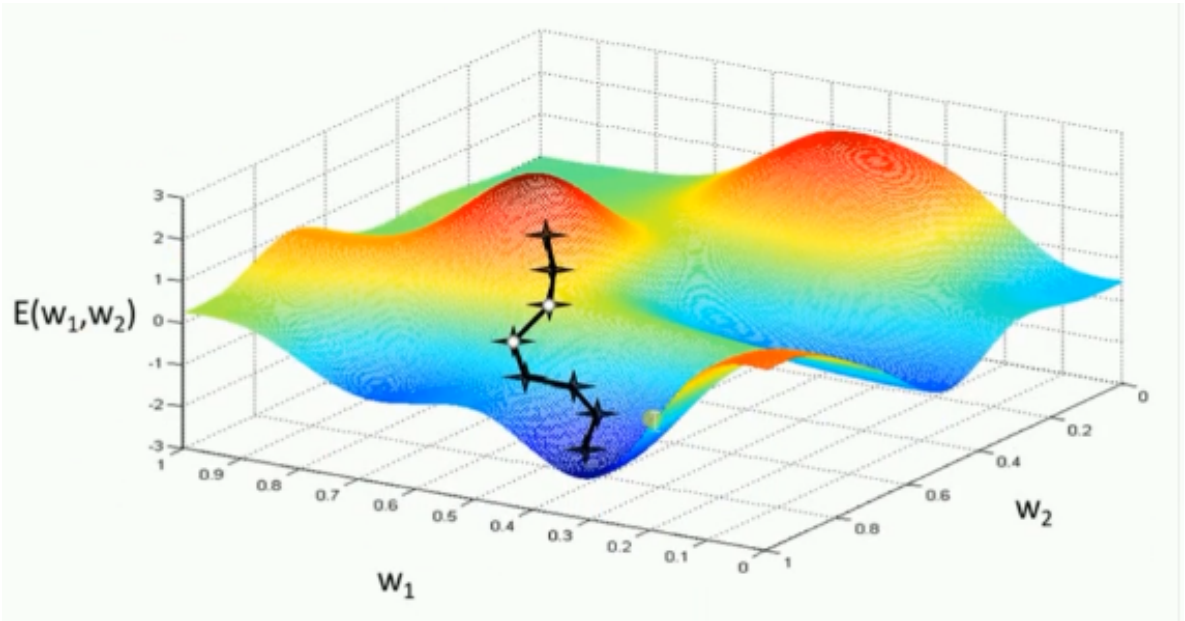
- Gradient Descent:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \eta_t \nabla J\left(\theta_t\right) \\ &= \theta_t - \eta_t \sum_{i=1}^{N} \nabla J_i\left(\theta_t, X^{(i)}, Y^{(i)}\right), \end{aligned}$$

where, θ_t represents the parameter at the t'th iteration.

Stochastic Gradient Descent

- Gradient Descent:



Drawbacks:

- Slow training: every iteration has to go through all the samples.
- It might converge to a local minima.

Stochastic Gradient Descent

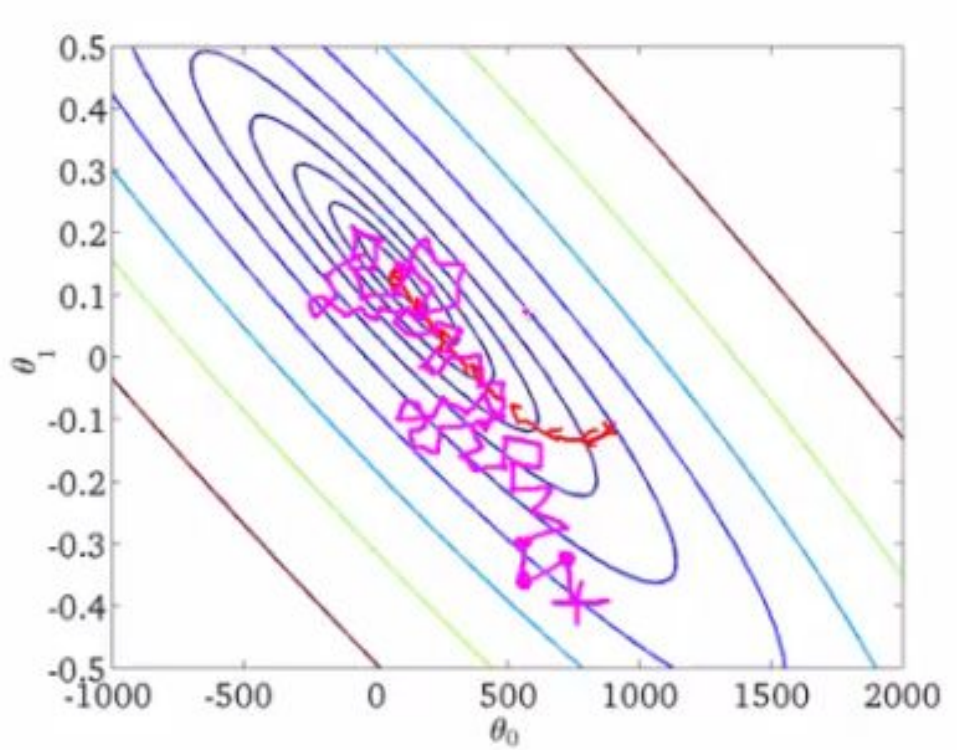
- Stochastic Gradient Descent:
  Suppose a sample i_s is randomly selected from a set of training samples.

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta_t g_t \\ &= \theta_t - \eta_t \nabla J_{i_s}\left(\theta_t; X^{(i_s)}; X^{(i_s)}\right), \quad i_s \in \{1, 2, \ldots, N\}\end{aligned}$$

  η_t : store an initial subset of the data, and try a range of η values on this subset; then choose the one that results in the fastest decrease in the objective and apply it to all the rest of the data.

Stochastic Gradient Descent

- Stochastic Gradient Descent(SGD):



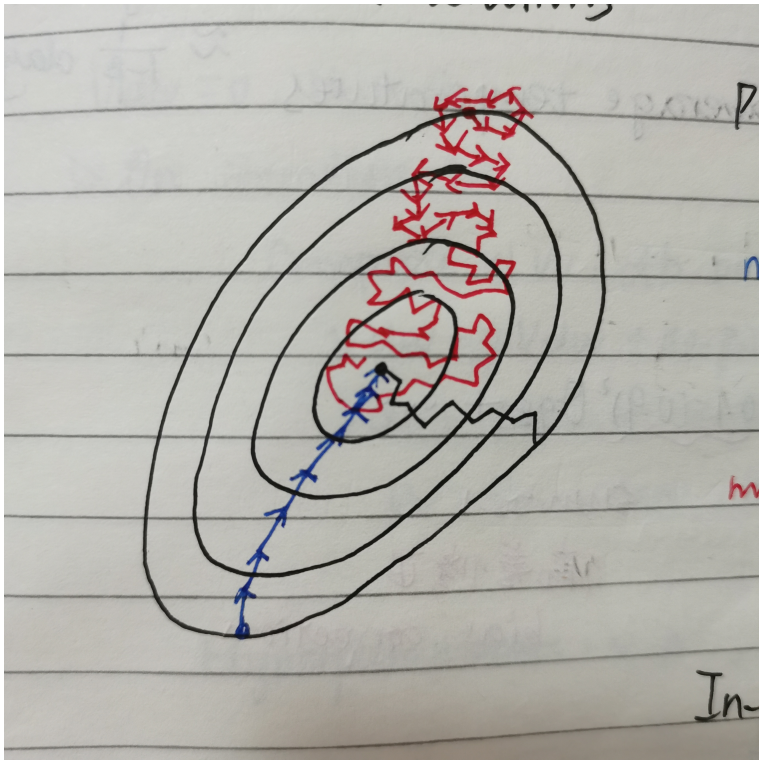
Advantages:

- Faster training.
- More likely to find a global minima.
- Remove redundant information.

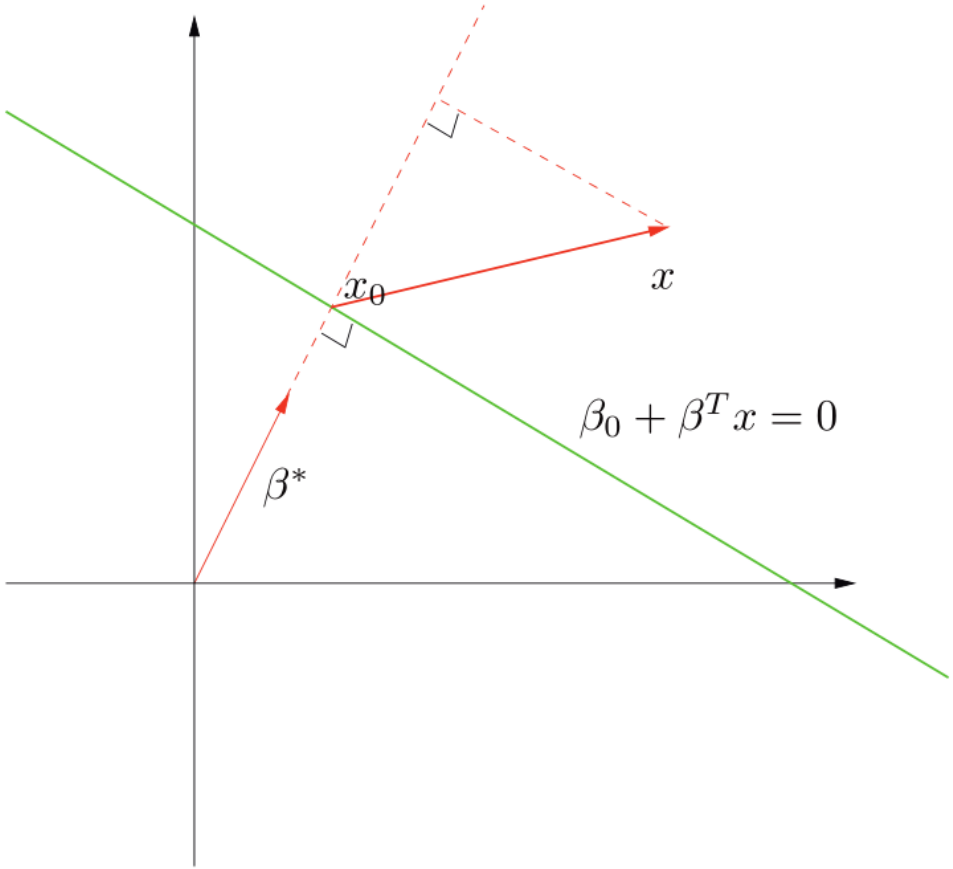
Stochastic Gradient Descent

- Batch Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta_t \nabla \theta \left(\theta_t\right)$$
$$= \theta_t - \eta_t \sum_{i=1}^{n} \nabla J_i \left(\theta_t, X^{(i)}, Y^{(i)}\right), n \leq N.$$

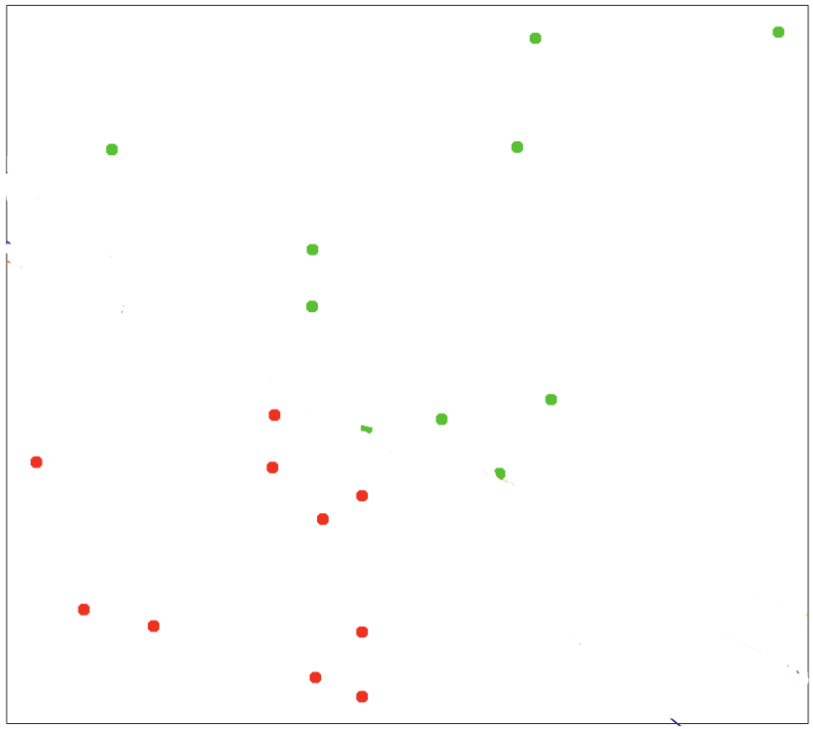


Perceptron Learning Algorithm



$$d = \beta^{*T}(x - x_0) = \frac{1}{\|\beta\|}\left(\beta^T x + \beta_0\right) = \frac{1}{\|f'(x)\|} f(x).$$

Perceptron Learning Algorithm



The perceptron learning algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.

Perceptron Learning Algorithm

If a response $y_i = 1$ is misclassified, then $x_i^T \beta + \beta_0 < 0$, and the opposite for a misclassified response with $y_i = -1$. The goal is to minimize

$$-\sum_{i \in \mathcal{M}} y_i \frac{\left(x_i^T \beta + \beta_0\right)}{\|\beta\|} \Rightarrow D\left(\beta, \beta_0\right) = -\sum_{i \in \mathcal{M}} y_i \left(x_i^T \beta + \beta_0\right),$$

where $\mathcal{M}$ indexes the set of misclassified points. The quantity is nonnegative and proportional to the distance of the misclassified points to the decision boundary defined by $\beta^T x + \beta_0 = 0$. The gradient (assuming $\mathcal{M}$ is fixed) is given by

$$\partial \frac{D\left(\beta, \beta_0\right)}{\partial \beta} = -\sum_{i \in \mathcal{M}} y_i x_i,$$

$$\partial \frac{D\left(\beta, \beta_0\right)}{\partial \beta_0} = -\sum_{i \in \mathcal{M}} y_i.$$

Perceptron Learning Algorithm

The algorithm in fact uses stochastic gradient descent to minimize this piecewise linear criterion.

This means that rather than computing the sum of the gradient contributions of each observation followed by a step in the negative gradient direction, a step is taken after each observation is visited. Hence the misclassified observations are visited in some sequence, and the parameters β are updated via

$$\begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i x_i \\ y_i \end{pmatrix}.$$

Here ρ is the learning rate, which in this case can be taken to be 1 without loss in generality.

Perceptron Learning Algorithm

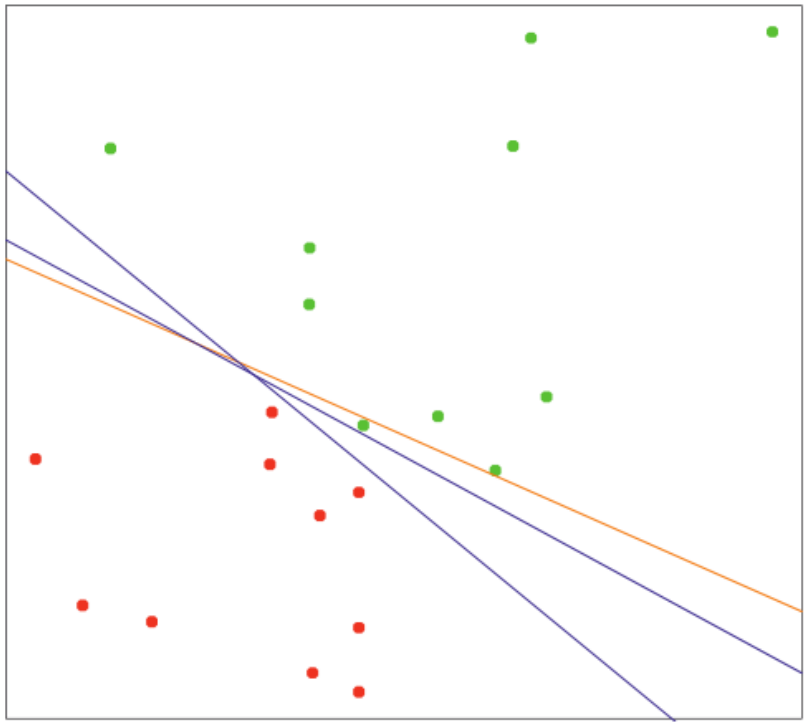


FIGURE 4.14. *A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the* perceptron learning algorithm *with different random starts.*

Figure 4.14 shows two solutions to a toy problem, each started at a different random guess.

Table of Contents

Offline Learning

Traditionally machine learning is performed offline, which means we have a batch of data, and we optimize an equation of the following form

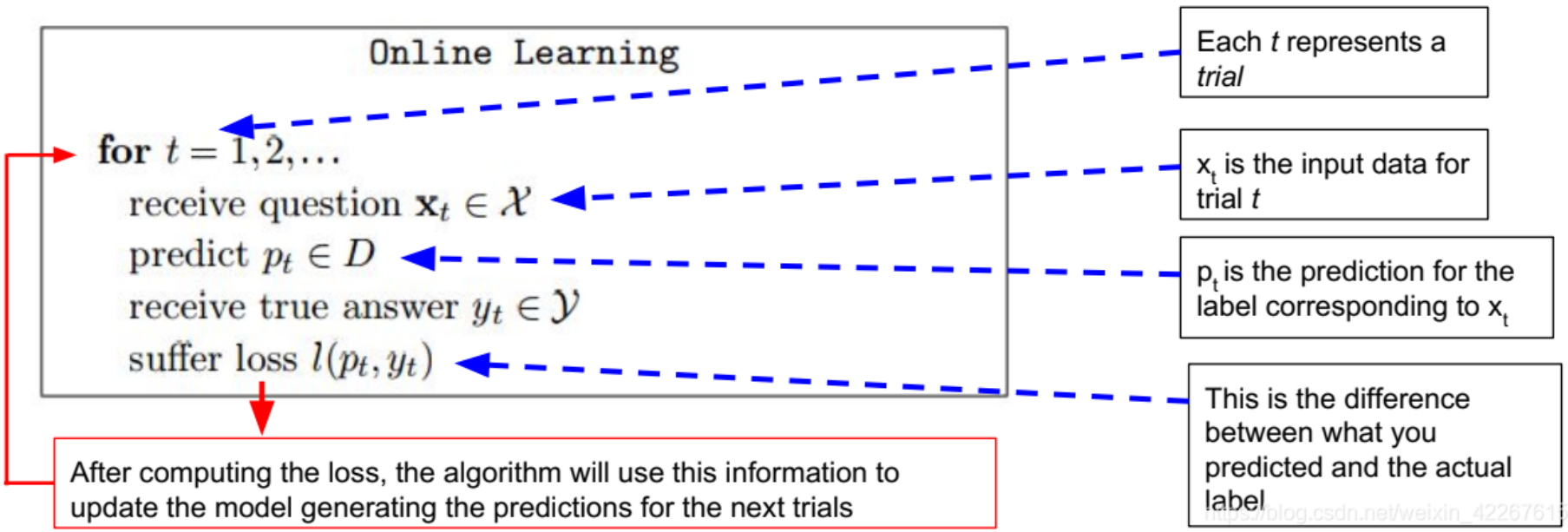
$$f(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} f\left(\boldsymbol{\theta}, \mathbf{z}_i\right),$$

where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ in the supervised case, or just $\mathbf{x}_i$ in the unsupervised case, and $f\left(\boldsymbol{\theta}, \mathbf{z}_i\right)$ is some kind of loss function.

However, if we have streaming data, we need to perform online learning, so we can update our estimates as each new data point arrives rather than waiting until "the end" (which may never occur). And even if we have a batch of data, we might want to treat it like a stream if it is too large to hold in main memory.

Online Learning

In computer science, online machine learning is a method of machine learning in which data becomes available in a sequential order and is used to update our best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once.



Online Learning

Suppose that at each step, "nature" presents a sample $\mathbf{z}_k$ and the "learner" must respond with a parameter estimate $\boldsymbol{\theta}_k$. In the theoretical machine learning community, the objective used in online learning is the regret, which is the averaged loss incurred relative to the best we could have gotten in hindsight using a single fixed parameter value:

$$\text{regret}_k \triangleq \frac{1}{k}\sum_{t=1}^{k} f\left(\boldsymbol{\theta}_t, \mathbf{z}_t\right) - \min_{\boldsymbol{\theta}^* \in \Theta} \frac{1}{k}\sum_{t=1}^{k} f\left(\boldsymbol{\theta}_*, \mathbf{z}_t\right).$$

Online Learning

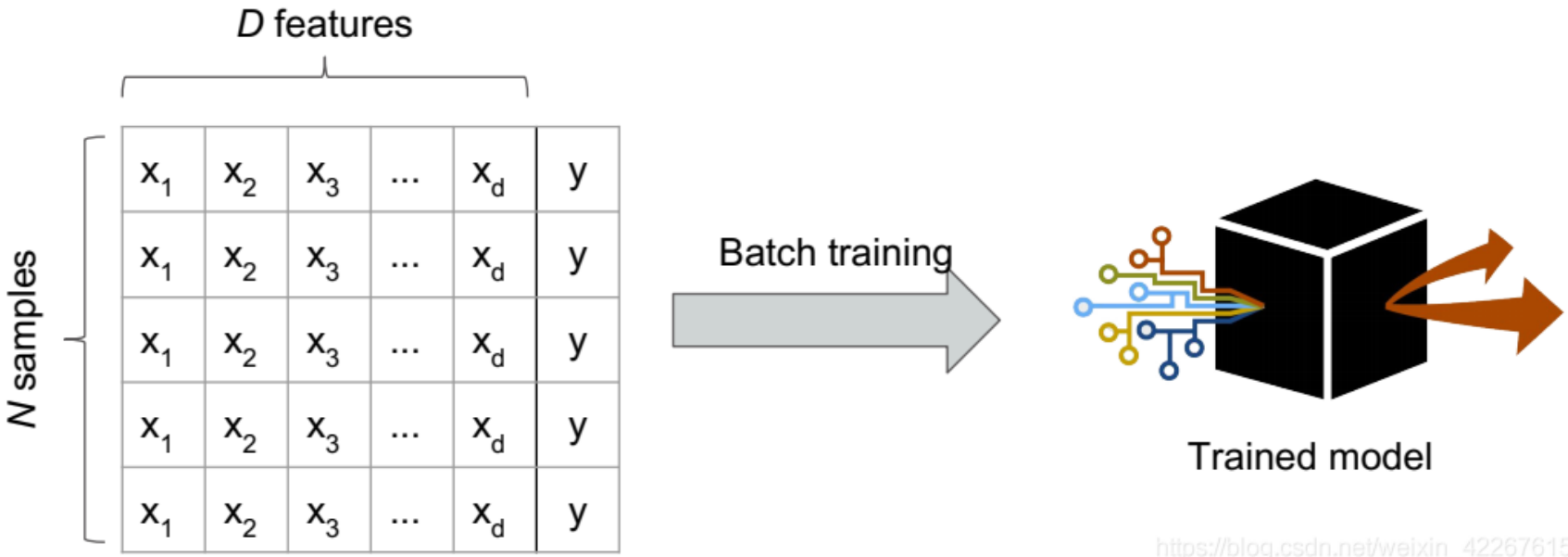
One simple algorithm for online learning is online gradient descent (Zinkevich 2003), which is as follows: at each step k, update the parameters using

$$\boldsymbol{\theta}_{k+1} = \text{proj}_{\Theta}\left(\boldsymbol{\theta}_k - \eta_k \mathbf{g}_k\right)$$

where $\text{proj}_{\mathcal{V}}(\mathbf{v}) = \text{argmin}_{\mathbf{w}\in\mathbf{V}} \|\mathbf{w} - \mathbf{v}\|_2$ is the projection of vector $\mathbf{v}$ onto space $\mathcal{V}, \mathbf{g}_k =, \nabla f\left(\boldsymbol{\theta}_k, \mathbf{z}_k\right)$ is the gradient, and η_k is the step size. (The projection step is only needed if the parameter must be constrained to live in a certain subset of $\mathbb{D}$.)

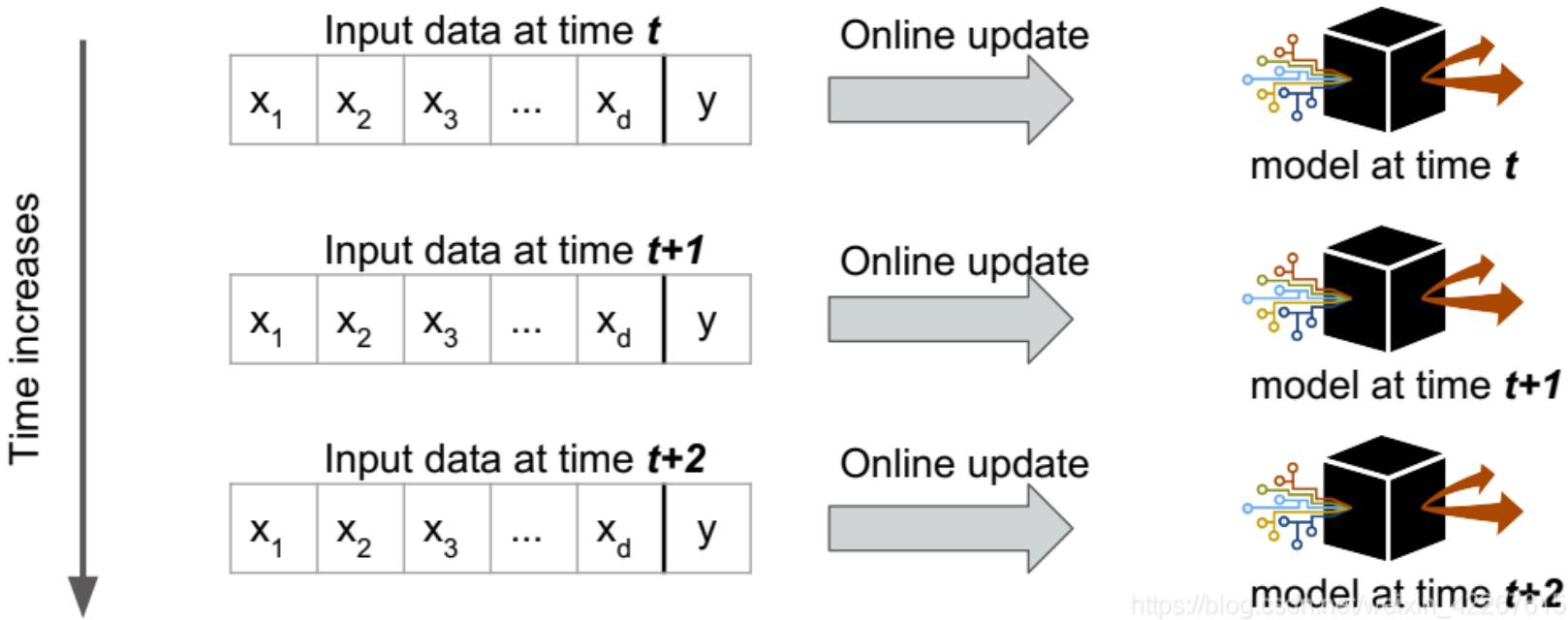
Online Learning

Offline learning:



Online Learning

Online learning:



Momentum optimization method

- Momentum + SGD:
  The main idea is to introduce a cumulative gradient information momentum to accelerate SGD. Take a mini batch from the training set: $\left\{X^{(1)}, X^{(2)}, \ldots, X^{(n)}\right\}$, and their respond variables are $Y^{(i)}$, then

$$\begin{cases} v_t = \alpha v_{t-1} + \eta_t \nabla J\left(\theta_t, X^{(i_s)}, Y^{(i_s)}\right) \\ \theta_{t+1} = \theta_t - v_t \end{cases}$$

  where v_t represents the acceleration that's accumulating at the moment, α represents the magnitude of the momentum, we usually take it as 0.9.

Momentum optimization method

- Nesterov accelerated gradient(NAG)
  Momentum:

$$\begin{cases} v_t = \alpha v_{t-1} + \eta_t \nabla J(\theta_t) \\ \theta_{t+1} = \theta_t - v_t \end{cases}$$

$$\theta_{t+1} = \theta_t - v_t = \theta_t - \alpha v_{t-1} - \eta_t \nabla J(\theta_t)$$

  NAG:

$$\begin{cases} v_t = \alpha v_{t-1} + \eta_t \nabla J(\theta_t - \alpha v_{t-1}) \\ \theta_{t+1} = \theta_t - v_t \end{cases}$$

Adaptive learning rate optimization algorithm

- AdaGrad

$$\theta_{t+1} = \theta_t - \frac{\eta_0}{\sqrt{\sum_{t'=1}^{t}(g_{t'}) + \epsilon}} g_t,$$

where $g_t = \nabla J(\theta_t)$, t represents the iteration time, η_0 represents the initial learning rate which is usually 0.01, ϵ is a very small number (usually 1e-8).

- RMSProp(Root mean square propagation)

$$\begin{cases} E\left[g^2\right]_t = \alpha E\left[g^2\right]_{t-1} + (1-\alpha) g_t^2 \\ \theta_{t+1} = \theta_t - \frac{\eta_0}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{cases}$$

where $g_t = \nabla J(\theta_t)$.

Adaptive learning rate optimization algorithm

- AdaDelta

$$\begin{cases} E\left[g^2\right]_t = \alpha E\left[g^2\right]_{t-1} + (1-\alpha) g_t^2 \\ E\left[\Delta\theta^2\right]_t = \alpha E\left[\Delta\theta^2\right]_{t-1} + (1-\alpha)\Delta\theta_t^2 \\ \Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]}}{\sqrt{E[g^2]_t + \epsilon}} g_t \\ \theta_{t+1} = \theta_t + \Delta\theta_t \end{cases}$$

$$\theta_{t+1} = \theta_t - \frac{J'(\theta)}{J''(\theta)} = \theta_t - \frac{1}{J''(\theta)} J'(\theta),$$

$$\Delta\theta \approx \frac{\frac{\partial J}{\partial \theta}}{\frac{\partial^2 J}{\partial \theta^2}} \Rightarrow \frac{1}{\frac{\partial^2 J}{\partial \theta^2}} = \frac{\Delta\theta}{\frac{\partial J}{\partial \theta}},$$

$$\theta_{t+1} = \theta_t - \frac{1}{\frac{\partial^2 J}{\partial \theta^2}} g_t = \theta_t + \frac{\Delta\theta}{\frac{\partial J}{\partial \theta}} g_t.$$

Adaptive learning rate optimization algorithm

- Adam(Adaptive Moment Estimation)

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \\ \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \end{cases}$$

where, m_t and v_t can be regarded as the first moment estimation and the second moment estimation of the gradient. β_1, β_2 represents the magnitude of the momentum which is usually taken as 0.9 and 0.999 respectively.