The basics of Optimisation in machine learning

Optimisation, whether minimisation or maximisation is a problem of the form: Given $L : \mathbb{R}^n \longrightarrow \mathbb{R}$

objective fnct.

Unconstrained or Constrained
$$\begin{cases} \overset{\text{minimise}}{\cancel{\text{Optimise}}}_{\theta \in \mathbb{R}^n} \; L(\theta) \\ \text{Constraints on } \theta \end{cases}$$

can be equalities or inequalities that θ has to satisfy

"Equivalently" can be written: Find θ^* s.t.

$$\begin{cases} \theta^* = \arg\min_{\theta \in \mathbb{R}^n} L(\theta) \\ \text{Constraints on } \theta \end{cases}$$

$\mathcal{C} = \{ \theta \in \mathbb{R}^n \;/\; \theta \text{ satisfies constraints} \}$ the feasible set

Formal definition: $\theta^* \in \mathcal{C}$ is a solution if and only if for all $\theta \in \mathcal{C}$: $L(\theta^*) \leq L(\theta)$

Does a solution exist, though? Not always!

Think of trying to minimise $L(\theta) = \theta$ over $\mathbb{R}$ or over $\mathcal{C} = (a,b)$. Think about minimising $L(\theta) = \theta^2$ over $\mathcal{C} = \{ \theta \in \mathbb{R} \;/\; \theta < 0 \text{ and } \theta > 1 \}$

To avoid any doubts about the existence of the solution for the rest of these notes, we assume that the minimisation problem is constructed so that a solution always exists. The safest 2 options would be to take: L smooth and

① * L convex ② * L convex, coercive

* $\mathcal{C}$ convex and compact * $\mathcal{C}$ convex, closed

If you are picky and want your solution to be unique, then you can further assume L to be strictly convex.

Quick example: fitting a line to some data points

Consider a set of $N \in \mathbb{N}$ data points $\{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R} \times \mathbb{R}$. We can find $\theta^* = (a^*, b^*) \in \mathbb{R}^2$ s.t $L(\theta) = \frac{1}{2}\sum_{i=1}^{N} |ax_i + b - y_i|^2$ is minimised. We just need to find the critical points of L, i.e. $\nabla_\theta L(\theta) = 0$.

$$\frac{\partial L}{\partial a} = \sum x_i (ax_i + b - y_i) = 0 \ , \quad \frac{\partial L}{\partial b} = \sum (ax_i + b - y_i) = 0$$

Eventually gives:

$$a^* = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{N \sum x_i^2 - (\sum x_i)^2} \ , \quad b^* = \frac{\sum y_i - a\sum x_i}{N}$$

Tip: watch out for vertical lines!

More to explore:

* Linear Programming,
* Lagrange Multiplier Method.

Relation of optimisation to machine learning?

In supervised learning, given a data set $\{(x_i,y_i)\}_{i=1}^{N} \subset X \times Y$ the goal is to find some $f^*: X \to Y$ s.t the size (1) of the error $y - f^*(x)$ is small (2) for any data point $(x,y) \in X \times Y$

(1). the size of the error is expressed using loss functions
$$\begin{cases} l: Y \times Y \longrightarrow \mathbb{R}^+ \\ (y, f^*(x)) \longmapsto l(y, f^*(x)) \end{cases}$$

an example : $l(y, f^*(x)) = \| y - f^*(x) \|_2^2$

taking $f^*(x) = f^*_\theta(x) = ax + b$ where $\theta(a,b)$

takes us back to the previous example.

(2). Think of the data points as a random sample from a joint probability distribution Passumed to be independent and identically distributed

$$D_n(P) = \{(x_1,y_1), \dots, (x_N, y_N)\}$$

the expression "for any data point" means drawing a random data point according to this distribution.

→ Optimality can be measured through the expected loss; commonly known as the expected risk

$$R(f^*) = \mathbb{E}[\ell(y, f^*(x))] = \int_{X \times Y} \ell(y, f^*(x))\, dp(x,y)$$

→ an optimal f^* $\Longleftrightarrow$ minimising $R(f^*)$

Cannot do this since p is unknown. We resort to its approximation, the empiracl risk

$$L(\theta) = \mathbb{E}[\ell(y, f^*_\theta(x))] \approx \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f^*_\theta(x_i))$$

this then will be our objective function.

Now that the relation is understood, for the rest of these notes we go back to the abstract formulation

$$\begin{cases} \min\limits_{\theta \in \mathbb{R}^n} L(\theta) \\ \text{s.t } \theta \in \mathcal{C} \end{cases}$$

How to solve such problems? There are some direct methods which tend to be expensive. We are interested in iterative methods which are more suitable for machine learning.

The Gradient Descent (GD).

We want to solve

$$\min_{\theta \in \mathbb{R}^n} L(\theta)$$

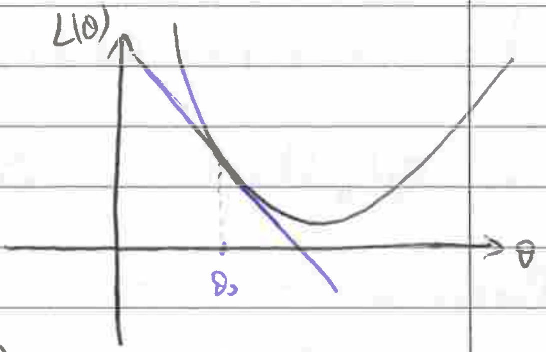
L might be difficult to handle. Let's then approximate it using Taylor expansion of the first order around some point θ_0:

Linear!

$$L(\theta) \approx L(\theta_0) + \langle \nabla L(\theta_0), \theta - \theta_0 \rangle$$

tangent line

Idea: minimise the approximation of L rather than L.



However, doing this will roll us down to $-\infty$. To avoid this, we penalise the approximation to avoid the never-ending descent, and minimise that instead: for $\alpha > 0$

$$\min_{\theta \in \mathbb{R}^n} L(\theta_0) + \langle \nabla L(\theta_0), \theta - \theta_0 \rangle + \underbrace{\frac{1}{2\alpha} \|\theta - \theta_0\|_2^2}$$

tries to keep me close to θ_0

Denoting the solution θ^* we write:

~~[illegible]~~

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} L(\theta_0) + \langle \nabla L(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2\alpha} \|\theta - \theta_0\|_2^2$$

We would like to repeat the same but this time taking our refrence point θ_0 as θ^*. We can therefore create an iterative algorithm to do this for us:

$$\theta_{k+1} = \arg\min_{\theta \in \mathbb{R}^n} \underbrace{L(\theta_k) + \langle \nabla L(\theta_k), \theta - \theta_k \rangle + \frac{1}{2\alpha} \|\theta - \theta_k\|_2^2}_{\tilde{L}_k(\theta)}$$

$\tilde{L}(\theta)$ is convex and smooth, we need the critical point,

i.e. $\nabla \tilde{L}_k(\theta) = 0 \implies \nabla L(\theta_k) + \frac{1}{\alpha}(\theta - \theta_k) = 0$

Therefore: $\boxed{\theta_{k+1} = \theta_k - \alpha \nabla L(\theta_k)}$ the gradient descent

$\rightarrow$ I start at θ_k travel in the direction of $-\nabla L(\theta_k)$ with step α $\leftarrow$

* α can depend on k, i.e. α updated with each step. We only consider the constant case for simplicity.

Demonstrating convergence for least-squares.

Find θ^* minimiser of $L(\theta) = \frac{1}{2}\|A\theta - y\|_2^2$

for given $A \in \mathbb{R}^{d \times n}$ and $y \in \mathbb{R}^d$

- Finding the gradient

$$\nabla L(\theta) = \frac{1}{2} A^T(A\theta - y) = \frac{1}{2}A^TA\theta - \frac{1}{2}A^Ty$$

note that the Hessian $H = A^TA$

L being convex $\Longleftrightarrow$ H positive semi-definite

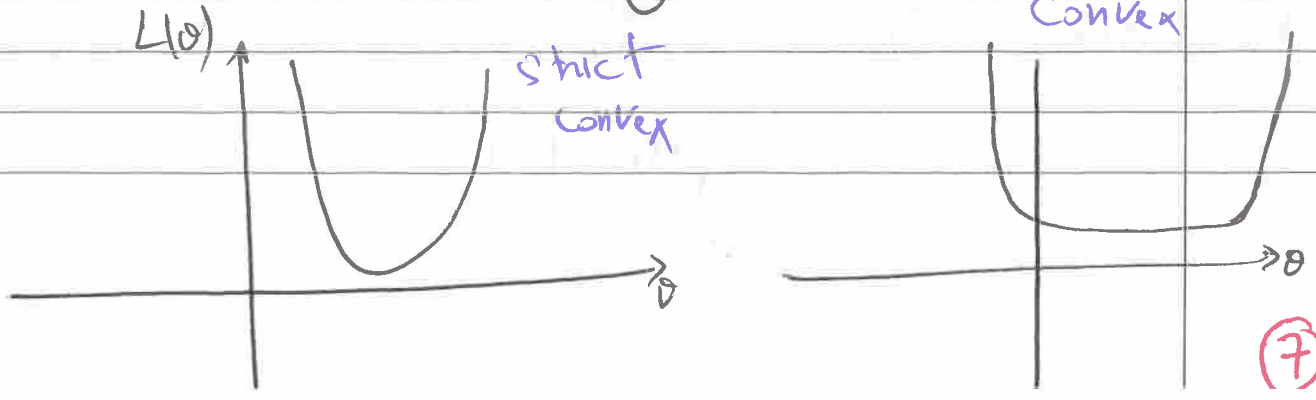
L strict convex $\Longleftrightarrow$ H positive definite.

then $\nabla L(\theta^*) = 0 \Longleftrightarrow H\theta^* = A^Ty$

* a solution exists since A^Ty is in the column space of H

* the solution is unique only when H is invertible (i.e. H pos. def meaning L strict convex)

* when minimiser not unique, all minimisers give the same minimum of L



From the GD algorithm

$$\theta_K = \theta_{K-1} - \alpha \nabla L(\theta_{K-1}) = \theta_{K-1} - \alpha H(\theta_{K-1} - \theta^*)$$

$$\Rightarrow \quad \theta_K - \theta^* = (I - \alpha H)(\theta_{K-1} - \theta^*)$$

$$\Rightarrow \quad \boxed{\theta_K - \theta^* = (I - \alpha H)^K (\theta_0 - \theta^*)} \quad - (*)$$

Using Taylor expansion around θ^* of L and the fact that $\nabla L(\theta^*) = 0$, we find

$$L(\theta) - L(\theta^*) = \frac{1}{2}(\theta - \theta^*)^T H (\theta - \theta^*)$$

Using $(*)$ gives

$$\boxed{L(\theta_K) - L(\theta^*) = \frac{1}{2}(\theta_0 - \theta^*)^T (I - \alpha H)^{2K} H (\theta_0 - \theta^*)} \quad - (**)$$

Tip: Use the fact that H and $(I - \alpha H)^K$ commute

$(*)$ and $(**)$ give two measures of convergence

1. $\|\theta_K - \theta^*\|_2^2 = (\theta_0 - \theta^*)^T (I - \alpha H)^{2K} (\theta_0 - \theta^*)$
2. $L(\theta_K) - L(\theta^*) = \frac{1}{2}(\theta_0 - \theta^*)^T (I - \alpha H)^{2K} H (\theta_0 - \theta^*)$.

To use 1 you need uniqueness and thus H has to be invertible. If not available, 2 can be used as a measure of optimality.

we can bound ① as

$$\|\theta_k - \theta^*\|_2^2 \leq \frac{1}{2} \underbrace{\lambda_{max}\left((I-\alpha H)^{2k}\right)}_{\text{Largest eigenvalue of } (I-\gamma H)^{2k}} \|\theta_0 - \theta^*\|_2^2$$

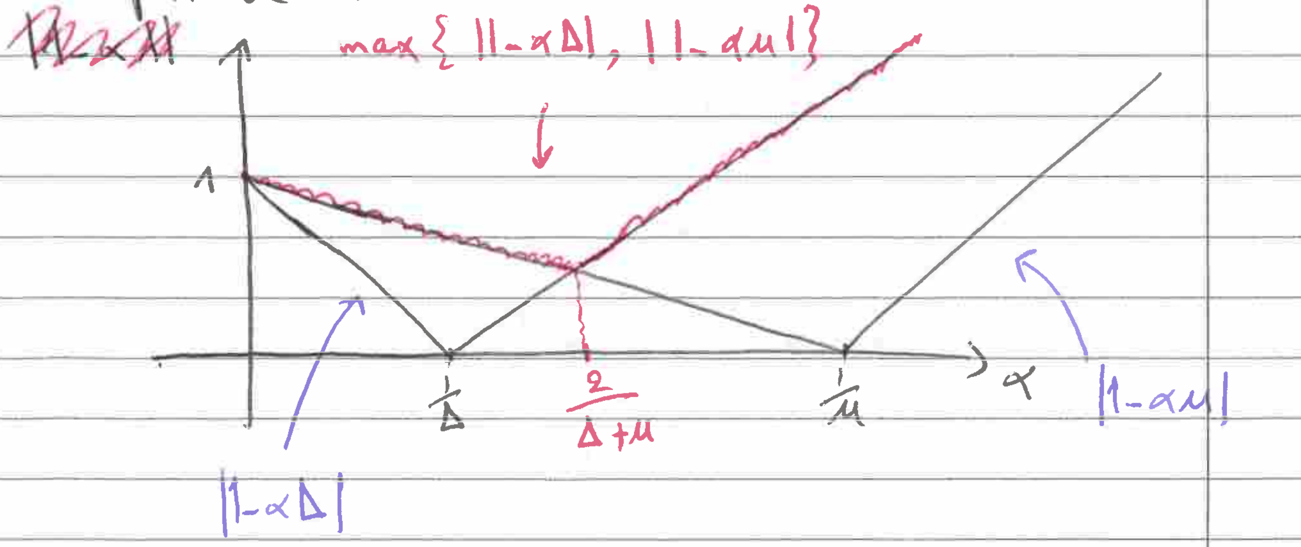
Tip: λ_{max} comes from the operator norm. This is true since $(I-\alpha H)^{2k}$ is symmetric.

The eigenvalues of $(I-\alpha H)^{2k}$ are $(1-\alpha\lambda)^{2k}$ for λ eigenvalues of H

Tip: this is true since I and αH commute.

Hence: $$\|\theta_k - \theta^*\|_2^2 \leq \frac{1}{2} \underbrace{\left(\max_\lambda |1-\alpha\lambda|\right)^{2k}}_{\text{maximise over } \lambda \in [\lambda_{min}, \lambda_{max}] =: [\mu, \Delta]} \|\theta_0 - \theta^*\|_2^2.$$

this is true for any α. So we choose an optimal α



Thus, we find

$$\|\theta_k - \theta^*\|_2^2 \le \left(1 - \frac{2}{\frac{L}{\mu}+1}\right)^{2k} \|\theta_0 - \theta^*\|_2^2 .$$

For $\mu = 0$ (i.e. H positive semi definite)

$\Rightarrow \frac{L}{\mu} = +\infty \Rightarrow \|\theta_k - \theta^*\|_2^2 \le \|\theta_0 - \theta^*\|_2^2$

which doesnot indicate convergence.

For such case we use $\textcircled{2}$ which can be written as

$$L(\theta_k) - L(\theta^*) \le \frac{1}{2} \max_{\lambda \in [\mu, L]} |(1-\alpha\lambda)^{2k}\lambda| \ \|\theta_0 - \theta^*\|_2^2 .$$

* note that,

$$|\lambda(1-\alpha\lambda)^{2k}| \le \lambda e^{-2k\alpha\lambda} \le \frac{\alpha 2k}{\alpha 2k} \lambda e^{-2k\alpha\lambda}$$

$$\le \frac{1}{2k\alpha} e^{-1} \le \frac{1}{4k\alpha}$$

Thus

$$F(\theta_k) - F(\theta^*) \le \overset{\frac{1}{8\alpha k}}{\cancel{\quad}} \|\theta_0 - \theta^*\|_2^2 .$$

More to explore:

- * general convex functions,
- * non convex functions,
- * Subgradients (for non smooth L),
- * Choosing optimal learning rate α.

Nesterov acceleration:

Momentum Gradient Descent, often called Accelerated Gradient Descent, was developed in the hope of achieving faster convergence. You see, the GD is blind, in the sense that it only sees in the current step θ_k. Based on this, the new idea is simple: the history of my movements can be a great indicator of where to move next! In other words, we will equip our GD with a memory. To implement this, we introduce "momentum" which will store the history of movements, i.e. momentum $:= \theta_k - \theta_{k-1}$

An accelerated GD iteration can then be given as:

$$\theta_{k+1} = -\alpha \nabla L(\theta_k) + \theta_k + \beta(\theta_k - \theta_{k-1}) \quad (\square)$$

$\beta \in [0,1]$ called the momentum parameter and we set $\theta_0 = \theta_1$

$(\square)$ simply says: first take a step in the direction $(\theta_k - \theta_{k-1})$ which is the momentum accumulated by the previous step, then preform the usual GD step.

In the literature, (□) is usually written with the help of an auxiliary parameter η_k such that:

η in (I) and (II) is different

$$\overset{(I)}{\begin{cases} \theta_{k+1} = \eta_k - \alpha \nabla L(\theta_k) \\ \eta_k = \theta_k + \beta(\theta_k - \theta_{k-1}) \end{cases}} \iff \overset{(II)}{\begin{cases} \theta_{k+1} = \theta_k + \eta_{k+1} \\ \eta_{k+1} = \beta \eta_k - \alpha \nabla L(\theta_{k+1}) \end{cases}}$$

The relatively slow convergence of GD can be geometrically seen by observing its "zig-zag" behaviour. The accelerated GD flattens these zig-zags.

To see how the history of my movements contribute to my current step, let's calculate the first few steps of η_k from (II)

$$\eta_1 = \beta \eta_0 - \alpha \nabla L(\theta_1)$$

$$\eta_2 = \beta \eta_1 - \alpha \nabla L(\theta_2) = \beta\left(\beta\eta_0 - \alpha \nabla L(\theta_1)\right) - \alpha \nabla L(\theta_2)$$

$$= \beta^2 \eta_0 - \beta\alpha \nabla L(\theta_1) - \alpha \nabla L(\theta_2)$$

$$\eta_3 = \beta^3 \eta_0 - \beta^2 \alpha \nabla L(\theta_1) - \beta\alpha \nabla L(\theta_2) - \alpha \nabla L(\theta_3)$$

$\vdots$

take initial $\eta_0 = 0$ since $\theta_0 = \theta_1$

$$\eta_k = -\sum_{i=0}^{\infty} \beta^i \alpha \nabla L(\theta_{k-i}) \leftarrow \text{the memory}$$

meaning that η_k stores all my previous gradients with a decreasing "importance" since $\beta \in [0,1]$

This formulation of accelerated GD has a downside. It tends to overshoot around the minima, i.e. it oscillates in and out of the minima region because of its momentum. Reducing these oscillations has been achieved by the "look-ahead" technique introduced by Russian mathematician Nesterov. The idea is simple: instead of calculating the gradient at the current step θ_K, let's calculate the gradient at the step θ_K pushed by the momentum $\beta(\theta_K - \theta_{K-1})$. The iterations then become:

$$\begin{cases} \eta_{K+1} = \theta_{K+1} + \beta(\theta_{K+1} - \theta_K) \\ \theta_{K+1} = \underbrace{\eta_K}_{\theta_K + \beta(\theta_K - \theta_{K-1})} - \alpha \underbrace{\nabla L(\eta_K)}_{\nabla L(\theta_K + \beta(\theta_K - \theta_{K-1}))} \quad \text{—} (\square\square) \end{cases}$$

$(\square\square)$ simply says that after you take a step in the direction of momentum, you calculate the gradient at that point you arrive at i.e. $(\theta_K + \beta(\theta_K - \theta_{K-1}))$

For appropriately chosen α and β, it can be shown that

$$L(\theta_k) - L(\theta^*) \leq \frac{C}{(k+1)^2} \|\theta_0 - \theta^*\|_2^2$$

which is indeed faster than GD.

Proximal Gradient Descent

Okay, now! what if the objective function L is not smooth. The method of subgradient (an extension of the gradient method) can be used. However, the gradient method has a rate of convergence $\frac{1}{K}$, while the subgradient converges at a rate $\frac{1}{\sqrt{K}}$.

- Proximal Gradient Descent aims to treat non smooth functions and converge with the same rate as the gradient descent.
- This, however, is achieved for objective functions that can be written as a composite of smooth and non smooth functions:

$$L(\theta) = \underbrace{f(\theta)}_{\text{smooth}} + \underbrace{g(\theta)}_{\text{non smooth}}$$

is this form common? yes! think about regularised least-squares using a "lasso" L^1 regularisation

$$L(\theta) = \|A\theta - y\|_2^2 + \lambda\|\theta\|_1$$

or think about constrained optimisation

$$\begin{cases} \min\limits_{\theta \in \mathbb{R}^n} \|A\theta - y\|_2^2 \\ \theta \in \mathcal{C} \end{cases} \Longleftrightarrow \min_{\theta \in \mathbb{R}^n} \|A\theta - y\|_2^2 + I_{\mathcal{C}}(\theta)$$

indicator function $= \begin{cases} +\infty & \theta \notin \mathcal{C} \\ 0 & \theta \in \mathcal{C} \end{cases}$

The basic idea: apply gradient descent to the smooth part. Handle the nonsmooth part separately.

- Set $\theta^* = \arg\min_{\theta \in \mathbb{R}^n} L(\theta) = \arg\min_{\theta^* \in \mathbb{R}^n} f(\theta) + g(\theta)$
- For some $\tilde{\theta}$, recall that we can motivate the gradient descent by writing:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} f(\tilde{\theta}) + \nabla f(\tilde{\theta})^T(\theta - \tilde{\theta}) + \frac{1}{2\alpha}\|\theta - \tilde{\theta}\|_2^2 + g(\theta)$$

- which gives

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} \underbrace{\frac{1}{2\alpha}\|\theta - (\tilde{\theta} - \alpha\nabla f(\tilde{\theta}))\|_2^2}_{\text{Keep me close to the grad update of } f} + \underbrace{g(\theta)}_{\text{make } g \text{ small}}$$

- The "Prox" operation is defined by:

$$\boxed{\mathrm{Prox}_{g,\alpha}(\bar{\theta}) = \arg\min_{\theta \in \mathbb{R}^n} \frac{1}{2\alpha}\|\theta - \bar{\theta}\|_2^2 + g(\theta)}$$

well-defined due to convexity.

- The algorithm then reads:

$$\begin{cases} \text{choose } \theta_0 \\ \theta_{k+1} = \mathrm{Prox}_{g,\alpha}(\theta_k - \alpha\nabla f(\theta_k)) \end{cases}$$

in a more familiar form:

$$\begin{cases} \theta_{k+1} = \theta_k - \alpha\,\Psi_\alpha(\theta_k) \\ \Psi_\alpha(\bar{\theta}) = \dfrac{\bar{\theta} - \mathrm{Prox}_{g,\alpha}(\bar{\theta} - \alpha\nabla f(\bar{\theta}))}{\alpha} \end{cases}$$

• "make g small": (I think) this is handled on a case-by-case basis. People have managed to derive a closed-form for many common forms of g.

Example for a 1D lasso regularisation:

$$L(\theta) = \frac{1}{2}(\theta - y)^2 + |\theta| \quad \text{for a given } y \in \mathbb{R}.$$

then $$\text{Prox}_{g\alpha}(\bar\theta) = \underset{\theta\in\mathbb{R}}{\arg\min}\ \frac{1}{2\alpha}(\theta-\bar\theta)^2 + |\theta|$$

* For $\theta > 0$ then

$$\text{Prox}_{g\alpha}(\bar\theta) = \underset{\theta>0}{\arg\min}\ \frac{1}{2\alpha}(\theta-\bar\theta)^2 + \theta$$

taking the derivative gives and setting to zero:

$$\theta = -\alpha + \bar\theta$$

meaning that $\bar\theta > \alpha$ since $\theta > 0$

* Continue with $\theta < 0$ and $\theta = 0$ to get

$$\text{Prox}_{g\alpha}(\bar\theta) = \begin{cases} \bar\theta - \alpha & \text{if } \bar\theta > \alpha \\ 0 & \text{if } -\alpha \le \bar\theta \le \alpha \\ \bar\theta + \alpha & \text{if } \bar\theta < -\alpha \end{cases}$$

• For the Convergence, it is shown that

$$L(\theta_k) - L(\theta^*) \leq \frac{1}{2k\alpha} \|\theta_0 - \theta^*\|_2^2$$

i.e. rate of convergence $\frac{1}{K}$, similar to gradient descent.

• One last remark, it is useful to note that speaking of "iterations" the prox operator is independent of the nonsmooth g. You do not need to update g at any iteration, only ∇f. Unlike the subgradient method where you will have to deal with updates of the (subgradient) of the nonsmooth g.

More to explore,

* non convex L
* accelerated proximal gradient
* closed-form of "prox" for other forms of g.

Stochastic Gradient Descent (SGD):

The gradient of the objective function might be expensive to compute. Indeed, for example, for an empirical risk $L(\theta) = \sum_{i=1}^{N} \ell(y_i, f_\theta^*(x_i))$ calculating ∇L requires calculating $\nabla \ell$ for all data points $\{(x_i, y_i)\}_{i=1}^{N}$. This can get easily expensive (time and memory wise) for tipical large values of N. In other words, the gradient descent method requires access to the entire data set.

Basic idea of SGD: we instead use an approximate $\tilde{\nabla} L$ of the gradient which does not require the whole data set to be updated. However, we require

$$\mathbb{E}[\tilde{\nabla} L(\theta)] = \nabla L(\theta) \quad - (*)$$

Then use this approximate "noisy" gradient to perform the gradient descent iterations.

$$\theta_{k+1} = \theta_k - \alpha \tilde{\nabla} L(\theta_k)$$

Since we have moved from minimising the expected risk $L(\theta) = \mathbb{E}[\ell(y, f_\theta^*(x))]$ to minimising the empirical risk $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f_\theta^*(x_i))$, we will make use of the "summation" structure to define a suitable noisy gradient $\tilde{\nabla} L(\theta)$.

We define $\widetilde{\nabla L(\theta)}$ as follows: choose a random index from the set of all indices of our data set

$$\rightsquigarrow \textcircled{1}\; I \sim \text{uniform}(1, 2, \dots, N)$$

then take $\tilde{\nabla} L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(y_I, f_\theta^*(x_I)) = \nabla \ell(y_I, f_\theta^*(x_I))$

i.e. the gradient of the loss is computed only "at the point" (x_I, y_I) chosen randomly.

$\textcircled{1}$ means that each index in $\{1, \dots, N\}$ has an equal chance of being picked, i.e.

$$P(I = i) = \frac{1}{N}$$

We can check the condition $\textcircled{*}$

$$\mathbb{E}[\tilde{\nabla} L(\theta)] = \mathbb{E}[\nabla \ell(y_I, f_\theta^*(x_I))] = \sum_{i=1}^{N} \nabla \ell(y_i, f_\theta^*(x_i)) \times \frac{1}{N}$$

Sum of all possible outcomes times the probability times the uniform probability ↗

$$= \nabla L(\theta)$$

This then is a good candidate for the noisy gradient and it is what we can now use in a gradient descent iteration.

Although noisy, the iterations of SGD do converge eventually.

* Given that we have a good candidate for $\tilde{\nabla} L$ and that $\mathbb{E}\left[\|\tilde{\nabla} L(\theta)\|_2^2\right] \leq C^2$ ← bounding the variance of the stochastic gradient from above

and assuming that $\mathbb{E}\left[\|\theta_1 - \theta^*\|_2^2\right] \leq G^2$ then

$$\boxed{\mathbb{E}\left[L(\bar{\theta}_M)\right] - L(\theta^*) \leq \frac{CG}{\sqrt{M}}}$$

where $\bar{\theta}_M = \frac{1}{M}\sum_{i=1}^{M} \theta_i$ and M is the total number of iterations, and θ_1 is the initial value.

* the estimate holds in "expectation $\mathbb{E}$" since the algorithm uses randomness. So the convergence is measured on average via "expectation"

Mini-Batching: While we know GD can be expensive to perform, on the other end, SGD introduces high variance due to the noisy gradient. To find balance between convergence speed and computational cost, we use a "mini-batch" of data points instead one one single data point to calculate a less noisy gradient (reduces variance). This is still cheaper than accounting for the full gradient.

In other words, we change our candidate for $\tilde{\nabla} L$ to

$$\tilde{\nabla} L(\theta) = \frac{1}{B} \sum_{j=1}^{B} \nabla \ell \left(y_{I_j}, f^*_\theta(x_{I_j}) \right)$$

such that $B =$ size of $\{I_1, I_2, \dots, I_B\}$ and

$$I_j \sim \text{uniform}\{1, 2, \dots, N\} \qquad \forall\, 1 \leq j \leq B$$

is this a good candidate? Yes! since $\mathbb{E}\left[\frac{1}{B} \sum_{j=1}^{B} \nabla \ell \left(y_{I_j}, f^*_\theta(x_{I_j}) \right) \right]$

$$= \frac{1}{B} \sum_{j=1}^{B} \mathbb{E}\left[\nabla \ell \left(y_{I_j}, f^*_\theta(x_{I_j}) \right) \right] = \text{[illegible]} = \nabla L(\theta)$$

Then we use our new noisy gradient in the usual gradient descent iterations: [illegible]

$$\theta_{k+1} = \theta_k - \alpha \tilde{\nabla} L(\theta_k)$$

It should be obvious by now that:

- $B = N \implies$ mini-Batch GD $\equiv$ GD
- $B = 1 \implies$ min-Batch ~~SGD~~ $\equiv$ SGD

In terms of convergence: the previous result for SGD holds.

More to explore:

* accelerated Stochastic Gradient Descent,

* Non-Gradient Based Optimisation.

References:

* Francis Bach, Learning Theory From First Principles 2024

* Every YouTube video there has ever been about optimisation up to 20 May 2025.