CHAPTER

PARAMETER LEARNING: A CONVEX ANALYTIC PATH 8

CHAPTER OUTLINE

8.1 INTRODUCTION

The theory of convex sets and functions has a rich history and has been the focus of intense study for over a century in mathematics. In the terrain of applied sciences and engineering, the revival of interest on convex functions and optimization is traced back in the early 1980s. In addition to the increased processing power that became available via the use of computers, certain theoretical developments were catalytic in demonstrating the power of such techniques. The advent of the so-called interior point methods opened a new path in solving the classical linear programming task. Moreover, it was increasingly realized that, despite its advantages, the least-squares cost function also has a number of drawbacks, particularly in the presence of non-Gaussian noise and the presence of outliers. It has been demonstrated that the use of alternative cost functions, which may not even be differentiable, can alleviate a number of problems that are associated with the least-squares methods. Furthermore, the increased interest in robust learning methods brought into the scene the need for nontrivial constraints, which the optimized solution has to respect. In the machine learning community, the discovery of support vector machines, to be treated in Chapter 11, played an important role in popularizing convex optimization techniques.

The goal of this chapter is to present some basic notions and definitions related to convex analysis and optimization in the context of machine learning and signal processing. Convex optimization is a discipline in itself, and it cannot be summarized in a chapter. Our emphasis here is on computationally light techniques with a focus on online versions, which are gaining in importance in the context of big data applications. A related discussion is also part of this chapter.

The material revolves around two families of algorithms. One goes back to the classical work of Von Neumann on projections on convex sets, which is reviewed together with its more recent online versions. The notions of projection and related properties are treated in some detail. The method of projections, in the context of constrained optimization, is gaining in popularity recently.

The other family of algorithms, that is considered, builds around the notion of subgradient for optimizing nondifferentiable convex functions and generalizations of the gradient descent family, discussed in Chapter 5. Further, we introduce a powerful tool for analyzing the performance of online algorithms for convex optimization, known as regret analysis, and present a case study. We touch on a current trend in convex optimization, involving proximal and mirror descent methods.

8.2 CONVEX SETS AND FUNCTIONS

Although most of the algorithms we will discuss in this chapter refer to vector variables in Euclidean spaces, which is in line with what we have done so far in this book, the definitions and some of the fundamental theorems will be stated in the context of the more general case of Hilbert spaces.[1] This is because the current chapter will also serve the needs of subsequent chapters, whose setting is that of infinite dimensional Hilbert spaces. For those readers who are not interested in such spaces, all they need to know is that a Hilbert space is a generalization of the Euclidean one, allowing for infinite dimensions. To serve the needs of these readers, we will be careful in pointing out the differences between Euclidean and the more general Hilbert spaces in the theorems, whenever this is required.

8.2.1 CONVEX SETS

Definition 8.1. A nonempty subset C of a Hilbert space $\mathbb{H}$, $C \subseteq \mathbb{H}$, is called *convex*, if $\forall\, \boldsymbol{x}_1, \boldsymbol{x}_2 \in C$ and $\forall \lambda \in [0, 1]$, the following holds true[2]

$$\boldsymbol{x} := \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in C. \tag{8.1}$$

Note that if $\lambda = 1$, $\boldsymbol{x} = \boldsymbol{x}_1$, and if $\lambda = 0$, $\boldsymbol{x} = \boldsymbol{x}_2$. For any other value of λ in $[0, 1]$, $\boldsymbol{x}$ lies in the line segment joining $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Indeed, from (8.1) we can write

$$\boldsymbol{x} - \boldsymbol{x}_2 = \lambda(\boldsymbol{x}_1 - \boldsymbol{x}_2), \quad 0 \le \lambda \le 1.$$

Figure 8.1 shows two examples of convex sets, in the two-dimensional Euclidean space, $\mathbb{R}^2$. In Figure 8.1a, the set comprises all points whose Euclidean (ℓ_2) norm is less than or equal to one,

$$C_2 = \left\{ \boldsymbol{x} : \sqrt{x_1^2 + x_2^2} \le 1 \right\}.$$

Sometimes we refer to C_2 as the ℓ_2-*ball* of radius equal to one. Note that the set includes all the points *on and inside the circle*. The set in Figure 8.1b comprises all the points *on and inside the rhombus* defined by,

$$C_1 = \left\{ \boldsymbol{x} : |x_1| + |x_2| \le 1 \right\}.$$

Because the sum of the absolute values of the components of a vector defines the ℓ_1-norm, that is, $\|\boldsymbol{x}\|_1 := |x_1| + |x_2|$, in analogy to C_2 we call the set C_1 as the ℓ_1-ball of radius equal to one. In contrast, the sets whose ℓ_2 and ℓ_1 norms are equal to one, or in other words,

$$\bar{C}_2 = \left\{ \boldsymbol{x} : x_1^2 + x_2^2 = 1 \right\}, \bar{C}_1 = \left\{ \boldsymbol{x} : |x_1| + |x_2| = 1 \right\},$$

are not convex (Problem 8.2). Figure 8.2 shows two examples of nonconvex sets.

[1] The mathematical definition of a Hilbert space is provided in Section 8.15.
[2] In conformity with Euclidean vector spaces and for the sake of notational simplicity, we will keep the same notation and denote the elements of a Hilbert space with lowercase bold letters.

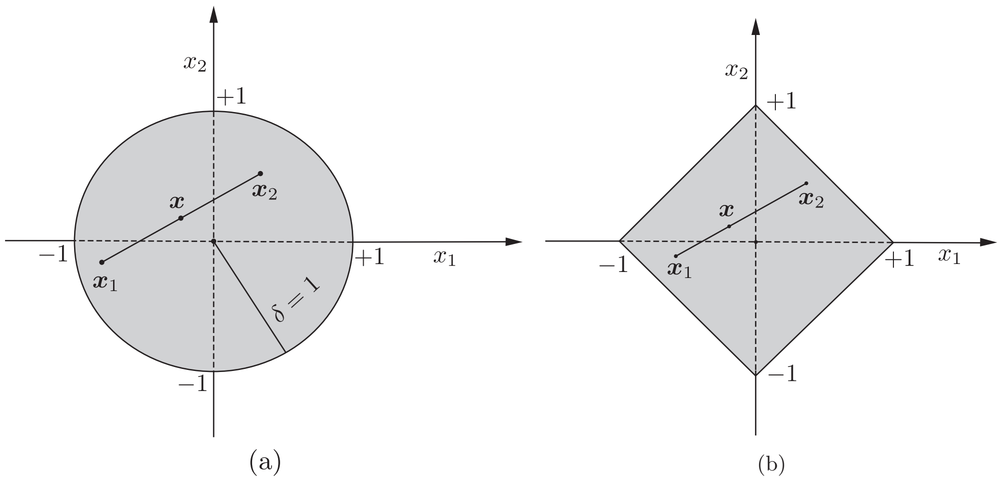


FIGURE 8.1

(a) The ℓ_2-ball of radius $\delta = 1$ comprises all points with Euclidean norm less than or equal to $\delta = 1$. (b) The ℓ_1-ball consists of all the points with ℓ_1 norm less than or equal to $\delta = 1$. Both are convex sets.

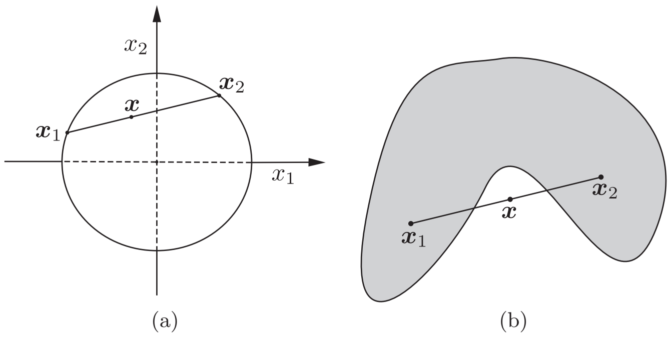


FIGURE 8.2

Examples of two nonconvex sets. In both cases, the point $\boldsymbol{x}$ does not lie on the same set in which $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ belong. In (a) the set comprises all the points whose Euclidean norm is equal to one.

8.2.2 CONVEX FUNCTIONS

Definition 8.2. A function

$$f:\ \mathcal{X} \subseteq \mathbb{R}^l \longmapsto \mathbb{R}$$

is called *convex* if $\mathcal{X}$ is convex and if $\forall\, \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$ the following holds true:

$$\boxed{f\big(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2\big) \leq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2),\ \lambda \in [0,1].} \tag{8.2}$$

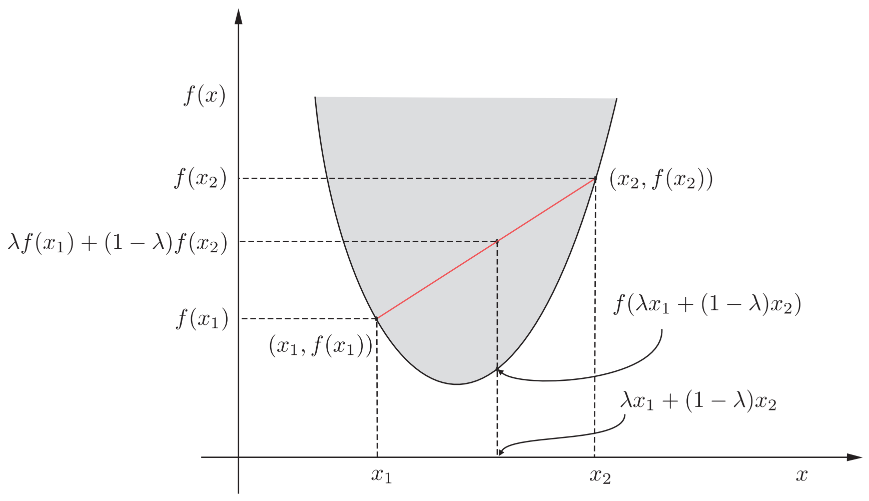

$f(x)$
$f(x_2)$
$\lambda f(x_1)+(1-\lambda)f(x_2)$
$f(x_1)$
$(x_2, f(x_2))$
$(x_1, f(x_1))$
$f(\lambda x_1+(1-\lambda)x_2)$
$\lambda x_1+(1-\lambda)x_2$
x_1
x_2
x

FIGURE 8.3

The line segment joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ lies above the graph of $f(x)$. The shaded region corresponds to the epigraph of the function.

The function is called *strictly convex* if (8.2) holds true with strict inequality when $\lambda \in (0,1)$, $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$. The geometric interpretation of (8.2) is that the line segment joining the points $(\boldsymbol{x}_1, f(\boldsymbol{x}_1))$ and $(\boldsymbol{x}_2, f(\boldsymbol{x}_2))$ lies above the graph of $f(\boldsymbol{x})$, as shown in Figure 8.3. We say that a function is *concave* (*strictly concave*) if the negative, $-f$, is convex (strictly convex). Next, we state three important theorems.

Theorem 8.1 (First order convexity condition)**.** *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set and*

$$f : \mathcal{X} \longmapsto \mathbb{R},$$

be a differentiable function. Then, $f(\cdot)$ is convex if and only if $\forall\, \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$,

$$\boxed{f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla^{\mathrm{T}} f(\boldsymbol{x})(\boldsymbol{y}-\boldsymbol{x}).} \tag{8.3}$$

The proof of the theorem is given in Problem 8.3. The theorem generalizes to nondifferentiable convex functions; it will be discussed in this context in Section 8.10.

Figure 8.4 illustrates the geometric interpretation of this theorem. It means that the graph of the convex function is located above the graph of the affine function

$$g : \boldsymbol{y} \longmapsto \nabla^T f(\boldsymbol{x})(\boldsymbol{y}-\boldsymbol{x}) + f(\boldsymbol{x}),$$

which defines the tangent hyperplane of the graph at the point $(\boldsymbol{x}, f(\boldsymbol{x}))$.

Theorem 8.2 (Second order convexity condition)**.** *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set. Then a twice differentiable function, $f : \mathcal{X} \longmapsto \mathbb{R}$, is convex (strictly convex) if and only if the Hessian matrix is positive semidefinite (positive definite).*

The proof of the theorem is given in Problem 8.5. Recall that in previous chapters, when we dealt with the squared error loss function, we commented that it is a convex one. Now we are ready to justify this argument. Consider the quadratic function,

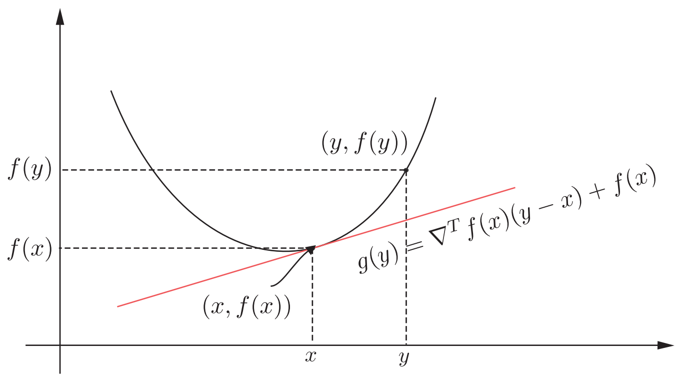


FIGURE 8.4

The graph of a convex function is above the tangent plane at any point of the respective graph.

$$f(\boldsymbol{x}) := \frac{1}{2}\boldsymbol{x}^T Q\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c,$$

where Q is a positive definite matrix. Taking the gradient, we have

$$\nabla f(\boldsymbol{x}) = Q\boldsymbol{x} + \boldsymbol{b},$$

and the Hessian matrix is equal to Q, which by assumption is positive definite, hence f is a (strictly) convex function.

In the sequel, two very important notions in convex analysis and optimization are defined.

Definition 8.3. The *epigraph* of a function, f, is defined as the set of points

$$\boxed{\text{epi}(f) := \left\{(\boldsymbol{x}, r) \in \mathcal{X} \times \mathbb{R} : f(\boldsymbol{x}) \leq r\right\} : \quad \text{Epigraph.}} \tag{8.4}$$

From a geometric point of view, the epigraph is the set of all points in $\mathbb{R}^l \times \mathbb{R}$ that lie on and above the graph of $f(\boldsymbol{x})$, as it is indicated by the gray shaded region in Figure 8.3. It is important to note that a function is convex *if and only if* its epigraph is a convex set (Problem 8.6).

Definition 8.4. Given a real number ξ, the *lower level set* of function, $f : \mathcal{X} \subseteq \mathbb{R}^l \longmapsto \mathbb{R}$, at *height* ξ, is defined as

$$\boxed{\text{lev}_{\leq \xi}(f) := \left\{\boldsymbol{x} \in \mathcal{X} : f(\boldsymbol{x}) \leq \xi\right\} : \text{Level Set at } \xi.} \tag{8.5}$$

In words, it is the set of all points at which the function takes a value less than or equal to ξ. The geometric interpretation of the level set is shown in Figure 8.5. It can easily be shown (Problem 8.7) that if a function, f, is convex then its lower level set is convex for any $\xi \in \mathbb{R}$. The converse is not true. We can easily check out that the function $f(x) = -\exp(x)$ is not convex (as a matter of fact it is concave) and all its lower level sets are convex.

Theorem 8.3 (Local and global minimizers)**.** *Let a convex function, $f : \mathcal{X} \longmapsto \mathbb{R}$. Then, if a point $\boldsymbol{x}_*$ is a local minimizer, it is also a global one and the set of all minimizers is convex. Further, if the function is strictly convex, the minimizer is unique.*

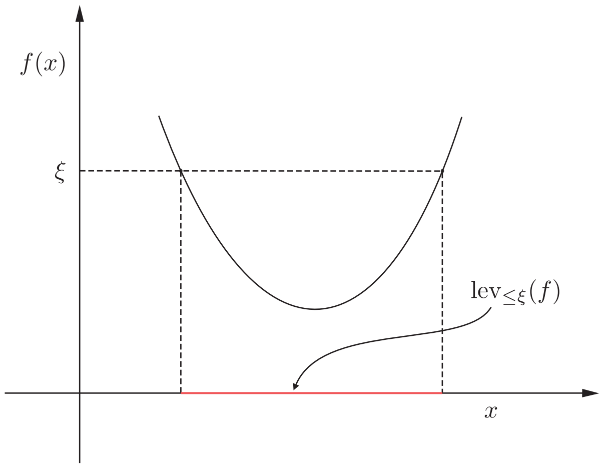


FIGURE 8.5

The level set at height ξ comprises all the points in the interval denoted as the "red" segment on the x-axis.

Proof. Inasmuch as the function is convex we know that, $\forall\, \boldsymbol{x} \in \mathcal{X}$,

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_*) + \nabla^{\mathrm{T}} f(\boldsymbol{x}_*)(\boldsymbol{x} - \boldsymbol{x}_*),$$

and because at the minimizer the gradient is zero, we get

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_*), \tag{8.6}$$

which proves the claim. Let us now denote as

$$f_* = \min_{\boldsymbol{x}} f(\boldsymbol{x}). \tag{8.7}$$

Note that the set of all minimizers coincides with the level set at height f_*. Then because the function is convex, we know that the level set $\mathrm{lev}_{f_*}(f)$ is convex, which verifies the convexity of the set of minimizers. Finally, for strictly convex functions, the inequality in (8.6) is a strict one, which proves the uniqueness of the (global) minimizer. The theorem is also true, even if the function is not differentiable (Problem 8.10). □

8.3 PROJECTIONS ONTO CONVEX SETS

The projection onto a hyperplane in finite dimensional Euclidean spaces was discussed and used in the context of the affine projection algorithm (APA) algorithm in Chapter 5. The notion of projection will now be generalized to include any closed convex set and also in the framework of general (infinite dimensional) Hilbert spaces.

The concept of projection is among the most fundamental concepts in mathematics, and everyone who has attended classes in basic geometry has used and studied it. What one may not have realized is that while performing a projection, for example, drawing a line segment from a point to a line or a plane, basically he or she solves an *optimization* task. The point $\boldsymbol{x}_*$ in Figure 8.6, that is, the projection of $\boldsymbol{x}$ onto the plane, H, in the three-dimensional space, is that point, among all the points lying on the plane, whose (Euclidean) distance from $\boldsymbol{x} = [x_1, x_2, x_3]^{\mathrm{T}}$ is minimum; in other words,

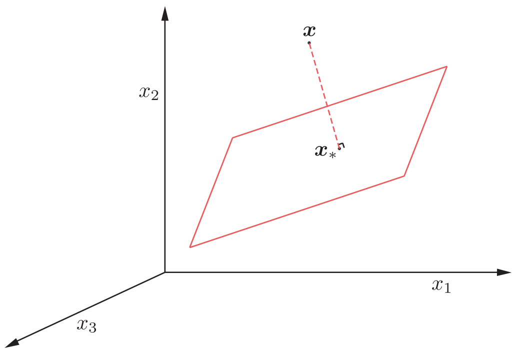


FIGURE 8.6

The projection $\boldsymbol{x}_*$ of $\boldsymbol{x}$ onto the plane, minimizes the distance of $\boldsymbol{x}$ from all the points lying on the plane.

$$\boldsymbol{x}_* = \min_{\boldsymbol{y} \in H} \left((x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 \right). \tag{8.8}$$

As a matter of fact, what we have learned to do in our early days at school is to solve a *constrained optimization* task. Indeed, (8.8) can equivalently be written as,

$$\boldsymbol{x}_* = \arg\min_{\boldsymbol{y}} ||\boldsymbol{x} - \boldsymbol{y}||^2,$$
$$\text{s.t.} \quad \boldsymbol{\theta}^T \boldsymbol{y} + \theta_0 = 0,$$

where the constraint is the equation describing the specific plane. Our goal herein focuses on generalizing the notion of projection, to employ it to attack more general and complex tasks.

Theorem 8.4. *Let C be a nonempty closed*[3] *convex set in a Hilbert space $\mathbb{H}$ and $\boldsymbol{x} \in \mathbb{H}$. Then, there exists a unique point, denoted as $P_C(\boldsymbol{x}) \in C$, such as*

$$\boxed{\|\boldsymbol{x} - P_C(\boldsymbol{x})\| = \min_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\| : \quad \text{Projection of } \boldsymbol{x} \text{ on } C.}$$

$P_C(\boldsymbol{x})$ is called the (metric) projection of $\boldsymbol{x}$ onto C. Note that if $\boldsymbol{x} \in C$, then $P_C(\boldsymbol{x}) = \boldsymbol{x}$, since this makes the norm $\|\boldsymbol{x} - P_C(\boldsymbol{x})\| = 0$.

Proof. The proof comprises two paths. One is to establish *uniqueness* and the other to establish *existence*. Uniqueness is easier, and the proof will be given here. Existence is slightly more technical, and it is provided in Problem 8.11.

To show *uniqueness*, assume that there are two points, namely $\boldsymbol{x}_{*,1}$ and $\boldsymbol{x}_{*,2}$, $\boldsymbol{x}_{*,1} \neq \boldsymbol{x}_{*,2}$, such as:

$$\|\boldsymbol{x} - \boldsymbol{x}_{*,1}\| = \|\boldsymbol{x} - \boldsymbol{x}_{*,2}\| = \min_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{8.9}$$

(a) If $\boldsymbol{x} \in C$, then $P_C(\boldsymbol{x}) = \boldsymbol{x}$ is unique, since any other point in C would make $\|\boldsymbol{x} - P_C(\boldsymbol{x})\| > 0$.

[3] For the needs of this chapter, it suffices to say that a set C is closed if the limit point of any sequence of points in C, lies in C.

(b) Let $\boldsymbol{x} \notin C$. Then, mobilizing the parallelogram law of the norm (Appendix 8.15, Eq. (8.149), Problem 8.8) we get

$$\|(\boldsymbol{x}-\boldsymbol{x}_{*,1})+(\boldsymbol{x}-\boldsymbol{x}_{*,2})\|^2+\|(\boldsymbol{x}-\boldsymbol{x}_{*,1})-(\boldsymbol{x}-\boldsymbol{x}_{*,2})\|^2=2\left(\|\boldsymbol{x}-\boldsymbol{x}_{*,1}\|^2+\|\boldsymbol{x}-\boldsymbol{x}_{*,2}\|^2\right),$$

or

$$\|2\boldsymbol{x}-(\boldsymbol{x}_{*,1}+\boldsymbol{x}_{*,2})\|^2+\|\boldsymbol{x}_{*,1}-\boldsymbol{x}_{*,2}\|^2=2\left(\|\boldsymbol{x}-\boldsymbol{x}_{*,1}\|^2+\|\boldsymbol{x}-\boldsymbol{x}_{*,2}\|^2\right),$$

and exploiting (8.9) and the fact that $\|\boldsymbol{x}_{*,1}-\boldsymbol{x}_{*,2}\|>0$, we have

$$\left\|\boldsymbol{x}-\left(\frac{1}{2}\boldsymbol{x}_{*,1}+\frac{1}{2}\boldsymbol{x}_{*,2}\right)\right\|^2<\|\boldsymbol{x}-\boldsymbol{x}_{*,1}\|^2. \tag{8.10}$$

However, due to the convexity of C, the point $\frac{1}{2}\boldsymbol{x}_{*,1}+\frac{1}{2}\boldsymbol{x}_{*,2}$ lies in C. Also, by the definition of projection, $\boldsymbol{x}_{*,1}$ is the point with the smallest distance, hence (8.10) cannot be valid.

□

For the existence, one has to use the property of closeness (every sequence in C has its limit in C) as well as the property of completeness of Hilbert spaces, which guarantees that every Cauchy sequence in $\mathbb{H}$ has a limit (Appendix 8.15). The proof is given in Problem 8.11.

Remarks 8.1.

- Note that if $\boldsymbol{x} \notin C \subseteq \mathbb{H}$, then its projection onto C lies on the *boundary* of C (Problem 8.12).

Example 8.1. Derive analytical expressions for the projections of a point $\boldsymbol{x} \in \mathbb{H}$, where $\mathbb{H}$ is a real Hilbert space, onto (a) a hyperplane, (b) a halfspace, and (c) the ℓ_2-ball of radius δ.

(a) A hyperplane, H, is defined as

$$H := \left\{\boldsymbol{y} : \langle\boldsymbol{\theta},\boldsymbol{y}\rangle+\theta_0=0\right\},$$

for some $\boldsymbol{\theta} \in \mathbb{H}$ and $\theta_0 \in \mathbb{R}$. If $\mathbb{H}$ breaks down to a Euclidean space, the projection is readily available by simple geometric arguments, and it is given by

$$\boxed{P_C(\boldsymbol{x})=\boldsymbol{x}-\frac{\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0}{\|\boldsymbol{\theta}\|^2}\boldsymbol{\theta}: \quad \text{Projection onto a Hyperplane,}} \tag{8.11}$$

and it is shown in Figure 8.7. For a general Hilbert space $\mathbb{H}$, the hyperplane is a closed convex subset of $\mathbb{H}$, and the projection is still given by the same formula (Problem 8.13).

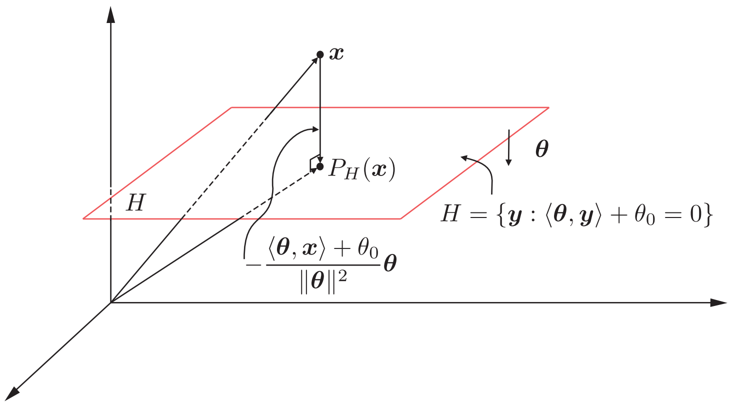
(b) The definition of a halfspace, H^+, is given by

$$H^+=\left\{\boldsymbol{y} : \langle\boldsymbol{\theta},\boldsymbol{y}\rangle+\theta_0\geq 0\right\}, \tag{8.12}$$

and it is shown in Figure 8.8, for the $\mathbb{R}^3$ case.

Because the projection lies on the boundary, if $\boldsymbol{x} \notin H^+$ its projection will lie onto the hyperplane defined by $\boldsymbol{\theta}$ and θ_0, and it will be equal to $\boldsymbol{x}$ if $\boldsymbol{x} \in H^+$; thus, the projection is easily checked out to be

$$\boxed{P_{H^+}(\boldsymbol{x})=\boldsymbol{x}-\frac{\min\{0,\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0\}}{\|\boldsymbol{\theta}\|^2}\boldsymbol{\theta}: \quad \text{Projection onto a Halfspace.}} \tag{8.13}$$



x
$P_H(x)$
θ
H
$H = \{y : \langle \theta, y \rangle + \theta_0 = 0\}$
$-\frac{\langle \theta, x \rangle + \theta_0}{\|\theta\|^2}\theta$

FIGURE 8.7

The projection onto a hyperplane in $\mathbb{R}^3$.

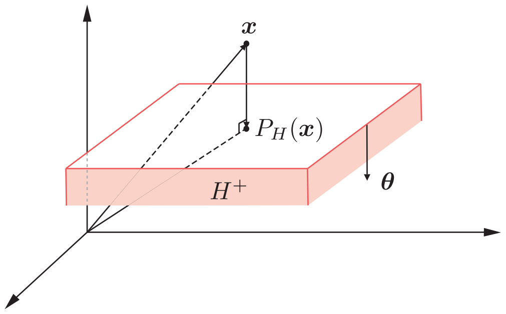


FIGURE 8.8

Projection onto a halfspace.

(c) The closed ball centered at $\mathbf{0}$ and of radius δ, denoted as $B[\mathbf{0}, \delta]$, in a general Hilbert space, $\mathbb{H}$, is defined as

$$B[\mathbf{0}, \delta] = \{ y : \|y\| \leq \delta \}.$$

The projection of $x \notin B[\mathbf{0}, \delta]$ onto $B[\mathbf{0}, \delta]$ is given by

$$\boxed{P_{B[\mathbf{0},\delta]}(x) = \begin{cases} x, & \text{if } \|x\| \leq \delta, \\ \delta \frac{x}{\|x\|}, & \text{if } \|x\| > \delta, \end{cases} \quad : \quad \text{Projection onto a Closed Ball,}} \tag{8.14}$$

and it is geometrically illustrated in Figure 8.9, for the case of $\mathbb{R}^2$ (Problem 8.14).

Remarks 8.2.

- In the context of sparsity-aware learning, which is dealt with in Chapter 10, a key point is the projection of a point in $\mathbb{R}^l$ ($\mathbb{C}^l$) onto the ℓ_1-ball. There it is shown that, given the size of the ball, this projection corresponds to the so-called soft thresholding operation (see Example 8.10 for a definition).

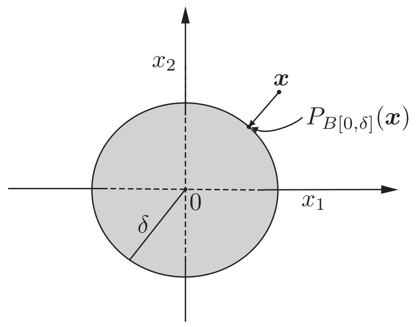


FIGURE 8.9

The projection onto a closed ball of radius δ centered at $\mathbf{0}$.

It should be stressed that a linear space equipped with the ℓ_1-norm is no more Euclidean (Hilbert), inasmuch as this norm *is not induced* by an inner product operation; moreover, uniqueness of the projection with respect to this norm is not guaranteed (Problem 8.15).

8.3.1 PROPERTIES OF PROJECTIONS

In this section, we summarize some basic properties of the projections. These properties are used to prove a number of theorems and convergence results, associated with algorithms that are developed around the notion of projection.

Readers who are interested only in the algorithms can bypass this section.

Proposition 8.1. *Let $\mathbb{H}$ be a Hilbert space, $C \subseteq \mathbb{H}$ be a closed convex set, and $\boldsymbol{x} \in \mathbb{H}$. Then the projection $P_C(\boldsymbol{x})$ satisfies the following two properties*[4]*:*

$$\boxed{\text{Real}\left\{\langle \boldsymbol{x} - P_C(\boldsymbol{x}), \boldsymbol{y} - P_C(\boldsymbol{x})\rangle\right\} \leq 0, \ \forall\, \boldsymbol{y} \in C,} \tag{8.15}$$

and

$$\boxed{\|P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\|^2 \leq \text{Real}\left\{\langle \boldsymbol{x} - \boldsymbol{y}, P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\rangle\right\}, \ \forall\, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{H}.} \tag{8.16}$$

The proof of the proposition is provided in Problem 8.16. The geometric interpretation of (8.15) for the case of real Hilbert space is shown in Figure 8.10. Note that for a real Hilbert space, the first property becomes,

$$\langle \boldsymbol{x} - P_C(\boldsymbol{x}), \boldsymbol{y} - P_C(\boldsymbol{x})\rangle \leq 0, \quad \forall \boldsymbol{y} \in C. \tag{8.17}$$

From the geometric point of view, (8.17) means that the angle formed by the two vectors, $\boldsymbol{x} - P_C(\boldsymbol{x})$ and $\boldsymbol{y} - P_C(\boldsymbol{x})$, is *obtuse*. The hyperplane that crosses $P_C(\boldsymbol{x})$ and is orthogonal to $\boldsymbol{x} - P_C(\boldsymbol{x})$ is known as *supporting* hyperplane and it leaves all points in C on one side and $\boldsymbol{x}$ on the other. It can be shown that if C is closed and convex and $\boldsymbol{x} \notin C$, there is always such a hyperplane; see, for example, [30].

[4] The theorems are stated here for the general case of complex numbers.

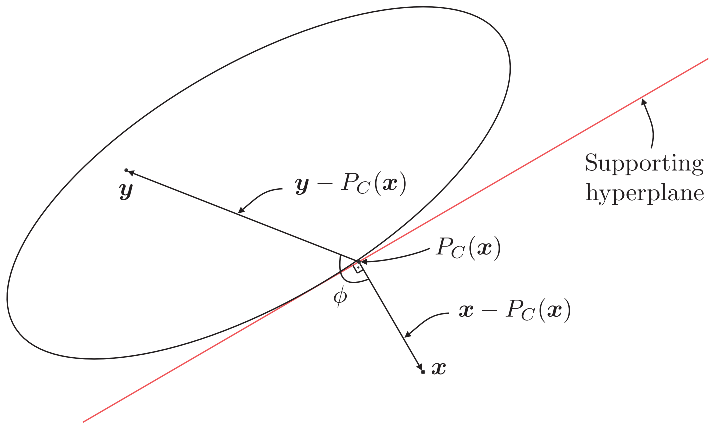


FIGURE 8.10

The vectors $y - P_C(x)$ and $x - P_C(x)$ form an obtuse angle, ϕ.

Lemma 8.1. *Let S be a closed subspace, $S \subseteq \mathbb{H}$ in a Hilbert space $\mathbb{H}$. Then $\forall\, x, y \in \mathbb{H}$, the following properties hold true:*

$$\boxed{\langle x, P_S(y)\rangle = \langle P_S(x), y\rangle = \langle P_S(x), P_S(y)\rangle,} \tag{8.18}$$

and

$$\boxed{P_S(ax + by) = aP_S(x) + bP_S(y),} \tag{8.19}$$

where a and b are arbitrary scalars. In other words, the projection operation on a closed subspace is a linear one (Problem 8.17). Recall that in a Euclidean space all subspaces are closed, hence the linearity is always valid.

It can be shown (Problem 8.18) that, if S is a closed subspace in a Hilbert space $\mathbb{H}$, its orthogonal complement, $S^\perp$, is also a closed subspace, such as $S \cap S^\perp = \{\mathbf{0}\}$; by definition, the orthogonal compliment, $S^\perp$, is the set whose elements are orthogonal to each element of S. Moreover, $\mathbb{H} = S \oplus S^\perp$; that is, each element, $x \in \mathbb{H}$, can be *uniquely* decomposed as,

$$\boxed{x = P_S(x) + P_{S^\perp}(x),\; x \in \mathbb{H}: \quad \text{For Closed Subspaces,}} \tag{8.20}$$

as this is demonstrated in Figure 8.11.

Definition 8.5. Let a mapping

$$T : \mathbb{H} \longmapsto \mathbb{H}.$$

T is called *nonexpansive* if $\forall\, x, y \in \mathbb{H}$

$$\boxed{\|T(x) - T(y)\| \leq \|x - y\| : \quad \text{Nonexpansive Mapping.}} \tag{8.21}$$

Proposition 8.2. *Let C be a closed convex set in a Hilbert space $\mathbb{H}$. Then, the associated projection operator*

$$P_C : \mathbb{H} \longmapsto C,$$

is nonexpansive.

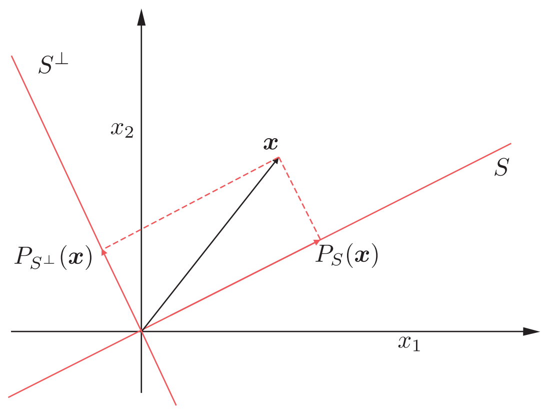


FIGURE 8.11

Every point in a Hilbert space $\mathbb{H}$ can be uniquely decomposed into the sum of its projections on any *closed* subspace S and its orthogonal complement $S^\perp$.

Proof. Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{H}$. Recall property (8.16), that is,

$$\|P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\|^2 \leq \text{Real}\left\{\langle \boldsymbol{x} - \boldsymbol{y}, P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\rangle\right\}. \tag{8.22}$$

Moreover, by employing the Schwarz inequality (Appendix 8.15, Eq. (8.147)), we get

$$|\langle \boldsymbol{x} - \boldsymbol{y}, P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\rangle| \leq \|\boldsymbol{x} - \boldsymbol{y}\| \|P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\|. \tag{8.23}$$

Combining (8.22) and (8.23), we readily obtain that

$$\|P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\| \leq \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{8.24}$$

□

Figure 8.12 provides a geometric interpretation of (8.24). The property of nonexpansiveness, as well as a number of its variants, for example, [6, 18, 30, 78, 79], are of paramount importance in convex set theory and learning. It is the property that guarantees the convergence of an algorithm, which comprises a sequence of successive projections (mappings), to the so-called *fixed point set*; that is, to the set whose elements are left unaffected by the respective mapping, T, shown as,

$$\boxed{\text{Fix}(T) = \left\{\boldsymbol{x} \in \mathbb{H} : T(\boldsymbol{x}) = \boldsymbol{x}\right\} : \quad \text{Fixed Point Set.}}$$

In the case of a projection operator on a closed convex set, we know that the respective fixed point set is the set C itself, since $P_C(\boldsymbol{x}) = \boldsymbol{x}$, $\forall \boldsymbol{x} \in C$.

Definition 8.6. Let C be a closed convex set C in a Hilbert space. An operator

$$T_C : \mathbb{H} \longmapsto C$$

is called *relaxed projection* if

$$T_C := I + \mu(P_C - I), \quad \mu \in (0, 2),$$

or in other words, $\forall\, \boldsymbol{x} \in \mathbb{H}$

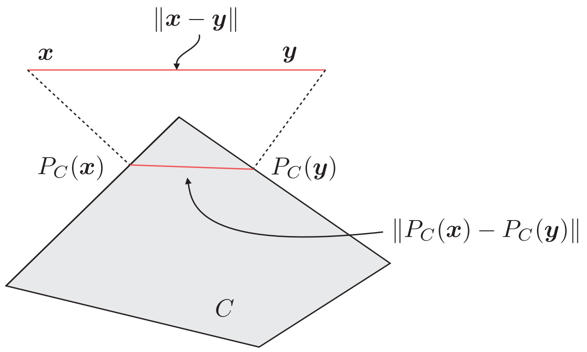


FIGURE 8.12

The nonexpansiveness property of the projection operator, $P_C(\cdot)$, guarantees that the distance between two points can never be smaller than the distance between their respective projections on a closed convex set.

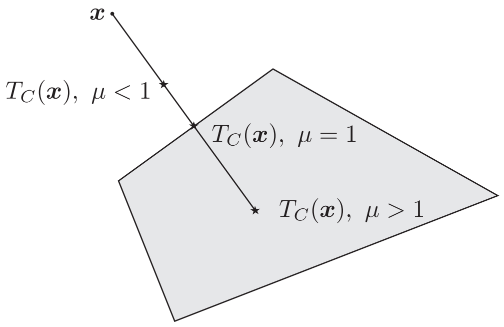


FIGURE 8.13

Geometric illustration of the relaxed projection operator.

$$\boxed{T_C(x) = x + \mu\big(P_C(x) - x\big),\ \mu \in (0,2): \quad \text{Relaxed Projection on } C.}$$

We readily see that for $\mu = 1$, $T_C(x) = P_C(x)$. Figure 8.13 shows the geometric illustration of the relaxed projection. Observe that for different values of $\mu \in (0,2)$, the relaxed projection traces all points in the line segment joining the points x and $x + 2\,(P_C(x) - x)$. Note that

$$T_C(x) = x,\ \forall x \in C,$$

that is, $\text{Fix}(T_C) = C$. Moreover, it can be shown that the relaxed projection operator is also nonexpansive, that is, (Problem 8.19)

$$\|T_C(x) - T_C(y)\| \le \|x - y\|,\ \forall \mu \in (0,2).$$

A final property of the relaxed projection, which can also be easily shown (Problem 8.20), is the following:

$$\boxed{\forall\, y \in C,\ \ \|T_C(x) - y\|^2 \le \|x - y\|^2 - \eta\|T_C(x) - x\|^2,\ \eta = \frac{2-\mu}{\mu}.} \tag{8.25}$$

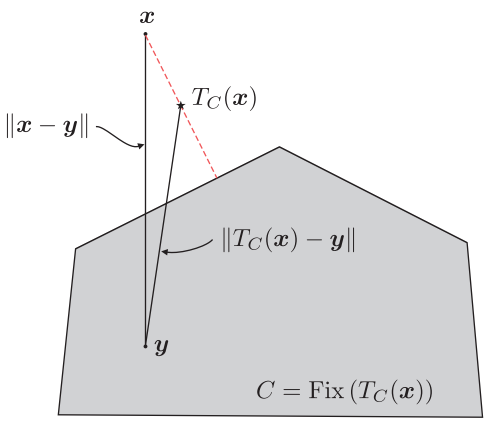

$\boldsymbol{x}$
$T_C(\boldsymbol{x})$
$\|\boldsymbol{x}-\boldsymbol{y}\|$
$\|T_C(\boldsymbol{x})-\boldsymbol{y}\|$
$\boldsymbol{y}$
$C = \text{Fix}\,(T_C(\boldsymbol{x}))$

FIGURE 8.14

The relaxed projection is a strongly attracting mapping. $T_C(\boldsymbol{x})$ is closer to any point $\boldsymbol{y} \in C = \text{Fix}(T_C)$ than the point $\boldsymbol{x}$ is.

Such mappings are known as η-*nonexpansive* or *strongly attracting* mappings; it is guaranteed that the distance $\|T_C(\boldsymbol{x}) - \boldsymbol{y}\|$ is *smaller* than $\|\boldsymbol{x} - \boldsymbol{y}\|$ at least by the positive quantity $\eta\|T_C(\boldsymbol{x}) - \boldsymbol{x}\|^2$; that is, the fixed point set $\text{Fix}(T_C) = C$ *strongly attracts* $\boldsymbol{x}$. The geometric interpretation is given in Figure 8.14.

8.4 FUNDAMENTAL THEOREM OF PROJECTIONS ONTO CONVEX SETS

In this section, one of the most celebrated theorems in the theory of convex sets is stated; the fundamental theorem of projections onto convex sets (POCS). This theorem is at the heart of a number of powerful algorithms and methods, some of which are described in this book. The origin of the theorem is traced back to Von Neumann [93], who proposed the theorem for the case of two subspaces.

Von Neumann was a Hungarian-born American of Jewish descent. He was a child prodigy who earned his Ph.D. at age 22. It is difficult to summarize his numerous significant contributions, which range from pure mathematics, to economics (he is considered the founder of the game theory) and from quantum mechanics (he laid the foundations of the mathematical framework in quantum mechanics) to computer science (he was involved in the development of ENIAC, the first general-purpose electronic computer). He was heavily involved in the Manhattan project for the development of the hydrogen bomb.

Let $C_k,\ k = 1, 2, \ldots, K$, be a *finite* number of *closed* convex sets in a Hilbert space $\mathbb{H}$, and assume that they share a *nonempty* intersection,

$$C = \bigcap_{k=1}^{K} C_k \neq \emptyset.$$

Let $T_{C_k},\ k = 1, 2, \ldots, K$, be the respective relaxed projection mappings

$$T_{C_k} = I + \mu_k(P_{C_k} - I),\ \mu_k \in (0,2), \quad k = 1, 2, \ldots, K.$$

Form the concatenation of these relaxed projections,

$$T := T_{C_k} T_{C_{k-1}} \cdots T_{C_1},$$

where the specific order is not important. In words, T comprises a sequence of relaxed projections, starting from C_1. In the sequel, the obtained point is projected onto C_2, and so on.

Theorem 8.5. *Let $C_k, k = 1, 2, \ldots, K$, be closed convex sets in a Hilbert space, $\mathbb{H}$, with nonempty intersection. Then, for any $\boldsymbol{x}_0 \in \mathbb{H}$, the sequence $(T^n(\boldsymbol{x}_0)),\ n = 1, 2, \ldots$ converges weakly to a point in $C = \bigcap_{k=1}^{K} C_k$.*

The theorem [16, 41] states the notion of *weak convergence*. When $\mathbb{H}$ becomes a Euclidean (finite dimensional) space, the notion of weak convergence coincides with the familiar "standard" definition of (strong) convergence. Weak convergence is a weaker version of the strong convergence, and it is met in infinite dimensional spaces. A sequence, $\boldsymbol{x}_n \in \mathbb{H}$, is said to converge weakly to a point $\boldsymbol{x}_* \in \mathbb{H}$, if $\forall\, \boldsymbol{y} \in \mathbb{H}$,

$$\langle \boldsymbol{x}_n, \boldsymbol{y} \rangle \xrightarrow[n\to\infty]{} \langle \boldsymbol{x}_*, \boldsymbol{y} \rangle,$$

and we write,

$$\boldsymbol{x}_n \xrightarrow[n\to\infty]{w} \boldsymbol{x}_*.$$

As already said, in Euclidean spaces, weak convergence implies strong convergence. This is not necessarily true for general Hilbert spaces. On the other hand, strong convergence always implies weak convergence, for example, [82] (Problem 8.21). Figure 8.15 gives the geometric illustration of the theorem.

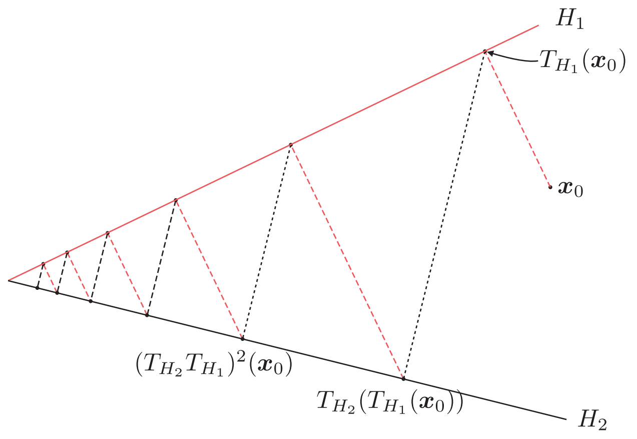

H_1
$T_{H_1}(\boldsymbol{x}_0)$
$\boldsymbol{x}_0$
$(T_{H_2}T_{H_1})^2(\boldsymbol{x}_0)$
$T_{H_2}(T_{H_1}(\boldsymbol{x}_0))$
H_2

FIGURE 8.15

Geometric illustration of the fundamental theorem of projections onto convex sets (POCS), for $T_{C_i} = P_{C_i}$, $i = 1, 2,\ (\mu_{C_i} = 1)$. The closed convex sets are the two straight lines in $\mathbb{R}^2$. Observe that the sequence of projections tends to the intersection of H_1, H_2.

The proof of the theorem is a bit technical for the general case, for example, [82]. However, it can be simplified for the case where the involved convex sets are closed subspaces (Problem 8.23). At the heart of the proof lie (a) the nonexpansiveness property of $T_{C_k}, k = 1, 2, \ldots, K$, which is retained by T and (b) the fact that the fixed point set of T is $\text{Fix}(T) = \bigcap_{k=1}^{K} C_k$.

Remarks 8.3.

- In the special case where all $C_k, k = 1, 2, \ldots, K$, are closed subspaces, then

$$T^n(\boldsymbol{x}_0) \longrightarrow P_C(\boldsymbol{x}_0).$$

  In other words, the sequence of relaxed projections converges *strongly* to the projection of $\boldsymbol{x}_0$ on C. Recall that if each $C_k, k = 1, 2, \ldots, K$, is a closed subspace of $\mathbb{H}$, it can easily be shown that their intersection is also a closed subspace. As said before, in a Euclidean space $\mathbb{R}^l$, all subspaces are closed.
- The previous statement is also true for *linear varieties*. A linear variety is the translation of a subspace by a constant vector $\boldsymbol{a}$. That is, if S is a subspace, and $\boldsymbol{a} \in \mathbb{H}$, then the set of points

$$S_{\boldsymbol{a}} = \{\boldsymbol{y} : \boldsymbol{y} = \boldsymbol{a} + \boldsymbol{x},\ \boldsymbol{x} \in S\}$$

  is a linear variety. Hyperplanes are linear varieties: see, for example, Figure 8.16.
- The scheme resulting from the POCS theorem, employing the relaxed projection operator, is summarized in Algorithm 8.1,

  Algorithm 8.1 (The POCS algorithm).

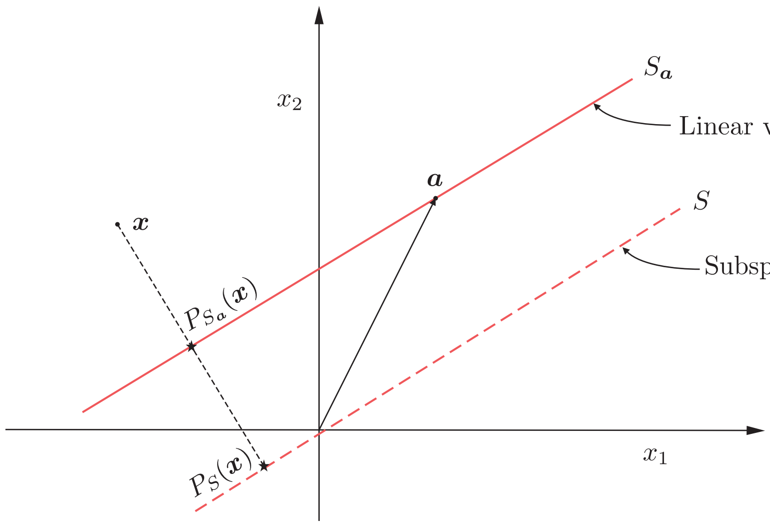


FIGURE 8.16

A hyperplane (not crossing the origin) is a linear variety. $P_{S_{\boldsymbol{a}}}$ and P_S are the projections of $\boldsymbol{x}$ onto $S_{\boldsymbol{a}}$ and S, respectively.

```
• Initialization.
    - Select $\mathbf{x}_0 \in \mathbb{H}$.
    - Select $\mu_k \in (0,2),\ k = 1, 2, \ldots, K$
• For $n = 1, 2, \ldots,$ Do
    - $\hat{\mathbf{x}}_{0,n} = \mathbf{x}_{n-1}$
    - For $k = 1, 2, \ldots, K$ Do
          $\hat{\mathbf{x}}_{k,n} = \hat{\mathbf{x}}_{k-1,n} + \mu_k \left(P_{C_k}(\hat{\mathbf{x}}_{k-1,n}) - \hat{\mathbf{x}}_{k-1,n}\right)$    (8.26)
    - End For
    - $\mathbf{x}_n = \hat{\mathbf{x}}_K$.
• End For
```

8.5 A PARALLEL VERSION OF POCS

In [65], a parallel version of the POCS algorithm was stated. In addition to its computational advantages (when parallel processing can be exploited), this scheme will be our vehicle for generalizations to the online processing, where one can cope with the case where the number of convex sets becomes infinite (or very large in practice). The proof for the parallel POCS is slightly technical and relies heavily on the results stated in the previous section. The concept behind the proof is to construct appropriate product spaces, and this is the reason that the algorithm is also referred to as POCS in *product spaces*. For the detailed proof, the interested reader may consult [65].

Theorem 8.6. *Let $C_k, k = 1, 2, \ldots, K$, be closed convex sets, in a Hilbert space, $\mathbb{H}$. Then, for any $\mathbf{x}_0 \in \mathbb{H}$, the sequence $\mathbf{x}_n$, defined as*

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mu_n \left(\sum_{k=1}^{K} \omega_k P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1} \right), \tag{8.27}$$

weakly converges to a point in $\bigcap_{k=1}^{K} C_k$, if

$$0 < \mu_n \leq M_n,$$

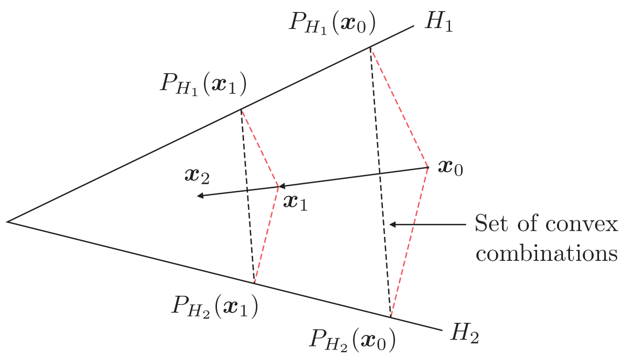
and

$$M_n := \sum_{k=1}^{K} \frac{\omega_k \|P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1}\|^2}{\|\sum_{k=1}^{K} \omega_k P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1}\|^2}, \tag{8.28}$$

where $\omega_k > 0,\ k = 1, 2, \ldots, K$, such that

$$\sum_{k=1}^{K} \omega_k = 1.$$

Update recursion (8.27) says that at each iteration, all projections on the convex sets take place *concurrently*, and then they are *convexly* combined. The extrapolation parameter, μ_n, is chosen in interval $(0, M_n]$, where M_n is recursively computed in (8.28), so that convergence in guaranteed. Figure 8.17 illustrates the updating process.



$P_{H_1}(\boldsymbol{x}_0)$
H_1
$P_{H_1}(\boldsymbol{x}_1)$
$\boldsymbol{x}_2$
$\boldsymbol{x}_0$
$\boldsymbol{x}_1$
Set of convex combinations
$P_{H_2}(\boldsymbol{x}_1)$
$P_{H_2}(\boldsymbol{x}_0)$
H_2

FIGURE 8.17

The parallel POCS algorithm for the case of two (hyperplanes) lines in $\mathbb{R}^2$. At each step, the projections on $H_1 and H_2$ are carried out in parallel and then they are convexly combined.

8.6 FROM CONVEX SETS TO PARAMETER ESTIMATION AND MACHINE LEARNING

Let us now see how this elegant theory can be turned into a useful tool for parameter estimation in machine learning. We will demonstrate the procedure using two examples.

8.6.1 REGRESSION

Consider the regression model, relating input-output observation points,

$$y_n = \boldsymbol{\theta}_o^{\mathrm{T}} \boldsymbol{x}_n + \eta_n, \ (y_n, \boldsymbol{x}_n) \in \mathbb{R} \times \mathbb{R}^l, \quad n = 1, 2, \ldots, N, \tag{8.29}$$

where $\boldsymbol{\theta}_o$ is the unknown parameter vector. Assume that η_n is a bounded noise sequence, that is,

$$|\eta_n| \leq \epsilon. \tag{8.30}$$

Then, (8.29), (8.30) guarantee that

$$|y_n - \boldsymbol{x}_n^T \boldsymbol{\theta}_o| \leq \epsilon. \tag{8.31}$$

Consider now the following set of points

$$\boxed{S_\epsilon = \left\{ \boldsymbol{\theta} : |y_n - \boldsymbol{x}_n^T \boldsymbol{\theta}| \leq \epsilon \right\} : \quad \text{Hyperslab.}} \tag{8.32}$$

This set is known as a *hyperslab*, and it is geometrically illustrated in Figure 8.18. The definition is generalized for any $\mathbb{H}$ by replacing the inner product notation as $\langle \boldsymbol{x}_n, \boldsymbol{\theta} \rangle$. The set comprises all the points that lie in the region formed by the two hyperplanes

$$\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{\theta} - y_n = \epsilon,$$
$$\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{\theta} - y_n = -\epsilon.$$

This region is trivially shown to be a *closed convex* set. Note that *every pair* of training points, $(y_n, \boldsymbol{x}_n), \ n = 1, 2, \ldots, N$, defines a hyperslab of different orientation (depending on $\boldsymbol{x}_n$) and position

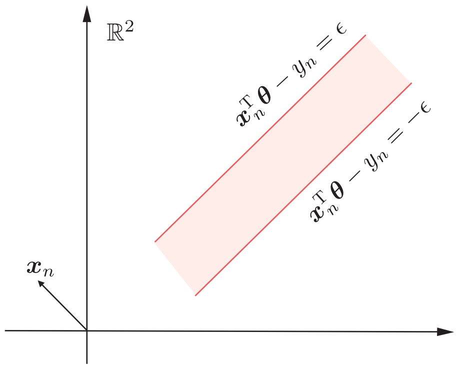

$\mathbb{R}^2$
$\boldsymbol{x}_n^T\boldsymbol{\theta} - y_n = \epsilon$
$\boldsymbol{x}_n^T\boldsymbol{\theta} - y_n = -\epsilon$
$\boldsymbol{x}_n$

FIGURE 8.18

Each pair of training points, $(y_n, \boldsymbol{x}_n)$, defines a hyperslab in the parameters' space.

in space (determined by y_n). Moreover, (8.31) guarantees that the unknown, $\boldsymbol{\theta}_o$, lies within all these hyperslabs; hence, $\boldsymbol{\theta}_o$ lies in their *intersection*. All we need now is to derive the projection operator onto hyperslabs (we will do it soon), and use one of the POCS schemes to find a point in the intersection. Assuming that enough training points are available and that the intersection is "small" enough, then any point in this intersection will be "close" to $\boldsymbol{\theta}_o$. Note that such a procedure is not based on optimization arguments. Recall, however, that even in optimization techniques, iterative algorithms have to be used, and in practice, iterations have to stop after a finite number of steps. Thus, one can only approximately reach the optimal value. More on these issues and related convergence properties will be discussed later in this chapter.

The obvious question now is what happens if the noise is not bounded. There are two answers to this point. First, in any practical application where measurements are involved, the noise has to be bounded. Otherwise, the circuits will be burned out. So, at least conceptually, this assumption does not conflict with what happens in practice. It is a matter of selecting the right value for ϵ. The second answer is that one can choose ϵ to be a few times the standard deviation of the assumed noise model. Then, $\boldsymbol{\theta}_o$ will lie in these hyperslabs with high probability. We will discuss strategies for selecting ϵ, but our goal in this section is to discuss the main rationale in using the theory in practical applications. Needless to say there is nothing divine around hyperslabs. Other closed convex sets can also be used, if the nature of the noise in a specific application suggests a different type of convex sets.

It is now interesting to look at the set where the solution lies, in this case at the hyperslab, from a different perspective. Consider the loss function,

$$\boxed{\mathcal{L}(y, \boldsymbol{\theta}^T\boldsymbol{x}) = \max\left(0,\ |y - \boldsymbol{x}^T\boldsymbol{\theta}| - \epsilon\right): \quad \text{Linear } \epsilon\text{-Insensitive Loss Function,}} \tag{8.33}$$

which is illustrated in Figure 8.19 for the case $\theta \in \mathbb{R}$. This is known as linear ϵ-insensitive loss function, and it has been popularized in the context of support vector regression (Chapter 11). For all $\boldsymbol{\theta}$s, which lie within the hyperslab defined in (8.32), the loss function scores a zero. For points outside the hyperslab, there is a linear increase of its value. Thus, the hyperslab is the *zero level set* of the linear ϵ-insensitive loss function, defined *locally* according to the point $(y_n, \boldsymbol{x}_n)$. Thus, although no optimization concept is associated with POCS, the choice of the closed convex sets can be done to minimize "locally," at each point, a convex loss function by selecting its zero level set.

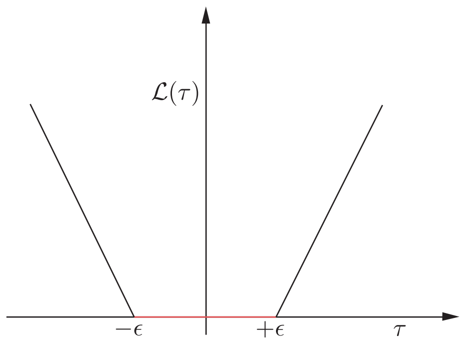


FIGURE 8.19

The linear ϵ-insensitive loss function, $\tau = y - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$. Its value is zero if $|\tau| < \epsilon$, and increases linearly for $|\tau| \geq \epsilon$.

We conclude our discussion by providing the projection operator of a hyperslab, S_ϵ. It is trivially shown that, given $\boldsymbol{\theta}$, its projection onto S_ϵ (defined by $(y_n, \boldsymbol{x}_n, \epsilon)$) is given by

$$P_{S_\epsilon} = \boldsymbol{\theta} + \beta_{\boldsymbol{\theta}}(y_n, \boldsymbol{x}_n)\boldsymbol{x}_n, \tag{8.34}$$

where

$$\beta_{\boldsymbol{\theta}}(y_n, \boldsymbol{x}_n) = \begin{cases} \frac{y_n - \langle \boldsymbol{x}_n, \boldsymbol{\theta}\rangle - \epsilon}{\|\boldsymbol{x}_n\|^2}, & \text{if } \langle \boldsymbol{x}_n, \boldsymbol{\theta}\rangle - y_n < -\epsilon, \\ 0, & \text{if } |\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_n| \leq \epsilon, \\ \frac{y_n - \langle \boldsymbol{x}_n, \boldsymbol{\theta}\rangle + \epsilon}{\|\boldsymbol{x}_n\|^2}, & \text{if } \langle \boldsymbol{x}_n, \boldsymbol{\theta}\rangle - y_n > \epsilon. \end{cases} \tag{8.35}$$

That is, if the point lies within the hyperslab, it coincides with its projection. Otherwise, the projection is on one of the two hyperplanes (depending on which side of the hyperslab the point lies), which define S_ϵ. Recall that the projection of a point lies on the boundary of the corresponding closed convex set.

8.6.2 CLASSIFICATION

Let us consider the two-class classification task, and assume that we are given the set of training points, $(y_n, \boldsymbol{x}_n), n = 1, 2, \ldots, N$.

Our goal now will be to design a linear classifier so that to score

$$\boldsymbol{\theta}^T\boldsymbol{x}_n \geq \rho, \text{ if } y_n = +1,$$

and

$$\boldsymbol{\theta}^T\boldsymbol{x}_n \leq -\rho, \text{ if } y_n = -1.$$

This requirement can be expressed as: Given $(y_n, \boldsymbol{x}_n) \in \{-1, 1\} \times \mathbb{R}^{l+1}$, design a linear classifier,[5] $\boldsymbol{\theta} \in \mathbb{R}^{l+1}$, such that

$$y_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n \geq \rho > 0. \tag{8.36}$$

[5] Recall from Chapter 3, that this formulation covers the general case where a bias term is involved, by increasing the dimensionality of $\boldsymbol{x}_n$ and adding 1 as its last element.

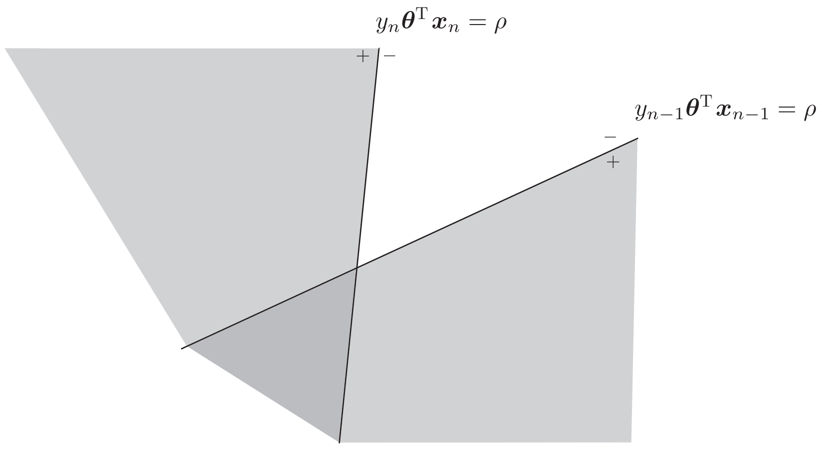

$y_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n = \rho$
$+$
$-$
$y_{n-1}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_{n-1} = \rho$
$-$
$+$

FIGURE 8.20

Each training point, $(y_n, \boldsymbol{x}_n)$, defines a halfspace in the parameters $\boldsymbol{\theta}$-space, and the linear classifier will be searched in the intersection of all these halfspaces.

Note that, given $y_n, \boldsymbol{x}_n$, and ρ, (8.36) defines a *halfspace* (Example 8.1); this is the reason that we used "$\geq \rho$" rather than a strict inequality. In other words, all $\boldsymbol{\theta}$'s, which satisfy the desired inequality (8.36) lie in this halfspace. Since each pair, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, defines a single halfspace, our goal now becomes that of trying to find a point at the intersection of all these halfspaces. This intersection is guaranteed to be nonempty if the classes are linearly separable. Figure 8.20 illustrates the concept. The more realistic case, of nonlinearly separable classes, will be treated in Chapter 11, where a mapping in a high dimensional (kernel) space makes the probability of two classes being linearly separable to tend to 1, as the dimensionality of the kernel space goes to infinity.

The halfspace associated with a training pair, $(y_n, \boldsymbol{x}_n)$, can be seen as the level set of height zero of the so-called *hinge loss* function, defined as

$$\boxed{\mathcal{L}_\rho(y, \boldsymbol{\theta}^T\boldsymbol{x}) = \max\left(0, \rho - y\boldsymbol{\theta}^T\boldsymbol{x}\right): \quad \text{Hinge Loss Function},} \tag{8.37}$$

whose graph is shown in Figure 8.21. Thus, choosing the halfspace as the closed convex set to represent $(y_n, \boldsymbol{x}_n)$, is equivalent with selecting the zero level set of the hinge loss, "adjusted" for the point $(y_n, \boldsymbol{x}_n)$.

Remarks 8.4.

- In addition to the two applications typical of the machine learning point of view, POCS has been applied in a number of other applications; see, for example, [18, 24, 82, 84], for further reading.
- If the involved sets do not intersect, that is, $\bigcap_{k=1}^{K} C_k = \emptyset$, then it has been shown [25] that, the parallel version of POCS in (8.27) converges to a point whose weighted squared distance from each one of the convex sets (defined as the distance of the point from its respective projection) is minimized.
- Attempts to generalize the theory to nonconvex sets have also been made, for example, [82] and more recently in the context of sparse modeling in [80].

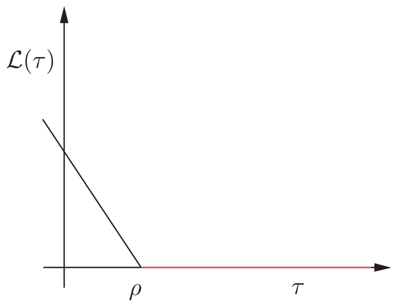


FIGURE 8.21

The hinge loss function. For the classification task, $\tau = y\boldsymbol{\theta}^\top \boldsymbol{x}$, its value is zero if $\tau \geq \rho$, and increases linearly for $\tau < \rho$.

- When $C := \bigcap_{k=1}^{K} C_K \neq \emptyset$, we say that the problem is *feasible* and the intersection C is known as the *feasibility* set. The closed convex sets, $C_k,\ k = 1, 2, \ldots, K$, are sometimes called the *property* sets, for obvious reasons. In both previous examples, namely the regression and the classification, we commented that the involved property sets resulted as the 0-level sets of a loss function $\mathcal{L}$. Hence, assuming that the problem is feasible (the cases of bounded noise in regression and linearly separable classes in classification), any solution in the feasible set C, will also be a *minimizer* of the respective loss functions in (8.33), (8.37), respectively. Thus, although optimization did not enter into our discussion, there can be an optimizing flavor in the POCS method. Moreover, note that in this case, the loss functions need *not* be differentiable and the techniques we discussed so far in the previous chapters are not applicable. We will return to this issue later on in Section 8.10.

8.7 INFINITE MANY CLOSED CONVEX SETS: THE ONLINE LEARNING CASE

In our discussion so far, we have assumed a finite number, K, of closed convex (property) sets. To land at their intersection (feasibility set) one has to cyclically project onto all of them or to perform the projections in parallel. Such a strategy is not appealing for the online processing scenario. At every time instant, a new pair of observations becomes available, which defines a new property set. Hence, in this case, the number of the available convex sets gets increased. Visiting all the available sets makes the complexity time dependent and after some time the required computational resources will become unmanageable.

An alternative viewpoint was suggested in [96–98], and later on extended in [73, 99, 100]. The main idea here is that at each time instant, n, a pair of output-input training data is received and a (property) closed convex set, C_n, is constructed. The time index, n, is left to grow unbounded. However, at each time instant, the q (a user-defined parameter) most recently constructed property sets are considered. In other words, the parameter q defines a *sliding window* in time. At each time instant, projections/relaxed projections are performed within this time window. The rationale is illustrated in Figure 8.22. Thus, the number of sets onto which projections are performed does not grow with time, their number remains finite, and it is fixed by the user. The developed algorithm is an offspring of the parallel version of

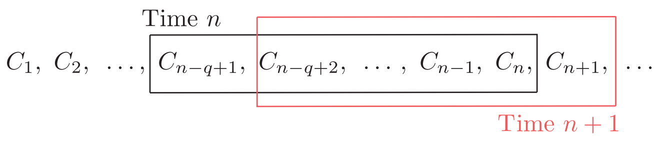

Time n
$C_1,\ C_2,\ \ldots,\ C_{n-q+1},\ C_{n-q+2},\ \ldots,\ C_{n-1},\ C_n,\ C_{n+1},\ \ldots$
Time $n+1$

FIGURE 8.22

At time n, the property sets $C_{n-q+1}, \ldots, C_n$ are used, while at time $n+1$, the sets $C_{n-q+2}, \ldots, C_{n+1}$ are considered. Thus, the required number of projections does not grow with time.

POCS and it is known as *adaptive projected subgradient method* (APSM), for reasons that will become clear later on in Section 8.10.3. We will describe the algorithm in the context of regression. Following the discussion in Section 8.6, as each pair, $(y_n, \boldsymbol{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, becomes available, a hyperslab, $S_{\epsilon,n}, n = 1, 2, \ldots$, is constructed and the goal is to find a $\boldsymbol{\theta} \in \mathbb{R}^l$, that lies in the intersection of all these property sets, starting from an arbitrary value, $\boldsymbol{\theta}_0 \in \mathbb{R}^l$.

Algorithm 8.2 (The APSM algorithm).

- Initialization
  - Choose $\boldsymbol{\theta}_0 \in \mathbb{R}^l$.
  - Choose q; The number of property sets to be processed at each time instant.
- **For** $n = 1, 2, \ldots, q-1$, **Do**; Initial period, that is, $n < q$.
  - Choose $\omega_1, \ldots, \omega_n: \quad \sum_{k=1}^{n} \omega_k = 1, \omega_k \geq 0$
  - Select μ_n

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{k=1}^{n} \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right) \tag{8.38}$$

- **End For**
- **For** $n = q, q+1, \ldots,$ **Do**
  - Choose $\omega_n, \ldots, \omega_{n-q+1}$; usually $\omega_k = \frac{1}{q}, k = n-q+1, \ldots, n$.
  - Select μ_n

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{k=n-q+1}^{n} \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right) \tag{8.39}$$

- **End For**

The extrapolation parameter can now be chosen in the interval $(0, 2M_n)$ in order for convergence to be guaranteed. For the case of (8.39)

$$M_n = \sum_{k=n-q+1}^{n} \frac{\omega_k \| P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \|^2}{\left\| \sum_{k=n-q+1}^{n} \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right\|^2}. \tag{8.40}$$

Note that this interval differs from that reported in the case of a finite number of sets, in Eq. (8.27). For the first iteration steps associated with Eq. (8.38), the summations in the above formula starts from $k = 1$ instead of $k = n - q + 1$.

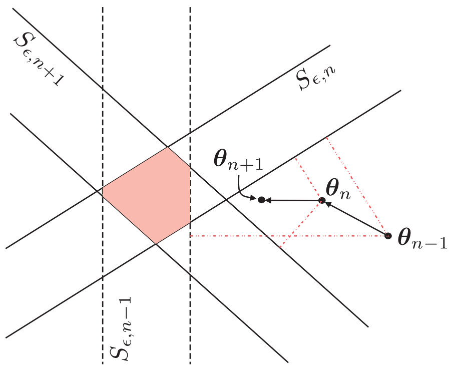


FIGURE 8.23

At time, n, $q = 2$ hyperslabs are processed, namely, $S_{\epsilon,n}$, $S_{\epsilon,n-1}$. $\boldsymbol{\theta}_{n-1}$ is *concurrently* projected onto both of them and the projections are convexly combined. The new estimate is $\boldsymbol{\theta}_n$. Next $S_{\epsilon,n+1}$ "arrives" and the process is repeated. Note that at every time, the estimate gets closer to the intersection; the latter will become smaller and smaller, as more hyperslabs arrive.

Recall that, $P_{S_{\epsilon,k}}$ is the projection operation given in (8.34)-(8.35). Note that this is a generic scheme and can be applied with different property sets. All that is needed is to change the projection operator. For example, if classification is considered, all we have to do is to replace $S_{\epsilon,n}$ by the halfspace H_n^+, defined by the pair $(y_n, \boldsymbol{x}_n) \in \{-1, 1\} \times \mathbb{R}^{l+1}$, as explained in Section 8.6.2 and use the projection from (8.13); see [75, 76]. At this point, it must be emphasized that the original APSM form (e.g., [98, 99]) is more general and can cover a wide range of convex sets and functions. We will come back to this in Remarks 8.8.

Figure 8.23 illustrates geometrically the APSM algorithm. We have assumed that the number of hyperslabs that are considered for projection at each time instant is $q = 2$. Each iteration comprises:

- q projections, which can be carried out in parallel,
- their convex combination, and
- the update step.

8.7.1 CONVERGENCE OF APSM

The proof of the convergence of the APSM is a bit technical and the interested reader can consult the related references. Here, we can be content with a geometric illustration that intuitively justifies the convergence, under certain assumptions. This geometric interpretation is at the heart of a stochastic approach to the APSM convergence, which was presented in [23].

Assume the noise to be bounded and that there is a true $\boldsymbol{\theta}_o$ that generates the data, that is,

$$y_n = \boldsymbol{x}_n^T \boldsymbol{\theta}_o + \eta_n. \tag{8.41}$$

By assumption,

$$|\eta_n| \le \epsilon,$$

hence,

$$|\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_o - y_n| \leq \epsilon.$$

Thus, $\boldsymbol{\theta}_o$ does lie in the intersection of all the hyperslabs of the form

$$|\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta} - y_n| \leq \epsilon,$$

and in this case the problem is feasible. The question that is raised is how close one can go, even asymptotically as $n \longrightarrow \infty$, to $\boldsymbol{\theta}_o$. For example, if the volume of the intersection is large, even if the algorithm converges to a point in the boundary of this intersection does not necessarily say much about how close the solution is to the true value $\boldsymbol{\theta}_o$. The proof in [23] establishes that the algorithm brings the estimate *arbitrarily close* to $\boldsymbol{\theta}_o$, under some general assumptions concerning the sequence of observations and that the noise is bounded.

To understand what is behind the technicalities of the proof, recall that there are two main geometric issues concerning a hyperslab: (a) its orientation, which is determined by $\boldsymbol{x}_n$ and (b) its width. In finite dimensional spaces, it is a matter of simple geometry to show that the width of a hyperslab is equal to

$$d = \frac{2\epsilon}{\|\boldsymbol{x}_n\|}. \tag{8.42}$$

This is a direct consequence of the fact that the distance[6] of a point, say $\tilde{\boldsymbol{\theta}}$, from the hyperplane defined by the pair $(y, \boldsymbol{x})$, that is,

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} - y = 0,$$

is equal to

$$\frac{|\boldsymbol{x}^{\mathrm{T}}\tilde{\boldsymbol{\theta}} - y|}{\|\boldsymbol{x}\|}.$$

Indeed, let $\bar{\boldsymbol{\theta}}$ be a point on one of the two boundary hyperplanes (e.g., $\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta} - y_n = \epsilon$), which define the hyperslab and consider its distance from the other one ($\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta} - y_n = -\epsilon$); then, (8.42) is readily obtained.

Figure 8.24 shows four hyperslabs in two different directions (one for the full lines and one for the dotted lines). The red hyperslabs are narrower than the black ones. Moreover, all four necessarily include $\boldsymbol{\theta}_o$. If $\boldsymbol{x}_n$ is left to vary randomly so that any orientation will occur, with high probability and for any orientation, the norm can also take small as well as arbitrarily large values, then intuition says that the volume of the intersection around $\boldsymbol{\theta}_0$ will become arbitrarily small.

Some practical hints

The APSM algorithm needs the setting of three parameters, namely, ϵ, μ_n, and q. It turns out that the algorithm is not particularly sensitive in their choice:

- The choice of the parameter μ_n is similar in concept to the choice of the step-size in the LMS algorithm. In particular, the larger the μ_n the faster the convergence speed, at the expense of a higher steady-state error floor. In practice, a step-size approximately equal to $0.5M_n$ will lead to a

[6] For Euclidean spaces, this can be easily established by simple geometric arguments; see also, Section 11.10.1.

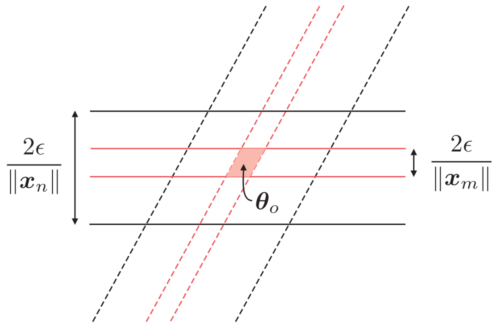

$\frac{2\epsilon}{\|\boldsymbol{x}_n\|}$
$\frac{2\epsilon}{\|\boldsymbol{x}_m\|}$
$\boldsymbol{\theta}_o$

FIGURE 8.24

For each direction, the width of a hyperslab varies inversely proportional to $\|\boldsymbol{x}_n\|$. In this figure, $\|\boldsymbol{x}_n\| < \|\boldsymbol{x}_m\|$ although both vectors point to the same direction. The intersection of hyperslabs of different directions and widths renders the volume of their intersection arbitrarily small around $\boldsymbol{\theta}_o$.

low steady-state error, albeit the convergence speed will be relatively slow. On the contrary, if one chooses a larger step-size, $1.5M_n$ approximately, then the algorithm enjoys a faster convergence speed, although the steady-state error after convergence is increased.

- Regarding the parameter ϵ, a typical choice is to set $\epsilon \approx \sqrt{2}\sigma$, where σ is the standard deviation of the noise. In practice (see, e.g. [46]), it has been shown that the algorithm is rather insensitive to this parameter. Hence, one needs only a *rough* estimate of the standard deviation.
- Concerning the choice of q, this is analogous to the q used for the APA in Chapter 5. The larger the q is, the faster the convergence becomes; however, large values of q increase complexity as well as the error floor after convergence. In practice, relatively small values of q, for example, a small fraction of the l, can significantly improve the convergence speed compared to the normalized least-mean-squares algorithm (NLMS). Sometimes, one can start with a relatively large value of q, and once the error decreases, q can be given smaller values to achieve lower error floors.

It is important to note that the past data reuse, within the sliding window of length q in the APA algorithm is implemented via the inversion of a $q \times q$ matrix. In the APSM, this is achieved via a sequence of q projections, leading to a complexity of linear dependence on q; moreover, these projections can be performed in parallel. Furthermore, the APA tends to be more sensitive to the presence of noise, since the projections are carried out on hyperplanes. In contrast, for the APSM case, projections are performed on hyperslabs, which implicitly care for the noise, for example, [97].

Remarks 8.5.

- If the hyperslabs collapse to hyperplanes ($\epsilon = 0$) and $q = 1$, the algorithm becomes the NLMS. Indeed, for this case the projection in (8.39) becomes the projection on the hyperplane, H, defined by $(y_n, \boldsymbol{x}_n)$, that is,

$$\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta} = y_n,$$

and from (8.11), after making the appropriate notational adjustments, we have

$$P_H(\boldsymbol{\theta}_{n-1}) = \boldsymbol{\theta}_{n-1} - \frac{\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_{n-1} - y_n}{\|\boldsymbol{x}_n\|^2}\boldsymbol{x}_n. \tag{8.43}$$

Plugging (8.43) into (8.39), we get

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu_n}{\|\boldsymbol{x}_n\|^2} e_n \boldsymbol{x}_n,$$

$$e_n = y_n - \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{\theta}_{n-1},$$

which is the normalized LMS, introduced in Section 5.6.1.

- Closely related to the APSM algorithmic family are the *set-membership* algorithms, for example, [29, 32–34, 60]. This family can be seen as a special case of the APSM philosophy, where only special types of convex sets are used, for example, hyperslabs. Also, at each iteration step, a single projection is performed onto the set associated with the most recent pair of observations. For example, in [34, 94] the update recursion of a set-membership APA is given by

$$\boldsymbol{\theta}_n = \begin{cases} \boldsymbol{\theta}_{n-1} + X_n (X_n^T X_n)^{-1} (\boldsymbol{e}_n - \boldsymbol{y}_n), & \text{if } |e_n| > \epsilon, \\ \boldsymbol{\theta}_{n-1}, & \text{otherwise,} \end{cases} \tag{8.44}$$

where $X_n = [\boldsymbol{x}_n, \boldsymbol{x}_{n-1}, \ldots, \boldsymbol{x}_{n-q+1}]$, $\boldsymbol{y}_n = [y_n, y_{n-1}, \ldots, y_{n-q+1}]^{\mathrm{T}}$, $\boldsymbol{e}_n = [e_n, e_{n-1}, \ldots, e_{n-q+1}]^{\mathrm{T}}$, with $e_n = y_n - \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{\theta}_{n-1}$. The stochastic analysis of the set-membership APA [34] establishes a mean-square error (MSE) performance, and the analysis is carried out by adopting energy conservation arguments (Chapter 5).

Example 8.2. The goal of this example is to demonstrate the comparative convergence performance of the NLMS, the APA, the APSM, and the recursive least-squares RLS algorithms. The experiments were performed in two different noise settings, one for low and one for high noise levels, to demonstrate the sensitivity of the APA algorithm compared to the APSM. Data were generated according to our familiar model

$$y_n = \boldsymbol{\theta}_o^T \boldsymbol{x}_n + \eta_n.$$

The parameters $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$ were randomly chosen from a $\mathcal{N}(0, 1)$ and then fixed. The input vectors were formed by a white noise sequence with samples i.i.d. drawn from a $\mathcal{N}(0, 1)$.

In the first experiment, the white noise sequence was chosen to have variance $\sigma^2 = 0.01$. The parameters for the three algorithms were chosen as $\mu = 1.2$ and $\delta = 0.001$ for the NLMS, $q = 30$, $\mu = 0.2$, and $\delta = 0.001$ for the APA and $\epsilon = \sqrt{2}\sigma$, $q = 30$, $\mu_n = 0.5 * M_n$ for the APSM. These parameters lead the algorithms to settle at the same error floor. Figure 8.25 shows the obtained squared error, averaged over 100 realizations, in dBs ($10 \log_{10}(e_n^2)$). For comparison, the RLS convergence curve is given for $\beta = 1$, which converges faster and at the same time settles at a lower error floor. If β is modified to a smaller value so that the RLS settles at the same error floor as the other algorithms, then its convergence gets even faster. However, this improved performance of the RLS is achieved at higher complexity, which becomes a problem for large values of l. Observe the faster convergence achieved by the APA and APSM, compared to the NLMS.

For the high-level noise, the corresponding variance was increased to 0.3. The obtained MSE curves are shown in Figure 8.26. Observe that now, the APA shows an inferior performance compared to APSM in spite of its higher complexity, due to the involved matrix inversion.

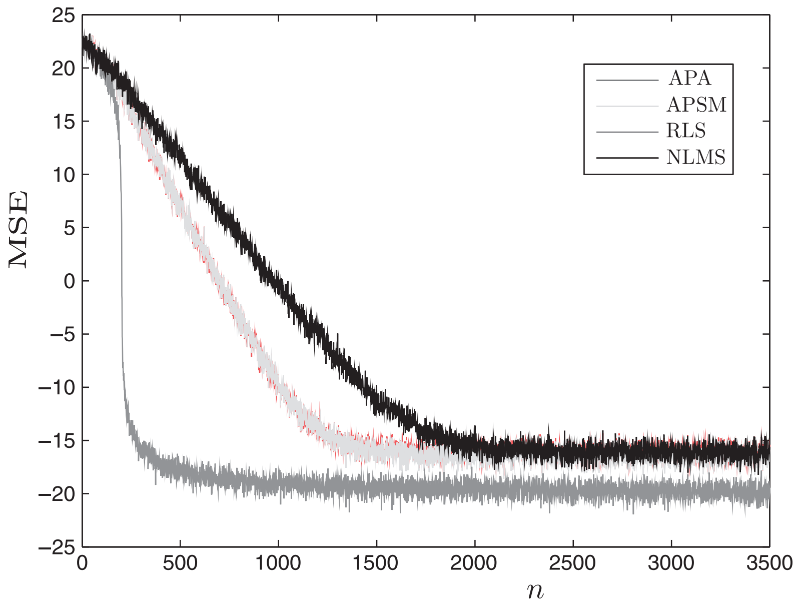


FIGURE 8.25

Mean-square error in dBs as a function of iterations. The data reuse ($q = 30$), associated with the APA and APSM, offer a significant improvement in convergence speed compared to the NLMS. The curves for the APA and APSM almost coincide in this low noise scenario.

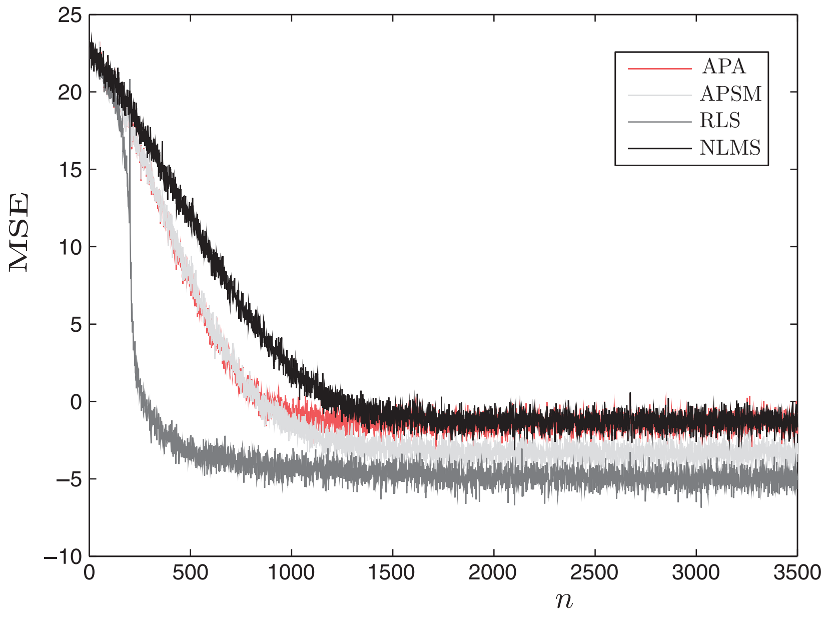


FIGURE 8.26

Mean-square error in dBs as a function of iterations for a high noise scenario. Compared to Figure 8.25, all curves settled at higher noise levels. Moreover, notice that now the APA settles at higher error floor than the corresponding APSM algorithm, for the same convergence rate.

8.8 CONSTRAINED LEARNING

Learning under a set of constraints is of significant importance in signal processing and machine learning, in general. We have already discussed a number of such learning tasks. Beamforming, discussed in Chapters 5 and 4, is a typical one. In Chapter 3, while introducing the concept of overfitting, we discussed the notion of regularization, which is another form of constraining the norm of the unknown parameter vector. In some other cases, we have available a priori information concerning the unknown parameters; this extra information can be given in the form of a set of constraints.

For example, if one is interested in obtaining estimates of the pixels in an image, then the values must be nonnegative. More recently, the unknown parameter vector may be known to be sparse; that is, only a few of its components are nonzero. In this case, constraining the respective ℓ_1 norm can significantly improve the accuracy as well as the convergence speed of an iterative scheme toward the solution. Schemes that explicitly take into consideration the underlying sparsity are known as *sparsity-promoting* algorithms, and they will be considered in more detail in Chapter 10.

Algorithms that spring from the POCS theory are particularly suited to treat constraints in an elegant, robust, and rather straightforward way. Note that the goal of each constraint is to define a region in the solution space, where the required estimate is "forced" to lie. For the rest of this section, we will assume that the required estimate must satisfy M constraints, each one defining a *convex* set of points, $C_m, m = 1, 2, \ldots, M$. Moreover,

$$\bigcap_{m=1}^{M} C_m \neq \emptyset,$$

which means that the constraints are consistent (there are also methods where this condition can be relaxed). Then, it can be shown that the mapping, T, defined as

$$T := P_{C_m} \ldots P_{C_1},$$

is a *strongly attracting nonexpansive* mapping, (8.25), for example, [6, 18]. Note that the same holds true if instead of the concatenation of the projection operators, one could convexly combine them.

In the presence of a set of constraints, the only difference in the APSM in Algorithm 8.2 is that the update recursion (8.39) is now replaced by

$$\boldsymbol{\theta}_n = T\left(\boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{k=n-q+1}^{n} \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1}\right)\right). \tag{8.45}$$

In other words, for M constraints, M extra projection operations have to be performed. The same applies to (8.38) with the difference being in the summation term in the brackets.

Remarks 8.6.

- The constrained form of the APSM has been successfully applied in the beamforming task, and in particular in treating nontrivial constraints, as it is required in the robust beamforming case [74, 77, 98, 99]. The constrained APSM has also been efficiently used for sparsity-aware learning, for example, [46, 80] (see also Chapter 10). A more detailed review of related techniques is presented in [84].

8.9 THE DISTRIBUTED APSM

Distributed algorithms were discussed in Chapter 5. In Section 5.13.2, two versions of the diffusion LMS were introduced, namely the adapt-then-combine and the combine-then-adapt schemes. Diffusion versions of the APSM algorithm have also appeared in both configurations [20, 22]. For the APSM case, both schemes result in very similar performance.

Following the discussion in Section 5.13.2, let the most recently received data pair at node $k = 1, 2, \dots, K$, be $(y_k(n), \boldsymbol{x}_k(n)) \in \mathbb{R} \times \mathbb{R}^l$. For the regression task, a corresponding hyperslab is constructed, that is,

$$S_{\epsilon,n}^{(k)} = \left\{\boldsymbol{\theta} : |y_k(n) - \boldsymbol{x}_k^T(n)\boldsymbol{\theta}| \le \epsilon_k\right\}.$$

The goal is the computation of a point that lies in the intersection of all these sets, for $n = 1, 2, \dots$. Following similar arguments as those employed for the diffusion LMS, the combine-then-adapt version of the APSM, given in Algorithm 8.3, is obtained.

Algorithm 8.3 (The combine-then-adapt diffusion APSM).

```
• Initialization
    • For $k = 1, 2, \dots, K$, Do
        - $\boldsymbol{\theta}_k(0) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
    • End For
    • Select $A$ :   $A^T\mathbf{1} = \mathbf{1}$
    • Select $q$; The number of property sets to be processed at each time instant.
• For $n = 1, 2, \dots, q-1$, Do; Initial period, that is, $n < q$.
    • For $k = 1, 2, \dots, K$, Do
        - $\boldsymbol{\psi}_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\theta}_m(n-1)$; $\mathcal{N}_k$ the neighborhood of node $k$.
    • End For
    • For $k = 1, 2, \dots, K$, Do
        - Choose $\omega_1, \dots, \omega_n$:   $\sum_{j=1}^n \omega_j = 1, \ \ \omega_j > 0$
        - Select $\mu_k(n) \in (0, \ 2M_k(n))$.
          $$\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k(n)\left(\sum_{j=1}^{n} \omega_j P_{S_{\epsilon,j}^{(k)}}\big(\boldsymbol{\psi}_k(n-1)\big) - \boldsymbol{\psi}_k(n-1)\right)$$
    • End For
• For $n = q, q+1 \dots$, Do
    • For $k = 1, 2, \dots, K$, Do
        - $\boldsymbol{\psi}_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\theta}_m(n-1)$
    • End For
    • For $k = 1, 2, \dots, K$, Do
        - Choose $\omega_n, \dots, \omega_{n-q+1}$:   $\sum_{j=n-q+1}^n \omega_j = 1, \ \ \omega_j > 0$.
        - Select $\mu_k(n) \in (0, \ 2M_k(n))$.
          $$\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k(n)\left(\sum_{j=n-q+1}^{n} \omega_j P_{S_{\epsilon,j}^{(k)}}\big(\boldsymbol{\psi}_k(n-1)\big) - \boldsymbol{\psi}_k(n-1)\right)$$
    • End For
• End For
```

The interval $M_{k,n}$ is defined as

$$M_k(n) = \sum_{j=n-q+1}^{n} \frac{\omega_j \| P_{S_{\epsilon,j}^{(k)}}(\boldsymbol{\psi}_k(n-1)) - \boldsymbol{\psi}_k(n-1)\|^2}{\| \sum_{j=n-q+1}^{n} \omega_j P_{S_{\epsilon,j}^{(k)}}(\boldsymbol{\psi}_k(n-1)) - \boldsymbol{\psi}_k(n-1)\|^2},$$

and similarly for the initial period.

Remarks 8.7.

- An important theoretical property of the APSM-based diffusion algorithms is that they enjoy *asymptotic consensus*. In other words, the nodes converge, asymptotically, to the *same* estimate. This asymptotic consensus is not in the mean, as it is the case with the diffusion LMS. This is interesting, since no explicit consensus constraints are employed.
- In [22], an extra projection step is used after the combination and prior to the adaptation step. The goal of this extra step is to “harmonize” the local information, which comprises the input/output measurements, with the information coming from the neighborhood, that is, the estimates obtained from the neighboring nodes. This speeds up convergence, at the cost of only one extra projection.
- A scenario in which some of the nodes are damaged and the associated observations are very noisy is also treated in [22]. To deal with such a scenario, instead of the hyperslab, the APSM algorithm is rephrased around the Huber loss function, developed in the context of robust statistics, to deal with cases where outliers are present (see also Chapter 11).

Example 8.3. The goal of this example is to demonstrate the comparative performance of the diffusion LMS and APSM. A network of $K = 10$ nodes is considered, and there are 32 connections among the nodes. In each node, data are generated according to a regression model, using the same vector $\boldsymbol{\theta}_o \in \mathbb{R}^{60}$. The latter was randomly generated via a normal $\mathcal{N}(0,1)$. The input vectors were i.i.d. generated according to the normal $\mathcal{N}(0,1)$. The noise level at each node varied between 20 and 25 dBs. The parameters for the algorithms were chosen for optimized performance (after experimentation) and for similar convergence rate. For the LMS, $\mu = 0.035$ and for the APSM $\epsilon = \sqrt{2}\sigma$, $q = 20$, $\mu_k(n) = 0.2M_k(n)$. The combination weights were chosen according to the Metropolis rule and the data combination matrix was the identity one (no observations are exchanged). Figure 8.27 shows the benefits of the data reuse offered by the APSM. The curves show the mean-square deviation (MSD$= \frac{1}{K}\sum_{k=1}^{K} ||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$) as a function of the number of iterations.

8.10 OPTIMIZING NONSMOOTH CONVEX COST FUNCTIONS

Estimating parameters via the use of convex loss functions in the presence of a set of constraints is an established and well-researched field in optimization, with numerous applications in a wide range of disciplines. The mainstream of the methods follow either the Lagrange multipliers’ philosophy [10, 14] or the rationale behind the so-called *interior point* methods [14, 85]. In this section, we will focus on an alternative path and consider iterative schemes, which can be considered as the generalization of the gradient descent method, discussed in Chapter 5. The reason is that such techniques give rise to variants that scale well with the dimensionality and have inspired a number of algorithms, which have

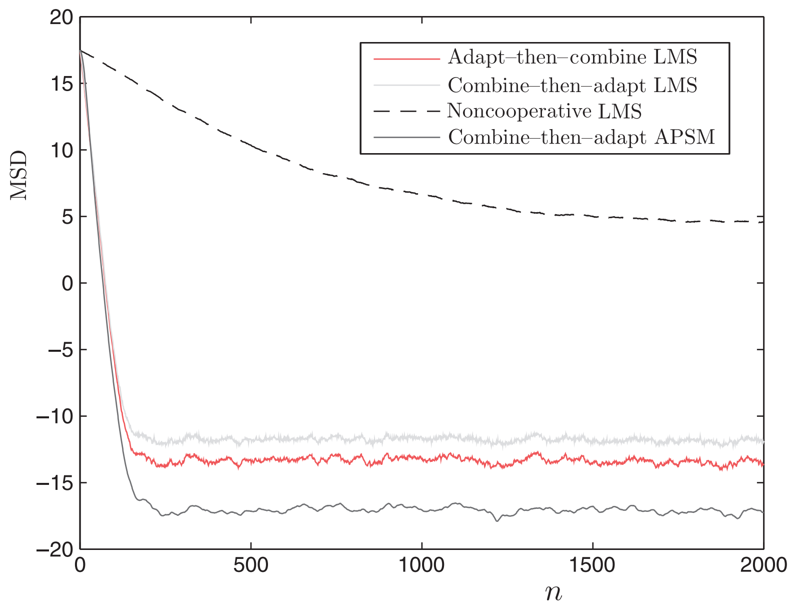


FIGURE 8.27

The MSD as a function of the number of iterations. The improved performance due to the data reuse offered by the diffusion ASPM is readily observed. Moreover, observe the significant performance improvement offered by all cooperation schemes, compared to the noncooperative LMS; for the latter, only one node is used.

been suggested for online learning within the machine learning and signal processing communities. Later on, we will move to more advanced techniques that build on the so-called *operator/mapping* and *fixed point* theoretic framework.

Although the stage of our discussion will be that of Euclidean spaces, $\mathbb{R}^l$, everything that will be said can be generalized to infinite dimensional Hilbert spaces; we will consider such cases in Chapter 11.

8.10.1 SUBGRADIENTS AND SUBDIFFERENTIALS

We have already met the first order convexity condition in (8.3) and it was shown that this is a sufficient and necessary condition for convexity, provided, of course, that the gradient exists. The condition basically states that the graph of the convex function lies above the hyperplanes, which are tangent at any point, $(\boldsymbol{x}, f(\boldsymbol{x}))$, that lies on this graph.

Let us now move a step forward and assume a function

$$f : \mathcal{X} \subseteq \mathbb{R}^l \longmapsto \mathbb{R},$$

to be convex, continuous but *nonsmooth*. This means that there are points where the gradient is not defined. Our goal now becomes that of generalizing the notion of gradient, for the case of convex functions.

Definition 8.7. A vector $\boldsymbol{g} \in \mathbb{R}^l$ is said to be the *subgradient* of a convex function, f, at a point, $\boldsymbol{x} \in \mathcal{X}$, if the following is true

$$\boxed{f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}), \quad \forall \boldsymbol{y} \in \mathcal{X}: \qquad \text{Subgradient.}} \tag{8.46}$$

It turns out that this vector is *not* unique. All the subgradients of a (convex) function at a point comprise a set.

Definition 8.8. The *subdifferential* of a convex function, f, at $\boldsymbol{x} \in \mathcal{X}$, denoted as $\partial f(\boldsymbol{x})$, is defined as the set

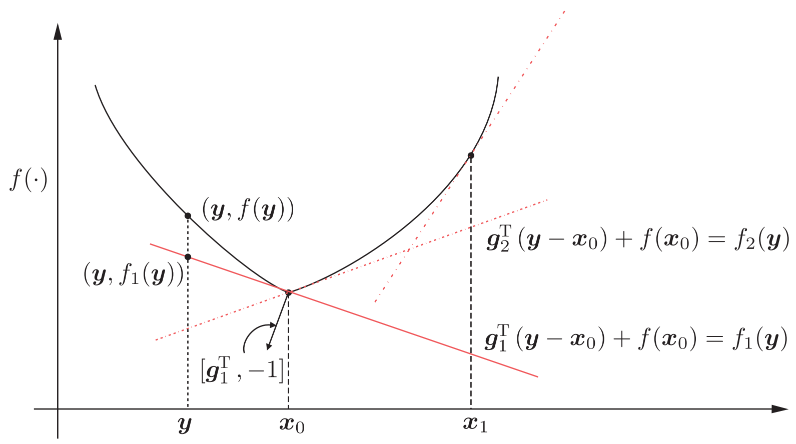
$$\boxed{\partial f(\boldsymbol{x}) := \left\{ \boldsymbol{g} \in \mathbb{R}^l : f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}),\ \forall \boldsymbol{y} \in \mathcal{X} \right\} : \quad \text{Subdifferential.}} \tag{8.47}$$

If f is differentiable at a point $\boldsymbol{x}$, then $\partial f(\boldsymbol{x})$ becomes a singleton, that is,

$$\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}.$$

Note that if $f(\boldsymbol{x})$ is convex, then the set $\partial f(\boldsymbol{x})$ is *nonempty and convex*. Moreover, $f(\boldsymbol{x})$ is differentiable at a point, $\boldsymbol{x}$, *if and only if* it has a unique subgradient [10]. From now on, we will denote a subgradient of f at a point, $\boldsymbol{x}$, as $f'(\boldsymbol{x})$.

Figure 8.28 gives a geometric interpretation of the notion of the subgradient. Each one of the subgradients at the point $\boldsymbol{x}_0$ defines a hyperplane that *supports* the graph of f. At $\boldsymbol{x}_0$, there is an infinity of subgradients, which comprise the subdifferential (set) at $\boldsymbol{x}_0$. At $\boldsymbol{x}_1$, the function is differentiable and there is a *unique* subgradient that coincides with the gradient at $\boldsymbol{x}_1$.



$f(\cdot)$
$(\boldsymbol{y}, f(\boldsymbol{y}))$
$(\boldsymbol{y}, f_1(\boldsymbol{y}))$
$\boldsymbol{g}_2^T(\boldsymbol{y} - \boldsymbol{x}_0) + f(\boldsymbol{x}_0) = f_2(\boldsymbol{y})$
$\boldsymbol{g}_1^T(\boldsymbol{y} - \boldsymbol{x}_0) + f(\boldsymbol{x}_0) = f_1(\boldsymbol{y})$
$[\boldsymbol{g}_1^T, -1]$
$\boldsymbol{y}$
$\boldsymbol{x}_0$
$\boldsymbol{x}_1$

FIGURE 8.28

At $\boldsymbol{x}_0$, there is an infinity of subgradients, each one defining a hyperplane in the extended $(\boldsymbol{x}, f(\boldsymbol{x}))$ space. All these hyperplanes pass through the point $(\boldsymbol{x}_0, f(\boldsymbol{x}_0))$ and support the graph of $f(\cdot)$. At the point $\boldsymbol{x}_1$, there is a unique subgradient that coincides with the gradient and defines the unique tangent hyperplane at the respective point of the graph.

Example 8.4. Let $x \in \mathbb{R}$ and

$$f(x) = |x|.$$

Then, show that

$$\partial f(x) = \begin{cases} \mathrm{sgn}(x), & \text{if } x \neq 0, \\ g \in [-1, 1], & \text{if } x = 0, \end{cases}$$

where sgn(·) is the sign function, being equal to 1 if its argument is positive and -1 if the argument is negative.

Indeed, if $x > 0$, then

$$g = \frac{\mathrm{d}x}{\mathrm{d}x} = 1,$$

and similarly $g = -1$, if $x < 0$. For $x = 0$, any $g \in [-1, 1]$ satisfies

$$g(y - 0) + 0 = gy \leq |y|,$$

and it is a subgradient. This is illustrated in Figure 8.29.

Lemma 8.2. *Given a convex function $f : \mathcal{X} \subseteq \mathbb{R}^l \longmapsto \mathbb{R}$, a point $\boldsymbol{x}_* \in \mathcal{X}$ is a minimizer of f, if and only if the zero vector belongs to its subdifferential set, that is,*

$$\boxed{\mathbf{0} \in \partial f(\boldsymbol{x}_*) : \quad \text{Condition for a Minimizer.}} \tag{8.48}$$

Proof. The proof is straightforward from the definition of a subgradient. Indeed, assume that $\mathbf{0} \in \partial f(\boldsymbol{x}_*)$. Then, the following is valid

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}_*) + \mathbf{0}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_*), \ \forall \boldsymbol{y} \in \mathcal{X}$$

and $\boldsymbol{x}_*$ is a minimizer. If now $\boldsymbol{x}_*$ is a minimizer, then we have that

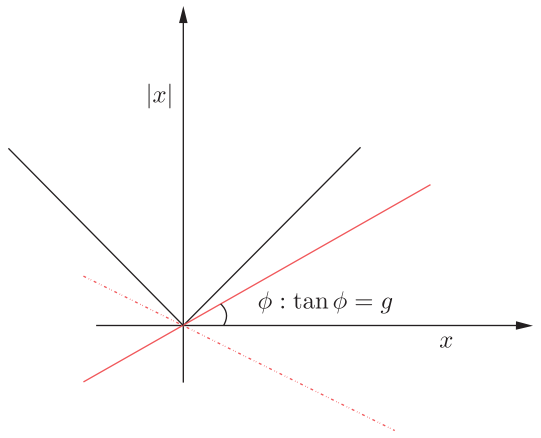


FIGURE 8.29

All lines with slope in $\in [-1, 1]$ comprise the subdifferential at $x = 0$.

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}_*) = f(\boldsymbol{x}_*) + \boldsymbol{0}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_*),$$

hence $\boldsymbol{0} \in \partial f(\boldsymbol{x}_*)$. □

Example 8.5. Let the metric *distance* function

$$d_C(\boldsymbol{x}) := \min_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\|.$$

This is the Euclidean distance of a point from its projection on a closed convex set, C, as defined in Section 8.3. Then show (Problem 8.24) that the subdifferential is given by

$$\partial d_C(\boldsymbol{x}) = \begin{cases} \frac{\boldsymbol{x} - P_C(\boldsymbol{x})}{\|\boldsymbol{x} - P_C(\boldsymbol{x})\|}, & \boldsymbol{x} \notin C, \\ N_C(\boldsymbol{x}) \cap B[0,1], & \boldsymbol{x} \in C, \end{cases} \tag{8.49}$$

where

$$N_C(\boldsymbol{x}) := \left\{ \boldsymbol{g} \in \mathbb{R}^l : \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) \leq 0, \forall \boldsymbol{y} \in C \right\},$$

and

$$B[0,1] := \left\{ \boldsymbol{x} \in \mathbb{R}^l : \|\boldsymbol{x}\| \leq 1 \right\}.$$

Moreover, if $\boldsymbol{x}$ is an *interior* point of C, then

$$\partial d_C(\boldsymbol{x}) = \{\boldsymbol{0}\}.$$

Observe that for all points $\boldsymbol{x} \notin C$ as well as for all interior points of C, the subgradient is a singleton, which means that $d_C(\boldsymbol{x})$ is differentiable. Recall that the function $d_C(\cdot)$ is nonnegative, convex, and continuous [43].

Note that (8.49) is also generalized to infinite dimensional Hilbert spaces.

8.10.2 MINIMIZING NONSMOOTH CONTINUOUS CONVEX LOSS FUNCTIONS: THE BATCH LEARNING CASE

Let J be a cost function[7]

$$J : \mathbb{R}^l \longmapsto [0, +\infty),$$

and C a closed convex set, $C \subseteq \mathbb{R}^l$. Our task is to compute a minimizer with respect to an unknown parameter vector, that is,

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),$$
$$\text{s.t.} \qquad \boldsymbol{\theta} \in C, \tag{8.50}$$

and we will assume that the set of solutions is *nonempty*. J is assumed to be convex, continuous, but not necessarily differentiable at all points. We have already seen examples of such loss function, such as the ϵ-insensitive linear function in (8.33) and the hinge one (8.37). The ℓ_1-norm function is another example, and it will be treated in Chapters 9 and 10.

[7] Recall what we have already said, that all the methods to be reported can be extended to general Hilbert spaces, $\mathbb{H}$.

The subgradient method

Our starting point is the simplest of the cases, where $C = \mathbb{R}^l$; that is, the minimizing task is unconstrained. The first thought that comes into mind is to consider the generalization of the gradient descent method, which was introduced in Chapter 5, and replace the gradient by the subgradient operation. The resulting scheme is known as the *subgradient algorithm* [71, 72].

Starting from an arbitrary estimate, $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^l$, the update recursions become

$$\boxed{\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i J'\left(\boldsymbol{\theta}^{(i-1)}\right): \quad \text{Subgradient Algorithm,}} \tag{8.51}$$

where $J'(\cdot)$ denotes *any* subgradient of the cost function, and μ_i is a step-size sequence judicially chosen so that convergence is guaranteed. In spite of the similarity in the appearance with our familiar gradient descent scheme, there are some major differences. The reader may have noticed that the new algorithm was not called subgradient "descent." This is because the update in (8.51) is not necessarily performed in the descent direction. Thus, during the operation of the algorithm, the value of the cost function may increase. Recall that in the gradient descent methods, the value of the cost function is guaranteed to decrease with each iteration step, which also led to a linear convergence rate, as we have pointed out in Chapter 5.

In contrast here, concerning the subgradient method, such comments cannot be stated. To establish convergence, a different route has to be adopted. To this end, let us define

$$J_*^{(i)} := \min\left\{J(\boldsymbol{\theta}^{(i)}), J(\boldsymbol{\theta}^{(i-1)}), \ldots, J(\boldsymbol{\theta}^{(0)})\right\}, \tag{8.52}$$

which can also be recursively obtained by

$$J_*^{(i)} = \min\left\{J_*^{(i-1)}, J(\boldsymbol{\theta}^{(i)})\right\}.$$

Then the following holds true.

Proposition 8.3. *Let J be a convex cost function. Assume that the subgradients at all points are bounded, that is,*

$$||J'(\boldsymbol{x})|| \leq G, \ \forall \boldsymbol{x} \in \mathbb{R}^l.$$

Let us also assume that the step-size sequence be a diminishing one, such as

$$\sum_{i=1}^{\infty} \mu_i = \infty, \quad \sum_{i=1}^{\infty} \mu_i^2 < \infty.$$

Then

$$\lim_{i \longrightarrow \infty} J_*^{(i)} = J(\boldsymbol{\theta}_*),$$

where $\boldsymbol{\theta}_*$ is a minimizer, assuming that the set of minimizers is not empty.

Proof. We have that

$$\begin{aligned} ||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*||^2 &= ||\boldsymbol{\theta}^{(i-1)} - \mu_i J'(\boldsymbol{\theta}^{(i-1)}) - \boldsymbol{\theta}_*||^2 \\ &= ||\boldsymbol{\theta}^{(i-1)} - \boldsymbol{\theta}_*||^2 - 2\mu_i J'^T(\boldsymbol{\theta}^{(i-1)})(\boldsymbol{\theta}^{(i-1)} - \boldsymbol{\theta}_*) \\ &\quad + \mu_i^2 ||J'(\boldsymbol{\theta}^{(i-1)})||^2. \end{aligned} \tag{8.53}$$

By the definition of the subgradient, we have

$$J(\boldsymbol{\theta}_*) - J(\boldsymbol{\theta}^{(i-1)}) \geq J'^T(\boldsymbol{\theta}^{(i-1)})(\boldsymbol{\theta}_* - \boldsymbol{\theta}^{(i-1)}). \tag{8.54}$$

Plugging (8.54) in (8.53) and after some algebraic manipulations, by applying the resulting inequality recursively (Problem 8.25), we finally obtain that

$$J_*^{(i)} - J(\boldsymbol{\theta}_*) \leq \frac{||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}_*||^2}{2\sum_{k=1}^{i}\mu_k} + \frac{\sum_{k=1}^{i}\mu_k^2}{2\sum_{k=1}^{i}\mu_k}G^2. \tag{8.55}$$

Leaving i to grow to infinity and taking into account the assumptions, the claim is proved. □

There are a number of variants of this proof. Also, other choices for the diminishing sequence can also guarantee convergence, such as $\mu_i = 1/\sqrt{i}$. Moreover, in certain cases, some of the assumptions may be relaxed. Note that the assumption of the subgradient being bounded is guaranteed, if J is γ-Lipschitz continuous (Problem 8.26), that is, there is $\gamma > 0$, such as

$$|J(\boldsymbol{y}) - J(\boldsymbol{x})| \leq \gamma||\boldsymbol{y} - \boldsymbol{x}||, \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^l.$$

Interpreting the proposition from a slightly different angle, we can say that the algorithm generates a *subsequence* of estimates, $\boldsymbol{\theta}_{i_*}$, which corresponds to the values of J_*, shown as, $J(\boldsymbol{\theta}_{i_*}) \leq J(\boldsymbol{\theta}_i), \ i \leq i_*$, which converges to $\boldsymbol{\theta}_*$. The *best possible* convergence rate that may be achieved is of the order of $\mathcal{O}(\frac{1}{\sqrt{i}})$, if one optimizes the bound in (8.55), with respect to μ_k, [61], which can be obtained if $\mu_i = \frac{c}{\sqrt{i}}$, where c is a constant. In any case, it is readily noticed that the convergence speed of such methods is rather slow. Yet due to their computational simplicity, they are still in use, especially in cases where the number of data samples is large. The interested reader can obtain more on the subgradient method from [9, 72].

Example 8.6 (The perceptron algorithm)**.** Recall the hinge loss function with $\rho = 0$, defined in (8.37),

$$\mathcal{L}(y, \boldsymbol{\theta}^T\boldsymbol{x}) = \max\left(0, -y\boldsymbol{\theta}^T\boldsymbol{x}\right).$$

In a two-class classification task, we are given a set of training samples, $(y_n, \boldsymbol{x}_n) \in \{-1, +1\} \times \mathbb{R}^{l+1}$, $n = 1, 2, \ldots, N$, and the goal is to compute a linear classifier to minimize the empirical loss function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N}\mathcal{L}(y_n, \boldsymbol{\theta}^T\boldsymbol{x}_n). \tag{8.56}$$

We will assume the classes to be *linearly separable*, which guarantees that there is a solution; that is, there exists a hyperplane that classifies correctly all data points. Obviously, such a hyperplane will score a zero for the cost function in (8.56). We have assumed that the dimension of our input data space has been increased by one, to account for the bias term for hyperplanes not crossing the origin.

The subdifferential of the hinge loss function is easily checked out to be (e.g., use geometric arguments, which relate a subgradient with a support hyperplane of the respective function graph),

$$\partial\mathcal{L}(y_n, \boldsymbol{\theta}^T\boldsymbol{x}_n) = \begin{cases} 0, & y_n\boldsymbol{\theta}^T\boldsymbol{x}_n > 0, \\ -y_n\boldsymbol{x}_n, & y_n\boldsymbol{\theta}^T\boldsymbol{x}_n < 0, \\ \boldsymbol{g} \in [-y_n\boldsymbol{x}_n, 0], & y_n\boldsymbol{\theta}^T\boldsymbol{x}_n = 0. \end{cases} \tag{8.57}$$

We choose to work with the following subgradient

$$\mathcal{L}'(y_n, \boldsymbol{\theta}^T\boldsymbol{x}_n) = -y_n\boldsymbol{x}_n\chi_{(-\infty,0]}\left(y_n\boldsymbol{\theta}^T\boldsymbol{x}_n\right), \tag{8.58}$$

where $\chi_A(\tau)$ is the characteristic function, defined as

$$\chi_A(\tau) = \begin{cases} 1, & \tau \in A, \\ 0, & \tau \notin A. \end{cases} \tag{8.59}$$

The subgradient algorithm now becomes,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \sum_{n=1}^{N} y_n \boldsymbol{x}_n \chi_{(-\infty,0]}\left(y_n \boldsymbol{\theta}^{(i-1)T} \boldsymbol{x}_n\right). \tag{8.60}$$

This is the celebrated *perceptron algorithm*, which we are going to see in more detail in Chapter 18. Basically, what the algorithm in (8.60) says is the following. Starting from an arbitrary $\boldsymbol{\theta}^{(0)}$, test all training vectors with $\boldsymbol{\theta}^{(i-1)}$. Select all those vectors that fail to predict the correct class (for the correct class, $y_n \boldsymbol{\theta}^{(i-1)T} \boldsymbol{x}_n > 0$), and update the current estimate toward the direction of the weighted (by the corresponding label) average of the *misclassified* patterns. It turns out that the algorithm converges in a *finite* number of steps, even if the step-size sequence is not a diminishing one. This is what it was said before, that in certain cases, convergence of the subgradient algorithm is guaranteed even if some of the assumptions in Proposition 8.3 do not hold.

The generic projected subgradient scheme

The generic scheme on which a number of variants draw their origin is summarized as follows. Select $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^l$, arbitrarily. Then the iterative scheme

$$\boxed{\boldsymbol{\theta}^{(i)} = P_C\left(\boldsymbol{\theta}^{(i-1)} - \mu_i J'(\boldsymbol{\theta}^{(i-1)})\right): \qquad \text{GPS Scheme,}} \tag{8.61}$$

where $J'(\cdot)$ denotes a respective subgradient and P_C is the projection operator onto C, converges (converges weakly in the more general case) to a solution of the constrained task in (8.50). The sequence of nonnegative real numbers, μ_i, is judicially selected. It is readily seen that this scheme is a generalization of the gradient descent scheme, discussed in Chapter 5, if we set $C = \mathbb{R}^l$ and J is differentiable.

The projected gradient method (PGM)

This method is a special case of (8.61), if J is *smooth* and we set $\mu_i = \mu$. That is,

$$\boxed{\boldsymbol{\theta}^{(i)} = P_C\left(\boldsymbol{\theta}^{(i-1)} - \mu \nabla J(\boldsymbol{\theta}^{(i-1)})\right): \qquad \text{PGM Scheme.}} \tag{8.62}$$

It turns out that if the gradient is γ-Lipschitz, that is,

$$\|\nabla J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{h})\| \leq \gamma \|\boldsymbol{\theta} - \boldsymbol{h}\|, \; \gamma > 0, \; \forall \boldsymbol{\theta}, \boldsymbol{h} \in \mathbb{R}^l,$$

and

$$\mu \in \left(0, \frac{2}{\gamma}\right),$$

then starting from an arbitrary point $\boldsymbol{\theta}^{(0)}$, the sequence in (8.62) converges (weakly in a general Hilbert space) to a solution of (8.50) [40, 51].

Example 8.7. *Projected Landweber Method*:
Let our optimization task be

$$\begin{aligned}&\text{minimize } \frac{1}{2}\|\boldsymbol{y}-X\boldsymbol{\theta}\|^2,\\ &\text{subject to } \boldsymbol{\theta}\in C.\end{aligned}$$

where $X\in\mathbb{R}^{m\times l}$, $\boldsymbol{y}\in\mathbb{R}^m$. Expanding and taking the gradient we get,

$$\begin{aligned}J(\boldsymbol{\theta}) &= \frac{1}{2}\boldsymbol{\theta}^T X^T X\boldsymbol{\theta} - \boldsymbol{y}^T X\boldsymbol{\theta} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y},\\ \nabla J(\boldsymbol{\theta}) &= X^T X\boldsymbol{\theta} - X^T\boldsymbol{y}.\end{aligned}$$

First we check that $\nabla J(\boldsymbol{\theta})$ is γ-Lipschitz. To this end, we have

$$\|X^T X(\boldsymbol{\theta}-\boldsymbol{h})\| \le \|X^T X\|\|\boldsymbol{\theta}-\boldsymbol{h}\| \le \lambda_{\max}\|\boldsymbol{\theta}-\boldsymbol{h}\|,$$

where the spectral norm of a matrix has been used (Section 6.4) and $\lambda_{\max}$ denotes the maximum eigenvalue $X^T X$. Thus, if

$$\mu\in\left(0,\frac{2}{\lambda_{\max}}\right)$$

the corresponding iterations in (8.62) converge to a solution of (8.50). The scheme has been used in the context of compressed sensing where (as we will see in Chapter 10) the task of interest is

$$\begin{aligned}&\text{minimize } \frac{1}{2}\|\boldsymbol{y}-X\boldsymbol{\theta}\|^2,\\ &\text{subject to } \|\boldsymbol{\theta}\|_1\le\rho.\end{aligned}$$

Then, it turns out that projecting on the ℓ_1-ball (corresponding to C) is equivalent to a soft thresholding operation[8] [35]. A variant occurs, if a projection on a weighted ℓ_1 ball is used, to speed up convergence (Chapter 10). Projection on a weighted ℓ_1 ball has been developed in [46], via fully geometric arguments, and it also results in soft-thresholding operations.

Projected subgradient method

Starting from an arbitrary point $\boldsymbol{\theta}^{(0)}$, then for the following recursion [2, 54],

$$\boxed{\boldsymbol{\theta}^{(i)} = P_C\left(\boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{\max\{1,\|J'(\boldsymbol{\theta}^{(i-1)})\|\}}J'\left(\boldsymbol{\theta}^{(i-1)}\right)\right) : \quad \text{PSMa,}} \tag{8.63}$$

- either a solution of (8.50) is achieved in a finite number of steps,
- or the iterations converge (weakly in the general case) to a point in the set of solutions of (8.50),

provided that

$$\sum_{i=1}^{\infty}\mu_i=\infty,\quad \sum_{i=1}^{\infty}\mu_i^2<\infty.$$

Another version of the projected subgradient algorithm was presented in [67]. Let $J_* = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ be the minimum (strictly speaking the infimum) of a cost function, whose set of minimizers is assumed to be nonempty. Then, the following iterative algorithm,

[8] See Chapter 10 and Example 8.10.

$$\boxed{\boldsymbol{\theta}^{(i)} = \begin{cases} P_C\left(\boldsymbol{\theta}^{(i-1)} - \mu_i \frac{J(\boldsymbol{\theta}^{(i-1)})-J_*}{\|J'(\boldsymbol{\theta}^{(i-1)})\|^2} J'(\boldsymbol{\theta}^{(i-1)})\right), & \text{if } J'(\boldsymbol{\theta}^{(i-1)}) \neq \mathbf{0}, \\ P_C(\boldsymbol{\theta}^{(i-1)}), & \text{if } J'(\boldsymbol{\theta}^{(i-1)}) = \mathbf{0}, \end{cases} \quad \text{PSMb}} \tag{8.64}$$

converges (weakly in infinite dimensional spaces) for $\mu_i \in (0, 2)$ and under some general conditions and assuming that the subgradient is bounded. The proof is a bit technical and the interested reader can obtain it from, for example, [67, 79].

Needless to say that, besides the previously reported major schemes discussed, there is a number of variants; for a related review see [79].

8.10.3 ONLINE LEARNING FOR CONVEX OPTIMIZATION

Online learning in the framework of the squared error loss function has been the focus in Chapters 5 and 6. One of the reasons that online learning was introduced was to give the potential to the algorithm to track time variations in the underlying statistics. Another reason was to cope with the unknown statistics when the cost function involved expectations, in the context of the stochastic approximation theory. Moreover, online algorithms are of particular interest when the number of the available training data as well as the dimensionality of the input space become very large, compared to the load that today's storage, processing, and networking devices can cope with. Exchanging information has now become cheap and databases have been populated with a massive number of data. This has rendered batch processing techniques, for learning tasks with huge datasets, impractical. Online algorithms that process one data point at a time have now become an indispensable algorithmic tool.

Recall from Section 3.14 that the ultimate goal of a machine learning task, given a loss function $\mathcal{L}$, is to minimize the expected loss/risk, which in the context of parametric modeling can be written as

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))\right] \\ &:= \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\right]. \end{aligned} \tag{8.65}$$

Instead, the corresponding empirical loss function is minimized, given a set of N training points,

$$J_N(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}(\boldsymbol{\theta}, y_n, \boldsymbol{x}_n). \tag{8.66}$$

In this context, the subgradient scheme would take the form

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{N}\sum_{n=1}^{N} \mathcal{L}_n'(\boldsymbol{\theta}^{(i-1)}),$$

where for notational simplicity we used

$$\mathcal{L}_n(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}, y_n, \boldsymbol{x}_n). \tag{8.67}$$

Thus, at each iteration, one has to compute N subgradient values, which for large values of N is computationally cumbersome. One way out is to adopt stochastic approximation arguments, as explained in Chapter 5, and come with a corresponding online version,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \mathcal{L}_n'(\boldsymbol{\theta}_{n-1}), \tag{8.68}$$

where now the iteration index, i, coincides with the time index, n. There are two different ways to view (8.68). Either n takes values in the interval $[1, N]$, and one cycles periodically until convergence, or n is left to grow unbounded. The latter is very natural for very large values of N, and we focus on this scenario from now on. Moreover, such strategy can cope with slow time variations, if this is the case. Note that in the online formulation, at each time instant a *different* loss function is involved and our task becomes that of an *asymptotic* minimization. Furthermore, one has to study the asymptotic *convergence properties*, as well as the respected *convergence conditions*. Soon, we are going to introduce a relatively recent tool, for analyzing the performance of online algorithms, namely, the *regret analysis*.

It turns out that for each one of the optimization schemes discussed in Subsection 8.10.2, we can write its online version. Given the sequence of loss functions, $\mathcal{L}_n,\ n = 1, 2 \ldots$, the online version of the generic projected subgradient scheme becomes

$$\boldsymbol{\theta}_n = P_C\left(\boldsymbol{\theta}_{n-1} - \mu_n \mathcal{L}_n'(\boldsymbol{\theta}_{n-1})\right), \quad n = 1, 2, 3, \ldots \tag{8.69}$$

In a more general setting, the constraint-related convex sets can be left to be time varying, too; in other words, we can write C_n. For example, such schemes with time-varying constraints have been developed in the context of sparsity-aware learning, where in place of the ℓ_1-ball, a weighted ℓ_1-ball is being used [46]. This has a really drastic effect in speeding up the convergence of the algorithm, see Chapter 10.

Another example is the so-called *adaptive gradient* (ADAGRAD) algorithm [38]. The projection operator is defined in a more general context, in terms of the Mahalanobis distance, that is,

$$P_C^G(\boldsymbol{x}) = \min_{z \in C} (\boldsymbol{x} - \boldsymbol{z})^{\mathrm{T}} G(\boldsymbol{x} - \boldsymbol{z}), \quad \forall \boldsymbol{x} \in \mathbb{R}^l. \tag{8.70}$$

In place of G, the square root of the average outer product of the computed subgradients is used, that is,

$$G_n = \left(\frac{1}{n} \sum_{k=1}^{n} \boldsymbol{g}_k \boldsymbol{g}_k^{\mathrm{T}} \right)^{1/2},$$

where, $\boldsymbol{g}_k = \mathcal{L}_k'(\boldsymbol{\theta}_{k-1})$ denotes the subgradient at time instant k. Also, the same matrix is used to weigh the gradient correction and the scheme has the form,

$$\boldsymbol{\theta}_n = P_C^{G_n}\left(\boldsymbol{\theta}_{n-1} - \mu_n G_n^{-1} \boldsymbol{g}_n\right).$$

The use of the (time-varying) weighting matrix accounts for the geometry of the data observed in earlier iterations, which leads to a more informative gradient-based learning. For the sake of computational savings, the structure of G_n is taken to be diagonal. Different algorithmic settings are discussed in [38], alongside the study of the converging properties of the algorithm.

Example 8.8. *The LMS Algorithm*:Let us assume that

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{2}\left(y_n - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n\right)^2,$$

and also set $C = \mathbb{R}^l$ and $\mu_n = \mu$. Then (8.69) becomes our familiar LMS recursion,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu(y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}} \boldsymbol{x}_n)\boldsymbol{x}_n,$$

whose convergence properties have been discussed in Chapter 5.

The PEGASOS algorithm

The *primal estimated subgradient solver for SVM* (PEGASOS) algorithm is an online scheme built around the hinge loss function *regularized* by the squared Euclidean norm of the parameters vector, [70]. From this point of view, it is an instance of the online version of the projected subgradient algorithm. This algorithm results if we set in (8.69),

$$\mathcal{L}_n(\boldsymbol{\theta}) = \max(0, 1 - y_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2, \tag{8.71}$$

where in this case, ρ in the hinge loss function has been set equal to one. The associated empirical cost function is

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\max\left(0, 1 - y_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n\right) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2, \tag{8.72}$$

whose minimization results in the celebrated *support vector machine* (SVM). Note that the only differences with the perceptron algorithm is the presence of the regularizer and the nonzero value of ρ. These seemingly minor differences have important implications in practice, and we are going to say more on this in Chapter 11, where nonlinear extensions treated in the more general context of Hilbert spaces will be considered.

The subgradient adopted by the PEGASOS is

$$\mathcal{L}_n'(\boldsymbol{\theta}) = \lambda\boldsymbol{\theta} - y_n\boldsymbol{x}_n\chi_{(-\infty,0]}(y_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n - 1). \tag{8.73}$$

The step-size is chosen as $\mu_n = \frac{1}{\lambda n}$. Furthermore in its more general formulation, at each iteration step, an (optional) projection on the $\frac{1}{\sqrt{\lambda}}$ length ℓ_2 ball, $B[\mathbf{0}, \frac{1}{\sqrt{\lambda}}]$, is performed. The update recursion then becomes,

$$\boldsymbol{\theta}_n = P_{B[\mathbf{0},\frac{1}{\sqrt{\lambda}}]}\Big(\big(1 - \mu_n\lambda\big)\boldsymbol{\theta}_{n-1} + \mu_n y_n\boldsymbol{x}_n\chi_{(-\infty,0]}\big(y_n\boldsymbol{\theta}_{n-1}^T\boldsymbol{x}_n - 1\big)\Big), \tag{8.74}$$

where $P_{B[\mathbf{0},\frac{1}{\sqrt{\lambda}}]}$ is the projection on the respective ℓ_2 ball given in (8.14). In (8.74), note that the effect of the regularization is to smooth out the contribution of $\boldsymbol{\theta}_{n-1}$. A variant of the algorithm for fixed number of points, N, suggests to average out a number of m subgradient values in an index set, $A_n \subseteq \{1, 2, \ldots, N\}$, such as $k \in A_n$ if $y_k\boldsymbol{\theta}_{n-1}^T\boldsymbol{x}_k < 1$. Different scenarios for the choice of the m indices can be employed, with the random one being a possibility. The scheme is summarized in Algorithm 8.4.

Algorithm 8.4 (The PEGASOS algorithm).

```
• Initialization
    • Select $\boldsymbol{\theta}^{(0)}$; Usually set to zero.
    • Select $\lambda$
    • Select $m$; Number of subgradient values to be averaged.
• For $n = 1, 2, \ldots, N$, Do
    • Select $A_n \subseteq \{1, 2, \ldots, N\}$: $|A_n| = m$, uniformly at random.
    • $\mu_n = \frac{1}{\lambda n}$
    • $\boldsymbol{\theta}_n = \big(1 - \mu_n\lambda\big)\boldsymbol{\theta}_{n-1} + \frac{\mu_n}{m}\sum_{k\in A_n} y_k\boldsymbol{x}_k$
    • $\boldsymbol{\theta}_n = \min\left(1, \frac{1}{\sqrt{\lambda}||\boldsymbol{\theta}_n||}\right)\boldsymbol{\theta}_n$; Optional.
• End For
```

Application of regret analysis arguments point out the required number of iterations for obtaining a solution of accuracy ϵ is $\mathcal{O}(1/\epsilon)$, when each iteration operates on a single training sample. The algorithm is very similar with the algorithms proposed in [44, 105]. The difference being in the choice of the step-size. We will come back to these algorithms in Chapter 11. There, we are going to see that the online learning in infinite dimensional spaces is more tricky. In [70], a number of comparative tests against well-established SVM algorithms have been performed using standard datasets. The main advantage of the algorithm is its computational simplicity, and it achieves comparable performance rates at lower computational costs.

Remarks 8.8.

- *The APSM revisited*: It can be seen that the APSM algorithm can be re-derived in a more general setting as an online version of the PSGb in (8.64), which also justifies its name. It suffices to consider as a loss function the weighted average of the distances of a point $\boldsymbol{\theta}$ from the q most recently formed, by the data, property sets, Problem 8.29.

  Moreover, such a derivation paves the way for employing the algorithm even if the projection onto the constraint set is not analytically available. It turns out that the mapping

$$T(\boldsymbol{\theta}) = \boldsymbol{\theta} - \mu \frac{\mathcal{L}_n(\boldsymbol{\theta})}{||\mathcal{L}'_n(\boldsymbol{\theta})||^2} \mathcal{L}'_n(\boldsymbol{\theta}),$$

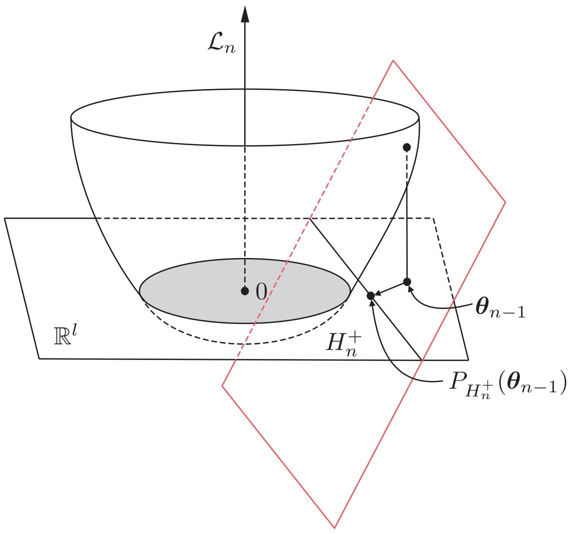
  maps $\boldsymbol{\theta}$ onto the hyperplane, H, which (in the extended space $\mathbb{R}^{l+1}$) is the intersection of $\mathbb{R}^l$ with the hyperplane that is defined by the respective subgradient of the loss function at $\boldsymbol{\theta}$. This is a *separating* hyperplane, that separates $\boldsymbol{\theta}$ from the zero level set of $\mathcal{L}_n$. Hence, after the mapping, $\boldsymbol{\theta}$ is mapped to a point closer to this zero level set.

  It can be shown that the algorithm monotonically converges into the intersection of all the zero level sets of $\mathcal{L}_n$, $n = 1, 2, \ldots$, assuming that it is nonempty and the problem is feasible. This version of the APSM does not need the projection onto the property convex sets to be given analytically. In the case the loss functions are chosen to be the ϵ-insensitive loss or the hinge loss, the algorithm results in the APSM of Algorithm 8.2. Figure 8.30 shows the geometry of the mapping for a quadratic loss function; note that in this case, the zero level set is an ellipse and the projection is not analytically defined. The interested reader can obtain more on these issues, as well as on full convergence proofs, from [79, 84, 98, 99].

8.11 REGRET ANALYSIS

A major effort when dealing with iterative learning algorithms is dedicated to the issue of convergence; where the algorithm converges, under which conditions it converges and how fast it converges to its steady-state. A large part of Chapter 5 was focused on the convergence properties of the LMS. Furthermore, in the current chapter, when we discussed the various subgradient-based algorithms, convergence properties were also reported.

In general, analyzing the convergence properties of online algorithms tends to be quite a formidable task and classical approaches have to adopt a number of assumptions, sometimes rather strong. Typical assumptions refer to the statistical nature of the data (e.g., being i.i.d. or the noise being white), and/or that the true model, which generates the data, is assumed to be known, and/or that the algorithm has reached a region in the parameter's space that is close to a minimizer.



$\mathcal{L}_n$
0
$\mathbb{R}^l$
H_n^+
$\boldsymbol{\theta}_{n-1}$
$P_{H_n^+}(\boldsymbol{\theta}_{n-1})$

FIGURE 8.30

At every time instant, the subgradient of the loss function $\mathcal{L}_n$ at $\boldsymbol{\theta}_{n-1}$, defines a hyperplane that intersects the input space. The intersection is a hyperplane, which separates $\boldsymbol{\theta}_{n-1}$ from the zero level set of $\mathcal{L}_n$. The APSM performs a projection on the halfspace that contains the zero level set and brings the update closer to it. For the case of ϵ-insensitive or the hinge loss functions, the separating hyperplane is a support hyperplane and the projection coincides with the projection on the respective hyperslab or halfspace, respectively.

More recently, an alternative methodology has been developed which bypasses the need for such assumptions. The methodology evolves around the concept of *cumulative loss*, which has already been introduced in Chapter 5, Section 5.5.2. The method is known as *regret analysis*, and its birth is due to developments in the interplay between game and learning theories; see, for example, [21].

Let us assume that the training samples, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots$, arrive sequentially and that an adopted online algorithm makes the corresponding predictions, $\hat{y}_n$. The quality of the prediction, for each time instant, is tested against a loss function, $\mathcal{L}(y_n, \hat{y}_n)$. The cumulative loss up to time N is given by

$$\mathcal{L}_{\text{cum}}(N) := \sum_{n=1}^{N} \mathcal{L}(y_n, \hat{y}_n). \tag{8.75}$$

Let f be a *fixed* predictor. Then the *regret* of the online algorithm relative to f, when running up to time instant N, is defined as

$$\boxed{\text{Regret}_N(f) := \sum_{n=1}^{N} \mathcal{L}(y_n, \hat{y}_n) - \sum_{n=1}^{N} \mathcal{L}\left(y_n, f(\boldsymbol{x}_n)\right) : \quad \text{Regret Relative to } f.} \tag{8.76}$$

The name regret is inherited from the game theory and it means how “sorry” the algorithm or the learner (in the ML jargon) is, in retrospect, not to have followed the prediction of the fixed predictor, f. The predictor f. is also known as the *hypothesis*. Also, if f is chosen from a set of functions, $\mathcal{F}$, this set is called the *hypothesis class*.

The regret relative to the family of functions, $\mathcal{F}$, when the algorithm runs over N time instants, is defined as

$$\text{Regret}_N(\mathcal{F}) := \max_{f\in\mathcal{F}} \text{Regret}_N(f). \tag{8.77}$$

In the context of regret analysis, the goal becomes that of designing an online learning rule so that the resulting regret with respect to an optimal fixed predictor to be small; that is, the regret associated with the learner should grow *sublinearly* (slower than linearly) with the number of iterations, N. Sublinear growth guarantees that the difference between the *average* loss suffered by the learner and the average loss of the optimal predictor will tend to zero asymptotically.

For the linear class of functions, we have

$$\hat{y}_n = \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n,$$

and the loss can be written as

$$\mathcal{L}(y_n, \hat{y}_n) = \mathcal{L}(y_n, \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n) := \mathcal{L}_n(\boldsymbol{\theta}_{n-1}).$$

Adapting (8.76) to the previous notation, we can write

$$\text{Regret}_N(\boldsymbol{h}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{h}), \tag{8.78}$$

where $\boldsymbol{h} \in C \subseteq \mathbb{R}^l$ is a *fixed* parameter vector in the set C where solutions are sought.

Before proceeding further, it is interesting to note that the cumulative loss is based on the loss suffered by the learner, against $y_n, \boldsymbol{x}_n$, using the estimate, $\boldsymbol{\theta}_{n-1}$, which has been trained on data up to and including time instant $n-1$. The pair $(y_n, \boldsymbol{x}_n)$ is not involved in its training. From this point of view, the cumulative loss is in line with our desire to guard against overfitting.

In the framework of regret analysis, the path to follow is to derive an upper bound for the regret, exploiting the convexity of the employed loss function. We will demonstrate the technique via a case study; that of the online version of the simple subgradient algorithm.

Regret analysis of the subgradient algorithm

The online version of (8.68), for minimizing the expected loss, $\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\right]$, is written as

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \boldsymbol{g}_n, \tag{8.79}$$

where for notational convenience, the subgradient is denoted as

$$\boldsymbol{g}_n := \mathcal{L}_n'(\boldsymbol{\theta}_{n-1}).$$

Proposition 8.4. *Assume that the subgradients of the loss function are bounded, shown as*

$$||\boldsymbol{g}_n|| \le G, \ \forall n. \tag{8.80}$$

Furthermore, assume that the set of solutions, $\mathcal{S}$, is bounded; that is, $\forall\ \boldsymbol{\theta},\ \boldsymbol{h} \in \mathcal{S}$, there exists a bound F, such that

$$||\boldsymbol{\theta} - \boldsymbol{h}|| \le F. \tag{8.81}$$

Let $\boldsymbol{\theta}_*$ be an optimal (desired) predictor. Then, if $\mu_n = \frac{1}{\sqrt{n}}$,

$$\boxed{\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{\theta}_{n-1}) \leq \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{\theta}_*) + \frac{F^2}{2\sqrt{N}} + \frac{G^2}{\sqrt{N}}.} \tag{8.82}$$

In words, as $N \longrightarrow \infty$ the average cumulative loss tends to the average loss of the optimal predictor.

Proof. Since the adopted loss function is assumed to be convex and by the definition of the subgradient, we have

$$\mathcal{L}_n(\boldsymbol{h}) \geq \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) + \boldsymbol{g}_n^T(\boldsymbol{h} - \boldsymbol{\theta}_{n-1}), \quad \forall \boldsymbol{h} \in \mathbb{R}^l, \tag{8.83}$$

or

$$\mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \mathcal{L}_n(\boldsymbol{h}) \leq \boldsymbol{g}_n^T(\boldsymbol{\theta}_{n-1} - \boldsymbol{h}). \tag{8.84}$$

However, recalling (8.79), we can write that

$$\boldsymbol{\theta}_n - \boldsymbol{h} = \boldsymbol{\theta}_{n-1} - \boldsymbol{h} - \mu_n \boldsymbol{g}_n, \tag{8.85}$$

which results in

$$\begin{aligned} ||\boldsymbol{\theta}_n - \boldsymbol{h}||^2 = ||\boldsymbol{\theta}_{n-1} - \boldsymbol{h}||^2 + \mu_n^2 ||\boldsymbol{g}_n||^2 \\ -2\mu_n \boldsymbol{g}_n^T(\boldsymbol{\theta}_{n-1} - \boldsymbol{h}). \end{aligned} \tag{8.86}$$

Taking into account the bound of the subgradient, Eq. (8.86) leads to the inequality,

$$\boldsymbol{g}_n^T(\boldsymbol{\theta}_{n-1} - \boldsymbol{h}) \leq \frac{1}{2\mu_n}\left(||\boldsymbol{\theta}_{n-1} - \boldsymbol{h}||^2 - ||\boldsymbol{\theta}_n - \boldsymbol{h}||^2\right) + \frac{\mu_n}{2}G^2. \tag{8.87}$$

Summing up both sides of (8.87), taking into account inequality (8.84) and after a bit of algebra (Problem 8.30) results in

$$\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{h}) \leq \frac{1}{2\mu_N}F^2 + \frac{G^2}{2}\sum_{n=1}^{N}\mu_n. \tag{8.88}$$

Setting $\mu_n = \frac{1}{\sqrt{n}}$, using the obvious bound

$$\sum_{n=1}^{N}\frac{1}{\sqrt{n}} \leq 1 + \int_1^N \frac{1}{\sqrt{t}}\,dt = 2\sqrt{N} - 1, \tag{8.89}$$

and dividing both sides of (8.88) by N, the proposition is proved for any $\boldsymbol{h}$. Hence, it will also be true for $\boldsymbol{\theta}_*$. □

The previous proof follows the one given in [106]; this was the first paper to adopt the notion of "regret" for the analysis of convex online algorithms. Proofs given later for more complex algorithms have borrowed, in one way or another, the arguments used there.

Remarks 8.9.

- Tighter regret bounds can be derived when the loss function is strongly convex, [42]. A function $f : \mathcal{X} \subseteq \mathbb{R}^l \longmapsto \mathbb{R}$ is said to be σ-*strongly convex*, if, $\forall\, \boldsymbol{y},\, \boldsymbol{x} \in \mathcal{X}$,

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}) + \frac{\sigma}{2}||\boldsymbol{y} - \boldsymbol{x}||^2, \tag{8.90}$$

for any subgradient $\boldsymbol{g}$ at $\boldsymbol{x}$. It also turns out that a function $f(\boldsymbol{x})$ is strongly convex if $f(\boldsymbol{x}) - \frac{\sigma}{2}||\boldsymbol{x}||^2$ is convex (Problem 8.31).

For σ-strongly convex loss functions, if the step-size of the subgradient algorithm is diminishing at a rate $\mathcal{O}(\frac{1}{\sigma n})$ then the average cumulative loss is approaching the average loss of the optimal predictor at a rate $\mathcal{O}(\frac{\ln N}{N})$ (Problem 8.32). This is the case, for example, for the PEGASOS algorithm, discussed in Section 8.10.3.

- In [4, 5], $\mathcal{O}(1/N)$ convergence rates are derived for a set of not strongly convex smooth loss functions (squared error and logistic regression) even for the case of constant step-sizes. The analysis method follows statistical arguments.

8.12 ONLINE LEARNING AND BIG DATA APPLICATIONS: A DISCUSSION

Online learning algorithms have been treated in Chapters 4, 5 and 6. The purpose of this section is first to summarize some of the findings and at the same time to present a discussion related to the performance of online schemes compared to their batch relatives.

Recall that the ultimate goal in obtaining a parametric predictor,

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}),$$

is to select $\boldsymbol{\theta}$ so that to optimize the *expected loss/risk* function, (8.65). For practical reasons, the corresponding empirical formulation in (8.66) is most often adopted instead. From the learning theory's point of view, this is justified provided the respective class of functions is sufficiently restrictive [89]. The available literature is quite rich in obtaining performance bounds that measure how close the optimal value obtained via the expected risk is to that obtained via the empirical one, as a function of the number of points N. Note that as $N \longrightarrow \infty$, and recalling well-known arguments from probability theory and statistics, the empirical risk tends to the expected risk (under general assumptions). Thus, for very large training data sets, adopting the empirical risk may not be that different from using the expected risk. However, for data sets of shorter lengths, a number of issues occur. Besides the value of N, another critical factor enters the scene; this is the *complexity* of the family of the functions, in which we search a solution. In other words, the generalization performance critically depends not only on N but also on how large or small this set of functions is. A related discussion for the specific case of the MSE was presented in Chapter 3, in the context of the bias-variance trade-off. The roots of the more general theory go back to the pioneering work of Vapnik-Chernovenkis; see [31, 90, 91], and [83] for a less mathematical summary of the major points.

In the sequel, we will summarize some of the available results tailored to the needs of our current discussion.

Approximation, estimation and optimization errors

Recall that all we are given in a machine learning task is the available training set of examples. To set up the "game," the designer has to decide on the selection of: (a) the loss function, $\mathcal{L}(\cdot,\cdot)$, which measures the deviation (error) between predicted and desired values and (b) the set of (parametric) functions $\mathcal{F}$,

$$\mathcal{F} = \left\{ f_{\boldsymbol{\theta}}(\cdot) : \boldsymbol{\theta} \in \mathbb{R}^K \right\}.$$

Based on the choice of $\mathcal{L}(\cdot,\cdot)$, the *benchmark* function, denoted as f_*, is the one that minimizes the expected risk (see also Chapter 3), that is,

$$f_*(\cdot) = \arg\min_f \mathbb{E}\left[\mathcal{L}\left(\mathrm{y}, f(\mathbf{x})\right)\right],$$

or equivalently

$$f_*(\boldsymbol{x}) = \arg\min_{\hat{y}} \mathbb{E}\left[\mathcal{L}\left(y,\hat{y}\right) \left| \boldsymbol{x}\right.\right]. \tag{8.91}$$

Let also $f_{\boldsymbol{\theta}_*}$ denote the optimal function that results by minimizing the expected risk constrained within the parametric family $\mathcal{F}$, that is,

$$\boxed{f_{\boldsymbol{\theta}_*}(\cdot):\ \boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}\left[\mathcal{L}\left(\mathrm{y}, f_{\boldsymbol{\theta}}(\mathbf{x})\right)\right].} \tag{8.92}$$

However, instead of $f_{\boldsymbol{\theta}_*}$, we obtain another function, denoted as f_N, by minimizing the empirical risk, $J_N(\boldsymbol{\theta})$,

$$\boxed{f_N(\boldsymbol{x}) := f_{\boldsymbol{\theta}_*(N)}(\boldsymbol{x}):\quad \boldsymbol{\theta}_*(N) = \arg\min_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta}).} \tag{8.93}$$

Once f_N has been obtained, we are interested in evaluating its generalization performance; that is, to compute the value of the expected risk at f_N, $\mathbb{E}[\mathcal{L}(\mathrm{y}, f_N(\mathbf{x}))]$. The excess error with respect to the optimal value can then decomposed as [12],

$$\mathcal{E} = \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_N(\mathbf{x}))\right] - \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_*(\mathbf{x})\right] = \mathcal{E}_{\text{appr}} + \mathcal{E}_{\text{est}} \tag{8.94}$$

where,

$$\boxed{\mathcal{E}_{\text{appr}} := \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_{\boldsymbol{\theta}_*}(\mathbf{x}))\right] - \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_*(\mathbf{x}))\right]:\quad \text{Approximation Error,}}$$

$$\boxed{\mathcal{E}_{\text{est}} := \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_N(\mathbf{x}))\right] - \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_{\boldsymbol{\theta}_*}(\mathbf{x}))\right]:\quad \text{Estimation Error,}}$$

where $\mathcal{E}_{\text{appr}}$ is known as the *approximation* error and $\mathcal{E}_{\text{est}}$ is known as the *estimation* error. The former measures how well the chosen family of functions can perform compared to the optimal/benchmark value and the latter measures the performance loss within the family $\mathcal{F}$, due to the fact that optimization is performed via the empirical loss function. Large families of functions lead to low approximation error but higher estimation error and vice versa. A way to improve upon the estimation error, while keeping the approximation error small, is to increase N. The size/complexity of the family $\mathcal{F}$ is measured by its *capacity*, which may depend on the number of parameters, but this is not always the whole story; see, for example, [83, 90]. For example, the use of regularization, while minimizing the empirical risk, can have a decisive effect on the approximation-estimation error trade-off.

In practice, while optimizing the (regularized) empirical risk, one has to adopt an iterative minimization or an online algorithm, which leads to an approximate solution, denoted as $\tilde{f}_N$. Then the excess error in (8.94) involves a third term, [12, 13],

$$\mathcal{E} = \mathcal{E}_{\text{appr}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}}, \tag{8.95}$$

where

$$\boxed{\mathcal{E}_{\text{opt}} := \mathbb{E}\left[\mathcal{L}(\mathrm{y}, \tilde{f}_N(\mathbf{x}))\right] - \mathbb{E}\left[\mathcal{L}(\mathrm{y}, f_N(\mathbf{x}))\right]:\quad \text{Optimization Error.}}$$

The literature is rich in deriving bounds concerning the excess error. More detailed treatment is beyond the scope of this book. As a case study, we will follow the treatment given in [13].

Let the computation of $\tilde{f}_N$ be associated with a predefined accuracy

$$\mathbb{E}\left[\mathcal{L}(\mathrm{y},\tilde{f}_N(\mathbf{x}))\right] < \mathbb{E}\left[\mathcal{L}(\mathrm{y},f_N(\mathbf{x}))\right] + \rho.$$

Then, for a class of functions that are often met in practice, for example, under strong convexity of the loss function [50] or under certain assumptions on the data distribution [87], the following equivalence relation can be established,

$$\mathcal{E}_{\mathrm{appr}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}} \sim \mathcal{E}_{\mathrm{appr}} + \left(\frac{\ln N}{N}\right)^{a} + \rho, \quad a \in \left[\frac{1}{2}, 1\right], \tag{8.96}$$

which verifies the fact that as $N \longrightarrow \infty$ the estimation error decreases and provides a rule for the respective convergence rate. The excess error $\mathcal{E}$, besides the approximation component, on which we have no access to control (given the family of functions, $\mathcal{F}$), it depends on (a) the number of data and (b) on the accuracy, ρ, associated with the algorithm used. How one can control these parameters depends on the type of learning task at hand.

- *Small scale tasks*: These types of tasks are constrained by the number of training points N. In this case, one can reduce the optimization error, since computational load is not a problem, and achieve the minimum possible estimation error, as this is allowed by the number of available training points. In this case, one achieves the approximation-estimation trade-off.
- *Large scale/big data tasks*: These types of tasks are constrained by the computational resources. Thus, a computationally cheap and less accurate algorithm may end up with lower excess error, since it has the luxury of exploiting more data, compared to a more accurate yet computationally more complex algorithm, given the maximum allowed computational load.

Batch versus online learning

Our interest in this subsection lies in investigating whether there is a performance loss if in place of a batch algorithm an online one is used instead. There is a very subtle issue involved here, which turns out to be very important from a practical point of view. We will restrict our discussion to differentiable convex loss functions.

Two major factors associated with the performance of an algorithm (in a stationary environment) are its convergence rate and its accuracy after convergence. The general form of a batch algorithm in minimizing (8.66) is written as

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu_i \Phi_i \nabla J_N(\boldsymbol{\theta}^{(i-1)}) \\ &= \boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{N}\Phi_i \sum_{n=1}^{N} \mathcal{L}'(\boldsymbol{\theta}^{(i-1)}, y_n, \boldsymbol{x}_n).\end{aligned} \tag{8.97}$$

For gradient descent, $\Phi_i = I$, and for Newton-type recursions, Φ_i is the inverse Hessian matrix of the loss function (Chapter 6).

Note that these are not the only possible choices for matrix Φ. For example, in the Levenberg-Marquardt method, the square Jacobian is employed, that is,

$$\Phi_i = \left[\nabla J(\boldsymbol{\theta}^{(i-1)}) \nabla^{\mathrm{T}} J(\boldsymbol{\theta}^{(i-1)}) + \lambda I\right]^{-1},$$

where λ is a regularization parameter. In [3], the *natural gradient* is proposed, which is based on the Fisher information matrix associated with the noisy distribution implied by the adopted prediction model, $f_{\boldsymbol{\theta}}(\boldsymbol{x})$. In both cases, the involved matrices asymptotically behave like the Hessian, yet they may provide improved performance during the initial convergence phase. For a further discussion, the interested reader may consult [48, 55].

As it has already being mentioned in Chapters 5 and 6 (Section 6.47), the convergence rate to the respective optimal value of the simple gradient descent method is *linear*, that is,

$$\ln \frac{1}{||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*(N)||^2} \propto i,$$

and the corresponding rate for a Newton-type algorithm is (approximately) quadratic, that is,

$$\ln \ln \frac{1}{||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*(N)||^2} \propto i.$$

In contrast, the online version of (8.97), that is,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \Phi_n \mathcal{L}'(\boldsymbol{\theta}_{n-1}, y_n, \boldsymbol{x}_n), \tag{8.98}$$

is based on a *noisy* estimation of the gradient, using the current sample point, $(y_n, \boldsymbol{x}_n)$, only. The effect of this is to slow down convergence, in particular when the algorithm gets close to the solution. Moreover, the estimate of the parameter vector fluctuates around the optimal value. We have extensively studied this phenomenon in the case of the LMS, when μ_n is assigned a constant value. This is the reason that in the stochastic gradient rationale, μ_n must be a decreasing sequence. However, it *must not* decrease very fast, which is guaranteed by the condition $\sum_n \mu_n \longrightarrow \infty$, Section 5.4. Furthermore, recall from our discussion there that the rate of convergence toward $\boldsymbol{\theta}_*$ is, on average, $\mathcal{O}(1/n)$. This result also covers the more general case of online algorithms given in (8.98), see, for example, [55]. Note, however, that all these results have been derived under a number of assumptions, for example, that the algorithm is close enough to a solution.

Our major interest now turns on comparing the rate at which a batch and a corresponding online algorithm converge to $\boldsymbol{\theta}_*$; that is, the value that minimizes the expected risk, which is the ultimate goal of our learning task. Since the aim is to compare performances, given the same number of training samples, let us use the same number, both for n in the online and N for the batch. Following [11] and applying a second order Taylor expansion on $J_n(\boldsymbol{\theta})$, it can be shown (Problem 8.28) that

$$\boldsymbol{\theta}_*(n) = \boldsymbol{\theta}_*(n-1) - \frac{1}{n} \Psi_n^{-1} \mathcal{L}'\left(\boldsymbol{\theta}_*(n-1), y_n, \boldsymbol{x}_n\right), \tag{8.99}$$

where

$$\Psi_n = \left(\frac{1}{n} \sum_{k=1}^{n} \nabla^2 \mathcal{L}\left(\boldsymbol{\theta}_*(n-1), y_k, \boldsymbol{x}_k\right) \right).$$

Note that (8.99) is similar in structure with (8.98). Also, as $n \longrightarrow \infty$, Ψ_n converges to the Hessian matrix, H, of the expected risk function. Hence, for appropriate choices of the involved weighting matrices and setting $\mu_n = 1/n$, (8.98) and (8.99) can converge to $\boldsymbol{\theta}_*$ at similar rates; thus, in both cases, the critical factor that determines how close to the optimal, $\boldsymbol{\theta}_*$, the resulting estimates are, is the number of data points used. It can be shown [11, 55, 88], that

$$\boxed{\mathbb{E}\left[||\theta_n - \boldsymbol{\theta}_*||^2\right] + \mathcal{O}\left(\frac{1}{n}\right) = \mathbb{E}\left[||\theta_*(n) - \boldsymbol{\theta}_*||^2\right] + \mathcal{O}\left(\frac{1}{n}\right) = \frac{C}{n},}$$

where C is a constant depending on the specific form of the associated expected loss function used. *Thus, batch algorithms and their online versions can be made to converge at similar rates to* $\boldsymbol{\theta}_*$*, after appropriate fine-tuning of the involved parameters.* Once more, since the critical factor in big data applications is not data but computational resources, a cheap online algorithm can achieve enhanced performance (lower excess error) compared to a batch, yet computationally more thirsty scheme. This is because for a given computational load, the online algorithm can process more data points (Problem 8.33). More importantly, an online algorithm needs not to store the data, which can be processed on the fly as they arrive. For a more detailed treatment of the topic, the interested reader may consult [13].

In [13], two forms of batch linear support vector machines (Chapter 11) were tested against their online stochastic gradient counterparts. The tests were carried out on the RCV1 data basis [52], and the training set comprised 781.265 documents represented by (relatively) sparse feature vectors consisting of 47.152 feature values. The stochastic gradient online versions, appropriately tuned with a diminishing step-size, achieved comparable error rates at substantially lower computational times (less than one tenth) compared to their batch processing relatives.

Remarks 8.10.

- Most of our discussion on the online versions has been focused on the simplest version, given in (8.98) for $\Phi_n = I$. However, the topic of stochastic gradient descent schemes, especially in the context of smooth loss functions, has a very rich history of over 60 years, and many algorithmic variants have been "born." In Chapter 5, a number of variations of the basic LMS scheme were discussed. Some more notable examples, which are still popular are:
  Stochastic gradient descent with momentum: The basic iteration of this variant is

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \mathcal{L}'_n(\boldsymbol{\theta}_{n-1}) + \beta_n(\boldsymbol{\theta}_{n-1} - \boldsymbol{\theta}_{n-2}). \tag{8.100}$$

  Very often, $\beta_n = \beta$ is chosen to be a constant; see, for example, [86].
  Gradient averaging: Another widely used version results if the place of the single gradient is taken by an average estimate, that is,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \frac{\mu_n}{n} \sum_{k=1}^{n} \mathcal{L}'_k(\boldsymbol{\theta}_{n-1}). \tag{8.101}$$

  Variants with different averaging scenarios (e.g., random selection instead of using all previously points) are also around. Such an averaging has a smoothing effect on the convergence of the algorithm. We have already seen this rationale in the context of the PEGASOS algorithm (Section 8.10.3). The general trend of all the variants of the basic stochastic gradient scheme is to improve upon the constants, but the convergence rate still remains to be $\mathcal{O}(1/n)$.

  In [49], the online learning rationale was used in the context of data sets of fixed size, N. Instead of using the gradient descent scheme in (8.97), the following version is proposed,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{N} \sum_{k=1}^{N} \boldsymbol{g}_k^{(i)}, \tag{8.102}$$

where

$$\boldsymbol{g}_k^{(i)} = \begin{cases} \mathcal{L}'_k(\boldsymbol{\theta}^{(i-1)}), & \text{if } k = i_k, \\ \boldsymbol{g}_k^{(i-1)}, & \text{otherwise.} \end{cases} \tag{8.103}$$

The index i_k is randomly chosen every time from $\{1, 2, \dots, N\}$. Thus, in each iteration only one gradient is computed and the rest are drawn from the memory. It turns out that, for strongly convex smooth loss functions, the algorithm exhibits linear convergence to the solution of the empirical loss in 8.56. Of course, compared to the basic online schemes, an $\mathcal{O}(N)$ memory is required, for keeping track of the gradient computations.

- The literature on deriving performance bounds concerning online algorithms is very rich, both in numbers as well as in ideas. For example, another line of research involves bounds for *arbitrary* online algorithms; see, [1, 19, 66] and references therein.

8.13 PROXIMAL OPERATORS

So far in the chapter, we have devoted a lot of space to the notion of the projection operator. In this section, we go one step further and we will introduce an elegant generalization of the notion of projection. Just to establish a clear understanding, when we refer to an operator, we mean a mapping from $\mathbb{R}^l \longmapsto \mathbb{R}^l$, in contrast to a function which is a mapping $\mathbb{R}^l \longmapsto \mathbb{R}$.

Definition 8.9. Let

$$f : \mathbb{R}^l \longmapsto \mathbb{R},$$

be a convex function and $\lambda > 0$. The corresponding *proximal* or *proximity* operator of index λ [59, 68],

$$\text{Prox}_{\lambda f} : \mathbb{R}^l \longmapsto \mathbb{R}^l, \tag{8.104}$$

is defined such as,

$$\boxed{\text{Prox}_{\lambda f}(\boldsymbol{x}) := \arg\min_{\boldsymbol{v} \in \mathbb{R}^l} \left\{ f(\boldsymbol{v}) + \frac{1}{2\lambda} ||\boldsymbol{x} - \boldsymbol{v}||^2 \right\} : \quad \text{Proximal Operator.}} \tag{8.105}$$

We stress that the proximal operator is a point in $\mathbb{R}^l$. The definition can also be extended to include functions defined as $f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\}$. A closely related notion to the proximal operator is the following.

Definition 8.10. Let f be a convex function as in the previous definition. We call the *Moreau* envelope, the function

$$\boxed{e_{\lambda f}(\boldsymbol{x}) := \min_{\boldsymbol{v} \in \mathbb{R}^l} \left\{ f(\boldsymbol{v}) + \frac{1}{2\lambda} ||\boldsymbol{x} - \boldsymbol{v}||^2 \right\} : \quad \text{Moreau Envelope.}} \tag{8.106}$$

Note that the Moreau envelope [58] is a *function* related to the proximal *operator* as

$$e_{\lambda f}(\boldsymbol{x}) = f\left(\text{Prox}_{\lambda f}(\boldsymbol{x})\right) + \frac{1}{2\lambda} ||\boldsymbol{x} - \text{Prox}_{\lambda f}(\boldsymbol{x})||^2. \tag{8.107}$$

The Moreau envelope can also be thought of as a regularized minimization, and it is also known as the *Moreau-Yosida regularization* [104].

A first point to clarify is whether the minimum in (8.105) exists. Note that the two terms in the brackets are both convex; namely, $f(\boldsymbol{v})$, and the quadratic term $||\boldsymbol{x}-\boldsymbol{v}||^2$. Hence, as it can easily be shown by recalling the definition of convexity, their sum is also convex. Moreover, the latter of the two terms is strictly convex, hence their sum is also strictly convex, which guarantees a *unique* minimum.

Example 8.9. Let us calculate $\text{Prox}_{\lambda\iota_C}$, where $\iota_C : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\}$ stands for the indicator function of a nonempty closed convex subset $C \subset \mathbb{R}^l$, defined as

$$\iota_C(\boldsymbol{x}) := \begin{cases} 0, & \text{if } \boldsymbol{x} \in C, \\ +\infty, & \text{if } \boldsymbol{x} \notin C. \end{cases}$$

It is not difficult to verify that

$$\begin{aligned} \text{Prox}_{\lambda\iota_C}(\boldsymbol{x}) &= \arg\min_{\boldsymbol{v}\in\mathbb{R}^l} \left\{ \iota_C(\boldsymbol{v}) + \frac{1}{2\lambda} \|\boldsymbol{x}-\boldsymbol{v}\|^2 \right\} \\ &= \arg\min_{\boldsymbol{v}\in C} \|\boldsymbol{x}-\boldsymbol{v}\|^2 = P_C(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbb{R}^l, \ \forall \lambda > 0, \end{aligned}$$

where P_C is the (metric) projection mapping onto C.

Moreover,

$$\begin{aligned} e_{\lambda\iota_C}(\boldsymbol{x}) &= \min_{\boldsymbol{v}\in\mathbb{R}^l} \left\{ \iota_C(\boldsymbol{v}) + \frac{1}{2\lambda} \|\boldsymbol{x}-\boldsymbol{v}\|^2 \right\} \\ &= \min_{\boldsymbol{v}\in C} \frac{1}{2\lambda} \|\boldsymbol{x}-\boldsymbol{v}\|^2 = \frac{1}{2\lambda} d_C^2(\boldsymbol{x}), \end{aligned}$$

where d_C stands for the (metric) distance function to C (Example 8.5), defined as $d_C(\boldsymbol{x}) := \min_{\boldsymbol{v}\in C} \|\boldsymbol{x}-\boldsymbol{v}\|$.

Thus, as said in the beginning of this section, the proximal operator can be considered as the generalization of the projection one.

Example 8.10. In the case where f becomes the ℓ_1-norm of a vector, that is,

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{l} |x_i|, \quad \forall \boldsymbol{x} \in \mathbb{R}^l,$$

then it is easily determined that (8.105) decomposes into a set of l scalar minimization tasks, that is,

$$\text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})|_i = \arg\min_{v_i\in\mathbb{R}} \left\{ |v_i| + \frac{1}{2\lambda}(x_i - v_i)^2 \right\}, \quad i = 1, 2, \ldots, l, \tag{8.108}$$

where $\text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})|_i$ denotes the respective ith element. Minimizing (8.108), is equivalent with requiring the subgradient to be zero, which results in

$$\begin{aligned} \text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})|_i &= \begin{cases} x_i - \text{sgn}(x_i)\lambda, & \text{if } |x_i| > \lambda, \\ 0, & \text{if } |x_i| \leq \lambda \end{cases} \\ &= \text{sgn}(x_i) \max\{0, |x_i| - \lambda\}. \end{aligned} \tag{8.109}$$

For the time being, the proof is left as an exercise. The same task is treated in detail in Chapter 9 and the proof is provided in Section 9.3. The operation in (8.109) is also known as *soft thresholding*. In other words, it sets to zero all values with magnitude less than a threshold value (λ) and adds a constant bias (depending on the sign) to the rest. To provoke the unfamiliar reader a bit, this is a way to impose sparsity on a parameter vector.

Having calculated $\text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})$, the Moreau envelope of $\|\cdot\|_1$ can be directly obtained by

$$\begin{aligned} e_{\lambda\|\cdot\|_1}(\boldsymbol{x}) &= \sum_{i=1}^{l}\left(\frac{1}{2\lambda}\left(x_i - \text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})|_i\right)^2 + \left|\text{Prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})|_i\right|\right) \\ &= \sum_{i=1}^{l}\left(\chi_{[0,\lambda]}(|x_i|)\frac{x_i^2}{2\lambda} + \chi_{(\lambda,+\infty)}(|x_i|)\left(|x_i - \text{sgn}(x_i)\lambda| + \frac{\lambda}{2}\right)\right) \\ &= \sum_{i=1}^{l}\left(\chi_{[0,\lambda]}(|x_i|)\frac{x_i^2}{2\lambda} + \chi_{(\lambda,+\infty)}(|x_i|)\left(|x_i| - \frac{\lambda}{2}\right)\right), \end{aligned}$$

where $\chi_{\mathcal{A}}(\cdot)$ denotes the characteristic function of the set $\mathcal{A}$, defined in (8.59). For the one-dimensional case, $l = 1$, the previous Moreau envelope boils down to

$$e_{\lambda|\cdot|}(x) = \begin{cases} |x| - \frac{\lambda}{2}, & \text{if } |x| > \lambda, \\ \frac{x^2}{2\lambda}, & \text{if } |x| \leq \lambda. \end{cases}$$

This envelope and the original $|\cdot|$ functions are depicted in Figure 8.31. It is worth-noticing here that $e_{\lambda|\cdot|}$ is a scaled version, more accurately $1/\lambda$ times, of the celebrated Huber's function; a loss function vastly used against outliers in robust statistics, which will be discussed in more detail in Chapter 11. Note that the Moreau envelope is a "blown-up" smoothed version of the ℓ_1 norm function and although the original function is not differentiable, its Moreau envelope is continuously differentiable; moreover, *they both share the same minimum*. This is most interesting and we will come back to that very soon.

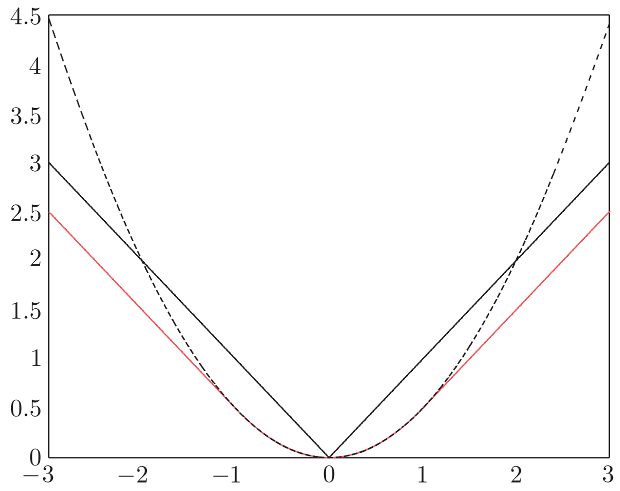


FIGURE 8.31

The $|x|$ function (black solid line), its Moreau envelope $e_{\lambda|\cdot|}(x)$ (red solid line), and $x^2/2$ (black dotted line), for $x \in \mathbb{R}$. Even if $|\cdot|$ is nondifferentiable at 0, $e_{\lambda|\cdot|}(x)$ is everywhere differentiable. Notice also that although $x^2/2$ and $e_{\lambda|\cdot|}(x)$ behave exactly the same for small values of x, $e_{\lambda|\cdot|}(x)$ is more conservative than $x^2/2$ in penalizing large values of x; this is the reason for the extensive usability of the Huber function, a scaled-down version of $e_{\lambda|\cdot|}(x)$, as a robust tool against outliers in robust statistics.

8.13.1 PROPERTIES OF THE PROXIMAL OPERATOR

We now focus on some basic properties of the proximal operator, which will soon be used to give birth to a new class of algorithms for the minimization of nonsmooth convex loss functions.

Proposition 8.5. *Let a convex function*

$$f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\},$$

and $\mathrm{Prox}_{\lambda f}(\cdot)$ *its corresponding proximal operator of index* λ*. Then,*

$$\boldsymbol{p} = \mathrm{Prox}_{\lambda f}(\boldsymbol{x}),$$

if and only if

$$\boxed{\langle \boldsymbol{y} - \boldsymbol{p}, \boldsymbol{x} - \boldsymbol{p} \rangle \leq \lambda \left(f(\boldsymbol{y}) - f(\boldsymbol{p}) \right), \quad \forall \boldsymbol{y} \in \mathbb{R}^l.} \tag{8.110}$$

Another necessary condition is

$$\boxed{\left\| \mathrm{Prox}_{\lambda f}(\boldsymbol{x}) - \mathrm{Prox}_{\lambda f}(\boldsymbol{y}) \right\|^2 \leq \left\langle \boldsymbol{x} - \boldsymbol{y}, \mathrm{Prox}_{\lambda f}(\boldsymbol{x}) - \mathrm{Prox}_{\lambda f}(\boldsymbol{y}) \right\rangle.} \tag{8.111}$$

The proofs of (8.110) and (8.111) are given in Problems 8.34 and 8.35, respectively. Note that (8.111) is of the same flavor as (8.16), which is inherited to the proximal operator from its more primitive ancestor. In the sequel, we will make use of these properties to touch upon the algorithmic front, where our main interest lies.

Lemma 8.3. *Consider the convex function*

$$f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\},$$

and its proximal operator $\mathrm{Prox}_{\lambda f}(\cdot)$*, of index* λ*. Then, the fixed point set of the proximal operator coincides with the set of minimizers of* f*, that is,*

$$\boxed{\mathrm{Fix}\left(\mathrm{Prox}_{\lambda f}\right) = \left\{ \boldsymbol{x} : \boldsymbol{x} = \arg\min_{\boldsymbol{y}} f(\boldsymbol{y}) \right\}.} \tag{8.112}$$

Proof. The definition of the fixed point set has been given in Section 8.3.1. We first assume that a point $\boldsymbol{x}$ belongs to the fixed point set, hence the action of the proximal operator leaves it unaffected, that is,

$$\boldsymbol{x} = \mathrm{Prox}_{\lambda f}(\boldsymbol{x}),$$

and making use of (8.110), we get

$$\langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{x} - \boldsymbol{x} \rangle \leq \lambda \left(f(\boldsymbol{y}) - f(\boldsymbol{x}) \right), \quad \forall \boldsymbol{y} \in \mathbb{R}^l, \tag{8.113}$$

which results in

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}), \quad \forall \boldsymbol{y} \in \mathbb{R}^l. \tag{8.114}$$

That is, $\boldsymbol{x}$ is a minimizer of f. For the converse, we assume that $\boldsymbol{x}$ is a minimizer. Then (8.114) is valid, from which (8.113) is deduced, and since this is a necessary and sufficient condition for a point to be equal to the value of the proximal operator, we have proved the claim. □

This is a very interesting and elegant result. One can obtain the set of minimizers of a nonsmooth convex function by solving an equivalent smooth one. From a practical point of view, the value of the method depends on how easy it is to obtain the proximal operator. For example, we have already seen that if the goal is to minimize the ℓ_1 norm, the proximal operator is a simple soft-thresholding operation. Needless to say that life is not always that generous!

8.13.2 PROXIMAL MINIMIZATION

In this section, we will exploit our experience from Section 8.4 to develop iterative schemes which asymptotically land their estimates in the fixed point set of the respective operator. All that is required is for the operator to own a nonexpansiveness property.

Proposition 8.6. *The proximal operator associated with a convex function is nonexpansive, that is,*

$$\boxed{||\operatorname{Prox}_{\lambda f}(\boldsymbol{x}) - \operatorname{Prox}_{\lambda f}(\boldsymbol{y})|| \leq ||\boldsymbol{x} - \boldsymbol{y}||.} \tag{8.115}$$

Proof. The proof is readily obtained as a combination of the property in (8.111) with the Cauchy-Schwarz inequality. Moreover, it can also be shown that the relaxed version of the proximal operator (also known as the *reflected* version),

$$R_{\lambda f}(\boldsymbol{x}) := 2 \operatorname{Prox}_{\lambda f}(\boldsymbol{x}) - I, \tag{8.116}$$

is also nonexpansive with the same fixed point set as that of the proximal operator, Problem 8.36. □

Proposition 8.7. *Let*

$$f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\},$$

be a convex function, with the Prox$_{\lambda f}$ *being the respective proximal operator of index* λ*. Then, starting from an arbitrary point,* $\boldsymbol{x}_0 \in \mathbb{R}^l$*, the following iterative algorithm*

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \mu_k \left(\operatorname{Prox}_{\lambda f}(\boldsymbol{x}_{k-1}) - \boldsymbol{x}_{k-1}\right), \tag{8.117}$$

where $\mu_k \in (0, 2)$ *is such as*

$$\sum_{k=1}^{\infty} \mu_k(2 - \mu_k) = +\infty,$$

converges to an element of the fixed point set of the proximal operator; that is, it converges to a minimizer of f*. Proximal minimization algorithms are traced back in the early 1970s [56, 69].*

The proof of the proposition is given in Problem 8.36 [81]. Observe that (8.117) is the counterpart of (8.26). A special case occurs if $\mu_k = 1$, which results in

$$\boldsymbol{x}_k = \operatorname{Prox}_{\lambda f}(\boldsymbol{x}_{k-1}), \tag{8.118}$$

also known as the *proximal point* algorithm.

Example 8.11. Let us demonstrate the previous findings via the familiar optimization task of the quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}} A \boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}} \boldsymbol{x}.$$

It does not take long to see that the minimizer occurs at the solution of the linear system of equations

$$A\boldsymbol{x}_* = \boldsymbol{b}.$$

From the definition in (8.105), taking the gradient of the quadratic function and equating to zero we readily obtain that,

$$\mathrm{Prox}_{\lambda f}(\boldsymbol{x}) = \left(A + \frac{1}{\lambda} I\right)^{-1} \left(\boldsymbol{b} + \frac{1}{\lambda}\boldsymbol{x}\right), \tag{8.119}$$

and setting $\epsilon = \frac{1}{\lambda}$, the recursion in (8.118) becomes

$$\boldsymbol{x}_k = (A + \epsilon I)^{-1} (\boldsymbol{b} + \epsilon \boldsymbol{x}_{k-1}). \tag{8.120}$$

After some simple algebraic manipulations (Problem 8.37), we finally obtain

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + (A + \epsilon I)^{-1} (\boldsymbol{b} - A\boldsymbol{x}_{k-1}). \tag{8.121}$$

This scheme is known from the numerical linear algebra as *iterative refinement* algorithm [57]. It is used when the matrix A is near singular, so the regularization via ϵ helps the inversion. Note that at each iteration, $\boldsymbol{b} - A\boldsymbol{x}_{k-1}$ is the error committed by the current estimate. The algorithm belongs to a larger family of algorithms, known as stationary iterative or iterative relaxation schemes; we will meet such schemes in Chapter 10.

The interesting point here is that since the algorithm results as a special case of the proximal minimization algorithm, convergence to the solution is guaranteed even if ϵ is not small!

Resolvent of the subdifferential mapping

We will look at the proximal operator from a slightly different view, which will be useful to us soon, when it will be used for the solution of more general minimization tasks. We will follow a more descriptive and less mathematically formal path.

According to Lemma 8.2 and since the proximal operator is a minimizer of (8.105), it must be chosen so that

$$\boldsymbol{0} \in \partial f(\boldsymbol{v}) + \frac{1}{\lambda}\boldsymbol{v} - \frac{1}{\lambda}\boldsymbol{x}, \tag{8.122}$$

or

$$\boldsymbol{0} \in \lambda \partial f(\boldsymbol{v}) + \boldsymbol{v} - \boldsymbol{x}, \tag{8.123}$$

or

$$\boldsymbol{x} \in \lambda \partial f(\boldsymbol{v}) + \boldsymbol{v}. \tag{8.124}$$

Let us now define the mapping

$$(I + \lambda \partial f) : \mathbb{R}^l \longmapsto \mathbb{R}^l, \tag{8.125}$$

such that

$$(I + \lambda \partial f)(\boldsymbol{v}) = \boldsymbol{v} + \lambda \partial f(\boldsymbol{v}). \tag{8.126}$$

Note that this mapping is one-to-many, due to the definition of the subdifferential, which is a set.[9] However, its inverse mapping, denoted as

$$(I + \lambda \partial f)^{-1} : \mathbb{R}^l \longmapsto \mathbb{R}^l, \tag{8.127}$$

is single-valued, and as a matter of fact it coincides with the proximal operator; this is readily deduced from (8.124), which can equivalently be written as,

$$\boldsymbol{x} \in \big(I + \lambda \partial f\big)(\boldsymbol{v}),$$

which implies that

$$(I + \lambda \partial f)^{-1}(\boldsymbol{x}) = \boldsymbol{v} = \text{Prox}_{\lambda f}(\boldsymbol{x}). \tag{8.128}$$

However, we know that the proximal operator is unique. The operator in (8.127) is known as the *resolvent of the subdifferential mapping* [69].

As an exercise, let us now apply (8.128) to the case of Example 8.11. For this case, the subdifferential set is a singleton comprising the gradient vector,

$$\text{Prox}_{\lambda f}(\boldsymbol{x}) = (I + \lambda \nabla f)^{-1}(\boldsymbol{x}) \Longrightarrow (I + \lambda \nabla f)(\text{Prox}_{\lambda f}(\boldsymbol{x})) = \boldsymbol{x},$$

or by definition of the mapping $(I + \lambda \nabla f)(\cdot)$ and taking the gradient of the quadratic function,

$$\text{Prox}_{\lambda f}(\boldsymbol{x}) + \lambda \nabla f(\text{Prox}_{\lambda f}(\boldsymbol{x})) = \text{Prox}_{\lambda f}(\boldsymbol{x}) + \lambda A\, \text{Prox}_{\lambda f}(\boldsymbol{x}) - \lambda \boldsymbol{b} = \boldsymbol{x},$$

which finally results in

$$\text{Prox}_{\lambda f}(\boldsymbol{x}) = \left(A + \frac{1}{\lambda} I\right)^{-1} \left(\boldsymbol{b} + \frac{1}{\lambda}\boldsymbol{x}\right).$$

8.14 PROXIMAL SPLITTING METHODS FOR OPTIMIZATION

A number of optimization tasks often comes in the form of a summation of individual convex functions, some of them being differentiable and some of them nonsmooth. Sparsity-aware learning tasks are typical examples that have received a lot of attention recently, where the regularizing term is nonsmooth, for example, the ℓ_1 norm.

Our goal in this section is to solve the following minimization task

$$\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}} \Big\{ f(\boldsymbol{x}) + g(\boldsymbol{x}) \Big\}, \tag{8.129}$$

where both involved functions are convex,

$$f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\}, \quad g : \mathbb{R}^l \longmapsto \mathbb{R},$$

and g is assumed to be differentiable while f is a nonsmooth one. It turns out that the following iterative scheme

$$\boxed{\boldsymbol{x}_k = \underbrace{\text{Prox}_{\lambda_k f}}_{\text{backward step}} \underbrace{\big(\boldsymbol{x}_{k-1} - \lambda_k \nabla g(\boldsymbol{x}_{k-1})\big)}_{\text{forward step}},} \tag{8.130}$$

[9] A point-to-set mapping is also called a relation on $\mathbb{R}^l$.

converges to a minimizer of the sum of the involved functions, that is,

$$\boldsymbol{x}_k \longrightarrow \arg\min_{\boldsymbol{x}} \left\{ f(\boldsymbol{x}) + g(\boldsymbol{x}) \right\}, \tag{8.131}$$

for a properly chosen sequence, λ_k, and provided that the gradient is continuous Lipschitz, that is,

$$||\nabla g(\boldsymbol{x}) - \nabla g(\boldsymbol{y})|| \leq \gamma ||\boldsymbol{x} - \boldsymbol{y}||, \tag{8.132}$$

for some $\gamma > 0$. It can be shown that if $\lambda_k \in \left(0, \frac{1}{\gamma}\right]$, then the algorithm converges to a minimizer at a sublinear rate, $\mathcal{O}(1/k)$ [7]. This family of algorithms is known as *proximal gradient* or *forward-backward splitting* algorithms. The term *splitting* is inherited from the split of the function into two (or more generally into more) parts. The term *proximal* indicates the presence of the proximal operator of f in the optimization scheme. The iteration involves an (explicit) forward gradient computation step performed on the smooth part and an (implicit) backward step via the use of the proximal operator of the nonsmooth part. The terms *forward-backward* are borrowed from numerical analysis methods involving discretization techniques [92]. Proximal gradient schemes are traced back in, for example, [17, 53], but their spread in machine learning and signal processing matured later on [26, 36].

There are a number of variants of the previous basic scheme. A version that achieves $\mathcal{O}(\frac{1}{k^2})$ rate of convergence is based on the classical Nesterov's modification of the gradient algorithm [62], and it is summarized in Algorithm 8.5, [7]. In the algorithm, the update is split into two parts. In the proximal operator, one uses a smoother version of the obtained estimates, using an averaging that involves previous estimates.

Algorithm 8.5 (Fast proximal gradient splitting algorithm).

```
• Initialization
    • Select $\boldsymbol{x}_0, \boldsymbol{z}_1 = \boldsymbol{x}_0, t_1 = 1$.
    • Select $\lambda$.
• **For** $k = 1, 2, \ldots,$ **Do**
    • $\boldsymbol{y}_k = \boldsymbol{z}_k - \lambda \nabla g(\boldsymbol{z}_k)$
    • $\boldsymbol{x}_k = \text{Prox}_{\lambda f}(\boldsymbol{y}_k)$
    • $t_{k+1} = \frac{1+\sqrt{4t_k^2+1}}{2}$
    • $\mu_k = 1 + \frac{t_k - 1}{t_{k+1}}$
    • $\boldsymbol{z}_{k+1} = \boldsymbol{x}_k + \mu_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})$
• **End For**
```

Note that the algorithm involves a step-size μ_k. The computation of the variables t_k is done in such a way so that convergence speed is optimized. However, it has to be noted that convergence of the scheme is no more guaranteed, in general.

The proximal forward-backward splitting operator

From a first look, the iterative update given in (8.130) seems to be a bit "magic." However, this is not the case and we can come to it by following simple arguments starting from the basic property of a minimizer. Indeed, let $\boldsymbol{x}_*$ be a minimizer of (8.129). Then, we know that it has to satisfy

$$\begin{aligned} &\boldsymbol{0} \in \partial f(\boldsymbol{x}_*) + \nabla g(\boldsymbol{x}_*), \text{ or equivalently} \\ &\boldsymbol{0} \in \lambda \partial f(\boldsymbol{x}_*) + \lambda \nabla g(\boldsymbol{x}_*), \text{ or equivalently} \\ &\boldsymbol{0} \in \lambda \partial f(\boldsymbol{x}_*) + \boldsymbol{x}_* - \boldsymbol{x}_* + \lambda \nabla g(\boldsymbol{x}_*), \end{aligned}$$

or equivalently

$$\big(I - \lambda \nabla g\big)(\boldsymbol{x}_*) \in \big(I + \lambda \partial f\big)(\boldsymbol{x}_*),$$

or

$$\big(I + \lambda \partial f\big)^{-1}\big(I - \lambda \nabla g\big)(\boldsymbol{x}_*) = \boldsymbol{x}_*,$$

and finally

$$\boldsymbol{x}_* = \mathrm{Prox}_{\lambda f}\big(I - \lambda \nabla g(\boldsymbol{x}_*)\big). \tag{8.133}$$

In other words, a minimizer of the task is a fixed point of the operator

$$\big(I + \lambda \partial f\big)^{-1}\big(I - \lambda \nabla g\big) : \mathbb{R}^l \longmapsto \mathbb{R}^l. \tag{8.134}$$

The latter is known as the *proximal forward-backward splitting operator* and it can be shown that if $\lambda \in \left(0, \frac{1}{\gamma}\right]$, where γ is the Lipschitz constant, then this operator is *nonexpansive* [103]. This short story justifies the reason that the iteration in (8.130) is attracted toward the set of minimizers.

Remarks 8.11.

- The proximal gradient splitting algorithm can be considered as a generalization of some previously considered algorithms. If we set $f(\boldsymbol{x}) = \iota_C(\boldsymbol{x})$, the proximal operator becomes the projection operator and the projected gradient algorithm of (8.62) results. If $f(\boldsymbol{x}) = 0$, we obtain the gradient algorithm and if $g(\boldsymbol{x}) = 0$ the proximal point algorithm comes up.
- Besides batch proximal splitting algorithms, online schemes have been proposed, see [101, 102], [36, 47], with an emphasis on the ℓ_1 regularization tasks.
- The application and development of novel versions of this family of algorithms in the fields of machine learning and signal processing is still an ongoing field of research and the interested reader can delve deeper into the field, via [18, 28, 64, 103].

Alternating direction method of multipliers (ADMM)

Extensions of the proximal splitting gradient algorithm for the case where both functions, f and g, are nonsmooth have also been developed, such as the *Douglas-Rachford* algorithm [27, 53]. Here, we are going to focus on one of the most popular schemes known as the *alternating direction method of multipliers* (ADMM) algorithm [39].

The ADMM algorithm is based on the notion of the *augmented Lagrangian* and at its very heart lies the Lagrangian duality concept (Appendix C).

The goal is to minimize the sum $f(\boldsymbol{x}) + g(\boldsymbol{x})$, where both f and g can be nonsmooth. This equivalently can be written as

$$\text{minimize with respect to } \boldsymbol{x}, \boldsymbol{y} \quad f(\boldsymbol{x}) + g(\boldsymbol{y}), \tag{8.135}$$

$$\text{subject to} \quad \boldsymbol{x} - \boldsymbol{y} = \boldsymbol{0}. \tag{8.136}$$

The augmented Lagrangian is defined as

$$L_\lambda(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) := f(\boldsymbol{x}) + g(\boldsymbol{y}) + \frac{1}{\lambda}\boldsymbol{z}^T(\boldsymbol{x} - \boldsymbol{y}) + \frac{1}{2\lambda}||\boldsymbol{x} - \boldsymbol{y}||^2, \tag{8.137}$$

where we have denoted the corresponding Lagrange multipliers[10] by $\boldsymbol{z}$. The previous equation can be rewritten as

$$L_\lambda(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) := f(\boldsymbol{x}) + g(\boldsymbol{y}) + \frac{1}{2\lambda}||\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{z}||^2 - \frac{1}{2\lambda}||\boldsymbol{z}||^2. \tag{8.138}$$

The ADMM is given in Algorithm 8.6.

Algorithm 8.6 (The ADMM algorithm).

```
• Initialization
    • Fix $\lambda > 0$.
    • Select $\boldsymbol{y}_0, \boldsymbol{z}_0$.
• For $k = 1, 2, \ldots$, Do
    • $\boldsymbol{x}_k = \text{prox}_{\lambda f}(\boldsymbol{y}_{k-1} - \boldsymbol{z}_{k-1})$
    • $\boldsymbol{y}_k = \text{prox}_{\lambda g}(\boldsymbol{x}_k + \boldsymbol{z}_{k-1})$
    • $\boldsymbol{z}_k = \boldsymbol{z}_{k-1} + (\boldsymbol{x}_k - \boldsymbol{y}_k)$
• End For
```

Looking carefully at the algorithm and (8.138), observe that the first recursion corresponds to the minimization of the augmented Lagrangian with respect to $\boldsymbol{x}$, keeping the $\boldsymbol{y}$ and $\boldsymbol{z}$ fixed from the previous iteration. The second recursion corresponds to the minimization with respect to $\boldsymbol{y}$, by keeping $\boldsymbol{x}$ and $\boldsymbol{z}$ frozen to their currently available estimates. The last iteration is an update of the *dual* variables (Lagrange multipliers) in the *ascent* direction; note that the difference in the parentheses is the gradient of the augmented Lagrangian with respect to $\boldsymbol{z}$. Recall from Appendix C, that the saddle point is found as a max-min problem of the primal $(\boldsymbol{x},\boldsymbol{y})$ and the dual variables. The convergence of the algorithm has been analyzed in [39]. For related tutorial papers the reader can look in Refs. [15, 45].

Mirror descent algorithms

A closely related algorithmic family to the forward-backward optimization algorithms is traced back to the work in [61] and it is known as *mirror descent algorithms* (MDA). The method has undergone a number of evolutionary steps, for example, [8, 63]. Our focus will be on adopting online schemes to minimize the regularized expected loss function

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\right] + \phi(\boldsymbol{\theta}),$$

where the regularizing function, ϕ, is assumed to be convex, but not necessarily a smooth one. In a recent representative of this algorithmic class, known also as *regularized dual averaging* (ARD) algorithm [95], the main iterative equation is expressed as

$$\boldsymbol{\theta}_n = \min_{\boldsymbol{\theta}} \left\{\langle \bar{\mathcal{L}}', \boldsymbol{\theta}\rangle + \phi(\boldsymbol{\theta}) + \mu_n\psi(\boldsymbol{\theta})\right\}, \tag{8.139}$$

where ψ is a *strongly convex* auxiliary function. For example, one possibility is to choose $\phi(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$ and $\psi(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_2^2$, [95]. $\bar{\mathcal{L}}'$ denotes the average subgradient of $\mathcal{L}$ up to and including time instant $n-1$, that is,

$$\bar{\mathcal{L}}' = \frac{1}{n-1}\sum_{j=1}^{n-1} \mathcal{L}_j'(\boldsymbol{\theta}_j),$$

where $\mathcal{L}_j(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}, y_j, \boldsymbol{x}_j)$.

[10] In the book, we have used λ for the Lagrange multipliers. However, here, we have already reserved λ for the proximal operator.

It can be shown that if the subgradients are bounded, and $\mu_n = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$, then following regret analysis arguments an $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ convergence rate is achieved. If, on the other hand, the regularizing term is strongly convex and $\mu_n = \mathcal{O}\left(\frac{\ln n}{n}\right)$, then an $\mathcal{O}\left(\frac{\ln n}{n}\right)$ rate is obtained. In [95], different variants are proposed. One is based on Nesterov's arguments, as used in Algorithm 8.5, which achieves an $\mathcal{O}\left(\frac{1}{n^2}\right)$ convergence rate.

A closer observation of (8.139) reveals that it can be considered as a generalization of the recursion given in (8.130). Indeed, let us set in (8.139)

$$\psi(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2,$$

and in place of the average gradient consider the most recent value, $\mathcal{L}'_{n-1}$. Then, (8.139) becomes equivalent to

$$\mathbf{0} \in \mathcal{L}'_{n-1} + \partial\phi(\boldsymbol{\theta}) + \mu_n(\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}). \tag{8.140}$$

This is the same relation that would result from (8.130), if we set

$$f \to \phi,\ \boldsymbol{x}_k \to \boldsymbol{\theta}_n,\ \boldsymbol{x}_{k-1} \to \boldsymbol{\theta}_{n-1},\ g \to \mathcal{L},\ \lambda_k \to \frac{1}{\mu_n}. \tag{8.141}$$

As a matter of fact, using these substitutions and setting $\phi(\cdot) = ||\cdot||_1$, the FOBOS algorithm, cited before as an example of an online forward-backward scheme [36], results. However, in the case of (8.139) one has the luxury of using other functions in place of the squared Euclidean distance from $\boldsymbol{\theta}_{n-1}$.

A popular auxiliary function that has been exploited is the *Bregman divergence*. The Bregman divergence with respect to a function, say ψ, between two points $\boldsymbol{x}$, $\boldsymbol{y}$, is defined as

$$\boxed{B_\psi(\boldsymbol{x},\boldsymbol{y}) = \psi(\boldsymbol{x}) - \psi(\boldsymbol{y}) - \langle\nabla\psi(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y}\rangle : \quad \text{Bregman Divergence.}} \tag{8.142}$$

It is left as a simple exercise to verify that the Euclidean distance results as the Bregman divergence if $\psi(\boldsymbol{x}) = ||\boldsymbol{x}||^2$.

Another algorithmic variant is the so-called *composite mirror descent*, which employs the currently available estimate of the subgradient, instead of the average, combined with the Bregman divergence; that is, $\bar{\mathcal{L}}$ is replaced by $\mathcal{L}'_{n-1}$ and $\psi(\boldsymbol{\theta})$ by $B_\psi(\boldsymbol{\theta}, \boldsymbol{\theta}_{n-1})$ for some function ψ, [37]. In [38], a time-varying ψ_n is involved by using a weighted average of the Euclidean norm, as pointed out already in Section 8.10.3. Note that in these modifications, although they may look simple, the analysis of the respective algorithms can be quite hard and substantial differences can be obtained in the performance.

At the time this book was being compiled, this area was still a hot topic of research, and it was still too early to draw definite conclusions. It may turn out, as it is often the case, that different algorithms are better suited for different applications and data sets.

PROBLEMS

8.1 Prove the Cauchy-Schwarz inequality in a general Hilbert space.

8.2 Show (a) that the set of points in a Hilbert space $\mathbb{H}$,

$$C = \{\boldsymbol{x} : \|\boldsymbol{x}\| \leq 1\}$$

is a convex set, and (b) the set of points

$$C = \{\boldsymbol{x} : \ \|\boldsymbol{x}\| = 1\}$$

is a nonconvex one.

8.3 Show the first order convexity condition.

8.4 Show that a function f is convex, if the one-dimensional function,

$$g(t) := f(\boldsymbol{x} + t\boldsymbol{y}),$$

is convex, $\forall \boldsymbol{x}, \ \boldsymbol{y}$ in the domain of definition of f.

8.5 Show the second order convexity condition.
Hint. Show the claim first for the one-dimensional case, and then use the result of the previous problem for the generalization.

8.6 Show that a function

$$f : \mathbb{R}^l \longmapsto \mathbb{R}$$

is convex iff its epigraph is convex.

8.7 Show that if a function is convex, then its lower level set is convex for any ξ.

8.8 Show that in a Hilbert space, $\mathbb{H}$, the parallelogram rule,

$$\|\boldsymbol{x} + \boldsymbol{y}\|^2 + \|\boldsymbol{x} - \boldsymbol{y}\|^2 = 2\left(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2\right), \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{H}.$$

holds true.

8.9 Show that if $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{H}$, where $\mathbb{H}$ is a Hilbert space, then the induced by the inner product norm satisfies the triangle inequality, as required by any norm, that is,

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$$

8.10 Show that if a point $\boldsymbol{x}_*$ is a local minimizer of a convex function, it is necessarily a global one. Moreover, it is the unique minimizer if the function is strictly convex.

8.11 Let C be a closed convex set in a Hilbert space, $\mathbb{H}$. Then show that $\forall \boldsymbol{x} \in \mathbb{H}$, there exists a point, denoted as $P_C(\boldsymbol{x}) \in C$, such that

$$\|\boldsymbol{x} - P_C(\boldsymbol{x})\| = \min_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\|.$$

8.12 Show that the projection of a point $\boldsymbol{x} \in \mathbb{H}$ onto a nonempty closed convex set, $C \subset \mathbb{H}$, lies on the boundary of C.

8.13 Derive the formula for the projection onto a hyperplane in a (real) Hilbert space, $\mathbb{H}$.

8.14 Derive the formula for the projection onto a closed ball, $B[\boldsymbol{0}, \delta]$.

8.15 Find an example of a point whose projection on the ℓ_1 ball is not unique.

8.16 Show that if $C \subset \mathbb{H}$, is a closed convex set in a Hilbert space, then $\forall \boldsymbol{x} \in \mathbb{H}$ and $\forall \boldsymbol{y} \in C$, the projection $P_C(\boldsymbol{x})$ satisfies the following properties:

- $\text{Real}\{\langle \boldsymbol{x} - P_C(\boldsymbol{x}), \ \boldsymbol{y} - P_C(\boldsymbol{x})\rangle\} \leq 0.$
- $\|P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\|^2 \leq \text{Real}\{\langle \boldsymbol{x} - \boldsymbol{y}, \ P_C(\boldsymbol{x}) - P_C(\boldsymbol{y})\rangle\}.$

8.17 Prove that if S is a closed subspace $S \subset \mathbb{H}$ in a Hilbert space $\mathbb{H}$, then $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{H}$,

$$\langle \boldsymbol{x}, P_S(\boldsymbol{y})\rangle = \langle P_S(\boldsymbol{x}), \boldsymbol{y}\rangle = \langle P_S(\boldsymbol{x}), P_S(\boldsymbol{y})\rangle.$$

and

$$P_S(a\boldsymbol{x} + b\boldsymbol{y}) = aP_S(\boldsymbol{x}) + bP_S(\boldsymbol{y}).$$

Hint. Use the result of Problem 8.18.

8.18 Let S be a closed convex subspace in a Hilbert space $\mathbb{H}$, $S \subset \mathbb{H}$. Let $S^{\perp}$ be the set of all elements $\boldsymbol{x} \in \mathbb{H}$ which are orthogonal to S. Then show that, (a) $S^{\perp}$ is also a closed subspace, (b) $S \cap S^{\perp} = \{\mathbf{0}\}$, (c) $\mathbb{H} = S \oplus S^{\perp}$; that is, $\forall \boldsymbol{x} \in \mathbb{H},\ \exists \boldsymbol{x}_1 \in S$ and $\boldsymbol{x}_2 \in S^{\perp} : \boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2$, where $\boldsymbol{x}_1, \boldsymbol{x}_2$ are *unique*.

8.19 Show that the relaxed projection operator is a nonexpansive mapping.

8.20 Show that the relaxed projection operator is a strongly attractive mapping.

8.21 Give an example of a sequence in a Hilbert space $\mathbb{H}$, which converges weakly but not strongly.

8.22 Prove that if $C_1 \dots C_K$ are closed convex sets in a Hilbert space $\mathbb{H}$, then the operator

$$T = T_{C_K} \cdots T_{C_1},$$

is a *regular* one; that is,

$$\|T^{n-1}(\boldsymbol{x}) - T^{n}(\boldsymbol{x})\| \longrightarrow 0,\ n \longrightarrow \infty,$$

where $T^n := TT \dots T$ is the application of T n successive times.

8.23 Show the fundamental POCS theorem for the case of closed subspaces in a Hilbert space, $\mathbb{H}$.

8.24 Derive the subdifferential of the metric distance function $d_C(\boldsymbol{x})$, where C is a closed convex set $C \subseteq \mathbb{R}^l$ and $\boldsymbol{x} \in \mathbb{R}^l$.

8.25 Derive the bound in (8.55).

8.26 Show that if a function is γ-Lipschitz, then any of its subgradients is bounded.

8.27 Show the convergence of the generic projected subgradient algorithm in (8.61).

8.28 Derive Eq. (8.99).

8.29 Consider the online version of PDMb in (8.64), that is,

$$\boldsymbol{\theta}_n = \begin{cases} P_C\Big(\boldsymbol{\theta}_{n-1} - \mu_n \frac{J(\boldsymbol{\theta}_{n-1})}{\|J'(\boldsymbol{\theta}_{n-1})\|^2} J'(\boldsymbol{\theta}_{n-1})\Big), & \text{if } J'(\boldsymbol{\theta}_{n-1}) \neq \mathbf{0}, \\ P_C(\boldsymbol{\theta}_{n-1}), & \text{if } J'(\boldsymbol{\theta}_{n-1}) = \mathbf{0}, \end{cases} \tag{8.143}$$

where we have assumed that $J_* = 0$. If this is not the case, a shift can accommodate for the difference. Thus, we assume that we know the minimum. For example, this is the case for a number tasks, such as the hinge loss function, assuming linearly separable classes, or the linear ϵ-insensitive loss function, for bounded noise. Assume that

$$\mathcal{L}_n(\boldsymbol{\theta}) = \sum_{k=n-q+1}^{n} \frac{\omega_k d_{C_k}(\boldsymbol{\theta}_{n-1})}{\sum_{k=n-q+1}^{n} \omega_k d_{C_k}(\boldsymbol{\theta}_{n-1})} d_{C_k}(\boldsymbol{\theta}).$$

Then derive that APSM algorithm of (8.39).

8.30 Derive the regret bound for the subgradient algorithm in (8.82).

8.31 Show that a function $f(\boldsymbol{x})$ is σ-strongly convex if and only if the function $f(\boldsymbol{x}) - \frac{\sigma}{2}||\boldsymbol{x}||^2$ is convex.

8.32 Show that if the loss function is σ-strongly convex, then if $\mu_n = \frac{1}{\sigma n}$, the regret bound for the subgradient algorithm becomes

$$\frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) \le \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta}_*) + \frac{G^2(1+\ln N)}{2\sigma N}. \tag{8.144}$$

8.33 Consider a batch algorithm that computes the minimum of the empirical loss function, $\boldsymbol{\theta}_*(N)$, having a quadratic convergence rate, that is,

$$\ln\ln \frac{1}{||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*(N)||^2} \sim i.$$

Show that an online algorithm, running for n time instants so that to spend the same computational processing resources as the batch one, achieves for large values of N better performance than the batch algorithm, shown as [11]

$$||\boldsymbol{\theta}_n - \boldsymbol{\theta}_*||^2 \sim \frac{1}{N\ln\ln N} << \frac{1}{N} \sim ||\boldsymbol{\theta}_*(N) - \boldsymbol{\theta}_*||^2.$$

Hint. Use the fact that

$$||\boldsymbol{\theta}_n - \boldsymbol{\theta}_*||^2 \sim \frac{1}{n}, \quad \text{and} \quad ||\boldsymbol{\theta}_*(N) - \boldsymbol{\theta}_*||^2 \sim \frac{1}{N}.$$

8.34 Show property (8.110) for the proximal operator.
8.35 Show property (8.111) for the proximal operator.
8.36 Prove that the recursion in (8.117) converges to a minimizer of f.
8.37 Derive (8.121) from (8.120).

MATLAB Exercises

8.38 Consider the regression model,

$$y_n = \boldsymbol{\theta}_o^{\mathrm{T}}\boldsymbol{x}_n + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$ ($l = 200$) and the coefficients of the unknown vector are obtained randomly via the Gaussian distribution $\mathcal{N}(0, 1)$. The noise samples are also i.i.d. having zero mean and variance $\sigma_\eta^2 = 0.01$. The input sequence is a white noise one, i.i.d. generated via the Gaussian, $\mathcal{N}(0, 1)$.

Using as training data the samples, $(y_n, \boldsymbol{x}_n) \in \mathbb{R} \times \mathbb{R}^{200}$, $n = 1, 2, \ldots$, run the APA (Algorithm 5.2), the NLMS (Algorithm 5.3), the RLS (Algorithm 6.1) and the APSM (Algorithm 8.2) algorithms to estimate the unknown $\boldsymbol{\theta}_o$.

For the APA algorithm, choose $\mu = 0.2$, $\delta = 0.001$, and $q = 30$. For the APSM $\mu = 0.5 \times M_n$, $\epsilon = \sqrt{2}\sigma$ and $q = 30$. Furthermore, in the NLMS set $\mu = 1.2$ and $\delta = 0.001$. Finally, for the RLS set the forgetting factor β is equal to 1. Run 100 independent experiments and plot the average error per iteration in dBs, that is, $10\log_{10}(e_n^2)$, where $e_n^2 = (y_n - \boldsymbol{x}_n^T\boldsymbol{\theta}_{n-1})^2$. Compare the performance of the algorithms.

Keep the same parameters, but alter the noise variance so that it becomes 0.3. Plot the average error per iteration as in the previous experiment. What do you observe regarding the performance of the APA compared to the previous low-noise scenario?

Keep playing with different parameters and study the effect on the convergence speed and the error floor in which the algorithms converge.

8.39 Create an ad hoc network, having 10 nodes and a total number of 32 connections. Generate at each node the data, which adhere to the following model:

$$y_k(n) = \boldsymbol{\theta}_o^{\mathrm{T}}\boldsymbol{x}_k(n) + \eta_k(n), \quad k = 1, \ldots, 10.$$

The unknown vector $\boldsymbol{\theta}_o \in \mathbb{R}^{60}$ and its coefficients are generated randomly via the Gaussian $\mathcal{N}(0, 1)$. The input vectors are i.i.d. and follow a $\mathcal{N}(0, 1)$. Moreover, the noise samples are i.i.d. generated from zero mean Gaussians with variances corresponding to different signal-to-noise levels, varying randomly from 20 to 25 dBs from node to node.

For the unknown vector estimation employ the combine-then-adapt diffusion APSM (Algorithm 8.3), the adapt-then-combine LMS (Algorithm 5.7), the combine-then-adapt LMS (Algorithm 5.8), and the noncooperative LMS (Algorithm 5.1). For the combine-then-adapt APSM set $\mu_n = 0.5 \times M_n$, $\epsilon_k = \sqrt{2}\sigma_k$, and $q = 20$. For the adapt-then-combine, combine-then-adapt and noncooperative LMS set the step-size equal to 0.03. Finally, choose the combination weights a_{mk} with respect to the Metropolis rule (Remark 5.4).

Run 100 independent experiments and plot the average MSD per iteration in dBs, that is,

$$\text{MSD}(n) = 10 \log_{10} \left(\frac{1}{K} \sum_{k=1}^{K} \|\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o\|^2 \right).$$

Compare the performance of the combine-then-adapt APSM with the performance of the LMS-based algorithms.

Keep playing with different parameters for the involved algorithms and observe their influence on the obtained performance.

8.40 Download the *banknote authentication* dataset.[11] Develop a Matlab program that implements the PEGASOS algorithm for classification (Algorithm 8.4). Keep 90% of the data as a training set and the remaining 10% as a test set. Set $\lambda = 0.1$ and $m = 1, 10, 30$. Once the training phase has been completed, freeze the parameters for the obtained classifier. Compute the classification error on the test set using the obtained classifier. Compare the classification error for the three choices of m.

8.15 APPENDIX TO CHAPTER 8

In this appendix, we summarize some basic definitions and theorems concerning Hilbert spaces and convex analysis. No proofs are provided and the interested reader may consult more specialized books, given in the references.

Definition 8.11 (Linear Spaces). A nonempty set of elements, V, is called *linear space* if there are defined two operations, *addition* and *scalar multiplication*, so that the following properties hold true:

- $\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}$, $\forall \boldsymbol{x}, \boldsymbol{y} \in V$.
- $(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{z} = \boldsymbol{x} + (\boldsymbol{y} + \boldsymbol{z})$, $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in V$.
- There exists an element $\mathbf{0} \in V$, known as the *zero vector*, such that, $\forall \boldsymbol{x} \in V$, $\boldsymbol{x} + \mathbf{0} = \boldsymbol{x}$.
- $\forall \boldsymbol{x} \in V$, there is an element $\boldsymbol{y} \in V$ such that

$$\boldsymbol{x} + \boldsymbol{y} = \mathbf{0}.$$

- For each pair of scalars, $\alpha, \beta \in \mathbb{C}$ and $\forall \boldsymbol{x}, \boldsymbol{y} \in V$,

$$(\alpha\beta)\boldsymbol{x} = \alpha(\beta\boldsymbol{x}) \text{ and } \alpha(\boldsymbol{x} + \boldsymbol{y}) = \alpha\boldsymbol{x} + \alpha\boldsymbol{y}.$$

[11] https://archive.ics.uci.edu/ml/datasets/banknote+authentication.

- For each pair of scalars, $\alpha, \beta \in \mathbb{C}$, $\forall \boldsymbol{x} \in V$

$$(\alpha + \beta)\boldsymbol{x} = \alpha\boldsymbol{x} + \beta\boldsymbol{x}.$$

- $\forall \boldsymbol{x} \in V$, and the scalar $1 \in \mathbb{C}$,

$$1\boldsymbol{x} = \boldsymbol{x}.$$

Linear spaces are sometimes called *vector spaces* and the elements in V vectors. If the scalars α, β are restricted in $\mathbb{R}$, then the linear space is known as *real linear space*; otherwise if $\alpha, \beta \in \mathbb{C}$, the linear space is known as *complex linear space*.

Example 8.12 (The Vector Space $\mathbb{R}^l$)**.** The set of l-tuples $\boldsymbol{x} := (x_1, \ldots, x_l)$, $x_i \in \mathbb{R}$, $i = 1, 2, \ldots, l$, is a real vector space, where the addition and scalar multiplication are defined as

$$\begin{aligned} \boldsymbol{x} + \boldsymbol{y} &= (x_1 + y_1, \ldots, x_l + y_l) \\ a\boldsymbol{x} &= (ax_1, \ldots, ax_l), \; a \in \mathbb{R}. \end{aligned}$$

Example 8.13. Let the set of all real functions

$$\mathcal{F}(\mathbb{R}) = \{\boldsymbol{f} := f(x) | f : \mathbb{R} \longmapsto \mathbb{R}\},$$

where $\boldsymbol{f}$ denotes the “entire” function rather than the value at a specific point $x \in \mathbb{R}$. Then $\mathcal{F}(\mathbb{R})$ is a real linear space with respect to the following operations,

$$(\boldsymbol{f} + \boldsymbol{h})(x) = f(x) + h(x), \; \forall \boldsymbol{f}, \boldsymbol{h} \in \mathcal{F}(\mathbb{R}),$$

and

$$(a\boldsymbol{f})(x) = af(x), \; \forall \boldsymbol{f} \in \mathcal{F}(\mathbb{R}), \; a \in \mathbb{R}.$$

Definition 8.12. Let V be a linear space and S a nonempty set, $S \subseteq V$. Then S is called a *subspace* of V, if

- $\forall \boldsymbol{x}, \boldsymbol{y} \in S, \; \boldsymbol{x} + \boldsymbol{y} \in S.$
- $\forall a \in \mathbb{C}$, and $\boldsymbol{x} \in S, \; a\boldsymbol{x} \in S.$

Definition 8.13 (Linear Independency)**.** Let V be a linear space and $S \subseteq V$. S is said to be *linearly dependent*, if there exist a *finite* number of *distinct* elements, $\boldsymbol{x}_k \in S$, $k = 1, 2, \ldots, K$, such that

$$\sum_{k=1}^{K} a_k \boldsymbol{x}_k = \boldsymbol{0},$$

for some combination of scalars $a_k \in \mathbb{C}, k = 1, 2, \ldots, K$, which are *not all zero*. If this is not the case, the set S is known as *linearly independent*.

Let S be a nonempty set, $S \subseteq V$. The set of all possible linear combinations, denoted as span$\{S\}$,

$$\text{span}\{S\} = \left\{ \boldsymbol{x} : \; \boldsymbol{x} = \sum_{k=1}^{K} a_k \boldsymbol{x}_k \middle| \boldsymbol{x}_k \in S, K \in \mathbb{N} \right\}$$

and it is known as the *span* of S. Note that span$\{S\}$ is always a subspace of V. Moreover, if span$\{S\} = V$, we say that S spans the linear space V.

Definition 8.14 (Bases)**.** Let V be a linear space and $S \subseteq V$. The set S is known as *basis* of V, *if and only if:*

- S is linearly independent
- S spans V.

If the number of elements comprising S is finite, we say that V is *finite dimensional* and the number of distinct elements of S defines the *dimension* of V. If the number of elements in S is not finite, we say that V is *infinite dimensional*. Note that there is not a unique basis in V. However, any basis of V has the same number of elements. Moreover, it has been shown that every linear space has a basis. This is known as Zorn's lemma. However, finding a basis is not necessarily a trivial task. The dimension of $\mathbb{R}^l$ is l, and the linear space $\mathcal{F}(\mathbb{R})$ is infinite dimensional.

Definition 8.15 (Inner Product)**.** Let V be a linear space. The *inner product* is a function

$$f : V \times V \longmapsto \mathbb{C},$$

which assigns a value in $\mathbb{C}$, denoted as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$, to every point of elements $\boldsymbol{x}, \boldsymbol{y} \in V$, with the following properties:

- $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, and $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ iff $\boldsymbol{x} = 0$.
- $\langle \boldsymbol{x} + \boldsymbol{y}, \boldsymbol{z} \rangle = \langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \boldsymbol{y}, \boldsymbol{z} \rangle$.
- $\langle a\boldsymbol{x}, \boldsymbol{y} \rangle = a\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.
- $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle^*$.

where $*$ denotes complex conjugation. A space where an inner product operation has been defined is known as an *inner product space*.

A direct consequence of the previous inner product properties are the following,

$$\langle \boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z} \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{x}, \boldsymbol{z} \rangle,$$

and

$$\langle \boldsymbol{x}, a\boldsymbol{y} \rangle = a^* \langle \boldsymbol{x}, \boldsymbol{y} \rangle.$$

Example 8.14. Let us consider the vector space $\mathbb{C}^l$. Then the operation

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^{l} x_i y_i^* = \boldsymbol{y}^H \boldsymbol{x}, \tag{8.145}$$

is an inner product with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^l$.

Definition 8.16 (Norm and Normed Spaces)**.** Let V be a linear space. A *norm* is a function

$$f : V \longmapsto [0, \infty),$$

that assigns a nonnegative real number to any $\boldsymbol{x} \in V$, it is denoted as $\|\boldsymbol{x}\|$, and it has the following properties:

- $\|\boldsymbol{x}\| \geq 0$, and $\|\boldsymbol{x}\| = 0$ iff $\boldsymbol{x} = \boldsymbol{0}$.
- $\|a\boldsymbol{x}\| = |a|\|\boldsymbol{x}\|, \forall a \in \mathbb{C}$, and $\boldsymbol{x} \in V$.
- $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in V$.

The vector space, V, together with its norm $\|\cdot\|$ is known as a *normed linear space*. The third property is known as the *triangle inequality*.

Given a linear space, one can define different norms. Take, for example, the vector space $\mathbb{C}^l$. Then define the ℓ_p norm as

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{l} |x_i|^p\right)^{\frac{1}{p}}, \quad p \geq 1.$$

It can be shown that the above definition complies with all the required properties for a function to be a norm. For $p = 1$, we refer to the ℓ_1 norm and for $p = 2$ it is known as the *Euclidean norm* or ℓ_2 norm. Note that the latter one results from the inner product operation as defined in (8.145), that is,

$$\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}^H\boldsymbol{x}}. \tag{8.146}$$

This is valid for any inner product linear space. That is, given an inner product linear space, V, with $\langle\cdot,\cdot\rangle$, then the inner product operation induces a norm, that is,

$$\|\boldsymbol{x}\| = \sqrt{\langle\boldsymbol{x},\boldsymbol{x}\rangle}, \quad \forall \boldsymbol{x} \in V.$$

Theorem 8.7 (Cauchy-Schwarz Inequality)**.** *Let V be an inner product space and $\|\cdot\|$ the induced by the inner product norm. Then*

$$\boxed{|\langle\boldsymbol{x},\boldsymbol{y}\rangle| \leq \|\boldsymbol{x}\|\|\boldsymbol{y}\|, \ \forall \boldsymbol{x},\boldsymbol{y} \in V: \quad \text{Cauchy-Schwarz Inequality.}} \tag{8.147}$$

This is one of among the most fundamental and important properties in the theory of linear spaces.

A direct consequence of the Cauchy-Schwarz inequality are the following properties:

Given a inner vector space and its induced norm $\|\cdot\|$, then

•

$$\Big|\|\boldsymbol{x}\| - \|\boldsymbol{y}\|\Big| \leq \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{8.148}$$

•

$$\|\boldsymbol{x}+\boldsymbol{y}\|^2 + \|\boldsymbol{x}-\boldsymbol{y}\|^2 = 2\left(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2\right), \tag{8.149}$$

The latter is known as the *parallelogram law*. Note that, all these properties, which may be known from the basic geometry, are valid for *any* linear space, even for infinite dimensional ones.

Example 8.15 (The $\boldsymbol{\ell^2}$ space)**.** This is the linear space of all sequences

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n, \ldots),$$

with inner product operation

$$\langle\boldsymbol{x},\boldsymbol{y}\rangle = \sum_{n=1}^{\infty} x_n y_n^*,$$

whose inner product induced norm satisfies the property

$$\|\boldsymbol{x}\| := \sqrt{\sum_{n=1}^{\infty} |x_n|^2} < \infty.$$

Example 8.16 (The $\boldsymbol{L^2}$ Space)**.** This is the linear space of all integrable functions

$$f : \mathbb{R} \longmapsto \mathbb{R},$$

with inner product operation

$$\langle \boldsymbol{f}, \boldsymbol{h} \rangle := \int_{-\infty}^{\infty} f(x)h(x)dx,$$

whose inner product induced norm satisfies the property

$$\|\boldsymbol{f}\| := \sqrt{\int_{-\infty}^{\infty} |f(x)|^2 dx} < +\infty.$$

Definition 8.17 (Convergence, Cauchy Sequences, and Complete Spaces)**.** Let V be a normed linear space, and let a sequence of elements in V, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n, \ldots$ The sequence is said to *converge* to $\boldsymbol{x}$ if

$$\lim_{n\to\infty} \|\boldsymbol{x}_n - \boldsymbol{x}\| = 0.$$

Note that if $\boldsymbol{x}$ exists, it is unique and is known as the limit of $\boldsymbol{x}_n$.

A sequence of elements in a normed linear space V is called a *Cauchy sequence* if it satisfies

$$\lim_{n,m\to\infty} \|\boldsymbol{x}_n - \boldsymbol{x}_m\| = 0.$$

In other words, the norm of the difference of *any* two elements in the sequence eventually becomes zero. It turns out that any convergent sequence is Cauchy, but the opposite is not always true.

A normed linear space, V, in which every Cauchy sequence converges in V is said to be *complete*.

Note that any finite dimensional linear space is complete. However, this is not always true for infinite dimensional spaces. Intuitively, a space is complete if there are no points missing from it (including points at the boundary or the interior). Note that a complete space is something stronger than a closed space. In a closed subspace, every convergent sequence has its limit point in the subspace.

Definition 8.18 (Hilbert Spaces)**.** An inner product space, which is complete with respect to the norm induced by the inner product, is called a *Hilbert space*.

Examples of Hilbert spaces are the ℓ^2 and L^2 spaces. Also, the vector spaces $\mathbb{C}^l$ and $\mathbb{R}^l$, equipped with the inner product operation in (8.145) and the Euclidean norm in (8.146), known as Euclidean spaces are special finite dimensional cases of Hilbert spaces. The spaces ℓ^2 and L^2 are of infinite dimensionality. Note that $\mathbb{C}^l$, equipped with the ℓ_p norm, $p \neq 2$, is not a Hilbert space, since such a norm is not induced by an inner product.

Definition 8.19 (Weak Convergence)**.** Let $\boldsymbol{x}_n$ be a sequence of elements/points in a Hilbert space $\mathbb{H}$. We say that $\boldsymbol{x}_n$ converges *weakly* to a point $\boldsymbol{x} \in \mathbb{H}$, if

$$\langle \boldsymbol{x}_n, \boldsymbol{y} \rangle \xrightarrow[n\to\infty]{} \langle \boldsymbol{x}, \boldsymbol{y} \rangle, \quad \forall \boldsymbol{y} \in \mathbb{H},$$

and we write

$$\boldsymbol{x}_n \xrightarrow[n\to\infty]{w} \boldsymbol{x}.$$

Theorem 8.8. *Let $\mathbb{H}$ be a Hilbert space. Then the following hold true:*

- *If $\boldsymbol{x}_n \xrightarrow[n\to\infty]{} \boldsymbol{x}$, then $\boldsymbol{x}_n \xrightarrow[n\to\infty]{w} \boldsymbol{x}$.*
- *If $\mathbb{H}$ is of finite dimensionality, then $\boldsymbol{x}_n \xrightarrow[n\to\infty]{w} \boldsymbol{x}$, implies $\boldsymbol{x}_n \xrightarrow[n\to\infty]{} \boldsymbol{x}$.*

Thus, in a Euclidean space $\mathbb{C}^l$, weak convergence and convergence coincide.

REFERENCES

[1] A. Agarwal, P. Bartlett, P. Ravikumar, M.J. Wainwright, Information-theoretic lower bounds on the oracle complexity of convex optimization, IEEE Trans. Inform. Theory 58(5) (2012) 3235-3249.

[2] A.E. Albert, L.A. Gardner, Stochastic Approximation and Nonlinear Regression, MIT Press, 1967.

[3] S. Amari, Natural gradient works efficiently in learning, Neural Comput. 10(2) (1998) 251-276.

[4] F. Bach, E. Moulines, Non-strongly-convex smooth stochastic approximation with convergence rate $O(1/n)$, arXiv:1306.2119v1[cs.LG], 2013.

[5] F. Bach, Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression, J. Machine Learn. Res. 15 (2014) 595-627.

[6] H.H. Bauschke, J.M. Borwein, On projection algorithms for solving convex feasibility problems, SIAM Rev. 38(3) (1996) 367-426.

[7] A. Beck, M. Teboulle, Gradient-based algorithms with applications to signal recovery problems, in: D. Palomar, Y. Eldar (Eds.), Convex Optimization in Signal Processing and Communications, Cambridge University Press, 2010, pp. 42-88.

[8] A. Beck, M. Teboulle, Mirror descent and nonlinear projected subgradient methods for convex optimization, Operat. Res. Lett. 31 (2003) 167-175.

[9] D.P. Bertsekas, Nonlinear Programming, second ed., Athena Scientific, 1999.

[10] D.P. Bertsekas, A. Nedic, A.E. Ozdaglar, Convex Analysis and Optimization, Athena Scientific, 2003.

[11] L. Bottou, Y. Le Cun, Large scale online learning, Advances in Neural Information Processing Systems, NIPS, MIT Press, 2003, pp. 2004-2011.

[12] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, Adv. Neural Inform. Process. Syst. 20 (2007) 161-168.

[13] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Y. Lechevallier, G. Saporta (Eds.), Proceedings 19th Intl. Conference on Computational Statistics, COMPSTAT 2010, Paris, France, Springer, 2010.

[14] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[15] S. Boyd, N. Parikh, E. Chu, P. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Machine Learn. 3(1) (2011) 1122, NOW.

[16] L.M. Bregman, The method of successive projections for finding a common point of convex sets, Soviet Math. Dokl. 6 (1965) 688-692.

[17] R. Bruck, An iterative solution of a variational inequality for certain monotone operator in a Hilbert space, Bull. Amer. Math. Soc. 81(5) (1975) 890-892.

[18] H.H. Bauschke, P.L. Combettes, Convex Analysis and Monotone Operator Theory in Hilbert Spaces, Springer, 2011.

[19] N. Cesa-Bianchi, A. Conconi, C. Gentile, On the generalization ability of on-line learning algorithms, IEEE Trans. Inform. Theory, 50(9) (2004) 2050-2057.

[20] R.L.G. Cavalcante, I. Yamada, B. Mulgrew, An adaptive projected subgradient approach to learning in diffusion networks, IEEE Trans. Signal Process. 57(7) (2009) 2762-2774.

[21] N. Cesa-Bianchi, G. Lugosi, Prediction, Learning, and Games, Cambridge University Press, 2006.

[22] S. Chouvardas, K. Slavakis, S. Theodoridis, Adaptive robust distributed learning in diffusion sensor networks, IEEE Trans. Signal Process. 59(10) (2011) 4692-4707.

[23] S. Chouvardas, K. Slavakis, S. Theodoridis, I. Yamada, Stochastic analysis of hyperslab-based adaptive projected subgradient method under boundary noise, IEEE Signal Process. Lett. 20(7) (2013) 729-732.

[24] P.L. Combettes, The foundations of set theoretic estimation, Proc. IEEE 81(2) (1993) 182-208.

[25] P.L. Combettes, Inconsistent signal feasibility problems: least-squares solutions in a product space, IEEE Trans. Signal Process. 42(11) (1994) 2955-2966.

[26] P.L. Combettes, V.R. Wajs, Signal recovery by proximal forward-backward splitting, Multiscale Model. Simulat. 4 (2005) 1168-1200.
[27] P.L. Combettes, J.-C. Pesquet, A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery, IEEE J. Select. Topics Signal Process. 1 (2007) 564-574.
[28] P.L. Combettes, J.-C. Pesquet, Proximal splitting methods in signal processing, in: H.H. Bauschke, R.S. Burachik, P.L. Combettes, V. Elser, D.R. Luke, H. Wolkowicz (Eds.), Fixed-Point Algorithms for Inverse Problems in Science and Engineering, Springer-Verlag, 2011, pp. 185-212.
[29] J.R. Deller, Set-membersip identification in digital signal processing, IEEE Signal Process. Mag. 6 (1989) 4-20.
[30] F. Deutsch, Best Approximation in Inner Product Spaces, CMS, Springer, 2000.
[31] L. Devroye, L. Györfi, G. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer, 1991.
[32] P.S.R. Diniz, S. Werner, Set-membership binormalized data-reusing LMS algorithms, IEEE Trans. Signal Process. 52(1) (2003) 124-134.
[33] P.S.R. Diniz, Adaptive Filtering: Algorithms and Practical Implementation, fourth ed., Springer Verlag, 2014.
[34] P.S.R. Diniz, Convergence performance of the simplified set-membership affine projection algorithm, J. Circuits Syst. Signal Process. 30(2) (2011) 439-462.
[35] J. Duchi, S.S. Shwartz, Y. Singer, T. Chandra, Efficient projections onto the ℓ_1-ball for learning in high dimensions, in: Proceedings of the International Conference on Machine Leaning (ICML), 2008, pp. 272-279.
[36] J. Duchi, Y. Singer, Efficient online and batch learning using forward backward splitting, J. Machine Learn. Res. 10 (2009) 2899-2934.
[37] J. Duchi, S. Shalev-Shwartz, Y. Singer, A. Tewari, Composite objective mirror descent, in: Proceedings of the 23rd Annual Conference on Computational Learning Theory, 2010.
[38] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Machine Learn. Res. 12 (2011) 2121-2159.
[39] M. Fortin, R. Glowinski, Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems, Elsevier Science, Amsterdam; North-Holland, 1983.
[40] A.A. Goldstein, Convex Programming in Hilbert spaces, Bull. Amer. Math. Soc. 70(5) (1964) 709-710.
[41] L.G. Gubin, B.T. Polyak, E.V. Raik, The method of projections for finding the common point of convex sets, USSR Comput. Math. Phys. 7(6) (1967) 1-24.
[42] E. Hazan, A. Agarwal, S. Kale, Logarithmic regret algorithms for online convex optimization, Machine Learn. 69(2-3) (2007) 169-192.
[43] J.B. Hiriart-Urruty, C. Lemarechal, Convex Analysis and Minimization Algorithms, Springer-Verlag, Berlin, 1993.
[44] J. Kivinen, A.J. Smola, R.C. Williamson, Online learning with kernels, IEEE Trans. Signal Process. 52(8) (2004) 2165-2176.
[45] N. Komodakis, J.-C. Peusquet, Playing with duality: An overview of recent primal-dual approaches for solving large scale optimization problems, arXiv:1406.5429v1 [cs.NA] 20 June 2014, 2014.
[46] Y. Kopsinis, K. Slavakis, S. Theodoridis, Online sparse system identification and signal reconstruction using projections onto weighted ℓ_1-balls, IEEE Trans. Signal Process. 59(3) (2011) 936-952.
[47] J. Langford, L. Li, T. Zhang, Sparse online learning via truncated gradient, J. Machine Learn. Res. 10 (2009) 747-776.
[48] Y. LeCun, L. Bottou, G.B. Orr, K.R. Müller, Efficient BackProp, in: G.B. Orr, K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade, Springer, 1998, pp. 9-50.
[49] N. Le Roux, M. Schmidt, F. Bach, A stochastic gradient method with an exponential convergence rate for finite training sets, arXiv:1202.6258v4 [math.OC], 2013.

[50] W.S. Lee, P.L. Bartlett, R.C. Williamson, The importance of convexity in learning with squared loss, IEEE Trans. Informat. Theory 44(5) (1998) 1974-1980.
[51] B.S. Levitin, B.T. Polyak, Constrained minimization methods, Zhurnal Vychishitel noi Mathematik Matematicheskoi Fiziki 6(5) (1966) 787-823.
[52] D.D. Lewis, Y. Yang, T.G. Rose, F. Li, RCV1: a new benchmark collection for text categorization research, J. Machine Learn. Res. 5 (2004) 361-397.
[53] P. Lions, B. Mercier, Splitting algorithms for the sum of two nonlinear operators, SIAM J. Numer. Anal. 16 (1979) 964-979.
[54] P.E. Maingé, Strong convergence of projected subgradient methods for nonsmooth and nonstrictly convex minimization, Set-Valued Anal. 16 (2008) 899-912.
[55] N. Murata, S. Amari, Statistical analysis of learning dynamics, Signal Process. 74(1) (1999) 3-28.
[56] B. Martinet, Regularisation d' inequations variationnelles par approximations successives, Revue Francaise de Informatique et Recherche Operationelle 4 (1970) 154-158.
[57] C. Moler, Iterative refinement in floating point, J. ACM 14(2) (1967) 316-321.
[58] J.J. Moreau, Fonctions convexes duales et points proximaux dans un espace Hilbertien, Rep. Paris Acad. Sci. A 255 (1962) 2897-2899.
[59] J.J. Moreau, Proximité et dualité dans un espace hilbertien, Bull. Soc. Math. France 93 (1965) 273-299.
[60] S. Nagaraj, S. Gollamudi, S. Kapoor, Y.F. Huang, BEACON: An adaptive set-membership filtering technique with sparse updates, IEEE Trans. Signal Process. 47(11) (1999) 2928-2941.
[61] A. Nemirovsky, D. Yudin, Problem Complexity and Method Efficiency in Optimization, J. Wiley & Sons, New York, 1983.
[62] Y.E. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Soviet Math. Dokl. 27 (2) (1983) 372-376.
[63] Y. Nesterov, Primal-dual subgradient methods for convex problems, Math. Program. 120(1) (2009) 221-259.
[64] N. Parish, S. Boyd, Proximal Algorithms, Found. Trends Oprim. 1(2) (2013) 123-231.
[65] G. Pierra, Decomposition through formalization in a product space, Math. Program. 8 (1984) 96-115.
[66] T. Poggio, S. Voinea, L. Rosasco, Online learning, stability and stochastic gradient descent, arXiv: 1105.4701 [cs.LG], Sep 2011.
[67] B.T. Polyak, Minimization of unsmooth functionals, Zhurnal Vychishitel noi Mathematik Matematicheskoi Fiziki 9(3) (1969) 509-521.
[68] R.T. Rockafellar, Convex Analysis, Princeton University Press, Princeton, NJ, 1970.
[69] R.T. Rockafellar, Monotone operators and the proximal point algorithm, SIAM J. Control Optim. 14 (1976) 877-898.
[70] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, PEGASOS: primal estimated sub-gradient solver for SVM, Math. Programm. B 127 (2011) 3-30.
[71] N.Z. Shor, On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems, PhD Thesis, Cybernetics Institute, Academy of Sciences, Kiev, 1964.
[72] N.Z. Shor, Minimization Methods for Non-differentiable Functions, Springer Series in Computational Mathematics, Springer, 1985.
[73] K. Slavakis, I. Yamada, N. Ogura, The adaptive projected subgradient method over the fixed point set of strongly attracting nonexpansive mappings, Numer. Funct. Anal. Optim. 27(7-8) (2006) 905-930.
[74] K. Slavakis, I. Yamada, Robust wideband beaamforming by the hybrid steepest descent method, IEEE Trans. Signal Process. 55(9) (2007) 4511-4522.
[75] K. Slavakis, S. Theodoridis, I. Yamada, Online classification using kernels and projection-based adaptive algorithms, IEEE Trans. Signal Process. 56(7) (2008) 2781-2797.
[76] K. Slavakis, S. Theodoridis, Sliding window generalized kernel affine projection algorithm using projection mappings, EURASIP J. Adv. Signal Process. 2008, Article ID 830381, 2008. doi: 10.1155/2008/830381.

[77] K. Slavakis, S. Theodoridis, I. Yamada, Adaptive constrained learning in reproducing kernel Hilbert spaces, IEEE Trans. Signal Process. 5(12) (2009) 4744-4764.
[78] K. Slavakis, I. Yamada, The adaptive projected subgradient method constrained by families of quasi-nonexpansive mappings and its application to online learning, SIAM J. Optim. 23(1) (2013) 126-152.
[79] K. Slavakis, A. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, in: S. Theodoridis, R. Chellapa (Eds.), E-reference for Signal Processing, Academic Press, 2013.
[80] K. Slavakis, Y. Kopsinis, S. Sheodoridis, S. McLaughlin, Generalized thresholding and online sparsity-aware learning in a union of subspaces, IEEE Trans. Signal Process. 61(15) (2013) 3760-3773.
[81] K. Slavakis, Personal Communication, March 2014.
[82] H. Stark, Y. Yang, Vector Space Projections, John Wiley, 1998.
[83] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.
[84] S. Theodoridis, K. Slavakis, I. Yamada, Adaptive learning in a world of projections, IEEE Signal Processing Magazine, Vol 28(1) ()97-123, 2011.
[85] M. Todd, Semidefinite Optimization, Acta Numerica 10 (2001) 515-560.
[86] O.P. Tseng, An incremental gradient (projection) method with momentum term and adaptive step size rule, SIAM J. Optim. 8(2) (1998) 506-531.
[87] A.B. Tsybakov, Optimal aggregation of classifiers in statistical learning, Ann. Stat. 32(1) (2004) 135-166.
[88] Y. Tsypkin, Foundation of the Theory of Learning Systems, Academic Press, 1973.
[89] V.N. Vapnik, Estimation of Dependences Based on Empirical Data, Springer Series in Statistics, Springer-Verlag, Berlin 1982.
[90] V.N. Vapnik, Statistical Learning Theory, John Wiley & Sons, 1998.
[91] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, 2000.
[92] R.S. Varga, Matrix Iterative Analysis, second ed., Springer-Verlag, New York, 2000.
[93] J. von Neumann, Functional operators, Vol II. The geometry of orthogonal spaces, Ann. Math. Stud. 22, Princeton Univ. Press, NJ, 1950 (Reprint of lecture notes first distributed in 1933).
[94] S. Werner, P.S.R. Diniz, Set-membership affine projection algorithm, IEEE Signal Process. Lett. 8(8) (2001) 231-235.
[95] L. Xiao, Dual averaging methods for regularized stochastic learning and online optimization, J. Machine Learn. Res. 11 (2010) 2543-2596.
[96] I. Yamada, The hybrid steepest descent method for the variational inequality problem over the intersection of fixed point sets of nonexpansive mappings, Stud. Comput. Math. 8 (2001) 473-504.
[97] I. Yamada, K. Slavakis, K. Yamada, An efficient robust adaptive filtering algorithm based on parallel subgradient projection techniques, IEEE Trans. Signal Process. 50(5) (2002) 1091-1101.
[98] I. Yamada, Adaptive projected subgradient method: a unified view of projection based adaptive algorithms, J. IEICE 86(6) (2003) 654-658 (in Japanese).
[99] I. Yamada, N. Ogura, Adaptive projected subgradient method for asymptotic minimization of nonnegative convex functions, Numer. Funct. Anal. and Optim. 25(7-8) (2004) 593-617.
[100] I. Yamada, N. Ogura, Hybrid steepest descent method for variational inequality problem over the fixed point set of certain quasi-nonexpansive mappings, Numer. Funct. Anal. Optim. 25 (2004) 619-655.
[101] I. Yamada, S. Gandy, M. Yamagishi, Sparsity-aware adaptive filtering based on Douglas-Rachfold splitting, in: Proceedings, 19th European Signal Processing Conference (EUSIPCO), Barcelona, Spain, 2011.
[102] M. Yamagishi, M. Yukawa, I. Yamada, Acceleration of adaptive proximal forward-backward slitting method and its application in systems identification, in: Proceedings IEEE International Conference on Acoustics Speech and Signal processing, ICASSP, Prague, 2011.
[103] I. Yamada, M. Yukawa, M. Yamagishi, Minimizing the Moreau envelope of non-smooth convex functions over the fixed point set of certain quasi-nonexpansive mappings, in: H.H. Bauschke, R.S. Burachik, P.L. Combettes, V. Elser, D.R. Luke, H. Wolkowicz (Eds.), Fixed-Point Algorithms for Inverse Problems in Science and Engineering, Springer, New York, 2011, pp. 345-390.

[104] K. Yosida, Functional Analysis, Springer, 1968.

[105] T. Zhang, Solving large scale linear prediction problems using stochastic gradient descent algorithms, in: Proceedings of the 21st International Conference on Machine Learning, ICML, Banff, Alberta, Canada, 2004, pp. 919-926.

[106] M. Zinkevich, Online convex programming and generalized infnitesimal gradient ascent, in: Proceedings of the 20th International Conference on Machine Learning (ICML), Washington, DC, 2003.