exponentially to a stationary distribution, known as the *Gibbs measure* [PSV77]:

$$p^\lambda(\boldsymbol{x}) = C^\lambda \exp\left(-\frac{1}{\lambda} f(\boldsymbol{x})\right), \tag{9.5.2}$$

where $C^\lambda > 0$ is a normalizing factor such that $\int_{\boldsymbol{x}} p^\lambda(\boldsymbol{x}) d\boldsymbol{x} = 1$. We are interested in what the density $p^\lambda(\boldsymbol{x})$ converges to, as the variance of the noise λ goes from small to zero.

The Most Basic Case.
To this end, we recall a well-known result from calculus:

LEMMA 9.12 (Laplace's Method: Scalar Case). *Suppose $f(x)$ is a twice continuously differentiable function with a unique maximizer x_0 and $f''(x_0) < 0$. Then we have*

$$\lim_{\lambda \to 0} \int e^{\frac{1}{\lambda} f(x)} dx \;=\; e^{\frac{1}{\lambda} f(x_0)} \sqrt{\frac{2\pi\lambda}{-f''(x_0)}} \;\propto\; \int e^{\frac{1}{\lambda} f(x)} \delta(x - x_0) dx. \tag{9.5.3}$$

Proof We here give a sketch of the proof that illustrates the reason why this is expected. We leave a more rigorous derivation and proof for the multivariate case (below) to the reader as exercises.

Since x_0 is a maximizer, we have $f'(x_0) = 0$. So with Taylor expansion, we may approximate the function up to second-order:

$$f(x) \approx f(x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2.$$

Then for the integral we have:

$$\begin{aligned} \int e^{\frac{1}{\lambda} f(x)} dx &\approx e^{\frac{1}{\lambda} f(x_0)} \int e^{\frac{1}{2\lambda} f''(x_0)(x - x_0)^2} dx \\ &= e^{\frac{1}{\lambda} f(x_0)} \int e^{-\frac{1}{2\lambda} |f''(x_0)|(x - x_0)^2} dx. \end{aligned}$$

Notice that the last integral is exactly a Gaussian integral with variance $\sigma^2 = \lambda / |f''(x_0))$ hence its value is $\sqrt{\frac{2\pi\lambda}{|f''(x_0)|}}$. So we have

$$\int e^{\frac{1}{\lambda} f(x)} dx \approx e^{\frac{1}{\lambda} f(x_0)} \sqrt{\frac{2\pi\lambda}{-f''(x_0)}}.$$

As $\lambda \to 0$ the approximation becomes exact in the sense that the ratio between the two sides approaches to 1. □

Based on this Lemma, when λ becomes small, the integral on the left hand side is well approximated by a point-mass distribution at the global maximizer x_0, and it has nothing to do with any other values (including local maximizers) of $f(x)$.

Multiple Global Optima.
As we have seen in Chapter 7, due to discrete symmetry, the objective functions we try to optimize often have multiple global optima, associated with the elements of the symmetry group (see Figure 7.3). It is easy to generalize the above lemma to this case. Suppose $f(x)$ has multiple global maximizers $x_1, \ldots, x_N \in \mathbb{R}$. We then have:

$$\lim_{\lambda \to 0} \int e^{\frac{1}{\lambda} f(x)} dx \;=\; \sum_{i=1}^{N} e^{\frac{1}{\lambda} f(x_i)} \sqrt{\frac{2\pi\lambda}{-f''(x_i)}}. \tag{9.5.4}$$

We leave the proof as an exercise to the reader, see Exercise 9.6. Notice that the integral above is very similar in style to

$$\int_{\boldsymbol{x}} p^{\lambda}(\boldsymbol{x}) d\boldsymbol{x} \;\propto\; \int_{\boldsymbol{x}} \exp\left(-\frac{1}{\lambda} f(\boldsymbol{x})\right) d\boldsymbol{x}$$

as $\lambda \downarrow 0$, except that $-f(\boldsymbol{x})$ here is a multivariate function with possibly multiple global maximizers at $\boldsymbol{x}_\star^1, \ldots, \boldsymbol{x}_\star^N$, corresponding to the multiple global minimizers of $f(\boldsymbol{x})$. Then one can show that, in this case, we have the following statement that generalizes the above lemma:

THEOREM 9.13 (Laplace Method: Multivariate and Multiple Global Minimizers). *Let $f(\boldsymbol{x})$ be a function with at least quadratic growth as $\boldsymbol{x} \to \infty$. Suppose $f(\boldsymbol{x})$ has multiple global minimizers at $\boldsymbol{x}_\star^1, \ldots, \boldsymbol{x}_\star^N$ and they are all non-degenerate. Then in the limit $\lambda \downarrow 0$, the density $p^{\lambda}(\boldsymbol{x})$ of the noisy gradient descent dynamics (9.5.1) converges to*

$$p^0(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} a_i \delta(\boldsymbol{x} - \boldsymbol{x}_\star^i)}{\sum_{i=1}^{N} a_i}, \quad \textit{with} \quad a_i = \det[\boldsymbol{H}(\boldsymbol{x}_\star^i)]^{-1/2}, \tag{9.5.5}$$

where $\boldsymbol{H}(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ is the Hessian of the function $f(\boldsymbol{x})$.

A Continuous Family of Global Optima.
As we have seen in Chapter 7, sometimes a nonconvex function $f(\boldsymbol{x})$ may have a continuous family of global minimizers, say due to rotational symmetry (see Figure 7.3). The above theorem also generalizes naturally to this case. Let us assume that the set of all minimizers make a continuous submanifold $\mathcal{M}$, and the Hessian of the function is non-degenerate along the directions orthogonal to the submanifold.[22] For simplicity, we still use $\boldsymbol{H}(\boldsymbol{x})$ to denote the Hessian restricted to the orthogonal directions to the submanifold (tangent space) at any global minimizer $\boldsymbol{x} \in \mathcal{M}$. In this case, the Gibbs distribution $p^{\lambda}(\boldsymbol{x})$ converges to a density on $\mathcal{M}$ given by:

$$p^0(\boldsymbol{x}) = \frac{\det[\boldsymbol{H}(\boldsymbol{x})]^{-1/2}}{\int_{\mathcal{M}} \det[\boldsymbol{H}(\boldsymbol{y})]^{-1/2} d\boldsymbol{y}}, \quad \boldsymbol{x} \in \mathcal{M}, \tag{9.5.6}$$

and $d\boldsymbol{y}$ is the naturally induced metric on $\mathcal{M}$.

[22] Such a function is called a Morse-Bott function in differential geometry.

A simple proof of Theorem 9.13 for the multivariate case, or the case with multiple global minimizers, or the case with a family of global minimizers can be found in [Sas83], which is very much in the same spirit of Lemma 9.12 for the scalar case. Only here that the second-order derivative is naturally replaced by the determinant of the Hessian. We leave the details to the reader as an exercise, see Exercise 9.6.

The above theorem states an interesting fact: under the noisy gradient flow (9.5.1), as the noise variance λ is gradually reduced to nearly zero, the density of the state converges to a point-mass distribution with support only on the global minimizers of the function $f(\boldsymbol{x})$. Historically, the above phenomenon has motivated optimization methods that leverage random noise for nonconvex optimization, including the well-known *simulated annealing* [KGV83].

Although the above theorem reveals a nice qualitative behavior of noisy gradient descent, it *by no means* suggests that this behavior can be exploited effectively and efficiently for optimization. In fact, in order for the diffusion process to converge to the density with support on the global minimizers, the noise variance λ needs to be reduced to zero *exponentially slowly* in time t [GH86, CHS87]:

$$\lambda = \frac{c}{\log t} \quad \text{for large } t \text{ and } c > 0.$$

9.5.2 Noisy Gradient with Langevin Monte Carlo

Inspired by properties of the above diffusion process, to minimize a function $f(\boldsymbol{x})$, one may consider a discrete approximation to the noisy gradient flow (9.5.1). The resulting discrete process is known as *Langevin Monte Carlo*:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) + \sqrt{2\alpha\lambda}\boldsymbol{n}_k, \tag{9.5.7}$$

where $\boldsymbol{n}_k \sim \mathcal{N}(0, \boldsymbol{I})$ is i.i.d. Gaussian noise and $\alpha > 0$ is a step size (correlated with the noise level). It can be shown that if the discretization is done properly, the above discrete Langevin process can asymptotically converge to the same Gibbs stationary distribution as in the continuous case mentioned above [RT96].[23] Algorithms based on the above discrete stochastic process for optimization have been long proposed and studied in the literature of stochastic control and optimization [Kus87, GM90]. Below we try to illustrate the fundamental rationale behind such schemes through analysis of the most basic cases.

To simplify the analysis, as in the previous section, we assume again that $f : \mathbb{R}^n \to \mathbb{R}$ is nonconvex, and is twice continuously differentiable, with Lipschitz continuous gradient $\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$ with Lipschitz constant L_1. Notice that if we

[23] A few words of caution though: the relationship between the continuous diffusion (9.5.1) and the discrete approximation (9.5.7) can be subtle. Even if the original diffusion converge, naive discretizations need not to. Or even if the original diffusion converges exponentially quickly to its stationary distribution, discretized versions need not to converge exponentially fast. For details of proper discretizing of the Langevin dynamics, one may refer to [RT96].

choose the step size to be the Lipschitz constant $\alpha = 1/L_1$, then the above scheme becomes

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{1}{L_1}\nabla f(\boldsymbol{x}_k) + \sqrt{2\lambda/L_1}\boldsymbol{n}_k. \tag{9.5.8}$$

Now let us consider a similar setting as in the negative curvature descent scheme studied in Theorem 9.5, with a prescribed precision $\varepsilon > 0$.[24] Then we have the following statement regarding the above noisy gradient descent scheme:

PROPOSITION 9.14 (Noisy Gradient Descent). *Considering the above noisy gradient descent scheme* (9.5.8)*, if* $\|\nabla f(\boldsymbol{x}_k)\|_2 \geq (2L_1\varepsilon)^{1/2}$*, then we have*

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) \mid \boldsymbol{x}_k] \leq f(\boldsymbol{x}_k) - \varepsilon + \lambda. \tag{9.5.9}$$

Proof From the Lipschitz condition, we have

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle + \frac{L_1}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2.$$

Also from the iteration (9.5.8), we have $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = -\frac{1}{L_1}\nabla f(\boldsymbol{x}_k) + \sqrt{2\lambda/L_1}\boldsymbol{n}_k$. So we have

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) &+ \langle \nabla f(\boldsymbol{x}_k), -\frac{1}{L_1}\nabla f(\boldsymbol{x}_k) + \sqrt{2\lambda/L_1}\boldsymbol{n}_k \rangle \\ &+ \frac{L_1}{2}\Big\|\frac{1}{L_1}\nabla f(\boldsymbol{x}_k) - \sqrt{2\lambda/L_1}\boldsymbol{n}_k\Big\|_2^2. \end{aligned}$$

Take the conditional expectation on both sides, we have

$$\begin{aligned} \mathbb{E}[f(\boldsymbol{x}_{k+1}) \mid \boldsymbol{x}_k] &\leq f(\boldsymbol{x}_k) - \frac{1}{L_1}\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \frac{1}{2L_1}\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \lambda \\ &= f(\boldsymbol{x}_k) - \frac{1}{2L_1}\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \lambda \\ &\leq f(\boldsymbol{x}_k) - \varepsilon + \lambda. \end{aligned}$$

□

This proposition reveals a simple and important relationship between the optimization precision ε and the noise variance λ. It has several implications. On one hand, it ensures that as long as the gradient is strictly over the threshold $\|\nabla f(\boldsymbol{x}_k)\|_2 > (2L_1\lambda)^{1/2}$, the noisy gradient descent scheme reduces the expected function value per iteration. Or equivalently, as long as we choose the noise level adaptively according to:

$$\lambda_k < \frac{1}{2L_1}\|\nabla f(\boldsymbol{x}_k)\|_2^2,$$

the scheme is expected to be descending always. On the other hand, if one uses a fixed noise variance $\lambda > 0$, then whenever the iterates approach to a critical point with gradient dropping below the threshold

$$\|\nabla f(\boldsymbol{x}_k)\|_2 < (2L_1\lambda)^{1/2},$$

[24] That is, we desire to eventually achieve $|f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)| \leq \varepsilon$.

the random effect starts to take over and to explore if the critical point is stable. This mechanism allows the noisy descent algorithm to escape from unstable critical points such as saddle points, as we will elucidate further below.

9.5.3 Negative Curvature Descent with Random Noise

Despite the asymptotic consistency, there is *no theoretical guarantee* that the Langevin Monte Carlo (9.5.7) is able to find the global minima of a general nonconvex functions *in polynomial time*. In fact, according to [BEGK11], it takes the Langevin diffusion at least $e^{\Omega(h/\lambda)}$ time to escape any local minima of height $h > 0$. This implies that for functions that contain deep local minima, it is unavoidable for noisy gradient descent to take *exponentially long* time to escape before finding global ones. Hence, contrary to our earlier hope, it is actually computationally intractable to use this method (alone) to find global minima of general nonconvex functions!

Dynamics of Noisy Gradient Descent around a Strict Saddle Point. It seems that noise is no magical sauce and there is *no free lunch* at all when it comes to solving general nonconvex optimization problems. Then what can noisy gradient descent methods actually end up with helping in practice with nonconvex optimization then? As it turns out, random noise helps gradient descent to escape non-stationary critical points, such as saddle points, efficiently.[25] As we have seen in the negative curvature descent methods, any non-degenerate saddle point has a direction with strict negative curvature. Intuitively such a point is very "unstable" (as shown in the Figure 7.2), and any random perturbation would drive the state away from it. The only question is how quickly this may take place, say under the noisy gradient descent scheme.

To see this, notice that without loss of generality, we may consider dynamics of the noisy gradient descent around the critical point $\boldsymbol{x} = \boldsymbol{0}$ of the standard quadratic function[26]

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^* \boldsymbol{H} \boldsymbol{x}$$

for a constant $\boldsymbol{H} \in \mathbb{R}^{n\times n}$, with the smallest eigenvalue $\lambda_{\min} < 0$, and the Lipschitz constant $L_1 = \max_i |\lambda_i(\boldsymbol{H})|$.

Proposition 9.15 (Escaping Saddle Point via Noisy Gradient Descent). *Consider the noisy gradient descent via the Langevin dynamics* (9.5.8) *for the function* $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^* \boldsymbol{H} \boldsymbol{x}$, *starting from* $\boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$. *Then after*

$$k \geq \frac{\log n - \log(|\lambda_{\min}|/L_1)}{2\log(1 + |\lambda_{\min}|/L_1)} \tag{9.5.10}$$

[25] Hence, this ensures that the process converges at least to local minima, in *polynomial time* [ZLC17].

[26] Since we only care about local behaviors, any nonconvex function is diffeomorphic to this standard form around a non-degenerate critical point $\boldsymbol{x}_\star$ with Hessian $\boldsymbol{H}(\boldsymbol{x}_\star)$.

steps, we have

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0)] \leq -\lambda. \tag{9.5.11}$$

Proof Notice that for this function, the Lipschitz constant for the gradient L_1 is exactly the spectral norm of the Hessian $\boldsymbol{H}$. So the Langevin dynamics (9.5.8) becomes:

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \frac{1}{L_1}\nabla f(\boldsymbol{x}_k) + \sqrt{2\lambda/L_1}\boldsymbol{n}_k \\ &= (\boldsymbol{I} - L_1^{-1}\boldsymbol{H})\boldsymbol{x}_k + \sqrt{2\lambda/L_1}\boldsymbol{n}_k.\end{aligned}$$

Notice that the matrix $\boldsymbol{A} \doteq \boldsymbol{I} - L_1^{-1}\boldsymbol{H}$ has eigenvalues outside of the unit circle if and only if the Hessian has a negative eigenvalue $\lambda_{\min} < 0$:

$$\lambda_{\max}(\boldsymbol{A}) = 1 - \frac{\lambda_{\min}(\boldsymbol{H})}{L_1} > 1.$$

This defines *an unstable linear dynamic system* with random noise as the input:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + b\,\boldsymbol{n}_k, \tag{9.5.12}$$

with $b \doteq \sqrt{2\lambda/L_1}$. Therefore we have

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}^{k+1}\boldsymbol{x}_0 + b\sum_{i=0}^{k}\boldsymbol{A}^{k-i}\boldsymbol{n}_i. \tag{9.5.13}$$

Notice that all the terms $\boldsymbol{A}^{k+1}\boldsymbol{x}_0$ and $\boldsymbol{A}^{k-i}\boldsymbol{n}_i$ on the right hand side are nothing but powers of the matrix $\boldsymbol{A}$ applied to a (random) vector.

From *the power iteration method* that we have seen in Section 9.3.2, as the power increases, each of these term converges to the eigenvector of the largest eigenvalue of $\boldsymbol{A}$,[27] or equivalently the eigenvector of the smallest (negative) eigenvalue of $\boldsymbol{H}$. That is exactly the direction of the most negative curvature of $f(\boldsymbol{x})$ that we have computed before in Section 9.3.2. Hence when the gradient is small, the noisy gradient descent implicitly performs negative curvature descent, exactly in the same spirit as the gradient and negative curvature descent algorithm in Figure 9.2.

Now we only have to turn the state evolution (9.5.13) to a bound for the descent of the expected value of the function $f(\boldsymbol{x})$. Let $\{\lambda_j\}_{j=1}^n$ be the n eigenvalues of the Hessian $\boldsymbol{H}$, sorted from the largest to the smallest. Notice that $\boldsymbol{A}$ and $\boldsymbol{H}$ share the same eigenvectors and can be diagonalized by the same orthogonal transform, the corresponding eigenvalues of $\boldsymbol{A}$ are $\{1 - \frac{\lambda_j}{L_1}\}_{j=1}^n$. Since $\boldsymbol{x}_0$ and $\boldsymbol{n}_k$

[27] We will also characterize the geometry of the landscape of the objective function for power iteration more precisely in Section 9.6.

are all independent zero mean random variables, we have

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0)] \;=\; \mathbb{E}\Big[\frac{1}{2}\boldsymbol{x}_{k+1}^*\boldsymbol{H}\boldsymbol{x}_{k+1} - \frac{1}{2}\boldsymbol{x}_0^*\boldsymbol{H}\boldsymbol{x}_0\Big]$$
$$= \frac{1}{2}\sigma^2\mathrm{trace}\left(\boldsymbol{A}^{2(k+1)}\boldsymbol{H}\right) + \frac{1}{2}b^2\sum_{i=0}^{k}\mathrm{trace}\left(\boldsymbol{A}^{2(k-i)}\boldsymbol{H}\right) - \frac{1}{2}\sigma^2\mathrm{trace}\left(\boldsymbol{H}\right).$$

For the first and third terms related to the initial condition $\boldsymbol{x}_0$, we have

$$\frac{1}{2}\sigma^2\mathrm{trace}\left(\boldsymbol{A}^{2(k+1)}\boldsymbol{H}\right) - \frac{1}{2}\sigma^2\mathrm{trace}\left(\boldsymbol{H}\right)$$
$$= \frac{1}{2}\sigma^2\sum_{j=1}^{n}\Big[\big(1-\frac{\lambda_j}{L_1}\big)^{2(k+1)}\lambda_j - \lambda_j\Big] \;\le\; 0$$

because $1-\frac{\lambda_j}{L_1}$ is smaller than 1 when λ_j is positive and larger than 1 when λ_j is negative. So without the random noise, the deterministic part of the system $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k$ always leads to descending in the objective value regardless of initial condition!

So we have

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0)] \le \frac{1}{2}b^2\sum_{i=0}^{k}\mathrm{trace}\left(\boldsymbol{A}^{2(k-i)}\boldsymbol{H}\right)$$
$$= \frac{1}{2}b^2\sum_{j=1}^{n}\Big(\sum_{i=0}^{k}\Big(1-\frac{\lambda_j}{L_1}\Big)^{2(k-i)}\lambda_j\Big).$$

Notice that we have

$$\sum_{i=0}^{k}\Big(1-\frac{\lambda_j}{L_1}\Big)^{2(k-i)}\lambda_j \le L_1, \quad \text{when } \lambda_j > 0;$$
$$\sum_{i=0}^{k}\Big(1-\frac{\lambda_j}{L_1}\Big)^{2(k-i)}\lambda_j < 0, \quad \text{when } \lambda_j < 0.$$

There are at most $n-1$ positive eigenvalues. Since $b = \sqrt{2\lambda/L_1}$, we have

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0)] \le \frac{1}{2}b^2\Big((n-1)L_1 + \lambda_{\min}\sum_{i=0}^{k}\Big(1-\frac{\lambda_{\min}}{L_1}\Big)^{2i}\Big)$$
$$\le \lambda\Big((n-1) + \frac{\lambda_{\min}}{L_1}\Big(1-\frac{\lambda_{\min}}{L_1}\Big)^{2k}\Big).$$

To have

$$\frac{\lambda_{\min}}{L_1}\Big(1-\frac{\lambda_{\min}}{L_1}\Big)^{2k} \le -n,$$

we only need to choose

$$k \ge \frac{\log n - \log(|\lambda_{\min}|/L_1)}{2\log(1+|\lambda_{\min}|/L_1)}. \tag{9.5.14}$$

With this choice of number of noisy descent iterations around the saddle point, we have

$$\mathbb{E}[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0)] \leq -\lambda. \tag{9.5.15}$$

□

In fact, one can see from the above expression for k, the number of iteration needed increases when the ratio $\kappa = L_1/|\lambda_{\min}|$ is large. In this case $\log(1 + |\lambda_{\min}|/L_1) \approx |\lambda_{\min}|/L_1 = \kappa^{-1}$. So from (9.5.14), the number of noisy gradient steps required to achieve the desired descent λ is simplified to:

$$k \geq \frac{\kappa}{2}\log(n).$$

Stopping Criteria.
Notice that the above lower bound for the number of noisy descent steps k is monotonic in $|\lambda_{\min}| = -\lambda_{\min}$: the smaller is $|\lambda_{\min}|$, the larger k needs to be. Then without computing and knowing the smallest eigenvalue $\lambda_{\min}$ of the Hessian $\boldsymbol{H}$, *how do we know what k to use and when should we stop, once the curvature becomes nearly non-negative?* Answers to these questions can be tricky if we do not resort to any explicit process to estimate $\lambda_{\min}$.

From the proof of the above Proposition 9.15, the noisy gradient descent essentially conducts negative curvature descent implicitly through power iteration on noise. If we choose the noise variance λ in the noisy gradient descent to be the same as the prescribed precision ε for the function value (as in Section 9.3.2):

$$\lambda = \varepsilon,$$

then k noisy gradient descent iterations are equivalent to one step of deterministic negative curvature descent (as characterized by Theorem 9.5.)

Following the same line of arguments in Theorem 9.5, as long as we have:

$$-\lambda_{\min}(\boldsymbol{H}) \geq \varepsilon_H = \left(1.5L_2^2\varepsilon\right)^{1/3},$$

we should expect to achieve a descent amount of $\lambda = \varepsilon$. So using $\varepsilon_H = \left(1.5L_2^2\varepsilon\right)^{1/3}$ as a lower bound[28] for $|\lambda_{\min}|$, we get an estimate of the number of noisy gradient descent needed:

$$k_{\max} \geq \frac{\log n - \log\left(L_1^{-1}\left(1.5L_2^2\varepsilon\right)^{1/3}\right)}{2\log\left(1 + L_1^{-1}\left(1.5L_2^2\varepsilon\right)^{1/3}\right)}. \tag{9.5.16}$$

Hence, if, after $k_{\max}$ noisy gradient descent, the function value drops less than ε, that indicates the minimum eigenvalue should have reached the desired threshold:

$$-\lambda_{\min}(\boldsymbol{H}) \leq \varepsilon_H = \left(1.5L_2^2\varepsilon\right)^{1/3}, \tag{9.5.17}$$

and the critical point reached is an approximate second-order stationary point.

[28] Notice that for the standard quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^*\boldsymbol{H}\boldsymbol{x}$, the Lipschitz constant L_2 can be as small as zero. However, for a general function, L_2 is not and we can choose any nonzero upper bound for this constant.

Hybrid Noisy Gradient Descent

Problem Class:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^n,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is nonconvex, and is twice continuously differentiable, with Lipschitz continuous gradient and Hessian with constants L_1 and L_2, respectively. Have access to the oracle: gradient $\nabla f(\boldsymbol{x})$ and random noise $\boldsymbol{n}$.

Setup: given a prescribed accuracy $\varepsilon > 0$, $\varepsilon_g = (2L_1\varepsilon)^{1/2}$ and $\varepsilon_H = \left(1.5L_2^2\varepsilon\right)^{1/3}$.

Initialization: Set $\boldsymbol{x}_0 \in \mathbb{R}^n$.

For $k = 0, 1, 2, \ldots$

1 Compute the gradient $\nabla f(\boldsymbol{x}_k)$.
2 **if** $\|\nabla f(\boldsymbol{x}_k)\|_2 \geq \varepsilon_g$, **then** gradient descent:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{1}{L_1}\nabla f(\boldsymbol{x}_k);$$

3 **else** $\boldsymbol{x}_k^0 = \boldsymbol{x}_k$, and negative curvature descent with noisy gradients:
for $i = 0, 1, 2, \ldots, k_{\max}$ as in (9.5.10)

$$\boldsymbol{x}_k^{i+1} = \boldsymbol{x}_k^i - \frac{1}{L_1}\nabla f(\boldsymbol{x}_k^i) + \sqrt{2\varepsilon/L_1}\boldsymbol{n}^i,$$

where $\boldsymbol{n}^i \sim \mathcal{N}(0, \boldsymbol{I})$.
end for and set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k^{i+1}$.

End for when $|f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)| \leq \varepsilon$ and return $\boldsymbol{x}_\star = \boldsymbol{x}_k$.

Convergence guarantee: $\|\nabla f(\boldsymbol{x}_\star)\| \leq \varepsilon_g$, $-\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_\star)) \leq \varepsilon_H$.

Figure 9.5 An overview of the Hybrid Noisy Gradient Descent Method.

Hybrid Noisy Gradient Descent.

As we have seen from the analysis in Section 9.5.2 and Section 9.5.3, when the gradient is large, it is not so helpful to add noise. Only when the gradient is small enough and we are near a strict saddle point, adding small random noise would help escape from it – but with a price about $O(\kappa \log n)$ noisy gradient steps to reach the same amount of descent. So to make the algorithm more efficient, we may modify the basic noisy gradient scheme with a hybrid scheme illustrated in Figure 9.5, in which we choose different descent strategies based on the local landscape. One should notice that this scheme is very similar to the hybrid gradient and negative curvature descent scheme in Figure 9.2. The only difference is that here we replace the negative curvature descent step with a sequence of $O(\kappa \log n)$ random gradient descent steps.

Optimize Overall Complexity.
As we have discussed above, around a critical point, to achieve the same amount of descent, say ε, by exploiting negative curvature using noisy gradient descent, it requires evaluating a number of, $k_{\max}$, gradients. If we use gradients as the oracle to evaluate overall complexity, the cost of the negative curvature step in the above algorithm will be more than the gradient step. From the analysis above, if we require $\lambda_{\min} \geq -O(\varepsilon^{1/3})$, to achieve ε amount of descent, we need $k_{\max} = O(\varepsilon^{-1/3}\log(n))$. Hence on average, the function value decreases per gradient evaluation is in the order of $O(\varepsilon^{-4/3}\log(n))$. So to guarantee $\|\nabla f(\boldsymbol{x})\| \leq \varepsilon_g = O(\varepsilon^{1/2})$, we need (up to a $\log(n)$ factor) $O(\varepsilon_g^{-8/3})$ number of gradients, which is actually worse than the rate $O(\varepsilon_g^{-2})$ of the gradient descent given in Proposition 9.1.

Since the negative curvature descent is much more costly, we may relax our requirements on the accuracy in the smallest eigenvalue, say from $-\lambda_{\min} \leq \varepsilon_H = O(\varepsilon^{1/3})$ to

$$-\lambda_{\min} \leq \varepsilon_H = O(\varepsilon^{1/4})$$

instead. Then the number of noisy gradient descent becomes

$$k_{\max} = O(\varepsilon^{-1/4}\log(n))$$

and the function value decreases by $O(\varepsilon^{3/4})$. On average, up to a $\log(n)$ factor, the function value decreases per gradient evaluation is $O(\varepsilon)$, the same as a gradient descent step. So to guarantee $\|\nabla f(\boldsymbol{x})\| \leq \varepsilon_g$, the number of total gradient evaluations needed is $O(\varepsilon_g^{-2})$, up to a $\log(n)$ factor.

9.5.4 Complexity of Perturbed Gradient Descent

In the above hybrid descent scheme, for simplicity and clarity of analysis, we have separated the normal gradient descent and noisy gradient descent around critical points. The hybrid scheme achieves a complexity of $O(\varepsilon_g^{-2})$. As we have seen in the previous section, the best complexity we are able to achieve is $O(\varepsilon_g^{-7/4})$. A remaining question is whether it is possible to achieve this rate with a noisy gradient descent scheme.

As we have mentioned before, the simple gradient descent is not the most efficient way to decrease the function value. The Newton descent introduced in Section 9.4.1 is precisely aiming to improve its efficiency. Nevertheless, it requires accessing or approximating the direction of negative curvature. For algorithmic simplicity, one may prefer only to conduct the (noisy) gradient descent. What else can we do to improve the efficiency of (noisy) gradient descent without explicitly computing the second-order information?

In fact, we have seen such a scheme in the context of convex optimization: *the Nesterov's acceleration* (see Section 8.3 of Chapter 8 or Section D.2 of Appendix D). The same acceleration scheme should also work for the nonconvex case (at least locally). Following this line of thought, a randomly *perturbed accelerated*

Perturbed Accelerated Gradient Descent

Problem Class:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^n,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is nonconvex, and is twice continuously differentiable, with Lipschitz continuous gradient and Hessian with constants L_1 and L_2, respectively. Have access to the oracle: gradient $\nabla f(\boldsymbol{x})$ and random noise $\boldsymbol{n}$.

Setup: given properly chosen parameters $\varepsilon_g, \varepsilon_H$, σ, s, and $k_{\min}$.

Initialization: Set the state $\boldsymbol{x}_0 \in \mathbb{R}^n$ and momentum $\boldsymbol{v}_0 = \boldsymbol{0}$.

For $k = 0, 1, 2, \ldots$

1 Compute the gradient $\nabla f(\boldsymbol{x}_k)$.

2 **If** $\|\nabla f(\boldsymbol{x}_k)\|_2 \le \varepsilon_g$ and there is no random perturbation in last $k_{\min}$ steps, **then** randomly perturb the current iterate:

$$\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k + \boldsymbol{n}_k, \quad \boldsymbol{n}_k \sim \mathcal{N}(0, \sigma \boldsymbol{I}).$$

3 Conduct accelerated gradient descent:

$$\begin{cases} \boldsymbol{p}_{k+1} &= \boldsymbol{x}_k + \beta \boldsymbol{v}_k, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{p}_{k+1} - \alpha \nabla f(\boldsymbol{p}_{k+1}), \\ \boldsymbol{v}_{k+1} &= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k. \end{cases} \tag{9.5.18}$$

4 **If**

$$f(\boldsymbol{x}_k) \le f(\boldsymbol{p}_{k+1}) + \langle \nabla(\boldsymbol{p}_{k+1}), \boldsymbol{x}_k - \boldsymbol{p}_{k+1} \rangle - \frac{\varepsilon_H}{2} \|\boldsymbol{x}_k - \boldsymbol{p}_{k+1}\|_2^2,$$

then use $\boldsymbol{v}_k$ to conduct negative curvature exploitation:

- **if** $\|\boldsymbol{v}_k\|_2 \ge s$ then $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$;
- **else** $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{\delta}$ with $\boldsymbol{\delta} = \pm s \frac{\boldsymbol{v}_k}{\|\boldsymbol{v}_k\|}$ that minimizes $f(\boldsymbol{x}_k + \boldsymbol{\delta})$.
- **reset** $\boldsymbol{v}_{k+1} = \boldsymbol{0}$.

End for.

Figure 9.6 An overview of the Perturbed Accelerated Gradient Descent Method.

gradient descent (PAGD) scheme has been proposed by [JNJ18], as illustrated in Figure 9.6.

One remarkably insightful and clever idea of this scheme is to directly leverage the momentum from the acceleration scheme, the vector $\boldsymbol{v}_k$ in Step 3 of the PAGD algorithm, as a candidate to exploit the negative curvature. This saves the effort to (approximately) compute the negative curvature direction as we have done in methods introduced earlier. By combining the random perturbation and the acceleration, careful analysis can show that the resulting scheme indeed achieves the best complexity of $O(\varepsilon_g^{-7/4})$ (saving some log factors) [JNJ18].

The reader should be aware that all the complexity guarantees characterized for all methods so far are for the worst case among a broad family of func-

tions considered.[29] As we have seen in Chapter 7, many of the problems that we encounter in low-dimensional structure recovery are much better than the worst case. Even to the opposite, the functions often have additional benign geometric structures. For instance, the objective functions have non-degenerate saddle points, the functions do not have any spurious local minima in conspicuous configurations [DJL+17], the functions are strongly convex around the minima etc. Hence, it is often observed in practice that, even much simplified vanilla versions of the randomly initialized or perturbed gradient descent can be surprisingly efficient and effective (in escaping strict saddles and converging to minimizers), far more than what is characterized for the worst case.

9.6 Leveraging Symmetry Structure: Generalized Power Iteration

This chapter so far has provided a rather systematic and complete characterization of convergence and complexity of first (and second) order methods for a very general class of (unconstrained) nonconvex programs. However, the complexities characterized are typically for the worst case (in a broad class of problems considered). In practice, very often the particular optimization problems we encounter for recovering low-dimensional models have special structures which can be exploited for much better computational efficiency. This clearly has been the case for convex optimization as we have seen in Section 8.6 where methods such as Franke-Wolfe and Stochastic Gradient Descent can be utilized to exploit the structures in the constraints and in the objective function, respectively.

In Chapter 7, we have argued that in processing structured data, nonconvexity often arises due to certain structural symmetries in the problems and the domain spaces are typically compact manifolds that are invariant under the associated symmetry group actions. Such special manifolds are known as *homogeneous spaces* in differential geometry [Lan01]. They include important cases that we have frequently encountered before: high-dimensional spheres, orthogonal groups, and Stiefel manifolds etc. The nice global geometric structures of these manifolds make them amenable to *global* analysis and computation. In this section, we illustrate several important instances for which we can go beyond the local gradient-descent type methods, and exploit more global geometric structures for more efficient optimization algorithms.

9.6.1 Power Iteration for Computing Singular Vectors

Consider the problem of computing the leading singular value-vector of a matrix $\boldsymbol{Y}$ that we have seen earlier in Chapter 4:

$$\begin{array}{ll} \min & \varphi(\boldsymbol{q}) \;\equiv\; -\frac{1}{2}\boldsymbol{q}^*\boldsymbol{\Gamma}\boldsymbol{q}, \\ \text{subject to} & \boldsymbol{q}^*\boldsymbol{q} = 1 \;\; (\text{or } \boldsymbol{q} \in \mathbb{S}^{n-1}) \end{array} \tag{9.6.1}$$

[29] Here functions with Lipschitz gradient and Hessian and only have strict saddles.

with $\mathbf{\Gamma} = \boldsymbol{Y}\boldsymbol{Y}^*$. As we have shown in Section 4.2.1, when φ is viewed as a function on the sphere, its saddle points are associated with the (ordered) eigenvalues λ_i of $\mathbf{\Gamma}$ with $\lambda_i > \lambda_{i+1}$, and we have

$$\varphi(\boldsymbol{q}(\lambda_{i+1})) > \varphi(\boldsymbol{q}(\lambda_i)), \quad \text{for } i = 1, \dots, n.$$

From the second derivative of the objective function (4.2.9), we see that we always have

$$\mathcal{S}^-[\boldsymbol{q}(\lambda_{i+1})] \supset \mathcal{S}^-[\boldsymbol{q}(\lambda_i)], \quad \text{for } i = 1, \dots, n,$$

where $\mathcal{S}^-$ indicates the unstable submanifold of a critical point. It suggests that unstable submanifolds of upstream saddle points contain the entire unstable submanifolds of the downstream saddle points. Further analysis shows that the direction towards the global minimum has the most negative curvature among all directions. Therefore, we expect most reasonable methods to converge to a global minimizer. For almost all the problems that we have studied in Chapter 7, the landscape of their objective functions has similar global geometric properties. In addition, the objective functions do not have any spurious local minima in conspicuous configurations. There are both theoretical and experimental reasons to expect standard, randomly initialized, gradient descent to converge to a small neighborhood of a global minimizer, in polynomial time.[30]

In fact, for the singular vector problem, the nice global geometry of the objective function φ may enable even more efficient methods than the vanilla gradient descent. For instance, we know, from the Lagrangian formulation of (9.6.1), the necessary condition of the critical points of φ gives

$$\nabla\varphi(\boldsymbol{q}) = \mathbf{\Gamma}\boldsymbol{q} = \lambda\boldsymbol{q}$$

for some λ (the eigenvalues of the matrix $\mathbf{\Gamma}$). Hence any critical point, including the optimal solution, is a "fixed point" to the following equation:

$$\boldsymbol{q} = \mathcal{P}_{\mathbb{S}^{n-1}}(\mathbf{\Gamma}\boldsymbol{q}) = \frac{\mathbf{\Gamma}\boldsymbol{q}}{\|\mathbf{\Gamma}\boldsymbol{q}\|_2}, \tag{9.6.2}$$

where $\mathcal{P}_{\mathbb{S}^{n-1}}$ means projection onto the sphere $\mathbb{S}^{n-1}$. If we view

$$g(\cdot) \doteq \mathcal{P}_{\mathbb{S}^{n-1}}[\mathbf{\Gamma}(\cdot)] : \mathbb{S}^{n-1} \to \mathbb{S}^{n-1}$$

as a map from $\mathbb{S}^{n-1}$ to $\mathbb{S}^{n-1}$ itself, the map is actually a contraction mapping. That is,

$$d(g(\boldsymbol{q}), g(\boldsymbol{p})) \le \rho \cdot d(\boldsymbol{q}, \boldsymbol{p})$$

for some $0 < \rho < 1$ and $d(\cdot,\cdot)$ a natural distance on the sphere. It is easy to show that here ρ is bounded from above by the ratio $\rho \le \lambda_2/\lambda_1$ where λ_2 is the second largest eigenvalue of $\mathbf{\Gamma}$. This leads to another more popular method to

[30] This has been proved for specific problems, including dictionary learning [GBW19] and generalized phase retrieval [CCFM18].

compute the eigenvector, known as the *power iteration method* (as we have seen in Exercise 4.6):

$$\boldsymbol{q}_{k+1} = g(\boldsymbol{q}_k) = \frac{\boldsymbol{\Gamma}\boldsymbol{q}_k}{\|\boldsymbol{\Gamma}\boldsymbol{q}_k\|_2} \in \mathbb{S}^{n-1}. \tag{9.6.3}$$

It can be shown that this iteration is much more efficient than gradient descent method for solving the singular vector problem (9.6.1).[31] The rate of convergence of the iteration is typically *linear*: If $\boldsymbol{q}_\star$ is a fixed point $\boldsymbol{q}_\star = g(\boldsymbol{q}_\star)$, we always have

$$d(\boldsymbol{q}_\star, \boldsymbol{q}_k) \le \rho^k \cdot d(\boldsymbol{q}_\star, \boldsymbol{q}_0).$$

That is, the error decreases geometrically according to the kth power of ρ, hence the name of the method.

9.6.2 Complete Dictionary Learning

In Chapter 7, we have introduced and studied *dictionary learning* as an important example of structured nonconvex problems. Now, consider solving a special case of dictionary learning where the dictionary is complete (i.e., square and invertible). Without loss of generality, we assume the dictionary is orthogonal[32] and we solve the problem via maximizing ℓ^4 norm: given a data matrix $\boldsymbol{Y} = \boldsymbol{D}_o\boldsymbol{X}_o$ where $\boldsymbol{D}_o$ is orthogonal and $\boldsymbol{X}_o$ is sparse, we try to recover the dictionary from solving the following optimization problem:

$$\begin{array}{ll} \min & \psi(\boldsymbol{A}) \equiv -\frac{1}{4}\|\boldsymbol{A}\boldsymbol{Y}\|_4^4, \\ \text{subject to} & \boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I} \ \ (\text{or } \boldsymbol{A} \in \mathsf{O}(n)). \end{array} \tag{9.6.4}$$

Notice that this is very similar in style to the singular vector problem (9.6.1). Unfortunately, a careful study would show that, unlike the singular vector problem, here ψ is in general *not* a Morse function on $\mathsf{O}(n)$.[33] Hence there is no guarantee that the gradient flow type algorithms would be efficient for solving this problem.

But what about the fixed point approach then? Let us consider the Lagrangian:

$$\mathcal{L}(\boldsymbol{A}, \boldsymbol{\Lambda}) \doteq -\frac{1}{4}\|\boldsymbol{A}\boldsymbol{Y}\|_4^4 + \langle \boldsymbol{\Lambda}, \boldsymbol{A}^*\boldsymbol{A} - \boldsymbol{I}\rangle. \tag{9.6.5}$$

This gives the necessary condition $\nabla_{\boldsymbol{A}}\mathcal{L}(\boldsymbol{A}, \boldsymbol{\Lambda}) = \boldsymbol{0}$:

$$-\nabla_{\boldsymbol{A}}\psi(\boldsymbol{A}) = (\boldsymbol{A}\boldsymbol{Y})^{\circ 3}\boldsymbol{Y}^* = \boldsymbol{A}\boldsymbol{S}, \tag{9.6.6}$$

for a symmetric matrix $\boldsymbol{S} = (\boldsymbol{\Lambda} + \boldsymbol{\Lambda}^*)$ (of Lagrange multipliers). Notice that if

[31] As we have seen the same scheme arises several times in the earlier sections of this Chapter, whenever a direction with negative curvature of the Hessian matrix is concerned.

[32] If the dictionary is not orthogonal, one can always convert the problem to an orthogonal one by certain normalization process, see [ZYL+20].

[33] One can show that when $n = 6$, there exist critical points whose Hessian has multiple zero eigenvalues.

$\boldsymbol{A}$ is an orthogonal matrix and $\boldsymbol{S}$ is symmetric, then the projection of $\boldsymbol{AS}$ onto the orthogonal group $\mathsf{O}(n)$ is

$$\mathcal{P}_{\mathsf{O}(n)}[\boldsymbol{AS}] = \boldsymbol{A}.$$

Then, by projecting both sides of (9.6.6) onto the orthogonal group $\mathsf{O}(n)$, the critical point, including the optimal solutions $\boldsymbol{A}_\star$, should satisfy the following "fixed point" equation:

$$\boldsymbol{A} = \mathcal{P}_{\mathsf{O}(n)}[(\boldsymbol{AY})^{\circ 3}\boldsymbol{Y}^*]. \tag{9.6.7}$$

So if we view

$$g(\cdot) \doteq \mathcal{P}_{\mathsf{O}(n)}[((\cdot)\boldsymbol{Y})^{\circ 3}\boldsymbol{Y}^*] : \mathsf{O}(n) \to \mathsf{O}(n)$$

as a map from $\mathsf{O}(n)$ to $\mathsf{O}(n)$ itself, one can show that this map is again a (locally) contraction map. This leads to the *matching, stretching, and projection* (MSP) algorithm [ZYL+20] for solving the dictionary learning problem:

$$\boldsymbol{A}_{k+1} = \mathcal{P}_{\mathsf{O}(n)}[(\boldsymbol{A}_k\boldsymbol{Y})^{\circ 3}\boldsymbol{Y}^*]. \tag{9.6.8}$$

Hence, the MSP can be viewed as a power iteration algorithm for solving the above fixed point problem.

The original Newton's method (9.1.13), introduced earlier in this Chapter, is precisely a "fixed-point" type algorithm for computing the roots of an equation. Only that it gives a contraction mapping locally around a critical point – see the proof of Proposition 9.2. The power iteration for computing eigenvectors or for dictionary learning is *unlike* any of the local (first-order or second-order) methods that we have introduced earlier in this chapter. It actually exploits *the global geometry* of the objective function over a nice manifold of the solution space: it has the ability to converge to the globally optimal solution from a random starting point, and enjoys much higher convergence rates. In fact, one can show that the MSP iteration for dictionary learning converges locally with *cubic rate* around the globally optimal solutions [ZYL+20], far more efficient than any first-order or second-order local methods introduced earlier.[34]

9.6.3 Optimization over Stiefel Manifolds

From the previous examples, we could try to generalize the method to a broader set of problems. Consider the problem of minimizing a *concave* function $f(\boldsymbol{X})$ over the so-called *Stiefel manifold*, for $m \leq n$:

$$\mathsf{V}_m(\mathbb{R}^n) \doteq \{\boldsymbol{X} \in \mathbb{R}^{n\times m} \mid \boldsymbol{X}^*\boldsymbol{X} = \boldsymbol{I}_{m\times m}\}. \tag{9.6.9}$$

Then for the program:

$$\min_{\boldsymbol{X}} f(\boldsymbol{X}) \quad \text{subject to} \quad \boldsymbol{X}^*\boldsymbol{X} = \boldsymbol{I}, \tag{9.6.10}$$

[34] However, the global convergence of MSP remains an open problem, despite compelling empirical evidences suggesting that is the case.

we consider the Lagrangian:

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{\Lambda}) \doteq f(\boldsymbol{X}) + \langle \boldsymbol{\Lambda}, \boldsymbol{X}^*\boldsymbol{X} - \boldsymbol{I} \rangle. \tag{9.6.11}$$

The necessary condition for optimality $\nabla_{\boldsymbol{X}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\Lambda}) = \boldsymbol{0}$ gives

$$-\nabla f(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{S}, \tag{9.6.12}$$

for a symmetric matrix $\boldsymbol{S} = (\boldsymbol{\Lambda} + \boldsymbol{\Lambda}^*)$. This gives

$$\nabla f(\boldsymbol{X})^* \nabla f(\boldsymbol{X}) = \boldsymbol{S}^*\boldsymbol{X}^*\boldsymbol{X}\boldsymbol{S} = \boldsymbol{S}^2. \tag{9.6.13}$$

We can solve for $\boldsymbol{S} = [\nabla f(\boldsymbol{X})^* \nabla f(\boldsymbol{X})]^{1/2}$. When $\boldsymbol{S}$ is invertible,[35] the necessary condition (9.6.12) for optimality becomes:

$$\boldsymbol{X} = -\nabla f(\boldsymbol{X})[\nabla f(\boldsymbol{X})^* \nabla f(\boldsymbol{X})]^{-1/2}. \tag{9.6.14}$$

For simplicity, we define:

$$g(\boldsymbol{X}) \doteq -\nabla f(\boldsymbol{X})[\nabla f(\boldsymbol{X})^* \nabla f(\boldsymbol{X})]^{-1/2} \tag{9.6.15}$$

as a mapping from $\mathsf{V}_m(\mathbb{R}^n)$ to itself:

$$g(\boldsymbol{X}) : \mathsf{V}_m(\mathbb{R}^n) \to \mathsf{V}_m(\mathbb{R}^n).$$

Hence the optimal solution $\boldsymbol{X}_\star$ can be viewed as the "fixed point" to the equation:

$$\boldsymbol{X} = g(\boldsymbol{X}).$$

To compute the fixed point, we can simply take the iteration:

$$\boldsymbol{X}_{k+1} = g(\boldsymbol{X}_k) = -\nabla f(\boldsymbol{X}_k)[\nabla f(\boldsymbol{X}_k)^* \nabla f(\boldsymbol{X}_k)]^{-1/2}. \tag{9.6.16}$$

It is easy to check that the iterations for the singular vector computation and the dictionary learning are precisely special cases to this iteration, with $m = 1$ and $m = n$, respectively.

The above descent scheme is also known as the *generalized power method* [JNRS10] (applied over Stiefel manifolds). With similar techniques as in the gradient descent case (in Section 9.1.1), one can show that when the objective function is concave, the iterative process converges to a first-order critical point at least in the rate $O(1/k)$ (see Exercise 9.8). However, as we see in the cases of singular vector and dictionary learning, when the function $f(\boldsymbol{X})$ has good properties, actual performance of the above scheme (9.6.16) can be far more efficient than the rate $O(1/k)$ for the worst case, especially when the associated function $g(\boldsymbol{X})$ is a (global or local) contraction mapping.

[35] which is normally the case, as we have seen in the cases with the singular vector and dictionary learning.

Fixed Point of a Contraction Mapping

Problem Class:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{M}, \text{ with } \mathcal{M} \text{ being a compact manifold.}$$

Critical points of f correspond to the fixed point of a (locally) contraction mapping $g(\cdot) : \mathcal{M} \to \mathcal{M}$:

$$g(\boldsymbol{x}) = \boldsymbol{x}.$$

Initialization: Set $\boldsymbol{x}_0 \in \mathbb{R}^n$ randomly (or locally near a critical point).

Iteration: For $k = 0, 1, 2, \ldots, K$,

$$\boldsymbol{x}_{k+1} = g(\boldsymbol{x}_k).$$

Convergence guarantee: $\boldsymbol{x}_k$ converges to a fixed point $\boldsymbol{x}_\star$ in at least geometrically fast (i.e., at least linear rate of convergence).

Figure 9.7 An overview of optimization via the fixed point of a contraction mapping.

9.6.4 Fixed Point of a Contraction Mapping

Notice that the power iteration algorithms for all three problems have one thing in common: they all rely on a (locally) contraction mapping from a compact manifold to itself. More generally speaking, let $\mathcal{M}$ be a compact smooth manifold with a distance metric $d(\cdot, \cdot)$.

Definition 9.16 (Contraction Mapping). *A map $g : \mathcal{M} \to \mathcal{M}$ is called a contraction mapping on $\mathcal{M}$ if there exists $\rho \in (0, 1)$ such that*

$$d(g(\boldsymbol{x}), g(\boldsymbol{y})) \leq \rho \cdot d(\boldsymbol{x}, \boldsymbol{y})$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{M}$.

The constant ρ can be viewed as the Lipschitz constant for g. For contraction mapping, we have the following well-known result:

Theorem 9.17 (Banach-Caccioppoli Fixed Point). *Let $(\mathcal{M}, d)$ be a complete metric space with a contraction mapping: $g : \mathcal{M} \to \mathcal{M}$. Then g has* a unique fixed point $\boldsymbol{x}_\star \in \mathcal{M}$*:*

$$g(\boldsymbol{x}_\star) = \boldsymbol{x}_\star.$$

In particular, as the previous examples have indicated, the unique fixed point $\boldsymbol{x}_\star$ can be found through the simple power iteration:

$$\boldsymbol{x}_{k+1} \leftarrow g(\boldsymbol{x}_k), \quad k = 0, 1, \ldots$$

and we have $\boldsymbol{x}_k \to \boldsymbol{x}_\star$ at least geometrically. Notice that, the fixed point scheme does not rely on local information such as gradient hence it can even escape

degenerate critical points. Empirically, we observe that the MSP algorithm works very well for the ℓ^4-based dictionary learning problem, even though the ℓ^4 norm has degenerate critical points on the orthogonal group $\mathsf{O}(n)$. In addition, the contraction factor ρ does not need to be a constant. If it scales with powers of the $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$, the contraction mapping enjoys higher than linear rates of convergence, as in the Newton or the MSP iteration. We summarize this in Figure 9.7 as a general algorithm of power iteration for solving the fixed point of a contraction mapping.

9.7 Notes

Modifications to Newton's Method.
Despite its simplicity and fast convergence, Newton's method (9.1.13) has several known problems. One problem is that for a nonconvex function the Hessian $\nabla^2 f(\boldsymbol{x})$ can sometimes be degenerate (hence not invertible). So the iteration (9.1.13) is not even defined. A popular fix to this problem is to regularize the Hessian with a unit matrix such that $\nabla^2 f(\boldsymbol{x}) + \lambda \boldsymbol{I} \succ \boldsymbol{0}$ is positive definite. The above Newton iteration is then modified to be:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}\right]^{-1} \nabla f(\boldsymbol{x}_k). \tag{9.7.1}$$

This is known as the popular *Levenberg-Marquardt regularization* [Lev44, Mar63, Mor78]. The above update rule can often be viewed as a mixture of gradient descent and Newton step with the parameter λ weighing between the two. Another, arguably more rigorous, justification of the above form of update is from the perspective of the trust region method [CGT00], which we will study in more details in Exercise 9.2. Due to its flexibility, the Levenberg-Marquardt method has been widely used in practice for solving nonconvex optimization problems, especially nonlinear least-squares type problems.

Complexity Bounds.
For functions with Lipschitz gradient and Hessian, [CDHS17] has derived the lower bound of first-order methods $O(\varepsilon_g^{-12/7})$, while it is believed that the best attainable upper bound is $O(\varepsilon_g^{-7/4})$. [AAZB+16, CDHS18] were among the first to make the attempt to develop algorithms that can achieve the optimal bound. Later the work [RW18, JNJ18] provided simplified approaches to achieve this bound by combining negative curvature and accelerated gradient descent. To a large extent, the methods presented in this chapter are inspired by these work.

Table 9.1 summarizes all the algorithms introduced in this chapter and their respective complexities in terms of associated oracles and convergence guarantees. These complexity bounds are for the worst case in the class of functions considered. If a particular function of interest has better structure or property (which is often the case in our settings), the complexity of even the vanilla gradient descent can be dramatically improved: For instance, if the function is locally

Method	Oracle	Stat. Point	Complexity
Vanilla gradient descent	first-order	first-order	$O(\varepsilon_g^{-2})$
Cubic Newton, Fig. 9.1	second-order	second-order	$O(\varepsilon_g^{-1.5})$
Gradient/negative curvature, Fig. 9.2	first-order	second-order	$O(\varepsilon_g^{-2})$
Negative curvature/Newton, Fig. 9.4	first-order	second-order	$O(\varepsilon_g^{-1.75})$
Hybrid noisy gradient, Fig. 9.5	first-order	second-order	$O(\varepsilon_g^{-2})$
Perturbed accelerated gradient, Fig. 9.6	first-order	second-order	$O(\varepsilon_g^{-1.75})$

Table 9.1 Oracles and complexities (up to log factors) of different optimization methods. "Stat. Point" stands for the type of stationary point $\boldsymbol{x}_\star$ to which the method guarantees to converge. All complexities are measured in terms of the number of oracles accessed before attaining a prescribed accuracy $\|\nabla f(\boldsymbol{x}_\star)\| \le \varepsilon_g$.

strongly convex around a minimizer, the local convergence rate becomes linear $O(\log \frac{1}{\varepsilon})$ (see Theorem D.4 of Appendix D).

Notice that these complexities are characterized for functions that have global Lipschitz gradient and Hessian. In practice, this may not be the case and we cannot easily decide on the step size without knowing the Lipschitz constants. So we generally may resort to a local line search scheme to determine the proper step size (see (D.1.2) of Appendix D) and then to establish corresponding convergence and complexity analysis.

Exploiting Geometric Structures.

In the last Section 9.6, we have shown a few important instances in which the optimization is over certain nonlinear manifold. The recent book [Bou20] gives a good introduction to optimization on general smooth manifolds. As we have seen in Chapter 7 and also will in the application chapters, optimization problems that arise in low-dimensional models often have nice global geometric structures such as certain symmetry over a homogeneous space. In such cases, we can develop extremely effective and scalable optimization algorithms, far beyond the local greedy gradient-based schemes. However, unlike the generic first-order or second-order methods summarized in Table 9.1, there is still a lack of systematic analysis of convergence and complexity for such geometric optimization problems. As we have alluded to before in Section 7.4 of Chapter 7, developing scalable algorithms for this class of problems and characterizing conditions under which there is guaranteed (global) convergence and complexity are certainly an important and pressing research topic for the future.

9.8 Exercises

9.1 (Examples for Newton's Method). *Apply Newton's method around the critical point, $x = 0$, of three functions: $f(x) = \frac{1}{2}x^2$, $f(x) = -\frac{1}{2}x^2$, and $f(x_1, x_2) = \frac{1}{2}(x_1^2 - x_2^2)$, and describe what Newton iteration does respectively in these three cases.*

9.2 (Trust Region Method). *In the Remark 9.3 about the trust region method, we need to find the minimizer to a constrained quadratic program of the form:*

$$\boldsymbol{w}_\star = \arg\min f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{w} \rangle + \frac{1}{2}\boldsymbol{w}^* \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{w} \;\; s.t. \;\; \|\boldsymbol{w}\|_2 \leq \delta_k. \quad (9.8.1)$$

To compute the optimal minimizer $\boldsymbol{w}_\star$, there are essentially three cases depending on the relationships between the gradient ∇f and the Hessian $\nabla^2 f$. Let λ_1 be the smallest eigenvalue of $\nabla^2 f$, and $\boldsymbol{e}_1$ be the associated eigenvector: $\nabla^2 f \boldsymbol{e}_1 = \lambda_1 \boldsymbol{e}_1$. If $\lambda_1 > 0$, the Hessian is positive definite. If $\lambda_1 < 0$, then $\boldsymbol{e}_1$ is the direction that the surface has the largest negative curvature. We denote the eigen-subspace associated with $\boldsymbol{e}_1$ as

$$\mathsf{S}_1 \doteq \{\alpha \boldsymbol{e}_1, \alpha \in \mathbb{R}\}.$$

- Case 1: *When $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite, i.e., $\lambda_1 > 0$, and*

$$\left\| \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f \right\|_2 < \delta,$$

  show that the optimal solution is given by $\boldsymbol{w}_\star = -\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$. Whenever this happens, the trust region Newton descent reduces to a regular Newton descent:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{w}_\star = \boldsymbol{x}_k - \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k). \quad (9.8.2)$$

  When the minimizer is not in the interior of the trust region, the problem becomes how to minimize a quadratic function over a sphere $\|\boldsymbol{w}\|_2 = 1$. The situation becomes a little more complicated as we see below.

- Case 2: *Show that if the gradient $\nabla f(\boldsymbol{x}_k)$ is not perpendicular to S_1, then the equation*

$$\left\| \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}\right]^{-1} \nabla f(\boldsymbol{x}_k) \right\|_2^2 = \delta^2 \quad (9.8.3)$$

  has a solution $\lambda_\star \geq 0$ in the range $(-\lambda_1, \infty)$ which gives the optimal minimizer $\boldsymbol{w}_\star = -\left[\nabla^2(\boldsymbol{x}_k) + \lambda_\star \boldsymbol{I}\right]^{-1} \nabla f(\boldsymbol{x}_k)$. In fact, here finding the optimal $\lambda_\star$ is itself a one-dimensional nonlinear optimization problem. One can use any optimization method (such as Newton's method) to solve it, and some specific options can be found in [CGT00]. Once the optimal minimizer $\boldsymbol{w}_\star$ is found,

the trust region Newton descent in this case is:[36]

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{w}_\star = \boldsymbol{x}_k - \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda_\star \boldsymbol{I}\right]^{-1} \nabla f(\boldsymbol{x}_k). \tag{9.8.4}$$

Further show that when the gradient ∇f *is perpendicular to* S_1*:* $\nabla f \perp \mathsf{S}_1$*, if the equation* $\left\| \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}\right]^{-1} \nabla f \right\|_2^2 = \delta^2$ *still has a solution* $\lambda_\star \geq 0$ *in the range* $(-\lambda_1, \infty)$*, then* $\boldsymbol{w}_\star = -\left[\nabla^2 f(\boldsymbol{x}_k) + \lambda_\star \boldsymbol{I}\right]^{-1} \nabla f$ *again is the desired minimizer.*

- Case 3: *The situation becomes a little trickier when* $\nabla f \perp \mathsf{S}_1$ *but the above equation does not have any solution. That is,* $\left\| \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}\right]^{-1} \nabla f \right\|_2^2 < \delta^2$ *for any* λ *that makes* $\nabla^2 f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}$ *positive definite. Show that this case only happens when* $\lambda_1 \leq 0$*. In this case, let* $\boldsymbol{w}_1$ *be the minimum norm solution to* $\left[\nabla^2 f(\boldsymbol{x}_k) - \lambda_1 \boldsymbol{I}\right] \boldsymbol{w} = -\nabla f$*, i.e.,* $\boldsymbol{w}_1 = -\left[\nabla^2 f(\boldsymbol{x}_k) - \lambda_1 \boldsymbol{I}\right]^\dagger \nabla f$*. Then show that the minimizer* $\boldsymbol{w}_\star$ *on the unit sphere is of the form:*

$$\boldsymbol{w}_\star = \boldsymbol{w}_1 + \beta \boldsymbol{e}_1 = -\left[\nabla^2 f(\boldsymbol{x}_k) - \lambda_1 \boldsymbol{I}\right]^\dagger \nabla f + \beta \boldsymbol{e}_1$$

with β *chosen such that* $\|\boldsymbol{w}_\star\|_2^2 = \delta^2$*. It is easy to see that so constructed* $\boldsymbol{w}_\star$ *satisfies the condition for minimizer:*

$$\left[\nabla^2 f(\boldsymbol{x}_k) - \lambda_1 \boldsymbol{I}\right] (\boldsymbol{w}_1 + \beta \boldsymbol{e}_1) = \left[\nabla^2 f(\boldsymbol{x}_k) - \lambda_1 \boldsymbol{I}\right] \boldsymbol{w}_1 = -\nabla f.$$

Geometrically, $\boldsymbol{w}_1$ *is the minimizer restricted to the subspace* $\mathsf{S}_1^\perp$*. If it is in the interior of the trust region, we then simply add a step along the direction of the largest negative curvature to reach the global minimizer* $\boldsymbol{w}_\star$ *on the boundary. The trust region Newton descent in this case becomes:*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{w}_\star = \boldsymbol{x}_k - \left[\nabla^2 f(\boldsymbol{x}_k) + \lambda_\star \boldsymbol{I}\right]^\dagger \nabla f(\boldsymbol{x}_k) + \beta \boldsymbol{e}_1. \tag{9.8.5}$$

9.3 (Cubic Regularized Newton's Method). *For the Cubic Newton's Method studied in Section 9.2,*

1 *Show that the cubic Newton step (9.2.5) reduces to solving a one-dimensional convex optimization problem, similar to the one that we have seen in the trust region method above.*
2 *Show that the optimal solution to the cubic Newton step satisfies the condition (9.2.11).*

9.4 (Power Iteration and Lanczos Method). *Implement in detail the power iteration and Lanczos method introduced in Section 9.3.2. Generate a real symmetric matrix* $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ *with* $\lambda_{\min}(\boldsymbol{A}) < 0$*, say for* $n = 1,000$*. Use the power iteration and the Lanczos to compute the eigenvector associated with the smallest (negative) eigenvalue. Plot the approximation error versus the number of iteration, and compare the two methods.*

[36] Notice that this update rule can be considered as a special case to the popular Levenberg-Marquardt regularization (9.4.2), with λ chosen to be a specific value, according to the second-order local geometry.

9.5 (Conjugate Gradient Method). *In this exercise, implement the conjugate gradient method mentioned in Section 9.4.3 to solve the linear equation* $\boldsymbol{A}\boldsymbol{s} = \boldsymbol{b}$, *say for a matrix* $\boldsymbol{A}$ *of size* $1,000 \times 1,000$. *Compare with efficiency with solving the equation by directly computing the inverse of* $\boldsymbol{A}$: $\boldsymbol{s} = \boldsymbol{A}^{-1}\boldsymbol{b}$.

9.6 (Laplace Method). *This exercise generalizes the basic ideas of the Laplace Method to general cases.*

1 *Prove the Lemma 9.12 for the case when the function* $f(x), x \in \mathbb{R}$ *has multiple global maximizers.*
2 *Prove the Theorem 9.13 for the case when the function* $f(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^n$ *has a unique global maximizer.*
3 *For the case when the function* $f(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^n$ *has a continuous family of global maximizers, the Gibbs distribution converges the density in (9.5.6).*

9.7 (Orthogonal Dictionary Learning). *Let us consider another formulation for finding an orthogonal dictionary* $\boldsymbol{A}_o$ *up to some signed permutations in one shot. We want to solve*

$$\min_{\boldsymbol{A},\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2 + \mu\|\boldsymbol{X}\|_1, \quad \textit{subject to} \quad \boldsymbol{A} \in \mathsf{O}(n),$$

where $\mathsf{O}(n) = \{\boldsymbol{Z} \in \mathbb{R}^{n\times n} \mid \boldsymbol{Z}^*\boldsymbol{Z} = \boldsymbol{I}\}$ *is the orthogonal group. Show that the problem can be reduced to*

$$\min_{\boldsymbol{A}} \operatorname{Huber}_\mu(\boldsymbol{A}^*\boldsymbol{Y}), \quad \textit{subject to} \quad \boldsymbol{A} \in \mathsf{O}(n). \tag{9.8.6}$$

where $\operatorname{Huber}_\mu(\cdot)$ *is the Huber loss, which is sum of scalar Huber function, introduced in (7.2.24), applies element-wise across all the matrix elements.*

9.8 (Generalized Power Iteration). *Show that the generalized power iteration (9.6.16) is equivalent to the following descent scheme:*

$$\boldsymbol{X}_{k+1} = \operatorname*{arg\,min}_{\boldsymbol{Y}\in\mathsf{V}_m(\mathbb{R}^n)} f(\boldsymbol{X}_k) + \langle \nabla f(\boldsymbol{X}_k), \boldsymbol{Y} - \boldsymbol{X}_k\rangle. \tag{9.8.7}$$

Use this fact to show that for a concave function, this descent scheme converges to a first-order critical point with a rate at least $O(1/k)$.

Part III

Applications to Real-World Problems

10 Magnetic Resonance Imaging

"If you want to find the secrets of the universe, think in terms of energy, frequency and vibration."

– Nikola Tesla

10.1 Introduction

Magnetic resonance imaging (MRI) is based on the science of nuclear magnetic resonance (NMR). Magnetic resonance states that certain atomic nuclei (such as protons in the water molecules) can absorb and emit radio frequency energy when placed in an external magnetic field. The emitted energy is proportional to important physical properties of a material such as proton density. Therefore in physics and chemistry, magnetic resonance is an important method for studying structures of chemical substances, and its discovery was awarded the Nobel Prize in 1952.

Later in 1970's, Paul Lauterbur and Peter Mansfield discovered that by introducing spatial gradients in the magnetic field, it is possible to create two-dimensional images of the structures, now known as magnetic resonance imaging (MRI). MRI soon proved to be extremely useful for medical diagnosis as it provides an accurate and non-invasive method for imaging internal organs of the human body. Unlike the X-rays or computed tomography (CT) scans, MRI does not exert ionizing radiation, hence is much less harmful. Today, MRI has become a routine medical examination in hospitals worldwide, especially for examining the brain and the spinal cord. For their contributions to MRI, Lauterbur and Mansfield were awarded a Nobel Prize in Physiology or Medicine in 2003.

Nevertheless, MRI machines can be rather expensive, and the acquisition process of MRI is considerably time-consuming as it needs to densely sample the magnetization responses with many different gradient fields. In order to lower the cost of MRI and improve patient comfort or safety,[1] in recent years, techniques from compressive sensing have proven to be extremely effective in improving the efficiency of MRI [LDSP08], which was briefly highlighted in Chapter 2 as one of the heralding successes.

[1] For young pediatric cancer patients, frequent exposures to strong magnetic fields for long periods of time can be unsafe and even fatal.

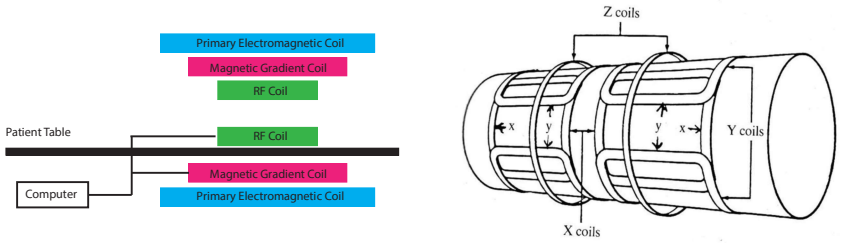


Figure 10.1 **Left:** Key components of a basic MRI machine. **Right:** The three-axis gradient coils.

In this chapter, we explain in more technical detail why MRI is particularly suitable for techniques of compressive sensing. First, a high-level review of the physics of MRI in Section 10.2 reveals that the MRI imaging process is amenable to compressive sampling as it naturally takes spatially encoded samples of the image in the frequency domain. Secondly, medical images of human organs are naturally structured and mostly piecewise-smooth. We can verify empirically that such images are highly compressible/sparse in a properly chosen transform domain, and introduce several effective sampling schemes in Section 10.3. Finally, we introduce in Section 10.4 some customized fast algorithms that can efficiently reconstruct the image from such compressive samples with high fidelity, despite imaging noise and other nuisance factors.

10.2 Formation of MR Images

In medical applications, MRI is based on measurements of a radio frequency (RF) signal, known as the *transverse magnetization*, generated by protons which exist in abundance as the hydrogen nuclei in the molecules of water and fat in human tissues. The signal measured is largely proportional to the density of protons at each spatial location, which indicates the presence or absence of such molecules. This information can then be used by physicians for diagnostic purposes. Here, we give a simplified mathematical model that captures the essence of this process. For a more detailed description of the physical process, one may refer, e.g., to [Wri97].

10.2.1 Basic Physics

It is known in quantum physics that each proton spins along an axis that creates an angular momentum. In the absence of any external magnetic field, the angular momenta of the protons are oriented randomly in their neutral state, hence collectively the protons (in the body tissue) do not produce any measurable magnetization. However, when a strong external magnetic field, denoted as $\boldsymbol{B}_0$,

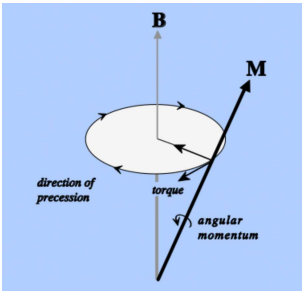


Figure 10.2 The direction of the magnetization $\boldsymbol{M}$ as a vector is precessing in a cone around $\boldsymbol{B}_0$, driven by the torque generated by the cross product $\boldsymbol{M} \times \boldsymbol{B}_0$, in a direction orthogonal to both $\boldsymbol{M}$ and $\boldsymbol{B}_0$. (Image from https://mri-q.com/bloch-equations.html and reprinted with permission from Allen D. Elster, MD.)

is applied to the tissue mass, it polarizes the protons and aligns their spins along the direction of $\boldsymbol{B}_0$ and produces a net magnetization, denoted as $\boldsymbol{M}$. $\boldsymbol{B}_0$ is also called the *primary magnetic field*, and its strength typically can range from 1.5 to 3 Tesla.[2] An MRI machine usually has three RF coils along the x, y, z axes respectively, as shown in Figure 10.1, and can produce a magnetic field in any direction by running electric currents through respective coils.

Following conventional notations in physics, we use $(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k})$ to denote the three unit vectors in the x, y, z axes of a (local) Cartesian frame. Without loss of generality, we may assume the direction of the external static magnetic field $\boldsymbol{B}_0$ aligns with the z-axis, that is, $\boldsymbol{B}_0 = B_0\boldsymbol{k}$. In general the magnetization $\boldsymbol{M}$ takes the form $\boldsymbol{M} = M_x\boldsymbol{i} + M_y\boldsymbol{j} + M_z\boldsymbol{k}$. If the external magnetic field is static, $\boldsymbol{M}$ will eventually reach an equilibrium magnetization of the form $M_0\boldsymbol{k}$.

Although protons can respond very quickly to the external magnetic field, the polarization itself does not yield any RF signal that can be measured by the machine. The key is that the magnetization $\boldsymbol{M}_{xy} = M_x\boldsymbol{i} + M_y\boldsymbol{j}$ in the *transverse plane* orthogonal to the primary axis undergoes very different dynamics and can be exploited for measurement. This transverse magnetization precesses about $\boldsymbol{B}_0$ according to the so called *Bloch equation*:

$$\frac{d\boldsymbol{M}_{xy}}{dt} = \gamma \boldsymbol{M}_{xy} \times \boldsymbol{B}_0, \tag{10.2.1}$$

where γ is a physical constant. Figure 10.2 visualizes the precessing of $\boldsymbol{M}_{xy}$ around $\boldsymbol{B}_0$. From this equation, we see that the precession frequency is $\omega_0 = \gamma B_0$, known as the *Larmor frequency*. This rotating magnetic moment radiates an electromagnetic signal which is picked up by the MRI machine.

So in order to produce a precessing magnetization orthogonal to $\boldsymbol{B}_0$, in the

[2] In comparison, the magnitude of the Earth's natural magnetic field only ranges from 25 to 65 micro Tesla.

second step of MRI imaging, one applies a second time-varying magnetic field $\boldsymbol{B}_1$ in the xy plane transverse to $\boldsymbol{B}_0 = B_0\boldsymbol{k}$. Typically $\boldsymbol{B}_1$ is chosen to be $\boldsymbol{B}_1 = \cos(\omega_0 t)\boldsymbol{i} + \sin(\omega_0 t)\boldsymbol{j}$ which rotates around $\boldsymbol{B}_0$ at the radian frequency ω_0. This magnetic field excites the protons to a higher energy state.

The excitation stops after a short period, and the protons gradually fall back to its equilibrium state. This relaxation process lasts from milliseconds to several seconds. As the transverse magnetization $\boldsymbol{M}_{xy}$ precesses, it induces an electromagnetic force in the RF coils. The magnitude, phase, and relaxation time of the signal represent different properties of the matter that can be recorded in different types of MR images.

10.2.2 Selective Excitation and Spatial Encoding

One question remains with regard to the imaging process, namely, how does the MRI machine isolate and measure the RF signal from different parts of the body, because if the body is affected by a single static magnetic field, then all the protons will be aligned homogeneously. Several clever (Nobel Prize worthy clever) techniques are required to address this issue, including the so-called *selective excitation* and *spatial encoding*. The goal is to be able to sample and measure magnetization signals from any spatial location (x, y, z) (up to certain resolution).

As the transverse magnetization $\boldsymbol{M}_{xy}$ is of interest, we may choose to excite the magnetic field in a thin slice along the z-axis, say around z_0. That is, we are interested in the plane (x, y, z_0). The selective excitation can be achieved by first making the Larmor frequency varying linearly in the z-direction with the magnetic field

$$\boldsymbol{B}_0(z) = \big(B_0 + G_z(z - z_0)\big)\boldsymbol{k}.$$

We then apply an additional RF excitation pulse with energy over a limited range of frequency bandwidth Ω corresponding to the Larmor frequency $\omega_0 = \gamma B_0$ at the slice z_0. Typically, we could choose the pulse to be

$$\boldsymbol{B}_1(t) = \operatorname{sinc}(\Omega t)\big(\cos(\omega_0 t)\boldsymbol{i} + \sin(\omega_0 t)\boldsymbol{j}\big),$$

which has a rectangular (energy) distribution around the frequency ω_0. As result, only protons around the slice (x, y, z_0) may resonate with the excitation and reach a high magnetization level, say $\boldsymbol{M}_{xy}(x, y) \doteq \boldsymbol{M}_{xy}(x, y, z_0)$. This is why the whole process is called *magnetic resonance imaging*.

The remaining question is how to image magnetization of different spatial locations inside the plane (x, y, z_0). As we see from the above, spatially selectivity (along the z-axis) can be achieved through introducing a spatially varying excitation with a gradient (G_z) in the z direction. Hence, we could generalize this idea by introducing an additional magnetic field $\boldsymbol{B}$ with a gradient G_x and G_y in the x and y directions, respectively. Moreover, we could vary the gradients as

functions of time t:

$$\boldsymbol{B} = \big(B_0 + G_x(t)x + G_y(t)y\big)\boldsymbol{k}.$$

After the selective excitation, the magnetic field in the transverse plane (x, y, z_0) is $\boldsymbol{M}_{xy}(x, y)$. Once the slice is subject to the above magnetic field, $\boldsymbol{M}_{xy}$ precesses according to the Bloch equation and we can measure the electromagnetic signal generated by it. Assume that the magnitude $|\boldsymbol{M}_{xy}|$ remains relatively constant during the acquisition period, then from the Bloch equation, we have:

$$M_{xy}(x, y, t) = |\boldsymbol{M}_{xy}(x, y)|e^{-\mathrm{i}\omega_0 t}e^{-\mathrm{i}\gamma\int_0^t (G_x(\tau)x + G_y(\tau)y)d\tau},$$

where $\mathrm{i} = \sqrt{-1}$ is the imaginary unit. From this equation, we can ascertain the true reason for introducing a gradient magnetic field: it allows us to manipulate the phase of the transverse magnetic field $\boldsymbol{M}_{xy}$ so as to encode the spatial information about $\boldsymbol{M}_{xy}$ that we needed in the first place.

To see this, note that the actual signal we measure is the collective effect of all $\boldsymbol{M}_{xy}$ in the xy-plane. To simplify the notation, let us define

$$k_x(t) \doteq \gamma\int_0^t G_x(\tau)d\tau, \quad k_y(t) \doteq \gamma\int_0^t G_y(\tau)d\tau.$$

In the MRI literature, the so-defined quantities (k_x, k_y) index a two-dimensional space called *k-space*. We then have the measured signal, say $s(t)$, as

$$s(t) = \exp^{-\mathrm{i}\omega_0 t}\int_x\int_y |\boldsymbol{M}_{xy}(x, y)|e^{-\mathrm{i}(k_x(t)x + k_y(t)y)}dxdy.$$

Notice that this measured signal $s(t)$, once with the $e^{-\mathrm{i}\omega_0 t}$ component demodulated, is essentially a *2D spatial Fourier transform* of $|\boldsymbol{M}_{xy}(x, y)|$ at the spatial frequency $(k_x(t), k_y(t))$:

$$S(k_x, k_y) = \int_x\int_y |\boldsymbol{M}_{xy}(x, y)|e^{-\mathrm{i}(k_x x + k_y y)}dxdy. \tag{10.2.2}$$

In the MRI literature, this technique is called *spatial frequency encoding*. So, in principle, once we have collected measurements of S at sufficiently many spatial frequencies (k_x, k_y)'s, we could recover $|\boldsymbol{M}_{xy}(x, y)|$ simply from its *inverse Fourier transform*:

$$|\boldsymbol{M}_{xy}(x, y)| \propto \int_{k_x}\int_{k_y} S(k_x, k_y)e^{\mathrm{i}(k_x x + k_y y)}dk_x dk_y, \tag{10.2.3}$$

which can be visualized as a 2D image, say $I(x, y)$, on the xy-plane (at z_0).

10.2.3 Sampling and Reconstruction

We have described above in a nutshell the physical and mathematical models of MR imaging. In short, we see that the value of the measured signal S at any given time t is essentially the 2D Fourier transform of the image of interest $I(x, y)$ (or $|\boldsymbol{M}_{xy}(x, y)|$) at a particular spatial frequency (k_x, k_y). For any given gradient

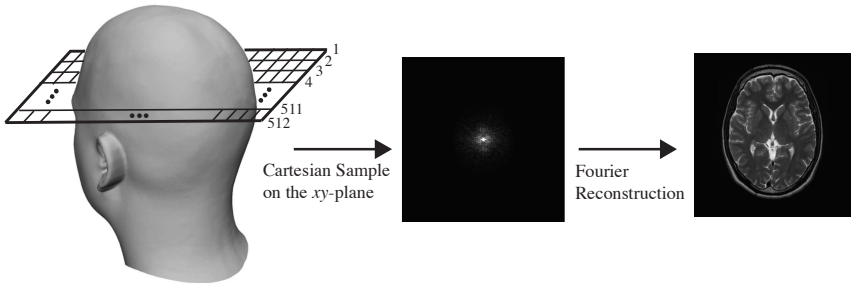


Figure 10.3 A Cartesian sample of the human brain (left) and its reconstructed MRI image (right). The sampling resolution in this example is $m = 512 \times 512$.

field generated by $(G_x(t), G_y(t))$, if we measure the signal S at a sequence of time $\{t_1, t_2, \ldots\}$, we obtain the samples of the Fourier transform of $I(x, y)$ at different frequencies $\{(k_x(t_1), k_y(t_1)), (k_x(t_2), k_y(t_2)), \ldots\}$ in the transform domain (the k-space).

In practice, we are interested in recovering the image up to certain spatial resolution. That is, instead of a function on a continuous domain (the entire xy-plane), we consider the image $I(x, y)$ is a function on a finite Cartesian grid (say of size $N \times N$). We denote the coordinates of the pixels as a vector $\boldsymbol{v} = (x, y)$. In this case, the measurements can be viewed as discrete Fourier transform of the image, which lie on a Cartesian grid (of size $N \times N$) in the k-space. We denote the coordinates of the frequencies as a vector $\boldsymbol{u} = (k_x, k_y)$. We collect all measurements as a vector $\boldsymbol{y} \in \mathbb{R}^m$ with $m = N^2$. That is, each entry of $\boldsymbol{y}$ is of the form:

$$y_i = \sum_{\boldsymbol{v}} I(\boldsymbol{v}) e^{-\mathrm{i}\boldsymbol{u}_i^* \boldsymbol{v}} \, \Delta \boldsymbol{v}, \quad i = 1, \ldots, m$$

where the sum is over the grid and $\Delta \boldsymbol{v}$ is the grid step size. If we also view the image $I(\boldsymbol{v})$ as a vector of dimension m, then we have:

$$\boldsymbol{y}(\boldsymbol{u}) = \mathcal{F}[I(\boldsymbol{v})](\boldsymbol{u}), \tag{10.2.4}$$

where $\mathcal{F}$ is an $m \times m$ matrix representing the discrete (2D) Fourier transform. As the matrix $\mathcal{F}$ is invertible, the MR image I can be simply recovered from such Cartesian samples as:

$$I(\boldsymbol{v}) = \mathcal{F}^{-1}[\boldsymbol{y}(\boldsymbol{u})](\boldsymbol{v}). \tag{10.2.5}$$

Figure 10.3 shows an example of recovered MR image from such Cartesian sampling scheme.

At first sight, sampling the entire Cartesian grid in the transform domain seems natural, and the reconstruction via inverse Fourier transform is straightforward. However, for practical images, it is too redundant to get all the $m = N^2$ samples

as illustrated in the example in Figure 10.3. Conventional signal processing techniques have been applied to reduce the number of samples. For instance, if the image largely consists of low-frequency components and has a cutoff bandwidth $f_{\max}$, then we only need to sample the transform domain on a sub grid according to the Nyquist rate

$$f_{\text{Nyquist}} \geq 2f_{\max}.$$

Nevertheless, the number of samples required by the Nyquist rate is still very large,[3] which makes the conventional MRI imaging process very time-consuming. For the rest of this chapter, we will see that by harnessing additional structures (e.g., sparsity and smoothness) of the MR image, one can significantly reduce the number of samples needed. We will first discuss the sparsity of MR images, and then introduce a few effective compressive sampling schemes. Finally, we will discuss numerical methods for reconstructing MR images from small sets of samples, since, in this under-sampled regime we can no longer simply rely on the inverse Fourier transform.

10.3 Sparsity and Compressive Sampling of MR Images

10.3.1 Sparsity of MR Images

In order to improve the sampling efficiency of MR images, we need to leverage additional structure of the target image I. We know from Chapters 2-3 that *sparsity* is a very powerful structural assumption, which, when present can substantially reduce the number of measurements that are required to reconstruct a signal of interest. However, MR images are not sparse – most of the pixels are nonzero! On the other hand, MR images *are* structured: they can be approximated as piecewise smooth functions with relatively few sharp edges. We will see that this type of structure actually leads to a form of sparsity, in an appropriately chosen transform domain.

From signal processing and harmonic analysis, we know that piecewise smooth functions are compressible (nearly sparse) when represented in terms of appropriate basis functions – say, wavelets. There is a deep theory associated with wavelets and related 2D signal representations. Here, we only sketch these constructions at a loose, operational level.

A wavelet transform $\mathbf{\Phi}$ maps an $N \times N$ image I to a collection of N^2 coefficients $\boldsymbol{x} = \mathbf{\Phi}[I]$. The inverse transform $\mathbf{\Psi} = \mathbf{\Phi}^{-1}$ maps the coefficients $\boldsymbol{x}$ to an image $I = \mathbf{\Psi}[\boldsymbol{x}]$. The inverse mapping can be interpreted as expressing the image as a superposition of basis function $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{N^2}$:

$$I = \mathbf{\Psi}[\boldsymbol{x}] = \sum_{i=1}^{N^2} \boldsymbol{\psi}_i x_i. \tag{10.3.1}$$

[3] For images with sharp edges and contours, its cutoff bandwidth may be very high.

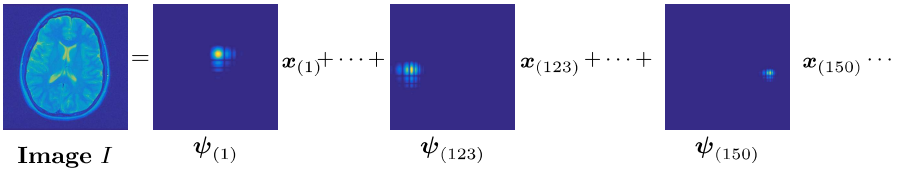


Figure 10.4 Wavelet Representation of an Image. The image I is expressed as a superposition of basis functions ψ_i, with coefficients $\boldsymbol{x}_{(i)}$. In this figure, we order the coefficients by magnitude, in descending order: $\boldsymbol{x}_{(1)}$ is the largest magnitude coefficient, and $\boldsymbol{\psi}_{(1)}$ its corresponding basis function, $\boldsymbol{x}_{(2)}$ the second largest, and so on. The largest coefficients capture low-frequency structure, as well as high-frequency structure around edges. Notice, e.g., that $\boldsymbol{\psi}_{(123)}$ and $\boldsymbol{\psi}_{(155)}$ are located near sharp edges at the left and right side of the brain.

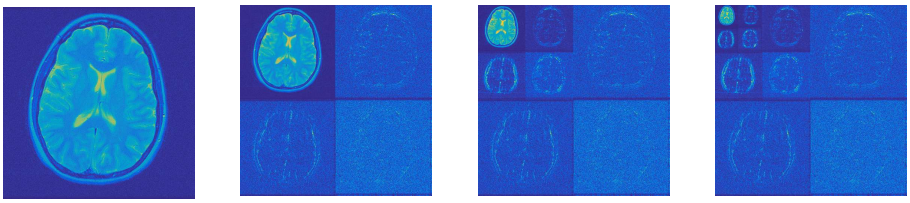


Figure 10.5 Wavelet Coefficients of an Image. From left to right, an original image, and the coefficients of one level, two level and three level wavelet decompositions using the Daubechies `db4` wavelet. The level one coefficients are organized as LL (upper left), LH (upper right), HL (lower left) and HH (lower right). The detail coefficients (high frequency) are concentrated near sharp edges.

Figure 10.4 visualizes several of the basis functions associated with a particular two-dimensional wavelet transform.[4]

The coefficients x_i have a very nice interpretation. To transform the image I, we split the image into four bands, which capture vertical and horizontal frequency content at different spatial locations in the image. The low frequency band, typically labeled LL, contains low frequency in both directions, while the high frequency band HH contains high frequency content in both directions. Two other bands HL and LH contain high frequency content in one direction and low-frequency content in the other direction. Figure 10.5 illustrates this operation. Notice that most of the significant entries occur in the LL band. By repeating this operation to the LL band, we obtain a two-level transform which captures localized frequency content at multiple scales in the image. We can continue in

[4] There are a variety of wavelets, leading to a variety of different transforms. In the experiments in this chapter, we adopt the Daubechies `db4` wavelet. Other choices of separable wavelets lead to different transforms, but their behavior is qualitatively similar.

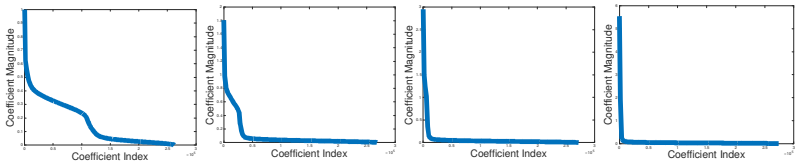


Figure 10.6 Decay of the Wavelet Coefficients. Left: the magnitudes of the image pixel values, plotted in descending order. **Right three:** the magnitudes of the wavelet coefficients $\boldsymbol{x}$ in descending order, for one, two and three level wavelet transforms. The wavelet coefficients decay much more rapidly than the original pixel values.

this manner. Figure 10.5 illustrates the three level to five level transforms of this image.

MR images tend to be piecewise smooth, with only a few sharp edges. The HL, LH and HH coefficients concentrate around edges, and so they tend to be quite sparse. Indeed, classical results in harmonic analysis can be paraphrased as arguing that the one dimensional version of this representation is nearly optimal for representing one-dimensional functions which are piecewise smooth with only a few discontinuities.[5] Figure 10.6 plots the sorted magnitudes of the coefficients $\boldsymbol{x}$, for each level l, from $l = 0$ (the original image) to $l = 3$. Notice that as we increase the number of levels in the transform, the coefficients become increasingly compressible.

Because the wavelet coefficients are nearly sparse, we can accurately approximate the input image using just a few wavelet coefficients. Let $\mathsf{J} = \{i_1, \ldots, i_k\}$ denote the indices of the k largest coefficients x_i (across all scales) in absolute value. We can form the *best k-term approximation*

$$\hat{I} = \sum_{i \in \mathsf{J}} \boldsymbol{\psi}_i x_i, \tag{10.3.2}$$

by retaining only these largest coefficients. Figure 10.7 visualizes approximations with the best 1%, 4% and 7% of the coefficients, respectively. It also visualizes the approximation error $|I - \hat{I}|$. Notice that the approximation errors are almost entirely populated with noise. For comparison, Figure 10.7 (bottom) shows approximations using the best 1%, 4% and 7% of the original image pixels. The wavelet approximations are dramatically more accurate than pixel approximations.

Of course, there is no reason to believe that wavelet sparsity captures *all* of the structure in an MR image. Other structural assumptions may lead to sparser representations, which can be leveraged to sample even more efficiently.

[5] For piecewise smooth functions on a two-dimensional domain, the situation is more complicated, and there is a large literature of image representations.

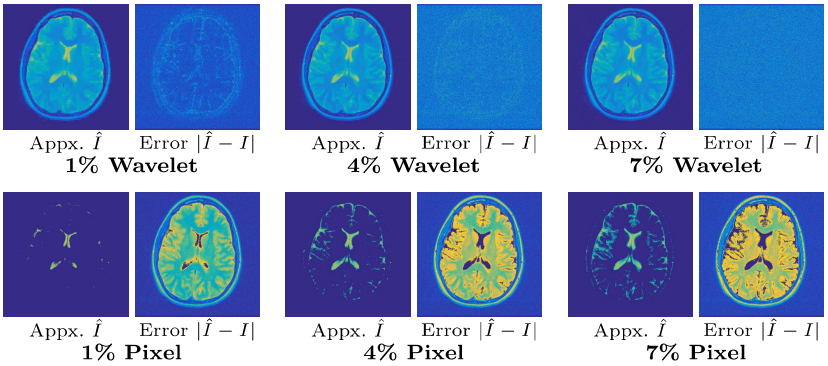


Figure 10.7 Wavelet Reconstructions. Top: Approximations of the brain image using the most significant wavelet coefficients. We plot reconstructions $\hat{I}$ using the largest 1%, 4% and 7% of the wavelet coefficients, as well as the approximation error $|\hat{I} - I|$. Retaining roughly 7% of the wavelet coefficients captures most of the important structure in the image; what remains is mostly noise. **Bottom:** For comparison purposes, we plot reconstructions and errors using the 1%, 4% and 7% largest image pixels. These approximations are very inaccurate: the image is nearly sparse in the wavelet domain, but not in the original pixel domain.

The literature is rich with alternatives, including representations that capture oriented edges, nonlocal representations that capture repeated structure, and learned representations that adapt to the specific classes of images. We will return to this point in Section 10.4, where we sketch one means of further reducing the sampling burden for MRI, by leveraging an additional form of sparsity. For now, we turn to the question of how we can use the knowledge that the wavelet coefficients are sparse to sample more efficiently.

10.3.2 Compressive Sampling of MR Images

Although a wavelet transform is able to sparsify the MR image I, notice that we cannot have access to the wavelet coefficients $\boldsymbol{x}$ unless we have acquired the entire image I (and then apply the transform $\boldsymbol{\Phi}$). Hence in conventional image processing, wavelet transforms have mostly been used in the post-processing of an image after it has been acquired, such as for compression. Now the question is how can we exploit the fact the MR image is sufficiently sparse in certain (wavelet) domains so that we can significantly reduce the number of measurements sampled in the acquisition time and still recover the image with good quality?

First we notice that the relationship between the measurements (Fourier coefficients) $\boldsymbol{y} \in \mathbb{C}^{N^2}$ and the (sparse) wavelet coefficients $\boldsymbol{x} \in \mathbb{R}^{N^2}$ is given by:

$$\boldsymbol{y} = \mathcal{F}[\boldsymbol{\Psi}\boldsymbol{x}].$$

From the physical model we have described above, we can directly measure any subset of the Fourier coefficients $\boldsymbol{y}$ or any linear superpositions of them. For convenience we denote the image I as a vector $\boldsymbol{z} \doteq I \in \mathbb{R}^{N^2}$:

$$\boldsymbol{z} = \boldsymbol{\Psi}\boldsymbol{x}.$$

Suppose, instead of taking all the N^2 Fourier coefficients, we measure only $m \ll N^2$ samples of (linear superpositions) of the Fourier coefficients. Then the transform from $\boldsymbol{z}$ to the m partial measurements $\boldsymbol{y}$ can be represented as an $m \times N^2$ matrix, denoted as $\mathcal{F}_U \in \mathbb{C}^{m \times N^2}$. Hence we have:

$$\boldsymbol{y} = \mathcal{F}_U[\boldsymbol{\Psi}\boldsymbol{x}] \doteq \boldsymbol{A}\boldsymbol{x}, \tag{10.3.3}$$

where we denote $\boldsymbol{A} \doteq \mathcal{F}_U\boldsymbol{\Psi} \in \mathbb{C}^{m \times N^2}$.

As we have learned from early chapters of the book, if the overall sampling matrix $\boldsymbol{A}$ is sufficiently *incoherent*, then we in principle can correctly recover all the sparse (wavelet) coefficients $\boldsymbol{x}$ from significantly fewer m samples. To ensure the matrix $\boldsymbol{A}$ is incoherent, we know from Chapter 3 (Section 3.4.3) that randomly chosen partial submatrix of the Fourier (or wavelet) transform $\mathcal{F}$ is incoherent. Hence, a conceptually simple compressive sampling scheme is to take some random measurements of the Fourier coefficients $\boldsymbol{y}$.

However, as we can notice in Figure 10.3, the most significant nonzero Fourier coefficients of a typical MR image are mainly in the low frequency region, and the coefficients in the high frequency region are already quite sparse and small. Hence a uniformly random sampling of the Fourier domain is not necessarily the most efficient. A more suitable sampling scheme for so-distributed coefficients is the *variable density random sampling*. It is designed specifically for 2D image objects where most of their energy is concentrated close the origin of the frequency domain. More specifically, although the locations of the samples are still randomly selected, it progressively gives higher chances for samples in lower frequencies to be selected than in the higher frequencies. Figure 10.8 shows one example of the variable density random sample pattern.

In practice, however, from the physical model of MRI that we have described above, we know that the MRI machine cannot take measurements at totally random locations from time to time. Instead, it produces a sequence of samples of the Fourier coefficients along a continuous trajectory $(k_x(t), k_y(t))$ in the k-space. Hence, the main challenge of compressive MRI is to design both practical and efficient sampling schemes in the Fourier domain for real MR images that are subject to the constraints of the physical process. To this end, some popular subsampling patterns have been proven (empirically) effective for MRI. Examples include a *radial* sampling pattern and a *spiral* pattern as shown in Figure 10.9. Clearly, both patterns are designed to have more coefficients densely sampled close to the origin and sparser coefficients far away from the origin.

To see the effectiveness of different sampling patterns, in Figure 10.10, we plot

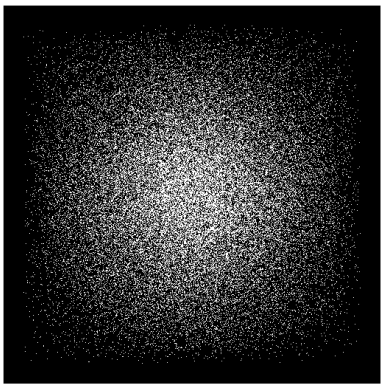


Figure 10.8 A variable density random sampling pattern in the Fourier domain.

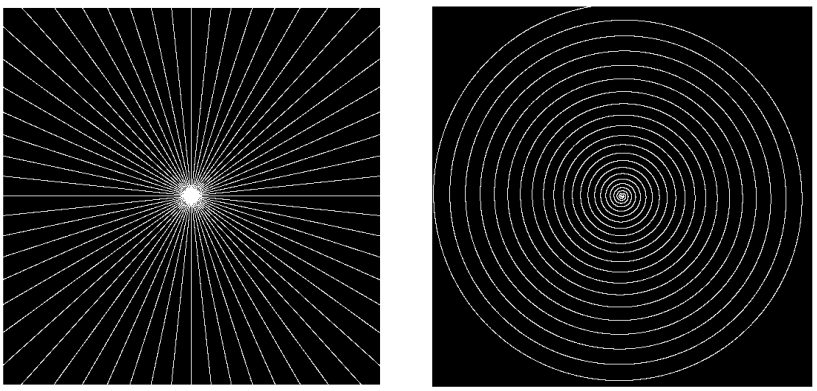


Figure 10.9 Examples of a radial sampling pattern and a spiral pattern.

the PSNRs of the reconstructed images against different sample percentages.[6] To establish a baseline, we first calculate the PSNR values when the most significant nonzero wavelet coefficients in $\boldsymbol{x}$ are given. The results are shown in the red curve. It clearly outperforms the other subsampling methods that do not have the knowledge of the ground truth sparse signal $\boldsymbol{x}$. Furthermore, compared to the deterministic radial and spiral patterns, the variable density random sampling initially achieves the worst reconstruction quality when the sampling percentage is low, namely, less than 20%. Then its performance increases significantly when the sampling percentage becomes higher, gradually surpassing the performance

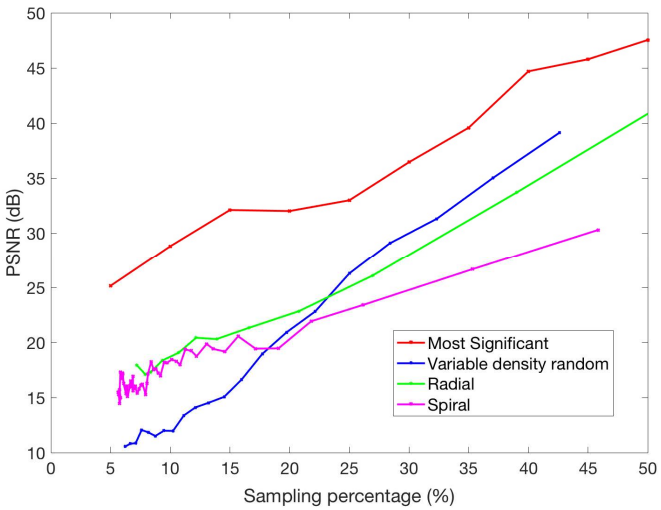


Figure 10.10 The reconstruction quality of the brain image using different subsampling patterns.

of the other subsampling patterns. The reader may refer to [LDSP08] for more discussions about compressive sampling of MRI.

10.4 Algorithms for MR Image Recovery

In this section, we discuss algorithms for reconstructing MR images from the sampled measurements $\boldsymbol{y}$. This is conceptually straightforward: many of the methods described in Chapter 8 can be applied to reconstruct the sparse coefficients $\boldsymbol{x}$ from the measurements $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. However, there are several practical considerations that demand additional attention. First, MR measurements are subject to various nonidealities, including noise. Second, because it is so important to make the sampling scheme as efficient as possible, it is often helpful to leverage other structural information about the target image, beyond sparsity of its wavelet coefficients $\boldsymbol{x}$.

Measurement Noise.
In practice, the measured MR image I can be degraded by thermal noise, so that the measurements $\boldsymbol{y}$ are

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{n}, \tag{10.4.1}$$

where $\boldsymbol{n}$ is a noise term with bounded norm $\|\boldsymbol{n}\|_2 < \varepsilon$ or assumed to be Gaussian for simplicity. One can accurately estimate the sparse coefficients $\boldsymbol{x}$ by looking

[6] We will describe details of the reconstruction algorithm in the next section.

for the minimum ℓ^1 norm coefficients that agree with the observations $\boldsymbol{y}$ up to the noise level (see Chapter 3, Section 3.5):

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \varepsilon, \tag{10.4.2}$$

Once $\boldsymbol{x}$ is recovered from solving this program, we can recover the image as $\hat{\boldsymbol{z}} = \boldsymbol{\Psi}[\hat{\boldsymbol{x}}]$.[7]

Gradient Sparsity.
The wavelet representation developed above is well-suited for representing piecewise smooth functions with smooth discontinuities. As we can see from the brain image, MR images may exhibit stronger properties than just piecewise smoothness: an image may be approximated as piecewise *constant* [MYZC08]. This means that the image value is constant away from a few sharp edges. The gradient of such an image is nonzero only at the edges, and hence is sparse.

To be more precise, let ∇_1 and ∇_2 represent finite-difference (differentiation) operators on the first (x) and second (y) coordinates of the image I, respectively. We use $(\nabla \boldsymbol{z})_i = ((\nabla_1 \boldsymbol{z})_i, (\nabla_2 \boldsymbol{z})_i) \in \mathbb{R}^2$ to denote the gradient vector at a pixel i. The ℓ^2 norm $\|(\nabla \boldsymbol{z})_i\|_2 = \left((\nabla_1 \boldsymbol{z})_i^2 + (\nabla_2 \boldsymbol{z})_i^2\right)^{1/2}$ measures the length of this vector.

A piecewise constant image has relatively few pixels at which the gradient is nonzero:

$$\sum_i \mathbb{1}_{\|(\nabla \boldsymbol{z})_i\|_2 \neq 0} \tag{10.4.3}$$

is small. This can be interpreted as a group sparsity assumption on the gradient vector field – see Chapter 6.

The number of points of nonzero gradient, (10.4.3), is conceptually simple, but is not well-suited to efficient computation. Following the intuition for group sparsity in Chapter 6, we can define a convex relaxation of this function, known as the *total variation* of the image $\boldsymbol{z}$:

$$\|\boldsymbol{z}\|_{\mathrm{TV}} \doteq \sum_i \|(\nabla \boldsymbol{z})_i\|_2 . \tag{10.4.4}$$

This is a convex function of $\boldsymbol{z}$.[8]

Using this convex function, we can verify experimentally that MR images are well-approximated as gradient-sparse. To do this, we take an image $\boldsymbol{z}$, and compute approximations to it using the proximal operator[9] for the total variation:

$$\hat{\boldsymbol{z}}_\lambda = \mathrm{prox}_{\lambda \mathrm{TV}}(\boldsymbol{z}) \doteq \arg\min_{\boldsymbol{x}} \lambda \|\boldsymbol{x}\|_{\mathrm{TV}} + \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2 . \tag{10.4.5}$$

For each $\lambda \geq 0$, we have an approximation $\hat{\boldsymbol{z}}_\lambda$, whose gradient is sparse. The

[7] Here by abuse of notation, we here use $\hat{\boldsymbol{z}}$ to denote both the 2D MR image I and its vectorized version as a vector in $\mathbb{R}^{N^2}$, as its meaning is clear from the context.
[8] Strictly speaking, it is not a norm, because it is not positive definite.
[9] For more details on proximal operators, see Sections 1-3 of Chapter 8.

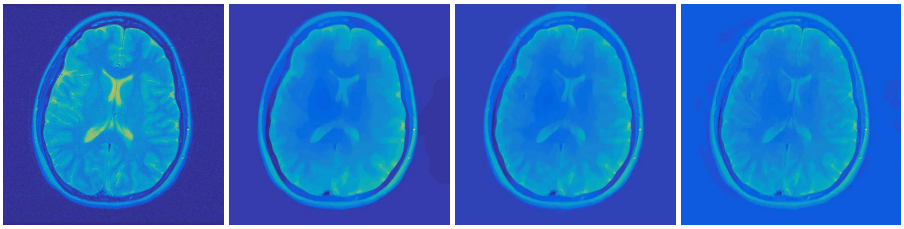


Figure 10.11 Gradient-sparse Approximations. Left: target MR image $\boldsymbol{z}$. **Right three:** gradient-sparse approximations computed using the proximal operator for the total variation, $\text{prox}_{\lambda \text{TV}}(\boldsymbol{z})$ for $\lambda = 0.6, 0.4, 0.2$, respectively. For each approximation, we also display the fraction of pixels at which the gradient is nonzero.

parameter λ trades off between gradient sparsity of $\hat{\boldsymbol{z}}$ and fidelity to the original image $\boldsymbol{z}$. Here, the proximal operator $\text{prox}_{\lambda \text{TV}}(\cdot)$ can be computed using the alternating directions method of multipliers (ADMM); we will describe this in more generality below.

Figure 10.11 shows approximations $\hat{\boldsymbol{z}}_\lambda$ to $\boldsymbol{z}$, with $\lambda = 0.6, 0.4, 0.2, 0.1$. From the figure, we see that the image admits visually plausible approximations with roughly 10% of the nonzero gradient vectors.

Combining Gradient Sparsity and Wavelet Sparsity.
To encourage the recovered image to have sparse gradients, we can incorporate the total variation into the above stable sparse recovery program (10.4.2) as an additional regularization term. Notice that $\boldsymbol{x} = \boldsymbol{\Phi z}$ and $\boldsymbol{Ax} = \mathcal{F}_U[\boldsymbol{z}]$. So the resulting program becomes:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \alpha \|\boldsymbol{\Phi z}\|_1 + \beta \|\boldsymbol{z}\|_{\text{TV}} \quad \text{subject to} \quad \|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2 < \varepsilon, \tag{10.4.6}$$

where $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$ are two positive weight parameters. The use of total variation and ℓ^1 norm together for MRI recovery was originally introduced by the work of [LDP07] and [MYZC08].

We can rewrite the above program in an unconstrained form as

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \alpha \|\boldsymbol{\Phi z}\|_1 + \beta \sum_i \|(\nabla \boldsymbol{z})_i\|_2 + \frac{1}{2} \|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2. \tag{10.4.7}$$

Since every term is a convex function of $\boldsymbol{z}$, the overall objective function is also convex. Such a program can be efficiently solved by the so-called fixed-point iteration (see [MYZC08] for more details).

Optimization Algorithm.
We introduce here a simpler (and arguably faster) algorithm by exploiting the special structure of the program. We observe that the main challenge in solving

the above program seems to be that the objective function contains two separate terms, one minimizing the ℓ^1-norm of $\boldsymbol{\Phi z}$ and the other minimizing the sum of ℓ^2 norms of the gradient of $\boldsymbol{z}$. If we optimize one of the two terms $\|\boldsymbol{\Phi z}\|_1$ and $\|\boldsymbol{z}\|_{\mathrm{TV}}$ while treating the other constant, each of the two sub-problems will be a relatively easy optimization problem. Hence one may utilize the alternating direction minimization method (ADMM) introduced in Chapter 8 Section 8.5 to solve this program. This was first suggested by the work of [YZY10]. We here give a brief description of the algorithm.

The first two terms of (10.4.7) both depend on $\boldsymbol{z}$. So to utilize ADMM, we need to separate the variables first. To this end, we introduce some auxiliary variables: $\boldsymbol{x} \doteq \boldsymbol{\Phi z} \in \mathbb{R}^{N^2}$ for the (sparse) wavelet coefficients and $\boldsymbol{v}_i \doteq (\nabla \boldsymbol{z})_i \in \mathbb{R}^2$ with $i = 1, \ldots, N^2$ for the (sparse) image gradients. With these auxiliary variables, the program (10.4.7) becomes

$$\begin{aligned} \min_{\boldsymbol{z},\boldsymbol{x},\boldsymbol{v}} \ & \alpha\|\boldsymbol{x}\|_1 + \beta \sum_i \|\boldsymbol{v}_i\|_2 + \tfrac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2 \\ \text{subject to} \quad & \boldsymbol{x} = \boldsymbol{\Phi z}, \ \ \boldsymbol{v}_i = (\nabla \boldsymbol{z})_i \in \mathbb{R}^2 \ \ \forall i. \end{aligned} \tag{10.4.8}$$

Consider the augmented Lagrangian formulation of (10.4.8). We define the two functions associated with these auxiliary variables:

$$g_1(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\lambda}_1) \doteq \alpha\|\boldsymbol{x}\|_1 + \boldsymbol{\lambda}_1^*(\boldsymbol{x} - \boldsymbol{\Phi z}) + \frac{\mu_1}{2}\|\boldsymbol{x} - \boldsymbol{\Phi z}\|_2^2 \tag{10.4.9}$$

and

$$g_2(\boldsymbol{z}, \boldsymbol{v}_i, (\boldsymbol{\lambda}_2)_i) \doteq \beta\|\boldsymbol{v}_i\|_2 + (\boldsymbol{\lambda}_2)_i^*(\boldsymbol{v}_i - (\nabla \boldsymbol{z})_i) + \frac{\mu_2}{2}\|\boldsymbol{v}_i - (\nabla \boldsymbol{z})_i\|_2^2. \tag{10.4.10}$$

The augmented Lagrangian function of (10.4.8) is given by

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) \doteq g_1(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\lambda}_1) + \sum_i g_2(\boldsymbol{z}, \boldsymbol{v}_i, (\boldsymbol{\lambda}_2)_i) + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2. \tag{10.4.11}$$

Then the above constrained optimization program (10.4.8) is equivalent to the unconstrained one:

$$\min_{\boldsymbol{z},\boldsymbol{x},\boldsymbol{v},\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2} \mathcal{L}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2), \tag{10.4.12}$$

which can be optimized iteratively following the alternating direction method[10]:

$$\begin{cases} \boldsymbol{x}^{(k+1)} &= \arg\min_{\boldsymbol{x}} g_1\big(\boldsymbol{z}^{(k)}, \boldsymbol{x}, \boldsymbol{\lambda}_1^{(k)}\big), \\ \boldsymbol{v}_i^{(k+1)} &= \arg\min_{\boldsymbol{v}_i} g_2\big(\boldsymbol{z}^{(k)}, \boldsymbol{v}_i, \boldsymbol{\lambda}_2^{(k)}\big), \\ \boldsymbol{z}^{(k+1)} &= \arg\min_{\boldsymbol{z}} \mathcal{L}\big(\boldsymbol{z}, \boldsymbol{x}^{(k+1)}, \boldsymbol{v}^{(k+1)}, \boldsymbol{\lambda}_1^{(k)}, \boldsymbol{\lambda}_2^{(k)}\big), \\ \boldsymbol{\lambda}_1^{(k+1)} &= \boldsymbol{\lambda}_1^{(k)} + \mu_1\big(\boldsymbol{x}^{(k+1)} - \boldsymbol{\Phi z}^{(k+1)}\big), \\ \boldsymbol{\lambda}_2^{(k+1)} &= \boldsymbol{\lambda}_2^{(k)} + \mu_2\big(\boldsymbol{v}^{(k+1)} - \nabla \boldsymbol{z}^{(k+1)}\big). \end{cases} \tag{10.4.13}$$

[10] Here, different from the notation we have used in the optimization chapters, we will use superscript k to indicate iteration of the algorithm since the subscript i is already used for indexing the pixels.

We notice that all terms of the Lagrangian function (10.4.11) are convex functions. Hence all the above sub-programs are convex optimization.

To solve the first subprogram in (10.4.13):

$$\boldsymbol{x}^{(k+1)} = \arg\min_{\boldsymbol{x}} g_1\big(\boldsymbol{z}^{(k)}, \boldsymbol{x}, \boldsymbol{\lambda}_1^{(k)}\big),$$

although g_1 is non-differentiable with respect to $\boldsymbol{x}$, it has a closed-form solution in terms of the proximal operator for ℓ^1-norm minimization:

$$\boldsymbol{x}^{(k+1)} = \text{soft}\Big(\boldsymbol{\Phi}\boldsymbol{z}^{(k)} - \boldsymbol{\lambda}_1^{(k)}/\mu_1, \alpha/\mu_1\Big), \tag{10.4.14}$$

where recall that $\text{soft}(\cdot,\cdot)$ is the soft-thresholding operator

$$\text{soft}(x,\tau) \doteq \max\{|x| - \tau, 0\} \cdot \text{sign}\,(x)\,, \quad x \in \mathbb{R} \tag{10.4.15}$$

applied to the vector $\boldsymbol{\Phi}\boldsymbol{z}^{(k)} - \boldsymbol{\lambda}_1^{(k)}/\mu_1$ entry-wise. We leave the derivation of this as an exercise to the reader.

To solve the second subprogram in (10.4.13):

$$\boldsymbol{v}_i^{(k+1)} = \arg\min_{\boldsymbol{v}_i} g_2\Big(\boldsymbol{z}^{(k)}, \boldsymbol{v}_i, \boldsymbol{\lambda}_2^{(k)}\Big),$$

notice that g_2 is essentially a 2D version of the 1D proximal operator for the ℓ^1 norm:

$$\min_{v} \beta|v| + \frac{\mu}{2}(v-x)^2.$$

It also has a closed-form solution in terms of a 2D version of the soft thresholding:

$$\boldsymbol{v}_i^{(k+1)} = \text{soft}_2\Big(\big(\nabla\boldsymbol{z}^{(k)}\big)_i - \big(\boldsymbol{\lambda}_2^{(k)}\big)_i/\mu_2,\ \beta/\mu_2\Big), \tag{10.4.16}$$

where $\text{soft}_2(\cdot,\cdot)$ indicates the 2D shrinkage operator:

$$\text{soft}_2(\boldsymbol{x},\tau) \doteq \max\{\|\boldsymbol{x}\|_2 - \tau, 0\} \cdot \boldsymbol{x}/\|\boldsymbol{x}\|_2, \quad \boldsymbol{x} \in \mathbb{R}^2. \tag{10.4.17}$$

Again, we leave the derivation of this as an exercise to the reader.

Finally, to solve the third subprogram in (10.4.13):

$$\boldsymbol{z}^{(k+1)} = \arg\min_{\boldsymbol{z}} \mathcal{L}\Big(\boldsymbol{z}, \boldsymbol{x}^{(k+1)}, \boldsymbol{v}^{(k+1)}, \boldsymbol{\lambda}_1^{(k)}, \boldsymbol{\lambda}_2^{(k)}\Big),$$

we notice that with $\boldsymbol{x}^{(k+1)}, \boldsymbol{v}^{(k+1)}, \boldsymbol{\lambda}_1^{(k)}, \boldsymbol{\lambda}_2^{(k)}$ all being fixed, each term of the Lagrangian function $\mathcal{L}(\cdot)$ is a quadratic function in $\boldsymbol{z}$. As the optimal solution $\boldsymbol{z}^{(k+1)}$ satisfies the condition $\frac{\partial\mathcal{L}}{\partial\boldsymbol{z}}\Big|_{\boldsymbol{z}^{(k+1)}} = 0$, this gives

$$\boldsymbol{M}\boldsymbol{z}^{(k+1)} = \boldsymbol{b} \quad \text{or} \quad \boldsymbol{z}^{(k+1)} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{10.4.18}$$

where

$$\begin{aligned}
\boldsymbol{M} &= \mathcal{F}_U^*\mathcal{F}_U + \mu_1\boldsymbol{I} + \mu_2\nabla^*\nabla,\\
\boldsymbol{b} &= \mathcal{F}_U^*[\boldsymbol{y}] + \boldsymbol{\Phi}^*\Big(\mu_1\boldsymbol{x}^{(k+1)} + \boldsymbol{\lambda}_1^{(k)}\Big) + \nabla^*\Big(\mu_2\boldsymbol{v}^{(k+1)} + \boldsymbol{\lambda}_2^{(k)}\Big).
\end{aligned}$$

Here, ∇^* denotes the adjoint of the discrete derivative operator ∇.[11]

One can show that as long as the step sizes μ_1, μ_2 are chosen to be reasonably small, the above alternating minimizing scheme (10.4.13) will always converge to the optimal solution, starting from any initial conditions [YZY10].

10.5 Notes

MRI was one of the early successful applications of compressive sensing and was first convincingly verified through a series of seminal work [LDP07, LDSP08]. Many follow-up work have continued to further improve efficiency of the associated optimization methods [MYZC08, YZY10] or sampling schemes. Today compressive sensing has been widely practiced in MRI as well as many other similar medical imaging systems. For interested readers, more extensive resources about compressive sensing MRI can be found at [Lus13].

As we have seen through the physical process of MRI, the full potential of compressive sampling is still somewhat limited by what measurements we can make and at what locations with the MRI machine. Physical restrictions of the machine limit what type of sensing matrix $\boldsymbol{A}$ we may construct and hence compromise its incoherent or isometric properties. In many scientific or recreational imaging systems, however, one may have much more freedom in controlling or designing the type of measurements we may acquire, say through the so-called *coded aperture* technique [CF80]. Such methods allow us to design flexible and rich sensing schemes that can acquire the physical signals with different spatial, temporal, and spectral patterns that best match the structures of the signals. One somewhat extreme example along this line of work is the the so-called "single-pixel" camera [DDT+08]. The intention is to maximize information captured by every additional measurement in scenarios where measurements are extremely expensive or difficult (say in some outer space astronomical physical observations).

10.6 Exercises

10.1 (Compressive Sensing of Shepp-Logan Phantom*). *Design and implement a pair of efficient encoder and decoder to encode the Shepp-Logan Phantom based on the principles of compressive sensing. To measure the performance of the encoder/decoder pair, plot the PSNR curve with respect to the dimension of the compressed signal.*

10.2 (Sparse Gradient Approximation with Debiasing). *For each $\lambda \geq 0$, equation (10.4.5) computes $\hat{\boldsymbol{z}}_\lambda$ from the proximal operator of the total variation. Based on*

[11] This is the linear operator that satisfies $\langle \boldsymbol{g}, \nabla \boldsymbol{z} \rangle = \langle \nabla^* \boldsymbol{g}, \boldsymbol{z} \rangle$ for all $\boldsymbol{g}$, $\boldsymbol{z}$.

$\hat{\boldsymbol{z}}_\lambda$, one may further computer the so-called debiased estimate

$$\hat{\boldsymbol{z}}_{\lambda,\text{debiased}} = \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \quad \textit{subject to} \quad \operatorname{supp}(\|\nabla \boldsymbol{x}\|_2) \subseteq \operatorname{supp}(\|\nabla \hat{\boldsymbol{z}}_\lambda\|_2)$$

Debiasing improves fidelity to the observation $\boldsymbol{z}$, by removing shrinkage effects on the nonzeros. Show that $\hat{\boldsymbol{z}}_{\lambda,\text{debiased}}$ can be computed from $\hat{\boldsymbol{z}}_\lambda$ simply by solving a linear system of equations.

10.3 (Proximal Operators). *What is the optimal solution to the following program:*

$$\min_v \beta|v| + \frac{\mu}{2}(v-x)^2 \,? \tag{10.6.1}$$

Based on this, prove that

1 *The optimal solution for $\boldsymbol{x}^{(k+1)}$ in (10.4.13) is given by (10.4.14).*
2 *The optimal solution for $\boldsymbol{v}_i^{(k+1)}$ in (10.4.13) is given by (10.4.16).*

10.4 (MRI Recovery with Anisotropic Total Variation [WYYZ08, Bir11, BUF07, CAB+16]). *Sometimes, for simplicity, people also consider the anisotropic total variation (ATV) of the image I:*

$$\|\boldsymbol{z}\|_{\text{ATV}} \doteq \sum_i |(\nabla_1 \boldsymbol{z})_i| + |(\nabla_2 \boldsymbol{z})_i|.$$

Notice that this is exactly the ℓ^1 norm of partial derivatives of the image at all pixels. Hence, minimizing $\|\boldsymbol{z}\|_{\text{ATV}}$ would encourage the image to have a sparse partial derivatives. Let ∇ be the (finite-difference) gradient operator (∇_1, ∇_2) on the image $\boldsymbol{z}$. Then we have $\|\boldsymbol{z}\|_{\text{ATV}} = \|\nabla \boldsymbol{z}\|_1$.

We may consider replacing the TV term in (10.4.7) with the ATV: $\|\boldsymbol{z}\|_{\text{TV}} \to \|\boldsymbol{z}\|_{\text{ATV}}$:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \alpha\|\boldsymbol{\Phi}\boldsymbol{z}\|_1 + \beta\|\boldsymbol{z}\|_{\text{ATV}} + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2. \tag{10.6.2}$$

The goal of this exercise is see how to derive a simpler algorithm for the ATV regulated problem using the ALM and ADMM method discussed in Section 8.5.

Using the operator ∇, the above program can be rewritten as:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \alpha\|\boldsymbol{\Phi}\boldsymbol{z}\|_1 + \beta\|\nabla\boldsymbol{z}\|_1 + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2, \tag{10.6.3}$$

$$= \arg\min_{\boldsymbol{z}} \left\| \begin{pmatrix} \alpha\boldsymbol{\Phi} \\ \beta\nabla \end{pmatrix} \boldsymbol{z} \right\|_1 + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2. \tag{10.6.4}$$

If we denote $\boldsymbol{W} \doteq \begin{pmatrix} \alpha\boldsymbol{\Phi} \\ \beta\nabla \end{pmatrix}$ and $\boldsymbol{w} \doteq \boldsymbol{W}\boldsymbol{z} \in \mathbb{C}^{3m}$, then the above program becomes

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z},\boldsymbol{w}} \|\boldsymbol{w}\|_1 + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2 \quad \textit{subject to} \quad \boldsymbol{w} = \boldsymbol{W}\boldsymbol{z}. \tag{10.6.5}$$

Then using the Augmented Lagrange Multiplier method discussed in Chapter 8

for ℓ^1-minimization, $\boldsymbol{z}^*$ can be solved by alternatively minimizing $\boldsymbol{z}$, $\boldsymbol{w}$ and a Lagrange multiplier vector $\boldsymbol{\lambda} \in \mathbb{R}^{3m}$ in

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z},\boldsymbol{w},\boldsymbol{\lambda}} \|\boldsymbol{w}\|_1 + \boldsymbol{\lambda}^*(\boldsymbol{w} - \boldsymbol{W}\boldsymbol{z}) + \frac{\mu}{2}\|\boldsymbol{w} - \boldsymbol{W}\boldsymbol{z}\|_2^2 + \frac{1}{2}\|\mathcal{F}_U[\boldsymbol{z}] - \boldsymbol{y}\|_2^2. \quad (10.6.6)$$

We leave as an exercise to the reader to derive a detailed algorithm for (10.6.6).

11 Wideband Spectrum Sensing

"*We'll have infinite bandwidth in a decade's time.*"
– Bill Gates, PC Magazine, October, 1994

In this chapter, we present an application of compressive sensing to a crucial problem in modern wireless (radio) communication: *How can cognitive radios efficiently identify available spectrum?* We will see that this problem can be cast as one of recovering the support of a sparse signal, in the presence of noise. We will see how the methods and algorithms described in this book will allow us to break theoretical limits of conventional approaches, and once properly implemented in hardware, they can significantly advance the state of the art, by enabling better tradeoffs between energy consumption and scan time. Besides its practical importance, this application is very interesting as it is kind of *dual* to the situation in the magnetic resonance imaging that we studied in the preceding chapter. In MRI, the measurements are the Fourier transform of the image of interest and the sparse patterns are in the image domain; whereas for spectrum sensing, the sparse patterns are in the Fourier domain which we do not measure directly.

11.1 Introduction

11.1.1 Wideband Communications

In modern wireless (radio) communication systems, it is common for a wide radio spectrum range to be shared by many users. A classic protocol for sharing a wide spectrum is to divide the spectrum into multiple narrow bands. Each individual user transmits a narrow-band signal within the designated channel band by modulation, typically by multiplying a periodic "carrier signal" with a frequency at the center of the assigned band. To be more precise, let us assume that the entire available spectrum is between $(f_{\min}, f_{\max})$[1]. We denote the bandwidth of the spectrum as $W = f_{\max} - f_{\min}$. If the spectrum is divided into N_0

[1] For a real signal, its Fourier transform is symmetric in the frequency domain. So for simplicity, we will only talk about the positive (or upper) range of the spectrum $(f_{\min}, f_{\max})$, the corresponding negative (lower) spectrum $(-f_{\max}, -f_{\min})$ is assumed to be available too by default.

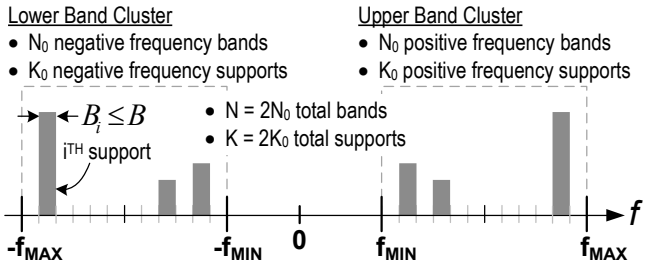


Figure 11.1 A wideband spectrum between $(f_{\min}, f_{\max})$ (and $(-f_{\max}, -f_{\min})$ is divided into multiple narrow bands of width B. At any given time a (sparse) number of channels are actively in use.

narrow bands, then each individual channel has a resolution bandwidth (RBW) $B = W/N_0$. See Figure 11.1 for an illustration.

11.1.2 Nyquist Sampling and Beyond

To recover the signals at the receiver side, one needs to demodulate the signal from its carrier, sample the signal at a high frequency through an Analog-to-Digital converter (ADC), and then filter through a low-pass filter. The classic *Nyquist sampling theorem* [OSB99] in digital signal processing stipulates that, to perfectly recover an analog band-limited signal, say $x(t)$, from its discrete (periodic) samples $\{x(nT)\}_{n\in\mathbb{Z}}$, one needs to sample the signal at a frequency $f_s = 1/T$ at least twice as the signal's possible bandwidth, known as the Nyquist rate. Hence in the above wideband setting, if a receiver does not know the carrier frequencies of the (active) channels,[2] in order to recover the narrow-band signals in every possible channel, one needs to demodulate the signals by sampling at a rate higher than twice the spectrum bandwidth W, that is:

$$f_s \geq 2W.$$

If so, any signal $x(t)$ within this spectrum can be perfectly recovered from its samples $\{x(nT)\}_{n\in\mathbb{Z}}$ via the so called *cardinal series*:

$$x(t) = \sum_{n\in\mathbb{Z}} x(nT)\mathrm{sinc}(t/T - n),$$

or other similar interpolation schemes [OSB99].

For wideband communication, however, the Nyquist rate $2W$ often exceeds the specifications of typical analog-to-digital converters (ADC) by magnitudes. For example, in year 2012, the US President's Council of Advisors on Science and Technology (PCAST) recommended sharing 1 GHz of federal government

[2] which is quite common in many applications such as interference detection. However, if the carrier frequency is known, the receiver can simply demodulate the signals at the carrier frequency [Lan67].

spectrum from 2.7 to 3.7 GHz with nongovernmental entities for public use. The Nyquist rate would require an ADC of 2 GHz! Given that the actual bandwidth B of the signals in each channel is rather small[3] compared to the entire spectrum, demodulating at the Nyquist rate for every channel seems rather demanding and likely unnecessary too.

As mobile wireless devices such as cellular phones and personal computers have become ubiquitous in modern day life, it has become increasingly critical to improve the efficiency of spectrum sharing as well as improve the power efficiency of individual mobile devices. In terms of spectrum usage, modern mobile devices are very different from conventional wireless communication systems such as radio broadcasting. At any given time and place, only a relatively small number of devices/users may be active. Hence such devices do not need designated channels at all time and can share a common spectrum via certain data transmission protocols (such as in WiFi). As Figure 11.1 has illustrated, although the PCAST spectrum can simultaneously support N_0 narrow bands, at any given time or place, only a small number of say K_0 bands are active and any new user does not know in advance which bands are being occupied. In such new scenarios, compressive sensing is relevant and beneficial: if the support of a signal is *sparse in the spectrum*, the necessary sampling rate for signal recovery can be significantly lower than the Nyquist rate $2W$. For instance, using techniques such as *random demodulation* [Tro10], one only needs a sampling rate at

$$f_s = O(K_0 \log(W/K_0))$$

to stably reconstruct the signal, which is exponentially lower than $2W$. A more practical scheme named *modulated wideband converter* [ME10, ME11] requires only a sampling rate at

$$f_s = 2K_0 B,$$

which is usually magnitudes lower than the Nyquist rate when $K_0 \ll N_0$.

11.2 Wideband Interferer Detection

The next generation 5G technologies like Long-Term Evolution (LTE) aim to utilize under-utilized unlicensed public spectrum (like the PCAST spectrum mentioned above) in addition to designated licensed spectrum. Figure 11.2 shows an example of such a deployment. In order to utilize and share the unlicensed spectrum efficiently with all other possible users, the user terminal needs to sense in real time which channels have been occupied by other users (called interferers) so that it can opportunistically use other idle channels for subsequence

[3] Radios stations are typically assigned a 200 KHz bandwidth. That is more than enough for most audio signals at 20 KHz $\sim$ 30 KHz range. For data transmission tasks of mobile devices, the desired bandwidth is typically 20MHz.

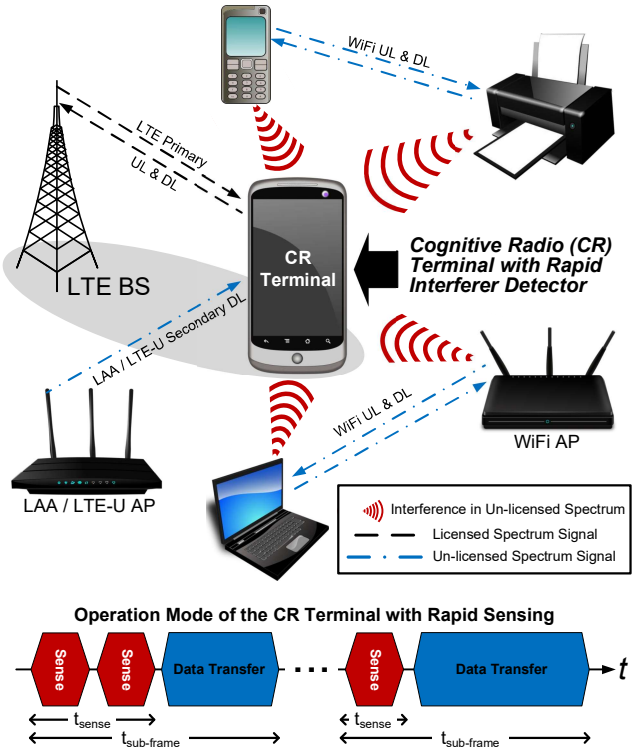


Figure 11.2 Illustration of deployment of LTE-Unlicensed using cognitive ratio (CR) to detect active interferers.

data transfer. Terminals with such capabilities are called cognitive radio (CR) terminals.

To model the interference, we may assume that the entire spectrum $(f_{\min}, f_{\max})$ are partitioned into N_0 bands. We say a band is occupied (or used) by an interferer (or another user) if the energy on that band is above certain threshold (say above background radio noise level). At any given time, we assume K_0 out of the N_0 bands have been occupied by interferers, as illustrated in Figure 11.3. We call the aggregated signal of all the interferers as $x(t)$. The problem of interference detection is to find out the supports of the K_0 bands of $X(f)$, the Fourier transform of $x(t)$.

11.2.1 Conventional Scanning Approaches

Conventionally, there are two straightforward approaches to detect (the support of) the interfering signal $x(t)$ in the frequency domain:

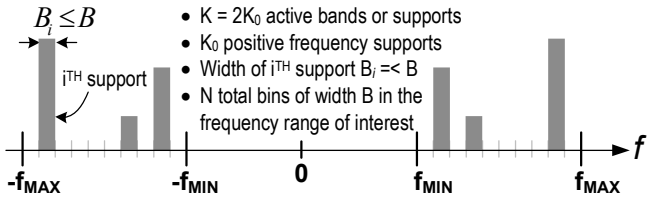


Figure 11.3 At any given time a sparse number of K_0 channels are actively in use.

1 *Scan one band at a time:* For each of the N_0 bands, one can first down-convert the signal using a local oscillator with frequency f_{lo} at the center of each band

$$f_{lo} = f_{\min} + 0.5B + iB, \quad i = 0, \ldots, N_0 - 1,$$

and then sample the signal at the Nyquist rate for each band

$$f_s = 2B.$$

This allows one to recover the component of $x(t)$ in each band and determine if that band has been occupied. Obviously, one needs to repeat this process N_0 times, one for each band, or one can build a system with N_0 parallel branches, again one for each band.

2 *Recover all bands together:* One can first down-convert the signal using a local oscillator with frequency f_{lo} at the center of the entire spectrum

$$f_{lo} = (f_{\min} + f_{\max})/2,$$

and then sample the signal at the Nyquist rate for the entire spectrum

$$f_s = 2W = 2(f_{\max} - f_{\min}).$$

This allows one to recover the entire signal $x(t)$ within the spectrum, regardless of which bands have been occupied.

Despite their simplicity, these approaches are costly either in time (e.g. scanning N_0 times), or in hardware complexity (e.g. building N_0 branches), or in energy consumption (e.g. sampling at the high Nyquist rate $2W$).

As an example, Figure 11.4 illustrates applying the above schemes to the PCAST spectrum. For a sweeping spectrum scanner (Fig. 11.4(a)), each frequency bin is scanned sequentially by progressively sweeping the local oscillator (LO) driving the downconverter. This architecture requires widely tunable, high quality RF components that are difficult to implement on a chip. Identifying signals over a 1GHz span with a 20MHz RBW requires a long scan time which is proportional to the number of bins $N_0 = 50$. This results in large energy consumption and the risk of missing fast changing interferers.

The scan time in sweeping scanners can in principle be reduced by using a

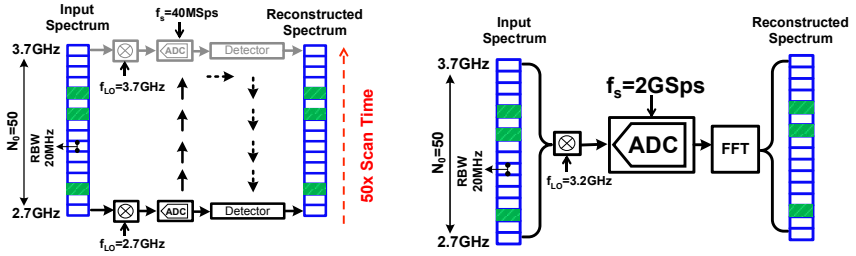


Figure 11.4 Conceptual illustration of the operation of traditional spectrum analysis techniques applied for a 2.7-3.7GHz spectrum analyzer with a 20MHz RBW; the occupied spectrum bins are shaded in green: (a) sweeping spectrum scanner, (b) Nyquist-rate FFT spectrum sensor.

multi-branch architecture with multiple narrowband scanners operating in parallel. However, the hardware complexity becomes impractical since each branch requires a separate phase-locked loop (PLL) to generate the LO signal and the 50 PLL frequencies would need to be spaced closely with a distance equal to the 20MHz RBW.

A Nyquist-rate FFT spectrum sensor (Fig. 11.4(b)) for a 1GHz bandwidth would require a prohibitively high aggregate analog-to-digital (A/D) conversion rate of 2GSps after I/Q downconversion. Even though the scan time is reduced, this is a power hungry approach due to the high sampling rate required for the Nyquist-rate wideband sensing.

How can we do better? As we have mentioned earlier, at any given time, the number of bands used by other users, K_0, is typically sparse with respect to N_0. By exploiting this additional knowledge about the spectrum of the interference $x(t)$, i.e., $X(f)$ being sparse, we can come up with much more efficient solutions than the above approaches using techniques from compressive sensing. It has been well studied in Chapter 3 that one can recover a sparse signal from a small number of random (incoherent) linear measurements. However, here the sparsity is in the frequency domain and we need to know how to effectively and efficiently take random linear measurements of $X(f)$.

11.2.2 Compressive Sensing in the Frequency Domain

To take a random linear measurement of the spectrum $X(f)$, a very clever scheme has been suggested by [ME10] and [HYP+15]: one can first multiply $x(t)$ with a periodic mixing function $p(t)$ say of period T_p. The mixed signal is then truncated with a low-pass filter $h(t)$ with cutoff frequency $1/(2T_s)$ and the filtered signal is then sampled at rate $f_s = 1/T_s$. The hope is that, for properly chosen mixing function $p(t)$, T_p, and T_s, the (discrete-time) Fourier transform of the

output sequence, say $y(n)$, would be precisely random linear measurements of the (sparse) spectrum $X(f)$. Below we give a brief sketch of this scheme.

The mixing function $p(t)$, as a T_p-periodic function, can be written as a Fourier expansion:

$$p(t) = \sum_{l=-\infty}^{\infty} c_l e^{\mathrm{i}\frac{2\pi}{T_p}lt}, \tag{11.2.1}$$

where $\mathrm{i} = \sqrt{-1}$ is the imaginary unit and c_l is the Fourier coefficient: $c_l = \frac{1}{T_p}\int_0^{T_p} p(t)e^{-\mathrm{i}\frac{2\pi}{T_p}lt}dt$.

After $x(t)$ is mixed with $p(t)$, the Fourier transform of the mixed signal $\tilde{x}(t) = x(t)p(t)$ would be

$$\tilde{X}(f) = \sum_{l=-\infty}^{\infty} c_l X(f - lf_p), \tag{11.2.2}$$

where $f_p = 1/T_p$. Since $X(f)$ is band-limited, the above sum will only have finite terms.

If the subsequent filter $h(t)$ is perfect low-pass filter, only the frequencies in the interval $\big(-\frac{1}{2}f_s, +\frac{1}{2}f_s\big)$ will stay in the sequence $y[n]$. Hence, the discrete-time Fourier transform of $y[n]$ has the expression:

$$Y(f) = \sum_{l=-L_0}^{L_0} c_l X(f - lf_p), \quad f \in \Big(-\frac{1}{2}f_s, +\frac{1}{2}f_s\Big), \tag{11.2.3}$$

where L_0 is large enough to cover the support of $X(f)$.

For simplicity, we may stack all the coefficients c_l into a vector

$$\boldsymbol{c} \doteq [c_{L_0}, \ldots, c_{-L_0}]^*$$

of length $L = 2L_0 + 1$ and $X(f - lf_p)$ into another vector:

$$\boldsymbol{z}(f) \doteq [X(f - L_0 f_p), \ldots, X(f + L_0 f_p)]^*. \tag{11.2.4}$$

The vector $\boldsymbol{z}(f)$ is sparse if $X(f)$ is. We can write the above expression as

$$Y(f) = \boldsymbol{c}^* \boldsymbol{z}(f). \tag{11.2.5}$$

The remaining question is how to properly choose the T_p-periodic mixing function $p(t)$ so that the expression in (11.2.3) would be a sufficiently random (or incoherent) measure of non-zero components in $X(f)$. An easy scheme is to make the values of $p(t)$ in each of its period $(0, T_p)$ be a pseudo-random bit sequence (PRBS) of length L:

$$p(t) = \alpha_k, \quad k\frac{T_p}{L} \le t \le (k+1)\frac{T_p}{L}, \quad 0 \le k \le L-1, \tag{11.2.6}$$

where α_k is a random variable taking binary values in $\{-1, +1\}$ of equal probability. For so-chosen $p(t)$, its Fourier coefficients c_l can be computed as

$$c_l = \frac{1}{T_p}\int_0^{\frac{T_p}{L}} \sum_{k=0}^{L-1} \alpha_k e^{-\mathrm{i}\frac{2\pi}{T_p}l(t+k\frac{T_p}{L})}dt = \sum_{k=0}^{L-1} \alpha_k e^{-\mathrm{i}\frac{2\pi}{L}lk}\frac{1}{T_p}\int_0^{\frac{T_p}{L}} e^{-\mathrm{i}\frac{2\pi}{T_p}lt}dt.$$

Let us define the scalar $d_l \doteq \frac{1}{T_p}\int_0^{\frac{T_p}{L}} e^{-\mathrm{i}\frac{2\pi}{T_p}lt}dt$ and let $\boldsymbol{D}$ be the diagonal matrix with d_l on its diagonal. Notice that $\{e^{-\mathrm{i}\frac{2\pi}{L}lk}\}$ are exactly the (k,l)-th entry of the discrete Fourier transform matrix $\boldsymbol{F}$ of size $L \times L$. So we have

$$\boldsymbol{c}^* = \boldsymbol{a}^*\boldsymbol{F}\boldsymbol{D}, \tag{11.2.7}$$

where $\boldsymbol{a} = [\alpha_0, \alpha_1, \ldots, \alpha_{L-1}]^*$ is the sequence of random bits.

Combining the above equation with the measurement equation (11.2.5), we have

$$Y(f) = \boldsymbol{a}^*\boldsymbol{F}\boldsymbol{D}\boldsymbol{z}(f). \tag{11.2.8}$$

The above equation is obtained from mixing with one signal $p(t)$ from one pseudo random bit sequence $\boldsymbol{a}$. To recover the sparse vector $\boldsymbol{z}(f)$, we can mix the input $x(t)$ with multiple signals $p_i(t), i = 1, \ldots, m$, each with an independent pseudo random bit sequence $\boldsymbol{a}_i$. We collect all the measurements $Y_i(f)$ into one vector $\boldsymbol{y}(f) = [Y_1(f), \ldots, Y_m(f)]^*$. Then we have

$$\boldsymbol{y}(f) = \boldsymbol{A}\boldsymbol{F}\boldsymbol{D}\boldsymbol{z}(f), \tag{11.2.9}$$

where $\boldsymbol{A}$ is $m \times L$ matrix containing all the independent pseudo random bit sequences $\boldsymbol{a}_i$ as its rows.

Notice that the diagonal operator $\boldsymbol{D}$ does not change the sparsity of $\boldsymbol{z}(f)$ and the DFT matrix $\boldsymbol{F}$ is unitary. As we have known from the analysis in Chapter 3, the $m \times L$ measurement matrix $\boldsymbol{A}\boldsymbol{F}$ would be highly incoherent and the so obtained measurements $\boldsymbol{y}(f)$ would be a set of incoherent measurements of $\boldsymbol{z}(f)$. As long as m is large enough, say in the order $O(K_0 \log(L/K_0))$, we are guaranteed to correctly recover the sparse vector $\boldsymbol{z}(f)$ using the ℓ^1-minimization:

$$\min \|\boldsymbol{z}(f)\|_1 \quad \text{subject to} \quad \boldsymbol{y}(f) = \boldsymbol{A}\boldsymbol{F}\boldsymbol{D}\boldsymbol{z}(f). \tag{11.2.10}$$

In theory, one may solve the above ℓ^1-minimization problem to identify the support of the used bands. However, to minimize processing memory and power in hardware implementation, instead of using generic convex optimization methods introduced in Chapter 8, the greedy algorithms such as the Orthogonal Matching Pursuit (OMP) Algorithm 8.11 of Chapter 8 becomes well suited for our purpose here: The OMP is a simple greedy heuristic for sparse recovery, which forms an estimate of the signal support one element at a time. In each iteration, the algorithm involves minimal set of columns of the sensing matrix, here the matrix $\boldsymbol{A}\boldsymbol{F}\boldsymbol{D}$. It offers an attractive trade-off between algorithm simplicity and recovery guarantees [TG07], hence better suited for low-level hardware implementation than other generic ℓ^1 solvers.

11.3 System Implementation and Performance

The remaining issue is how one can implement the above spectrum sensing scheme with a practical real hardware system design? The resulting system

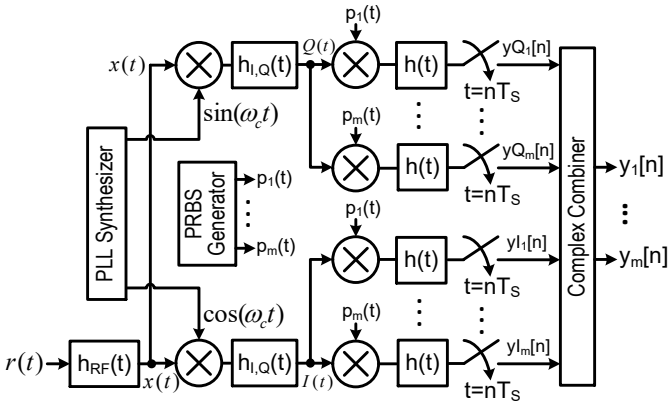


Figure 11.5 System diagram of the quadrature analog to information converter (QAIC).

should be able to realize the theoretical benefits of compressive sensing and break a good balance between power consumption, scanning time, and hardware complexity. The goal is to achieve significantly improved performance than the conventional approaches mentioned earlier. We here introduce one such system, the so-called Quadrature Analog to Information Converter (QAIC) system [HYP+15, YHW+15], for energy-efficient wideband spectrum sensing.

11.3.1 Quadrature Analog to Information Converter

The QAIC illustrated in Figure 11.5 consists of three major functional blocks - an RF downconverter, I and Q path modulator banks (mixers, filters and analog-to-digital converters), and a pairwise complex combiner. The input signal $x(t)$ is first down-converted to complex baseband with the in-phase branch I and the quadrature-phase Q. The downconverter outputs $I(t)$ and $Q(t)$ are multiplied by a periodic pseudo-random bit sequence (PRBS) $p_i(t)$, then filtered and sampled at a low rate in the I and Q path modulator banks. The QAIC exploits the compressive spectrum sensing principles discussed above: multiplication by the PRBS aliases the spectrum such that a potion from each band of the downconverter output signals $I(t)$ and $Q(t)$ appears at a low frequency centered around DC. The outputs of the I and Q path modulator banks are pairwise added by the complex combiner to select either the upper $(f_{\min}, f_{\max})$ or lower $(-f_{\max}, -f_{\min})$ band cluster of the input signal $x(t)$. The I and Q modulator banks of the QAIC consist of multiple branches each employing a different PRBS such that in principle a sufficiently large number of band mixtures output $y_1[n] \dots y_m[n]$ allows us to recover the sparse multiband signal $x(t)$.

For QAIC, the frequency of the downconverter is chosen to be

$$f_c = (f_{\max} + f_{\min})/2$$

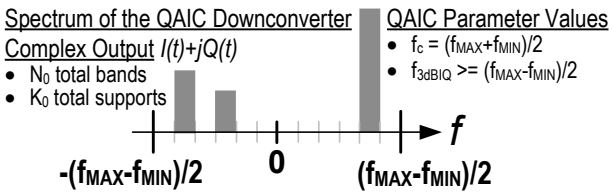


Figure 11.6 QAIC downconverter output at complex baseband.

and $\omega_c = 2\pi f_c$. This shifts the spectrum of $x(t)$ from $(f_{\min}, f_{\max})$[4] to the base range $\big(-(f_{\max}-f_{\min})/2, +(f_{\max}-f_{\min})/2\big)$, centered around the DC, as shown in Figure 11.6. The low-pass filter $h_{I,Q}(t)$ extracts this base band with a cutoff frequency

$$f_{I,Q} = (f_{\max} - f_{\min})/2.$$

The I and Q path modulator banks employed by the QAIC together process a complex signal $I(t) + j \cdot Q(t)$ at baseband. As a result, the QAIC is able to isolate and process either the upper $(f_{\min}, f_{\max})$ or the lower $(-f_{\max}, -f_{\min})$ band cluster of the $x(t)$. The spectrum of the QAIC downconverter complex output configured to retain the upper band cluster of $x(t)$ is shown in Figure 11.6.

The span of the QAIC extends from roughly $f_{\min}$ to $f_{\max}$ and QAIC simultaneously observes all bands within this frequency span. Therefore the instantaneous bandwidth of the QAIC is roughly $(f_{\max} - f_{\min})$ Hz, which is partitioned into $N_0 = \lceil (f_{\max} - f_{\min})/B \rceil$ bands with K_0 active bands. With the downconversion, the frequency of the pseudo random bit sequence f_p can be chosen to be

$$f_p = (f_{\max} - f_{\min})/2.$$

Based on the theory of compressive sensing, the number of measurements we need is $m = C_Q K_0 \log(N_0/K_0)$. Due to the quadrature configuration, the total number of branches is then

$$M = 2m = 2C_Q K_0 \log(N_0/K_0),$$

and the output sampling rate (hence the cutoff frequency of the filter $h(t)$ in Figure 11.5) can be half of the band resolution:

$$f_s = B/2.$$

The number of branches M may be traded (say reduced by an integer factor q) for the branch sampling rate f_s (increased by the same factor q).

[4] Similar for the lower $(-f_{\max}, -f_{\min})$ band cluster.

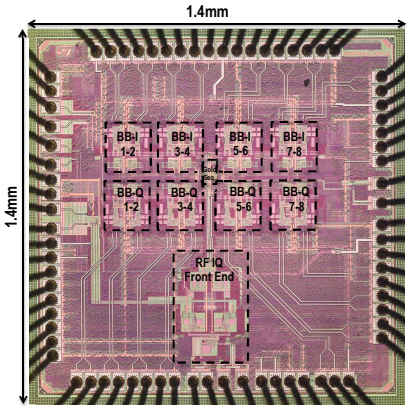


Figure 11.7 Die photograph of the 65nm QAIC prototype

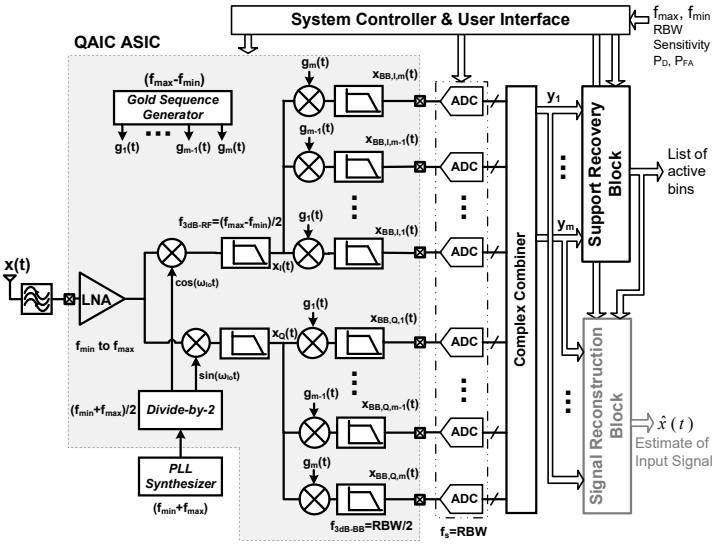


Figure 11.8 Block diagram of the rapid interferer detector based on band pass compressed sampling with a QAIC.

11.3.2 A Prototype Circuit Implementation

Based on the above design, a first prototype circuit implementation of the QAIC system for detecting up to three interferers in the 2.7–3.7 GHz PCAST spectrum was introduced by [YHW+15]. The circuit was integrated in a chip implemented with the 65 nm CMOS GP technology, with an active area of 0.428 mm^2. A photograph of the die of the prototype system is shown in Figure 11.7. We in this section give a brief description of the prototype system.

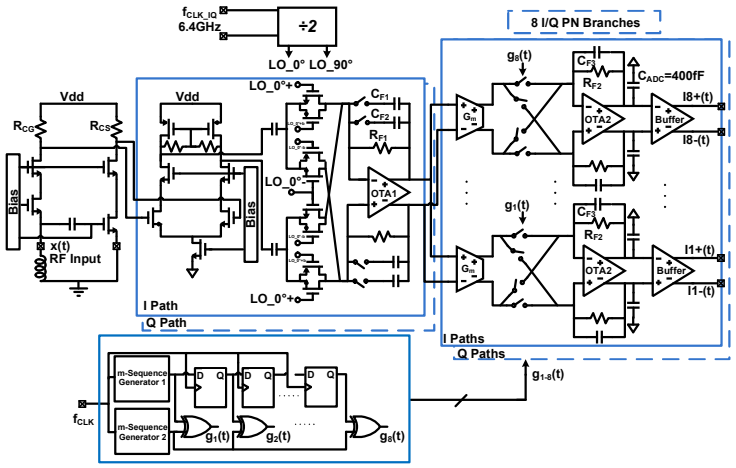


Figure 11.9 Circuit implementation details of the QAIC front end.

Figure 11.8 shows the block diagram of the prototype system that employs the QAIC design. The system controller configures the QAIC hardware and software resources according to user specified system constants and performance targets such as RBW, sensitivity, maximum and minimum frequencies of interest, $f_{\max}$ and $f_{\min}$ etc.

The PCAST spectrum is a 1GHz spectrum ranging from 2.7 to 3.7GHz with a RBW of 20MHz. For the QAIC design, $m = 8$ I/Q branches would be sufficient, which is a total of $M = 16$ physical branches. The length of random sequence is chosen to be $L = 63$. More detailed justification of the chosen parameters and other specifications of the system can be found in [YHW+15].

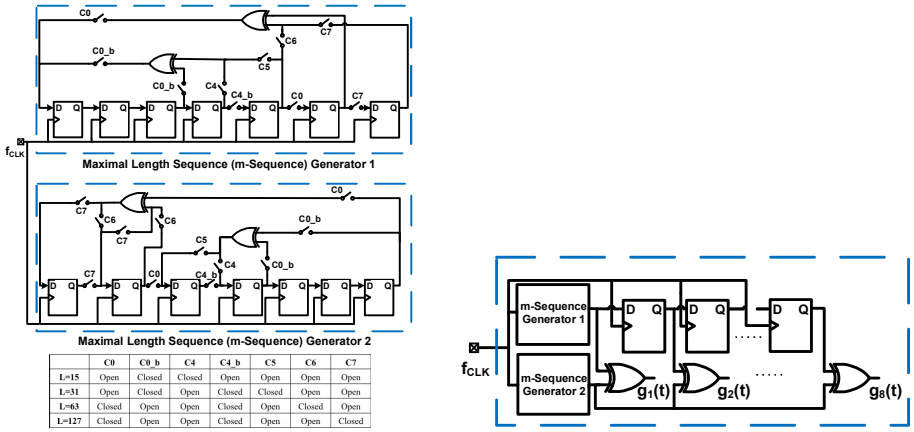
Compared to the conventional approaches mentioned in section 11.2.1, the QAIC based spectrum sensor is 50 times faster scan time compared to the sweeping spectrum scanners while 6.3 times compression in the aggregate sampling rate (or in the number of branches) compared to multi-branch spectrum sensors and Nyquist-rate FFT spectrum scanners.

Circuit Implementation of the RF Front-End Blocks.
The 2.7-3.7GHz QAIC prototype circuit implementation is shown in Fig. 11.9. It implements the functions in the shaded box in the system diagram in Fig. 11.8. The chip has been implemented in a 65nm CMOS GP technology. The QAIC chip uses a wideband noise-canceling low-noise amplifier (LNA) [BKN04,BKLN08]. A wideband noise-canceling LNA is preferred since impedance matching is required for an instantaneous bandwidth of 1GHz. The post-layout simulated LNA gain for typical process corner is 15.8dB to 14.6dB from 2.7 to 3.7GHz and the simulated $S_{11} < -10dB$ for a wide bandwidth from 1 to 3.7GHz for typical process corner. The measured LNA power consumption is 14mW from 1.1V supply.

The LNA is followed by current-driven passive I/Q mixers and transimpedance amplifiers (TIAs) [MDL+09, BMC+06, Raz98]. The input stage is implemented as a transconductance G_m amplifier operating at an RF frequency range 2.7 to 3.7GHz followed by four pairs of CMOS transmission gate switches driven by complementary clock phases at 3.2GHz. An off-chip RF clock fed to the chip is 6.4GHz and 3.2GHz quadrature LO signals with a 50% duty cycle driving the RF I/Q downconverter mixers, $\cos(\omega_{lo}t)$ and $\sin(\omega_{lo}t)$, are generated by the on-chip divide-by-2 circuit that is followed by clock buffers and a non overlap generator that is formed by two cross-coupled NAND gates with inverter chains to generate complementary phase clocks for transmission gate type passive mixer switches. The down-converted current signal is converted into a voltage output by a transimpedance amplifier that is configured as an RF I/Q filter. Single stage OTA topology [Raz01] is chosen for RF I/Q filter design since it was critical to achieve a wide 500MHz bandwidth while driving the 8 I/Q paths and minimizing the power consumption. Measured power consumption of the RF I/Q downconversion stage including the current-driven passive I/Q mixers, TIA based filters and I/Q LO generation based on divide-by-2 circuitry is 20.9mW from 1.1V supply.

PN Sequence Generation and CS Baseband Circuits.
The RF TIAs drive eight I/Q paths, each with a current-driven passive mixer and TIA used as a baseband filter loaded with 400fF emulating the equivalent load of an 8-bit ADC (C_{ADC} in Fig. 11.9). Measured power consumption of the 8 I/Q PN branches is 38.9mW from a 1.1V supply.

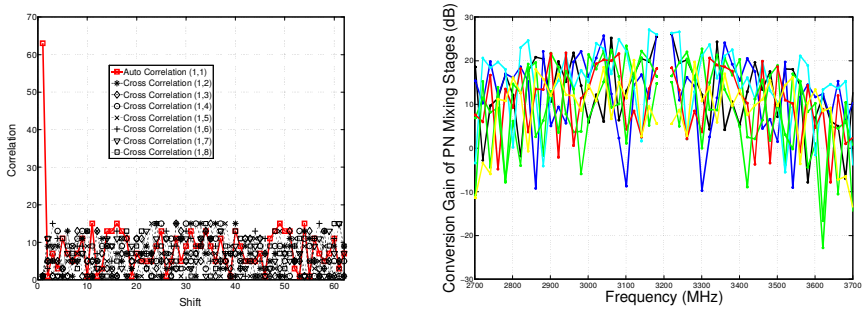
The I/Q mixing stages are driven by 8 unique gold sequences [PSM82, Gol67] generated on-chip with a gold sequence generator. Gold sequences are preferred because a large set of periodic sequences with good cross-correlation and auto-correlation properties can be generated with less circuitry compared to a shift register implementation [PSM82]. Gold sequences generated from preferred m-sequence pairs satisfy the following inequalities for cross-correlation, θ [PSM82, Gol67]: $|\theta| \leq t = 2^{(n+2)/2} + 1$, *n even* and $|\theta| \leq t = 2^{(n+1)/2} + 1$, *n odd*. The on-chip gold sequence generator (Fig. 11.10) has various length options of 15, 31, 63 and 127 for programmable RBW options and the switches $C0$, $C0_b$, $C4$, $C4_b$, $C5$, $C6$ and $C7$ are used to control the length of the gold sequences by changing the length of the m-sequences. It generates 8 $(2^n - 1)$ long gold sequences by XORing two m-sequences generated by two n-flip-flop LFSRs. By keeping one m-sequence (Fig. 11.10(a)) the same and delaying the other one before the XOR, up to $2^n - 1$ distinct gold sequences (Fig. 11.10(b)) can be generated with sufficiently low cross-correlation. Fig. 11.11(a) shows the autocorrelation and cross-correlation properties of one of the 8 unique gold sequences for a length of 63 which satisfy the sequence requirements (i.e. θ). Fig. 11.11(b) shows the measured input referred conversion gain from 2.7 to 3.7GHz of the 8 PN I/Q



	C0	C0_b	C4	C4_b	C5	C6	C7
L=15	Open	Closed	Closed	Open	Open	Open	Open
L=31	Open	Closed	Open	Closed	Closed	Open	Open
L=63	Closed	Open	Open	Closed	Open	Closed	Open
L=127	Closed	Open	Open	Closed	Open	Open	Closed

(a) (b)

Figure 11.10 Circuit implementation details of gold sequence generator for 8 unique gold sequences with low cross-correlation operating at 1.26GHz for length 15, 31, 63 and 127; (a) Two unique m-sequence generators based on an LFSR implementation (b) 8 unique gold sequences generation based on the two unique m-sequences with RBW programmability.



(a) (b)

Figure 11.11 Properties of the 8 unique gold sequences generated on chip; (a) Autocorrelation and cross-correlation properties of one of the 8 gold sequences is shown for a shift of 63 for a length 63; (b) Input referred conversion gain from 2.7 to 3.7GHz of the 8 PN mixing stages driven by the 8 gold sequences for a length of 63, and RBW of 20MHz.

mixing stages driven by 8 unique gold sequences for a RBW of 20MHz[5]. Mea-

[5] Some of the implemented gold sequences are balanced while others are unbalanced. Balanced gold sequences have better spectral properties (i.e. are more evenly distributed) [Hol07]. Also 8 unique m-sequences that are known to have uniform (evenly distributed) spectrum can be used in future work to overcome the conversion gain fluctuations over frequency.

sured power consumption of the on-chip gold sequence generator for the nominal length of 63 is 7.04mW from 1.1V supply.

CS Digital Signal Processing.
As we have mentioned before, the Orthogonal Matching Pursuit (OMP) Algorithm 8.11 of Chapter 8 is used to identify the input bands that exceed a user-defined threshold. The OMP stopping criterion is derived from the system dimension and a user-defined threshold. This threshold can be set to maximize the detection probability P_D or minimize the false alarm probability P_{FA}. In this work, the threshold is set close to the QAIC noise floor to maximize P_D performance of the system.

Overall System Performance.
The prototype QAIC system front end is implemented in 65 nm CMOS with a size 0.43 mm^2 and consumes 81 mW from a 1.1 V supply. It can detect up to three interferers in a frequency span of 1 GHz ranging from 2.7 to 3.7 GHz (PCAST Band) with a resolution bandwidth of 20 MHz in 4.4μS, 50 times faster than traditional sweeping spectrum scanners. Rapid interferer detector with the bandpass QAIC is two orders of magnitude more energy efficient than traditional Nyquist-rate architectures and one order of magnitude more energy efficient than previous low-pass CS methods. The aggregate sampling rate of the QAIC interferer detector is compressed by 6.3 compared to traditional Nyquist-rate architectures for the same instantaneous bandwidth.

11.3.3 Recent Developments in Hardware Implementation

Since the first prototype [YHW+15], two new chips have been designed to further improve the system's efficiency and compatibility with other communication hardware systems.

Time-segmented QAIC.
[YHK+16] introduced a new chip design that realizes a rapid interferer sensing solution that employs compressed sampling with a time-segmented quadrature analog-to-information converter (TS-QAIC). TS-QAIC enables system scalability by adaptive thresholding in the information recovery engine and by extending the 8 physical I/Q branches of the QAIC to 16 with time segmentation, while limiting the silicon cost and complexity. TS-QAIC can detect up to 6 interferers (compared to 3 for QAIC) over a 1GHz bandwidth between 2.7-3.7GHz in 10.4μS with only 8 I/Q physical branches. The TS-QAIC prototype is implemented in 65nm CMOS on a 0.517 mm^2 active area and consumes 81.2mW from a 1.2V supply.

Direct RF-to-Information Converter.
The Direct RF-to-information Converter (DRF2IC) [HBZ+17] unifies high sensitivity signal reception, narrowband spectrum sensing, and energy-efficient wideband interferer detection into a fast-reconfigurable and easily scalable architecture. In reception mode, the DRF2IC RF front-end (RFFE) consumes 46.5mW and delivers 40MHz RF bandwidth, 41.5dB conversion gain, 3.6dB NF and -2dBm B1dB. 72dB out-of-channel blocker rejection is achieved in narrowband sensing mode. In compressed sensing wideband interferer detection mode, 66dB operational dynamic range, 40dB instantaneous dynamic range, 1.43GHz instantaneous bandwidth is demonstrated and 6 interferers scattered over 1.26GHz are detected in 1.2μS consuming 58.5mW.

11.4 Notes

This chapter is based on a series of work [YHW+15, YHK+16, HBZ+17]. Acute readers may already draw some interesting comparisons between the spectrum sensing problem and the MRI problem studied in the previous Chapter 10: For MRI, our direct measurements are Fourier transform of a signal (the brain image) in the spectral domain and yet the sparse structure of the signal is in a different wavelet domain or is in the spatial characteristics (spatial derivatives) of the signal. In the spectrum sensing problem, our measurements are samples of a temporal signal whose sparse structure is in its spectral domain. Hence in MRI, we need to transform the measurements back to the spatial domain to impose sparsity whereas in spectrum sensing, we are very much doing the opposite: need to transform the signal to its spectral domain first in order to discover the sparse structure.

Signal versus Support Recovery.
There is also another difference in what we are interested in about the signals. In the MRI problem, we are interested in recovering the signal as accurately as possible whereas in the spectrum sensing, we are interested in recovering *only the support* of the sparse pattern in the spectral domain, as long as the signal is above certain confidence threshold on a band of interest. Related theory was characterized in Section 3.6.4 of Chapter 3. This will also be the case for the face *recognition* problem to be studied in Chapter 13 and more general *classification* problems in Chapter 16. The difference in purpose would determine how much resources we should allocate in terms of the number of measurements and the computational complexity. In particular, we can choose different algorithms to achieve different accuracies in the sparse solution recovered. Of course, the choice in algorithms and accuracies also depend on whether the recovery needs to be in real time (for spectrum sensing) or can be done offline (for recovering MRI). The principles and methods introduced in this book, once properly customized

for every different application, would enable us to achieve different goals with the minimal measurement and computational resources.

12 Scientific Imaging Problems

"Where the telescope ends, the microscope begins. Which of the two has the grander view?"

– Victor Hugo, *Les Misérables*

12.1 Introduction

In this chapter, we consider a form of low-dimensional structure that arises in many applications in scientific data analysis: we consider datasets consisting of a few basic motifs, repeated at different locations in space and/or time. Figure 12.1 shows three examples of this structure: in neuroscience, in which the motifs represent spike patterns of a neuron [SGHK03, GK12], in image deblurring [CW98, RBZ06, LWDF11], and in microscopy, in which the motifs represent repeated features of interest in a sample [CSL+20]. This is a very simple and fundamental type of low-dimensional structure. However, it raises challenges both for theory and computation. Typically, both the motifs and their locations are not known ahead of time. As discussed in Chapter 7, this naturally leads to *nonconvex* optimization problems, which can be studied through their symmetries, and solved efficiently using methods introduced in Chapter 9. In this chapter, we motivate this model in more depth using an example from a particular scientific imaging modality, scanning tunneling microscopy [BR83]. We also emphasize the particular challenges arising in motif finding, which force us to go beyond the simpler theoretical settings of Chapter 7.

12.2 Data Model and Optimization Formulation

In this section, we focus on one particular imaging modality, *Scanning Tunneling Microscopy* (STM), which gives rise to images that consist of repeated motifs. STM produces atomic resolution images of the quantum electronic structure of the surface of a material [BR83]. In this modality, a conducting tip is rastered across the surface of a sample of interest. A two-dimensional image of the surface can be constructed by recording at each spatial location the tip height needed

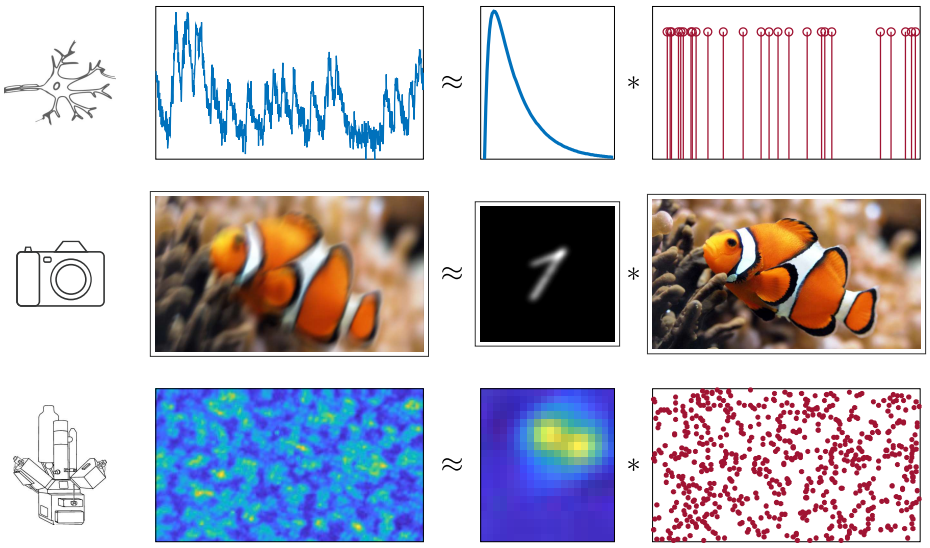


Figure 12.1 Natural Signals with Short-and-Sparse Structures. In calcium imaging (top), each neuronal spike induces a fluorescence pattern measuring a transient increase in calcium concentration. In photography (middle), photos with sharp edges (sparse in the gradient domain) are often obfuscated by blurring due to shaking the camera. In scanning tunneling microscopy (bottom), dopants embedded in some base material produce individual electronic signatures. For each of these cases, the observed signal can be modeled as a convolution between a *short* kernel and a *sparse* activation map.

to maintain a constant current. It is also possible to interrogate the quantum mechanical structure of the material at different energies by operating the device in an open-loop fashion and varying the voltage difference between the tip and surface. This produces a *three-dimensional* observation $\boldsymbol{y} \in \mathbb{R}^{w \times h \times E}$, which we visualize in Figure 12.2 (left).

STM Data Analysis as Finding Repeated Motifs.
The broad goal in analyzing STM data is to extract information about the quantum electronic structure of the material; this bears on questions about physical phenomena such as superconductivity. In many concrete instances, this problem boils down to extracting repeated motifs from the observation $\boldsymbol{y}$. Physical properties in the material are strongly influenced by way in which electrons interact with "defects" in the crystal lattice, which occur at different locations $(i_1, j_1), \ldots, (i_k, j_k)$ in space. The interaction between electrons and a defect produces a characteristic motif $\boldsymbol{a} \in \mathbb{R}^{w \times h \times E}$, which is a three dimensional function of both spatial location and energy. An example of such a pattern is visualized in Figure 12.2 (middle). Typically, these motifs are spatially localized, i.e., their spatial extent is small relative to the size of the sample. The overall observation $\boldsymbol{y}$ can be modeled as a superposition of translated versions of the motif $\boldsymbol{a}$, one

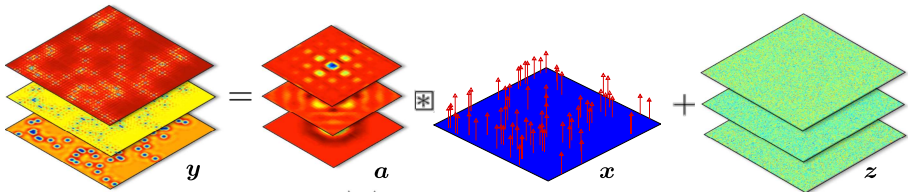


Figure 12.2 Convolutional Data Model for STM: the data $\boldsymbol{y}$ (left) is expressed as a convolution of a basic motif $\boldsymbol{a}$ and a sparse spike train $\boldsymbol{x}$, plus noise $\boldsymbol{z}$. Here, each two-dimensional slice of $\boldsymbol{y}$ is the convolution the corresponding two-dimensional slice of $\boldsymbol{a}$ and the common sparse signal $\boldsymbol{x}$. The data analysis goal is, given $\boldsymbol{y}$, to determine both the motif $\boldsymbol{a}$ and the sparse spike train $\boldsymbol{x}$, neither of which is known ahead of time.

for each of the defect locations (i_ℓ, j_ℓ):

$$\underset{\text{data}}{\boldsymbol{y}(i,j,e)} = \sum_{\ell=1}^{k} \underset{\text{translated motif}}{\boldsymbol{a}(i-i_\ell, j-j_\ell, e)} \quad + \quad \underset{\text{noise}}{\boldsymbol{z}(i,j,e)}. \tag{12.2.1}$$

This expression can be written more concisely, as the convolution of $\boldsymbol{a}(\cdot,\cdot,e)$ and a two-dimensional sparse signal $\boldsymbol{x} \in \mathbb{R}^{w\times h}$, which takes on value 1 at locations (i_ℓ, j_ℓ) and zero elsewhere:

$$\boldsymbol{y}(\cdot,\cdot,e) = \boldsymbol{a}(\cdot,\cdot,e) * \boldsymbol{x} \quad + \quad \boldsymbol{z}(\cdot,\cdot,e). \tag{12.2.2}$$

Combining these equations for all energy levels e, we obtain a model for the dataset as a whole, which we write as

$$\underset{\text{data}}{\boldsymbol{y}} = \underset{\text{motif}}{\boldsymbol{a}} * \underset{\text{sparse spikes}}{\boldsymbol{x}} + \underset{\text{noise}}{\boldsymbol{z}}, \tag{12.2.3}$$

where in this expression, each two-dimensional slice of $\boldsymbol{a}$ is convolved with the two-dimensional spike train $\boldsymbol{x}$ to produce one two-dimensional slice of $\boldsymbol{y}$ [CSL+20]. This model is visualized in Figure 12.2.

Sparse Optimization for Motif Finding.
Our goal is to recover both the motif $\boldsymbol{a}$ and spike train $\boldsymbol{x}$ from the observation $\boldsymbol{y}$. This is an underdetermined problem; to make progress we need to leverage low-dimensional structure in both $\boldsymbol{a}$ and $\boldsymbol{x}$. We will use the fact that

1 $\boldsymbol{a}$ is spatially localized, i.e., it is a *short* signal, whose spatial extent is small compared to that of $\boldsymbol{y}$;
2 $\boldsymbol{x}$ is *sparse*, since it contains only one nonzero entry for each instance of the motif in $\boldsymbol{y}$.

We call this a *short-and-sparse* model, and call the corresponding recovery problem *short-and-sparse deconvolution* (SaSD). This model was studied in the 90's [Hay94b, LB95b, KH96] and has drawn increased interests in recent years due to

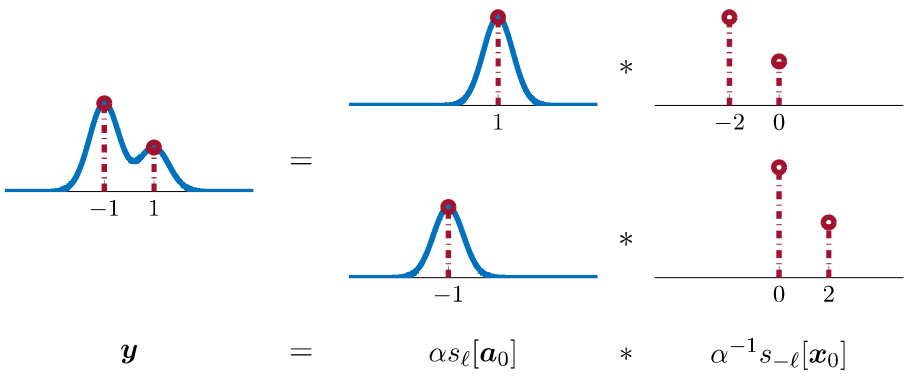


Figure 12.3 Scaling-Shift Symmetry. The SaS convolution model exhibits a scaled shift symmetry: $\alpha s_\ell[\boldsymbol{a}_0]$ and $\alpha^{-1} s_{-\ell}[\boldsymbol{x}_0]$ have the same convolution as $\boldsymbol{a}_0$ and $\boldsymbol{x}_0$. Therefore, the ground truth $(\boldsymbol{a}_0, \boldsymbol{x}_0)$ can only by identified up to some scale and shift ambiguity.

improved computation capability and understanding of its geometry [ZLK+17, KZLW19]. This structure is common to many motif finding problems, in microscopy, neuroscience, astronomy, etc. Using ideas from Chapters 2, 3, and 7, we can formulate an optimization problem that attempts to simultaneously recover both $\boldsymbol{a}$ and $\boldsymbol{x}$:

$$\min_{\boldsymbol{a},\boldsymbol{x}} \varphi_{\mathrm{BL}}(\boldsymbol{a},\boldsymbol{x}) \doteq \underbrace{\tfrac{1}{2}\|\boldsymbol{y} - \boldsymbol{a} * \boldsymbol{x}\|_F^2}_{\text{data fidelity}} + \underbrace{\lambda\|\boldsymbol{x}\|_1}_{\boldsymbol{x}\text{ sparse}} \quad \text{such that} \quad \underbrace{\boldsymbol{a} \in \mathcal{A}}_{\boldsymbol{a}\text{ short}}. \tag{12.2.4}$$

Here, the data fidelity term is the sum of the squared differences between $\boldsymbol{a} * \boldsymbol{x}$ and $\boldsymbol{y}$. The regularizer $\|\boldsymbol{x}\|_1$ encourages $\boldsymbol{x}$ to be sparse. This objective function is sometimes referred to as the *Bilinear Lasso* (hence, the notation φ_{BL}, since it composes the Lasso objective with the bilinear map $(\boldsymbol{a},\boldsymbol{x}) \mapsto \boldsymbol{a} * \boldsymbol{x}$ [CSL+20, ZLK+17, KZLW19]. The constraint $\boldsymbol{a} \in \mathcal{A}$ asks $\boldsymbol{a}$ to be short. One way of doing this is constraining $\boldsymbol{a}$ to be supported on a relatively small region $\{1,\dots,w\} \times \{1,\dots,h\} \times \{1,\dots,E\}$, with $w \ll W$ and $h \ll H$. There is a further bilinear degree of freedom between $\boldsymbol{a}$ and $\boldsymbol{x}$: for any nonzero λ, $(\lambda\boldsymbol{a}) * (\lambda^{-1}\boldsymbol{x}) = \boldsymbol{a} * \boldsymbol{x}$. We eliminate this degree of freedom by constraining $\boldsymbol{a}$ to have unit Frobenius norm, setting

$$\mathcal{A} \doteq \{\boldsymbol{a} \mid \mathrm{supp}(\boldsymbol{a}) \subseteq \{1,\dots,w\} \times \{1,\dots,h\} \times \{1,\dots,E\},\ \|\boldsymbol{a}\|_F = 1\}. \tag{12.2.5}$$

As we will see in the next section, due to the symmetries of the convolution operator $*$, this problem is nonconvex. As with the simpler model problems in Chapter 7, particular choice $\|\boldsymbol{a}\|_F = 1$ plays an important role in shaping the geometry of this nonconvex problem, by interacting with the objective to create regions of negative curvature.

12.3 Symmetry in Short-and-Sparse Deconvolution

The short-and-sparse model admits a basic shift symmetry, which is inherited from the symmetries of the convolution operator $*$: letting s_τ denote a shift by τ pixels, we have

$$s_\tau[\boldsymbol{a}] * s_{-\tau}[\boldsymbol{x}] = \boldsymbol{a} * \boldsymbol{x}. \tag{12.3.1}$$

In the two dimensional setting of STM, τ represents a two-dimensional shift in space. In Figure 12.3, we illustrate this symmetry in a one-dimensional setting. The shift symmetry is a form of discrete symmetry, similar to those studied in Chapter 7. Because of this symmetry, natural formulations of short-and-sparse deconvolution admit multiple equivalent solutions, and are nonconvex. Indeed, viewed as a function of the pair $(\boldsymbol{a}, \boldsymbol{x})$, the objective (12.2.4) is a nonconvex function; the constraint set is also nonconvex.

Similar to Chapter 7, it is possible to study the geometry of deconvolution problems through their symmetries [ZLK+17, ZKW18, KZLW19]. For example, it is possible to derive simpler approximations $\varphi_{\text{ABL}} \approx \varphi_{\text{BL}}$ to the objective in (12.2.4) that can be studied mathematically. If $\boldsymbol{a}$ is *shift incoherent*, i.e., for any shift τ $\langle \boldsymbol{a}, s_\tau[\boldsymbol{a}] \rangle \approx 0$, then the loss in (12.2.4) can be approximated as

$$\begin{aligned} \tfrac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{a} * \boldsymbol{x}\right\|_F^2 &= \tfrac{1}{2}\|\boldsymbol{y}\|_F^2 + \tfrac{1}{2}\|\boldsymbol{a} * \boldsymbol{x}\|_F^2 - \langle \boldsymbol{y}, \boldsymbol{a} * \boldsymbol{x} \rangle \\ &\approx \tfrac{1}{2}\|\boldsymbol{y}\|_F^2 + \tfrac{1}{2}\|\boldsymbol{x}\|_F^2 - \langle \boldsymbol{y}, \boldsymbol{a} * \boldsymbol{x} \rangle . \end{aligned} \tag{12.3.2}$$

This gives:

$$\varphi_{\text{ABL}}(\boldsymbol{a}, \boldsymbol{x}) \doteq \tfrac{1}{2}\|\boldsymbol{y}\|_F^2 + \tfrac{1}{2}\|\boldsymbol{x}\|_F^2 - \langle \boldsymbol{y}, \boldsymbol{a} * \boldsymbol{x} \rangle + \lambda \|\boldsymbol{x}\|_1, \quad \boldsymbol{a} \in \mathcal{A}. \tag{12.3.3}$$

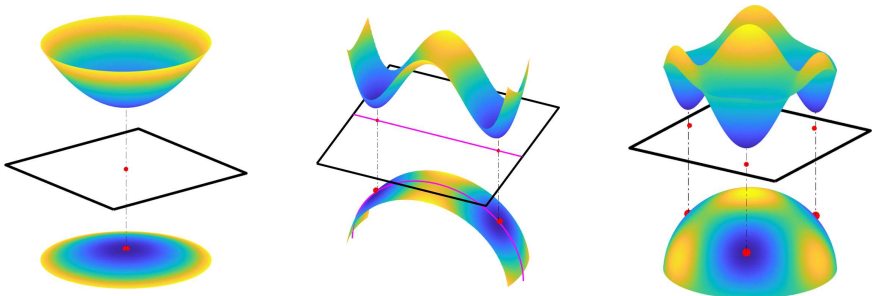
Exercise 12.1 explores this approximation in more detail; the key intuition is that this approximation is accurate when the *shift-coherence*

$$\mu_s = \max_{\tau \neq 0} |\langle \boldsymbol{a}, s_\tau[\boldsymbol{a}] \rangle| \tag{12.3.4}$$

is small. Figure 12.4 visualizes the geometry of this approximation for shift-incoherent problems. As expected, equivalent (symmetric) solutions are local minimizers, and there is negative curvature in symmetry breaking directions.

The theory points to a key difference between real deconvolution problems and the idealized models studied in Figure 12.4 and Chapter 7. As the motif $\boldsymbol{a}$ becomes more shift-coherent, the problem becomes more challenging, both numerically and theoretically. This can be quantified in terms of a sparsity-coherence tradeoff, illustrated in Figure 12.5. The less coherent $\boldsymbol{a}$ is, the denser $\boldsymbol{x}$ can be. This tradeoff is reminiscent of our discussion of coherence in matrix completion and recovery in Chapters 4-5.

This tradeoff points to an important challenge in practical deconvolution problems: deconvolution problems in imaging and the sciences tend to be highly coherent – the motif or blur kernel $\boldsymbol{a}$ is typically spatially smooth. In contrast to



(a) a single shift $s_{\ell_1}[\boldsymbol{a}_0]$ (b) two shifts $s_{\ell_1}[\boldsymbol{a}_0]$, $s_{\ell_2}[\boldsymbol{a}_0]$ (c) multiple shifts

Figure 12.4 Geometry of Approximate Bilinear Lasso Objective $\varphi_{\mathrm{ABL}}(\boldsymbol{a})$ near superpositions of shifts of $\boldsymbol{a}_0$ [KZLW19]. **Top:** function values of $\varphi_{\mathrm{ABL}}(\boldsymbol{a})$ visualized as height. **Bottom:** heat maps of $\varphi_{\mathrm{ABL}}(\boldsymbol{a})$ on the sphere $\mathbb{S}^{n-1}$. **(a)** the region near a single shift is strongly convex; **(b)** the region between two shifts contains a saddle-point, with negative curvature pointing towards each shift and positive curvature pointing away; **(c)** region near the span of several shifts of $\boldsymbol{a}_0$.

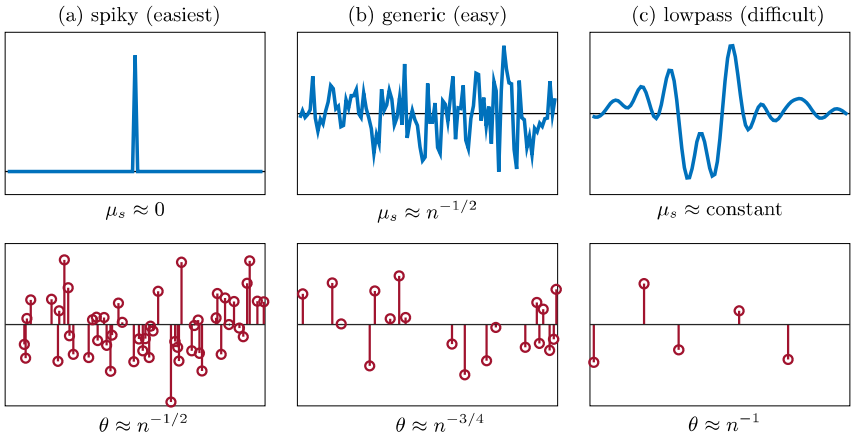


Figure 12.5 Sparsity-Coherence Tradeoff [KZLW19]: examples with varying coherence parameter $\mu_s(\boldsymbol{a}_0)$ and sparsity rate θ (i.e., probability a given entry is nonzero). Smaller shift-coherence $\mu_s(\boldsymbol{a}_0)$ allows SaSD to be solved with higher θ, and vice versa. In order of increasing difficulty: (a) when $\boldsymbol{a}_0$ is a Dirac delta function, $\mu_s(\boldsymbol{a}_0) = 0$; (b) when $\boldsymbol{a}_0$ is sampled from a uniform distribution on the sphere $\mathbb{S}^{n-1}$, its shift-coherence is roughly $\mu_s(\boldsymbol{a}_0) \approx n^{-1/2}$; (c) when $\boldsymbol{a}_0$ is low-pass, $\mu_s(\boldsymbol{a}_0) \to$ const. as n grows.

orthogonal dictionary learning (Chapter 7) and certain neural network learning problems (Chapter 16), we have to contend directly with highly coherent kernels. Approximations such as (12.3.3) break down, and we have to contend directly with the more complicated geometry of (12.2.4). In the next section, we

Algorithm 12.1 Alternating Descent Method (ADM)

Input: observation $\boldsymbol{y}$, stepsizes t_0 and τ_0; penalty $\lambda > 0$.
Initialize $\boldsymbol{a}_0$ at random on the sphere, $\boldsymbol{x}_0 \leftarrow \boldsymbol{0}_n$, and $k \leftarrow 0$.
while not converged **do**
Fix $\boldsymbol{a}_k$ and take a proximal gradient step on $\boldsymbol{x}$ with stepsize t_k

$$\boldsymbol{x}_{k+1} \leftarrow \mathrm{prox}_{t_k \lambda g}\left[\boldsymbol{x}_k - t_k \nabla_{\boldsymbol{x}} \psi\left(\boldsymbol{a}_k, \boldsymbol{x}_k\right)\right].$$

Fix $\boldsymbol{x}_{k+1}$ and take a Riemannian gradient step on $\boldsymbol{a}$ with stepsize τ_k

$$\boldsymbol{a}_{k+1} \leftarrow \mathcal{P}_{\mathcal{A}}\left[\boldsymbol{a}_k - \tau_k \nabla_{\boldsymbol{a}} \psi\left(\boldsymbol{a}_k, \boldsymbol{x}_{k+1}\right)\right].$$

Update $k \leftarrow k + 1$.
end while
Output: Final iterate $\boldsymbol{a}_\star$, $\boldsymbol{x}_\star$.

will describe some algorithmic ideas for coping with high-coherence instances of SaSD.

12.4 Algorithms for Short-and-Sparse Deconvolution

In this section, we describe practical algorithms for (12.2.4), which leverage ideas from Chapters 8-9, to contend with the complicated geometry of practical deconvolution problems. Our problem is a specific instance of the general form

$$\min_{\boldsymbol{a},\boldsymbol{x}} \Psi(\boldsymbol{a},\boldsymbol{x}) \;=\; \psi(\boldsymbol{a},\boldsymbol{x}) \;+\; \lambda \cdot g(\boldsymbol{x}), \qquad \text{s.t.} \quad \boldsymbol{a} \;\in\; \mathcal{M}, \tag{12.4.1}$$

where $\psi(\boldsymbol{a},\boldsymbol{x})$ is twice continuously differentiable, and $g(\boldsymbol{x})$ is a convex (possibly nonsmooth) sparse promoting penalty, and $\mathcal{M}$ is a smooth Riemannian manifold such as a sphere.

12.4.1 Alternating Descent Method

We begin by introducing a vanilla first-order method for solving (12.4.1) based on an alternating descent method (ADM). The method reduces the objective by alternating between taking descent steps on one variable with the other fixed. The basic algorithm pipeline is summarized in Algorithm 12.1. We provide more detailed explanation for each of the steps below.

Fix $\boldsymbol{a}$ and take a proximal gradient step on $\boldsymbol{x}$.
Fix $\boldsymbol{a}$, and consider the marginal function

$$\Psi_{\boldsymbol{a}}(\boldsymbol{x}) = \psi(\boldsymbol{a},\boldsymbol{x}) + \lambda g(\boldsymbol{x}) \tag{12.4.2}$$

as a function of $\boldsymbol{x}$ only. This is a sum of a smooth function ψ which is convex in $\boldsymbol{x}$ and a nonsmooth convex function $g(\boldsymbol{x})$. This form is familiar from our discussion of proximal gradient methods in Chapter 8. We can express the derivative of ψ with respect to $\boldsymbol{x}$ as

$$\nabla_{\boldsymbol{x}}\psi(\boldsymbol{a},\boldsymbol{x}) = \boldsymbol{\iota}_{\boldsymbol{x}}^{*}\,\check{\boldsymbol{a}} * (\boldsymbol{a} * \boldsymbol{x} - \boldsymbol{y}). \tag{12.4.3}$$

Here, $\boldsymbol{\iota}_{\boldsymbol{x}}^{*}$ restricts to the interval $\{1,\ldots,n\}$. $\check{\boldsymbol{a}}$ denotes the reversal of the signal $\boldsymbol{a}$. In Exercise 12.3 we verify this formula for the gradient, by verifying that convolution $\check{\boldsymbol{a}} * \cdot$ with the reversal of $\boldsymbol{a}$ is the formal adjoint of the convolution operator $\boldsymbol{x} \mapsto \boldsymbol{a} * \boldsymbol{x}$. For $\boldsymbol{a}$ fixed, the gradient $\nabla_{\boldsymbol{x}}\psi(\boldsymbol{a},\boldsymbol{x})$ is a Lipschitz function of $\boldsymbol{x}$. The Lipschitz constant L is the norm of the operator $\boldsymbol{x} \mapsto \boldsymbol{\iota}_{\boldsymbol{x}}^{*}\check{\boldsymbol{a}} * \boldsymbol{a} * \boldsymbol{x}$.[1] Following our discussion in Chapter 8, we can reduce the function $\Psi_{\boldsymbol{a}}(\boldsymbol{x})$ by taking a gradient step and then applying the proximal operator associated with the regularizer λg:

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_{t\lambda g}\left[\boldsymbol{x}_k - t\nabla_{\boldsymbol{x}}\psi(\boldsymbol{a}_k,\boldsymbol{x}_k)\right]. \tag{12.4.4}$$

As long as $t \leq 1/L$, this reduces the objective: $\Psi(\boldsymbol{a}_k,\boldsymbol{x}_{k+1}) \leq \Psi(\boldsymbol{a}_k,\boldsymbol{x})$. The Lipschitz constant L can be estimated from the discrete Fourier transform of $\boldsymbol{a}_k$, or an effective step size can be determined by backtracking (i.e., reducing the step size until a sufficient reduction in objective is observed). Our regularizer g is the ℓ^1 norm; the proximal operator associated with this convex function is simply the soft thresholding operation:

$$\operatorname{prox}_{\tau\lambda g}(\boldsymbol{x}) = \mathcal{S}_{t\lambda}(\boldsymbol{x}), \tag{12.4.5}$$

where $\mathcal{S}_{t\lambda}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{x})\,(|\boldsymbol{x}| - \lambda t)_{+}$ shrinks the entries of the vector $\boldsymbol{x}$ towards zero (see Section 8.2 for more discussion and derivation of this operator).

Fix $\boldsymbol{x}$ and take a projected gradient step on $\boldsymbol{a}$.
Proceeding in a similar manner, we can calculate the derivative of $\psi(\boldsymbol{a},\boldsymbol{x})$ with respect to $\boldsymbol{a}$:

$$\nabla_{\boldsymbol{a}}\psi(\boldsymbol{a},\boldsymbol{x}) = \boldsymbol{\iota}_{\boldsymbol{a}}^{*}\check{\boldsymbol{x}} * (\boldsymbol{a} * \boldsymbol{x} - \boldsymbol{y}). \tag{12.4.6}$$

Here, again $\boldsymbol{\iota}_{\boldsymbol{a}}^{*}$ restricts to the allowed support of $\boldsymbol{a}$. Taking a gradient step $\boldsymbol{a} \mapsto \boldsymbol{a} - \tau\nabla_{\boldsymbol{a}}\psi(\boldsymbol{a},\boldsymbol{x})$ for appropriate step size (chosen smaller than $1/L_{\boldsymbol{a}}$, where $L_{\boldsymbol{a}}$ is the norm of the operator $\boldsymbol{a} \mapsto \boldsymbol{\iota}_{\boldsymbol{a}}^{*}\check{\boldsymbol{x}} * \boldsymbol{x} * \boldsymbol{a}$ and the Lipschitz constant of $\nabla_{\boldsymbol{a}}\psi$) reduces the objective function Ψ. However, the resulting $\boldsymbol{a}_{+}$ may not have unit norm, and hence may not reside in the feasible set $\mathcal{A}$. We address this by projecting onto $\mathcal{A}$, simply by scaling $\boldsymbol{a}_{+}$ to have unit ℓ^2 norm:

$$\boldsymbol{a}_{k+1} = \mathcal{P}_{\mathcal{A}}\left[\boldsymbol{a}_k - \tau_k\nabla_{\boldsymbol{a}}\psi\left(\boldsymbol{a}_k,\boldsymbol{x}_{k+1}\right)\right]. \tag{12.4.7}$$

This projected gradient approach to $\boldsymbol{a}$ update is simple and often quite effective in practice. It is also possible to derive a variety of algorithms through the

[1] Since convolution in time is equivalent to multiplication in frequency, this can be controlled in terms of the largest Fourier coefficient of $\boldsymbol{a}$.

perspective of Riemannian optimization – viewing the constraint $\|\boldsymbol{a}\|_F = 1$ as forcing $\boldsymbol{a}$ to reside on a particular smooth manifold.

12.4.2 Additional Heuristics for Highly Coherent Problems

Although the Bilinear Lasso is able to account for interactions between $\boldsymbol{a}$ and $\boldsymbol{x}$ even when $\boldsymbol{a}$ is highly coherent, the smooth term $\|\boldsymbol{a} * \boldsymbol{x} - \boldsymbol{y}\|_2^2$ nonetheless becomes ill-conditioned as $\mu(\boldsymbol{a})$ increases, leading to slow convergence for practical problem instances. Here we will discuss a number of heuristics which will help to obtain faster algorithmic convergence and produce better solutions in such settings.

Momentum Acceleration.
When $\mu_s(\boldsymbol{a})$ is large, the Hessian of ψ_{BL} becomes ill-conditioned as $\boldsymbol{a}$ converges to single shifts. the objective landscape contains "narrow valleys" in which first-order methods tend to exhibit severe oscillations. For a nonconvex problem such as the Bilinear Lasso, iterates of first-order methods could encounter many narrow and flat valleys along the descent trajectory, resulting in slow convergence.

One remedy here is to add *momentum* [Pol64, BT09] to standard first-order iterations, as we have introduced in Appendix D. For example, when updating $\boldsymbol{x}$, we could modify the iterate in (12.4.4) by

$$\boldsymbol{w}_k = \boldsymbol{x}_k + \beta \cdot \underbrace{(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})}_{\text{inertial term}}, \tag{12.4.8}$$

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_{t_k g}\left[\boldsymbol{w}_k - t_k \nabla_{\boldsymbol{x}} \psi\left(\boldsymbol{a}_k, \boldsymbol{w}_k\right)\right]. \tag{12.4.9}$$

Here, the *inertial term* incorporates the momentum from previous iterations, and $\beta \in (0, 1)$ controls the inertia[2]. In a similar fashion, we can modify the iterate for updating $\boldsymbol{a}$ in (12.4.7). This algorithm is sometimes referred to as *inertial alternating descent method* (iADM) [AMS09][3].

The additional inertial term improves convergence by substantially reducing oscillation effects for ill-conditioned problems. The acceleration of momentum methods for convex problems are well-known in practice[4]. Recently, momentum methods have also been proven to improve convergence for nonconvex and nonsmooth problems [PS16, JNJ18].

Homotopy Continuation.
It is also possible to improve optimization by modifying the objective Ψ_{BL} directly through the sparsity penalty λ. Variations of this idea appear in both

[2] Setting $\beta = 0$ here removes momentum and reverts to standard proximal gradient descent.
[3] It modifies iPALM [PS16] to perform updates on $\boldsymbol{a}$ via retraction on the sphere.
[4] In the setting of strongly convex and smooth function $f(\boldsymbol{z})$, the momentum method improves the iteration complexity from $O(\kappa \log(1/\varepsilon))$ to $O(\sqrt{\kappa} \log(1/\varepsilon))$ with κ being the condition number, while leaving the computational complexity approximately unchanged [B+15].

[ZLK+17] and [KZLW19], and can also help to mitigate the effects of large shift-coherence in practical problems.

When solving (12.2.4) in the noise-free case, it is clear that larger choices of λ encourage sparser solutions for $\boldsymbol{x}$. Conversely, smaller choices of λ place local minimizers of the marginal objective $\varphi_{\mathrm{BL}}(\boldsymbol{a}) \doteq \min_{\boldsymbol{x}} \psi_{\mathrm{BL}}(\boldsymbol{a}, \boldsymbol{x})$ closer to signed-shifts of $\boldsymbol{a}_0$ by emphasizing reconstruction quality. When $\mu(\boldsymbol{a})$ is large, however, φ_{BL} becomes ill-conditioned as $\lambda \to 0$ due to the poor spectral conditioning of $\boldsymbol{a}_0$, leading to severe flatness near local minimizers or the creation of spurious local minimizers when noise is present. At the expense of precision, larger values of λ limit $\boldsymbol{x}$ to a small set of support patterns and simplify the landscape of φ_{BL}. It is therefore important both for fast convergence and accurate recovery for λ to be chosen appropriately.

When problem parameters – such as noise level or sparsity – are not known a priori, a *homotopy continuation method* [HYZ08, WNF09, XZ13] can be used to obtain a *range* of solutions for SaSD. Using a random initialization as in ADM, a rough estimate $(\hat{\boldsymbol{a}}_1, \hat{\boldsymbol{x}}_1)$ is first obtained by solving (12.2.4) with iADM using a large choice for λ_1; this estimate is refined by gradually decreasing λ_n to produce the *solution path* $\{(\hat{\boldsymbol{a}}_n, \hat{\boldsymbol{x}}_n; \lambda_n)\}$. By ensuring that $\boldsymbol{x}$ remains sparse along the solution path, homotopy provides the objective Ψ_{BL} with (restricted) strong convexity w.r.t. both $\boldsymbol{a}$ and $\boldsymbol{x}$ throughout optimization [ANW10]; in numerical experiments, this often leads to a linear rate of convergence.

Data Driven Initialization.
The structure of the SaSD problem suggests a means of initializing the motif $\boldsymbol{a}_0$. Our goal is to recover $\boldsymbol{a}$, up to shift symmetry. That is, the goal is to recover a single shift of $\boldsymbol{a}$. The data $\boldsymbol{y}$ is the convolution of $\boldsymbol{a}$ with a sparse signal $\boldsymbol{x}$. This implies that small pieces of $\boldsymbol{y}$ are themselves superpositions of a few shifted copies of $\boldsymbol{a}_0$. This suggests a means of initialization: one selects a small window of the data and then normalizes it to lie on the sphere.

12.4.3 Computational Examples

Figure 12.6 shows an example of motif finding in STM data using the method introduced above, which is featured in the recent work [CSL+20]. The particular dataset consists of measurements across a $100 \times 100nm^2$ area at $E = 41$ different bias voltages. In the left pane, the figure shows the modulus of the two dimensional Fourier transform of two spatial slices. This relatively noisy product is the basis for conventional data analysis techniques in the area. In the right pane is a much cleaner analysis produced by solving a SaSD problem. Panels (f) and (g) show two slices of the motif signature $\boldsymbol{a}_\star$, while pane (e) shows the sparse activation map $\boldsymbol{x}_\star$. The modulus Fourier transforms of (f) and (g) are cleaner and easier to interpret than their counterparts in (c) and (d).

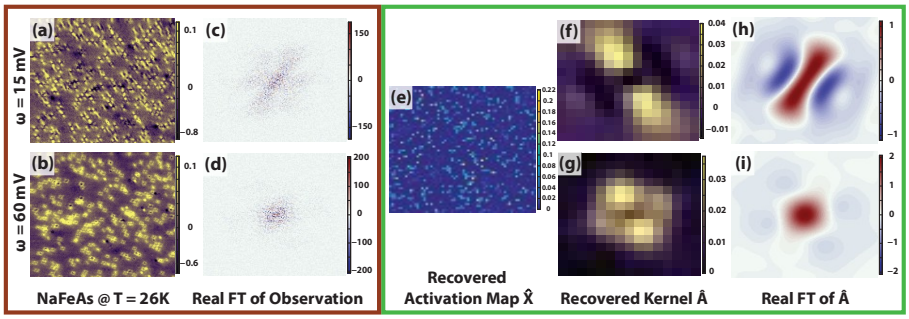


Figure 12.6 Short and Sparse Deconvolution on Real NaFeAs Data. Left: two slices of a dataset at different energy levels (a)-(b). One conventional approach to analyzing such data is to visualize the magnitude of the Fourier transform (c)-(d). In dense samples, this produces a "phase noise" which obscures physically meaningful structures. **Right:** deconvolution via the Bilinear Lasso. Sparse activation map $\boldsymbol{x}$ (e) and motif $\boldsymbol{a}$ (f)-(g). The Fourier transform (h)-(i) of the motif is clearer and reveals more structure than that of the original data (c)-(d).

12.5 Extensions: Multiple Motifs

In many scientific problems, the data consist of superpositions of more than one type of basic motif. For example, in scanning tunneling microscopy, data may contain multiple types of impurities, or multiple states of matter. In neural spike sorting, data may consist of spike patterns from multiple neurons. Many of the algorithmic ideas discussed above extend very naturally to handle data with multiple motifs. One simply introduces variables of optimization $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_K$, with corresponding sparse spike trains $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K$, and solves:

$$\min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_K,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_K} \frac{1}{2}\Big\|\boldsymbol{y} - \sum_{\ell=1}^{K} \boldsymbol{a}_\ell * \boldsymbol{x}_\ell\Big\|_F^2 + \lambda \sum_{\ell=1}^{K} \|\boldsymbol{x}_\ell\|_1 \quad \text{subject to} \quad \boldsymbol{a}_\ell \in \mathcal{A},\ \ell = 1, \ldots, K. \tag{12.5.1}$$

This extension is sometimes referred to as multi-channel sparse blind deconvolution [QLZ19] or *convolutional dictionary learning* [QZL+20a]. This problem is again nonconvex. In addition to the shift symmetry described above, this problem exhibits a permutation symmetry: reordering the motifs $\boldsymbol{a}_\ell$ and their corresponding sparsity maps $\boldsymbol{x}_\ell$ does not change the objective. Nevertheless, many of the algorithmic ideas described above generalize naturally to this higher-dimensional problem. Ideas of momentum acceleration, continuation and reweighting remain essential to obtaining high quality results on practical data. There are many open theoretical issues associated with deconvolution and convolutional dictionary learning. One avenue to theoretical progress is to solve for the motifs $\boldsymbol{a}_\ell$ one at a time. If the shifts of the $\boldsymbol{a}_\ell$ are mutually incoherent, it is possible to analyze the geometry of the resulting problem and prove that nonconvex methods

produce accurate estimates of the ground truth. Interested readers may refer to some of the latest progress on these topics [QLZ19, QZL+20a].

12.6 Exercises

12.1 (Approximate Bilinear Lasso and Incoherent Problems). *Consider a length-k signal $\boldsymbol{a} \in \mathbb{R}^k$ of unit ℓ^2 norm. Consider the partial convolution matrix*

$$\boldsymbol{C}_{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{a} & s_1[\boldsymbol{a}] & s_2[\boldsymbol{a}] & \dots & s_{k-1}[\boldsymbol{a}] \end{bmatrix}. \tag{12.6.1}$$

Argue that

$$\|\boldsymbol{C}_{\boldsymbol{a}}^* \boldsymbol{C}_{\boldsymbol{a}} - \boldsymbol{I}\| \leq k(k-1)\mu_s(\boldsymbol{a}), \tag{12.6.2}$$

where μ_s is the shift coherence. For what $\boldsymbol{a}$ is the approximation $\|\boldsymbol{a} * \boldsymbol{x}\|_2^2 \approx \|\boldsymbol{x}\|_2^2$ accurate?

12.2 (Coherence of a Gaussian Motif). *Consider a Gaussian signal $\boldsymbol{a}$ of length k, with $a_i = \beta \exp(-(i-k)^2/\sigma^2)$, $(i = 1, \dots, k)$, where β is chosen to ensure that $\boldsymbol{a}$ has unit ℓ^2 norm. Argue that (i) as $\sigma \to 0$, $\mu_s(\boldsymbol{a})$ approaches 0, while as $\sigma \to \infty$, $\mu_s \to 1/k$, $\mu_s \to k - 1/\sqrt{k}$. In the latter (large coherence) setting, the approximation φ_{ABL} is inaccurate.*

12.3 (Gradient of Quadratic Loss under Convolution). *Consider the quadratic loss*

$$\psi(\boldsymbol{a}, \boldsymbol{x}) = \tfrac{1}{2} \|\boldsymbol{a} * \boldsymbol{\iota x} - \boldsymbol{y}\|_2^2 . \tag{12.6.3}$$

Show that the gradient of this loss with respect to $\boldsymbol{x}$ is given by

$$\nabla_{\boldsymbol{x}} \psi = \boldsymbol{\iota}^* (\boldsymbol{a} * \boldsymbol{\iota x} - \boldsymbol{y}) . \tag{12.6.4}$$

13 Robust Face Recognition

"*Machines take me by surprise with great frequency.*"
– Alan Turing, *Computing Machinery and Intelligence*

13.1 Introduction

In human perception, the role of sparse representation has been studied extensively. As we have alluded to in the Introduction, Chapter 1, investigators in neuroscience have revealed that in both low-level and mid-level human vision, many neurons in the visual pathway are selective for recognizing a variety of specific stimuli, such as color, texture, orientation, scale, and even view-tuned object images [OF97,Ser06]. Considering these neurons to form an over-complete dictionary of base signal elements at each visual stage, the firing of the neurons with respect to a given input image is typically highly sparse.

As we discussed in the earlier part of the book, the original goal of sparse representation was not inference nor classification *per se*, but rather representation and compression of signals, potentially using lower sampling rates than the Shannon-Nyquist bound. Therefore, the algorithm performance was measured mainly by the sparsity of the representation and the fidelity to the original signals. Furthermore, individual base elements in the dictionary were not assumed to have any particular semantic meaning – they were typically chosen from standard bases (e.g., Fourier, Wavelets, Curvelets, Gabor filters etc.), or learned from data with PCA [PMS94, CJG+15], or a deep convolution neural network (as we will detail in Chapter 16), or even generated from random projections [WYG+09, CJG+15]. Nevertheless, the sparsest representation *is* naturally discriminative: amongst all subsets of base vectors, it would select the subset which most compactly expresses the input signal and rejects all other possible but less compact representations.

In this chapter, we exploit the discriminative nature of sparse representation to perform *classification*.[1] Instead of using the generic dictionaries mentioned above, we represent a test sample using a data-driven dictionary, whose base elements are *the training samples themselves*. If sufficient training samples are

[1] In Chapter 16, we will revisit the discriminative nature of low-dimensional models, including sparsity, in a broader context of deep networks for classification.

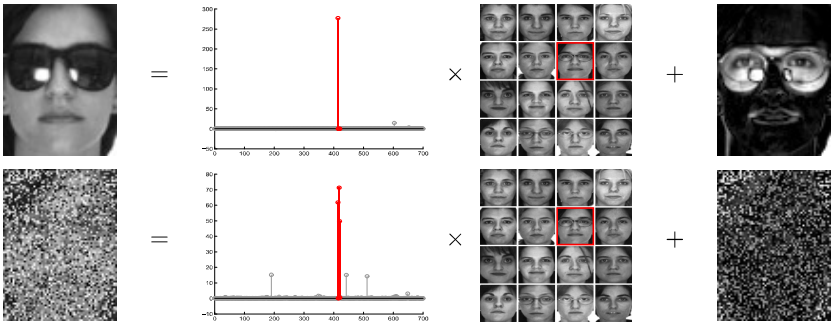


Figure 13.1 An Overview of the Formulation. We represent a test image (left), which is potentially occluded (top) or corrupted (bottom), as a sparse linear combination of all the training images (middle) plus sparse errors (right) due to occlusion or corruption. Red (darker) coefficients correspond to training images of the correct individual. The algorithm determines the true identity (indicated with a red box at second row and third column) from 700 training images of 100 individuals (7 each) in the standard AR face database [MB98].

available from each class, it will be possible to represent the test sample as a linear combination of just those training samples from the same class. This representation is naturally sparse, involving only a small fraction of the overall training database. We will see that in many problems of interest, it is actually the *sparsest* linear representation of the test sample in terms of this dictionary, and can be recovered efficiently via sparse optimization. Seeking the sparsest representation therefore automatically discriminates between the various classes present in the training set. Figure 13.1 illustrates this simple idea using face recognition as an example. Sparse representation also provides a simple yet surprisingly effective means of rejecting invalid test samples not arising from any class in the training database: these samples' sparsest representations tend to involve many dictionary elements, spanning multiple classes.

We will motivate and study this new approach to classification within the context of automatic face recognition. Human faces are arguably the most extensively studied object in image-based recognition. This is partly due to the remarkable face recognition capability of the human visual system [SBOR06], and partly due to numerous important applications for face recognition technology [ZCPR03]. In addition, technical issues associated with face recognition are sufficiently representative of object recognition and even data classification in general. In this chapter, the application of sparse representation and compressed sensing to face recognition yields new insights into compensating gross image error or facial occlusion in the context of face recognition.

It has been known that facial occlusion or disguise poses a significant obstacle to robust real-world face recognition [LB00, Mar02, SSL06]. This difficulty is mainly due to the unpredictable nature of the error incurred by occlusion: it

may affect any part of the image, and may be arbitrarily large in magnitude. Nevertheless, this error typically corrupts only a fraction of the image pixels, and is therefore sparse in the standard pixel space basis. When the error has such a sparse representation, it can be handled uniformly within the classical sparse representation framework (see Figure 13.1 for an example). Yet in experiments, we further discovered that as the dimension of the problem grows higher, sparsity solvers such as ℓ^1-minimization seem to be able to recover dense error with ease. In this context, the general theory of sparse representation and compressive sensing falls short in explaining the phenomena of dense error correction with a special kind of dictionaries, called the *cross-and-bouquet* model. We will discuss the conditions in which ℓ^1-minimization guarantees to recover dense error approaching 100% under the cross-and-bouquet model.

13.2 Classification Based on Sparse Representation

A basic problem in object recognition is to use labeled training samples from k distinct object classes to correctly determine the class to which a new test sample belongs. We arrange the given n_i training samples from the i-th class as columns of a matrix $\boldsymbol{A}_i \doteq [\boldsymbol{v}_{i,1}, \boldsymbol{v}_{i,2}, \ldots, \boldsymbol{v}_{i,n_i}] \in \mathbb{R}^{m \times n_i}$. In the context of face recognition, we will identify a $w \times h$ grayscale image with the vector $\boldsymbol{v} \in \mathbb{R}^m$ ($m = wh$) given by stacking its columns. Then the columns of $\boldsymbol{A}_i$ are the training face images of the i-th subject.

An immense variety of statistical models have been proposed for exploiting the structure of the $\boldsymbol{A}_i$ for recognition. One particularly simple and effective approach models the samples from a single class as lying on a linear subspace. Subspace models are flexible enough to capture much of the variation in real datasets. In particular in the context of face recognition, it has been observed that images of a face under varying lighting and expression lie on a special low-dimensional subspace [BHK97, BJ03], often called a *face subspace*. This is the only prior knowledge about the training samples we will be using in proposing a solution using sparse representation.

Given sufficient training samples of the i-th object class, $\boldsymbol{A}_i \in \mathbb{R}^{m \times n_i}$, any new (test) sample $\boldsymbol{y} \in \mathbb{R}^m$ from the same class approximately lie in the linear span of the training samples associated with object i:

$$\boldsymbol{y} = \alpha_{i,1}\boldsymbol{v}_{i,1} + \alpha_{i,2}\boldsymbol{v}_{i,2} + \cdots + \alpha_{i,n_i}\boldsymbol{v}_{i,n_i}, \tag{13.2.1}$$

for some scalars $\alpha_{i,j} \in \mathbb{R}, j = 1, 2, \ldots, n_i$.

Since the membership i of the test sample is initially unknown, we define a new matrix $\boldsymbol{A}$ for the entire training set as the concatenation of the $n = n_1 + \cdots + n_k$ training samples from all k object classes:

$$\boldsymbol{A} \doteq [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_k] = [\boldsymbol{v}_{1,1}, \boldsymbol{v}_{1,2}, \ldots, \boldsymbol{v}_{k,n_k}]. \tag{13.2.2}$$

Then the linear representation of $\boldsymbol{y}$ can be rewritten in terms of all training

samples as

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o \quad \in \mathbb{R}^m, \tag{13.2.3}$$

where $\boldsymbol{x}_o = [0, \ldots, 0, \alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,n_i}, 0, \ldots, 0]^* \in \mathbb{R}^n$ is a coefficient vector whose entries are zero except those associated with the i-th class.[2]

This motivates us to seek the sparsest solution to $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ via sparse optimization, such as ℓ^1-minimization:

$$\hat{\boldsymbol{x}} = \arg\min \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \tag{13.2.4}$$

Given a new test sample $\boldsymbol{y}$ from one of the classes in the training set, we first compute its sparse representation $\hat{\boldsymbol{x}}$ via (13.2.4). Ideally, the nonzero entries in the estimate $\hat{\boldsymbol{x}}$ will be all associated with the columns of $\boldsymbol{A}$ from a single object class i, and we can easily assign the test sample $\boldsymbol{y}$ to that class. However, noise and modeling error may lead to small nonzero entries associated with multiple object classes (for example, see Figure 13.1 bottom case). Based on the global, sparse representation, one can design many possible classifiers to resolve this. For instance, we can classify $\boldsymbol{y}$ based on how well the coefficients associated with all the training samples of each object reproduce $\boldsymbol{y}$.

More specifically, for each class i, let $\delta_i(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ be the characteristic function which selects the coefficients associated with the i-th class. For $\boldsymbol{x} \in \mathbb{R}^n$, $\delta_i(\boldsymbol{x}) \in \mathbb{R}^n$ is a new vector whose only nonzero entries are the entries in $\boldsymbol{x}$ that are associated with class i. Using only the coefficients associated with the i-th class, one can approximate the given test sample $\boldsymbol{y}$ as $\hat{\boldsymbol{y}}_i = \boldsymbol{A}\delta_i(\hat{\boldsymbol{x}})$. We then classify $\boldsymbol{y}$ based on these approximations by assigning it to the object class that minimizes the residual between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}_i$:

$$\min_i \; r_i(\boldsymbol{y}) \doteq \|\boldsymbol{y} - \hat{\boldsymbol{y}}_i\|_2. \tag{13.2.5}$$

Algorithm 13.1 below summarizes the complete recognition procedure, in which for the ℓ^1-minimization problem (13.2.6) in Step 3, one can use any of the method introduced in Chapter 8 to solve. In particular, the ALM method in Section 8.4 is well suited for this constrained optimization problem.

EXAMPLE 13.1. *(ℓ^1-Minimization vs. ℓ^2-Minimization) To illustrate how Algorithm 13.1 works, we randomly select half of the* $2,414$ *images in the Extended Yale Face Database B [GBK01], as the training set, and the rest for testing. In this example, we subsample the images from the original* 192×168 *to size* 12×10*. The pixel values of the downsampled image are used as 120-D features – stacked as columns of the matrix* $\boldsymbol{A}$ *in the algorithm. Hence matrix* $\boldsymbol{A}$ *has size* 120×1207*, and the system* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ *is underdetermined. Figure 13.2 top illustrates the sparse coefficients recovered by Algorithm 13.1 for a test image from the first subject.*

[2] Notice that in the practice of deep network for classification (as we will see in Chapter 16), people typically use the network to map a given image, here $\boldsymbol{y}$, to a "one-hot" vector $[0, \ldots, 0, 1, 0 \ldots, 0]^* \in \mathbb{R}^k$ that indicates its class out of k classes. So essentially, the deep network plays the same role as any algorithm that solves the sparse solution $\boldsymbol{x}$ here.

Algorithm 13.1 : Sparse Representation-based Classification (SRC)

1: **Input:** a matrix of training samples $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_k] \in \mathbb{R}^{m \times n}$ for k classes, a test sample $\boldsymbol{y} \in \mathbb{R}^m$.
2: Normalize the columns of $\boldsymbol{A}$ to have unit ℓ^2-norm.
3: Solve the ℓ^1-minimization problem (13.2.4).

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \tag{13.2.6}$$

4: Compute the residuals $r_i(\boldsymbol{y}) = \|\boldsymbol{y} - \boldsymbol{A}\,\delta_i(\hat{\boldsymbol{x}})\|_2$ for $i = 1, \ldots, k$.
5: **Output:** identity$(\boldsymbol{y}) = \arg\min_i r_i(\boldsymbol{y})$.

The figure also shows the features and the original images that correspond to the two largest coefficients. The two largest coefficients are both associated with training samples from subject 1. Figure 13.2 bottom shows the residuals with respect to the 38 projected coefficients $\delta_i(\hat{\boldsymbol{x}}_1)$, $i = 1, 2, \ldots, 38$. With 12×10 downsampled images as features, Algorithm 13.1 achieves an overall recognition rate of 92.1% across the Extended Yale B database. Whereas the more conventional minimum ℓ^2-norm solution to the underdetermined system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ is typically quite dense, minimizing the ℓ^1-norm favors sparse solutions, and provably recovers the sparsest solution when this solution is sufficiently sparse. To illustrate this contrast, Figure 13.3 top shows the coefficients of the same test image given by the conventional ℓ^2-minimization, and Figure 13.3 bottom shows the corresponding residuals with respect to the 38 subjects. The coefficients are much less sparse than those given by ℓ^1-minimization (in Figure 13.2), and the dominant coefficients are not associated with subject 1. As a result, the smallest residual in Figure 13.3 does not correspond to the correct subject.

13.3 Robustness to Occlusion or Corruption

In many real-world scenarios, the test image $\boldsymbol{y}$ could be partially occluded or corrupted. In this case, the linear model (13.2.3) should be modified as

$$\boldsymbol{y} = \boldsymbol{y}_o + \boldsymbol{e}_o \;\; = \;\; \boldsymbol{A}\,\boldsymbol{x}_o + \boldsymbol{e}_o, \tag{13.3.1}$$

where $\boldsymbol{e}_o \in \mathbb{R}^m$ is a vector of errors – a fraction, ρ, of its entries are nonzero. The nonzero entries of $\boldsymbol{e}_o$ represent which pixels in $\boldsymbol{y}$ are corrupted or occluded. The locations of corruption can differ for different test images and are not known to the algorithm. The errors may have arbitrary magnitude and therefore cannot be ignored or treated with techniques designed for small noise.

A fundamental principle of coding theory [MS81] is that *redundancy* in the measurement is essential to detecting and correcting gross errors. Redundancy

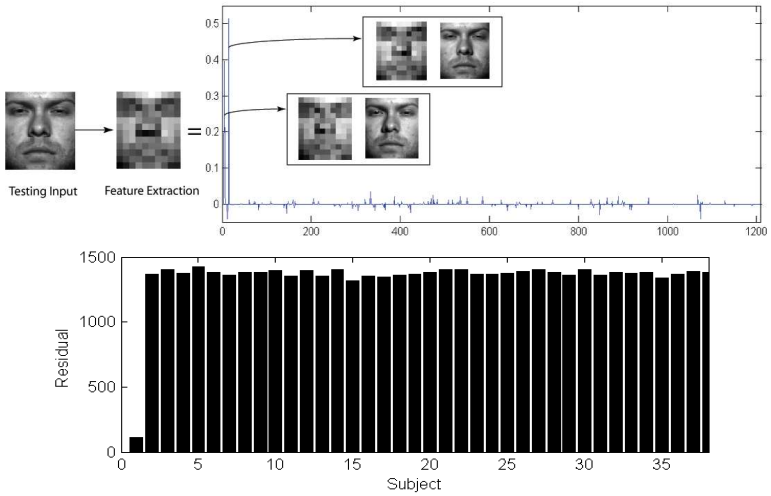


Figure 13.2 A Valid Test Image. Top: Recognition with 12×10 downsampled images as features. The test image $\boldsymbol{y}$ belongs to subject 1. The values of the sparse coefficients recovered from Algorithm 13.1 are plotted on the right together with the two training examples that correspond to the two largest sparse coefficients. Bottom: The residuals $r_i(\boldsymbol{y})$ of a test image of subject 1 with respect to the projected sparse coefficients $\delta_i(\hat{\boldsymbol{x}})$ by ℓ^1-minimization. The ratio between the two smallest residuals is about 1:8.6.

arises in object recognition because the number of image pixels is typically far greater than the number of subjects that have generated the images. In this case, even if a fraction of the pixels are completely corrupted, recognition may still be possible based on the remaining pixels. On the other hand, traditional feature extraction schemes discussed in the previous section would discard useful information that could help compensate for the occlusion. In this sense, no representation is more redundant, robust, or informative than the original images. Thus, when dealing with occlusion and corruption, we should always work with the highest possible resolution, performing downsampling or feature extraction only if the resolution of the original images is too high to process.

Of course, redundancy would be of no use without efficient computational tools for exploiting the information encoded in the redundant data. The difficulty in directly harnessing the redundancy in corrupted raw images has led researchers to instead focus on *spatial locality* as a guiding principle for robust recognition. Local features computed from only a small fraction of the image pixels are clearly less likely to be corrupted by occlusion than holistic features. In face recognition, methods such as ICA [KCYT05] and LNMF [LHZC01] exploit this observation by adaptively choosing filter bases that are locally concentrated. Local Binary Patterns [AHP06] and Gabor wavelets [LVB+93] exhibit similar properties, since they are also computed from local image regions. A related approach partitions the image into fixed regions and computes features for

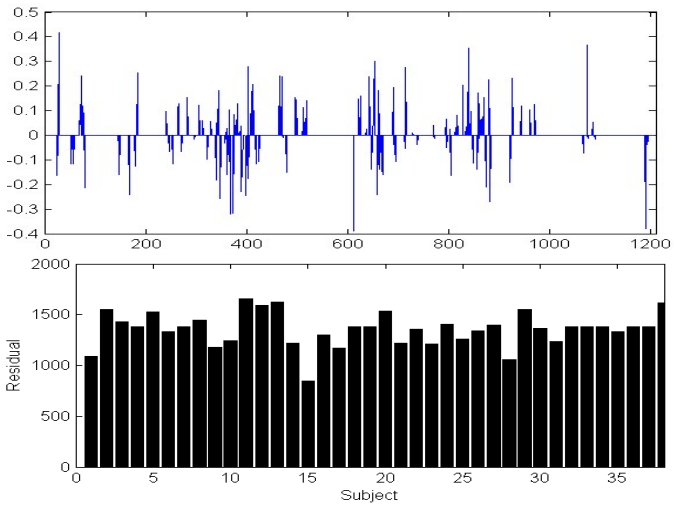


Figure 13.3 Non-sparsity of the ℓ^2-Minimizer. Top: Coefficients from ℓ^2-minimization, using the same test image as Figure 13.2. The recovered solution is not sparse and hence less informative for recognition (large coefficients do not correspond to training images of this test subject). Bottom: The residuals of the test image from subject 1 with respect to the projection $\delta_i(\hat{\boldsymbol{x}})$ of the coefficients obtained by ℓ^2-minimization. The ratio between the two smallest residuals is about 1:1.3. The smallest residual is not associated with subject 1.

each region [PMS94, Mar02]. Notice, though, that projecting onto locally concentrated bases transforms the domain of the occlusion problem, rather than eliminating the occlusion. Errors on the original pixels become errors in the transformed domain, and may even become less local. The role of feature extraction in achieving spatial locality is therefore questionable, since *no bases or features are more spatially localized than the original image pixels themselves.* In fact, the most popular approach to robustifying feature-based methods is based on randomly sampling individual pixels [LB00].

Now, let us show how the sparse representation classification framework can be extended to deal with occlusion. Let us assume that the corrupted pixels are a relatively small portion ρ of the total image pixels. Then the error vector $\boldsymbol{e}_o$, like the vector $\boldsymbol{x}_o$, should be sparse nonzero entries. Since $\boldsymbol{y}_o = \boldsymbol{A}\boldsymbol{x}_o$, we can rewrite (13.3.1) as

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{A}, & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_o \\ \boldsymbol{e}_o \end{bmatrix} \doteq \boldsymbol{B}\,\boldsymbol{w}_o. \tag{13.3.2}$$

Here, $\boldsymbol{B} = [\boldsymbol{A}, \ \boldsymbol{I}] \in \mathbb{R}^{m \times (n+m)}$, so the system $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{w}$ is always underdetermined and does not have a unique solution for $\boldsymbol{w}$. However, in theory, the correct generating $\boldsymbol{w}_o = [\boldsymbol{x}_o, \ \boldsymbol{e}_o]$ has at most $n_i + \rho m$ nonzeros. We might therefore hope to recover $\boldsymbol{w}_o$ as the sparsest solution to the system $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{w}$. As before, we attempt to recover the sparsest solution $\boldsymbol{w}_o$ via sparse optimization, such as solving

Algorithm 13.2 Robust Sparse Representation-based Classification

1: **Input:** a matrix of training samples $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_k] \in \mathbb{R}^{m \times n}$ for k classes, a test sample $\boldsymbol{y} \in \mathbb{R}^m$, (and an optional error tolerance $\varepsilon > 0$.)
2: Normalize the columns of $\boldsymbol{A}$ to have unit ℓ^2-norm.
3: Solve the ℓ^1-minimization problem:

$$\begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{e}} \end{bmatrix} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 + \|\boldsymbol{e}\|_1 \quad \text{subject to} \quad [\boldsymbol{A}, \boldsymbol{I}] \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{e} \end{bmatrix} = \boldsymbol{y}. \tag{13.3.4}$$

4: Compute the residuals $r_i(\boldsymbol{y}) = \|\boldsymbol{y} - \hat{\boldsymbol{e}} - \boldsymbol{A}\,\delta_i(\hat{\boldsymbol{x}})\|_2$ for $i = 1, \ldots, k$.
5: **Output:** $\text{identity}(\boldsymbol{y}) = \arg\min_i r_i(\boldsymbol{y})$.

the following ℓ^1-minimization problem:

$$\hat{\boldsymbol{w}} = \arg\min \|\boldsymbol{w}\|_1 \quad \text{subject to} \quad \boldsymbol{B}\boldsymbol{w} = \boldsymbol{y}. \tag{13.3.3}$$

Algorithm 13.2 summarizes the complete recognition procedure.

More generally, one can assume that the corrupting error $\boldsymbol{e}_o$ has a sparse representation with respect to some basis $\boldsymbol{A}_e \in \mathbb{R}^{m \times n_e}$. That is, $\boldsymbol{e}_o = \boldsymbol{A}_e \boldsymbol{u}_o$ for some sparse vector $\boldsymbol{u}_o \in \mathbb{R}^m$. Here, we have chosen the special case $\boldsymbol{A}_e = \boldsymbol{I} \in \mathbb{R}^{m \times m}$ as $\boldsymbol{e}_o$ is assumed to be sparse in the natural pixel coordinates. If the error $\boldsymbol{e}_o$ is instead more sparse with respect to another basis, e.g., Fourier or Haar, we can simply redefine the matrix $\boldsymbol{B}$ by appending $\boldsymbol{A}_e$ to $\boldsymbol{A}$ and instead seek the sparsest solution $\boldsymbol{w}_o$ to the equation:

$$\boldsymbol{y} = \boldsymbol{B}\boldsymbol{w} \quad \text{with} \quad \boldsymbol{B} = [\boldsymbol{A},\ \boldsymbol{A}_e] \quad \in \mathbb{R}^{m \times (n + n_e)}. \tag{13.3.5}$$

In this way, the same formulation can handle more general classes of sparse corruption.

Experimental Verification of the Algorithm.
We test the robust version of SRC applied to face recognition using the Extended Yale Face Database B [GBK01]. We choose Subsets 1 and 2 (717 images, normal-to-moderate lighting conditions) for training, and Subset 3 (453 images, more extreme lighting conditions) for testing. Without occlusion, this is a relatively easy recognition problem. This choice is deliberate, in order to isolate the effect of occlusion. The images are resized to 96×84 pixels, so in this case $\boldsymbol{B} = [\boldsymbol{A},\ \boldsymbol{I}]$ is an $8,064 \times 8,761$ matrix, a manageable size for most computers.

We then corrupt a percentage of randomly chosen pixels from each of the test images, replacing their values with i.i.d. samples from a uniform distribution. The corrupted pixels are randomly chosen for each test image and the locations are unknown to the algorithm. We vary the percentage of corrupted pixels from 0% to 90%. Figure 13.4 shows several example test images. To the human eye, beyond 50% corruption, the corrupted images (Figure 13.4(a) second and third rows) are barely recognizable as face images; determining their identity seems out

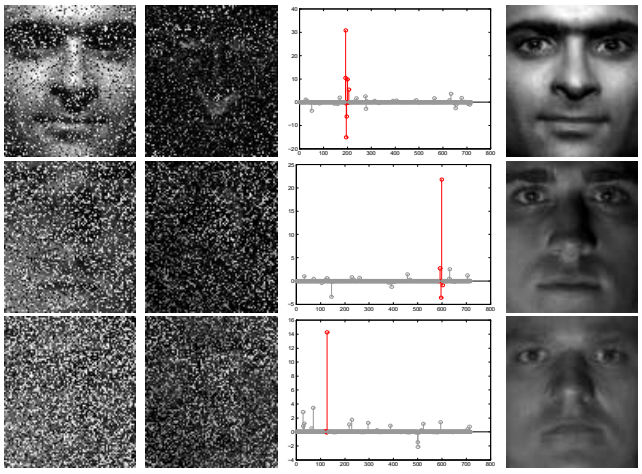


Figure 13.4 Recovered Sparse Representation and Sparse Error under Random Corruption. (a) Test images $\boldsymbol{y}$ from the Extended Yale Face Database B [GBK01], with random corruption. Top row: 30% of pixels are corrupted, Middle row: 50% corrupted, Bottom row: 70% corrupted. (b) Estimated errors $\hat{\boldsymbol{e}}_1$. (c) Estimated sparse coefficients $\hat{\boldsymbol{x}}_1$. (d) Reconstructed images $\boldsymbol{y}_r$. SRC correctly identifies all three corrupted face images.

of the question. Yet even in this extreme circumstance, SRC correctly recovers the identity of the subjects.

We quantitatively compare the sparse method to four popular techniques for face recognition in the vision literature. The Principal Component Analysis (PCA) approach of [TP91] is not robust to occlusion. Although there are many variations to make PCA robust to corruption or incomplete data, some of which have been applied to robust face recognition, e.g., [SSL06], here we use the basic PCA to provide a standard baseline for comparison. The remaining three techniques are designed to be more robust to occlusion. Independent Component Analysis (ICA) architecture I [KCYT05] attempts to express the training set as a linear combination of statistically independent basis images. Local Nonnegative Matrix Factorization (LNMF) [LHZC01] approximates the training set as an additive combination of basis images, computed with a bias toward sparse bases. For PCA, ICA and LNMF, the number of basis components is chosen to give the best performance over the range $\{100, 200, 300, 400, 500, 600\}$. Finally, to demonstrate that the improved robustness is really due to the use of the ℓ^1-norm, we compare to a least-squares technique that first projects the test image onto the subspace spanned by all face images, and then performs nearest subspace.

Figure 13.5 plots the recognition performance of SRC and five competitors, as a function of the level of corruption. We see that the algorithm dramatically outperforms others. From 0% up to 50% occlusion, SRC correctly classifies all

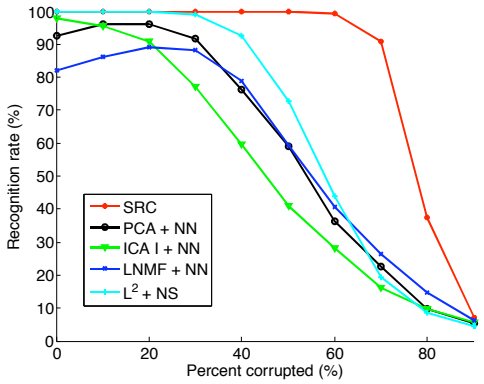


Figure 13.5 Recognition Rates with Random Corruption. The recognition rate across the entire range of corruption for various algorithms. SRC (red curve) significantly outperforms others, performing almost perfectly up to 60% random corruption (see table below).

subjects. At 50% corruption, none of the others achieves higher than 73% recognition rate, while the proposed algorithm achieves 100%. Even at 70% occlusion, the recognition rate is still 90.7%. This greatly surpasses the theoretical bound of worst-case corruption (13.3%) that the algorithm is ensured to tolerate. Clearly, the worst-case analysis is too conservative for random corruption.

13.4 Dense Error Correction with the Cross and Bouquet

In this section, we will take a closer look at the sparsity model (13.3.2). Recall in the classical sparse representation theory, one of the conditions for the successful recovery of a sparse signal is that the dictionary under which the sparsity is represented must be sufficiently incoherent. However, the dictionary $\boldsymbol{B} = [\boldsymbol{A}, \boldsymbol{I}] \in \mathbb{R}^{m\times(n+m)}$ is quite special.

In its first part, the matrix $\boldsymbol{A}$ consists of column vectors that represent the pixel values of all face images. As m grows higher, the convex hull spanned by all the face image vectors becomes an extremely tiny portion of the unit sphere in $\mathbb{R}^m$, which means they are highly correlated.[3] As an example, all the face images of the example in Figure 13.4 lie in $\mathbb{R}^m$ where $m = 8,064$, and calculation shows that all the image vectors are contained within a spherical cap of volume $\leq 1.5 \times 10^{-229}$, as shown in Figure 13.6. These vectors are tightly bundled together as a "bouquet." Hence it is fair to say that identifying a face

[3] Notice that we have encountered a similar issue with coherence with real world problems in the scientific imaging problems in Chapter 12: the sifted versions of the motif can be coherent. In such cases, certain heuristics can be used to improve the performance, as discussed in Section 12.4.2.

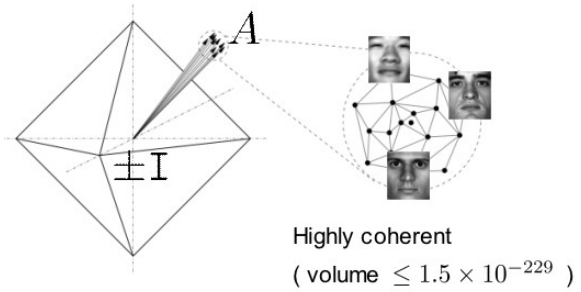


Figure 13.6 The Cross-and-Bouquet Model for Robust Face Recognition. The raw images of human faces expressed as columns of $\boldsymbol{A}$ are clustered with a very tiny variance.

image within a big pool of candidates is like *finding a needle in a haystack which, in turn, lies on a tip of a needle.*

In the second part of $\boldsymbol{B}$, $\boldsymbol{I}$ is a standard m-by-m identity matrix, which is also called a standard pixel basis. Then $\boldsymbol{I}$ and its negative copy $-\boldsymbol{I}$ form a "cross" in $\mathbb{R}^m$, also illustrated in Figure 13.6. We call this type of dictionaries a *cross and bouquet* (CAB) model. It is important to understand how such special (coherent and incoherent) structures may affect the performance of ℓ^1 minimization in finding the correct (sparse) solution.

The CAB model belongs to a special class of sparse representation problems where the dictionary is a concatenation of two or more sub-dictionaries. Examples include the merger of wavelet and heavy-side dictionaries [CDS01] and the combination of texture and cartoon dictionaries in morphological component analysis [ESQD05]. However, in contrast to most other examples, not only is the CAB dictionary as a whole inhomogeneous as we discussed above, in fact the ground-truth signal $(\boldsymbol{x}_o, \boldsymbol{e}_o)$ is also very inhomogeneous, namely, the sparsity of $\boldsymbol{x}_o$ is limited by the number of training images per subject for the purpose of recognition, while we would like to handle as dense corruption error $\boldsymbol{e}_o$ as possible, to guarantee good error correction performance. The above experiment is indeed a concrete demonstration that shows sparse optimization such as ℓ^1-minimization seems to be able to recover very dense error $\boldsymbol{e}_o$. This further contradicts our understanding in the classical sparse representation theory, where the corruption error to be recovered is typically assumed to be sparse.

The reason sparse optimization could recover even dense error is mainly due to the special nature of the sparsity of the signal $\boldsymbol{x}_o$, which is called *weak proportional growth*. Again, assume the signal

$$\boldsymbol{w}_o = \boldsymbol{A}\boldsymbol{x}_o + \boldsymbol{e}_o,$$

where $\boldsymbol{e}_o \in \mathbb{R}^m$ is a vector of error of arbitrary magnitude. We also assume the

columns of $\boldsymbol{A}$ are i.i.d. samples from a Gaussian distribution: $\boldsymbol{A} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] \in \mathbb{R}^{m \times n}$, where $\boldsymbol{v}_i \sim_{iid} \mathcal{N}(\boldsymbol{\mu}, \frac{\nu^2}{m}\boldsymbol{I}_m)$, and $\|\boldsymbol{\mu}\|_2 = 1$, $\|\boldsymbol{\mu}\|_\infty \leq C_\mu m^{-1/2}$.

Definition 13.2 (Weak Proportional Growth). *A sequence of signel-error problems $(\boldsymbol{x}_o, \boldsymbol{e}_o)$ exhibits weak proportional growth with parameters $\delta > 0$, $\rho \in (0, 1)$, $C_0 > 0$, and $\eta_0 > 0$, denoted as $WPG_{\delta,\rho,C_0,\eta_0}$, if as $m \to \infty$,*

$$\frac{n}{m} \to \delta, \quad \frac{\|\boldsymbol{e}_o\|_0}{m} \to \rho, \quad \|\boldsymbol{x}_o\|_0 \leq C_0 m^{1-\eta_0}. \tag{13.4.1}$$

In other words, in the weak proportional growth scenario, $\|\boldsymbol{e}_o\|_0$ grows linearly with respect to m, but $\|\boldsymbol{x}_o\|_0$ is sublinear.

Theorem 13.3 (Dense Error Correction with the Cross and Bouquet). *For any $\delta > 0$, there exists $\nu_0(\delta) > 0$ such that if $\nu < \nu_0$ and $\rho < 1$, in $WPG_{\delta,\rho,C_0,\eta_0}$ with $\boldsymbol{A}$ distributed according to (13.4.1), if the error support and the signs of nonzero elements are chosen uniformly at random then as $m \to \infty$, the probability of successfully recovering $(\boldsymbol{x}_o, \boldsymbol{e}_o)$ via Algorithm 13.2 approaches to one.*

That is, as long as the bouquet is sufficiently tight, under the assumption of weak proportional growth, asymptotically ℓ^1-minimization recovers any nonnegative sparse signal from almost any error with support size less than 100%! A detailed proof of this theorem can be found in [WM10]. Although in general sparse representation problems may not satisfy the weak proportional growth assumption, the assumption is valid in the face recognition example, whereby the number of training samples per subject n_i usually does not grow proportionally with the dimension of the image.

13.5 Notes

Results in this chapter are based on the work [WYG+09], which has provided the earliest evidences that sparse representation can be extremely discriminative and robust for object recognition purposes. As we will reveal in Chapter 16, similar properties of sparse representation have been implicitly exploited by modern deep neural networks for general classification tasks.

Nonuniform Incoherence.
The robust face recognition example also provides a good lesson on the practice of principles introduced in this book. Depending on the applications, "incoherence" is not an absolute notion: Relative to corruptions and errors, the face images are rather coherent among themselves. That is actually the reason why error correction for face images (together) can be so effective. Nevertheless, the seemingly coherent face images have just enough incoherence to allow correct identification of the input image. Like the spectrum identification problem studied in Chapter 11, here we are only interested in the support of the recovered sparse signals; furthermore, unlike applications where one needs to recover a signal with as many

nonzero entries as possible, here the correct solution is preferably the sparser the better. These special conditions significantly relax the incoherence requirement on the sensing matrix (here face images or features). Motivated by these empirical observations, a more rigorous analysis of this type of incoherence was given in the follow-up work [WM10].

Contiguous Occlusion.
In this chapter, for corruption or occlusion of the pixels, we only consider their sparse structure as individual pixels. In practice, however, if the corruption is due to real occlusions, the occluded pixels are not entirely random in their locations. Occluded pixels are often spatially contiguous in their locations. Such structures can probably be better captured by the notion of group sparsity mentioned in Section 6.1.1 of Chapter 6. Empirically, it has been verified that if one can explicitly model and harness such structure in occluded pixels, say as a Markov random field, one can achieve even higher level of robustness to contiguous occlusion for face recognition [ZWMM09]. But to our best knowledge, a rigorous analysis and justification remains elusive at this point.

Importance of Alignment.
In this chapter, we have assumed the test image and images in the gallery are all well aligned. This is however not necessarily the case with real-world face images. As one may see from testing the algorithm, although the scheme is extremely robust to corruption in the pixels, it is rather sensitive to any (small) misalignment in the input face images. This is the case even for the highly engineered and trained modern deep neural networks [AW18,ETT+17], as we will discuss more in Chapter 16. In this situation, we need to consider an extended model to (13.3.1):

$$\boldsymbol{y} \circ \tau = \boldsymbol{A}\boldsymbol{x}_o + \boldsymbol{e}_o$$

for some unknown deformation τ of the image domain (say translation). Nevertheless, in such cases, one may still exploit sparsity for finding the correct alignment and identification simultaneously, as shown in the work [WWG+09, WWG+12]. In the case even gallery images (as columns $\{\boldsymbol{v}_i\}$ of the matrix $\boldsymbol{A}$) are not so well aligned themselves, one may have to align them first before applying the scheme here. To align multiple gallery face images together, one may exploit the fact that the images become highly correlated (hence form a low-rank matrix) when they are correctly aligned. That is, the following matrix

$$\boldsymbol{M}(\tau) = [\boldsymbol{v}_1 \circ \tau_1, \boldsymbol{v}_2 \circ \tau_2, \ldots \boldsymbol{v}_n \circ \tau_n]$$

would have the lowest rank when the correct transformations $\{\tau_i\}$ are found for each of the gallery image $\boldsymbol{v}_i$. Hence efficient techniques on robust low-rank matrix recovery introduced in this book can be used to automatically align multiple face images, as demonstrated in the work [PGW+12]. We will see how similar robust low-rank techniques can be utilized to correct other common deformations in images in Chapter 15.

13.6 Exercises

13.1 (Robust Face Recognition*). *Download the Extended Yale B database. Using the cropped face image set in the database to form a gallery set and a query set. Code a robust face recognition system and demonstrate its performance in the same setting discussed in the experiment of this chapter.*

13.2 (Randomfaces*). *In the literature, there are facial feature extraction methods that reduce the dimensionality of face images according to some linear transformations. In this exercise, we will implement two well-established methods, and compare their performance in terms of recognition accuracy with the results using random projection in compressive sensing.*

1 *Code a function that extracts Eigenface features. Demonstrate the recognition accuracy of robust face recognition in Exercise 13.1 in the Eigenface space with respect to different feature dimensions.*
2 *Code a function that extracts Fisherface features. Demonstrate its recognition accuracy with respect to different feature dimensions.*
3 *Code a function that extracts lower-dimensional features using random projection. This is called* Randomface *features. Demonstrate its recognition accuracy with respect to different feature dimensions, and compare with those of Eigenface and Fisherface features.*

13.3 (Receiver Operating Characteristic (ROC)*). *In the presence of potential irrelevant test samples, it is important to evaluate the performance of a classifier not only based on the* true positive rate, *but often more importably on* false positive rate. *The curve that measures the true positive rates under various false positive rates is known as the* receiver operating characteristic *(ROC) curve.*[4]

In this exercise, code a program that plots a representative ROC curve of the robust face recognition algorithm. Exclude half of the subject classes from the gallery set of the Extended Yale B database, and designate them as outlying subjects. Implement the outlier rejection rule based on the sparse coefficient concentration, and plot the ROC curve with respect to different threshold values of the concentration index.

[4] There are different definitions of the ROC curve. There are four basic performance rates: true positive, false positive, true negative, and false negative.

14 Robust Photometric Stereo

"All the variety, all the charm, all the beauty of life is made up of light and shadow."
– Leo Tolstoy, *Anna Karenina*

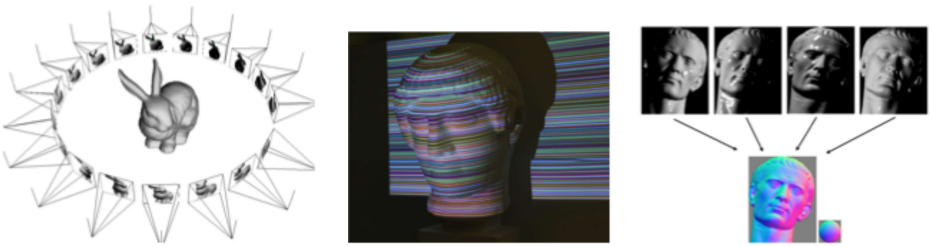
14.1 Introduction

One of the most fundamental problems in computer vision is to capture the 3D shape of an object or a scene. Most popular 3D shape capturing techniques fall into one of the two categories:

1 The so-called *structure from motion* approach reconstructs the 3D geometry by taking multiple images of an object or a scene from different viewpoints [HZ00, MSKS04]. See Figure 14.1(a) for illustration. The images are usually taken under the same or a similar lighting condition since such methods rely on establishing correspondence of common feature points across all the images.
2 The *active light* approach captures the 3D shape by taking multiple images of the object or a scene under different illumination conditions or patterns, but usually at a fixed viewpoint. Methods such as structured lights, photometric stereo, and shape from shading all belong to this category. See Figure 14.1(b) and (c) for illustration.

One can tell from the setup that these two approaches are rather complementary to each other: one varies the camera viewpoints while fixing the lighting whereas the other varies the lighting conditions with a fixed view. Their results are also complementary to each other: structure from motion techniques typically recover 3D positions of a sparse set of points in the scene that have distinguishable local textures for easy correspondence across views; whereas active lighting techniques usually recover a dense per-pixel geometry (depth or surface normal) of the scene even for non-textured regions.

Both approaches have been developed in computer vision and related fields with a long and rich history, and there has been a vast body of literature associated with each method within both categories. In hindsight, though, it would be illuminating to understand now, from the perspective of high-dimensional data analysis, how 3D geometric information of the scene is encoded in the vast



(a) structure from motion (b) structured lights (c) photometric stereo

Figure 14.1 Representative Techniques for Capturing 3D Shapes: (a) structure-from-motion takes multiple images of an object from different viewpoints to triangulate its shape; (b) structured light methods cast different light patterns onto an object surface to reveal its 3D geometry; (c) photometric stereo illuminates the object with multiple directional lights to recover its surface normal.

data measured in both approaches,[1] and why one can efficiently and accurately recover such information from the data.

According to the settings of both approaches, all the images are capturing a common object or a scene. Then, under some reasonable assumptions such as the scene being mostly static and most surfaces having well-conditioned photometric properties, these imagery data should be highly correlated. It has been well-studied and understood that in the structure-from-motion setting, no matter how many corresponding feature points are captured in arbitrarily many views, they form a large measurement matrix, the so-called *multiple-view matrix*, whose rank will always be bounded below *one or two*. Such a low-rank matrix precisely encodes all the camera poses and the depths of all the feature points. Essentially all structure from motion algorithms harness the same low-rank properties to recover the camera poses and depth of feature points. We refer interested readers to [MSKS04] for a full account.

The situation is similar in the active light approach. In this chapter, we will use photometric stereo as an example to show that how low-dimensional structures naturally arise from the physical model of the data generation process and how to harness such low-dimensional structures (using tools from this book) to deal with imperfections in the measurement process so as to accurately recover the object's 3D geometry.

14.2 Photometric Stereo via Low-Rank Matrix Recovery

Photometric stereo [Woo80,Sil80] has been a very popular method for 3D shape capture. It estimates surface orientations of the scene from images taken from a fixed viewpoint under multiple directional lights. As we will soon see, photometric

[1] Typically hundreds or thousands of feature points in structure-from-motion and millions of pixels for the active light methods.

stereo can produce a dense field of surface normals at the level of detail that cannot be achieved by any other feature-based approaches such as structure-from-motion.

14.2.1 Lambertian Surface under Directional Lights

In the setting for photometric stereo, the relative position of the camera and object is usually fixed. The intrinsic parameters of the camera is usually pre-calibrated and known. We do not need to know the camera pose (the extrinsic parameters) as all geometric quantities can be expressed with respect to the camera frame.

For simplicity, we assume a static object is illuminated by a single point light source at infinity.[2] The direction of the light source can be represented as a vector $\boldsymbol{l} \in \mathbb{R}^3$ (with respect to the camera frame). If we take multiple, say n, images under n different lighting directions, we denote the directions as vectors $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_n \in \mathbb{R}^3$. The magnitude of the vector $\boldsymbol{l}$ is assigned to be proportional to the power of the light source.

Next, we need to know that under the illumination, how much light is reflected from the surface and then measured by the sensor of the camera. Notice that this could be a very complicated process. For every point on the surface, we need to describe by how much the incoming light energy, known as irradiance in radiometry, in any direction is absorbed and emitted in any other outgoing direction, known as radiance. This relationship fully characterizes the photometric properties of the surface and is formally known as the *bidirectional reflectance distribution function* (BRDF). In general, the BRDFs for different material surfaces can be very different. For example, metal, plastic, and cloth look very different under the same light.

Nevertheless, for the majority of the objects and scenes we encounter in the real world, their surface photometric property can be approximately modeled by a simple reflectance function known as the *Lambertian model*. For an ideal Lambertian surface, when illuminated by a light source, the surface diffuses and reflects the light equally in all directions. The fraction of light reflected only depends on the angle between the incoming light direction and the surface normal. More precisely, for a point p on a Lambertian surface illuminated under a light in direction $\boldsymbol{l}$, if the surface normal vector at p is $\boldsymbol{n} \in \mathbb{R}^3$, then the amount of light radiated from point p in all direction is given by (the radiance R):

$$R \doteq \rho \langle \boldsymbol{n}, \boldsymbol{l} \rangle = \rho \boldsymbol{n}^* \boldsymbol{l} = \rho \cos(\theta) \|\boldsymbol{l}\|_2, \tag{14.2.1}$$

where ρ is the diffuse albedo that models the percentage of light gets reflected by the surface at point p, $\langle \cdot, \cdot \rangle$ is the inner product, and θ is the angle between the light direction $\boldsymbol{l}$ and surface $\boldsymbol{n}$. See Figure 14.2 for a basic idea. It is easy to see from the model that the brightness of the point p does not depend on the view direction $\boldsymbol{v}$.

[2] In practice, we only need the light source to be relatively far from the object.

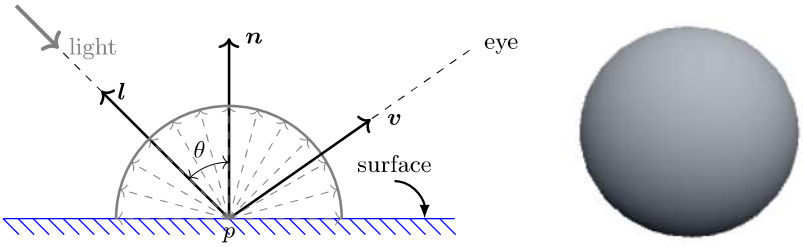


Figure 14.2 Illustration of an Ideal Lambertian Surface Reflectance Model: incoming light is diffused equally to all directions and the amount of diffused light is proportional to the angle θ between the light direction $\boldsymbol{l}$ and the surface normal $\boldsymbol{n}$. Right: an image of a Lambertian (diffusive) sphere.

The albedo ρ of a purely black surface would be zero, hence photometric stereo does not apply to black surfaces or surfaces with very small albedo. Note that the above expression is only valid when $\boldsymbol{n}$ and $\boldsymbol{l}$ form an acute angle ($\theta < 90°$) since radiance R is nonnegative. That is, the surface needs to face the light source. In the case when the surface is facing away from the light source (i.e. $\theta > 90°$), it receives no irradiance hence $R = 0$. We say such area is in the shadow. See the bottom of the image of the sphere in Figure 14.2.

We further assume that there is no inter-reflection,[3] which is often the case if the object is convex or approximately convex. So the corresponding pixel on imaging sensor receives only radiance R contributed from a single point p. If the imaging sensor responses linearly to the radiance, the value of the pixel (x, y) (at the image of the point p) would simply be

$$I(x, y) = R = \rho \boldsymbol{n}^* \boldsymbol{l}. \tag{14.2.2}$$

Let the region of interest be composed of a total of m pixels in each image.[4] We order the pixels with a single index $i \in \{1, \ldots, m\}$, and let $I_j(i)$ denote the observed intensity at pixel i in image I_j. With this notation, we have the following relation about the observation $I_j(i)$:

$$I_j(i) = \rho_i \, \boldsymbol{n}_i^* \boldsymbol{l}_j, \tag{14.2.3}$$

where ρ_i is the albedo of the scene at pixel i, $\boldsymbol{n}_i \in \mathbb{R}^3$ is the (unit) surface normal of the scene at pixel i, and $\boldsymbol{l}_j \in \mathbb{R}^3$ represents the light direction vector corresponding to image I_j.[5]

Consider the matrix $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ constructed by stacking all the vectorized

[3] Inter-reflection is a phenomenon where lights bound off surfaces multiple times before reaching the sensor.

[4] Typically, m is much larger than the number of images n.

[5] The convention here is that the lighting direction vectors point from the surface of the object to the light source.

images $\mathrm{vec}(I)$ as

$$\boldsymbol{D} \doteq [\mathrm{vec}(I_1) \mid \cdots \mid \mathrm{vec}(I_n)]\,, \tag{14.2.4}$$

where $\mathrm{vec}(I_j) = [I_j(1), \ldots, I_j(m)]^*$ for $j = 1, \ldots, n$. It follows from (14.2.3) that $\boldsymbol{D}$ can be factorized as follows:

$$\boldsymbol{D} = \boldsymbol{N} \cdot \boldsymbol{L}, \tag{14.2.5}$$

where $\boldsymbol{N} \doteq [\rho_1 \boldsymbol{n}_1 \mid \cdots \mid \rho_m \boldsymbol{n}_m]^* \in \mathbb{R}^{m\times 3}$, and $\boldsymbol{L} \doteq [\boldsymbol{l}_1 \mid \cdots \mid \boldsymbol{l}_n] \in \mathbb{R}^{3\times n}$. Suppose that the number of images $n \geq 3$. Here $\boldsymbol{N} \cdot \boldsymbol{L}$ is a regular matrix multiplication between $\boldsymbol{N}$ and $\boldsymbol{L}$ and we use a "$\cdot$" to emphasize its (i, j)-th entry is the inner product between the surface normal $\boldsymbol{n}_i$ and the light direction $\boldsymbol{l}_j$. Then, irrespective of the number of pixels m and the number of images n, the rank of the matrix $\boldsymbol{D}$ is at most

$$\mathrm{rank}\,(\boldsymbol{D}) \leq 3. \tag{14.2.6}$$

14.2.2 Modeling Shadows and Specularities

The low-rank structure of the observation matrix $\boldsymbol{D}$ (14.2.5) is seldom observed with real images. This is due to the presence of shadows and specularities in real images.

Shadows

Shadows arise in real images in two possible ways. As we have discussed before in the Lambertian model, some areas on the object will be entirely dark in the image because they face away from the light source. Such dark pixels in the image are referred to as *attached shadows* [KMK97]. See the image of a sphere in Figure 14.2 as an example, where the bottom of the sphere is dark as that part of surface is facing away from the light source. In deriving the low-rank model (14.2.5) from (14.2.3), we have implicitly assumed that all pixels of the object are illuminated by the light source in every image. However, that is impossible to achieve in reality: for a generic object (other than a flat surface), almost in every image, there will always be some pixels facing away from the light source and in the shadows. Mathematically, this implies that (14.2.3) should be modified as follows:

$$I_j(i) = \max\{\rho_i\, \boldsymbol{n}_i^* \boldsymbol{l}_j, 0\}\,. \tag{14.2.7}$$

Shadows can also occur in images when the shape of the object's surface is not entirely convex: parts of the surface can be occluded from the light source by other parts. Even though the normal vectors at such occluded pixels may form an acute angle with the lighting direction, these pixels appear entirely dark. We refer to such dark pixels as *cast shadows*. See the image of Caesar in Figure 14.5 as an example, where, unlike the sphere, the face is not exactly convex and the sporadic shadows around the left side of Caesar's face are cast shadows due to occlusions.

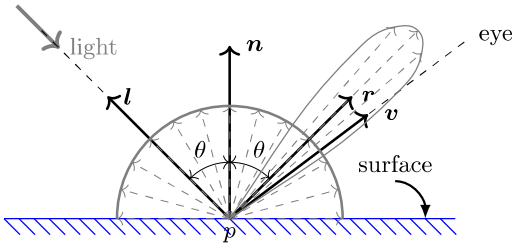


Figure 14.3 A Phong Reflectance Model: incoming light is diffused equally to all directions and an extra amount of light is reflected close to the direction of reflection $\boldsymbol{r}$, known as a specular lobe.

We may pre-detect all the dark shadowed pixels in each images by testing if

$$I_j(i) \approx 0.$$

These pixels are associated with a set entries $\{(k, j)\}$ in the data matrix $\boldsymbol{D}$ where $\boldsymbol{D}(i, j) \approx 0$. We denote the support of these shadowed entries as Ω^c and all the other valid entries as its complement as Ω. With this notation, the valid measurements of the (low-rank) data matrix $\boldsymbol{D}$ are given by

$$\mathcal{P}_\Omega[\boldsymbol{D}] = \mathcal{P}_\Omega[\boldsymbol{N} \cdot \boldsymbol{L}]. \tag{14.2.8}$$

For the remaining pixels not in the shadows, we have assumed that each pixel measures the radiance directly from each point on the surface. For a non-convex object like a human face, that is not entirely the case. Lights can bound back and forth between different part of the surfaces and create the so-called *inter-reflection*. The radiance that some pixels receive might be compounded by such inter-reflection. Nevertheless, studies have shown that if the object is approximately convex, pixels that are affected by inter-reflection will be relatively few [ZMKW13]. We may model such effect as a sparse error $\boldsymbol{E}_1$ in the data matrix:

$$\mathcal{P}_\Omega[\boldsymbol{D}] = \mathcal{P}_\Omega[\boldsymbol{N} \cdot \boldsymbol{L} + \boldsymbol{E}_1]. \tag{14.2.9}$$

Specularities

Specular reflection arises when the object of interest is not perfectly diffusive, i.e., when the surface luminance is not purely isotropic. Mirror is an extreme case which reflects the light with the same angle as the in coming light on the opposite side of the surface normal:

$$\boldsymbol{r} = 2(\boldsymbol{n}^* \boldsymbol{l})\boldsymbol{n} - \boldsymbol{l}.$$

Many real surfaces have both diffusive and reflective characteristics and their reflectance model is a combination of a Lambertian component and a reflective component. The so-called *Phong model* [Pho75] is a correction to the pure

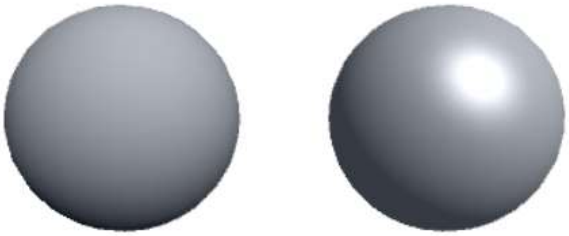


Figure 14.4 Comparison of a Lambertian (diffusive) sphere and a Phong (specular) sphere under the same (directional) lighting condition.

Lambertian model with such a reflective component:

$$R = \rho \boldsymbol{n}^* \boldsymbol{l} + k(\boldsymbol{r}^* \boldsymbol{v})^\alpha, \tag{14.2.10}$$

where $\boldsymbol{v}$ is the viewing direction (to the sensor), $k \geq 0$ is a weight parameter, and $\alpha > 0$ is an exponent parameter. Figure 14.3 illustrates such a reflectance model. In the computer vision and graphics literature, people have also used other functions to model the reflective component, such as the Cook-Torrance reflectance model [CT81].

In general, for a surface of the Phong model (or of the Cook-Torrance model), the intensity of radiance depends on the viewing direction: part of the light is reflected in a mirror-like fashion that generates a specular lobe when the viewing direction $\boldsymbol{v}$ is close to the reflecting direction $\boldsymbol{r}$. This gives rise to some bright spots or shiny patches on the surface of the object, known as *specularities*. Figure 14.3 illustrates this concept and Figure 14.4 compares the Phong model to the Lambertian model with the images of a sphere.

For most real surfaces, the reflective components are usually benign in the sense that the value of the reflective term is significant only when the view direction is very close to the reflecting direction.[6] The specular lobe is usually very small, and from any given viewing angle, only a small fraction of the surface has the specular effect. See Figure 14.5 for some examples of object surfaces with specular effect.

As the surface normals and the viewing angles are not known a priori, we cannot determine which part of the surface is specular. Nevertheless, knowing that specularities are few and sporadic, we may model them as an additional sparse error $\boldsymbol{E}_2$ to the measured data matrix $\boldsymbol{D}$:

$$\boldsymbol{D} = \boldsymbol{N} \cdot \boldsymbol{L} + \boldsymbol{E}_2, \tag{14.2.11}$$

Now, if we combine the sparse errors $\boldsymbol{E}_1$ due to inter-reflections and $\boldsymbol{E}_2$ due

[6] In the Phong model, that corresponds to choosing a large exponent α.

(a) Sphere (b) Caesar (c) Elephant (d) Caesar (with texture)

Figure 14.5 Synthetic image samples used for experiments.

to specularities and let $\boldsymbol{E} = \boldsymbol{E}_1 + \boldsymbol{E}_2$, then, instead of the ideal low-rank model: (14.2.5), a more realistic model for the image measurements should be:

$$\mathcal{P}_\Omega[\boldsymbol{D}] = \mathcal{P}_\Omega[\boldsymbol{N} \cdot \boldsymbol{L} + \boldsymbol{E}], \tag{14.2.12}$$

where Ω marks out pixels in the shadows and the sparse matrix $\boldsymbol{E}$ accounts for corruptions by inter-reflections or specularities.

In order to find out the light directions $\boldsymbol{L}$ and the surface normals $\boldsymbol{N}$, we need to recover the complete matrix $\boldsymbol{A} = \boldsymbol{N} \cdot \boldsymbol{L}$. Since $\boldsymbol{A}$ is of rank at most 3, the problem becomes a low-rank matrix completion problem subject to sparse errors $\boldsymbol{E}$. That is we need to solve the following optimization problem:

$$\min_{\boldsymbol{A},\boldsymbol{E}} \operatorname{rank}(\boldsymbol{A}) + \gamma \|\boldsymbol{E}\|_0 \quad \text{subject to} \quad \mathcal{P}_\Omega[\boldsymbol{D}] = \mathcal{P}_\Omega[\boldsymbol{A} + \boldsymbol{E}], \tag{14.2.13}$$

where $\|\cdot\|_0$ denotes the ℓ^0-norm (number of non-zero entries in the matrix), and $\gamma > 0$ is a parameter that trades off the rank of the solution $\boldsymbol{A}$ versus the sparsity of the error $\boldsymbol{E}$.

Let $(\boldsymbol{A}_\star, \boldsymbol{E}_\star)$ be the optimal solution to (14.2.13). If the lighting directions $\boldsymbol{L}$ are given, we can easily recover the matrix $\boldsymbol{N}$ of surface normals from $\boldsymbol{A}_\star$ as:

$$\boldsymbol{N} = \boldsymbol{A}_\star \boldsymbol{L}^\dagger, \tag{14.2.14}$$

where $\boldsymbol{L}^\dagger$ denotes the Moore-Penrose pseudo-inverse of $\boldsymbol{L}$. The surface normals $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_m$ can then be estimated by normalizing each row of $\boldsymbol{N}$ to have unit norm.

14.3 Robust Matrix Completion Algorithm

While (14.2.13) follows from our formulation, it is not tractable since both rank and ℓ^0-norm are non-convex and discontinuous functions. As we have learned from earlier chapters, we can try to solve the convex version of this program:

$$\min_{\boldsymbol{A},\boldsymbol{E}} \|\boldsymbol{A}\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \mathcal{P}_\Omega[\boldsymbol{D}] = \mathcal{P}_\Omega[\boldsymbol{A} + \boldsymbol{E}]. \tag{14.3.1}$$

The above problem is almost identical to the PCP program studied in Chapter 5, except that the linear equality constraint is now applied only on the subset Ω of pixels that are not in the shadows. In the rest of this section, we show how the Augmented Lagrange Multiplier (ALM) method, earlier introduced for matrix completion or matrix recovery in Chapter 5, can be adapted to efficiently solve the problem (14.3.1) that requires simultaneously completing and correcting a low-rank matrix.

Recall the basic idea of the ALM method, introduced in Section 8.4 of Chapter 8, is to minimize the augmented Lagrangian function instead of the original constrained optimization problem. For our problem (14.3.1), the augmented Lagrangian is given by

$$\mathcal{L}_\mu(\boldsymbol{A}, \boldsymbol{E}, \boldsymbol{Y}) = \|\boldsymbol{A}\|_* + \lambda\|\boldsymbol{E}\|_1 + \langle \boldsymbol{Y}, \mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A} - \boldsymbol{E}]\rangle + \frac{\mu}{2}\|\mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A} - \boldsymbol{E}]\|_F^2, \tag{14.3.2}$$

where $\boldsymbol{Y} \in \mathbb{R}^{m\times n}$ is a Lagrange multiplier matrix, μ is a positive constant, $\langle \cdot, \cdot \rangle$ denotes the matrix inner product,[7] and $\|\cdot\|_F$ denotes the Frobenius norm. For appropriate choice of the Lagrange multiplier matrix $\boldsymbol{Y}$ and sufficiently large constant μ, it can be shown that the augmented Lagrangian function has the same minimizer as the original constrained optimization problem. The ALM algorithm iteratively estimates both the Lagrange multiplier and the optimal solution. The basic ALM iteration is given by

$$\left\{\begin{array}{rcl} (\boldsymbol{A}_{k+1}, \boldsymbol{E}_{k+1}) & = & \operatorname{argmin}_{\boldsymbol{A},\boldsymbol{E}}\ \mathcal{L}_{\mu_k}(\boldsymbol{A}, \boldsymbol{E}, \boldsymbol{Y}_k), \\ \boldsymbol{Y}_{k+1} & = & \boldsymbol{Y}_k + \mu_k\, \mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A}_{k+1} - \boldsymbol{E}_{k+1}], \\ \mu_{k+1} & = & \rho \cdot \mu_k, \end{array}\right. \tag{14.3.3}$$

where $\{\mu_k\}$ is a monotonically increasing positive sequence ($\rho > 1$).

We now focus our attention on solving the non-trivial first step of the above iteration. Since it is difficult to minimize $\mathcal{L}_{\mu_k}(\cdot)$ with respect to both $\boldsymbol{A}$ and $\boldsymbol{E}$ simultaneously, we adopt an alternating minimization strategy as follows:

$$\left\{\begin{array}{rcl} \boldsymbol{E}_{j+1} & = & \operatorname{argmin}_{\boldsymbol{E}}\ \lambda\|\boldsymbol{E}\|_1 - \langle \boldsymbol{Y}_k, \mathcal{P}_\Omega[\boldsymbol{E}]\rangle + \frac{\mu_k}{2}\|\mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A}_j - \boldsymbol{E}]\|_F^2, \\ \boldsymbol{A}_{j+1} & = & \operatorname{argmin}_{\boldsymbol{A}}\ \|\boldsymbol{A}\|_* - \langle \boldsymbol{Y}_k, \mathcal{P}_\Omega[\boldsymbol{A}]\rangle + \frac{\mu_k}{2}\|\mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A} - \boldsymbol{E}_{j+1}]\|_F^2. \end{array}\right. \tag{14.3.4}$$

Without loss of generality, we assume that the $\boldsymbol{Y}_k$'s and the $\boldsymbol{E}_k$'s (and hence, $\boldsymbol{Y}$ and $\boldsymbol{E}$, respectively) have their support in Ω^c. Then, the above minimization problems in (14.3.4) can be solved as described below.

Recall from the proximal gradient method in Chapter 8 that the soft-thresholding (or *shrinkage*) operator for scalars is as follows:

$$\operatorname{soft}(x, \alpha) = \operatorname{sign}(x) \cdot \max\{|x| - \alpha, 0\}, \tag{14.3.5}$$

where $\alpha > 0$.[8] When applied to vectors or matrices, the shrinkage operator acts

[7] $\langle \boldsymbol{X}, \boldsymbol{Y}\rangle \doteq \operatorname{trace}(\boldsymbol{X}^*\boldsymbol{Y})$.

[8] If $\alpha = 0$, then the shrinkage operator reduces to the identity operator.

Algorithm 14.1 (Matrix Completion and Recovery via ALM).

INPUT: $\boldsymbol{D} \in \mathbb{R}^{m\times n}$, $\Omega \subset \{1,\ldots,m\}\times\{1,\ldots,n\}$, $\lambda > 0$.
Initialize $\boldsymbol{A}_1 \leftarrow 0$, $\boldsymbol{E}_1 \leftarrow 0$, $\boldsymbol{Y}_1 \leftarrow 0$.
while not converged $(k = 1, 2, \ldots)$ **do**
  $\boldsymbol{A}_{k,1} = \boldsymbol{A}_k$, $\boldsymbol{E}_{k,1} = \boldsymbol{E}_k$;
  while not converged $(j = 1, 2, \ldots)$ **do**
    $\boldsymbol{E}_{k,j+1} = \text{soft}\left(\mathcal{P}_\Omega[\boldsymbol{D}] + \frac{1}{\mu_k}\boldsymbol{Y}_k - \mathcal{P}_\Omega[\boldsymbol{A}_{k,j}], \frac{\lambda}{\mu_k}\right)$;
    $t_1 = 1$; $\boldsymbol{Z}_1 = \boldsymbol{A}_{k,j}$; $\boldsymbol{A}_{k,j,1} = \boldsymbol{A}_{k,j}$;
    while not converged $(i = 1, 2, \ldots)$ **do**
      $(\boldsymbol{U}_i, \boldsymbol{\Sigma}_i, \boldsymbol{V}_i) = \text{SVD}\left(\frac{1}{\mu_k}\boldsymbol{Y}_k + \mathcal{P}_\Omega[\boldsymbol{D}] - \boldsymbol{E}_{k,j+1} + \mathcal{P}_{\Omega^c}[\boldsymbol{Z}_i]\right)$;
      $\boldsymbol{A}_{k,j,i+1} = \boldsymbol{U}_i\,\text{soft}\left(\boldsymbol{\Sigma}_i, \frac{1}{\mu_k}\right)\boldsymbol{V}_i^*$, $t_{i+1} = 0.5\left(1 + \sqrt{1+4t_i^2}\right)$;
      $\boldsymbol{Z}_{i+1} = \boldsymbol{A}_{k,j,i+1} + \frac{t_i - 1}{t_{i+1}}(\boldsymbol{A}_{k,j,i+1} - \boldsymbol{A}_{k,j,i})$, $\boldsymbol{A}_{k,j+1} = \boldsymbol{A}_{k,j,i+1}$;
    end while
    $\boldsymbol{A}_{k+1} = \boldsymbol{A}_{k,j+1}$; $\boldsymbol{E}_{k+1} = \boldsymbol{E}_{k,j+1}$;
  end while
  $\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \mu_k\,\mathcal{P}_\Omega[\boldsymbol{D} - \boldsymbol{A}_{k+1} - \boldsymbol{E}_{k+1}]$, $\mu_{k+1} = \rho\cdot\mu_k$;
end while
OUTPUT: $(\boldsymbol{A}_\star, \boldsymbol{E}_\star) = (\boldsymbol{A}_k, \boldsymbol{E}_k)$.

element-wise. Then, the first step in (14.3.4) has a closed-form solution given by

$$\boldsymbol{E}_{j+1} = \text{soft}\left(\mathcal{P}_\Omega[\boldsymbol{D}] + \frac{1}{\mu_k}\boldsymbol{Y}_k - \mathcal{P}_\Omega[\boldsymbol{A}_j], \frac{\lambda}{\mu_k}\right). \tag{14.3.6}$$

Since it is not possible to express the solution to the second step in (14.3.4) in closed-form, we adopt an iterative strategy based on the Accelerated Proximal Gradient (APG) algorithm discussed in Section 8.3 of Chapter 8 to solve it. The iterative procedure is given as:

$$\begin{cases} (\boldsymbol{U}_i, \boldsymbol{\Sigma}_i, \boldsymbol{V}_i) &= \text{SVD}\left(\frac{1}{\mu_k}\boldsymbol{Y}_k + \mathcal{P}_\Omega[\boldsymbol{D}] - \boldsymbol{E}_{j+1} + \mathcal{P}_{\Omega^c}[\boldsymbol{Z}_i]\right), \\ \boldsymbol{A}_{i+1} &= \boldsymbol{U}_i\,\text{soft}\left(\boldsymbol{\Sigma}_i, \frac{1}{\mu_k}\right)\boldsymbol{V}_i^*, \\ \boldsymbol{Z}_{i+1} &= \boldsymbol{A}_{i+1} + \frac{t_i - 1}{t_{i+1}}(\boldsymbol{A}_{i+1} - \boldsymbol{A}_i), \end{cases} \tag{14.3.7}$$

where $\text{SVD}(\cdot)$ denotes the singular value decomposition operator, and $\{t_i\}$ is a positive sequence satisfying $t_1 = 1$ and $t_{i+1} = 0.5\left(1 + \sqrt{1+4t_i^2}\right)$. The entire algorithm to solve (14.3.1) has been summarized as Algorithm 14.1.

14.4 Experimental Evaluation

In this section, we verify the effectiveness of the proposed method using both synthetic and real-world images. We compare results of the above robust matrix

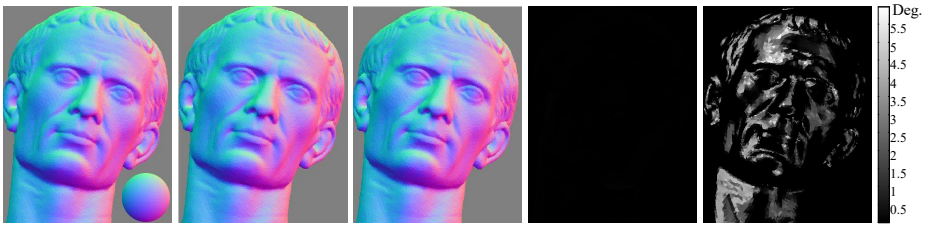


Figure 14.6 Specular Scene Results. 40 different images of Caesar were generated using the Cook-Torrance model for specularities. (a) Ground truth normal map with reference sphere. (b) and (c) show the surface normals recovered by the robust matrix completion (RMC) method and LS, respectively. (d) and (e) show the pixel-wise angular error w.r.t. the ground truth.

completion (RMC) method with a simple Least Squares (LS) approach, which assumes the ideal diffusive model given by (14.2.5). However, we do not use those pixels that were classified as shadows (the set Ω). Thus, the LS method can be summarized by the following optimization problem:

$$\min_{\boldsymbol{N}} \|\mathcal{P}_{\Omega}[\boldsymbol{D} - \boldsymbol{N} \cdot \boldsymbol{L}]\|_F. \tag{14.4.1}$$

We first test the algorithms using synthetic images whose ground-truth normal maps are known. In these experiments, we quantitatively verify the correctness of the algorithms by computing the angular errors between the estimated normal map and the ground-truth. We then test the algorithms on more challenging real images. Throughout this section, we denote by m the number of pixels in the region of interest in each image, and by n the number of input images (typically, $m \gg n$).

14.4.1 Quantitative Evaluation with Synthetic Images

In this section, we use synthetic images of three different objects (see Fig. 14.5(a)-(c)) under different scenarios to evaluate the performance of the algorithms. Since these images are free of any noise, we use a pixel threshold value of zero to detect shadows in the images. Unless otherwise stated, we set $\lambda = 1/\sqrt{m}$ in (14.3.1).

a. Specular Objects.
In this experiment, we generate images of an object under 40 different lighting conditions, where the lighting directions are chosen at random from a hemisphere with the object placed at the center. The images are generated with some specular reflection. For all experiments, we use the Cook-Torrance reflectance model [CT81] to generate images with specularities. Thus, there are two sources of corruption in the images – attached shadows and specularities.

A quantitative evaluation of our method and the Least Squares approach is presented in Table 14.1. The estimated normal maps are shown in Fig. 14.6(b),(c).

Object	Mean error		Max. error		% of corrupted pixels	
	LS	RMC	LS	RMC	Shadow	Specularity
Sphere	0.99	$\mathbf{5.1 \times 10^{-3}}$	8.1	**0.20**	18.4	16.1
Caesar	0.96	$\mathbf{1.4 \times 10^{-2}}$	8.0	**0.22**	20.7	13.6
Elephant	0.96	$\mathbf{8.7 \times 10^{-3}}$	8.0	**0.29**	18.1	16.5

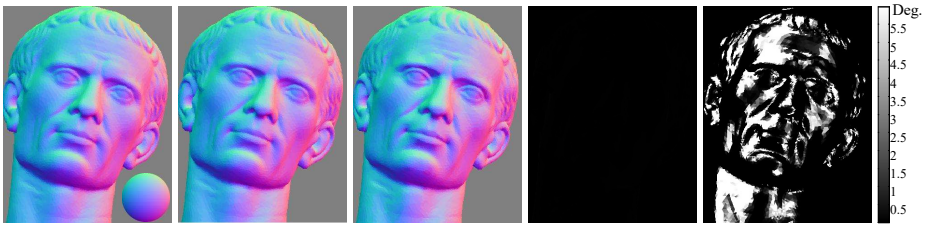
Table 14.1 Specular Scene Results. Statistics of angle error (in degrees) in the normals for different objects. In each case, 40 images were used. In the rightmost column, we indicate the average percentage of pixels corrupted by attached shadows and specularities in each image.

We use the RGB channel to encode the 3 spatial components (XYZ) of the normal map for display purposes. The error is measured in terms of the angular difference between the ground truth normal and the estimated normal at each pixel location. The pixel-wise error maps are shown in Fig. 14.6(d),(e). From the mean and the maximum angular error (in degrees) in Table 14.1, we see that the RMC method is much more accurate than the LS approach. This is because specularities introduce large magnitude errors to a small fraction of pixels in each image whose locations are unknown. The LS algorithm is not robust to such corruptions while RMC can correct these errors and recover the underlying rank-3 structure of the matrix. The column on the extreme right of Table 14.1 indicates the average percentage of pixels in each image (averaged over all images) that were corrupted by shadows and specularities, respectively. We note that even when more than 30% of the pixels are corrupted by shadows and specularities, RMC can efficiently retrieve the surface normals.

b. Textured Objects.

We also test the RMC method using a textured scene. Like the traditional photometric stereo approach, the RMC method does not have a dependency on the albedo distribution and works well on such scenes.

We use 40 images of Caesar for this experiment with each image generated under a different lighting condition (see Fig. 14.5(d) for example input image). The estimated normal maps as well as the pixel-wise error maps are shown in Fig. 14.7. We provide a quantitative comparison in Table 14.2 with respect to the ground-truth normal map. From the mean and maximum angular errors, it is evident that the RMC performs much better than the LS approach in this scenario.



(a) Ground truth (b) Our method (c) Least Squares (d) Error map (our method) (e) Error map (LS)

Figure 14.7 Textured Scene with Specularity. 40 different images of Caesar were generated with texture and using the Cook-Torrance model for specularities. (a) Ground truth normal map with reference sphere. (b) and (c) show the surface normals recovered by RMC and LS, respectively. (d) and (e) show the pixel-wise angular error w.r.t. ground truth.

Object	Mean error (in degrees)		Max error (in degrees)	
	LS	RMC	LS	RMC
Caesar	2.4	**0.016**	32.2	**0.24**

Table 14.2 Textured Scene with Specularity: Statistics of angle errors. We use 40 images under different illuminations.

c. Effect of the Number of Images.

In the above experiments, we have used images of the object under 40 different illuminations. In this experiment, we study the effect of the number of illuminations used. In particular, we would like to find out empirically the minimum number of images required for the RMC method to be effective. For this experiment, we generate images of Caesar using the Cook-Torrance reflectance model, where the lighting directions are generated at random. The mean percentage of specular pixels in the input images is maintained approximately constant at 10%. The angular difference between the estimated normal map and the ground truth is used as a measure of accuracy of the estimate.

Num of images		10	20	30	40
Mean error (in degrees)	LS	0.52	0.53	0.59	0.57
	RMC	**0.23**	**0.026**	**0.019**	**0.013**
Max. error (in degrees)	LS	**34.5**	9.0	7.6	7.0
	RMC	56.6	**5.8**	**0.48**	**0.37**

Table 14.3 Effect of Number of Images. We use synthetic images of Caesar under different lighting conditions. The number of illuminations is varied from 10 to 40. The angle error is measured with respect to the ground truth normal map. The illuminations are chosen at random, and the error has been averaged over 20 different sets of illumination.

The experimental results are given in Table 14.3. We observe that with less

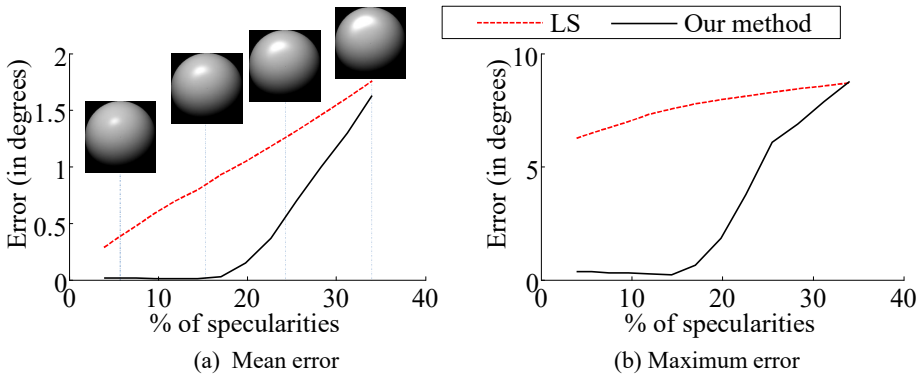


Figure 14.8 Effect of Increasing Size of Specular Lobes. We use synthetic images of Caesar under 40 randomly chosen lighting conditions. (a) Mean angular error, (b) Maximum angular error w.r.t. the ground truth. The illuminations are chosen at random, and the error has been averaged over 10 different sets of illumination. (a) contains illustrations of increasing size of specular lobe.

than 10 input illuminations, estimates of both algorithms are very inaccurate but RMC is worse than LS. However, when the number of illuminations is larger than 10, we observe that the mean error in the LS estimate becomes higher than that RMC. Upon increasing the number of images further, the proposed method consistently outperforms the LS approach. If the number of input images is less than 20, then the maximum error in the LS estimate is smaller than that of RMC. However, RMC performs much better when more than 30 different illuminations are available. Thus, the proposed technique performs significantly better as the number of input images increases.

d. Varying Amount of Specularities.
From the above experiments, it is clear that the proposed technique is quite robust to specularities in the input images when compared to the LS method. In this experiment, we empirically determine the maximum amount of specularity that can be handled by RMC. We use the Caesar scene under 40 randomly chosen illumination conditions for this experiment. On an average, about 20% of the pixels in each image is corrupted by attached shadows. We vary the size of the specular lobe in the input images (as illustrated in Fig. 14.8(a)), thereby varying the number of corrupted pixels. We compare the accuracy of RMC against the LS technique using the angular error of the estimates with respect to the ground-truth.

The experimental results are illustrated in Fig. 14.8. We observe that RMC is very robust when up to 16% of all pixels in the input images are corrupted by specularities. The LS method, on the other hand, is extremely sensitive to

C	1.0	0.8	0.6	0.4
Mean error (in degrees)	1.42	0.78	0.19	0.029
Max. error (in degrees)	8.78	8.15	1.86	0.91

Table 14.4 Handling More Specularities by Tuning λ. We use 40 images of Caesar under different lighting conditions with about 28% specularities and 20% shadows, and set $\lambda = C/\sqrt{m}$.

even small amounts of specularities in the input images. The angular error in the estimates of both methods rises as the size of the specular lobe increases.

e. Enhancing Performance by Tuning λ.
We recall that λ is a weighting parameter in our formulation given by (14.3.1). In all the above experiments, we have fixed the value of the parameter $\lambda = 1/\sqrt{m}$, as suggested by the theory in Chapter 5. While this choice promises a certain degree of error correction, it may be possible to correct larger amounts of corruption by choosing λ appropriately, as demonstrated in [GWL+10] for instance. Unfortunately, the best choice of λ depends on the input images, and cannot be determined analytically.

We demonstrate the effect of the weighting parameter λ on a set of 40 images of Caesar used in the previous experiments. In this set of images, approximately 20% of the pixels are corrupted by attached shadows and about 28% by specularities. We choose $\lambda = C/\sqrt{m}$, and vary the value of C. We evaluate the results using angular error with respect to the ground-truth normal map. We observe from Table 14.4 that the choice of C influences the accuracy of the estimated normal map. For real-world applications, where the data is typically noisy, the choice of λ could play an important role in the efficacy of RMC.

f. Computation.
The core computation of RMC is solving a convex program (14.3.1). For the specular Caesar data (Fig. 14.5(b)) with 40 images of 450×350 resolution, a single-core MATLAB implementation of RMC takes about 7 minutes on a Macbook Pro with a 2.8 GHz Core 2 Duo processor and 4 GB memory, as against 42 seconds taken by the LS approach. While RMC is slower than the LS approach, it is much more accurate in a wide variety of scenarios and is more efficient than other methods (e.g. [MHI10]).

14.4.2 Qualitative Evaluation with Real Images

We now test the algorithms on real images. We use a set of 40 images of a toy Doraemon and Two-face taken under different lighting conditions (see Fig. 14.9(a), (d)). A glossy sphere was placed in the scene for light source calibration when capturing the data. We used a Canon 5D camera with the RAW image mode.[9]

[9] We did not apply Gamma correction.

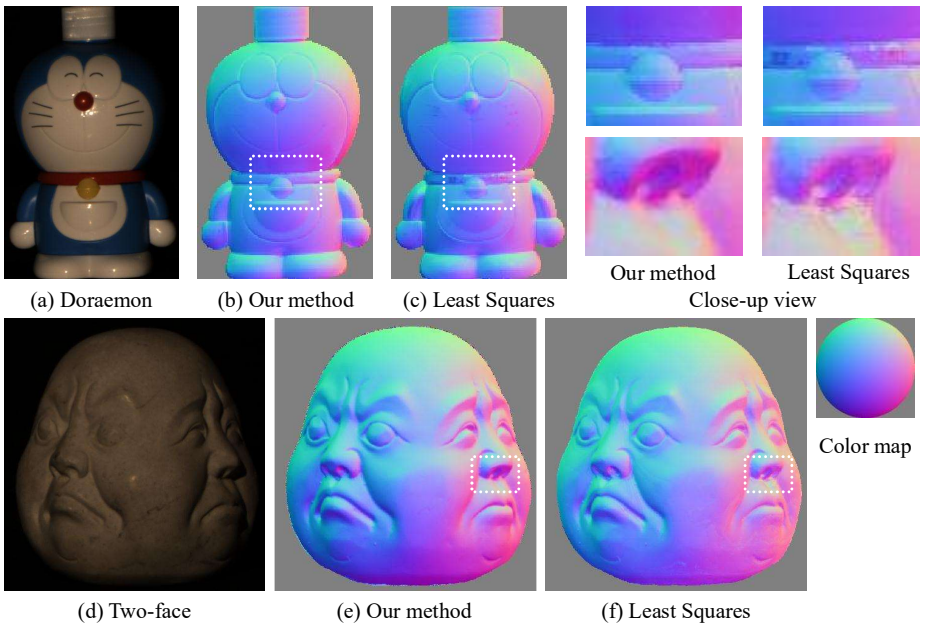


Figure 14.9 Qualitative Comparison on Real Images. We use images of Doraemon and Two-face taken under 40 different lighting conditions to qualitatively evaluate the performance of the RMC method against the LS approach. (a),(d) Sample input images. (b),(e) Normal map estimated by RMC. (c),(f) Normal map estimated by Least Squares. Close-up views of the dotted rectangular areas are shown on the top-right.

These images present new challenges to RMC. In addition to shadows and specularities, there is potentially additional noise inherent to the acquisition process as well as possible deviations from the idealistic Lambertian model illuminated by distant lights. In this experiment, we use a threshold of 0.01 to detect shadows in images.[10] We also found experimentally that setting $\lambda = 0.3/\sqrt{m}$ works well for these datasets.

Since the ground truth normal map is not available for these scenes, we compare the RMC method and the LS approach by visual inspection of the output normal maps shown in Fig. 14.9(b),(c),(e),(f). We observe that the normal map estimated by RMC appears smoother and hence, more realistic. This can be observed particularly around the necklace area in Doraemon and nose area in Two-face (see Fig. 14.9) where the LS estimate exhibits some discontinuity in the normal map.

[10] All pixels are normalized to have intensity between 0 and 1.

14.5 Notes

Low-dimensionality from Illumination.
It is well understood that when a Lambertian surface is illuminated by at least three known lighting directions, the surface orientation at each visible point can be uniquely determined from its intensities. From different perspectives, it has long been shown that if there are no shadows, the appearance of a convex Lambertian scene illuminated from different lighting directions span a three-dimensional subspace [Sha92] or an illumination cone [BK96]. Basri and Jacobs [BJ03] and Georghiades *et al.* [GBK01] have further shown that the images of a convex-shaped object with cast shadows can also be well-approximated by a low-dimensional linear subspace. The more recent study [ZMKW13] has shown that even for a nonconvex Lambertian object, the images can be well modeled as a low-rank matrix plus some sparse errors. The aforementioned works indicate that there exists a degenerate structure in the appearance of Lambertian surfaces under variation in illumination. This is the key property that all photometric stereo methods harness to determine the surface normals.

Classical Methods for Photometric Stereo.
Previously, photometric stereo algorithms for Lambertian surfaces generally find surface normals as the *Least Squares* solution to a set of linear equations that relate the observations and known lighting directions, or equivalently, try to identify the low-dimensional subspace using conventional Principal Component Analysis (PCA) [Jol86]. Such a solution is known to be optimal if the measurements are corrupted by only *i.i.d.* Gaussian noise of small magnitude. Unfortunately, in reality, photometric measurements rarely obey such a simplistic noisy linear model: the intensity values at some pixels can be severely affected by specular reflections (deviation from the basic Lambertian assumption), sensor saturations, or shadowing effects. As a result, the Least Squares solution normally ends up with incorrect estimates of surface orientations in practice. To overcome this problem, researchers have explored various heuristic approaches to eliminate such deviations by treating the corrupted measurements as outliers, e.g., using the so-called RANSAC scheme [FB81, CHC08], or a median-based approach [MHI10]. To identify the different types of corruptions in images more carefully, Mukaigawa *et al.* [MMMS01, MIS07] have proposed a method for classifying diffuse, specular, attached, and cast shadow pixels based on RANSAC and outlier elimination.

Low-rank Matrix Approach.
The method presented in this chapter was first introduced through the work [WGS+10]. In contrast to previous robust approaches, this method is computationally more efficient and provides theoretical guarantees for robustness to large errors. More importantly, the method is able to use all the available information simultaneously for obtaining the optimal result, instead of pre-processing

measurements which might discard useful information, e.g., by either selecting the best set of illumination directions [CHC08] or using the median estimator [MHI10]. The method in this chapter can also be used to improve virtually any existing photometric stereo method, including uncalibrated photometric stereo [Hay94a], where traditionally, corruption in the data (e.g., by specularities) is either neglected or ineffectively dealt with conventional heuristic robust estimation methods.

15 Structured Texture Recovery

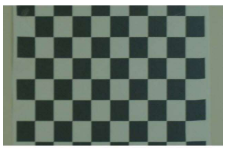
"What humans do with the language of mathematics is to describe patterns... To grow mathematically children must be exposed to a rich variety of patterns appropriate to their own lives through which they can see variety, regularity, and interconnections."
– Lynn Arthur Steen, *The Future of Mathematics Education*

15.1 Introduction

In man-made environments, most objects of interest are rich of regular, repetitive, symmetric structures. Figure 15.1 shows images of some representative structured objects. An image of such an object clearly inherits such regular structures and encodes rich information about the 3D shape, pose, or identity of the object. If we view the image of such an object as a matrix, columns of the matrix will obviously be correlated to one another hence the rank of the matrix will be very low, or approximately so. For example, for reflectively symmetric objects like a face or a car, the rank of their images will be at most half of the size of the matrices. Besides being symmetric, images of such objects typically have other additional structures (e.g. piecewise smooth etc) which will render the rank of the image even much lower.[1] We generally refer to images (or image regions) associated with such structure objects as "structured textures" to separate them from other random textures.

In this chapter, we will study how the low-dimensional structures of such structured textures may help us to robustly and accurately recover the appearance, pose, and shape of the associated objects in 2D or 3D. This makes structured textures extremely important for many computer vision tasks such as recognition, localization, and reconstruction of objects in man-made environments. From a compressive sensing perspective, we will see in this chapter how to recover a low-rank matrix (that models structured textures) despite significant corruption (due to occlusion) or transformation[2] (due to pose, shape or camera lens distortion etc.) To be more precise, we first introduce some notation.

[1] The reader should test validity of this assumption by computing the actual rank of real images similar to those in Figure 15.1. We leave this as an exercise.
[2] in the 2D domain of the matrix

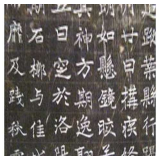

(a) a calibration rig


(b) a carpet


(c) windows


(d) a door

(e) a license plate


(f) characters


(g) a car


(h) a face

Figure 15.1 Representative examples of structured objects. These images viewed as matrices are all (approximately) low-rank matrices. The face image is from the Extended Yale Face Database B [GBK01].

15.2 Low-Rank Textures

Strictly speaking, an image (viewed as a matrix[3]) is a discrete sampling of a continuous texture (function) defined on a 2D domain. Consider a 2D texture as a function $\boldsymbol{I}_o(x, y)$, defined in $\mathbb{R}^2$. We say that $\boldsymbol{I}_o$ is a *low-rank texture* if the family of one-dimensional functions $\{\boldsymbol{I}_o(x, y) \mid y \in \mathbb{R}\}$ span a finite low-dimensional linear subspace *i.e.*,

$$r \doteq \dim(\text{span}\{\boldsymbol{I}_o(x, y) \mid y \in \mathbb{R}\}) \leq k \tag{15.2.1}$$

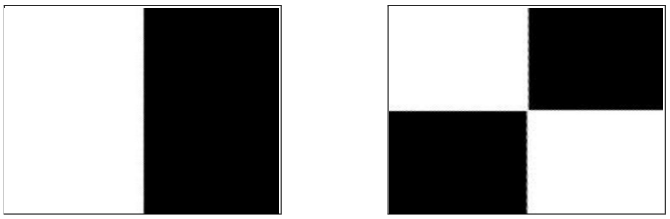
for some small positive integer k. If r is finite, then we refer to $\boldsymbol{I}_o$ as a rank-r texture. It is easy to see that a rank-1 function $\boldsymbol{I}_o(x, y)$ must be of the form $u(x) \cdot v(y)$ for some functions $u(x)$ and $v(y)$; and in general, a rank-r function $\boldsymbol{I}_o(x, y)$ can be explicitly factorized as as the combination of r rank-1 functions:

$$\boldsymbol{I}_o(x, y) \doteq \sum_{i=1}^{r} u_i(x) \cdot v_i(y). \tag{15.2.2}$$

Figure 15.2 shows some ideal low-rank textures: edges and corners were traditionally used in computer vision to characterize local features of an object and they can be viewed as the simplest low-rank textures. An ideal vertical edge (or slope) as shown in Figure 15.2 left can be considered a rank-1 texture with $u(x) = -\text{sign}(x)$ and $v(y) = 1$. An ideal corner as shown in Figure 15.2 is also a rank-1 texture with $u(x) = \text{sign}(x)$ and $v(y) = \text{sign}(y)$.

Thus, in a sense, the notion of low-rank textures unifies many of the conventional local features but goes beyond that: By its definition, it is easy to see

[3] Hence, in this chapter, we will use the bold face symbol $\boldsymbol{I}$ to denote an image because we will mostly identify the image as a matrix.



(a) an edge (b) a corner

Figure 15.2 Examples of some ideal low-rank textures: an edge and a corner.

that *images of regular, repetitive, symmetric patterns typically lead to low-rank textures*. Low-rank textures of rank higher than one would be able to represent much richer class of structured objects than local edges or corners and they can capture the global characteristics of a structured object.

Given a low-rank texture, obviously its rank is *invariant* under any scaling of the function, as well as scaling or translation in the x and y coordinates. That is, if

$$\boldsymbol{I}(x, y) \doteq \alpha \cdot \boldsymbol{I}_o(ax + t_1, by + t_2)$$

for some constants $\alpha, a, b \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R}$, then $\boldsymbol{I}(x, y)$ and $\boldsymbol{I}_o(x, y)$ have the same rank according to the definition in (15.2.1). For most practical purposes, it suffices to recover any scaled or translated version of the low-rank texture $\boldsymbol{I}_o(x, y)$, as the remaining ambiguity left in the scaling can often be easily resolved in practice by imposing additional constraints on the texture. Hence, in this chapter, unless otherwise stated, we view two low-rank textures *equivalent* if they are scaled and translated versions of each other:

$$\boldsymbol{I}_o(x, y) \sim \boldsymbol{I}_o(ax + t_1, by + t_2),$$

for some $a, b, c \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R}$. In homogeneous representation, this equivalence group of transformations consists of all elements of the form:

$$g \in \left\{ \begin{bmatrix} a & 0 & t_1 \\ 0 & b & t_2 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3} \,\middle|\, a, b \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R} \right\}. \tag{15.2.3}$$

It is however easy to see that the low-rank form 15.2.2 will *not* be preserved under a general linear transform of the domain:

$$\boldsymbol{I}(x, y) \doteq \boldsymbol{I}_o(ax + bx + t_1, cx + dy + t_2) \tag{15.2.4}$$

will have a different (usually much higher) rank than that of $\boldsymbol{I}_o(x, y)$. For instance, if we rotate the edge or the corner in Figure 15.2 by 45°, the resulting image will become full rank (as a matrix). Similarly, the rank will mostly increase under more general nonlinear distortions or transformations of the domain. As we will see in this chapter, this fact is actually rather beneficial: it suggests that

the correct transformation that can undo the distortion would be the one that makes the rank of the texture the lowest.

In practice, an image of a 2D texture is not a continuous function defined on $\mathbb{R}^2$. We only have its values sampled on a finite discrete grid in $\mathbb{Z}^2$, of size $m \times n$ say. In this case, the 2D texture $\boldsymbol{I}_o(x, y)$ is represented by an $m \times n$ matrix of real numbers. For a low-rank texture, we always assume that the size of the sampling grid is significantly larger than the intrinsic rank of the texture,[4] i.e.

$$r \ll \min\{m, n\}.$$

It is easy to show that as long as the sampling rate is not one of the aliasing frequencies of the functions $u_i(x)$ or $v_i(x)$ of the continuous $\boldsymbol{I}_o(x, y)$ defined in (15.2.2), the resulting matrix has the same rank as the continuous function.[5] Thus, the 2D texture $\boldsymbol{I}_o(x, y)$ when discretized as a matrix, denoted by $\boldsymbol{I}_o(i, j)$ for convenience, has very low rank relative to its dimensions.

For the remaining of this chapter, for convenience, we will treat the continuous 2D function and its sampled matrix form as the same, with the understanding that whenever we talk about distortion or transformation of a texture or an image, we mean a transformation in the 2D domain of its underlying continuous function. When only an image (a matrix of sampled values) is given, values of the function off the sampling grid can be obtained through any reasonable interpolation schemes.[6]

15.3 Structured Texture Inpainting

In this section, we will see how to automatically repair a structured texture when it is severely corrupted or occluded. From Chapters 4, we know if a texture $\boldsymbol{I}_o$ is a low-rank matrix, we can recover it even if only a small fraction of its entries (pixels), say with support Ω, are observable. Let Ω be the set of pixels given. The problem to recover the full texture image $\boldsymbol{I}_o$ is simply a low-rank matrix completion problem:

$$\min_{\boldsymbol{L}} \operatorname{rank}(\boldsymbol{L}) \quad \text{subject to} \quad \boldsymbol{L}(i, j) = \boldsymbol{I}_o(i, j)\ \forall (i, j) \in \Omega. \tag{15.3.1}$$

Although being low-rank is a necessary condition for most regular, structured textures, it is certainly *not sufficient*. Figure 15.3 shows three images that have exactly the same rank. Obviously the first two are more smooth and regular than the third one. As discussed in the preceding section, a rank-r texture is a 2D function $\boldsymbol{I}_o(x, y)$, defined on $\mathbb{R}^2$. Then $\boldsymbol{I}_o(x, y)$ can be factored as

$$\boldsymbol{I}_o(x, y) = \sum_{i=1}^{r} u_i(x) v_i(y).$$

[4] The scale of the window needs to be large enough to meet this assumption.
[5] In other words, the resolution of the image cannot be too low.
[6] From our experience, the bicubic interpolation is good enough for most purposes.

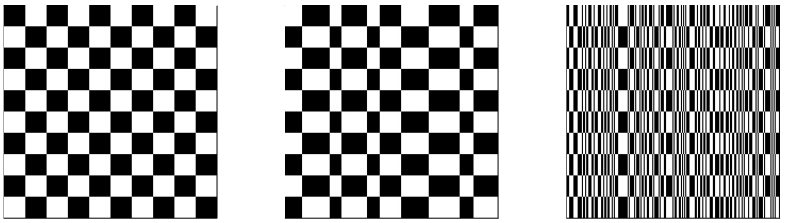


Figure 15.3 Different textural patterns: all three textures have exactly the same rank, but they go from purely regular, to nearly regular, and to almost irregular texture.

If $\boldsymbol{I}_o$ represents a more realistic regular or near regular pattern, it is typically piecewise smooth. Hence, the functions u_i and v_i are not arbitrary and they may have additional structures. As we have discussed in earlier chapters of the book, piecewise smooth functions are typically sparse in certain transformed domain (say, by a wavelet transform).

So, in the discrete setting, the low-rank matrix $\boldsymbol{I}_o$ can be factorized as

$$\boldsymbol{I}_o = \boldsymbol{U}\boldsymbol{V}^*,$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ can be represented as

$$\boldsymbol{U} = \boldsymbol{B}_1\boldsymbol{X}_1 \quad \text{and} \quad \boldsymbol{V} = \boldsymbol{B}_2\boldsymbol{X}_2$$

for some pair of bases $(\boldsymbol{B}_1, \boldsymbol{B}_2)$. If the bases are properly chosen, both $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ will be sufficiently sparse. Or equivalently, if we write

$$\boldsymbol{I}_o = \boldsymbol{B}_1\boldsymbol{X}_1\boldsymbol{X}_2^*\boldsymbol{B}_2^* \doteq \boldsymbol{B}_1\boldsymbol{W}_o\boldsymbol{B}_2^*,$$

then the matrix $\boldsymbol{W}_o \doteq \boldsymbol{X}_1\boldsymbol{X}_2^*$ will be a sparse matrix, which has the same (low) rank as $\boldsymbol{I}_o$.

Hence, if we want the recovered image from a partially observed $\boldsymbol{I}$ (of the ground truth $\boldsymbol{I}_o$) to be both low-rank and sparse (in certain transformed domain), we could modify the low-rank matrix completion problem (15.3.1) as follows to impose additional spatial structures:

$$\min_{\boldsymbol{L},\boldsymbol{W}} \ \operatorname{rank}(\boldsymbol{L}) + \lambda\|\boldsymbol{W}\|_0 \quad \text{s.t.} \quad \mathcal{P}_\Omega[\boldsymbol{L}] = \mathcal{P}_\Omega[\boldsymbol{I}],\ \boldsymbol{L} = \boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^*, \tag{15.3.2}$$

where $\|\boldsymbol{W}\|_0$ denotes the number of non-zero entries in $\boldsymbol{W}$. That is, we aim to find the ground truth texture image $\boldsymbol{I}_o$ as the lowest possible rank matrix $\boldsymbol{L}_\star$ and the matrix $\boldsymbol{W}_\star = \boldsymbol{B}_1^*\boldsymbol{L}_\star\boldsymbol{B}_2$ with the fewest possible non-zero entries that agrees with the partial observation $\mathcal{P}_\Omega[\boldsymbol{I}]$. Here, λ is a weighting parameter which trades off the rank and sparsity of the recovered image.

As we have learned from earlier chapters, in the above problem (15.3.2), both the rank function and the ℓ^0-norm are difficult to optimize directly. Instead, they can be replaced by their convex surrogates: the matrix nuclear norm $\|\boldsymbol{L}\|_*$ for

rank $(\boldsymbol{L})$ and the ℓ_1-norm $\|\boldsymbol{W}\|_1$ for $\|\boldsymbol{W}\|_0$, respectively. Thus, we end up with the following optimization problem:

$$\min_{\boldsymbol{L},\boldsymbol{W}} \ \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{W}\|_1 \quad \text{s.t.} \quad \mathcal{P}_\Omega[\boldsymbol{L}] = \mathcal{P}_\Omega[\boldsymbol{I}],\ \boldsymbol{L} = \boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^*. \tag{15.3.3}$$

If we further assume that the bases $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ used are orthonormal, we have $\|\boldsymbol{I}_o\|_* = \|\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^*\|_* = \|\boldsymbol{W}\|_*$. The convex program (15.3.3) is equivalent to:

$$\min_{\boldsymbol{W}} \ \|\boldsymbol{W}\|_* + \lambda\|\boldsymbol{W}\|_1 \quad \text{s.t.} \quad \mathcal{P}_\Omega[\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^*] = \mathcal{P}_\Omega[\boldsymbol{I}]. \tag{15.3.4}$$

This formulation allows us to enforce that the recovered texture image be simultaneously low-rank and sparse in certain transformed domain. As we have discussed in Section 6.3 of Chapter 6, the above convex relaxation is only suboptimal for enforcing simultaneous sparse and low-rank structures on $\boldsymbol{W}_o$. Nevertheless, as we will see, in practice this formulation is sufficient for our purposes of recovering low-rank images.

Notice that entry-wise observation operator $\mathcal{P}_\Omega[\cdot]$ is not incoherent with a sparse matrix. However, here instead of directly sampling $\boldsymbol{W}_o$, the operator samples a transformed version of $\boldsymbol{W}_o$ by the bases $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$. As we will see in experiments, apparently such transforms make the operator $\mathcal{P}_\Omega[\cdot]$ "incoherent" with both the sparse and low-rank structures $\boldsymbol{W}_o$.[7]

Furthermore, notice that the above convex program (15.3.4) is different from the convex program we have encountered before in problems like PCP where the nuclear norm and ℓ^1 norm were for two different matrices. To utilize the same optimization techniques, we only have to introduce an auxiliary variable $\boldsymbol{L}$ to replace $\boldsymbol{W}$ in the low-rank term and render the variables separable:

$$\min_{\boldsymbol{L},\boldsymbol{W}} \ \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{W}\|_1 \quad \text{s.t.} \quad \boldsymbol{L} = \boldsymbol{W},\ \mathcal{P}_\Omega[\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^*] = \mathcal{P}_\Omega[\boldsymbol{I}]. \tag{15.3.5}$$

The reader may recognize this program falls into the same class of programs as PCP that we have dealt with in Chapter 5, and they can be solved efficiently by methods such as ALM and ADMM introduced in Chapter 8.[8]

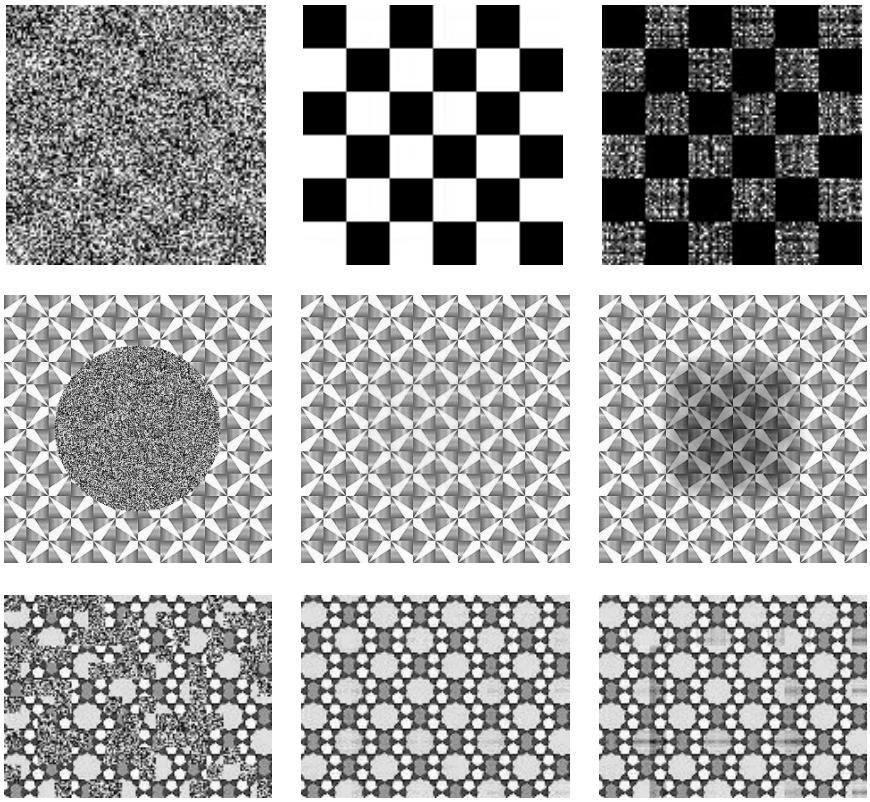
Example 15.1. *(Texture Inpainting). To demonstrate the importance of enforcing the sparse and low-rank prior together, we here conduct some texture inpainting experiments on real images and compare the solution to the above program with that to low-rank matrix completion algorithm from Chapter 4.*

Here we choose $\lambda = 0.001$*, and we use the discrete cosine transform (DCT) basis for both* $\boldsymbol{B}_1$ *and* $\boldsymbol{B}_2$.[9] *We test the recovery under three different types of corruptions: uniform random corruptions, one disk corruption, and random block*

[7] To our best knowledge, there is little result that rigorously characterizes conditions on such transforms such that correct recovery of $\boldsymbol{W}_o$ can be guaranteed, despite compelling empirical success.

[8] More detailed implementation of this particular program can be found in [LRZM12].

[9] DCT is the basis used in the JPEG image compression standard. Here the choice of DCT is for simplicity. One may also use a wavelet basis, such as the one used in JPEG2000, to obtain likely better performance.



(a) Input image (b) Sparse + Low-rank (c) Low-rank only

Figure 15.4 Qualitative comparison of sparse low-rank texture recovery and low-rank completion only. The first row is a checkerboard texture with 91% randomly chosen pixels corrupted (recall Theorem 5.10 of Chapter 5); the second row is a real texture with about 30% pixels occluded by a disk; and the third row is a real texture with about 40% pixels corrupted by random small blocks.

corruptions on three representative low-rank textures: a checkerboard image (typically used in camera calibration) and two real texture images. The checkerboard is of precise rank 2 *and the other two are full rank but approximately low-rank. From the completion results it is clear to see that the recovered results are significantly better by imposing both low-rank and sparse priors than low-rank alone.*

Almost all methods for image inpainting or completion need information about the support Ω of the corrupted regions (e.g., [BSBC00], [MES08], [FlSM09], etc.). This information is usually obtained through manually marked out by the user or detected by other independent methods. This often severely limits the applicability of all the image completion or inpainting methods.

In many practical scenarios, the information about the support of the corrupted regions might not be known or only partially known. Hence the pixels in the given region Ω can also contain some corruptions that violate the low-rank and sparse structures. Similar to the Robust PCA problem in Chapter 5, we could model such corruptions with unknown support as a sparse error term $\boldsymbol{E}_o$:

$$\boldsymbol{I} = \boldsymbol{I}_o + \boldsymbol{E}_o = \boldsymbol{B}_1 \boldsymbol{W}_o \boldsymbol{B}_2^* + \boldsymbol{E}_o.$$

To recover the image $\boldsymbol{I}_o = \boldsymbol{B}_1 \boldsymbol{W}_o \boldsymbol{B}_2^*$, we now only have to solve the following PCP like program:

$$\min_{\boldsymbol{W}} \|\boldsymbol{W}\|_* + \lambda \|\boldsymbol{W}\|_1 + \alpha \|\boldsymbol{E}\|_1 \quad \text{s.t.} \quad \mathcal{P}_\Omega[\boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^* + \boldsymbol{E}] = \mathcal{P}_\Omega[\boldsymbol{I}]. \tag{15.3.6}$$

Notice that if we know nothing about the corruption areas, we only need to set Ω to be the entire image. Just like PCP, the above convex program will decompose the image into a low-rank component and a sparse one. Of course, the nonzero entries in estimated $\boldsymbol{E}$ can help us to further refine the support Ω. For instance, we could simply set:

$$\operatorname{supp}(\boldsymbol{E}) \doteq \{(i,j) \in \Omega,\ |\boldsymbol{E}_{ij}| > \varepsilon\}, \tag{15.3.7}$$

for some threshold $\varepsilon > 0$. Or we could estimate the support of $\boldsymbol{E}$ using more sophisticated model to encourage additional structures such as spatial continuity [ZWMM09]. Once $\operatorname{supp}(\boldsymbol{E})$ is known, we can exclude those corrupted entries from Ω (the support of presumably good entries).

We could further iterate between the image completion and support estimation:

$$\begin{aligned} (\boldsymbol{W}_k, \boldsymbol{E}_k) &= \operatorname{argmin}_{\boldsymbol{W},\boldsymbol{E}} \|\boldsymbol{W}\|_* + \lambda \|\boldsymbol{W}\|_1 + \alpha \|\boldsymbol{E}\|_1 \\ &\quad \text{subject to } \mathcal{P}_{\Omega_k}[\boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^* + \boldsymbol{E}] = \mathcal{P}_{\Omega_k}[\boldsymbol{I}], \\ \Omega_{k+1} &= \Omega_k \setminus \operatorname{supp}(\boldsymbol{E}_{k+1}), \end{aligned} \tag{15.3.8}$$

where α is a weighting parameter between sparsity and low-rankness. We could continue the above process till convergence and obtain the repaired image $\boldsymbol{I}_\star = \boldsymbol{B}_1 \boldsymbol{W}_\star \boldsymbol{B}_2^*$. In practice, we notice a good side effect of adding the additional $\boldsymbol{E}$ term: It not only helps estimate support of the corrupted regions but also helps reduce noise on the repaired texture image $\boldsymbol{I}_\star$.

Example 15.2. *(Texture Recovery) In this experiment, we conduct some comparison between the above method and some typical image completion methods used in highly engineered commercial systems: Patch Match (PM) used by Adobe Photoshop [BSFG09, BSGF10], Image Completion with Structure Propagation (SP) developed by Microsoft [SYJS05]. Figure 15.5 shows the result on three different images: a simulated non-uniform low-rank texture, a uniform building facade, and a somewhat less uniform building facade, which correspond to three rows in Figure. 15.5, respectively.*

The other two methods all share the spirit of sample-based texture synthesis: they stitch sampled local patches together to ensure certain global statistical consistency. As these methods rely mostly on local statistics and structures, they

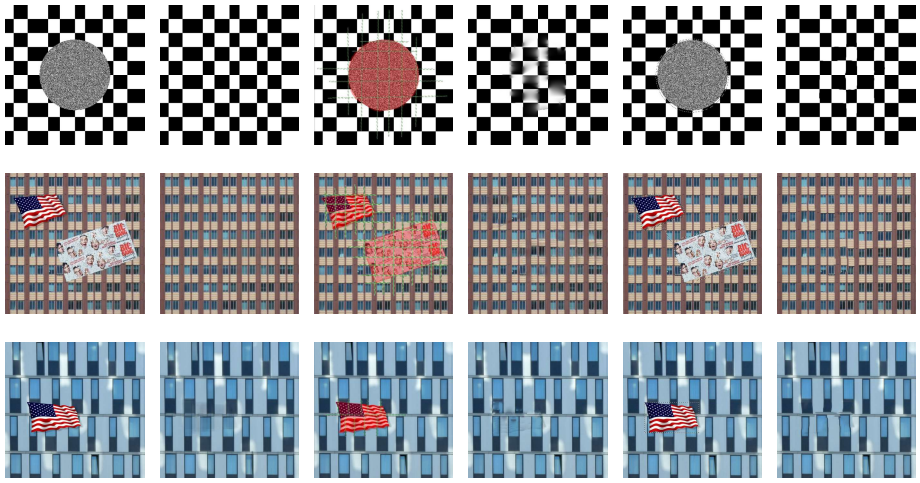


Figure 15.5 Comparison Results with Microsoft SP [SYJS05] and Adobe Photoshop [BSFG09]. Columns 1-2: inputs and results of the structured texture recovery method; Columns 3-4: inputs and results of SP; Columns 5-6: inputs and results of Adobe Photoshop.

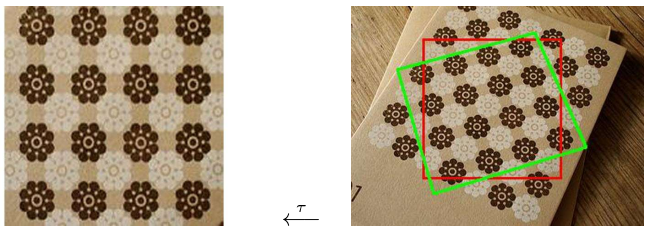
tend to work on natural images or random textures too while our method does not. However, as we see from the results, when applied to completing or repairing regular or near regular *low-rank patterns, they often fail to preserve the global regularity accurately. The reason is partially because these methods normally do not or cannot exploit global structural information about the textures.*

Unlike the structured texture recovery method introduced here, these image completion systems typically require the user to mark out rather precisely the to-be-corrected region or regions (as the contours of the regions for Photoshop shown in the figure), and even to provide additional information about the structures to be recovered (such as suggested lines marked out in the red regions that required by the SP method). However, the structured texture recovery method does not need any knowledge about the support of the corrupted regions nor any information about the structure.

15.4 Transform Invariant Low-Rank Textures

15.4.1 Deformed and Corrupted Low-rank Textures

Although a structured object that has regular or repetitive 2D or 3D textural patterns in space is often low-rank, its image $\boldsymbol{I}$ under an arbitrary camera viewpoint may exhibit much higher rank compared to its upright frontal view $\boldsymbol{I}_o(x, y)$. An example is illustrated in Figure 15.6. In order to extract the intrinsic



(a) Low-rank texture $\boldsymbol{I}_o$ (b) Its image $\boldsymbol{I}$ under a different viewpoint

Figure 15.6 An example of a transformed low-rank texture: The upright low-rank texture $\boldsymbol{I}_o$ on the left is associated with the region in the green window. The matrix $\boldsymbol{I}$ associated with the region in the red window is clearly not low-rank.

low-rank textures from such deformed images, we need to carefully model the effect of deformation and see how to undo it correctly.

Deformed Low-rank Textures.
Suppose a low-rank texture $\boldsymbol{I}_o(x, y)$ lies on certain surface in the scene. The image $\boldsymbol{I}$ is $\boldsymbol{I}_o$ taken from a certain camera viewpoint. We use τ to denote the transform where $\tau : \mathbb{R}^2 \to \mathbb{R}^2$ belongs to a certain Lie group $\mathbb{G}$ on $\mathbb{R}^2$ (We here only consider the case where τ is invertible). Hence $\boldsymbol{I} \circ \tau = \boldsymbol{I}_o$ or the image $\boldsymbol{I}$ can be viewed as transformed version of the original function $\boldsymbol{I}_o(x, y)$:

$$\boldsymbol{I}(x, y) = \boldsymbol{I}_o \circ \tau^{-1}(x, y) = \boldsymbol{I}_o \left(\tau^{-1}(x, y)\right), \quad \tau \in \mathbb{G}.$$

If the texture is on a planar surface in the 3D space, under typical perspective camera model, one can assume $\mathbb{G}$ to be either the 2D affine group $\mathsf{Aff}(2, \mathbb{R})$, or the homography group $\mathsf{GL}(3, \mathbb{R})$ acting linearly on the image domain. Nevertheless, in principle, the formulation also works for more general classes of domain deformations or camera projection models as long as they can be modeled well by a finite-dimensional parametric group [ZMM11, ZLM11]. We will see a few concrete examples soon in Section 15.5.

Corrupted Low-rank Textures.
In addition to domain transformations, the observed image of the texture might be corrupted by pixel noise and occlusion. As before, we can model such nuisance by an error matrix $\boldsymbol{E}_o$ as follows:

$$\boldsymbol{I} = \boldsymbol{I}_o + \boldsymbol{E}_o.$$

As a result, the image $\boldsymbol{I}$ might no longer be a low-rank texture. In the low-rank texture framework, we assume that only a small fraction of the image pixels are corrupted by gross errors. Hence, $\boldsymbol{E}_o$ is generally a sparse matrix.

So the problem we are facing here is: *Given a possibly corrupted and deformed*

image of a low-rank texture: $\boldsymbol{I} = (\boldsymbol{I}_o + \boldsymbol{E}_o) \circ \tau^{-1}$*, can we recover both the intrinsic low-rank texture representation* $\boldsymbol{I}_o$ *and the domain transformation* $\tau \in \mathbb{G}$*?*

The answer to this problem is whether we are able to find solutions to the following optimization program:

$$\min_{\boldsymbol{L},\boldsymbol{E},\tau} \operatorname{rank}(\boldsymbol{L}) + \gamma \|\boldsymbol{E}\|_0 \quad \text{subject to} \quad \boldsymbol{I} \circ \tau = \boldsymbol{L} + \boldsymbol{E}. \tag{15.4.1}$$

That is, we aim to find the upright ground truth texture $\boldsymbol{I}_o$ as the lowest possible rank $\boldsymbol{L}_\star$ and the error $\boldsymbol{E}_\star$ with the fewest possible nonzero entries that agrees with the observation $\boldsymbol{I}$ up to a domain transformation τ. Here, $\gamma > 0$ is a weighting parameter that trades off the rank of the texture versus the sparsity of the error. For convenience, we refer to the so rectified solution $\boldsymbol{I}_o$ of the observed tilted pattern $\boldsymbol{I}$ as a *Transform Invariant Low-rank Texture* (TILT), coined by [ZLGM10, ZGLM12] .[10]

15.4.2 The TILT Algorithm

As we studied in previous chapters, the rank function and the ℓ^0-norm in the original problem (15.4.1) are extremely difficult to optimize, let alone with an unknown deformation τ. However, under fairly broad conditions, they can be replaced by their convex surrogates: the matrix nuclear norm $\|\boldsymbol{I}_o\|_*$ for $\operatorname{rank}(\boldsymbol{I}_o)$ and the ℓ^1-norm $\|\boldsymbol{E}\|_1$ for $\|\boldsymbol{E}\|_0$, respectively. Thus, we end up with the following optimization problem:

$$\min_{\boldsymbol{L},\boldsymbol{E},\tau} \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \boldsymbol{I} \circ \tau = \boldsymbol{L} + \boldsymbol{E}. \tag{15.4.2}$$

Dealing with Domain Deformation via Linearization.
Note that although the objective function in (15.4.2) is convex, the constraint $\boldsymbol{I} \circ \tau = \boldsymbol{L} + \boldsymbol{E}$ is nonlinear in $\tau \in \mathbb{G}$, and hence the overall problem is no longer convex. We have seen a similar problem in Section 5.5. As we have discussed there, we may overcome this difficulty by linearizing the constraint around the current estimate and iterate, which is a typical technique to deal with nonlinearity in mathematical programming [LK81, BM04]. If we approximate the nonlinear constraint up to its first order (with respect the deformation parameter τ), the constraint for the linearized version of the above program becomes

$$\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau \approx \boldsymbol{L} + \boldsymbol{E}, \tag{15.4.3}$$

[10] By a slight abuse of terminology, we also refer to the procedure of solving the optimization problem as TILT.

Algorithm 15.1 (The TILT Algorithm)

INPUT: Input image $\boldsymbol{I} \in \mathbb{R}^{w \times h}$, initial transformation $\tau \in \mathbb{G}$ (affine or projective), and a weight $\lambda > 0$.

WHILE not converged **DO**

Step 1: Normalization and compute Jacobian:

$$\boldsymbol{I} \circ \tau \leftarrow \frac{\boldsymbol{I} \circ \tau}{\|\boldsymbol{I} \circ \tau\|_F}; \quad \nabla \boldsymbol{I} \leftarrow \frac{\partial}{\partial \zeta} \left(\frac{\mathrm{vec}(\boldsymbol{I} \circ \zeta)}{\|\mathrm{vec}(\boldsymbol{I} \circ \zeta)\|_2} \right) \Big|_{\zeta = \tau};$$

Step 2 (inner loop): Solve the linearized problem:

$$\begin{aligned} (\boldsymbol{L}_\star, \boldsymbol{E}_\star, d\tau_\star) \quad \leftarrow \quad & \arg\min_{\boldsymbol{L}, \boldsymbol{E}, d\tau} \ \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{E}\|_1 \\ & \text{subject to} \quad \boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{L} + \boldsymbol{E}; \end{aligned}$$

Step 3: Update the transformation: $\tau \leftarrow \tau + d\tau_\star$;

END WHILE

OUTPUT: Converged solution $\boldsymbol{L}_\star$, $\boldsymbol{E}_\star$, $\tau_\star$ to problem (15.4.2).

where $\nabla \boldsymbol{I}$ is the Jacobian (derivatives of the image with respect to the transformation parameters in τ).[11] The optimization problem in (15.4.2) reduces to

$$\min_{\boldsymbol{L}, \boldsymbol{E}, d\tau} \ \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{L} + \boldsymbol{E}. \tag{15.4.4}$$

The linearized problem above is a convex program as the constraint is linear in all unknowns $\boldsymbol{L}, \boldsymbol{E}, d\tau$ hence it is amenable to efficient solution. One may use the algorithms introduced in Chapter 8 to solve the above convex program.

Since the linearization is only a local approximation to the original nonlinear problem, we solve it iteratively in order to converge to a (local) minimum of the original non-convex problem (15.4.2). The resulting optimization scheme is summarized as Algorithm 15.1.

The iterative linearization scheme outlined above is a common technique in optimization to solve nonlinear problems. It can be shown that this kind of iterative linearization converges quadratically to a local minimum of the original non-linear problem. A complete proof is out of the scope of this paper. We refer the interested reader to [Cro78, JO80] and the references therein.

Solving the Linearized Inner Loop Program.
To implement Algorithm 15.1 numerically, the most computationally expensive part is solving the inner loop convex program in Step 2. This can be cast as a semidefinite program and solved using conventional algorithms such as interior-point methods. However, as we discussed in Chapter 8, while interior-point methods have excellent convergence properties, they do not scale very well with the

[11] Strictly speaking, $\nabla \boldsymbol{I}$ is a 3D tensor: it gives a vector of derivatives at each pixel whose length is the number of parameters in the transformation τ. When we "multiply" $\nabla \boldsymbol{I}$ with another matrix or vector, it contracts in the obvious way which should be clear from the context.

problem size. As TILT is a very very useful tool for computer vision, we here derive in more details a fast implementation based on the augmented Lagrangian method (ALM) via *alternating direction method of multipliers* (ADMM), which was also covered in Chapter 8.

First, for the problem given in (15.4.4), its augmented Lagrangian is defined as:

$$\begin{aligned}\mathcal{L}_\mu(\boldsymbol{L},\boldsymbol{E},d\tau,\boldsymbol{Y}) \doteq \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{E}\|_1 + \langle \boldsymbol{Y}, \boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau - \boldsymbol{L} - \boldsymbol{E}\rangle \\ + \frac{\mu}{2}\|\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau - \boldsymbol{L} - \boldsymbol{E}\|_F^2,\end{aligned} \tag{15.4.5}$$

where $\mu > 0$, $\boldsymbol{Y}$ is a Lagrange multiplier matrix, $\langle\cdot,\cdot\rangle$ denotes the matrix inner product. To optimize the above augmented Lagrangian, the augmented Lagrangian method require to solve the following steps iteratively:

$$\begin{aligned}(\boldsymbol{L}_{k+1},\boldsymbol{E}_{k+1},d\tau_{k+1}) &= \arg\min_{\boldsymbol{L},\boldsymbol{E},d\tau} \mathcal{L}_{\mu_k}(\boldsymbol{L},\boldsymbol{E},d\tau,\boldsymbol{Y}_k),\\ \boldsymbol{Y}_{k+1} &= \boldsymbol{Y}_k + \mu_k(\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau_{k+1} - \boldsymbol{L}_{k+1} - \boldsymbol{E}_{k+1}).\end{aligned}$$

Throughout the rest of the paper, we will always assume that $\mu_k = \rho^k\,\mu_0$ for some $\mu_0 > 0$ and $\rho > 1$, unless otherwise specified.

We only have to solve the first step of the above iterative scheme. In general, it is computationally expensive to minimize over all the variables $\boldsymbol{L}$, $\boldsymbol{E}$ and $d\tau$ simultaneously. So, we adopt a common strategy to solve it *approximately* by adopting an *alternating minimizing*, strategy *i.e.* minimizing with respect to $\boldsymbol{L}$, $\boldsymbol{E}$ and $d\tau$ one at a time:

$$\begin{cases}\boldsymbol{L}_{k+1} &= \arg\min_{\boldsymbol{L}} \mathcal{L}_{\mu_k}(\boldsymbol{L},\boldsymbol{E}_k,d\tau_k,\boldsymbol{Y}_k),\\ \boldsymbol{E}_{k+1} &= \arg\min_{\boldsymbol{E}} \mathcal{L}_{\mu_k}(\boldsymbol{L}_{k+1},\boldsymbol{E},d\tau_k,\boldsymbol{Y}_k),\\ d\tau_{k+1} &= \arg\min_{d\tau} \mathcal{L}_{\mu_k}(\boldsymbol{L}_{k+1},\boldsymbol{E}_{k+1},d\tau,\boldsymbol{Y}_k).\end{cases} \tag{15.4.6}$$

Due to the special structure of our problem, each of the above optimization problems has a simple closed-form solution, and hence, can be solved in a single step. More precisely, recall the proximal operators for the ℓ^1 norm and the nuclear norm in Chapter 8, the solutions to (15.4.6) can be expressed explicitly using the soft-thresholding operator as follows:

$$\begin{cases}\boldsymbol{L}_{k+1} &\leftarrow \boldsymbol{U}_k\mathrm{soft}(\boldsymbol{\Sigma}_k,\mu_k^{-1})\boldsymbol{V}_k^*,\\ \boldsymbol{E}_{k+1} &\leftarrow \mathrm{soft}(\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau_k - \boldsymbol{L}_{k+1} + \mu_k^{-1}\boldsymbol{Y}_k, \lambda\mu_k^{-1}),\\ d\tau_{k+1} &\leftarrow (\nabla\boldsymbol{I})^\dagger(-\boldsymbol{I}\circ\tau + \boldsymbol{L}_{k+1} + \boldsymbol{E}_{k+1} - \mu_k^{-1}\boldsymbol{Y}_k),\end{cases} \tag{15.4.7}$$

where $\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^*$ is the SVD of $\big(\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau_k - \boldsymbol{E}_k + \mu_k^{-1}\boldsymbol{Y}_k\big)$, and $(\nabla\boldsymbol{I})^\dagger$ denotes the Moore-Penrose pseudo-inverse of $\nabla\boldsymbol{I}$.

We summarize the ADMM scheme for solving (15.4.4) as Algorithm 15.2. We note that the operations in each step of the algorithm are very simple with the SVD computation being the most computationally expensive step.[12]

[12] Empirically, we notice that for larger window sizes (over 100×100 pixels), it is much faster to run the partial SVD instead of the full SVD, if the rank of the texture is known to be very low.

Algorithm 15.2 (Inner Loop of TILT)

INPUT: The current (deformed and normalized) image $\boldsymbol{I} \circ \tau \in \mathbb{R}^{m \times n}$ and its Jacobian $\nabla \boldsymbol{I}$ against current deformation τ (from the outer loop), and $\lambda > 0$.

Initialization: $k = 0, \boldsymbol{Y}_0 = 0, \boldsymbol{E}_0 = 0, d\tau_0 = 0, \mu_0 > 0, \rho > 1$;

WHILE not converged **DO**

$(\boldsymbol{U}_k, \boldsymbol{\Sigma}_k, \boldsymbol{V}_k) = \mathrm{SVD}\big(\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_k - \boldsymbol{E}_k + \mu_k^{-1}\boldsymbol{Y}_k\big)$;

$\boldsymbol{L}_{k+1} = \boldsymbol{U}_k \mathrm{soft}(\boldsymbol{\Sigma}_k, \mu_k^{-1})\boldsymbol{V}_k^*$;

$\boldsymbol{E}_{k+1} = \mathrm{soft}(\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_k - \boldsymbol{L}_{k+1} + \mu_k^{-1}\boldsymbol{Y}_k, \lambda\mu_k^{-1})$;

$d\tau_{k+1} = (\nabla \boldsymbol{I})^\dagger(-\boldsymbol{I} \circ \tau + \boldsymbol{L}_{k+1} + \boldsymbol{E}_{k+1} - \mu_k^{-1}\boldsymbol{Y}_k)$;

$\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \mu_k(\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_{k+1} - \boldsymbol{L}_{k+1} - \boldsymbol{E}_{k+1})$;

$\mu_{k+1} = \rho\mu_k$;

END WHILE

OUTPUT: Converged solution $(\boldsymbol{L}_\star, \boldsymbol{E}_\star, d\tau_\star)$ to problem (15.4.4).

Connection to Compressive Principal Component Pursuit.
Notice that there is another way to view the linearized constraint (15.4.3):

$$\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{L} + \boldsymbol{E}. \tag{15.4.8}$$

Let Q be the left kernel of the Jacobian $\nabla \boldsymbol{I}$, that is $\mathcal{P}_{\mathsf{Q}}[\nabla \boldsymbol{I}] = 0$. Applying $\mathcal{P}_{\mathsf{Q}}[\cdot]$ to both sides of the equation, we obtain:

$$\mathcal{P}_{\mathsf{Q}}[\boldsymbol{I} \circ \tau] = \mathcal{P}_{\mathsf{Q}}[\boldsymbol{L} + \boldsymbol{E}]. \tag{15.4.9}$$

Then the program (15.4.4) becomes equivalent to:

$$\min_{\boldsymbol{L}, \boldsymbol{E}, d\tau} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \mathcal{P}_{\mathsf{Q}}[\boldsymbol{I} \circ \tau] = \mathcal{P}_{\mathsf{Q}}[\boldsymbol{L} + \boldsymbol{E}]. \tag{15.4.10}$$

Notice that this is exactly the compressive principal component pursuit problem (5.5.2) discussed in Chapter 5. However, Theorem 5.9 only provides recovery guarantee for random projection operator $\mathcal{P}_{\mathsf{Q}}[\cdot]$ but the above kernel projection is certainly not random. To our best knowledge, there is little result that characterizes how such projection is incoherent with the low-rank and sparse component so that correct recovery is guaranteed. Empirical results below with images show that that is obviously the case for typical types of transformations (e.g. 2D linear or affine transformation groups or 3D curved surfaces). Rigorous theoretical analysis of the interplay of group transformations and low-dimensional structures remains to be established. We will discuss more in the Notes as well as see more interplay between the two in the next Chapter.

Now putting all together, we see that the original problem (15.4.2) is essentially a nonlinear optimization problem that tries to recover both the low rank texture and its deformation. The TILT Algorithm 15.1 relies on linearizing the nonlinear constraint locally and then iteratively solves the locally linearized version using Algorithm 15.2. Hence, in general, there is no guarantee if the algorithm converges to the globally optimal (usually the correct) solution. As studies in

the literature [ZLGM10, ZGLM12] has shown, if the above algorithm is properly implemented, the range of convergence for typical deformations encountered in practice can be surprisingly large. For instance, for a typical checkerboard pattern tilted in front of a camera, the algorithm manages to converge correctly even if the tilting angle is around 50°! For details of implementing the TILT algorithm and a careful quantitative examination of the range of convergence for the TILT algorithm, the reader can refer to [ZLGM10, ZGLM12]. Again, a rigorous characterization of the global landscape of this nonlinear program and justification for the large range of contraction remains widely open.

15.5 Applications of TILT

The above algorithm is derived for τ being an arbitrary (parametric) transform in a prescribed group $\mathbb{G}$. In this section, we show how to apply the above algorithm to several typical types of transformations we often encounter in computer vision applications:

1 The low-rank texture is (approximately) on a planar surface and the camera is an ideal perspective projection. In this case, the deformation τ belongs to the group of general linear transforms on a plane (also known as the homography in the computer vision literature). The TILT algorithm allows us to recover the precise location and orientation of the plane in 3D relative to the camera.
2 The low-rank texture is on a generalized cylindrical surface. The TILT algorithm would allow use to recover both the 3D shape of the surface as well as its location and orientation relative to the camera.
3 The camera is not projective and its lens has certain nonlinear distortion. The images of a standard calibration rig (a planar checkerboard pattern) would allow us to recover the camera lens distortion.

15.5.1 Rectifying Planar Low-Rank Textures

If a low-rank texture $\boldsymbol{I}_o$ is on a planar surface, then at an arbitrary viewpoint, its image $\boldsymbol{I}$ (under ideal perspective projection) is related to the original (rectified) texture $\boldsymbol{I}_o$ by a homography τ [MSKS04], or formally known as a projective transformation. Figure 15.6 shows one such example. More precisely, let (u, v) be the coordinates of the image $\boldsymbol{I}$, and (x, y) be the coordinates of the original texture $\boldsymbol{I}_o$. If we represent both image planes with the homogeneous coordinates $[u, v, 1]^* \in \mathbb{R}^3$ and $[x, y, 1]^* \in \mathbb{R}^3$, respectively. Then the coordinates of a point on $\boldsymbol{I}_o$ and those of its (projective) image on $\boldsymbol{I}$ will be related by

$$\tau(x, y) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{15.5.1}$$

where "$\sim$" means equal up to a scale and $\boldsymbol{H} = [h_{ij}] \in \mathbb{R}^{3\times 3}$ is an invertible matrix belonging to the general linear group $\mathsf{GL}(3)$:

$$\mathsf{GL}(3) \doteq \left\{ \boldsymbol{H} \in \mathbb{R}^{3\times 3} \,\middle|\, \det(\boldsymbol{H}) \neq 0 \right\}. \tag{15.5.2}$$

However, there is a little caveat in this formulation. If we allow the transformation τ in the TILT algorithm to be free in the entire group $\mathsf{GL}(3)$, it could lead to a trivial solution in which the algorithm will choose a black region and a τ to blow it up to the size of $\boldsymbol{I}_o$ so that the value of the objective function will be nearly zero. To avoid such degenerate solutions, one way is to fix the scale of the region which we like to rectify. We may restrict the transform to be in a subgroup of $\mathsf{GL}(3)$ with scale normalized, known as the special linear group:

$$\mathsf{SL}(3) \doteq \left\{ \boldsymbol{H} \in \mathbb{R}^{3\times 3} \,\middle|\, \det(\boldsymbol{H}) = 1 \right\}. \tag{15.5.3}$$

This imposes additional (nonlinear) constraints among the parameters of the transformation.[13] In practical implementation of the TILT algorithm for the projective case, to fix the scale, we may simply specify and fix two diagonal corners of the region we would like to rectify. Interested readers may find more implementation details in [ZGLM12]. Figure 15.7 shows some of the representative results of the TILT algorithm applied to low-rank textures on a plane. Notice that the rectification is typically accurate to the pixel level.

15.5.2 Rectifying Generalized Cylindrical Surfaces

In man-made environments, structured objects do not always have planar surfaces. In many cases, the surface can be curved and cannot be approximated by a planar one, as the example in Figure 15.8 left shows. Clearly, if we approximate the surface by a plane outlined as the red window (adapted to the orientation of the facade), the texture will not be regular. Instead, the texture enclosed in the green window would be close to an ideal low-rank texture. However, to recover such low-rank texture, it would require us to recover the shape of the surface as well (in addition to the unknown camera projection in the planar case).

In this section, we see how TILT can be extended to rectify and recover low-rank texture on such curved surfaces in 3D space. Let us assume the image $\boldsymbol{I}(u, v)$ is a transformed version of low-rank texture $\boldsymbol{I}_o(x, y)$ wrapped on a curved surface C, as illustrated in Figure 15.8 right.

Presumably there exists a composite map from the intrinsic texture coordinate (x, y) to the image coordinate (u, v) as

$$g(x, y): \quad (x, y) \mapsto (u, v), \tag{15.5.4}$$

[13] A systematical way to handle any additional constraints on the parameters of the transformation τ is to linearize it and then add the additional linear constraints to the Inner Loop of the TILT algorithm.

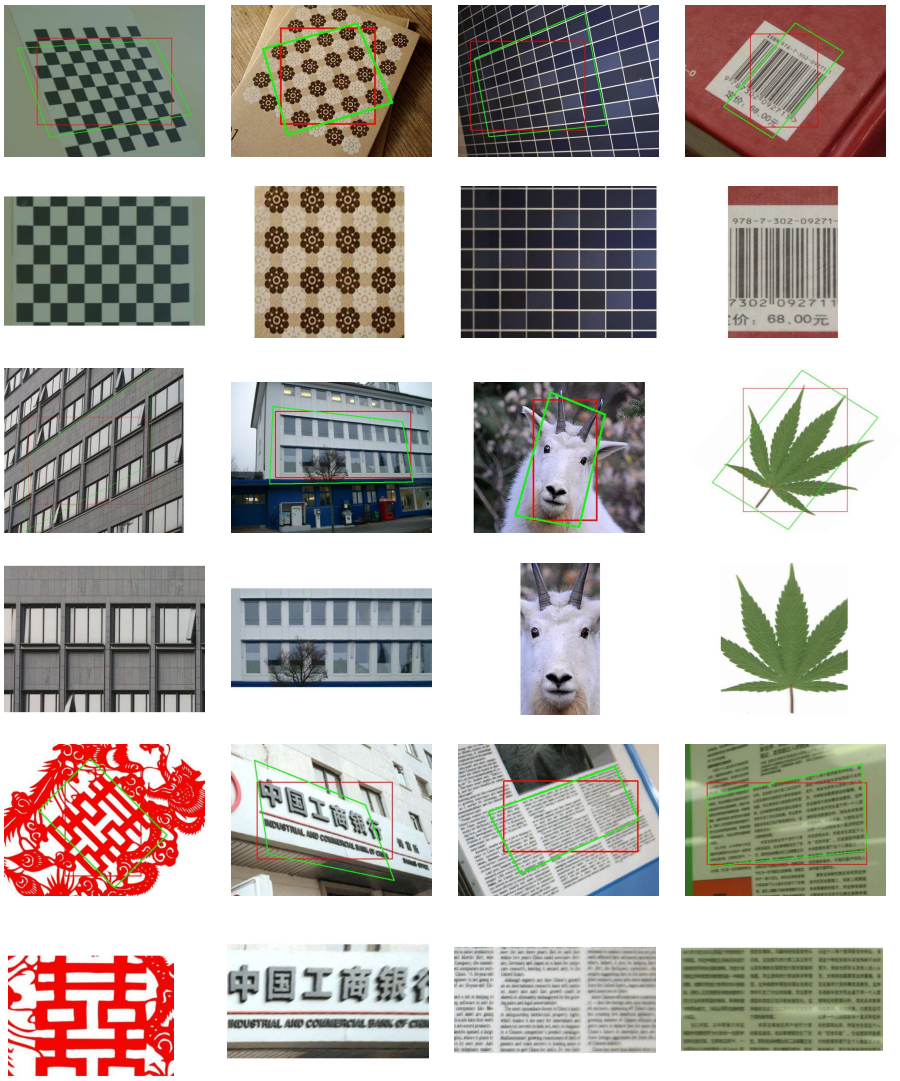


Figure 15.7 Representative Results of TILT on several categories of structured objects: textures with repetitive patterns, building facades, bar codes, characters and texts, bilateral symmetrical objects etc. In each case, the red window denotes the initial input of the TILT algorithm and the green window denotes the final converged output. The (matrix associated with the) green window is displayed to highlight the recovered low-rank texture.

such that $\boldsymbol{I} \circ g = \boldsymbol{I}_o$ in the noise-free case. In this section, we will explain how to parametrize such a transformation g based on a generalized cylindrical surface model for the surface [ZLM11]. The model represents a very important family of 3D shapes as they describe majority of curved building facades or deformed

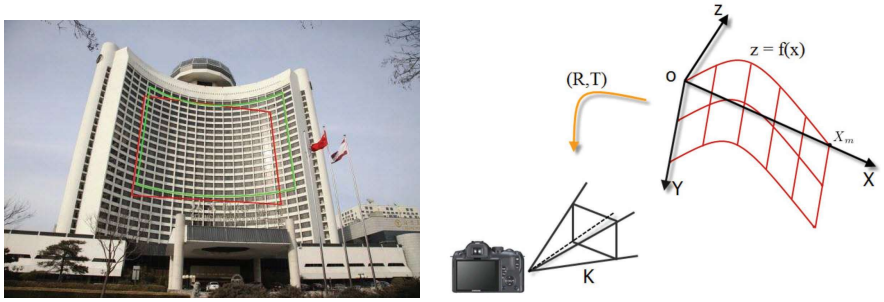


Figure 15.8 **Left:** An example of a curve building facade. Red window is a planar approximation to the surface; and green window outlines the true low-rank texture of the facade. **Right:** generalized cylindrical surface C viewed by a perspective camera $\boldsymbol{K}$.

texts on curved surfaces. Mathematically, a generalized cylindrical surface can be described as

$$\boldsymbol{c}(s,t) = t\boldsymbol{p} + \boldsymbol{h}(s) \quad \in \mathbb{R}^3, \tag{15.5.5}$$

where $s, t \in \mathbb{R}$, $\boldsymbol{p}, \boldsymbol{h}(s) \in \mathbb{R}^3$, and $\boldsymbol{p} \perp \partial\boldsymbol{h}(s)$.

Without loss of generality, we may choose a 3D coordinate frame (X, Y, Z) for the surface such that the center o is on the surface and the Y-axis aligns with the direction of $\boldsymbol{p}$. If we limit our calculation within a "rectangular" section of the surface whose X-coordinate is in the interval $[0, X_m]$, then the expression of the function $\boldsymbol{h}(\cdot)$ can be simplified and uniquely determined by a scalar function $Z = f(X)$, as shown in Figure 15.8 right.

Without loss of generality, we may choose to parametrize the function $Z = f(X)$ by a polynomial up to degree $d+2$, where typically $d \leq 4$ for most natural images. So an explicit expression of the surface can be written as:

$$Z = f_c(X) \doteq X(X - X_m)\sum_{i=0}^{d} a_i X^i, \tag{15.5.6}$$

where we use $\boldsymbol{c} \doteq \{a_0, a_1, \ldots, a_d\}$ to denote the collection of surface parameters. Further note that when all a_i's are zero, the surface reduces to a planar surface $Z = 0$ in 3D as considered in the previous section.

For any point (x, y) in the rectified and flattened texture coordinates, we need to find its 3D coordinates (X_c, Y_c, Z_c) on the cylindrical surface C. We can calculate the geodesic distance from the origin O to $(X_m, 0, 0)$ on the surface as

$$L_c \doteq \int_0^{X_m} \sqrt{1 + f_c'(X)^2}\, dX. \tag{15.5.7}$$

The the following set of equations uniquely determine the wrapping map between

(x, y) and (X_c, Y_c, Z_c):

$$\begin{cases} x &= \frac{X_m}{L_c} \int_0^{X_c} \sqrt{1 + f_c'(X)^2} dX, \\ y &= Y_c, \\ Z_c &= f_c(X_c). \end{cases} \tag{15.5.8}$$

Finally, suppose we are given a perspective camera model with intrinsic parameters $\boldsymbol{K} = \begin{bmatrix} f_x & \alpha & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$,[14] and its relative position with respect to the surface coordinate frame is described by an unknown Euclidean transform $(\boldsymbol{R}, \boldsymbol{T}) \in \mathsf{SE}(3, \mathbb{R})$, where $\boldsymbol{R} \in \mathbb{R}^{3\times 3}$ is a rotation matrix and $\boldsymbol{T} \in \mathbb{R}^3$ is a translation vector. Then a point with 3D coordinates (X_c, Y_c, Z_c) is mapped to the image pixel coordinates (u, v) according to the following relationship:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \frac{R_{11}X_c + R_{12}Y_c + R_{13}Z_c + T_1}{R_{31}X_c + R_{32}Y_c + R_{33}Z_c + T_3} \\ \frac{R_{21}X_c + R_{22}Y_c + R_{23}Z_c + T_2}{R_{31}X_c + R_{32}Y_c + R_{33}Z_c + T_3} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} f_x x_n + \alpha y_n + o_x \\ f_y y_n + o_y \\ 1 \end{bmatrix}. \tag{15.5.9}$$

Therefore the transformation g from the texture coordinates (x, y) to the image coordinates (u, v) is a composition of the following mappings specified above:

$$g : (x, y) \mapsto (X_c, Y_c, Z_c) \mapsto (x_n, y_n) \mapsto (u, v). \tag{15.5.10}$$

The parameters needed to specify g include the parameters for the surface $\boldsymbol{c}$, the camera pose $(\boldsymbol{R}, \boldsymbol{T})$ (and the camera intrinsic parameters $\boldsymbol{K}$ if unknown). Although here the deformation group $\mathbb{G}$ is not explicitly defined, the so defined mappings g are all one-to-one and invertible. For our purposes, it suffices if such transformations are defined in a range around the identity mapping.[15]

In order to recover the low-rank component $\boldsymbol{I}_o$ subject to some possible sparse error component $\boldsymbol{E}_o$, we can now solve the following optimization problem:

$$\min_{\boldsymbol{L}, \boldsymbol{E}, \boldsymbol{c}, \boldsymbol{R}, \boldsymbol{T}} \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \boldsymbol{I} \circ g = \boldsymbol{L} + \boldsymbol{E}. \tag{15.5.11}$$

As we have done in the TILT algorithm, this nonlinear problem can be estimated *iteratively* by solving its linearized version as:

$$\min_{\boldsymbol{L}, \boldsymbol{E}, dg} \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{subject to} \quad \boldsymbol{I} \circ g + \nabla \boldsymbol{I}_g \cdot dg = \boldsymbol{L} + \boldsymbol{E}, \tag{15.5.12}$$

where $\nabla \boldsymbol{I}_g$ is the Jacobian matrix of the image with respect to both the unknown general cylindrical surface parameters $\boldsymbol{c}$ and the unknown Euclidean transform $(\boldsymbol{R}, \boldsymbol{T})$ (and the camera calibration $\boldsymbol{K}$ if unknown too), and dg is the differential of these unknown variables. Notice that the above program is exactly the same at the program that we have solved for the TILT problem in the previous Section. The only difference is the deformation τ (or $d\tau$) is replaced by g (or dg) here.

[14] Be aware that here f_x, f_y stand for focus length and are not to be confused with the curve function f_c above. In practice, the intrinsic parameters can be well approximated from the EXIF information in the image file recorded by modern digital cameras. For more details, please refer to [ZLM11].

[15] Strictly speaking, such set of transformations form a groupoid.

We can use the same Algorithm 15.1 to solve the above program. For a more detailed implementation, please refer to [ZLM11]. Figure 15.9 shows some real examples and results of curved low-rank textures unwrapped and recovered by the TILT Algorithm 15.1 with the transformation g described above.

15.5.3 Calibrating Camera Lens Distortion

In the above planar and curved surface cases, we have always assumed that the image of a low-rank texture is taken by a (calibrated) camera which can be represented by an ideal perspective projection. However, with the proliferation of low-cost cameras, many cameras (and their images) we encounter in practice are not carefully calibrated by the manufacturer; and some cameras are deliberated made with different projection models from the perspective one (in order to maximize the use of the imaging sensors and increase the field of view), such as the fisheye cameras or omnidirectional cameras. Figure 15.10 left shows an example of an image taken by a fisheye camera. Figure 15.10 right shows an image of the same building by an ideal perspective camera. Notice that for the later case, there is no distortion caused by the lens and all straight lines in the scene are straight in the image.

Camera calibration is arguably the most crucial task for any applications that requires the computer or machine to perceive and interact with the 3D world through a camera (such as 3D reconstruction, mapping, navigation, and manipulation etc.) When calibrating a camera (with respect to certain world coordinate frame), in addition to the aforementioned *lens distortion*, we also need to calibrate the *intrinsic parameters* $\boldsymbol{K}$ for its perspective projection and the *extrinsic parameters* $(\boldsymbol{R}, \boldsymbol{T})$ for the camera viewpoint. For conventional calibration methods such as the popular toolbox [Bou], one needs to take a few images of a calibration rig (usually a planar checkerboard pattern with known geometry [Zha00]). The corners of the checkerboard then need to be carefully marked out for calibration. Figure 15.11 shows an example.

As we see, the calibration rig is typically a regular pattern hence is low-rank. The imaging process of an uncalibrated camera is a sequence of mappings that transform the low-rank texture to the image plane. Instead of marking the corners manually which is tedious and time-consuming,[16] in principle we could utilize the low-rankness of the calibration pattern and use the TILT algorithm to automatically estimate all unknown parameters associated with the imaging process.

Similar to the previous section, we here give a brief description of the sequence of mappings involved in the imaging process of an uncalibrated camera, which

[16] Automatically detecting the corner features is a seemingly simple but remains a difficult problem that is not yet entirely solved under general conditions.

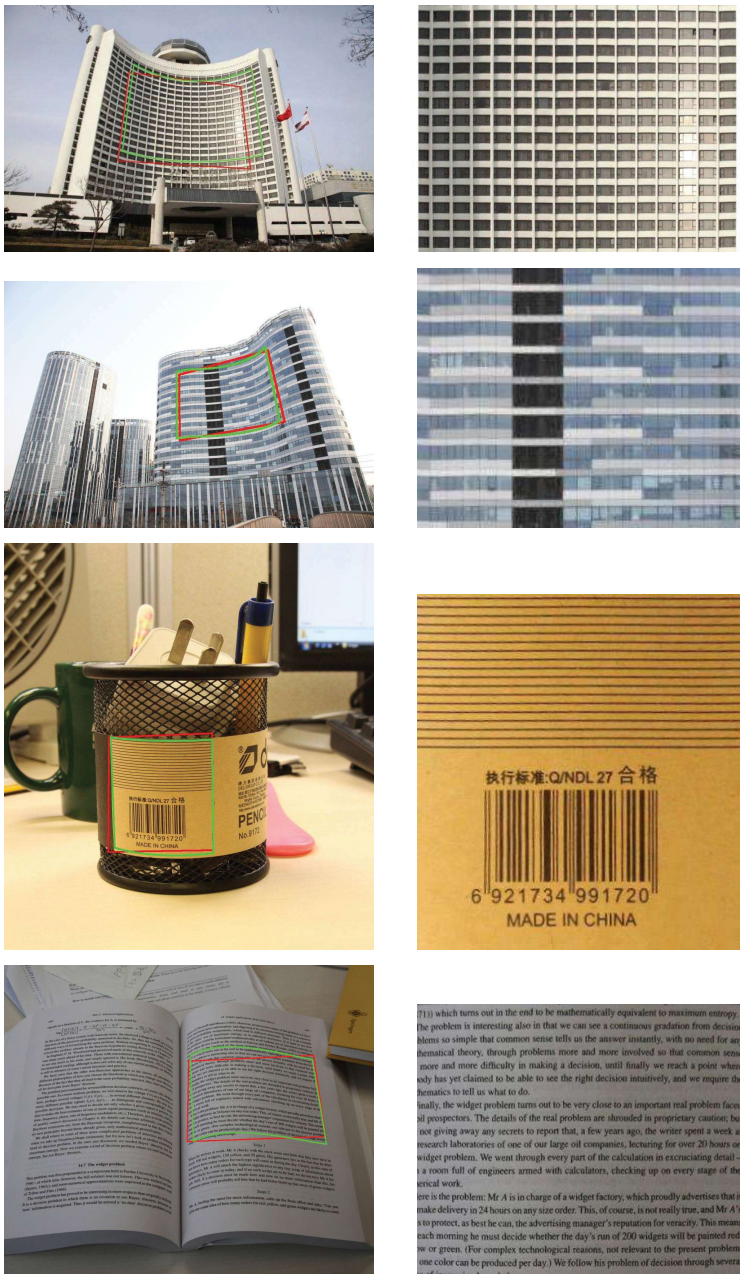


Figure 15.9 Unwrapping low-rank texture on curved surfaces based on a generalized cylindrical surface model. **Left:** Input image. The red bounding box indicates the manually labeled initial position of the texture region. The green bounding box indicates the recovered texture surface. **Right:** Unwrapped low-rank texture.

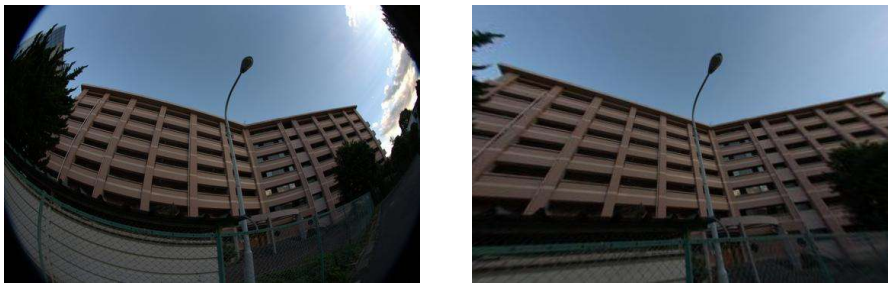


Figure 15.10 **Left:** typical image of a fisheye camera. **Right:** image of a perspective camera.

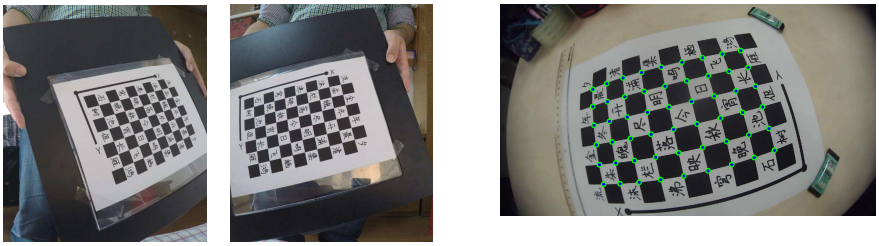


Figure 15.11 **Left two:** Images of a typical calibration rig. **Right:** Corners need to be marked (or detected) for conventional calibration methods.

can then be used in customizing the TILT algorithm for calibration purposes. For a more careful and complete description of camera models, the reader may refer to [HZ00, MSKS04].

We first briefly describe the common mathematical model used for camera calibration and introduce notation used in this paper. We use a vector $\boldsymbol{P} = [X_0, Y_0, Z_0]^* \in \mathbb{R}^3$ to denote the 3D coordinates of a point in the world coordinate frame, use $\boldsymbol{p}_n = [x_n, y_n]^* \in \mathbb{R}^2$ to denote its projection on the canonical image plane in the camera coordinate frame. For convenience, we denote the homogeneous coordinates of a point $\boldsymbol{p}$ as $\tilde{\boldsymbol{p}} = \left[\begin{smallmatrix}\boldsymbol{p}\\1\end{smallmatrix}\right] \in \mathbb{R}^3$.

As before, we use $\boldsymbol{R} \in \mathsf{SO}(3)$ and $\boldsymbol{T} \in \mathbb{R}^3$ to denote the rotation and translation from the world coordinate frame to the camera frame – so-called extrinsic parameters.[17] So we have

$$\tilde{\boldsymbol{p}}_n \sim \boldsymbol{R}\boldsymbol{P} + \boldsymbol{T} \quad \in \mathbb{R}^3.$$

If the lens of the camera is distorted, on the image plane, the coordinates of a point $\boldsymbol{p}_n$ may be transformed to a different one, denoted as $\boldsymbol{p}_d = [x_d, y_d]^* \in \mathbb{R}^2$. A very commonly used general mathematical model for this distortion $D(\cdot) : \boldsymbol{p}_n \mapsto$

[17] As in [MSKS04], the rotation $\boldsymbol{R}$ can be parameterized by a vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^* \in \mathbb{R}^3$ using the Rodrigues formula: $\boldsymbol{R}(\boldsymbol{\omega}) = \boldsymbol{I} + \sin\|\boldsymbol{\omega}\|\frac{\hat{\boldsymbol{\omega}}}{\|\boldsymbol{\omega}\|} + (1 - \cos\|\boldsymbol{\omega}\|)\frac{\hat{\boldsymbol{\omega}}^2}{\|\boldsymbol{\omega}\|^2}$, where $\hat{\boldsymbol{\omega}}$ denotes the 3×3 matrix form of the rotation vector $\boldsymbol{\omega}$, defined as $\hat{\boldsymbol{\omega}} = [0, -\omega_3, \omega_2; \omega_3, 0, -\omega_1; -\omega_2, \omega_1, 0] \in \mathbb{R}^{3\times 3}$.

$\boldsymbol{p}_d$ is given by a polynomial distortion model by neglecting any higher-order terms as below [Bro71]:

$$\begin{cases} r &\doteq \sqrt{x_n^2 + y_n^2}, \\ f(r) &\doteq 1 + k_c(1)r^2 + k_c(2)r^4 + k_c(5)r^6, \\ \boldsymbol{p}_d &= \begin{bmatrix} f(r)x_n + 2k_c(3)x_n y_n + k_c(4)(r^2 + 2x_n^2) \\ f(r)x_n + 2k_c(4)x_n y_n + k_c(3)(r^2 + 2y_n^2) \end{bmatrix}. \end{cases} \tag{15.5.13}$$

Notice that this model has a total of five unknowns $k_c(1), \ldots, k_c(5) \in \mathbb{R}$. If there is no distortion, simply set all $k_c(i)$ to be zero, and then it becomes $\boldsymbol{p}_d = \boldsymbol{p}_n$.

The intrinsic matrix $\boldsymbol{K} \in \mathbb{R}^{3\times 3}$ represents a linear transformation of points on the image plane to their pixel coordinates, denoted as $\boldsymbol{p} = [u, v]^* \in \mathbb{R}^2$. $\boldsymbol{K}$ also has five unknowns; the focal length along x and y-axes f_x and f_y, skew parameter θ, and coordinates of the principle point (o_x, o_y). In the matrix form, it is described as

$$\boldsymbol{K} \doteq \begin{bmatrix} f_x & \theta & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad \in \mathbb{R}^{3\times 3}. \tag{15.5.14}$$

With all the notation, the overall imaging process of a point $\boldsymbol{P}$ in the world to the camera pixel coordinates $\boldsymbol{p}$ by a pinhole camera can be described as:

$$\tilde{\boldsymbol{p}} = \boldsymbol{K}\tilde{\boldsymbol{p}}_d = \boldsymbol{K} \circ D(\tilde{\boldsymbol{p}}_n); \quad \lambda \tilde{\boldsymbol{p}}_n = \boldsymbol{R}\boldsymbol{P} + \boldsymbol{T}, \tag{15.5.15}$$

where λ is the depth of the point. If there is no lens distortion ($\tilde{\boldsymbol{p}}_d = \tilde{\boldsymbol{p}}_n$), the above model reduces the typical perspective projection with an uncalibrated camera: $\lambda\tilde{\boldsymbol{p}} = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{T})$.

For compact presentation, we will let τ_0 denote the intrinsic parameters and lens distortion parameters all together. When we take multiple, say N, images to calibrate the intrinsic parameters and the lens distortion, we use τ_i $(i = 1, 2, \ldots, N)$ to denote the extrinsic parameters $\boldsymbol{R}_i$ and $\boldsymbol{T}_i$ for the i-th image. By a slight abuse of notation, we will occasionally use τ_0 to represent the combined transformation of $\boldsymbol{K}$ and $D(\cdot)$ acting on the image domain, i.e., $\tau_0(\cdot) = \boldsymbol{K} \circ D(\cdot)$, and use τ_i $(i = 1, 2, \ldots, N)$ to represent the transforms from the world frame to each individual image plane. Using this notation, each image $\boldsymbol{I}_i$ and the calibration rig (low-rank texture) $\boldsymbol{I}_o$ are related by:

$$\boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{I}_o + \boldsymbol{E}_i, \quad i = 1, 2, \ldots, N, \tag{15.5.16}$$

where we use a sparse error term $\boldsymbol{E}_i$ to model possible occlusion or corruption introduced in the imaging process.

It seems that we now can use TILT to estimate all the transformation parameters in τ_0 and τ_i without using any marked feature points. However, there is one caveat: as we have discussed in the beginning of the chapter in Section 15.2, there is some ambiguity in the notion of a low-rank texture, a scaled or translated version of the same texture would have the same rank. Hence, if we use TILT to each individual image $\boldsymbol{I}_i$, the so recovered $\hat{\boldsymbol{I}}_o$ might be scaled or

translated version to one another. More precisely, if we solve the following robust rank-minimization problems individually:

$$\min \|\boldsymbol{L}_i\|_* + \lambda\|\boldsymbol{E}_i\|_1, \quad \text{subject to} \quad \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i, \tag{15.5.17}$$

with $\boldsymbol{L}_i, \boldsymbol{E}_i, \tau_i$ and τ_0 as unknowns. Then we can only expect $\boldsymbol{L}_\star$ to recovers the low-rank pattern $\boldsymbol{I}_o$ up to a translation and scaling in each axis, i.e.,

$$\boldsymbol{L}_\star = \boldsymbol{I}_o \circ \tau, \quad \text{for some} \quad \tau = \begin{bmatrix} a & 0 & t_1 \\ 0 & b & t_2 \\ 0 & 0 & 1 \end{bmatrix}. \tag{15.5.18}$$

Each of the N programs provide its own estimate of the interested τ_0.

There are many possible ways we can use all the N images together and try to estimate a common solution for τ_0 and $\boldsymbol{I}_o$. The most straightforward way is to lump all the objective functions together and enforce that the recovered $\boldsymbol{L}_i$ are all the same: Therefore, the natural optimization problem associated with this problem becomes

$$\begin{aligned} &\min \sum_{i=1}^{N} \|\boldsymbol{L}_i\|_* + \|\boldsymbol{E}_i\|_1, \\ &\text{subject to} \quad \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i,\ \boldsymbol{L}_i = \boldsymbol{L}_j. \end{aligned} \tag{15.5.19}$$

One can use optimization techniques similar to that of TILT to solve the above optimization problem, such as ALM and ADMM. However, having too many constraining terms affects the convergence of such algorithms.

To relax the equality constraints $\boldsymbol{L}_i = \boldsymbol{L}_j$, we may only need to require they are correlated. So an alternative is to concatenate all the recovered low-rank textures as submatrices of a joint low-rank matrix:

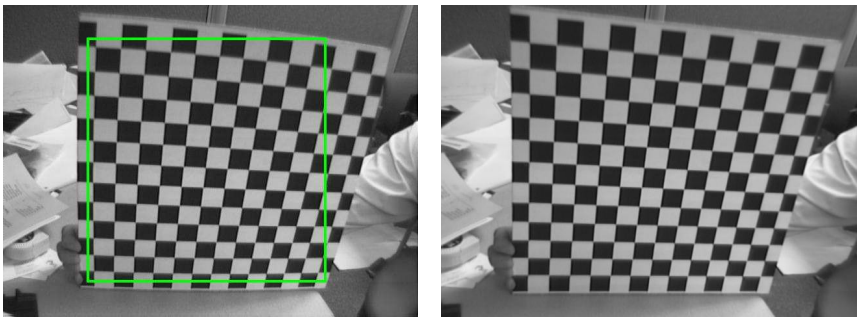
$$\boldsymbol{L}_c \doteq [\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_N], \quad \boldsymbol{L}_r \doteq [\boldsymbol{L}_1^*, \boldsymbol{L}_2^*, \ldots, \boldsymbol{L}_N^*], \quad \boldsymbol{E} \doteq [\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_N].$$

Then we try to simultaneously align the columns and rows of $\boldsymbol{L}_i$ by minimizing the ranks of $\boldsymbol{L}_c$ and $\boldsymbol{L}_r$:

$$\begin{aligned} &\min\ \|\boldsymbol{L}_c\|_* + \|\boldsymbol{L}_r\|_* + \lambda\|\boldsymbol{E}\|_1, \\ &\text{subject to} \quad \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i, \end{aligned} \tag{15.5.20}$$

with $\boldsymbol{L}_i, \boldsymbol{E}_i, \tau_0, \tau_i$ as unknowns. Notice that, by comparing to equation (15.5.19), the new optimization program has just half number of constraints and hence is easier to solve. In addition, it is insensitive to illumination and contrast change across different images as it does not require the recovered the images $\boldsymbol{L}_i$ to be equal in values.

Remark 15.3 (High-order Low-rank Tensors). *In fact, one may view the stack of images $\boldsymbol{L}_c = [\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_N]$ as a three-dimensional tensor. When calibrated correctly, this tensor is supposed to be highly structured: not only is each slice $\boldsymbol{L}_i$ a low-rank matrix, but also all slides are highly correlated. As we have discussed in Section 6.3 of Chapter 6, the above convex relaxation of $\boldsymbol{L}_c$ is only for one Tucker*



(a) Input image with an initial window (b) Lens distortion removed

Figure 15.12 Calibration from a single image in the Toolbox [Bou].

rank of this tensor. Based on our study there, from the compressive sensing perspective this relaxation is not optimal. Nevertheless, here for higher calibration accuracy, we actually desire higher resolution of the images. The computational cost is usually not a main concern as the calibration process is typically done offline.

It can be shown (see [ZMM11]) that under general conditions, when the number of images $N \geq 5$, then the optimal solution to the above program will be unique and

$$\tau_{0\star} = \tau_0, \quad \boldsymbol{K}_\star = \boldsymbol{K}, \quad \boldsymbol{R}_{i\star} = \boldsymbol{R}_i, \quad i = 1, \ldots, N.$$

In practice, the method actually works with as few as a single ($N = 1$) image (of low-rank texture). To calibrate the camera from a single image, we have to work with fairly strong assumptions, say that the principal point (o_x, o_y) is known (and simply set as the center of the image) and the pixel is square $f_x = f_y$. Then from the image, one can calibrate the focal length as well as eliminating the lens distortion k_c's. Figure 15.12 shows an example with an image given in the standard toolbox. As we see in Figure 15.12(b), the radial distortion is completely removed by the TILT algorithm.

Notice that the low-rank texture based calibration method does not require precise geometry about the calibration rig. Hence one does not have to use a checkerboard pattern and in principle can utilize any low-rank structures (such as a building facade) for calibration purposes. Figure 15.13 shows two examples of using the TILT algorithm to estimate and correct the radial lens distortion of a fisheye camera (from a single image).

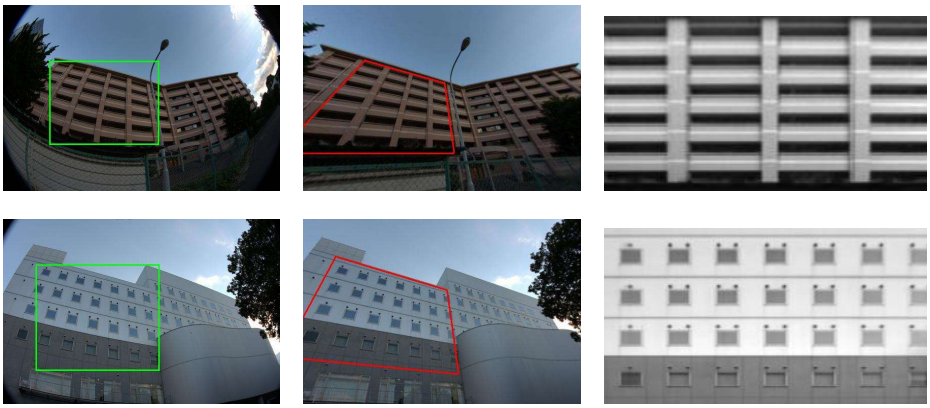


Figure 15.13 Rectify fisheye images with significant lens distortion. **Left:** input images with selected initialization windows (green); **Middle:** lens distortion removed images with final converged windows (red); **Right:** rectified low-rank textures.

15.6 Notes

The story presented in this chapter follows from the original work of [ZLGM10, ZGLM12] on the TILT method and its extensions to curved surface [ZLM11], camera calibration [ZMM11], and texture inpainting [LRZM12]. There are many other extensions that we give a brief account below.

3D Vision in Structure Scenes.
Man-made environments are rich of objects with structured shapes and textures. TILT provides a useful tool to harness one important type of holistic structure in a scene for 3D reconstruction purposes. It enables us to process and extract geometric information that is accurately encoded in large regions of the image, instead of local primitives such as corner-like or edge-like features which are used in conventional 3D reconstruction methods. Successfully harnessing holistic structures seems to be the key to future more accurate and robust 3D reconstruction methods. One may refer to [MZYM11] for some early promising results.

Learning to Detect Structures.
Currently, the TILT method requires to know the general location of the structured region in the image. To automatically initialize TILT with a region of interest is essentially a detection or recognition problem. To this end, one can develop effective low-rank texture detectors with learning-based methods, similar to learning to detect other holistic structures such as wireframes [ZQZ+19,ZQM19], vanishing points [ZQHM19], 2D planes [LKG+19], and 3D symmetry [ZLM20].

Alignment of Multiple Correlated Images.
In the same spirit of TILT, transformed sparse or low-rank models have also been explored and utilized in the work for sparsity-based robust face alignment and recognition [WWG+09, WWG+12] and for low-rank based robust multiple-image alignment method RASL [PGW+12]. As we have discussed in Section 6.3 of Chapter 6, when a matrix is simultaneously low-rank and sparse or multiple aligned images form a three-dimensional tensor, the convex relaxation approach may not be the optimal choice. Hence it would be very interesting to investigate whether certain nonconvex formulations can lead to even better solutions for these tasks.

Extension to Multiple Nonlinear Low-dimensional Structures.
In this chapter, we see how to recover primarily one low-dimensional structure under certain nonlinear transformation. We see the convex nuclear norm and ℓ^1 norm are rather effective in promoting the desired low-dimensional property in the solution. In the next and final chapter of the book, we will see how to recover a mixture of multiple low-dimensional structures under certain nonlinear transformation. In that more challenging context, we will resort to a more accurate, but nonconvex, measure for compactness: the lossy coding length in terms of the "$\log\det(\cdot)$" function (as we have seen in Exercise 7.4 of Chapter 7).

Low-dimensional Structures and Group Equivariance and Invariance.
From the work of transformed low-dimensional structures, one may observe a common phenomenon: a class of deformations $\mathbb{G}$ can be correctly recovered from a deformed low-dimensional structure as long as the deformations (or their infinitesimal actions) are "incoherent" with the low-dimensional structure. That is, the Jacobian $\nabla_\tau \boldsymbol{I}$ with respect to a deformation $\tau \in \mathbb{G}$ needs to be "incoherent" to the low-dimensional structure of $\boldsymbol{I}$. The precise conditions that would guarantee (at least local) correctness and success of the recovered deformation merit a more thorough examination in the future. Methods like TILT [ZLGM10, ZGLM12], RASL [PGW+12], and face alignment [WWG+12] provide compelling empirical evidences that low-dimensional structures in the data are the key to ensure true "equivariance," with respect to any group of transformations of interest.

In the next Chapter, we will encounter yet again another interaction between group invariance and low-dimensional structures. In particular, we will see why sparsity is actually necessary in order to ensure that classification tasks (such as face recognition) can be truly "invariant" to certain group transformations such as translation. Both work in this chapter and that in the next suggest it is extremely important to understand the relationship (or tradeoff) between low-dimensional structures and group transformations. Our current understanding (and results) remain rather limited and this is definitely a promising new direction for future study.

16 Deep Networks for Classification

"*What I cannot create, I do not understand.*"
– Richard Feynman

16.1 Introduction

In the past decade or so, (deep) neural networks have captured people's imagination through their empirical success in learning problems involving real-world high-dimensional data such as images, speech, and text [LBH15]. Nevertheless, there is quite a bit of mystery as to how deep networks achieve such striking results. Modern deep networks are typically designed through trial and error. Then they are typically trained and deployed as "black boxes" which implement desired input-output relationships, but whose inner workings are unclear. As a consequence, it is not easy to rigorously guarantee the performance of a trained network, such as being truly invariant to transformation [AW18,ETT+17] or not overfitting noisy or arbitrarily assigned class labels [ZBH+17].

In the final chapter of this book, we establish fundamental connections between the practice of deep neural networks and the theory for low-dimensional structures developed in this book. Hence, mathematical concepts, principles, and methods developed in this book can help us to understand, interpret, and even improve the practice of deep learning, or learning from high-dimensional data in general. As this is still an active research field, we will provide only an overview of one promising framework, which approaches the *data classification* problem[1] from the perspective of data compression and discriminative representation.

As we have seen in the previous chapter, low-dimensional structures of real-world data often are not linear (low-rank) nor piecewise linear (sparse). The structure can be deformed by certain nonlinear transform. For a classification task then, the (mixed) data from all classes typically lie on *multiple* nonlinear low-dimensional structures (or distributions), one per class. In this chapter, we will see how a few key ingredients that we have introduced and studied in this

[1] Image classification is where deep learning demonstrated the initial success that has catalyzed the recent explosion of interest in these models and techniques [KSH12]. Although we will focus on classification only in this chapter, the basic ideas and same principles can be naturally generalized to the case of regression.

book, including measures that promote low dimensionality, gradient schemes for optimization, sparsifying dictionaries, and convolutions for shift invariance, can be naturally integrated to learn a discriminative linear representation for such mixed low-dimensional data. Deep networks most naturally arise in this context as an optimization scheme to achieve this objective. In particular, as we will see they can be constructed as "white boxes" from *first principles*.

In the remainder of this section, we give a brief introduction to deep networks. In Section 16.2, we introduce a measure of low-dimensionality, namely coding *rate reduction*, as a principled objective for learning a discriminative and informative representations for classification. In Section 16.3, we show how a basic iterative gradient scheme to optimize this objective naturally leads to a typical deep network, entirely as a "white box." All modern deep networks (for classification) share the same characteristics of its architecture. If we further enforce the classification to be invariant to shift or translation, the network naturally becomes a deep convolutional network. In Section 16.4, we use a basic problem of classifying data lying on one-dimensional (nonlinear) submanifolds to illustrate why a deep network, of tractable size, can provide rigorous guarantees for correct classification under proper conditions. The network's width and depth can be naturally interpreted as statistical and computational resources, respectively, similar to those needed in a compressive sensing scheme for low-dimensional models.

So at high levels, one may view that Section 16.3 justifies the *necessity* of deep network architectures, from the perspective of optimizing a principled objective; while Section 16.4 characterizes *sufficient conditions* on when such a deep network provides tractable guarantees for the given classification task, if additional fine-tuning by back propagation is conducted. Finally in Section 16.5, we lay out some exciting *open problems* that emerge from interpreting deep networks from the perspective of *learning low-dimensional models via iterative optimization.*

16.1.1 Deep Learning in a Nutshell

It is arguably easiest to illustrate deep learning through the task of classification. The typical setting is: we are given labelled samples $\{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^m, \boldsymbol{y}^m)\}$, where the $\boldsymbol{x}^i$ are drawn from a mixture of k distributions $\mathcal{D} = \{\mathcal{D}_j\}_{j=1}^k$, and the $\boldsymbol{y}^i$ indicate which mixture component generated each observation $\boldsymbol{x}^i$. Here, we assume that the class labels $\boldsymbol{y}^i \in \mathbb{R}^k$ are encoded in "one-hot" format:[2]:

$$\boldsymbol{y}^i = [0, \ldots, 0, \underbrace{1}_{j\text{-th entry}}, 0, \ldots, 0]^* \quad \in \mathbb{R}^k.$$

[2] In a more general interpretation of the label information, one may use the k-dimensional vector $\boldsymbol{y}^i$ to indicate the probability of a sample $\boldsymbol{x}^i$ belonging to each of the k classes. Hence each entry of $\boldsymbol{y}$ can be a continuous number in $[0, 1]$ and all entries sum to 1. In the case of regression, when one tries to approximate a continuous function (defined on each of the classes), we may also allow the value of the corresponding entry to be a continuous number.

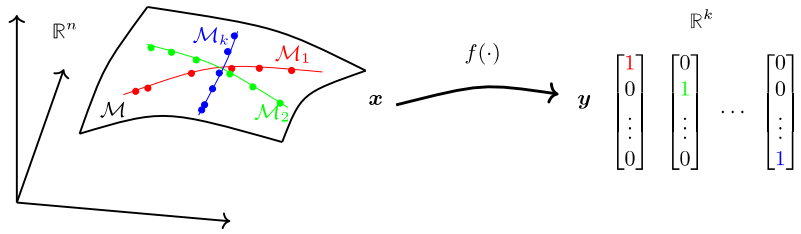


Figure 16.1 Classification as Sparse Representation: High-dimensional data $\boldsymbol{x} \in \mathbb{R}^n$ which lie on a mixture of low-dimensional submanifolds $\mathcal{M}_j$ within a manifold $\mathcal{M}$. $\boldsymbol{y}$ is the class label of $\boldsymbol{x}$ represented as a one-hot vector in $\mathbb{R}^k$. The goal is to learn a nonlinear mapping $f(\cdot) : \boldsymbol{x} \mapsto \boldsymbol{y}$.

Notice that although the number of classes k may be large, the vector $\boldsymbol{y}^i$ is always one-sparse.[3] For the task of classification, the goal of (deep) learning is to solve the inverse problem of mapping the input $\boldsymbol{x} \in \mathbb{R}^n$ to its (sparse) label vector $\boldsymbol{y} \in \mathbb{R}^k$.[4] We denote this mapping as $f : \mathbb{R}^n \to \mathbb{R}^k$:

$$f(\cdot) : \boldsymbol{x} \mapsto \boldsymbol{y}.$$

As we will see, when the observations $\boldsymbol{x}^i$ are high dimensional, this task is greatly facilitated by leveraging low-dimensional structure in the class distributions $\mathcal{D}_1, \ldots \mathcal{D}_k$. That is, the support of each of the distributions $\mathcal{D}_j$ is on certain low-dimensional submanifolds, denoted as $\mathcal{M}_j$, as illustrated in Figure 16.1.

Extensive empirical studies in recent years have shown that for many practical datasets (images, audios, and natural languages, etc.), the possibly complicated and nonlinear mapping $\boldsymbol{y} = f(\boldsymbol{x})$ can be effectively modeled by a deep network [GBC16].[5] A deep network is a composition of a series of simple maps, called "layers." Each layer, say denoted as $f^\ell(\cdot)$, is composed of a linear transform, represented by a matrix $\boldsymbol{W}_\ell$, followed by a simple (entry-wise) nonlinear activation function $\phi(\cdot)$.[6] More precisely, a network of L layers can be defined recursively as:

$$\boldsymbol{z}_{\ell+1} = f^\ell(\boldsymbol{z}_\ell) \doteq \phi(\boldsymbol{W}_\ell \boldsymbol{z}_\ell), \quad \ell = 0, 1, \ldots, L-1, \quad \boldsymbol{z}_0 = \boldsymbol{x}, \tag{16.1.1}$$

where $\{\boldsymbol{W}_\ell\}_{\ell=0}^{L-1}$ are tunable parameters of the network[7] and $\phi(\cdot)$ is the nonlinear

[3] We have seen a similar situation before in which we interpreted class labels as sparse vectors: the robust face recognition problem studied in Chapter 13.

[4] Be aware that in the literature of deep learning, it is customary to use $\boldsymbol{x}$ as the input and $\boldsymbol{y}$ as the output. In compressive sensing, as in the face recognition application of Chapter 13, we instead use $\boldsymbol{x}$ as the sparse signal to be recovered from an input image $\boldsymbol{y}$.

[5] Notice that in the setting of face recognition, such an inverse problem is solved by an iterative algorithm such as the ISTA or FISTA introduced in Chapter 8.

[6] For simplicity, we here ignore for now, some other operations between layers such as batch normalization and dropouts etc. but will discuss their roles later.

[7] There might be additional structures such as convolution in the linear transform $\boldsymbol{W}_\ell$.

activation function.[8] For simplicity, we denote the overall map as: $f(\boldsymbol{x}, \boldsymbol{\theta}) : \boldsymbol{x} \mapsto \boldsymbol{y}$ and use $\boldsymbol{\theta} \in \Theta$ to denote all the network parameters $\{\boldsymbol{W}_\ell\}_{\ell=0}^{L-1}$ and possibly some in the activation function ϕ too:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) \doteq \phi(\boldsymbol{W}_{L-1}\phi(\cdots\phi(\boldsymbol{W}_1\phi(\boldsymbol{W}_0\boldsymbol{x}))\cdots)) \tag{16.1.2}$$

$$= f^{L-1} \circ \cdots \circ f^1 \circ f^0(\boldsymbol{x}). \tag{16.1.3}$$

The goal of tuning the parameters $\boldsymbol{\theta}$ of the network is for the output of this map to best match the class label $\boldsymbol{y}$ for samples $\boldsymbol{x}$ from the distribution $\mathcal{D}$. In machine learning, this is often done by minimizing the *cross-entropy loss*:[9]

$$\min_{\boldsymbol{\theta}\in\Theta} L_{CE}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) \doteq -\mathbb{E}\big[\langle \boldsymbol{y}, \log[f(\boldsymbol{x}, \boldsymbol{\theta})]\rangle\big]. \tag{16.1.4}$$

Hence, given a large (presumably correctly) labelled dataset $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^m$, one solves the following (nonconvex) program:

$$\min_{\boldsymbol{\theta}\in\Theta} L_{CE}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) \doteq -\frac{1}{m}\sum_{i=1}^{m}\langle \boldsymbol{y}^i, \log[f(\boldsymbol{x}^i, \boldsymbol{\theta})]\rangle, \tag{16.1.5}$$

where $\log[\cdot]$ is entry-wise for the vector-valued $f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathbb{R}^k$. Since this loss function is in the form of a finite sum and the sample size m is very large (e.g. millions), the function is typically optimized by using variants of the stochastic gradient method (SGD) introduced in Section 8.6.4 of Chapter 8:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma_k \cdot \frac{\partial L(\boldsymbol{X}^k, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}, \tag{16.1.6}$$

where the gradient $\frac{\partial L}{\partial \boldsymbol{\theta}}\big|_{\boldsymbol{\theta}_k}$ is evaluated approximately using a random batch $\boldsymbol{X}^k \subset \boldsymbol{X}$ of samples at each iteration. Such optimization schemes have been efficiently implemented on many software platforms, (e.g., Caffe, PyTorch and TensorFlow). These numerical tools have significantly boosted the utility and popularity of deep learning.

16.1.2 The Practice of Deep Learning

Above, we have briefly described basic deep network structures and training methods. A vast array of modifications to the basic approach have been proposed, with the goal of improving the ease of training or performance of the learned network. An incomplete list of examples include:

- choices in loss functions or regularizations on parameters $\boldsymbol{W}_\ell$ [KH92, SZ14];
- different choices in the activation function ϕ (16.1.1) [XWCL15, NIGM18];
- width and depth of the networks [BC14, SLJ+15, LPW+17, DLL+19, AZLS19];
- skip connections across layers $f^\ell(\cdot)$ [SGS15, RFB15, HZRS16];

[8] Popular choices for $\phi(\cdot)$ include the Sigmod, Arctan, and more recently the rectified linear unit (ReLU) function. Sometimes $\phi(\cdot)$ may also contain some tunable parameters.

[9] The cross entropy loss is convenient for multi-class classification tasks. In practice, for tasks such as functional regression, the typical ℓ^2 loss: $\|\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\|_2^2$ is also commonly used.

- normalization of feature $\boldsymbol{z}_\ell$ in each layer [IS15,BKH16,UVL16,WH18,MKKY18];
- structures (convolutions) in the linear transform $\boldsymbol{W}_\ell$ [LBBH98,KSH12,Cho17];
- downsampling (pooling) and upsampling operations between layers [SMB10];
- initialization of the parameters $\boldsymbol{\theta}$ [LBOM12,GB10,HZRS15,XBSD+18,HXP20];
- choices in the batch size $|\boldsymbol{X}^k|$ (16.1.6) [HHS17, ML18, LSPJ18];
- learning rates γ_k for the SGD algorithm (16.1.6) [LBOM12, LH16, GKXS18];
- random dropout of connections during training [SHK+14, CMH+18];
- early stopping of the training sometimes [GJP95, Pre98, YRC07];
- different optimization algorithms [LNC+11,KB14,Mar14,MG15,BCN18,BJT19].

It can be challenging for practitioners to navigate this somewhat dizzying array of variations over the basic themes of deep learning. One recent trend in industrial practice is to leverage random search to identify architectures or training strategies that give better empirical performance, e.g., Neural Architecture Search (NAS) [ZL17, BGNR17], AutoML [HKV19], and Learning to Learn [ADG+16].

In subsequent sections, we will describe how ideas from low-dimensional data modeling can suggest principled architectures and clarify the roles of various architectural and algorithmic choices. In fact, a number of themes from low-dimensional data modeling recur throughout the deep learning practice:

- *Network Architectures from Unrolled Optimization Algorithms.* The widely used ReLU nonlinearity closely resembles the proximal operator for the (nonnegative) ℓ^1 norm. In fact, the proximal gradient algorithms that we introduced in Chapters 8–9 can be interpreted as particular neural networks, since they interleave linear operations with nonlinearities [GL10, LCWY19, SPRE18, PRSE18, MLE19]. This connection suggests new network designs from unrolling sparse coding algorithms for solving inverse problems from data with intrinsic *low-dimensional structures* [WLY+15, SLX+16, SLLB17, BJPD17, JMFU17, MJU17, NWMS18, OJB+20]. Some even outperform popular generic networks (e.g. ResNet and U-Net) with much more compact or simpler models [SNT20, LCBD20].
- *Isometry as a Design Principle.* Because network training algorithms propagate information through a large number of layers, it is important that these operations implement near-isometries. This can be achieved by properly initializing the weights [GB10, HZRS15], normalizing the features [IS15], or regularizing the network structure [SGS15, HZRS16], and can suggest modifications to network components such as nonlinearities [QYW+20]. This (empirical) principle suggests analogies to the *restricted isometry property* arising in sparse and low-rank recovery (see Chapters 3–4).
- *Explicit or Implicit Regularization.* Certain regularization strategies can be interpreted as encouraging low-dimensional structure in the learned network. A principal example is *dropout* [SHK+14], which has been shown to induce a form of low-rank (nuclear norm) regularization [CMH+18, MAV18, PLVH20] (see the exercises of Chapter 7). *Sparse routing* is also proven to be the key to enhance training and performance of ultra-large scale models [FZS21].

Many current mysteries around the generalization of learned networks can be approached from the perspective of implicit regularization induced by particular optimization methods or low-dimensional structures of the data [GLSS18, SHN+18, LMZ18, YZQM20].

In the remainder of this chapter, we will sketch a new approach to deriving neural networks from *first principles* and guaranteeing their performance on data exhibiting low-dimensional structure. The approach will provide some plausible explanations to the above connections. It leverages a connection to lossy data compression, which effectively encourages the network to embed mixed nonlinear data structures onto unions of incoherent linear subspaces.

16.1.3 Challenges with Nonlinearity and Discriminativeness

This chapter entails a significant expansion of scope compared to the first part of this book, which studied the recovery of sparse, low-rank or atomic structures from linear measurements. These models are, in a sense, piecewise *linear*. For instance, k-sparse vectors in the space $\mathbb{R}^n$ model data that lie on a particular union of k-dimensional linear subspaces, aligned with the standard basis. Our discussion of dictionary learning in Chapters 6 and 7 shows how to extend these models to unions of subspaces that are not aligned with the standard basis (and not known ahead of time). Nevertheless, real-world high-dimensional data, such as images, often exhibit nonlinear structure, due to nonlinear nuisance factors such as deformation. We have seen many examples of this in the application to *structured texture recovery* in Chapter 15.

In general, the distribution $\mathcal{D} = \{\mathcal{D}_j\}_{j=1}^k$ of a real (mixed) dataset, say in a typical setting for classification or clustering, is more likely to have its support on a mixture of low-dimensional *nonlinear* submanifolds $\{\mathcal{M}_j\}_{j=1}^k$, as illustrated in Figure 16.2 left. Hence, for the models and methods of this book to be applicable to real-world classification tasks, we have to overcome at least two major challenges:

- *From Nonlinear to Linear:* How to learn from the data a nonlinear (feature) mapping, say $f(\cdot, \boldsymbol{\theta}) : \boldsymbol{x} \mapsto \boldsymbol{z}$, such that we can first transform $\boldsymbol{x}$ on nonlinear submanifolds to $\boldsymbol{z}$ with linear structures, such as (a union of) low-dimensional subspaces.[10]
- *From Separable to Discriminative:* How to transform the resulting (separable) linear subspaces to be highly discriminative ones, *i.e.*, in positions such that the subspaces are highly incoherent (preferably orthogonal) to one another, as illustrated in Figure 16.2 right.

[10] Such a linear representation is highly desirable for many practical purposes. For instance, a linear superposition of features in the same subspace could be interpreted as a new instance in the same class. There is evidence that linear subspace is the kind of representations preferred by nature too, say for object recognition [CT17].

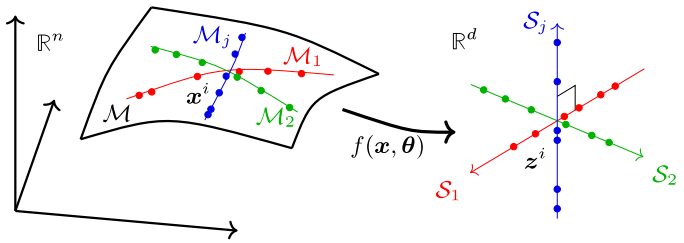


Figure 16.2 A mixed distribution $\mathcal{D} = \{\mathcal{D}_j\}$ of high-dim data $\boldsymbol{x} \in \mathbb{R}^n$ is supported on a manifold $\mathcal{M}$ which can be a mixture of multiple low-dimensional submanifolds $\{\mathcal{M}_j\}$. We want to learn a map $f(\boldsymbol{x}, \boldsymbol{\theta})$ such that $\boldsymbol{z}^i = f(\boldsymbol{x}^i, \boldsymbol{\theta})$ are on a union of low-dimensional subspaces $\{\mathcal{S}_j\}$.

Such a discriminative linear representation $\boldsymbol{z}$ can easily facilitate subsequent tasks: its linear nature makes linear interpolation or extrapolation of the features $\boldsymbol{z}$ (in each subspace) meaningful, and its discriminative nature makes prediction of the class label $\boldsymbol{y}$ easy, say by training a (linear) classifier $h(\boldsymbol{z})$:[11]

$$\boldsymbol{x} \xrightarrow{f(\boldsymbol{x},\boldsymbol{\theta})} \boldsymbol{z}(\boldsymbol{\theta}) \xrightarrow{h(\boldsymbol{z})} \boldsymbol{y}. \tag{16.1.7}$$

Notice that both challenges require us to perform *nonlinear* transformations of the data or features. Acute readers may have guessed it: The role of a deep network is precisely to model and perform such a nonlinear transformation! Now the remaining difficult questions are *why* such a nonlinear map should be represented by a composition of many simple layers, and *what* structures and properties the layers and operators need to have in order to efficiently realize such a map? Which parts of the network need to be learned and trained and which can be determined in advance? In the end, how to evaluate the optimality of the resulting network? To provide answers to these fundamental questions, we need a principled approach.

16.2 Desiderata for Learning Discriminative Representation

Whether the given data $\boldsymbol{X}$ of a mixed distribution $\mathcal{D}$ can be effectively classified depends on how separable (or discriminative) the component distributions $\mathcal{D}_j$ are (or can be made). One good working assumption is that the distribution of each class has relatively *low-dimensional* intrinsic structures.[12] Hence we may assume the distribution $\mathcal{D}_j$ of each class has a support on a low-dimensional submanifold,

[11] Intuitively speaking, the more incoherent the subspaces are, the larger the margin hence more generalizable the classifier would be.

[12] There are many reasons why this assumption is plausible: 1. high dimensional data are highly redundant; 2. data that belong to the same class should be similar and correlated to each other; 3. typically we only care about equivalent structures of $\boldsymbol{x}$ that are invariant to certain classes of transformations, as we will see in the next section.

say $\mathcal{M}_j$ with dimension $d_j \ll n$, and the distribution $\mathcal{D}$ of $\boldsymbol{x}$ is supported on the mixture of those submanifolds, $\mathcal{M} = \cup_{j=1}^{k} \mathcal{M}_j$, in the high-dimensional ambient space $\mathbb{R}^n$, as illustrated in Figure 16.2 left.

With the manifold assumption in mind, we want to learn a smooth mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ that maps each of the submanifolds $\mathcal{M}_j \subset \mathbb{R}^n$ to a *linear* subspace $\mathcal{S}_j \subset \mathbb{R}^d$ (see Figure 16.2 right). For the resulting features to be easy to classify or cluster, we require the learned representation to have the following properties:

1 *Between-Class Discriminative:* Features of samples from different classes or clusters should belong to different linear subspaces that are highly *incoherent* or uncorrelated.
2 *Within-Class Compressible:* Features of samples from the same class/cluster should be *compressible* in a sense that they belong to a relatively low-dimensional linear subspace.
3 *Maximally Informative Representation:* Dimension (or variance) of features for each class/cluster should be *as large as possible* as long as they stay incoherent from those of the other classes.

Notice that, although the intrinsic structures of each class/cluster may be low-dimensional, they are by no means simply linear in their original form (as we will elaborate on more in Section 16.4). The more ideal case when the data $\boldsymbol{X}$ lie on multiple linear subspaces has been systematically studied as *generalized principal component analysis* (GPCA) [VMS16]. Here the subspaces $\{\mathcal{S}_j\}$ obtained after the nonlinear mapping $f(\cdot)$ can be viewed as *nonlinear generalized principal components* for the original (mixed) data $\boldsymbol{X}$. If the resulting optimal subspaces are orthogonal (or statistically independent), they can also be viewed as *nonlinear independent components* of the data.

16.2.1 Measure of Compactness for a Representation

Although the above properties are all highly desirable for the learned representation $\boldsymbol{z}$, they are by no means easy to achieve. Recent work [PHD20] shows that the representations learned via the popular cross-entropy loss (16.1.5) expose a *neural collapsing* phenomenon, where within-class variability and structural information are completely suppressed and ignored, as we will also see in the experiments. So are the above list of properties compatible so that we can expect to achieve them all at once? More specifically, is it possible to find a *simple but principled* objective that can promote all these desired properties for the resulting representations?[13]

The key to these questions is to find a principled "measure of compactness" for the distribution of a random variable $\boldsymbol{z}$ or from its finite samples $\boldsymbol{Z}$. Such a measure should directly and accurately characterize intrinsic geometric or statistical properties of the distribution, in terms of its intrinsic dimension or volume.

[13] In a similar spirit of the ℓ^1 norm promoting sparsity and the nuclear norm $\|\cdot\|_*$ promoting low-rankness.

Unlike cross-entropy (16.1.4), such a measure should not depend explicitly on class labels so that it can work uniformly in all supervised, semi-supervised, self-supervised, and unsupervised settings.

Low-dimensional Degenerate Distributions.
In information theory [CT91], the notion of entropy $H(\boldsymbol{z})$ is designed to be such a measure.[14] However, entropy is not well-defined for continuous random variables with degenerate distributions.[15] This is unfortunately the case here. To alleviate this difficulty, another related concept in information theory, more specifically in lossy data compression, that measures the "compactness" of a random distribution is the so-called *rate distortion* [CT91]: Given a random variable $\boldsymbol{z}$ and a prescribed precision $\varepsilon > 0$, the rate distortion $R(\boldsymbol{z}, \varepsilon)$ is the minimal number of binary bits needed to encode $\boldsymbol{z}$ such that the expected decoding error is less than ε. That is, say in terms of the ℓ^2-norm, we have

$$\mathbb{E}[\|\boldsymbol{z} - \hat{\boldsymbol{z}}\|_2] \leq \varepsilon$$

for the decoded $\hat{\boldsymbol{z}}$.

Nonasymptotic Rate Distortion for Finite Samples.
When evaluating the lossy coding rate R, one practical difficulty is that we normally do not know the distribution of $\boldsymbol{z}$. Instead, we have a finite number of samples as learned representations where $\boldsymbol{z}^i = f(\boldsymbol{x}^i, \boldsymbol{\theta}) \in \mathbb{R}^d, i = 1, \ldots, m$, for the given data samples $\boldsymbol{X} = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m]$. Fortunately, from the perspective of lossy data compression, [MDHW07, VMS16] have provided a precise estimate on the number of binary bits needed to encoded finite samples from a subspace-like distribution. In order to encode the learned representation $\boldsymbol{Z} = [\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m]$ up to a precision ε, the total number of bits needed is given by the following expression[16]:

$$\mathcal{L}(\boldsymbol{Z}, \varepsilon) \doteq \left(\frac{m+d}{2}\right) \log\det\left(\boldsymbol{I} + \frac{d}{m\varepsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right). \tag{16.2.1}$$

See Figure 16.3 for an illustration. Therefore, the compactness of learned features *as a whole* can be measured in terms of the average coding length per sample (as the sample size m is large), a.k.a. the *coding rate* subject to the distortion ε:

$$R(\boldsymbol{Z}, \varepsilon) \doteq \frac{1}{2} \log\det\left(\boldsymbol{I} + \frac{d}{m\varepsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right). \tag{16.2.2}$$

As we have seen in Exercise 7.4 of Chapter 7, the $\log\det(\cdot)$ function is a smooth but nonconvex surrogate for promoting low-dimensionality of the representation

[14] given the probability density $p(\boldsymbol{z})$ of a random variable, $H(\boldsymbol{z}) \doteq -\int p(\boldsymbol{z}) \log p(\boldsymbol{z})\, d\boldsymbol{z}$.
[15] The same difficulty resides with evaluating mutual information $I(\boldsymbol{x}, \boldsymbol{z})$ for degenerate distributions.
[16] This formula can be derived either by packing ε-balls into the space spanned by $\boldsymbol{Z}$ or by computing the number of bits needed to quantize the SVD of $\boldsymbol{Z}$ subject to the precision, see [MDHW07] for proofs.

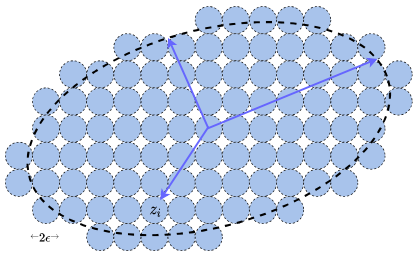


Figure 16.3 Lossy coding scheme: Given a precision ε, we pack the space/volume spanned by the data $\boldsymbol{Z}$ with small balls of diameter 2ε. The number of balls needed to pack the space gives the number of bits needed to record the location of each data point $\boldsymbol{z}^i$, up to the given precision.

$\boldsymbol{Z}$. We will soon discuss why here we need the a more accurate nonconvex surrogate for low-dimensionality rather than the convex nuclear norm $\|\cdot\|_*$. In addition, the particular choice of $\log\det(\cdot)$ seems to be rather fundamental and we will reveal many of its magical properties soon.

Rate Distortion of Data with a Mixed Distribution.
In general, the features $\boldsymbol{Z}$ of multi-class data may belong to multiple low-dimensional subspaces. To evaluate the rate distortion of such mixed data *more accurately*, we may partition the data $\boldsymbol{Z}$ into multiple subsets: $\boldsymbol{Z} = \boldsymbol{Z}^1 \cup \cdots \cup \boldsymbol{Z}^k$, with each in one low-dim subspace. So the above coding rate (16.2.2) is accurate for each subset. For convenience, let $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j \in \mathbb{R}^{m\times m}\}_{j=1}^k$ be a set of diagonal matrices whose diagonal entries encode the membership of the m samples in the k classes.[17] Then, according to [MDHW07], with respect to this partition, the average number of bits per sample (the coding rate) is

$$R^c(\boldsymbol{Z}, \varepsilon \mid \boldsymbol{\Pi}) \doteq \sum_{j=1}^{k} \frac{\operatorname{trace}\left(\boldsymbol{\Pi}^j\right)}{2m} \log\det\left(\boldsymbol{I} + \frac{d}{\operatorname{trace}\left(\boldsymbol{\Pi}^j\right)\varepsilon^2}\boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^*\right). \quad (16.2.3)$$

Notice that when $\boldsymbol{Z}$ is given, $R^c(\boldsymbol{Z}, \varepsilon \mid \boldsymbol{\Pi})$ is a concave function of $\boldsymbol{\Pi}$. The function $\log\det(\cdot)$ in the above expressions has been long known as an effective heuristic for rank minimization problems [FHB03], as we have explored in the exercises of Chapter 7. As it tightly characterizes the rate distortion of Gaussian or subspace-like distributions, finding the clustering $\boldsymbol{\Pi}$ that minimizes

$$\min_{\boldsymbol{\Pi}} R^c(\boldsymbol{Z}, \varepsilon \mid \boldsymbol{\Pi}) \quad (16.2.4)$$

has been shown to be very effective in clustering or classification of data with mixed low-dimensional (linear) structures [MDHW07, WTL+08, KPCC15]. We

[17] More precisely, the diagonal entry $\boldsymbol{\Pi}^j(i,i)$ of $\boldsymbol{\Pi}^j$ indicates the probability of sample i belonging to class j. So $\boldsymbol{\Pi}$ lies in a simplex: $\Omega \doteq \{\boldsymbol{\Pi} \mid \boldsymbol{\Pi}^j \geq \boldsymbol{0},\ \boldsymbol{\Pi}^1 + \cdots + \boldsymbol{\Pi}^k = \boldsymbol{I}_{m\times m}\}$.

will soon reveal a few surprisingly good properties of this function in the new context.

16.2.2 Principle of Maximal Coding Rate Reduction

On one hand, in the supervised learning setting, $\boldsymbol{\Pi}$ is given in advance, and we like to learn a good representation $\boldsymbol{Z}$. For learned features to be discriminative, features of different classes/clusters are preferred to be *maximally incoherent* to each other, similar to the notion of "incoherence" that we have studied in Chapter 3. Hence they together should span a space of the largest possible volume (or dimension) and the coding rate of the whole set $\boldsymbol{Z}$ should be as large as possible. On the other hand, learned features of the same class/cluster should be highly correlated and coherent. Hence, each class/cluster should only span a space (or subspace) of a very small volume and the coding rate should be as small as possible. Therefore, a good representation $\boldsymbol{Z}$ of $\boldsymbol{X}$ is one such that, given a partition $\boldsymbol{\Pi}$ of $\boldsymbol{Z}$, achieves a large difference between the coding rate for the whole and the average rate for all the subsets:

$$\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \varepsilon) \doteq R(\boldsymbol{Z}, \varepsilon) - R^c(\boldsymbol{Z}, \varepsilon \mid \boldsymbol{\Pi}). \tag{16.2.5}$$

If we choose the feature mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ to be a deep neural network, the overall process of the feature representation and the resulting rate reduction w.r.t. certain partition $\boldsymbol{\Pi}$ can be illustrated by the following diagram:

$$\boldsymbol{X} \xrightarrow{f(\boldsymbol{x},\boldsymbol{\theta})} \boldsymbol{Z}(\boldsymbol{\theta}) \xrightarrow{\boldsymbol{\Pi},\varepsilon} \Delta R(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \varepsilon). \tag{16.2.6}$$

Note that ΔR is *monotonic* in the scale of the features $\boldsymbol{Z}$. So to make the amount of reduction comparable between different representations,[18] we need to *normalize the scale* of the learned features, either by constraining the Frobenius norm of each class $\boldsymbol{Z}^j$ to scale with the number of features in $\boldsymbol{Z}^j \in \mathbb{R}^{d \times m_j}$: $\|\boldsymbol{Z}^j\|_F^2 = m_j$ or by normalizing each feature to be on the unit sphere: $\boldsymbol{z}^i \in \mathbb{S}^{d-1}$. This formulation offers a natural justification for the need of "batch normalization" in the practice of training deep neural networks [IS15]. An alternative, arguably simpler, way to normalize the scale of learned representations is to ensure that the mapping of each layer of the network is approximately *isometric* [QYW+20], as we have discussed in the previous subsection.

Once the representations are comparable, our goal becomes to learn a set of features $\boldsymbol{Z}(\boldsymbol{\theta}) = f(\boldsymbol{X}, \boldsymbol{\theta})$ and their partition $\boldsymbol{\Pi}$ (if not given in advance) such that they maximize the reduction between the coding rate of all features and that of the sum of features w.r.t. their classes:

$$\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \Delta R\big(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \varepsilon\big) \doteq R(\boldsymbol{Z}(\boldsymbol{\theta}), \varepsilon) - R^c(\boldsymbol{Z}(\boldsymbol{\theta}), \varepsilon \mid \boldsymbol{\Pi}), \;\; \text{s.t. } \boldsymbol{Z} \subset \mathbb{S}^{d-1}, \; \boldsymbol{\Pi} \in \Omega. \tag{16.2.7}$$

[18] Here different representations can be either representations associated with different network parameters or representations learned after different layers of the same deep network.

We refer to this as the principle of *maximal coding rate reduction* (MCR^2), an embodiment of a famous saying:

"*The whole is greater than the sum of the parts.*" – Aristotle.

Note that for the clustering purpose alone, one may only care about the sign of ΔR for deciding whether to partition the data or not, which leads to the greedy clustering algorithm in [MDHW07].[19] Here to seek or learn the most discriminative representation, we further desire:

The whole is *maximally* greater than the sum of the parts!

Remark 16.1 (Relationship to Information Gain). *The maximal coding rate reduction can be viewed as a generalization to* Information Gain *(IG), which aims to maximize the reduction of entropy of a random variable, say $\boldsymbol{z}$, with respect to an observed attribute, say $\boldsymbol{\pi}$:* $\max_{\boldsymbol{\pi}} IG(\boldsymbol{z}, \boldsymbol{\pi}) \doteq H(\boldsymbol{z}) - H(\boldsymbol{z} \mid \boldsymbol{\pi})$*, i.e., the* mutual information *between $\boldsymbol{z}$ and $\boldsymbol{\pi}$ [CT91]. Maximal information gain has been widely used in areas such as decision trees [Qui86]. However, MCR^2 is used differently in several ways: 1) One typical setting of MCR^2 is when the data class labels are given, i.e. $\boldsymbol{\Pi}$ is known, MCR^2 focuses on learning representations $\boldsymbol{z}(\boldsymbol{\theta})$ rather than fitting labels. 2) In traditional settings of IG, the number of attributes in $\boldsymbol{z}$ cannot be so large and their values are discrete (typically binary). Here the "attributes" $\boldsymbol{\Pi}$ represent the probability of a multi-class partition for all samples and their values can even be continuous. 3) As mentioned before, entropy $H(\boldsymbol{z})$ or mutual information $I(\boldsymbol{z}, \boldsymbol{\pi})$ [HFLM+18] is not well-defined for degenerate continuous distributions whereas the rate distortion $R(\boldsymbol{z}, \varepsilon)$ is and can be accurately and efficiently computed for (mixed) subspaces, at least.*

16.2.3 Properties of the Rate Reduction Function

In theory, the MCR^2 principle (16.2.7) benefits from great generalizability and can be applied to representations $\boldsymbol{Z}$ of *any* distributions with *any* attributes $\boldsymbol{\Pi}$ as long as the rates R and R^c for the distributions can be accurately and efficiently evaluated. The optimal representation $\boldsymbol{Z}_\star$ and partition $\boldsymbol{\Pi}_\star$ should have some interesting geometric and statistical properties. We here reveal nice properties of the optimal representation with the special case of subspaces, which have many important use cases in machine learning. When the desired representation for $\boldsymbol{Z}$ is multiple subspaces, the rates R and R^c in (16.2.7) are given by (16.2.2) and (16.2.3), respectively. Let us assume the maximal rate reduction is achieved at the optimal representation, denoted as $\boldsymbol{Z}_\star = \boldsymbol{Z}_\star^1 \cup \cdots \cup \boldsymbol{Z}_\star^k \subset \mathbb{R}^d$, with the dimension of each subspace rank $\left(\boldsymbol{Z}_\star^j\right) \leq d_j$. Then, one can show that $\boldsymbol{Z}_\star$ has the

[19] Strictly speaking, in the context of clustering *finite* samples, one needs to use the more precise measure of the coding length mentioned earlier, see [MDHW07] for more details.

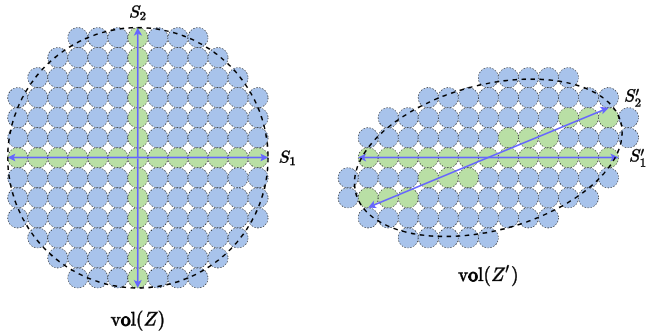


Figure 16.4 Comparison of two learned representations $\boldsymbol{Z}$ and $\boldsymbol{Z}'$ via reduced rates: R is the number of ε-balls packed in the joint distribution and R^c is the sum of the numbers for all the subspaces (the green balls). ΔR is their difference (the number of blue balls). The MCR^2 principle prefers $\boldsymbol{Z}$ (the left one).

following desired properties (see [YCY+20] for a formal statement and detailed proofs).

THEOREM 16.2 (Optimal Representation (informal statement)). *Suppose* $\boldsymbol{Z}_\star = \boldsymbol{Z}_\star^1 \cup \cdots \cup \boldsymbol{Z}_\star^k$ *is the optimal solution that maximizes the rate reduction* (16.2.7). *We have:*

- Between-class Discriminative*: As long as the ambient space is adequately large* ($d \geq \sum_{j=1}^k d_j$)*, the subspaces are all orthogonal to each other,* i.e. $(\boldsymbol{Z}_\star^i)^* \boldsymbol{Z}_\star^j = \boldsymbol{0}$ *for* $i \neq j$.
- Maximally Diverse Representation*: As long as the coding precision is adequately high, i.e.,* $\varepsilon^4 < \min_j \left\{ \frac{m_j}{m} \frac{d^2}{d_j^2} \right\}$*, each subspace achieves its maximal dimension, i.e.* $\operatorname{rank}\left(\boldsymbol{Z}_\star^j\right) = d_j$*. In addition, the largest* $d_j - 1$ *singular values of* $\boldsymbol{Z}_\star^j$ *are equal.*

In other words, in the case of subspaces, the MCR^2 principle promotes embedding of data into multiple independent subspaces, with features distributed nearly *isotropically* in each subspace (except for possibly one dimension). In addition, among all such discriminative representations, it prefers the one with the highest dimensions in the ambient space.

REMARK 16.3 (Rate Distortion $\log\det(\cdot)$ versus the Nuclear Norm). *To encourage the learned features to be incoherent between classes, the work of [LQMS18] has proposed to maximize the difference between the nuclear norm of the whole* $\boldsymbol{Z}$ *and its subsets* $\boldsymbol{Z}^j$*, called the* orthogonal low-rank embedding *(OLE) loss:*

$$\max_{\boldsymbol{\theta}} \mathit{OLE}(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}) \doteq \|\boldsymbol{Z}(\boldsymbol{\theta})\|_* - \sum_{j=1}^k \|\boldsymbol{Z}^j(\boldsymbol{\theta})\|_*, \tag{16.2.8}$$

added as a regularizer to the cross-entropy loss (16.1.4). *As we have learned from Chapter 4, the nuclear norm* $\|\cdot\|_*$ *is a* nonsmooth convex *surrogate for*

low-rankness, whereas $\log\det(\cdot)$ *is* smooth concave *instead. One can show that unlike the rate reduction* ΔR*, OLE is always* negative *and achieves the maximal value* 0 *when the subspaces are orthogonal, regardless of their dimensions. So in contrast to* ΔR*, this loss serves as a geometric heuristic for discriminativeness but does not promote diversity of the representation. In fact, OLE typically promotes learning one-dimensional representations per class [LQMS18], whereas MCR*^{2} *encourages learning subspaces with maximal dimensions.*

Remark 16.4 (Relation to Contrastive Learning.). *If samples are* evenly *drawn from* k *classes, a randomly chosen pair* $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ *is of high probability belonging to difference classes if* k *is large.*[20] *We may view the learned features of two samples together with their augmentations* $\boldsymbol{Z}^i$ *and* $\boldsymbol{Z}^j$ *as two classes. Then the rate reduction*

$$\Delta R^{ij} = R\left(\boldsymbol{Z}^i \cup \boldsymbol{Z}^j, \varepsilon\right) - \frac{1}{2}\left(R(\boldsymbol{Z}^i, \varepsilon) + R(\boldsymbol{Z}^j, \varepsilon)\right) \tag{16.2.9}$$

gives a "distance" measure for how far the two sample sets are. We may try to further "expand" pairs that likely belong to different classes. From Theorem 16.2, the (averaged) rate reduction ΔR^{ij} *is maximized when features from different samples are uncorrelated* $(\boldsymbol{Z}^i)^*\boldsymbol{Z}^j = \boldsymbol{0}$ *(see Figure 16.4) and features* $\boldsymbol{Z}^i$ *from the same sample are highly correlated. Hence, when applied to sample pairs, MCR*^{2} *naturally conducts the so-called* contrastive learning *[HCL06, OLV18, HFW+19]. But MCR*^{2} *is* not *limited to expand (or compress) pairs of samples and can uniformly conduct "contrastive learning" for a subset with* any number *of samples as long as we know they likely belong to different (or the same) classes, say by randomly sampling subsets from a large number of classes or with a good clustering method.*

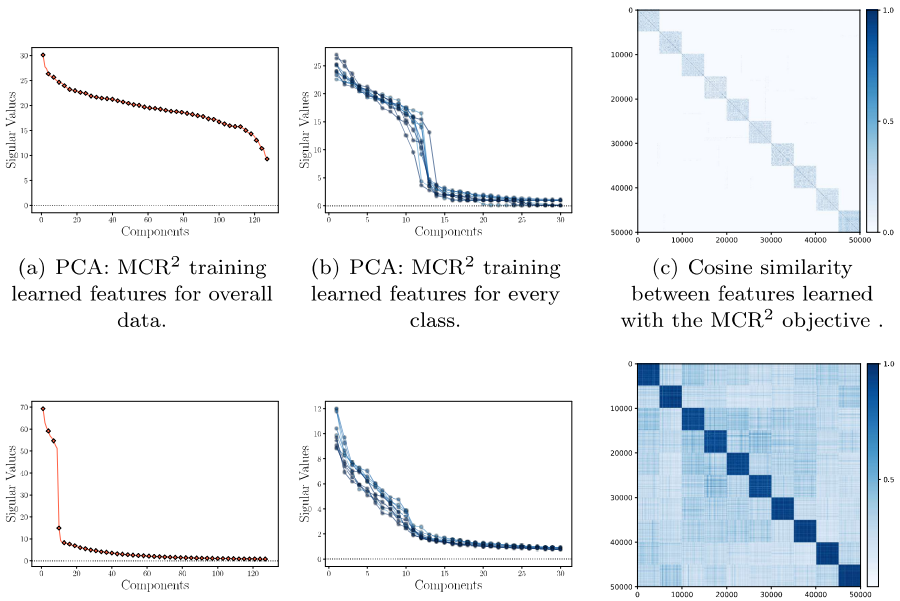
16.2.4 Experiments on Real Data

When class labels are provided during training, we assign the membership (diagonal) matrix $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j\}_{j=1}^k$ as follows: for each sample $\boldsymbol{x}^i$ with label j, set $\boldsymbol{\Pi}^j(i,i) = 1$ and $\boldsymbol{\Pi}^l(i,i) = 0, \forall l \neq j$. Then the mapping $f(\cdot, \boldsymbol{\theta})$ can be learned by optimizing (16.2.7), where $\boldsymbol{\Pi}$ remains constant. We apply stochastic gradient descent to optimize MCR^2, and for each iteration we use mini-batch data $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^m$ to approximate the MCR^2 loss.

As we will see, in the supervised setting, the learned representation has very clear subspace structures. So to evaluate the learned representations, we consider a natural nearest subspace classifier. For each class of learned features $\boldsymbol{Z}^j$, let $\boldsymbol{\mu}_j \in \mathbb{R}^d$ be its mean and $\boldsymbol{U}_j \in \mathbb{R}^{d \times r_j}$ be the first r_j principal components for $\boldsymbol{Z}^j$, where r_j is the estimated dimension of class j. The predicted label of a test data $\boldsymbol{x}'$ is given by[21] $j' = \operatorname{argmin}_{j \in \{1,\ldots,k\}} \|(\boldsymbol{I} - \boldsymbol{U}_j\boldsymbol{U}_j^*)(f(\boldsymbol{x}', \boldsymbol{\theta}) - \boldsymbol{\mu}_j)\|_2^2$.

[20] For example, when $k \geq 100$, a random pair is of probability 99% belonging to different classes.

[21] This is definitely not the best one can do to use the learned subspaces for classification. This particular classifier is chosen only for its simplicity.



(a) PCA: MCR^2 training learned features for overall data.

(b) PCA: MCR^2 training learned features for every class.

(c) Cosine similarity between features learned with the MCR^2 objective .

(d) PCA: cross-entropy training learned features for overall data.

(e) PCA: cross-entropy training learned features for every class.

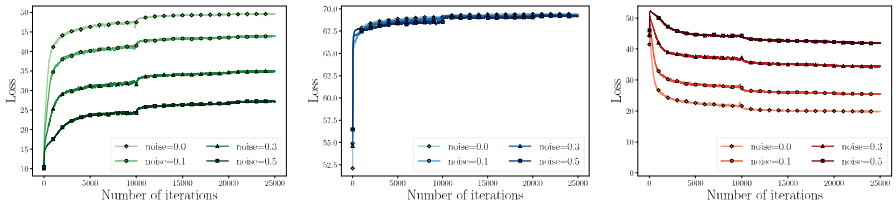
(f) Cosine similarity between features learned with the CE loss.

Figure 16.5 **Left:** Principal component analysis (PCA) of features learned with the MCR^2 objective or the cross-entropy. **Middel:** Principal components of features in individual classes. **Right:** Cosine similarity between learned features of all samples.

We consider CIFAR10 dataset [Kri09] and ResNet-18 [HZRS16] for $f(\cdot, \boldsymbol{\theta})$. We replace the last linear layer of ResNet-18 by a two-layer fully connected network with ReLU activation function such that the output dimension is 128. We set the mini-batch size as $m = 1,000$ and the precision parameter $\varepsilon^2 = 0.5$.

Discriminative and Diverse Linear Features.
We calculate the principal components of representations learned by MCR^2 training and cross-entropy training (16.1.5). For cross-entropy training, we take the output of the second last layer as the learned representation. The results are summarized in Figure 16.5. As shown in Figure 16.5 left, we observe that representations learned by MCR^2 are much more diverse, the dimension of learned features (each class) is around a dozen, and the dimension of the overall features is nearly 120, and the output dimension is 128. In contrast, the dimension of the overall features learned using entropy is slightly greater than 10,[22] which is much smaller than that learned by MCR^2. For visualization purposes, we also compare the cosine similarity between learned representations for both MCR^2

[22] This observation is consistent with the *neural collapsing* phenomenon associated with conventional loss function like cross entropy reported in the recent work [PHD20].



(a) $\Delta R\big(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \varepsilon\big)$. (b) $R(\boldsymbol{Z}(\boldsymbol{\theta}), \varepsilon)$. (c) $R^c(\boldsymbol{Z}(\boldsymbol{\theta}), \varepsilon \mid \boldsymbol{\Pi})$.

Figure 16.6 Evolution of rates $R, R^c, \Delta R$ of MCR^2 during training with corrupted labels.

Table 16.1 Classification results with features learned with labels corrupted at different levels.

Corrupt Ratio	10%	20%	30%	40%	50%
CE Training	90.91%	86.12%	79.15%	72.45%	60.37%
MCR^2 Training	**91.16%**	**89.70%**	**88.18%**	**86.66%**	**84.30%**

training and cross-entropy training, and the results are presented in Figure 16.5 right. From the figure, for MCR^2, we find that the features of different class are almost orthogonal and yet features of the same class are distributed rather evenly inside its subspace.

Robustness to Corrupted Labels.
Because MCR^2 by design encourages richer representations that preserves intrinsic structures from the data $\boldsymbol{X}$, training relies less on class labels than traditional loss such as cross-entropy (CE). To verify this, we train the same network[23] using both CE and MCR^2 with certain ratios of *randomly corrupted* training labels. Figure 16.6 illustrates the learning process: for different levels of corruption, while the rate for the whole set always converges to the same value, the rates for the classes are inversely proportional to the ratio of corruption, indicating the method only compresses samples with valid labels. The classification results are summarized in Table 16.1. By applying *exact the same* training parameters, MCR^2 is significantly more robust than CE, especially with higher ratio of corrupted labels. This can be an advantage in the settings of self-supervised learning or constrastive learning when the grouping information can be very noisy.

16.3 Deep Networks from First Principles

In the previous section, we have shown the optimal representation $\boldsymbol{Z}_\star$ that maximizes the rate reduction would indeed be both maximally discriminative and

[23] Both CE and MCR^2 can have better performance by choosing larger models for the mapping.

informative. Nevertheless, we do not know what the optimal feature mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ is and how to obtain it. In the above experiments, we have adopted a conventional deep network (e.g. the ResNet) as a black box to model the mapping and learned its parameters via back propagation. It has empirically shown that with such a choice, one can effectively optimize the MCR^2 objective and obtain discriminative and diverse representations for classifying real image data sets.

However, there are several unanswered questions. Although the objective is more intrinsic and the learned feature representation is arguably more interpretable, the network itself is still not interpretable. It is *not* clear why any chosen network is able to optimize the desired MCR^2 objective: would there be any potential limitations? The good empirical results (say with a ResNet) do not necessarily justify the particular choice in architectures and operators of the network: why is a layered deep model necessary in the first place; how wide and deep is adequate; and is there any rigorous justification for the particular convolutional and nonlinear operators used?

16.3.1 Deep Networks from Optimizing Rate Reduction

To simplify the presentation, we assume for now that the feature $\boldsymbol{z}$ and the input $\boldsymbol{x}$ have the same dimension $d = n$. But in general they can be different as we will soon see, say in the case $\boldsymbol{z}$ are multi-channel features extracted from $\boldsymbol{x}$.

Let us consider maximizing the rate reduction objective defined in (16.2.5):

$$\max_{\boldsymbol{Z}} \Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \varepsilon) \doteq \underbrace{\frac{1}{2}\log\det\Big(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^*\Big)}_{R(\boldsymbol{Z},\varepsilon)} - \underbrace{\sum_{j=1}^{k} \frac{\gamma_j}{2}\log\det\Big(\boldsymbol{I} + \alpha_j \boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^*\Big)}_{R^c(\boldsymbol{Z},\varepsilon|\boldsymbol{\Pi})}, \tag{16.3.1}$$

where to simplify the notation we define $\alpha = n/(m\varepsilon^2)$, $\alpha_j = n/(\mathsf{tr}(\boldsymbol{\Pi}^j)\varepsilon^2)$, $\gamma_j = \mathsf{tr}(\boldsymbol{\Pi}^j)/m$ for $j = 1, \ldots, k$.

Gradient Ascent for Rate Reduction on the Training Samples.
First let us directly try to optimize the objective $\Delta R(\boldsymbol{Z})$ as a function in the training samples $\boldsymbol{Z} \subset \mathbb{S}^{n-1}$. To this end, we may adopt the simplest (projected) *gradient ascent* scheme (introduced in Chapter 2), for some step size $\eta > 0$:

$$\boldsymbol{Z}_{\ell+1} \propto \boldsymbol{Z}_\ell + \eta \cdot \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} \quad \text{subject to} \quad \boldsymbol{Z}_{\ell+1} \subset \mathbb{S}^{n-1}. \tag{16.3.2}$$

This scheme can be interpreted as how one should incrementally adjust locations of the current features $\boldsymbol{Z}_\ell$ in order for the resulting $\boldsymbol{Z}_{\ell+1}$ to improve the rate reduction $\Delta R(\boldsymbol{Z})$, as illustrated in Figure 16.7.

Simple calculation shows that the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ entails evaluating the following

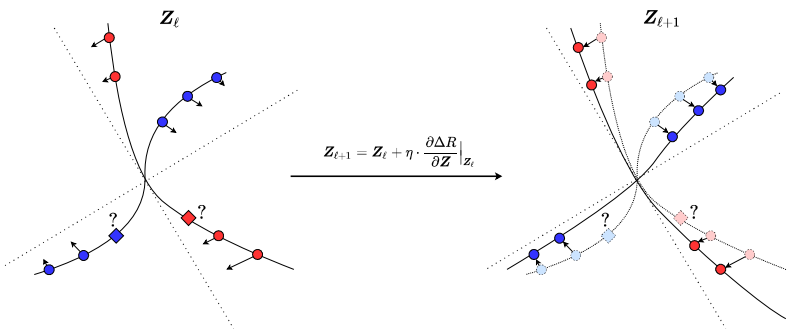


Figure 16.7 Incremental deformation of the training data, marked as "○", via gradient flow. Notice that for points whose memberships are unknown, those marked as "◇", their gradient cannot be directly calculated yet.

derivatives of the terms in (16.3.1) (we leave the derivation as an exercise to the reader):

$$\frac{1}{2}\frac{\partial \log\det(\boldsymbol{I}+\alpha \boldsymbol{Z}\boldsymbol{Z}^*)}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \underbrace{\alpha(\boldsymbol{I}+\alpha \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*)^{-1}}_{\boldsymbol{E}_\ell \in \mathbb{R}^{n\times n}} \boldsymbol{Z}_\ell, \tag{16.3.3}$$

$$\frac{1}{2}\frac{\partial \left(\gamma_j \log\det(\boldsymbol{I}+\alpha_j \boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^*)\right)}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \gamma_j \underbrace{\alpha_j(\boldsymbol{I}+\alpha_j \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j \boldsymbol{Z}_\ell^*)^{-1}}_{\boldsymbol{C}_\ell^j \in \mathbb{R}^{n\times n}} \boldsymbol{Z}_\ell\boldsymbol{\Pi}^j. \tag{16.3.4}$$

Then the complete gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$ is of the following form:

$$\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \underbrace{\boldsymbol{E}_\ell}_{\text{Expansion}} \boldsymbol{Z}_\ell - \sum_{j=1}^{k} \gamma_j \underbrace{\boldsymbol{C}_\ell^j}_{\text{Compression}} \boldsymbol{Z}_\ell\boldsymbol{\Pi}^j \quad \in \mathbb{R}^{n\times m}. \tag{16.3.5}$$

Notice that in the above, the matrix $\boldsymbol{E}_\ell$ only depends on $\boldsymbol{Z}_\ell$ and it aims to *expand* all the features to increase the overall coding rate; the matrix $\boldsymbol{C}_\ell^j$ depends on features from each class and aims to *compress* them to reduce the coding rate of each class.

Interpretation of the Two Linear Operators.
For any $\boldsymbol{z}_\ell$ we have

$$(\boldsymbol{I}+\alpha \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*)^{-1}\boldsymbol{z}_\ell = \boldsymbol{z}_\ell - \boldsymbol{Z}_\ell \widehat{\boldsymbol{q}}_\ell, \tag{16.3.6}$$

where

$$\widehat{\boldsymbol{q}}_\ell \doteq \operatorname*{argmin}_{\boldsymbol{q}_\ell} \alpha\|\boldsymbol{z}_\ell - \boldsymbol{Z}_\ell \boldsymbol{q}_\ell\|_2^2 + \|\boldsymbol{q}_\ell\|_2^2. \tag{16.3.7}$$

Notice that $\widehat{\boldsymbol{q}}_\ell$ is exactly the solution to the ridge regression of $\boldsymbol{z}_\ell$ with all the data points $\boldsymbol{Z}_\ell$ as regressors. Therefore, $\boldsymbol{E}_\ell$ is approximately (i.e. when m is large enough) the projection onto the orthogonal complement of the subspace spanned by columns of $\boldsymbol{Z}_\ell$. Another way to interpret the matrix $\boldsymbol{E}_\ell$ is through eigenvalue

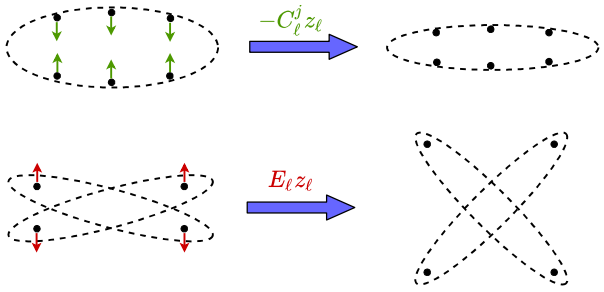


Figure 16.8 A little exaggerated interpretation of $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$: $\boldsymbol{E}_\ell$ expands all features by contrasting and repelling features across different classes; $\boldsymbol{C}_\ell^j$ compresses each class by contracting the features to a low-dimensional subspace.

decomposition of the covariance matrix $\boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*$. Assuming that $\boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^* \doteq \boldsymbol{U}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{U}_\ell^*$ where $\boldsymbol{\Lambda}_\ell \doteq \operatorname{diag}\{\sigma_1, \ldots, \sigma_d\}$, we have

$$\boldsymbol{E}_\ell = \alpha \boldsymbol{U}_\ell \operatorname{diag}\left\{\frac{1}{1+\alpha\sigma_1}, \ldots, \frac{1}{1+\alpha\sigma_d}\right\} \boldsymbol{U}_\ell^*. \tag{16.3.8}$$

Therefore, the matrix $\boldsymbol{E}_\ell$ operates on a vector $\boldsymbol{z}_\ell$ by stretching in a way that directions of large variance are shrunk while directions of vanishing variance are kept. These are exactly the directions (16.3.3) in which we move the features so that the overall volume expands and the coding rate will increase, hence the positive sign in (16.3.5). $\boldsymbol{C}_\ell^j$ has a similar interpretation as $\boldsymbol{E}$. But to the opposite effect, the directions associated with (16.3.4) are exactly "residuals" of features of each class deviate from the subspace to which they are supposed to belong. These are exactly the directions in which the features need to be compressed back onto their respective subspace, hence the negative sign in (16.3.5). This is illustrated in Figure 16.8.

Essentially, the two linear operations are determined by data conducting "auto-regressions" among themselves. The reader may recall that in the Introduction Chapter 1, we have mentioned the importance of regressions, especially the ridge regression. As we now see, regression is very much likely one of the ruling operations inside deep (neural) networks too. The recent renewed understanding about ridge regression in an over-parameterized setting [YYY+20, WX20] indicates that using seemingly redundantly sampled data (from each subspaces) as regressors do not lead to overfitting.

Gradient Flow Guided Feature Map Increment.
Notice that in the above, the gradient ascent considers all the features $\boldsymbol{Z}_\ell = [\boldsymbol{z}_\ell^1, \ldots, \boldsymbol{z}_\ell^m]$ as free variables. The increment $\boldsymbol{Z}_{\ell+1} - \boldsymbol{Z}_\ell = \eta \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$ does not yet give a transform on the entire feature domain $\boldsymbol{z}_\ell \in \mathbb{R}^n$. This is because gradients at points not in the training cannot be computed from (16.3.5), as illustrated by points marked as "$\diamond$" in Figure 16.7. Hence, in order to find the

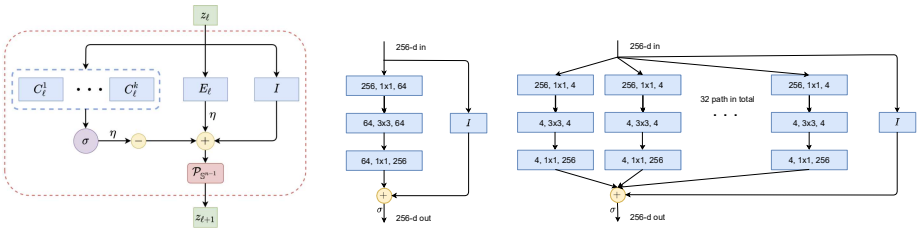


Figure 16.9 Comparison of Network Architectures. **Left**: Layer structure of the **ReduNet** derived from one iteration of gradient ascent for optimizing rate reduction. **Middle**: A layer of ResNet [HZRS16]; and **Right**: A layer of ResNeXt [XGD+17]. As we will see in the next section, the linear operators $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ of the ReduNet naturally become (multi-channel) convolutions when shift-invariance is imposed.

optimal feature mapping $f(\boldsymbol{x}, \boldsymbol{\theta})$ explicitly, we may consider constructing a small increment transform $g(\cdot, \boldsymbol{\theta}_\ell)$ on the ℓ-th layer feature $\boldsymbol{z}_\ell$ to emulate the above (projected) gradient scheme:

$$\boldsymbol{z}_{\ell+1} \propto \boldsymbol{z}_\ell + \eta \cdot g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \quad \text{subject to} \quad \boldsymbol{z}_{\ell+1} \in \mathbb{S}^{n-1} \tag{16.3.9}$$

such that: $\big[g(\boldsymbol{z}_\ell^1, \boldsymbol{\theta}_\ell), \ldots, g(\boldsymbol{z}_\ell^m, \boldsymbol{\theta}_\ell)\big] \approx \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$. That is, we need to approximate the gradient flow $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ that locally deforms each (training) feature $\{\boldsymbol{z}_\ell^i\}_{i=1}^m$ with a continuous mapping $g(\boldsymbol{z})$ defined on the entire feature space $\boldsymbol{z}_\ell \in \mathbb{S}^{n-1}$.

By inspecting the structure of the gradient (16.3.5), it suggests that a natural candidate for the increment transform $g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell)$ is of the form:

$$g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \doteq \boldsymbol{E}_\ell \boldsymbol{z}_\ell - \sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \quad \in \mathbb{R}^n, \tag{16.3.10}$$

where $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \in [0, 1]$ indicates the probability of $\boldsymbol{z}_\ell$ belonging to the j-th class.[24] Notice that the increment depends on 1). A set of linear maps represented by $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}_\ell^j\}_{j=1}^k$ that depend only on statistics of all features $\boldsymbol{Z}_\ell$ of the training; 2). membership $\{\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)\}_{j=1}^k$ of any feature $\boldsymbol{z}_\ell$.

Since we only have the membership $\boldsymbol{\pi}^j$ for the training samples, the function $g(\cdot)$ defined in (16.3.10) can only be evaluated on the training samples. To extrapolate the function $g(\cdot)$ to the entire feature space, we need to estimate $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$ in its second term. In the conventional deep learning, this map is typically modeled as a deep network and learned from the training data, say via *back propagation*. Nevertheless, our goal here is not to learn a precise classifier $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$ already. Instead, we only need a good enough estimate of the class information in order for $g(\cdot)$ to approximate the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ well.

From the previous geometric interpretation of the linear operators $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$, the term $\boldsymbol{p}_\ell^j \doteq \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell$ can be viewed as projection of $\boldsymbol{z}_\ell$ onto the orthogonal

[24] Notice that on the training samples $\boldsymbol{Z}_\ell$, for which the memberships $\boldsymbol{\Pi}^j$ are known, the so defined $g(\boldsymbol{z}_\ell, \boldsymbol{\theta})$ gives exactly the values for the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$.

complement of each class j. Therefore, $\|\boldsymbol{p}_\ell^j\|_2$ is small if $\boldsymbol{z}_\ell$ is in class j and large otherwise. This motivates us to estimate its membership based on the following "softmax" function:

$$\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell) \doteq \frac{\exp\left(-\lambda\|\boldsymbol{C}_\ell^j\boldsymbol{z}_\ell\|\right)}{\sum_{j=1}^k \exp\left(-\lambda\|\boldsymbol{C}_\ell^j\boldsymbol{z}_\ell\|\right)} \in [0,1]. \tag{16.3.11}$$

Hence the second term of (16.3.10) can be approximated by this estimated membership:[25]

$$\sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \approx \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \cdot \widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell) \;\doteq\; \boldsymbol{\sigma}\Big([\boldsymbol{C}_\ell^1\boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k\boldsymbol{z}_\ell]\Big), \tag{16.3.12}$$

which is denoted as a nonlinear operator $\boldsymbol{\sigma}(\cdot)$ on outputs of the feature $\boldsymbol{z}_\ell$ through k banks of filters: $[\boldsymbol{C}_\ell^1, \ldots, \boldsymbol{C}_\ell^k]$. Notice that the nonlinearity arises due to a "soft" assignment of class membership based on the feature responses from those filters. Overall, combining (16.3.9), (16.3.10), and (16.3.12), the increment feature transform from $\boldsymbol{z}_\ell$ to $\boldsymbol{z}_{\ell+1}$ now becomes:

$$\boldsymbol{z}_{\ell+1} \;\propto\; \boldsymbol{z}_\ell + \eta \cdot \boldsymbol{E}_\ell \boldsymbol{z}_\ell - \eta \cdot \boldsymbol{\sigma}\Big([\boldsymbol{C}_\ell^1\boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k\boldsymbol{z}_\ell]\Big) \quad \text{s.t.} \quad \boldsymbol{z}_{\ell+1} \in \mathbb{S}^{n-1}, \tag{16.3.13}$$

with the nonlinear function $\boldsymbol{\sigma}(\cdot)$ defined above and $\boldsymbol{\theta}_\ell$ collecting all the layer-wise parameters including $\boldsymbol{E}_\ell, \boldsymbol{C}_\ell^j, \gamma_j$ and λ. Note that features at each layer are always "normalized" onto a sphere $\mathbb{S}^{n-1}$, denoted as $\mathcal{P}_{\mathbb{S}^{n-1}}$. The form of increment in (16.3.13) can be illustrated by a diagram in Figure 16.9 left.

Deep Network from Rate Reduction.
Notice that the increment is constructed to emulate the gradient ascent for the rate reduction ΔR. Hence by transforming the features iteratively via the above process, we expect the rate reduction to increase, as we will see in the experimental section. This iterative process, once converged say after L iterations, gives the desired feature map $f(\boldsymbol{x}, \boldsymbol{\theta})$ on the input $\boldsymbol{z}_0 = \boldsymbol{x}$, precisely in the form of a *deep network*, in which each layer has the structure shown in Figure 16.9 left:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \phi^L \circ \phi^{L-1} \circ \cdots \circ \phi^0(\boldsymbol{x}), \quad \text{with} \tag{16.3.14}$$

$$\phi^\ell(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \doteq \mathcal{P}_{\mathbb{S}^{n-1}}[\boldsymbol{z}_\ell + \eta \cdot g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell)]. \tag{16.3.15}$$

As this deep network is derived from maximizing the rate **redu**ced, we call it the **ReduNet**. Notice that all parameters of the network are explicitly constructed layer by layer in a *forward propagation* fashion. Once constructed, there is no need of any additional supervised learning, say via back propagation. As we will see in the experiments, the so learned features can be directly used for classification, say via a nearest subspace classifier.

[25] The choice of the softmax is mostly for its simplicity as it is widely used in other (forward components of) deep networks for selection purposes, such as gating [SMM+17, FZS21] and routing [SFH17]. In principle, this term can be approximated by other operators, say using ReLU that is more amenable to training with back propagation, see Exercise 16.3.

Comparison with Other Approaches and Architectures.
As we have mentioned earlier, structural similarities between deep networks and iterative optimization schemes, especially those for solving sparse coding, have long been observed. For example in the work of Learned ISTA [GL10], one may view a fixed number of iterations of the ISTA Algorithm 8.1 as layers of a network. One can then use the back propagation to refine the parameters (say $\boldsymbol{A}$ in each layer) to improve convergence or accuracy of the resulting sparse codes. Later [GEBS18, MLE19, SNT20] have proposed similar interpretation of deep networks as unrolling algorithms for sparse coding.

Like all networks that are inspired by unfolding certain iterative optimization schemes, the structure of the ReduNet naturally contains a skip connection between adjacent layers as in the ResNet [HZRS16] (see Figure 16.9 middle). Nevertheless, the remaining $K+1$ parallel channels $\boldsymbol{E}, \{\boldsymbol{C}^j\}_{j=1}^K$ of the ReduNet actually draw resemblance to the parallel structures that people later found empirically beneficial for deep networks, e.g. ResNeXt [XGD+17] (see Figure 16.9 right) or the mixture of experts (MoE) module adopted in the latest large-scale language models [SMM+17, FZS21], in which the number of parallel banks (or experts) K can be in the thousands and the number of parameters can be in the billions and even trillions.

A major difference here is that these conventional networks are all found empirically or designed heuristically whereas all components (layers, operators, and parameters) of the ReduNet architecture are by explicit construction from the objective of maximizing the rate reduction ΔR. All operators have precise optimization, statistical and geometric interpretation consistent with the objective. Notice that even values of the parameters in the ReduNet can be constructed in a forward-propagation manner, although in principle one could still fine-tune the ReduNet with back-propagation if needed (as we will discuss more in the epilogue, Section 16.5 of the chapter). Furthermore, as the ReduNet architecture is based on choosing arguably the simplest gradient ascent scheme (16.3.2), we can expect more advanced optimization schemes introduced in Chapters 8–9 can lead to new architectures with improved efficiency (see Exercise 16.6 for a possible extension).

16.3.2 Convolutional Networks from Invariant Rate Reduction

So far, we have considered the data and features to be classified as vectors. In many applications, such as serial data or imagery data, the semantic meaning (labels) of the data and their features are *invariant* to certain transformations $\mathfrak{g} \in \mathbb{G}$ (for some group $\mathbb{G}$). For example, the meaning of an audio signal is invariant to shift in time; and the identity of an object in an image is invariant to translation in the image plane.[26] Hence, we prefer the feature mapping $f(\boldsymbol{x}, \boldsymbol{\theta})$

[26] The transform invariant textures (TILT) studied in the previous Chapter 15 are examples with more general groups of transformations, such as 2D affine transform or homography.

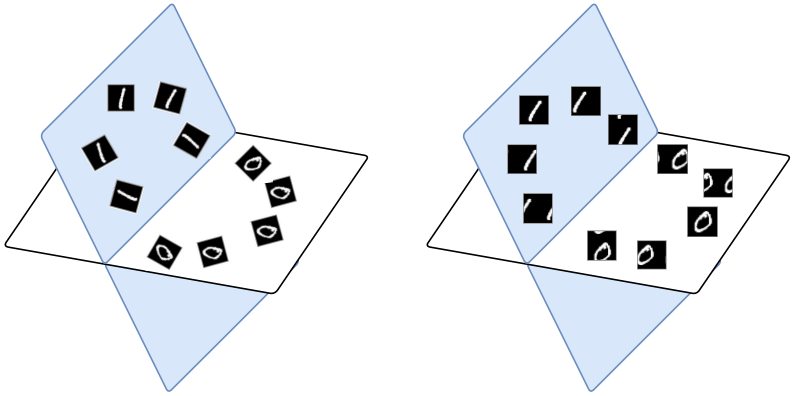


Figure 16.10 Illustration of the sought representation that is equivariant/invariant to image rotation (left) or translation (right): all transformed images of each class are mapped into the same subspace that are incoherent to other subspaces. The features embedded in each subspace are equivariant to transformation group whereas each subspace is invariant to such transformations.

is invariant to such transformations:

$$\textit{Group Invariance:} \quad f(\boldsymbol{x} \circ \mathfrak{g}, \boldsymbol{\theta}) \sim f(\boldsymbol{x}, \boldsymbol{\theta}), \quad \forall \mathfrak{g} \in \mathbb{G}, \tag{16.3.16}$$

where "$\sim$" indicates two features belonging to the same equivalent class. The submanifolds associated with such equivalent classes are known to have sophisticated geometric and topological structures [WDCB05]. This may explain why it has been very challenging for empirically designed deep networks to ensure invariance to even simple transformations such as translation and rotation [AW18, ETT+17].[27]

In this section, we show that the MCR^2 principle is compatible with invariance in a very natural and rigorous way: we only need to assign all transformed versions $\{\boldsymbol{x} \circ \mathfrak{g} \mid \mathfrak{g} \in \mathbb{G}\}$ into the same class as $\boldsymbol{x}$ and map them all to the same subspace $\mathcal{S}$.[28] See Figure 16.10 for an illustration of the examples of 1D rotation and 2D translation. Then one can show that, when the group $\mathbb{G}$ is (discrete) circular 1D shifting or 2D translation, the resulting deep network, the ReduNet, naturally becomes a *multi-channel convolutional network*!

1D Serial Data and Shift Invariance.
For one-dimensional data $\boldsymbol{x} \in \mathbb{R}^n$ under shift symmetry, we take $\mathbb{G}$ to be the group of circular shifts. Each observation $\boldsymbol{x}^i$ generates a family $\{\boldsymbol{x}^i \circ \mathfrak{g} \,|\, \mathfrak{g} \in \mathbb{G}\}$

[27] Recent study starts to reveal necessary conditions for a deep network to be invariant or equivariant to certain group transforms [CW16, CGW19].
[28] Hence, any subsequent classifiers defined on the resulting set of subspaces will be automatically invariant to such transformations.

of shifted copies, which are the columns of the circulant matrix $\mathsf{circ}(\boldsymbol{x}^i) \in \mathbb{R}^{n\times n}$ (see Appendix A.7 or [KS12] for properties of circulant matrices).

What happens if we construct the ReduNet from these families

$$\boldsymbol{Z}_1 = \big[\mathsf{circ}(\boldsymbol{x}^1), \ldots, \mathsf{circ}(\boldsymbol{x}^m)\big]?$$

The data covariance matrix:

$$\begin{aligned}\boldsymbol{Z}_1\boldsymbol{Z}_1^* &= \big[\mathsf{circ}(\boldsymbol{x}^1), \ldots, \mathsf{circ}(\boldsymbol{x}^m)\big]\,\big[\mathsf{circ}(\boldsymbol{x}^1), \ldots, \mathsf{circ}(\boldsymbol{x}^m)\big]^* \\ &= \sum_{i=1}^{m} \mathsf{circ}(\boldsymbol{x}^i)\mathsf{circ}(\boldsymbol{x}^i)^* \ \in \mathbb{R}^{n\times n}\end{aligned}$$

associated with this family of samples is *automatically* a (symmetric) circulant matrix. Moreover, because the circulant property is preserved under sums, inverses, and products (see Appendix A.7), the matrices $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$ are also automatically circulant matrices, whose application to a feature vector $\boldsymbol{z}$ can be implemented using cyclic convolution "$\circledast$." Specifically, we have the following proposition.

PROPOSITION 16.5 (Convolution Structures of $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$). *The matrix* $\boldsymbol{E}_1 = \alpha\big(\boldsymbol{I} + \alpha\boldsymbol{Z}_1\boldsymbol{Z}_1^*\big)^{-1}$ *is a circulant matrix and represents a circular convolution:*

$$\boldsymbol{E}_1\boldsymbol{z} = \boldsymbol{e}_1 \circledast \boldsymbol{z},$$

where $\boldsymbol{e}_1 \in \mathbb{R}^n$ *is the first column vector of* $\boldsymbol{E}_1$ *and "*$\circledast$*" is cyclic convolution defined as*

$$(\boldsymbol{e}_1 \circledast \boldsymbol{z})_i \doteq \sum_{j=0}^{n-1} e_1(j)x(i+n-j \bmod n). \tag{16.3.17}$$

Similarly, the matrices $\boldsymbol{C}_1^j$ *associated with any subsets of* $\boldsymbol{Z}_1$ *are also circular convolutions.*

From Proposition 16.5, we have

$$\begin{aligned}\boldsymbol{z}_2 &\propto \boldsymbol{z}_1 + \eta\cdot g(\boldsymbol{z}_1, \boldsymbol{\theta}_1) \\ &= \boldsymbol{z}_1 + \eta\cdot \boldsymbol{e}_1 \circledast \boldsymbol{z}_1 - \eta\cdot\boldsymbol{\sigma}\Big(\big[\boldsymbol{c}_1^1 \circledast \boldsymbol{z}_1, \ldots, \boldsymbol{c}_1^k \circledast \boldsymbol{z}_1\big]\Big).\end{aligned} \tag{16.3.18}$$

Because $g(\cdot, \boldsymbol{\theta}_1)$ consists only of operations that co-vary with cyclic shifts, the features $\boldsymbol{Z}_2$ at the next level again consist of families of shifts:

$$\boldsymbol{Z}_2 = \big[\mathsf{circ}(\boldsymbol{x}^1 + \eta g(\boldsymbol{x}^1, \boldsymbol{\theta}_1)), \ldots, \mathsf{circ}(\boldsymbol{x}^m + \eta g(\boldsymbol{x}^m, \boldsymbol{\theta}_1))\big]. \tag{16.3.19}$$

Continuing inductively, we see that all matrices $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ based on such $\boldsymbol{Z}_\ell$ are circulant. By virtue of the equivariant properties of the data, the ReduNet has taken the form of a convolutional network, *with no need to explicitly choose this structure!*

The Role of Multiple Channels.
There is one problem though: in general, the set of all circular permutations of a vector $\boldsymbol{x}$ give a full-rank matrix. That is, the n "augmented" features associated with each sample (hence each class) typically already span the entire space $\mathbb{R}^n$. The MCR^2 objective (16.3.1) will not be able to distinguish classes as different subspaces.

One natural remedy is to improve the separability of the data by "lifting" the signals $\boldsymbol{x}$ to a higher dimensional space,[29] e.g., by taking their responses to multiple, filters $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_C \in \mathbb{R}^n$:

$$\boldsymbol{z}[c] = \boldsymbol{k}_c \circledast \boldsymbol{x} = \mathsf{circ}(\boldsymbol{k}_c)\boldsymbol{x} \ \in \mathbb{R}^n, \quad c = 1, \ldots, C. \tag{16.3.20}$$

The filers can be pre-designed invariance-promoting filters,[30] or adaptively learned from the data,[31] or randomly selected as we do in the experiments. This operation lifts each original signal (vector) $\boldsymbol{z} \in \mathbb{R}^n$ to a C-channel feature vector, denoted $\bar{\boldsymbol{z}} \doteq [\boldsymbol{z}[1], \ldots, \boldsymbol{z}[C]]^* \in \mathbb{R}^{C \times n}$. If we stack the multiple channels of a feature $\bar{\boldsymbol{z}}$ as a column vector $\mathrm{vec}(\bar{\boldsymbol{z}}) \in \mathbb{R}^{nC}$, the associated circulant version $\mathsf{circ}(\bar{\boldsymbol{z}}) \in \mathbb{R}^{nC \times n}$ and its data covariance matrix, denoted as $\bar{\boldsymbol{\Sigma}} \in \mathbb{R}^{nC \times nC}$, for all its shifted versions are given as:

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix}, \quad \bar{\boldsymbol{\Sigma}} \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix} \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1])^*, \ldots, \mathsf{circ}(\boldsymbol{z}[C])^* \end{bmatrix}, \tag{16.3.21}$$

where $\mathsf{circ}(\boldsymbol{z}[c]) \in \mathbb{R}^{n \times n}$ with $c \in [C]$ is the circulant version of the c-th channel of the feature $\bar{\boldsymbol{z}}$. Then the columns of $\mathsf{circ}(\bar{\boldsymbol{z}})$ will only span at most an n-dimensional proper subspace in $\mathbb{R}^{nC}$.

Tradeoff between Invariance and Sparsity.
However, this simple (linear) lifting operation is not sufficient to render the classes separable still – features associated with other classes will likely span the *same* n-dimensional subspace in the lifted space. This reflects a fundamental conflict between linear (subspace) modeling and invariance: on one hand, we desire the resulting representation to be linear hence superposition of features of signals (including their shifted versions) in the same class remain in the same subspace (in the lifted feature space); on the other hand, we want features of signals in different classes can be separated and belong to different (incoherent) subspaces.

One way, and probably the only way, to resolve this conflict is to impose

[29] There are evidences in neuroscience that suggest such an expansion of dimension brings benefits to cognition [FMR16].
[30] For 1D signals like audio, one may consider the conventional short time Fourier transform (STFT); for 2D images, one may consider 2D wavelets as in the ScatteringNet [BM13].
[31] For learned filters, one can learn filters from the given data as the principal components of samples as in the PCANet [CJG+15] or from convolution dictionary learning [LB19, QLZ19].

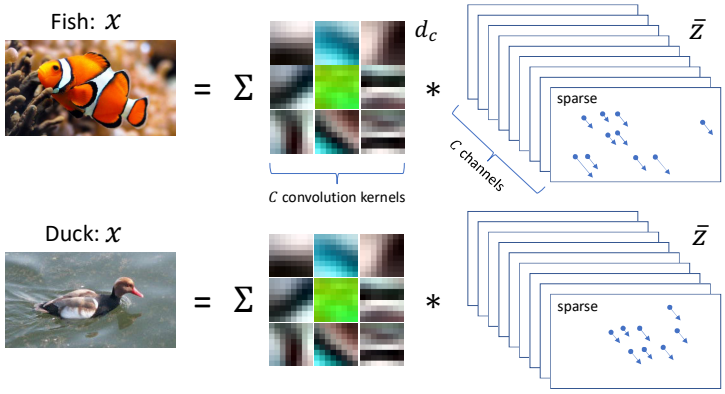


Figure 16.11 Each input signal $\boldsymbol{x}$ (an image here) can be represented as a superposition of sparse convolutions with multiple kernels $\boldsymbol{d}_c$ in a dictionary $\boldsymbol{D}$.

additional structures on signals in each class, in the form of *sparsity*: we may assume all signals $\boldsymbol{x}$ (including their shifted versions) within each class j are generated by only *sparse* combinations of shifted atoms (or motifs) in a dictionary $\boldsymbol{D}_j$:

$$\boldsymbol{x} = \mathsf{circ}(\boldsymbol{D}_j)\boldsymbol{z}_j \tag{16.3.22}$$

for some sparse $\boldsymbol{z}_j$, as shown in Figure 16.11.[32] Furthermore, we assume the dictionaries $\boldsymbol{D} = \{\boldsymbol{D}_j\}_{j=1}^k$ between the k classes are mutually incoherent. Hence signals in one class are unlikely to be sparsely represented by atoms in any other class. Then, all signals in the k classes can be sparsely represented by the all the dictionaries together:

$$\boldsymbol{x} = \big[\mathsf{circ}(\boldsymbol{D}_1), \mathsf{circ}(\boldsymbol{D}_2), \ldots, \mathsf{circ}(\boldsymbol{D}_k)\big]\bar{\boldsymbol{z}} \tag{16.3.23}$$

for some sparse $\bar{\boldsymbol{z}}$ which encodes the membership of the signal $\boldsymbol{x}$ with respect to the k classes. The reader may have recognized that this model is very similar to the face recognition setting that we have seen in Chapter 13. There is a vast literature on how to learn the most compact and optimal sparsifying dictionaries from sample data, as we have touched upon before in Chapters 7, 9, and 12. One may also refer to [LB19, QLZ19] for more references on this subject.

Nevertheless, here we are not interested in the precise optimal sparse code for every individual signal. We are only interested whether the set of sparse codes

[32] In practice, one can further assume the atoms are "short" (or have small supports) so that the generative model is similar to the "short and sparse" model that we have studied in Chapter 12.

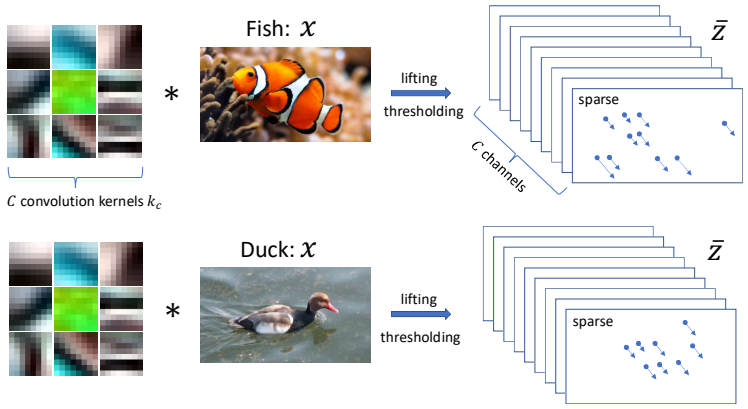


Figure 16.12 Estimate the sparse code $\bar{\boldsymbol{z}}$ of an input signal $\boldsymbol{x}$ (an image here) by taking convolutions with multiple kernels $\boldsymbol{k}_c$ and then sparsifying.

for each class are collectively separable from those of other classes.[33] Under the assumption of the sparse generative model, if the convolution kernels $\{\boldsymbol{k}_c\}$ match well with the "transpose" or "inverse" of the above sparsifying dictionaries, also known as the *analysis filters* [NDEG13, RE14], signals in one class will only have high responses to a small subset of those filters and low responses to others (due to the incoherence assumption). Figure 16.12 illustrates the basic ideas. Nevertheless, in practice, often a sufficient number of random filters suffice the purpose of ensuring features of different classes have different response patterns to different filters hence make different classes separable [CJG+15]. We will use the simple random filter design in the experiments to verify the concept.[34]

Hence the multi-channel responses $\bar{\boldsymbol{z}}$ should be sparse. So to approximate the sparse code $\bar{\boldsymbol{z}}$, we may take an entry-wise *sparsity-promoting nonlinear thresholding*, say $\boldsymbol{\tau}(\cdot)$, on the filter outputs by setting low (say absolute value below ε) or negative responses to be zero:[35]

$$\bar{\boldsymbol{z}} = \boldsymbol{\tau}\big[\mathsf{circ}(\boldsymbol{k}_1)\boldsymbol{x}, \ldots, \mathsf{circ}(\boldsymbol{k}_C)\boldsymbol{x}\big] \quad \in \mathbb{R}^{n \times C}. \tag{16.3.24}$$

One may refer to [RE14] for a more systematical study on the design of the sparsifying thresholding operator. Nevertheless, here we are not so interested in obtaining the best sparse codes as long as the codes for different classes are

[33] Note that this is rather different from our goal of computing sparse codes in earlier chapters of this book.

[34] Better sparse coding schemes may surely lead to better classification performance, though at a higher computational cost for learning or designing the filters and computing the sparse codes more precisely.

[35] The reader should be aware that, besides the feature scale normalization and membership assignment operation, this is the third, and final, type of nonlinear operation that we have encountered.

sufficiently separable. Hence the nonlinear operator $\boldsymbol{\tau}(\cdot)$ can be simply chosen to be a soft thresholding or a ReLU. These presumably highly sparse features $\bar{\boldsymbol{z}}$ can be assumed to lie on a lower-dimensional submanifold in $\mathbb{R}^{n\times C}$, which can be linearized and separated from the other classes by subsequent ReduNet layers.

The ReduNet constructed from the circulant version of these multi-channel features $\bar{\boldsymbol{z}}$ retains the good invariance properties described above: the linear operators, now denoted as $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j \in \mathbb{R}^{nC\times nC}$ as they are computed from the lifted and sparsified features $\bar{\boldsymbol{z}}$, remain block circulant. Hence, they represent *multi-channel 1D circular convolutions* (see [CYY+20] for a rigorous statement and proof):

$$\bar{\boldsymbol{E}}(\bar{\boldsymbol{z}}) = \bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}}, \quad \bar{\boldsymbol{C}}^j(\bar{\boldsymbol{z}}) = \bar{\boldsymbol{c}}^j \circledast \bar{\boldsymbol{z}} \quad \in \mathbb{R}^{n\times C}, \quad j = 1, \ldots, k, \tag{16.3.25}$$

where $\bar{\boldsymbol{e}}, \bar{\boldsymbol{c}}^j \in \mathbb{R}^{C\times C\times n}$ are the associated multi-channel convolution kernels. Hence by virtue of the equivariant data structures, the resulting ReduNet is naturally a deep convolutional network for multi-channel 1D signals. Notice that the number of channels remain constant through the layers (or iterations).

Fast Computation in the Spectral Domain.
Since all circulant matrices can be simultaneously diagonalized by the discrete Fourier transform (DFT) matrix[36] $\boldsymbol{F}$: $\mathsf{circ}(\boldsymbol{z}) = \boldsymbol{F}^*\boldsymbol{D}\boldsymbol{F}$ (see Theorem A.32 in Appendix A.7), all $\bar{\boldsymbol{\Sigma}}$ of the form (16.3.21) can be converted to a standard "blocks of diagonals" form:

$$\bar{\boldsymbol{\Sigma}} = \begin{bmatrix} \boldsymbol{F}^* & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F}^* \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{11} & \cdots & \boldsymbol{D}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C1} & \cdots & \boldsymbol{D}_{CC} \end{bmatrix} \begin{bmatrix} \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F} \end{bmatrix} \in \mathbb{R}^{nC\times nC}, \tag{16.3.26}$$

where each block $\boldsymbol{D}_{kl}$ is an $n\times n$ diagonal matrix. The middle of RHS of (16.3.26) is a block diagonal matrix after a permutation of rows and columns. There are n blocks of size $C \times C$. Hence, to compute $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j \in \mathbb{R}^{nC\times nC}$, we only have to compute in the frequency domain the inverse of $C \times C$ blocks for n times and the overall complexity would be $O(nC^3)$ instead of $O((nC)^3)$ for inverting a generic $nC \times nC$ matrix. Notice that the advantage of the spectral domain would have not been as significant had the computation of the operators $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$ did not involve matrix inverse. We leave this as an exercise for the reader (Exercise 16.5).

2D Images and Translation Invariance.
In the case of classifying images invariant to arbitrary 2D translation, we may view the image (feature) $\bar{\boldsymbol{z}} \in \mathbb{R}^{(W\times H)\times C}$ as a function defined on a torus $\mathcal{T}^2$ (discretized as a $W \times H$ grid) and consider $\mathbb{G}$ to be the (Abelian) group of all 2D (circular) translations on the torus. See Figure 16.18 for an illustration and example. Analogous to the 1D case, the associated linear operators $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$'s act

[36] Here we scaled the matrix $\boldsymbol{F}$ to be unitary, hence it differs from the conventional DFT matrix by a $1/\sqrt{n}$.

on the image feature $\bar{\boldsymbol{z}}$ as *multi-channel 2D circular convolutions*. The resulting network will be a deep convolutional network that shares the same multi-channel convolution structures as conventional CNNs for 2D images [LJB+95, KSH12]. The difference is that, again, the architecture of the network and parameters of the convolutions are all derived from the rate reduction objective, including the (layer) normalization and the nonlinear activations $\widehat{\boldsymbol{\pi}}^j$ and $\boldsymbol{\tau}$. Again, one can show that this multi-channel 2D convolutional network can be constructed more efficiently in the spectral domain (see [CYY+20] for a rigorous statement and proof). One may see [CYY+20] for implementation details of such a ReduNet in the spectral domain, for translation invariance of both 1D serial data and 2D imagery data.

Connections to Convolutional and Recurrent Sparse Coding.
We see from above that in order to find a discriminative linear representation for multiple classes of signals/images that is invariant to translation, sparse coding via lifting, a multi-layer architecture with multi-channel convolutions, and spectrum computing all become necessary components for achieving the objective effectively and efficiently. Figure 16.13 illustrates the whole process of learning such a representation via invariant rate reduction on the input sparse codes. Conceptual and algorithmic similarities between sparse coding and deep networks have long been observed, especially in the work of Learned ISTA [GL10]. It was later extended to be convolutional for imagery data or recurrent networks for serial data, e.g. [WPPA16, PRE16, SPRE18, MLE19]. Although both sparsity and convolution have been widely advocated as desired characteristics for deep networks, their precise roles for the classification task have never been clearly revealed nor justified. For instance, using convolutional operators to ensure equivarience has been common practice in deep networks [LB95a, CW16], but the number of convolutions needed is not clear and their parameters need to be learned via back propagation from randomly initialized ones. Of course, one may also predesign convolution filters of each layer to ensure translational invariance for a wide range of signals, say using wavelets as in ScatteringNet [BM13] and many followup works [WB18]. However, the number of convolutions needed usually grow exponentially with the number of layers. That is the reason why ScatteringNet type networks cannot be so deep, usually only 2-3 layers. It has never been clear in these framework how to design multi-channel convolutions. In contrast, in the rate reduction framework, we see that the roles of the multi-channel convolutions $(\bar{\boldsymbol{E}}, \bar{\boldsymbol{C}}^j)$ are explicitly derived and justified, the number of filters (channels) remains constant through all layers, and their parameters are determined by the data of interest.[37] As we see from the above derivation, both the convolution filters and sparsity requirements are *necessary* for success in the objective: incrementally learning a discriminative linear representation that is invariant to translation.

[37] Of course, the values of the parameters can be further fine-tuned if needed.

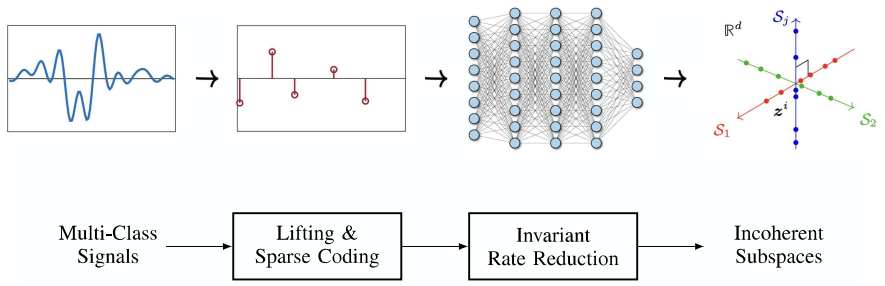


Figure 16.13 Overview of the process for classifying multi-class signals with shift invariance: Multi-channel lifting and sparse coding followed by a (convolutional) ReduNet for invariant rate reduction. These operations are *necessary* to map shift-invariant multi-class signals to incoherent (linear) subspaces. Note that the architectures of most modern deep neural networks resemble this process.

16.3.3 Simulations and Experiments

We now *verify* whether the so constructed ReduNet achieves its design objectives through some basic experiments on synthetic data and real images. The datasets and experiments are chosen to clearly demonstrate the behaviors of the networks obtained this way, in terms of learning the correct discriminative representation and achieving invariance. Although these basic and early experiments are very promising, it remains active and exciting research to further improve the performance and scalability of such networks in practice. We will leave some of the discussions to the epilogue of the chapter.

Simulation: Learning Mixture of Gaussians in $\mathbb{S}^2$.
We consider mixture of three Gaussian distributions in $\mathbb{R}^3$ with means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$ uniformly in $\mathbb{S}^2$, and variance $\sigma_1 = \sigma_2 = \sigma_3 = 0.1$. We sample $m = 500$ points from the distribution and all data points are projected onto $\mathbb{S}^2$ (see Figure 16.14). To construct the network (computing $\boldsymbol{E}, \boldsymbol{C}^j$ for each layer), we set the # of iterations/layers $L = 2,000$,[38] step size $\eta = 0.5$, and precision $\varepsilon = 0.1$. As shown by the two plots on the left of Figure 16.14, we can observe that after the mapping $f(\cdot, \boldsymbol{\theta})$, samples from the same class converge to a single cluster and the angle between different clusters is nearly orthogonal, which agrees with properties of the optimal solution $\boldsymbol{Z}_\star$ of the MCR^2 objective, characterized by Theorem 16.2. The values associated with the MCR^2 objective for features on different layers can be found in Figure 16.14 right. Empirically, we find that the constructed ReduNet is able to maximize MCR^2 loss and converges stably. Moreover, we sample new data points from the same distributions and find that new samples

[38] It is remarkable to see how easily this framework leads to working deep networks with thousands of layers! But this also indicates the efficiency of the layers is not so high. Given the optimization nature of the deep network, it would then be natural to expect that the acceleration techniques introduced in the earlier optimization chapters can be used to improve the efficiency of the layers (iterations). We leave this as an exercise to the reader.

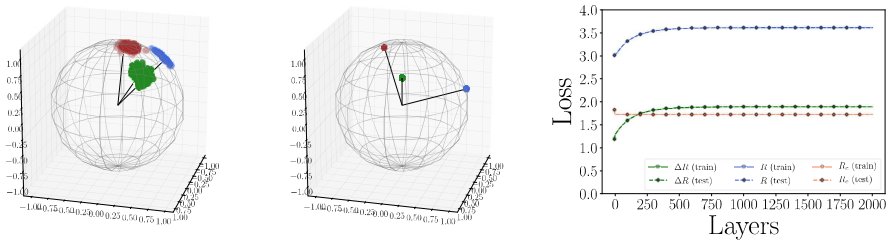


Figure 16.14 Original samples and learned representations for a mixture of three Gaussians in $\mathbb{R}^3$. We visualize data points $\boldsymbol{X}$ (before mapping) and features $\boldsymbol{Z}$ (after mapping) by scatter plots on the left and in the middle, respectively. In each scatter plot, each color represents one class of samples. We also show the plots for the progression of values of the objective function, for both training and testing data.

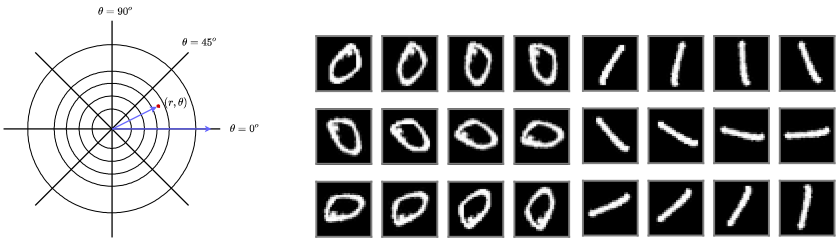


Figure 16.15 Examples of rotated images of MNIST digits for testing rotation invariance, each rotated by 18°. **Left**: diagram for polar coordinate representation; **Right**: rotated digit '0' and digit '1'.

form the same class consistently converge to the same cluster as the training samples.

Experiment I: 1D Rotational Invariance on MNIST Digits.
We study the ReduNet on learning *rotation* invariant features on the MNIST dataset [LeC98]. Examples of rotated images are shown in Figure 16.15. We impose a polar grid on the image $\boldsymbol{x} \in \mathbb{R}^{H \times W}$, with its geometric center being the center of the 2D polar grid. For each radius r_i, $i \in [C]$, we can sample Γ pixels with respect to each angle $\gamma_l = l \cdot (2\pi/\Gamma)$ with $l \in [\Gamma]$. Then given an image sample $\boldsymbol{x}$ from the dataset, we represent the image in a polar coordinate representation $\boldsymbol{x}(p) = (\gamma_{l,i}, r_{l,i}) \in \mathbb{R}^{\Gamma \times C}$.

Our goal is to learn rotation invariant features, i.e., we expect to learn $f(\cdot, \boldsymbol{\theta})$ such that $\{f(\boldsymbol{x}(p) \circ \mathfrak{g}, \boldsymbol{\theta})\}_{\mathfrak{g} \in \mathbb{G}}$ lie in the same subspace, where $\mathfrak{g}$ is the shift transformation in polar angle. By performing polar coordinate transformation for images from digit '0' and digit '1' in the training dataset, we can obtain the data matrix $\boldsymbol{X}(p) \in \mathbb{R}^{(\Gamma \cdot C) \times m}$. We use $m = 2,000$ training samples, set $\Gamma = 200$ and $C = 5$ for polar transformation, and set the number of iterations (or layers)

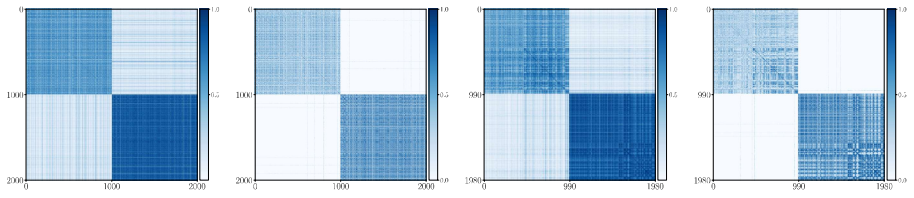


Figure 16.16 Cosine similarity (absolute value) of training/test data as well as traning/test representations for learning rotational invariant representations on MNIST. From left to right: $\boldsymbol{X}_{\text{train}}$, $\boldsymbol{Z}_{\text{train}}$, $\boldsymbol{X}_{\text{test}}$, and $\boldsymbol{Z}_{\text{test}}$.

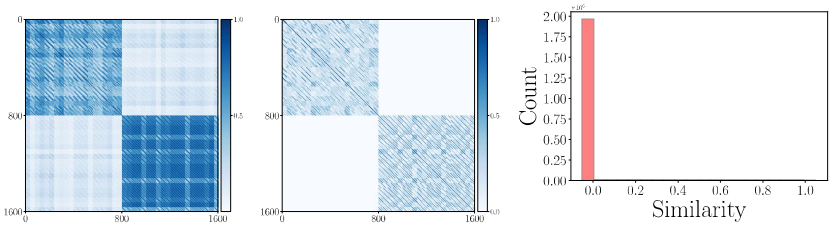


Figure 16.17 Heatmaps of cosine similarity between data $\boldsymbol{X}_{\text{shift}}$/learned features $\bar{\boldsymbol{Z}}_{\text{shift}}$ and histogram of cosine distance between shifted samples between the two classes.

$L = 3,500$, precision $\varepsilon = 0.1$, step-size $\eta = 0.5$. We randomly generate test samples with random rotations followed by the same procedure.

To visualize the effect of the feature mapping, we show cosine similarity (absolute value) of training/test data in Figure 16.16. We can see that the ReduNet is able to map nearly all random samples from different classes to orthogonal subspaces. To verify that the resulting representation is truly invariant for *all* rotations, we pick one sample from each class and augment the sample with every possible shifted ones, then calculate the cosine similarity between these augmented samples, shown on the left of Figure 16.17. Furthermore, we augment each samples in the dataset with its every possible shifted ones, then we evaluate the cosine similarity (in absolute value) between pairs across classes: for each pair, one sample is from training and one sample is from test which belong to different classes. The histogram of the cosine similarity is plotted on the right of Figure 16.17. We can clearly see that the learnt features are invariant to all shift transformation in polar angle (i.e., arbitrary rotation in $\boldsymbol{x}$).

We compare the accuracy (both on the original test data and the shifted test data) of the ReduNet (without considering invariance) and the shift invariant ReduNet. For the ReduNet (without considering invariance), we use the same training dataset as the shift invariant ReduNet, we set iteration $L = 3,500$, step size $\eta = 0.5$, and precision $\varepsilon = 0.1$. The results are summarized in Table 16.2.

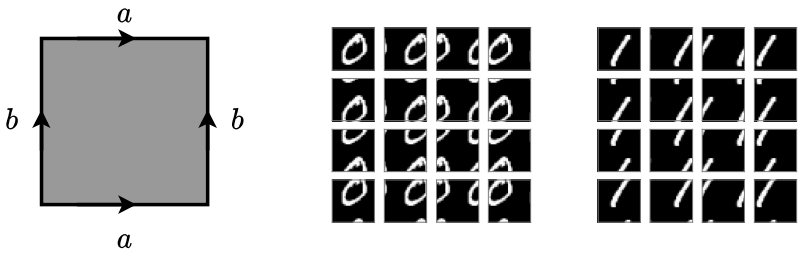


Figure 16.18 Examples of (cyclically) translated images of MNIST digits (with stride=7) for testing cyclic translation invariance of the ReduNet. **Left**: for cyclic 2D translation, we view a rectangular image as on a torus by identifying their opposite sides; **Right**: cyclic translated digit '0' and digit '1'.

With the invariant design, we can see from Table 16.2 that the shift invariant ReduNet achieves better performance in terms of invariance on the MNIST binary classification task.

Table 16.2 Comparing network performance on learning rotational-invariant representations on MNIST.

	ReduNet	ReduNet (invariant)
Acc (Original Test Data)	0.983	0.996
Acc (Test Data with All Possible Shifts)	0.707	0.993

Experiment II: 2D Cyclic Translation Invariance on MNIST Digits. In this part, we provide experimental results for verifying the invariance property of the ReduNet under 2D translations. We construct 1). a ReduNet (without considering invariance) and 2). a 2D translation-invariant ReduNet for classifying digit '0' and digit '1' on MNIST dataset. We use $m = 1,000$ samples (500 samples from each class) for training the models, and use another 500 samples (250 samples from each class) for evaluation. To evaluate the 2D translational invariance, for each test image $\boldsymbol{x}_{\text{test}} \in \mathbb{R}^{H \times W}$, we consider *all* translation augmentations of the test image with a stride=7. More specifically, for the MNIST dataset, we have $H = W = 28$. So for each image, the total number of all cyclic translation augmentations (with stride=7) is $4 \times 4 = 16$. Examples of translated images are shown in Figure 16.18. Notice that such translations are considerably larger than normally considered in the literature since we consider invariance to the entire group of cyclic translations on the $H \times W$ grid as a torus. See Figure 16.18 for some representative test samples.

For the ReduNet (without considering translation invariance), we set iteration $L = 2,000$, step size $\eta = 0.1$, and precision $\varepsilon = 0.1$. For the translation-invariant ReduNet, we set $L = 2,000$, step size $\eta = 0.5$, precision $\varepsilon = 0.1$, number of channels $C = 5$, and kernel size for the random lifting kernels in (16.3.24) is set

as 3×3.[39] We summarize the results in Table 16.3. Similar to the 1D rotational results on the MNIST dataset, the translation-invariant ReduNet achieves better performance under translations compared with the RedeNet without considering invariance. The accuracy drop of the translation-invariant ReduNet is much less than the one of the ReduNet without invariance design.

Table 16.3 Comparing network performance on learning 2D translation-invariant representations on MNIST.

	ReduNet	ReduNet (invariant)
Acc (Original Test Data)	0.980	0.975
Acc (Test Data with All Possible Shifts)	0.540	0.909

16.4 Guaranteed Manifold Classification by Deep Networks

The previous sections have shown how to construct (nonlinear) deep networks that embed labelled data into a union of incoherent subspaces, one per class. In contrast to our previous studies of linear and piecewise linear structure, these models can accommodate data that reside on *nonlinear* manifolds, by iteratively linearizing them. To a large extent, the constructive approach in the previous section reveals why a deep network architecture and many commonly adopted processes and operators are *necessary* for the classification task. However, due to the nonconvex nature of the objective and the greedy nature of the construction, there is yet no guarantee for the so obtained network to succeed in finding the optimal representation. This naturally raises the questions: When can data residing on nonlinear submanifolds be accurately classified by a deep network? What resources (data, network depth and width, training time) would be *sufficient* to correctly label the data? These questions are motivated both by the observed successes of deep networks in coping with nonlinear data, and the prevalence of nonlinear, low-dimensional structure in real data.

16.4.1 Minimal Case: Two 1D Submanifolds

In this section, we study this problem in what is arguably the simplest possible case: *two one-dimensional submanifolds on a high-dimensional sphere*. The experiment of classifying two digits, "0" and "1", with arbitrary rotation that we saw in Figure 16.15, can be viewed as an example of this problem. This is analogous to our discussion of dictionary learning in Chapter 7, where we illustrated

[39] Using more channels or better designed filters may certainly improve the performance. Here we choose the very basic ones just to verify the concept.

the basic ideas in the simplistic setting of one-sparse vectors, and extracted intuitions that carry over to more general situations.[40]

The precise setup is illustrated in Figure 16.19: we observe a finite set of labelled samples $\{(\boldsymbol{x}^i, y^i)\}_{i=1}^N$ residing on two one-dimensional submanifolds $\mathcal{M}_+$ and $\mathcal{M}_-$ on a high-dimensional sphere, and wish to understand what resources are needed to correctly label *every* point on $\mathcal{M}_+$ and $\mathcal{M}_-$. This is a strong form of *generalization* since it guarantees that the learned (or constructed) classifier $f(\boldsymbol{x}, \boldsymbol{\theta})$ outputs the correct label on every possible input.

Clearly, the resources required depend on the geometry of the manifolds, which here we capture through their curvature κ and separation Δ. We will describe how this question can be studied through the lens of supervised learning, where one fits a network to data by minimizing the loss on the training data:

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(\boldsymbol{x}^i, \boldsymbol{\theta}), y^i\big), \tag{16.4.1}$$

starting from a random initialization $\boldsymbol{\theta}_0$. This approach is, in a loose sense, dual to the approach taken in the previous sections: instead of constructing networks in the *forward* direction in order to minimize a loss, we start with a random network and train it by gradient descent, which propagates information about desired outputs *backward* through the network to determine how the parameters should be adjusted. Back propagation has been the dominant method for training deep networks [RHW86]. Nevertheless, we believe ultimately, these two approaches can (and should) be combined, e.g., the analytically constructed nominal weights of the network in the previous section can be further adjusted by gradient descent, potentially reducing the number of layers needed to embed the data on orthogonal subspaces.

One major challenge in analyzing neural network training arises from the nonconvexity of the objective $L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$. In the language of Chapter 7, deep networks exhibit complicated, compound symmetries (e.g., permutation or shift symmetries at each layer). We currently lack a comprehensive understanding of the optimization landscapes of deep networks. This has two implications for analysis: first, it is easier to analyze the training procedure in terms of the input-output relationship $\boldsymbol{x} \mapsto f(\boldsymbol{x}, \boldsymbol{\theta})$, rather than the weights themselves, which exhibit complicated symmetries. Second, rather than exhaustively characterizing local/global minimizers over the entire space, it is easier to analyze the dynamics of training starting from a random initial network $f(\cdot, \boldsymbol{\theta}_0)$. This enables us to bring tools from high-dimensional probability to bear on the problem: as the number of network parameters increases, the behavior of training becomes increasingly regular. Moreover, the initial distribution of the parameters can be chosen such that the layers of the network implement near isometries.

[40] Any fundamental ideas that work for general cases must be explained, arguably more clearly, for the most basic case first. Typically, going from 0 to 1 is the key step in advancing our knowledge, and after that, from 1 to n is merely natural extension.

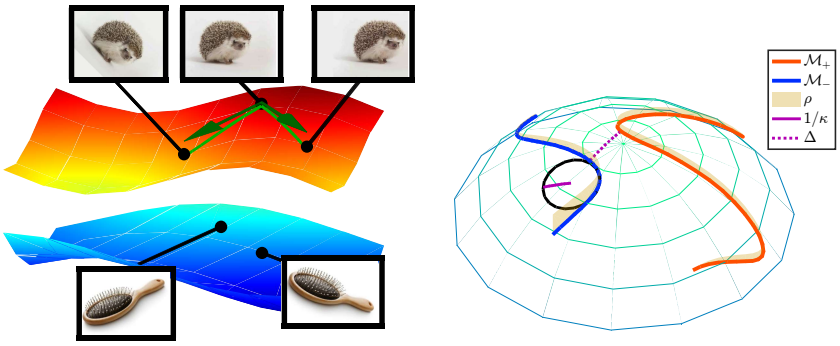


Figure 16.19 **Left:** data in image classification with standard augmentations, as well as other domains in which neural networks are commonly used, lies on low dimensional manifolds—in this case those generated by the action of continuous transformations, say rotation, on images in the training set. The dimension of the manifold is determined by the dimension of the symmetry group, and is typically small. **Right:** the *multiple manifold problem*. Our model problem, capturing this low dimensional structure, is the classification of low-dimensional submanifolds of a sphere $\mathbb{S}^{n_0-1}$. The difficulty of the problem is set by the inter-manifold separation Δ and the curvature κ. The depth and width of the network required to provably reduce the generalization error efficiently are set by these parameters.

16.4.2 Problem Formulation and Analysis

To make the above discussion more concrete, we consider a model network training problem, in which our labels take values in $\{\pm 1\}$, corresponding to the two components $\mathcal{M}_\pm$. Our goal is to fit a fully connected neural network to input data $\boldsymbol{x}^i$ of dimension n_0, with layers of width n, so that $\boldsymbol{W}_0 \in \mathbb{R}^{n\times n_0}$, $\boldsymbol{W}_\ell \in \mathbb{R}^{n\times n}$ for $\ell = 1, \ldots, L-2$ and $\boldsymbol{W}_{L-1} \in \mathbb{R}^{1\times n}$. We attempt to find these weights by minimizing the square loss over the training data:[41]

$$\min_{\boldsymbol{\theta}} \frac{1}{2N} \sum_{i=1}^{N} \big(f(\boldsymbol{x}^i, \boldsymbol{\theta}) - y^i\big)^2 = \int_{\boldsymbol{x}} \tfrac{1}{2}(f(\boldsymbol{x}, \boldsymbol{\theta}) - y(\boldsymbol{x}))^2 d\mu_N(\boldsymbol{x}), \tag{16.4.2}$$

where in the final expression, we have let $\mu_N(\boldsymbol{x}) = \frac{1}{N}\sum_i \delta(\boldsymbol{x} - \boldsymbol{x}^i)$ denote the measure (distribution) associated with the training data. Let $\zeta(\boldsymbol{x})$ denote the signed error at point $\boldsymbol{x}$:

$$\zeta(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{\theta}) - y(\boldsymbol{x}). \tag{16.4.3}$$

[41] In the two-class case, it is convenient to represent the two classes as ± 1, and the choice of squired loss, instead of cross entropy, is mainly for simplicity.

To understand how this error evolves during training, we can study a continuous time variant of gradient descent[42] in which the parameters evolve as

$$\begin{aligned}\frac{d}{dt}\boldsymbol{\theta}_t &= -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t, \boldsymbol{X}, \boldsymbol{Y}) = -\int_{\boldsymbol{x}} (f(\boldsymbol{x}, \boldsymbol{\theta}_t) - y(\boldsymbol{x})) \frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_t d\mu_N(\boldsymbol{x}) \\ &= -\int_{\boldsymbol{x}} \zeta_t(\boldsymbol{x}) \frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_t} d\mu_N(\boldsymbol{x}). \end{aligned} \tag{16.4.4}$$

As mentioned above, characterizing the evolution of the network parameters $\boldsymbol{\theta}$ themselves is challenging; often it is easier to think in terms of the error ζ_t, which evolves as

$$\begin{aligned}\frac{d}{dt}\zeta_t(\boldsymbol{x}) &= \frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_t} \frac{d}{dt}\boldsymbol{\theta}_t = -\int_{\boldsymbol{x}'} \left\langle \frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{x}', \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right\rangle\Big|_{\boldsymbol{\theta}_t} \zeta_t(\boldsymbol{x}') d\mu_N(\boldsymbol{x}') \\ &\doteq -\boldsymbol{\Theta}_t \zeta_t, \end{aligned} \tag{16.4.5}$$

where $\boldsymbol{\Theta}_t$ is a (linear) integral operator that maps a function $h(\boldsymbol{x})$ to

$$\int_{\boldsymbol{x}'} \left\langle \frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{x}', \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right\rangle h(\boldsymbol{x}') d\mu_N(\boldsymbol{x}'). \tag{16.4.6}$$

This operator is positive definite: for every h, we have $\langle h, \boldsymbol{\Theta}_t h \rangle_{\mu_N} \geq 0$ where $\langle f, g \rangle_{\mu_N} = \int_{\boldsymbol{x}} f(\boldsymbol{x}) g(\boldsymbol{x}) d\mu_N(\boldsymbol{x})$. This means that the error is nonincreasing:

$$\frac{d}{dt} \|\zeta_t\|^2_{L^2(\mu_N)} \leq 0,$$

with $\|f\|^2_{L^2(\mu_N)} = \langle f, f \rangle_{\mu_N}$.

How rapidly does the error reduce? This depends on the properties of the operator $\boldsymbol{\Theta}_t$ and the error ζ_t. $\boldsymbol{\Theta}_t$ is a positive definite linear operator, sometimes referred to as the *neural tangent kernel* [JGH18].[43] The entries $\boldsymbol{\Theta}_t(\boldsymbol{x}, \boldsymbol{x}')$ measure our ability to independently modify the outputs $f(\boldsymbol{x}, \boldsymbol{\theta})$ and $f(\boldsymbol{x}', \boldsymbol{\theta})$ at points $\boldsymbol{x}$ and $\boldsymbol{x}'$. If

$$|\boldsymbol{\Theta}_t(\boldsymbol{x}, \boldsymbol{x}')| \ll \min\{\boldsymbol{\Theta}_t(\boldsymbol{x}, \boldsymbol{x}), \boldsymbol{\Theta}_t(\boldsymbol{x}', \boldsymbol{x}')\},$$

the operator $\boldsymbol{\Theta}_t$ is close to diagonal, and it is possible to independently modify the two outputs with only small changes to the parameters $\boldsymbol{\theta}$.

The operator $\boldsymbol{\Theta}_t$ can also be studied both through its eigenvalue/eigenvector decomposition:

$$\boldsymbol{\Theta}_t = \sum_i \lambda_i v_i v_i^* \tag{16.4.7}$$

Because $\frac{d}{dt}\zeta_t = -\boldsymbol{\Theta}_t \zeta_t$, the error will decrease rapidly as long as it is aligned with eigenvectors v_i that correspond to *large* eigenvalues λ_i. Conversely, if the error is aligned with eigenvectors that correspond to *small* eigenvalues, it will decrease slowly.

[42] ... with the understanding that in this setting, conclusions transfer rigorously to discrete time (finite stepping) gradient methods.

[43] For the sake of developing intuition, it can be thought of as an infinitely large symmetric matrix.

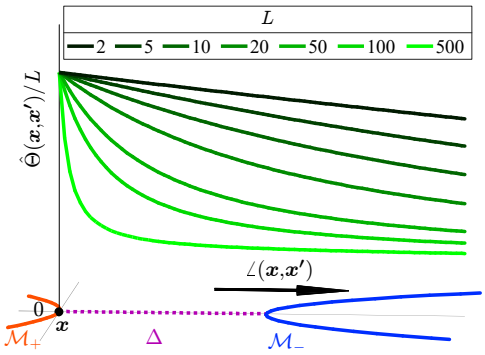


Figure 16.20 Role of Network Depth. As depth L increases, the kernel $\mathbf{\Theta}_0(\boldsymbol{x}, \boldsymbol{x}')$ becomes sharper, reflecting a greater capacity to fit functions that vary spatially. In our setup, the required depth is set by the separation Δ and the curvature κ of the manifolds $\mathcal{M}_\pm$.

We can develop insights into both values and eigenvalues by making the following idealizations: first, we consider the behavior of $\mathbf{\Theta}$ at initialization (time $t = 0$). At initialization, the network parameters are independent random variables, and $\mathbf{\Theta}_0$ is a random operator.[44] We can study its behavior using tools from high dimensional probability.[45] Second, we imagine that the network is wide (here, $n \gg n_0$). This means that $\mathbf{\Theta}_0$ is a function of *many* independent random variables. It should be no surprise that as the network width increases, this operator concentrates about its expectation. Moreover, this expectation depends on the points $\boldsymbol{x}$ and $\boldsymbol{x}'$ in a very simple way: because of the rotational invariance of the Gaussian distribution, it is not difficult to show that $\mathbb{E}[\mathbf{\Theta}_0(\boldsymbol{x}, \boldsymbol{x}')]$ depends on the points $\boldsymbol{x}$ and $\boldsymbol{x}'$ only through their angle:

$$\mathbb{E}[\mathbf{\Theta}_0(\boldsymbol{x}, \boldsymbol{x}')] = \xi_L\Big(\angle(\boldsymbol{x}, \boldsymbol{x}')\Big), \tag{16.4.8}$$

where L is the depth of the network. Figure 16.20 plots the function ξ_L as a function of the angle $\angle(\boldsymbol{x}, \boldsymbol{x}')$ for various network depths L. Notice that ξ_L is always maximized at $\angle(\boldsymbol{x}, \boldsymbol{x}') = 0$; as L increases, $\mathbf{\Theta}$ becomes sharper, suggesting that the network will be able to fit more complicated functions. Here, *depth L serves as an approximation resource*: deeper networks can fit more complicated functions. In the model problem of manifold classification, this suggests that greater curvature κ and the smaller the separation Δ, the deeper the network needs to be.

Second, the limiting expression $\mathbb{E}[\mathbf{\Theta}]$ provides insights into the eigenvectors and eigenvalues of $\mathbf{\Theta}$. Because $\mathbb{E}[\mathbf{\Theta}]$ is a function of angle only, in the spe-

[44] For concreteness, here we take the initial weights to be independent $\mathcal{N}(0, 2/n)$ random variables, and the nonlinearity ϕ to be the ReLU. These particular choices ensure that each layer implements a near isometry.

[45] In particular, because the network operations are applied sequentially, tools from Martingale theory are especially appropriate here.

cial case when the data are uniformly distributed on the unit sphere, $\mathbb{E}[\mathbf{\Theta}]$ is a (rotationally) invariant operator, which acts by *spherical convolution*. It can be diagonalized in the frequency domain[46], with large eigenvalues corresponding to low frequencies, and small eigenvalues corresponding to high frequencies. Of course, we are interested in data that are not uniformly distributed on the sphere – rather, natural data tend to have lower-dimensional structure. However these basic intuitions carry over to more structured situations: eigenvectors corresponding to large eigenvalues tend to be smoother or "lower frequency", while eigenvectors corresponding to small eigenvalues tend to be oscillatory, or "higher frequency". Assuming the error ζ is aligned with these "lower frequency" eigenvectors, gradient descent will rapidly drive the error toward zero.

The extent to which the error is aligned with "low-frequency" eigenvectors can be captured implicitly through a notion of "certificates": if we can exhibit ζ in the form $\zeta \approx \mathbf{\Theta} g$, where g is some function of small L^2 norm, then ζ must not be too concentrated on directions that correspond to small eigenvectors of $\mathbf{\Theta}$. This construction is loosely analogous to our constructions of dual certificates in Chapters 3-5. In those chapters, we proved recovery by convex optimization, by exhibiting a subgradient in the range of a certain random operator (the row-space of the measurement operators), using (random) measurements as an approximation resource. In a similar sense, here, we prove that gradient descent makes significant progress, by illustrating that the error ζ is near the range of a certain random operator $\mathbf{\Theta}$, using network depth and (random) parameters as approximation resources.

16.4.3 Main Conclusion

Summing up, in this setting we have the following resources:

- **Network depth** is a computation resource (via incremental approximation); deeper networks have sharper kernels $\mathbf{\Theta}$, which can fit more complicated functions, or adapt to more complicated geometries (larger κ, smaller Δ).
- **Network width** is a statistical resource; as width increases, the early behavior of training becomes increasingly regular, due to two effects: (i) concentration of $\mathbf{\Theta}_0$ about $\mathbb{E}[\mathbf{\Theta}_0]$, and (ii) the ability to make large progress in the objective before $\mathbf{\Theta}_t$ deviates from $\mathbf{\Theta}_0$. The later can be viewed as a consequence of overparameterization, analogous to our discussions of overparameterized low rank recovery in Chapters 4–7.
- **Data samples** are a statistical resource; as the number of samples increases, the learned network $f(\cdot, \boldsymbol{\theta})$ is more likely to uniformly label the manifolds $\mathcal{M}_{\pm}$. The number of samples N is set by the width of the kernel ξ_L: intuitively speaking, to generalize, we need the manifold to be covered more finely than the "aperture" of the kernel.

[46] More precisely, in terms of spherical harmonics.

Combining all of these considerations, it is possible identify (tractable) conditions under which gradient descent correctly labels the manifolds. For concision, we only sketch these conditions here:

THEOREM 16.6 (Sufficient Conditions for Manifold Classification (informal statement) [BGW20]). *Suppose that the network width* $n > \text{poly}(L \log(n_0))$*, the network depth* $L > \max\{\kappa^2, \text{polylog}(n_0)\}$*, and the number of samples* $N \geq \text{poly}(L)$*. If there exists* g *satisfying* $\|g\|_{L^2} \leq c/n$ *and* $\|\boldsymbol{\Theta}_0 g - \zeta_0\|_{L^2} < c/L$*, then with high probability randomly initialized gradient descent correctly labels every point on the manifolds* $\mathcal{M}_{\pm}$*.*

The work [BGW20] demonstrates how to construct such certificates g for simple geometries on the sphere, giving end-to-end guarantees for these simple classification problems. Although the results described in this section are limited in generality (pertaining to one-dimensional manifolds, with wide networks), they illustrate basic *sufficient conditions* for learning with structured data, and basic tensions between data properties, network architecture and sample complexity. Both at the level of proofs and at the level of phenomena, our intuitions from the first part of this book continue to serve us well in this new setting.

16.5 Epilogue: Open Problems and Future Directions

This chapter has sketched some fundamental and substantial connections between low-dimensional models and deep neural networks. This is an active area of research, with many open problems. As an epilogue of the chapter (and of the book), we lay out a number of promising directions for future work.

Simpler and Better Networks.
In practice, it is typically efficient (and even desirable from an accuracy perspective) to implement convolutional networks with very short or small (and separable) kernels [Cho17]. Identifying conditions on the data that lead naturally to short (and separable) convolutions is an interesting problem for future work. One possible conjecture is that when the data exhibit "short-and-sparse" structure, as in Chapter 12, the neural network naturally takes on a "short-and-sparse" structure. More generally, the problem of identifying simple networks is important both for efficiency of implementation and for robustness. As in the first part of this book, various notions of simplicity could be relevant, depending on the structure of the data and the processing task.

In the current straightforward implementation, the width of the ReduNet seems to grow linearly in the number of classes (i.e., the number of operators $\boldsymbol{C}^j$). Nevertheless, notice that $\boldsymbol{C}^j$ are not independent from $\boldsymbol{E}$. In fact, one can show that, near the optimal representation, the range of each $\boldsymbol{C}^j$ becomes an eigen-subspace of $\boldsymbol{E}$ [CYY+20]. This is consistent with the learning objective

discussed in Section 16.2 – seeking a nonlinear mapping for independent component analysis of the given data. Hence in practice, to save both space and computation, one could approximate $\boldsymbol{C}^j$ with a subset of operations from $\boldsymbol{E}$.

At a more theoretical level, the guarantees for classification described in Section 16.4 currently demand that the network be quite wide: n should be larger than a large degree polynomial in L. While we have motivated the need for wide networks in terms of concentration of measure, in fact it is possible to perform sharp analyses that show that the kernel $\boldsymbol{\Theta}$ concentrates uniformly over the (low-dimensional) data manifolds when the width is roughly $d\text{polylog}n_0$, which is optimal. Rather, the culprit in these analyses is the requirement that $\boldsymbol{\Theta}_t \approx \boldsymbol{\Theta}_0$. Relaxing this requirement seems to demand a better understanding of the (nonconvex) geometry of neural network training, in the spirit of Chapter 7.

Guaranteed Invariance.
Our discussion of networks-by-design in Section 16.3 revealed a tension between sparsity (low-dimensionality) and invariance (to certain transformation group). Understanding the interplay or tradeoff between these two different, ubiquitous forms of low-dimensional structure is an important direction both for neural networks and for low-dimensional data modeling in general. An important potential impact is to help in guaranteeing uniform performance across a large family of structured transformations, such as affine transforms, homographies, general smooth deformations, or dynamics from certain differential equations. Current standard approaches to this problem combine architectural features such as convolution and pooling with learning with augmented datasets (the parameters from random initialization). However, the literature is rich with alternative "networks-by-design" proposals (the ScatteringNet [BM13], spatial transformer networks [JSZK15], capsule networks [HKW11], convolutions with respect to larger groups [CW16], etc.). A major theoretical question underlying all of these approaches is *what resources (data, network, test-time computation) are required to achieve equivarient detection or invariant classification.*

Some of the most elegant proposed approaches incur either data or computational costs that are exponential in the number of parameters of the transformation or in the number of layers/iterations. Nevertheless, there are some evidences that the problem may not be *so* difficult: in some settings such as low-rank textures in Chapter 15, the TILT algorithm (via local optimization by repeatedly linearizing the transformation) achieves a surprisingly large region of convergence; the derivation of the ReduNet suggests that if we only learn an invariant/equivalent representation for dataset from a specific low-dimensional structure, the resources needed may scale gracefully to that of the task (say in terms of number of classes or data size). Hence, as the empirical success of TILT and ReduNet has suggested, it might be more practical to provide invariance guarantee for any given instance (rather than an entire family) of low-dimensional structures, in similar vein to Theorem 4.26 for low-rank matrix completion in Chapter 4.

Understanding Generalizability.
Modern deep neural networks are often highly over-parameterized models with more parameters than necessary to perfectly fit any training data [ZBH+17]. While the classical *bias-variance tradeoff* principle in statistics predicts that a large model leads to high variance error and overfitting [HTW15], modern practice with deep learning almost always favors models that are deeper, wider, and larger. Increasing studies have revealed that a fundamental reason for this is due to implicit regularization induced by the optimization algorithms [SHN+18, GLSS18, GBLJ19], the low-dimensional structures of the data [MWCC18], or both [YZQM20]. We believe a clear understanding of the generalizability of deep networks relies on a full understanding of over-parameterized models for nonlinear low-dimensional data structures such as submanifolds, in a similar spirit of understanding over-complete dictionaries for sparsity studied in this book,. This would require us to go well beyond the (bilinear) sparse dictionary learning or low-rank models discussed in Chapter 7.

Ensuring Robustness.
Despite extensive engineering, modern deep networks remain rather vulnerable to input perturbations, label noises, or adversarial attacks [CAD+18]. Empirical designs based on *trial and error* cannot provide any rigorous guarantee of robustness. Nevertheless, as we have seen throughout this book, from Boscovich's original proposal of ℓ^1 minimization, to Logan's phenomenon, to sparse error correction [CT05], and to dense error correction [WM10], surprisingly good tradeoffs between accuracy and robustness can be achieved if the corrupting errors are *incoherent* to low-dimensional structures of the data. The robust face recognition of Chapter 13 and structured texture recovery of Chapter 15 are two striking examples. It would be interesting to see whether one can generalize the notion of incoherent errors to low-dimensional submanifolds. If so, leveraging discriminative low-dimensional structures learned by the deep networks (e.g. the ReduNet) to provide strong guarantees of classification robustness (to mislabeled training data or/and to random corruptions on the input) could become a promising direction for future development.

A Unified Objective and Framework for Unsupervised Learning.
This chapter sketches an approach to deriving neural networks for classification with labeled training data, based on principles from data compression and compressive sensing. Note that the lossy coding and compression approach was originally developed for (unsupervised) clustering problems [MDHW07, VMS16] (also see equation (16.2.4)) and later extended to classification [WTL+08, KPCC15]: both mathematically are equivalent to maximizing the rate reduction against the membership $\boldsymbol{\Pi}$:

$$\max_{\boldsymbol{\Pi}} \Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \varepsilon) \tag{16.5.1}$$

with the representation $\boldsymbol{Z}$ given and fixed. So the rate reduction framework can be naturally extended to unsupervised learning of both representation and class membership, or a variety of intermediate settings such as semi-supervised learning, self-supervised learning, and incremental/online learning. The main technical challenge in these settings is that the class labels are partially or entirely unknown. So the membership $\boldsymbol{\Pi}$ need to be identified while learning the representation $\boldsymbol{Z}$:

$$\max_{\boldsymbol{Z},\boldsymbol{\Pi}} \Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \varepsilon), \tag{16.5.2}$$

with $\boldsymbol{Z} \subset \mathbb{S}^{n-1}$ and $\boldsymbol{\Pi} \in \Omega$ (or a constraint set based on partially known membership).

Recently, there have been promising attempts to simultaneously learn the representation $\boldsymbol{Z}$ and the class membership $\boldsymbol{\Pi}$ in the unsupervised setting with conventional deep networks [ARV20] and with contrastive learning objectives [CMM+20]. The rate reduction objective (16.5.2) might be able to unify both the learning objective and network architecture in this setting: following the same idea of the ReduNet, one may construct networks that emulate the joint (gradient flow) dynamics and optimize the representation $\boldsymbol{Z}$ and the membership $\boldsymbol{\Pi}$ simultaneously or alternatively:

$$\dot{\boldsymbol{Z}} = \eta \cdot \frac{\partial \Delta R}{\partial \boldsymbol{Z}}, \quad \dot{\boldsymbol{\Pi}} = \gamma \cdot \frac{\partial \Delta R}{\partial \boldsymbol{\Pi}}. \tag{16.5.3}$$

Of course, the basic gradient flow can be regularized with other additional information. For instance, the class membership $\boldsymbol{\Pi}$ can also be updated according to another (probably learned) similarity measure among the samples.[47] Success of such a scheme (or its variants) would entail understanding a nonconvex landscape with both continuous and discrete symmetries, which could potentially be studied through the lens of Chapter 7.

Forward Deep Networks as Optimization.
We want to point out that it is rather insightful and beneficial to view deep (forward) networks as unfolded or *unrolled optimization* schemes for optimizing rate reduction or other intrinsic measures of compactness, as depicted in Section 16.3. This allows us to utilize the rich arsenal of techniques from optimization to design and justify a variety of deep networks. Powerful ideas that we introduced in Chapters 8–9 (e.g. acceleration, alternating minimization, or augmented Lagrangian etc.) can be readily deployed to design effective optimization schemes that can in turn be emulated by deep networks. See Exercise 16.6 for a possible improvement of the ReduNet.

Similar to the above discussion on "*Understanding Generalizability*," one could

[47] For example, the "self-attention" or "transformer" type component [VSP+17] recently incorporated into deep networks can be viewed as actively learning similarity among the samples (or their features).

also study what *implicit regularization*

$$\mathcal{R}(\cdot) : f \mapsto \mathbb{R}_+$$

has been imposed upon the family of mappings $\mathcal{F} = \{f\}$ as they are constructed through the incremental gradient-based schemes as in the ReduNet. Or what additional regularization $\mathcal{R}(f)$ could have been imposed explicitly on the mapping f to make the representation learning problem better-defined – in the sense that the optimal representation $f_\star$ would have other desired properties (e.g. smoothness) as well as unique (or belong to a class of equivalent solutions)?

Backward Propagation as Variational Fine-Tuning.
The forward unrolling process depicted in Section 16.3 allows us to construct the deep network $f(\boldsymbol{x}, \boldsymbol{\theta}_0)$ – its architectures, operators, and parameters – as the *nominal* optimization path for the rate reduction ΔR. The popular back propagation for training deep networks [RHW86], analyzed in Section 16.4, can be viewed as variational methods for fine-tuning the network parameters $f(\boldsymbol{x}, \boldsymbol{\theta}_0 + d\boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{\theta}_0) + \delta f$, around the nominal path $f(\boldsymbol{x}, \boldsymbol{\theta}_0)$. The fine-tuning may achieve a better tradeoff between accuracy and efficiency of the nominal network (say when only a limited number of iterations, or layers, are allowed) [GEBS18] or to better customize the network to certain subsequent tasks or new data.

Nevertheless, for networks like the ReduNet whose operators and parameters have clear geometric and statistical interpretation, it remains open how to develop new back-propagation methods that respect structures and functionalities of these components (say compression or expansion) during fine-tuning. This can also be viewed as imposing certain additional regularization $\mathcal{R}$ (or constraints) onto the rate reduction objective:

$$\min_{f \in \mathcal{F}} \Delta R(f) + \lambda \cdot \mathcal{R}(f).$$

At least conceptually, by not allowing all the parameters to be set completely free for update, such regularization would help avoid over-fitting or help avoid the so-called "catastrophic forgetting" in the sequential or *incremental learning* setting [MC89, WBYM21].

Another potential advantage of this variational perspective for network fine-tuning is that it opens the door to employ rigorous and powerful tools from *calculus of variations* (e.g. [Lib12]) to study properties of the optimal mapping $f_\star$ (represented by a deep network):

$$\delta \Delta R(f) + \lambda \cdot \delta \mathcal{R}(f)\big|_{f_\star} = \mathbf{0}. \tag{16.5.4}$$

This may potentially leads to new ideas and variational algorithms for fine-tuning the network besides the conventional back propagation. There are already evidences that the forward-constructed ReduNet can be fine-tuned in a *forward propagation* fashion.

Sparse Coding, Spectral Computing, and Subspace Embedding in Nature. In this chapter, we have seen both sparse coding and spectral computing (or multi-channel convolution) arise naturally as *necessary* processes for effective and efficient classification of (visual) data invariant to translation. Recall that "sparse coding," as mentioned in Chapter 1, has been hypothesized as a guiding principle for the visual cortex of primates [OF96b]. Interestingly, there have also been strong scientific evidences that neurons in the visual cortex compute in the spectral domain: they encode and transmit information through the rate of spiking, hence called "spiking neurons" [SK93, EA03, BGM+08]. Recent studies in neuroscience have started to reveal how these mechanisms might be integrated in the inferotemporal (IT) cortex, where neurons encode and process information about high-level object identity (e.g. face recognition), invariant to various transformations [MHSD15, CT17]. The recent studies in [CT17] went even further to hypothesize that high-level neurons encode the face space as a "linear subspace" with each cell likely encoding one axis of the subspace (rather than previously thought "an exemplar"). The framework laid out in this chapter suggests that such a "high-level" compact (linear) representation can be efficiently and effectively learned via an arguably much simpler and more natural "forward propagation" mechanism.

So remarkably, nature might have already "learned" through millions years of evolution to exploit benefits of the mathematical principles depicted in this chapter, in particular the computational efficiency and simplicity in sparse coding, spectral computing, and subspace embedding for achieving invariant (visual) recognition! It remains a largely open, highly intriguing, question whether there will be concrete scientific evidences that suggest truly deep and broad connections between the guiding principles for Perception/Cognition and the computational principles for Data Compression/Representation developed in this chapter and this book. Regardless, we, the authors, strongly believe that *the law of parsimony*, a.k.a. Occam's Razor, has always been and will always be the central governing principle for all sciences and intelligences, artificial or natural. Hence, we leave the readers with a slogan:

We learn to compress, and compress to learn!

16.6 Exercises

16.1 (Properties of OLE). *Show that the OLE objective* (16.2.8) *is always* negative *and achieves the maximal value* 0 *when the subspaces are orthogonal, regardless of their dimensions.*

16.2 (Gradient of Rate Reduction). *Derive equation* (16.3.3) *and equation* (16.3.4) *from the definition of the rate reduction function* (16.3.1).

16.3 (Approximation of Regression Residual with ReLU). *Notice that the geometric meaning of $\boldsymbol{\sigma}$ in (16.3.12) is to compute the regression "residual" of each feature against the subspace to which it belongs. So when we restrict all features to be in the first (positive) quadrant of the feature space,*[48] *argue that one can approximate this residual using the rectified linear units operation, $ReLU(x) = \max(0, x)$, on $\boldsymbol{p}_j = \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell$ or its orthogonal complement:*

$$\boldsymbol{\sigma}(\boldsymbol{z}_\ell) \;\propto\; \boldsymbol{z}_\ell - \sum_{j=1}^{k} ReLU\big(\boldsymbol{P}_\ell^j \boldsymbol{z}_\ell\big), \tag{16.6.1}$$

where $\boldsymbol{P}_\ell^j = (\boldsymbol{C}_\ell^j)^\perp$ is the projection onto the j-th class[49]*. Discuss under what conditions or assumptions, the above approximate is good.*

16.4 ($\boldsymbol{E}$ and $\boldsymbol{C}^j$ as Convolutions). *Prove Proposition 16.5.*

16.5 (Benefits in the Spectral Domain). *Show that any circulant matrix can be diagonalized by the discrete Fourier transform $\boldsymbol{F}$:*

$$\mathsf{circ}(\boldsymbol{z}) = \boldsymbol{F}^* \mathrm{diag}\left(DFT(\boldsymbol{z})\right) \boldsymbol{F}. \tag{16.6.2}$$

Using this relationship, show that $\bar{\boldsymbol{E}}$ in (16.3.25) can be computed as

$$\bar{\boldsymbol{E}} = \begin{bmatrix} \boldsymbol{F}^* & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F}^* \end{bmatrix} \cdot \alpha \left(\boldsymbol{I} + \alpha \begin{bmatrix} \boldsymbol{D}_{11} & \cdots & \boldsymbol{D}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C1} & \cdots & \boldsymbol{D}_{CC} \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F} \end{bmatrix}, \tag{16.6.3}$$

where $\boldsymbol{D}_{cc'}$ are all diagonal matrices. Discuss how to exploit this structure to compute the inverse more efficiently.

16.6 (Network Architecture from Accelerated Gradient Methods). *Empirically, people have found that additional skip connections across multiple layers may improve the network performance, e.g. highway network [SGS15] or the DenseNet [HLVDMW17]. In the ReduNet, the role of each layer is precisely interpreted as one iterative gradient ascent step for the objective function ΔR. In the experiments, we have observed that the basic gradient scheme sometimes converges slowly, resulting in deep networks with thousands of layers (iterations)! To improve the efficiency of the basic ReduNet, one may consider in the accelerated gradient methods introduced in Chapters 8 and 9. Say to minimize or maximize a function $h(\boldsymbol{z})$, such accelerated methods usually take the form:*

$$\begin{cases} \boldsymbol{p}_{\ell+1} &= \boldsymbol{z}_\ell + \beta_\ell \cdot (\boldsymbol{z}_\ell - \boldsymbol{z}_{\ell-1}), \\ \boldsymbol{z}_{\ell+1} &= \boldsymbol{p}_{\ell+1} + \eta \cdot \nabla h(\boldsymbol{p}_{\ell+1}). \end{cases} \tag{16.6.4}$$

Sketch the resulting network architecture based on the accelerated gradient scheme, and verify empirically (say on the mixture of Gaussian or the handwritten digits) if the new architecture based on accelerated gradient would lead to faster convergence hence networks with fewer layers (or iterations).

[48] Most current neural networks seem to adopt this regime.

[49] $\boldsymbol{P}_\ell^j$ can be viewed as the orthogonal complement to $\boldsymbol{C}_\ell^j$.

16.7 (Programming with Deep Invariant ReduNet). *In Section 16.3.3, we have seen a basic example of constructing a ReduNet that is invariant to rotation (or translation) for two classes of digits, "0" and "1". Now knowing from "0" to "1," you are asked to take it from 1 to n in this exercise. This would allow you to gain some real experience with constructing ReduNets under more practical settings. Take 100 random samples from all 10 classes, 10 per class, from the MNIST dataset of 10 hand-written digits [LeC98].*

- *First, you get a chance to construct a ReduNet that can map these 100 samples to 10 orthogonal subspaces that are invariant to* all *rotation.*
  1. *First, convert each image to a multi-channel cyclic signal by following the same polar transform illustrated in Figure 16.15. Here choose* $\Gamma = 200$ *and* $C = 15$.
  2. *Second, try to lift these signals with a number of random Gaussian filters. Try a different number of filters in the range 10 to 30; or filter kernel size from 3 to 9 etc.*
  3. *Construct the rest of the ReduNet. Again, try your construction with different choices of key parameters: say quantization error* $\varepsilon \in [0.01, 0.5]$, *optimization step size* $\eta \in [0.1, 1]$, *and a number of layers* $L \in [20, 100]$.
- *Second, finalize and evaluate your resulting ReduNet:*
  1. *Monitor how the different rates* R, R_c *and* ΔR *evolve with the number of layers.*
  2. *Compute the cosine similarity for the training set, before and after the ReduNet mapping.*
  3. *Randomly select an independent set of 100 samples, 10 per class, and evaluate the effect of ReduNet on these new samples.*
- *Finally, some bonus tasks:*
  1. *Repeat for the task of constructing a ReduNet invariant to 2D translation for these digits. (Maybe you want to try a different range for the number of channels or layers.)*
  2. *Can you try to refine the so-obtained ReduNet network via back propagation by training it on the entire standard training set of MNIST? How would you evaluate what has gained (or lost) through such refinement?*

Appendices

Appendix A Facts from Linear Algebra and Matrix Analysis

"Everything is linear algebra."
– attributed to Gene H. Golub

Linear algebra studies linear systems of equations and their solutions. This topic is extremely important for engineering applications. Linear models represent a simple, tractable first choice for modeling complicated systems. Moreover, many devices for measuring the physical world are designed to produce measurements that are as close as possible to linear functions of the signal to be measured, such as the MR imaging studied in Chapter 10. Even if the measurements are nonlinear or the signals of interests have nonlinear structures, a common and effective practice in engineering is to approximate any nonlinearality with a sequence of (local) linearization, as we see in Chapter 15 for recovering deformed low-rank textures and in Chapter 16 for learning submanifolds with a deep network.

In early parts of this appendix, we review several fundamental definitions, constructions, and facts from linear algebra and matrix analysis. For readers with a background in engineering, statistics or applied mathematics, much of this material is likely to be familiar. They may use this appendix and the next few to refresh their memory and get familiar with the notation used in this book. Section A.9 contains a brief review of norms on matrices and spectral functions of matrices, two more advanced topics which we use extensively throughout the book. We have attempted to make this introduction as simple and self-contained as possible; readers looking for a more thorough introduction to this area could consult the excellent books of Horn and Johnson [HJ85], Golub and Van Loan [GV96], or Bhatia [Bha96].[1]

[1] Or Boyd and Vandenberghe [BV18] for a more elementary introduction.

A.1 Vector Spaces, Linear Independence, Bases and Dimension

We use the notation $\mathbb{R}$ for the real numbers, and $\mathbb{R}^n$ for the n-dimensional real vectors of the form:

$$\boldsymbol{x} \equiv \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \quad \boldsymbol{x}^* = [x_1, \ldots, x_n] \in \mathbb{R}^n, \tag{A.1.1}$$

where in this book, we use $\boldsymbol{x}^*$ to denote the transpose of a column vector $\boldsymbol{x}$. In case the vector is complex, it represents the conjugate transpose. The space $\mathbb{R}^n$ is an example of a vector space – a space in which we can perform addition and scalar multiplication in a way that conforms to our intuition from $\mathbb{R}^3$. More formally:

Definition A.1 (Vector Space). *A* vector space $\mathbb{V}$ *over a field of scalars* $\mathbb{F}$ *is a set* $\mathbb{V}$ *(with a special distinguished zero element* $\mathbf{0} \in \mathbb{V}$*) endowed with two operations:*

- **vector addition** $+$*, which takes two vectors* $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{V}$ *and produces another vector* $\boldsymbol{v} + \boldsymbol{w} \in \mathbb{V}$*,*
- **scalar multiplication***, which takes a vector* $\boldsymbol{v} \in \mathbb{V}$ *and a scalar* $\alpha \in \mathbb{F}$*, and produces a vector* $\alpha\boldsymbol{v} \in \mathbb{V}$*,*

such that (1) addition is associative: $\boldsymbol{v} + (\boldsymbol{w} + \boldsymbol{x}) = (\boldsymbol{v} + \boldsymbol{w}) + \boldsymbol{x}$*, (2) addition is commutative:* $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$*, (3) zero is the additive identity:* $\boldsymbol{v} + \mathbf{0} = \boldsymbol{v}$*, (3) every element has an additive inverse: for each* $\boldsymbol{v} \in \mathbb{V}$*, there exists an element "*$-\boldsymbol{v}$*"* $\in \mathbb{V}$ *such that* $\boldsymbol{v} + (-\boldsymbol{v}) = \mathbf{0}$*, (5)* $\alpha(\beta\boldsymbol{v}) = (\alpha\beta)\boldsymbol{v}$*, (6) multiplicative identity:* $1\boldsymbol{v} = \boldsymbol{v}$*, where* $1 \in \mathbb{F}$ *is the multiplicative identity in* $\mathbb{F}$*, (7)* $\alpha(\boldsymbol{v} + \boldsymbol{w}) = \alpha\boldsymbol{v} + \alpha\boldsymbol{w}$*, (8)* $(\alpha + \beta)\boldsymbol{v} = \alpha\boldsymbol{v} + \beta\boldsymbol{v}$*.*

Example A.2. *The following are examples of vector spaces (check this!)*

- *The* n*-dimensional real vectors* $\mathbb{R}^n$*, over the scalar field* $\mathbb{F} = \mathbb{R}$*.*
- *The* $m \times n$ *real matrices*

$$\mathbb{R}^{m\times n} \doteq \left\{ \boldsymbol{X} = \begin{bmatrix} X_{11} & \ldots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \ldots & X_{mn} \end{bmatrix} \,\middle|\, X_{ij} \in \mathbb{R} \right\}, \tag{A.1.2}$$

  over the scalar field $\mathbb{F} = \mathbb{R}$*.*
- *The complex vectors* $\mathbb{C}^n$ *or complex matrices* $\mathbb{C}^{m\times n}$*, over the scalar field* $\mathbb{F} = \mathbb{C}$*.*
- *Function spaces, e.g.,*

$$\mathcal{C}^0[0,1] \doteq \{ f : [0,1] \to \mathbb{R} \mid \ f \text{ continuous} \}, \tag{A.1.3}$$

  over $\mathbb{R}$*. Vector spaces of functions defined on the continuum arise naturally in the study sampling problems, in which we wish to derive information about the physical world from digital measurements.*

By itself, the notion of a vector space is not particularly rich: it is simply a space in which linear operations make sense. A vector space can be viewed as the "playing field" on which much more interesting models can be built, and much richer questions can be asked. As a step in this direction, we can note that it makes sense to take linear combinations of elements of a vector space. A *linear combination* is an expression of the form

$$\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_k \boldsymbol{v}_k,$$

where $\alpha_1, \ldots, \alpha_k \in \mathbb{F}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{V}$.

Definition A.3 (Linear Independence). *A set of vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ *are* linearly independent *if*

$$\sum_{i=1}^{k} \alpha_i \boldsymbol{v}_i = \boldsymbol{0} \quad \Longrightarrow \quad \alpha_1 = 0, \ldots, \alpha_k = 0.$$

If a collection of vectors are not linearly independent, then there exists some choice of (α_i) not all zero, for which $\sum_i \alpha_i \boldsymbol{v}_i = \boldsymbol{0}$. In this case, we say that the set $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ is *linearly dependent.*

Definition A.4 (Basis for a Vector Space). *A* basis B *for the vector space* $\mathbb{V}$ *is defined as a* maximal, linearly independent set.

Here, *maximal* means that B is not contained in any larger linearly independent set. Any basis B for $\mathbb{V}$ *spans* $\mathbb{V}$, in the sense that every element of $\mathbb{V}$ can be written as a linear combination of elements of B:

$$\forall\, \boldsymbol{v} \in \mathbb{V},\ \exists\, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathsf{B},\ \alpha_1, \ldots, \alpha_k \in \mathbb{F}, \quad \text{such that} \quad \boldsymbol{v} = \sum_{i=1}^{k} \alpha_i \boldsymbol{b}_i. \quad \text{(A.1.4)}$$

Moreover, if B is a basis, the coefficients $\alpha_1, \ldots, \alpha_k$ in the above expression are unique.

Example A.5. *In* $\mathbb{R}^n$*, we often use the* standard basis $\mathsf{B} = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ *of coordinate vectors*

$$\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad \boldsymbol{e}_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \qquad \text{(A.1.5)}$$

In $\mathbb{R}^{m \times n}$ *we may work with the standard basis of coordinate matrices* $\boldsymbol{E}_{ij}$ *that are one in entry* (i, j) *and zero elsewhere.*

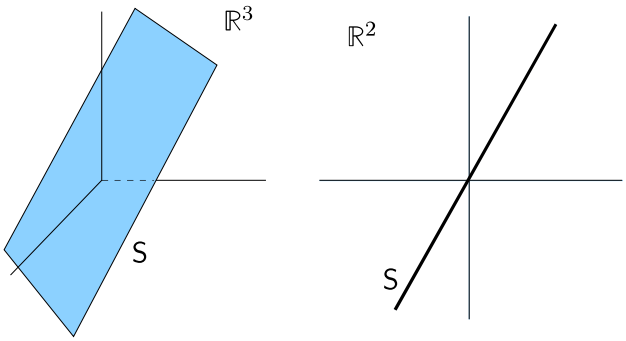


Figure A.1 Linear subspaces of $\mathbb{R}^2$ and $\mathbb{R}^3$.

Every vector space $\mathbb{V}$ has a basis.[2] One very fundamental result in linear algebra states that every basis has the same size:

THEOREM A.6 (Invariance of Dimension). *For any vector space* $\mathbb{V}$*, every basis* B *has the same cardinality, which we denote* $\dim(\mathbb{V})$*, and call the* dimension *of* $\mathbb{V}$*.*

The notion of dimension is especially useful for talking about subspaces of the vector space $\mathbb{V}$.

DEFINITION A.7 (Linear Subspace). *A* linear subspace *of a vector space* $\mathbb{V}$ *is a set* $\mathsf{S} \subseteq \mathbb{V}$ *that is also a vector space.*

For $\mathsf{S} \subseteq \mathbb{V}$ to be a linear subspace, it is necessary and sufficient that S be stable under linear combinations: for all $\alpha, \beta \in \mathbb{F}$ and $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathsf{S}$, $\alpha\boldsymbol{v}_1 + \beta\boldsymbol{v}_2 \in \mathsf{S}$. Linear subspaces play a very important dual role, both as cleanly characterizing the solvability of linear equations, and as geometric data models. Geometrically, we can visualize a subspace as a generalization of a line, or plane, which must pass the origin: $\mathbf{0} \in \mathsf{S}$ (see Figure A.1).

A.2 Inner Products

The most important geometric relationship between subspaces is that of *orthogonality*. To describe it clearly, we need the notion of an inner product. Below, we will assume that we are working with a vector space over either the real or complex numbers, and so the complex conjugate $\bar{\alpha}$ of $\alpha \in \mathbb{F}$ is well-defined.

DEFINITION A.8 (Inner Product). *A function* $\langle \cdot, \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$ *is an* inner product *if it satisfies:*

[2] This statement may seem obvious, but is tricky: it turns out to be equivalent to the *axiom of choice* in set theory, and hence is best viewed as an assumption. For the vector spaces we consider in this course ($\mathbb{R}^n$, $\mathbb{C}^n$, ect.), it will be very easy to construct a basis, and so for our purposes, the question is essentially moot.

- **linearity**: $\langle \alpha \boldsymbol{v} + \beta \boldsymbol{w}, \boldsymbol{x} \rangle = \alpha \langle \boldsymbol{v}, \boldsymbol{x} \rangle + \beta \langle \boldsymbol{w}, \boldsymbol{x} \rangle$;
- **conjugate symmetry** $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \overline{\langle \boldsymbol{w}, \boldsymbol{v} \rangle}$;
- **positive definiteness** $\langle \boldsymbol{v}, \boldsymbol{v} \rangle \geq 0$, *with equality iff* $\boldsymbol{v} = \boldsymbol{0}$.

We then say that $\boldsymbol{v}$ and $\boldsymbol{w}$ are *orthogonal* (with respect to inner product $\langle \cdot, \cdot \rangle$) if $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0$. In this case, we write $\boldsymbol{v} \perp \boldsymbol{w}$. For a given set $\mathsf{S} \subseteq \mathbb{V}$, we define its orthogonal complement as the set of all vectors that are orthogonal to every element of S:

Definition A.9 (Orthogonal Complement). *For* $\mathsf{S} \subseteq \mathbb{V}$,

$$\mathsf{S}^{\perp} = \{\boldsymbol{v} \in \mathbb{V} \mid \langle \boldsymbol{v}, \boldsymbol{s} \rangle = 0 \; \forall \, \boldsymbol{s} \in \mathsf{S}\} .$$

It is worth noting that for any set S, $\mathsf{S}^{\perp} \subseteq \mathbb{V}$ is a linear subspace. This holds even if S is not a subspace itself.

We will use (and return to) two main examples of inner products. The first is the canonical inner product on $\mathbb{R}^n$, which simply sets

$$\langle \boldsymbol{x}, \boldsymbol{z} \rangle = \sum_{i=1}^{n} x_i z_i . \tag{A.2.1}$$

This extends to a canonical inner product on $\mathbb{R}^{m \times n}$, which is sometimes called the Frobenius inner product:

$$\langle \boldsymbol{X}, \boldsymbol{Z} \rangle \doteq \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} Z_{ij} . \tag{A.2.2}$$

Recall that the trace of a square matrix is simply the sum of its diagonal elements:

Definition A.10. *For* $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, $\operatorname{trace}(\boldsymbol{M}) = \sum_{i=1}^{n} M_{ii}$.

Using the trace, we can give an expression for the Frobenius inner product which appears more complicated, but actually turns out to be tremendously useful:

$$\langle \boldsymbol{X}, \boldsymbol{Z} \rangle = \operatorname{trace}(\boldsymbol{X}^* \boldsymbol{Z}) = \operatorname{trace}(\boldsymbol{X} \boldsymbol{Z}^*) . \tag{A.2.3}$$

For manipulating this expression, it is worth noting that the trace is invariant under a cyclic permutation of its argument:

Theorem A.11. *For any matrices* $\boldsymbol{A}$, $\boldsymbol{B}$ *of compatible size,* $\operatorname{trace}(\boldsymbol{A}\boldsymbol{B}) = \operatorname{trace}(\boldsymbol{B}\boldsymbol{A})$. *More generally, if* $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n$ *are matrices of compatible size, and* π *is a cyclic permutation on* $\{1, \ldots, n\}$,

$$\operatorname{trace}(\boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_n) \;=\; \operatorname{trace}\left(\boldsymbol{A}_{\pi(1)} \boldsymbol{A}_{\pi(2)} \cdots \boldsymbol{A}_{\pi(n)}\right) . \tag{A.2.4}$$

A.3 Linear Transformations and Matrices

A mapping $\mathcal{L}$ between vector spaces $\mathbb{V}$ and $\mathbb{V}'$ over a common field $\mathbb{F}$ is a *linear transformation* (or linear map) if it respects the vector space operations:

Definition A.12 (Linear Map). *A* linear map *is a function* $\mathcal{L} : \mathbb{V} \to \mathbb{V}'$ *such for all* $\alpha, \beta \in \mathbb{F}$ *and* $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{V}$, $\mathcal{L}[\alpha\boldsymbol{v} + \beta\boldsymbol{w}] = \alpha\mathcal{L}[\boldsymbol{v}] + \beta\mathcal{L}[\boldsymbol{w}]$.

If $\mathbb{V}' = \mathbb{V}$ then we call $\mathcal{L}$ a *linear operator*.

Example A.13. *Let* $\mathbb{V} = \mathbb{R}^{m\times n}$, *and* $\Omega \subseteq \{1,\dots,m\} \times \{1,\dots,n\}$. *Let* $\mathcal{P}_\Omega : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ *via*

$$(\mathcal{P}_\Omega[\boldsymbol{X}])_{ij} = \begin{cases} X_{ij} & (i,j) \in \Omega, \\ 0 & \text{else}, \end{cases} \tag{A.3.1}$$

i.e., the restriction of $\boldsymbol{X}$ *to* Ω. *Then* $\mathcal{P}_\Omega$ *is a linear operator.*

The special case of $\mathbb{V} = \mathbb{R}^n$, $\mathbb{V}' = \mathbb{R}^m$ is of special importance. It turns out that there is a bijective correspondence between linear operators $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$ and $m \times n$ matrices:

Theorem A.14. *For* $\boldsymbol{x} \in \mathbb{R}^n$ *and* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, *let*

$$(\boldsymbol{A}\boldsymbol{x})_i = \sum_j A_{ij}x_j. \tag{A.3.2}$$

Then for every $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, *the mapping* $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}$ *is a linear map from* $\mathbb{R}^n$ *to* $\mathbb{R}^m$. *Conversely for every linear map* $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$ *there exists a unique* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ *such that for every* $\boldsymbol{x}$, $\mathcal{L}[\boldsymbol{x}] = \boldsymbol{A}\boldsymbol{x}$.

This fact justifies the seemingly awkward standard definition of matrix multiplication – it is simply the correct way of representing the composition of two linear maps:

Theorem A.15. *If* $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^p$ *and* $\mathcal{L}' : \mathbb{R}^p \to \mathbb{R}^m$ *are linear maps, with corresponding matrix representations* $\boldsymbol{A} \in \mathbb{R}^{p\times n}$ *and* $\boldsymbol{A}' \in \mathbb{R}^{m\times p}$, *and* $\mathcal{L}' \circ \mathcal{L}$ *denotes the composition* $\mathcal{L}' \circ \mathcal{L}(\boldsymbol{x}) = \mathcal{L}'[\mathcal{L}[\boldsymbol{x}]]$, *then* $\mathcal{L}' \circ \mathcal{L}$ *is a linear map, and its matrix representation is given by the matrix product* $\boldsymbol{A}'\boldsymbol{A}$ *whose* (i,j) *entry is*

$$\left(\boldsymbol{A}'\boldsymbol{A}\right)_{ij} = \sum_{k=1}^{p} a'_{ik}a_{kj}. \tag{A.3.3}$$

The (conjugate) transpose of a matrix $\boldsymbol{A} \in \mathbb{C}^{m\times n}$ is the $n \times m$ matrix $\boldsymbol{A}^* \in \mathbb{C}^{n\times m}$ given by:

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} \end{bmatrix} \Rightarrow \boldsymbol{A}^* = \begin{bmatrix} \overline{A_{11}} & \dots & \overline{A_{m1}} \\ \vdots & \ddots & \vdots \\ \overline{A_{1n}} & \dots & \overline{A_{mn}} \end{bmatrix}. \tag{A.3.4}$$

When $\boldsymbol{A}$ is real, this is just the transpose. Transposition is a very simple operation on the entries of a matrix, but it has a basic reason for existing:

THEOREM A.16. *Let $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, with corresponding matrix $\boldsymbol{A}$. Its adjoint map is the unique linear map $\mathcal{L}^* : \mathbb{R}^m \to \mathbb{R}^n$ satisfying*

$$\forall\, \boldsymbol{x},\, \boldsymbol{y}, \qquad \langle \boldsymbol{y}, \mathcal{L}[\boldsymbol{x}] \rangle = \langle \mathcal{L}^*[\boldsymbol{y}], \boldsymbol{x} \rangle\,. \tag{A.3.5}$$

The matrix $\boldsymbol{A}^*$ is the matrix representation of the adjoint map $\mathcal{L}^*$.

A linear map $\mathcal{L} : \mathbb{V} \to \mathbb{V}'$ is *invertible* if for every $\boldsymbol{y} \in \mathbb{V}'$, there is a unique $\boldsymbol{x} \in \mathbb{V}$ such that $\mathcal{L}[\boldsymbol{x}] = \boldsymbol{y}$. In particular, if $\mathbb{V} = \mathbb{V}' = \mathbb{R}^n$, we call $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ invertible if it corresponds to an invertible linear map. This means that the system of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \tag{A.3.6}$$

always has a unique solution

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}. \tag{A.3.7}$$

It is not too difficult to show that if $\mathcal{L}$ is a linear map, its inverse $\mathcal{L}^{-1}$ is also linear. So, the notation $\boldsymbol{A}^{-1}$ above can be taken to mean "the matrix representation of the inverse mapping $\mathcal{L}^{-1}$". Fortunately, there are much more concrete criteria for determining if a given matrix $\boldsymbol{A}$ is invertible, and if so, for calculating $\boldsymbol{A}^{-1}$.

DEFINITION A.17 (Determinant). *The determinant of $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the signed volume of the parallelepiped defined by the columns of $\boldsymbol{A}$:*

$$\det(\boldsymbol{A}) \;=\; \sum_{\pi \text{ a permutation on } \{1,\ldots,n\}} \operatorname{sgn}(\pi) \times \prod_{i=1}^{n} A_{i,\pi(i)}, \tag{A.3.8}$$

The explicit expression (A.3.8) is not usually of direct use. More important is the geometric intuition: if $\det(\boldsymbol{A})$ is zero, the columns of $\boldsymbol{A}$ span a parallelpiped of zero volume, and so they lie on some lower dimensional subspace of $\mathbb{R}^n$. Vectors $\boldsymbol{y}$ that do not reside in this subspace cannot be generated as linear combinations of the columns of $\boldsymbol{A}$, and $\boldsymbol{A}$ is not invertible. Conversely, if $\det \boldsymbol{A} \neq 0$, the columns of $\boldsymbol{A}$ span all of $\mathbb{R}^n$, and $\boldsymbol{A}$ is invertible. Making this reasoning formal, one obtains

THEOREM A.18 (Matrix Inverse). *A matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is invertible if and only if $\det \boldsymbol{A} \neq 0$. If $\boldsymbol{A}$ is invertible, we can express its inverse as $\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})}\boldsymbol{C}$, where $\boldsymbol{C} \in \mathbb{R}^{n\times n}$ is the companion matrix:*

$$\boldsymbol{C} \doteq \begin{bmatrix} (-1)^{1+1}\det(\boldsymbol{A}_{\setminus 1,\setminus 1}) & (-1)^{1+2}\det(\boldsymbol{A}_{\setminus 2,\setminus 1}) & \cdots & (-1)^{1+n}\det(\boldsymbol{A}_{\setminus n,\setminus 1}) \\ (-1)^{2+1}\det(\boldsymbol{A}_{\setminus 1,\setminus 2}) & (-1)^{2+2}\det(\boldsymbol{A}_{\setminus 2,\setminus 2}) & \cdots & (-1)^{2+n}\det(\boldsymbol{A}_{\setminus n,\setminus 2}) \\ \vdots & \vdots & \ddots & \vdots \\ (-1)^{n+1}\det(\boldsymbol{A}_{\setminus 1,\setminus n}) & (-1)^{n+2}\det(\boldsymbol{A}_{\setminus 2,\setminus n}) & \cdots & (-1)^{n+n}\det(\boldsymbol{A}_{\setminus n,\setminus n}) \end{bmatrix},$$

where the matrix $\boldsymbol{A}_{\setminus i,\setminus j}$ is constructed from $\boldsymbol{A}$ by removing the i-th row and j-th column.

Again, the above expression for $\boldsymbol{A}^{-1}$ is of little use computationally, but is conceptually helpful, since it shows that the entries of the inverse are rational functions of the entries of $\boldsymbol{A}$.

It is worth noting that for any matrices $\boldsymbol{A}$ and $\boldsymbol{B}$,

$$\det(\boldsymbol{AB}) = \det(\boldsymbol{A})\det(\boldsymbol{B}). \tag{A.3.9}$$

This corroborates the fact that a product of invertible linear maps is invertible, and a product of invertible matrices is invertible. In particular,

$$(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}. \tag{A.3.10}$$

It is also useful to note that for every matrix $\boldsymbol{A}$,

$$\det(\boldsymbol{A}) = \det(\boldsymbol{A}^*). \tag{A.3.11}$$

A.4 Matrix Groups

Because the product of two $n \times n$ matrices is again an $n \times n$ matrix, this operation can produce objects with interesting algebraic structure. We will not emphasize the algebra of matrix groups – or even formally define a group. Rather, we just recall the names of several groups that will recur throughout the course:

- **The general linear group** $\mathsf{GL}(n, \mathbb{R})$ consists of the invertible matrices:

$$\mathsf{GL}(n, \mathbb{R}) = \left\{ \boldsymbol{A} \in \mathbb{R}^{n \times n} \mid \det(\boldsymbol{A}) \neq 0 \right\}. \tag{A.4.1}$$

  Similarly, $\mathsf{GL}(n, \mathbb{C})$ denotes the $n \times n$ invertible matrices with complex entries.
- **The orthogonal group** $\mathsf{O}(n)$ consists of the real $n \times n$ matrices that satisfy $\boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^* = \boldsymbol{I}$:

$$\mathsf{O}(n) = \left\{ \boldsymbol{A} \in \mathbb{R}^{n \times n} \mid \boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I} \right\}. \tag{A.4.2}$$

  The expression $\boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I}$ implies that $\boldsymbol{A}$ is invertible, and that $\boldsymbol{A}^{-1} = \boldsymbol{A}^*$. Hence, $\mathsf{O}(n) \subset \mathsf{GL}(n, \mathbb{R})$. Two notes are in order: first, since $\boldsymbol{I} = \boldsymbol{I}^* = (\boldsymbol{A}^*\boldsymbol{A})^* = \boldsymbol{A}\boldsymbol{A}^*$, it is enough to keep only the expression $\boldsymbol{A}^*\boldsymbol{A}$ in the definition. Second, because $\det(\boldsymbol{A}) = \det(\boldsymbol{A}^*)$, we have $\det(\boldsymbol{A})^2 = 1$, and so every $\boldsymbol{A} \in \mathsf{O}(n)$ has determinant ± 1.
- **The special orthogonal group** $\mathsf{SO}(n)$ consists of the $n \times n$ matrices that satisfy $\boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^* = \boldsymbol{I}$, *and* $\det(\boldsymbol{A}) = +1$:

$$\mathsf{SO}(n) = \left\{ \boldsymbol{A} \in \mathbb{R}^{n \times n} \mid \boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I},\ \det(\boldsymbol{A}) = +1 \right\}. \tag{A.4.3}$$

  Clearly, $\mathsf{SO}(n) \subset \mathsf{O}(n) \subset \mathsf{GL}(n, \mathbb{R})$. In $\mathbb{R}^3$, the group $\mathsf{SO}(3)$ corresponds to the rotation matrices; $\mathsf{O}(3)$ contains rotations and reflections.
- **The unitary and special unitary groups** are subgroups of $\mathsf{GL}(n, \mathbb{C})$. The unitary group $\mathsf{U}(n)$ contains those matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ satisfying $\boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I}$. The special unitary group $\mathsf{SU}(n)$ contains those $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ satisfying $\boldsymbol{A}^*\boldsymbol{A} = \boldsymbol{I}$ and $\det(\boldsymbol{A}) = 1$. So, $\mathsf{SU}(n) \subset \mathsf{U}(n) \subset \mathsf{GL}(n, \mathbb{C})$.

A.5 Subspaces Associated with a Matrix

To each linear operator $\mathcal{L} : \mathbb{V} \to \mathbb{V}'$, we associate two important subspaces, the range and the null space:

DEFINITION A.19 (Range, null space). *For $\mathcal{L} : \mathbb{V} \to \mathbb{V}'$,*

$$\operatorname{range}(\mathcal{L}) = \{\mathcal{L}[\boldsymbol{x}] \mid \boldsymbol{x} \in \mathbb{V}\} \subseteq \mathbb{V}', \tag{A.5.1}$$

$$\operatorname{null}(\mathcal{L}) = \{\boldsymbol{x} \in \mathbb{V} \mid \mathcal{L}[\boldsymbol{x}] = \boldsymbol{0}\} \subseteq \mathbb{V}. \tag{A.5.2}$$

The range is a linear subspace of $\mathbb{V}'$, while the null space is a linear subspace of $\mathbb{V}$.

Specializing these definitions to $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, represented by a matrix $\boldsymbol{A}$, we obtain

$$\operatorname{null}(\boldsymbol{A}) = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}, \tag{A.5.3}$$

$$\operatorname{range}(\boldsymbol{A}) = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n\} \;=\; \operatorname{col}(\boldsymbol{A}), \tag{A.5.4}$$

$$\operatorname{row}(\boldsymbol{A}) = \{\boldsymbol{w}^*\boldsymbol{A} \mid \boldsymbol{w} \in \mathbb{R}^m\}. \tag{A.5.5}$$

The sets $\operatorname{null}(\boldsymbol{A})$, $\operatorname{range}(\boldsymbol{A})$ and $\operatorname{row}(\boldsymbol{A})$ are all linear subspaces. They satisfy several very important relationships:

THEOREM A.20. *For $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the following relationships hold:*

- $\operatorname{null}(\boldsymbol{A})^\perp = \operatorname{range}(\boldsymbol{A}^*)$.
- $\operatorname{range}(\boldsymbol{A})^\perp = \operatorname{null}(\boldsymbol{A}^*)$.
- $\operatorname{null}(\boldsymbol{A}^*) = \operatorname{null}(\boldsymbol{A}\boldsymbol{A}^*)$.
- $\operatorname{range}(\boldsymbol{A}) = \operatorname{range}(\boldsymbol{A}\boldsymbol{A}^*)$.

From this, we obtain that $\dim(\operatorname{row}(\boldsymbol{A})) + \dim(\operatorname{null}(\boldsymbol{A})) = n$.

THEOREM A.21 (Matrix Rank). *For any $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\dim(\operatorname{row}(\boldsymbol{A})) = \dim(\operatorname{range}(\boldsymbol{A}))$. We call the common value the* rank *of $\boldsymbol{A}$. It is equal to the maximum size of a set of linearly independent rows, which is in turn equal to the maximum size of a set of linearly independent columns.*

The rank satisfies many useful properties:

THEOREM A.22 (Facts about Rank). *The rank satisfies:*

- $\operatorname{rank}(\boldsymbol{A}\boldsymbol{B}) \leq \min\{\operatorname{rank}(\boldsymbol{A}), \operatorname{rank}(\boldsymbol{B})\}$.
- **Sylvester's inequalty** *For $\boldsymbol{A} \in \mathbb{R}^{m\times p}$, $\boldsymbol{B} \in \mathbb{R}^{p\times n}$,*

$$\operatorname{rank}(\boldsymbol{A}\boldsymbol{B}) \geq \operatorname{rank}(\boldsymbol{A}) + \operatorname{rank}(\boldsymbol{B}) - p.$$

- **Subadditivity** $\forall\ \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m\times n}$, $\operatorname{rank}(\boldsymbol{A} + \boldsymbol{B}) \leq \operatorname{rank}(\boldsymbol{A}) + \operatorname{rank}(\boldsymbol{B})$.
- $\operatorname{rank}(\boldsymbol{A}) = \operatorname{rank}(\boldsymbol{A}\boldsymbol{A}^*) = \operatorname{rank}(\boldsymbol{A}^*\boldsymbol{A})$.

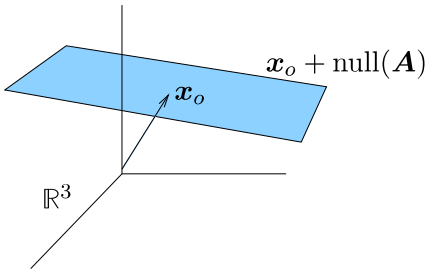


Figure A.2 Geometry of solution sets for linear equations.

A.6 Linear Systems of Equations

Using the range and null space, we can decide if the system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ has a solution, and how many solutions it has:

THEOREM A.23. *Consider a linear system of equations* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$.

- **Existence**: *The system* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ *has a solution* $\boldsymbol{x}$ *if and only if* $\boldsymbol{y} \in \mathrm{range}(\boldsymbol{A})$.
- **Uniqueness**: *Suppose that* $\boldsymbol{x}_o$ *satisfies* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$. *Every solution to the equation* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ *can be generated as* $\boldsymbol{x}_o + \boldsymbol{v}$, *where* $\boldsymbol{v} \in \mathrm{null}(\boldsymbol{A})$. *The solution* $\boldsymbol{x}_o$ *is* unique *if and only if the null space is trivial (*$\mathrm{null}(\boldsymbol{A}) = \{\boldsymbol{0}\}$*).*

The last point means that whenever $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ has a solution $\boldsymbol{x}_o$, the solution set has the form

$$\boldsymbol{x}_o + \mathrm{null}(\boldsymbol{A}). \tag{A.6.1}$$

The "+" here is "in the sense of Minkowski", which just means that $\boldsymbol{x} + \mathsf{S} = \{\boldsymbol{x} + \boldsymbol{s} \mid \boldsymbol{s} \in \mathsf{S}\}$. Since $\mathrm{null}(\boldsymbol{A})$ is a linear subspace, the resulting set is a translate of a linear subspace. We call such a set an *affine subspace*. Unlike a linear subspace, an affine subspace need not contain $\boldsymbol{0}$.

DEFINITION A.24 (Affine Combination and Affine Subspace). *Let* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{V}$. *An* affine combination *is an expression of the form* $\sum_i \alpha_i \boldsymbol{v}_i$, *with* $\sum_i \alpha_i = 1$. *An* affine subspace *is a set* $\mathsf{A} \subset \mathbb{V}$ *which is stable under affine combinations.*

It is easy to check that A is an affine subspace if and only if $\mathsf{A} = \boldsymbol{x} + \mathsf{S}$ for some linear subspace S. So, geometrically, we can visualize the solution set of $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ as living on a plane which does not contain $\boldsymbol{0}$ – see Figure A.2.

Invertible Systems and Conjugate Gradient

If $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ is square, and has full rank m, then for every $\boldsymbol{y} \in \mathbb{R}^m$, the system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ has exactly one solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^{-1}\boldsymbol{y}$, where $\boldsymbol{A}^{-1}$ can be computed according to Theorem A.18. However, when the matrix $\boldsymbol{A}$ is large, computing the inverse is very expensive. A much more efficient numerical method to compute the solution to above linear system is the so-called *conjugate gradient method.*

For completeness, we here only describe the method but refer the readers to detailed derivation and analysis to [She94, NW06].

The conjugate gradient is essentially an accelerated optimization method that solves the quadratic minimization problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \tag{A.6.2}$$

with an iterative procedure. Let $\boldsymbol{r}_i \in \mathbb{R}^m$ to denote the residual and $\boldsymbol{d}_i \in \mathbb{R}^m$ be the direction of incremental descent. The iterative process starts from any given initial state $\boldsymbol{x}_0 \in \mathbb{R}^m$, and the residual and descent direction are initialized as

$$\boldsymbol{d}_0 = \boldsymbol{r}_0 = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_0.$$

The conjugate gradient descent process is given by the following iteration: For $i = 0, 1, 2, \ldots,$

$$\text{Conjugate Gradient:} \begin{cases} \alpha_i &= \frac{\boldsymbol{r}_i^* \boldsymbol{r}_i}{\boldsymbol{d}_i^* \boldsymbol{A} \boldsymbol{d}_i}, \\ \boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + \alpha_i \boldsymbol{d}_i, \\ \boldsymbol{r}_{i+1} &= \boldsymbol{r}_i - \alpha_i \boldsymbol{A}\boldsymbol{d}_i, \\ \beta_{i+1} &= \frac{\boldsymbol{r}_{i+1}^* \boldsymbol{r}_{i+1}}{\boldsymbol{r}_i^* \boldsymbol{r}_i}, \\ \boldsymbol{d}_{i+1} &= \boldsymbol{r}_{i+1} + \beta_{i+1} \boldsymbol{d}_i. \end{cases} \tag{A.6.3}$$

The process terminates when the error reach a prescribed accuracy: $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_i\|_2 \leq \varepsilon$. The precise complexity of the conjugate gradient descent is characterized in Theorem 9.11, which plays a crucial role in achieving the optimal rate for the Newton descent scheme studied there.

In practice, we very frequently encounter linear systems of equations $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ for which $\boldsymbol{A}$ is not invertible. We describe two important cases below.

Overdetermined Systems.
Suppose that $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $m > n$. Since $\operatorname{rank}(\boldsymbol{A}) \leq \min\{m, n\} < m$, the range of $\boldsymbol{A}$ is a lower-dimensional subspace of $\mathbb{R}^m$. Hence, in general, the system of equations $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ will not have a solution. Hence, we resort to seeking an approximate solution. Classically, this was often done via the method of *least squares*. Define the Euclidean length $\|\boldsymbol{z}\|_2 = \sqrt{\sum_i z_i^2}$ of a vector $\boldsymbol{z} \in \mathbb{R}^n$. A least-squares solution solves

$$\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 . \tag{A.6.4}$$

If $\boldsymbol{A}$ has full column rank n, the solution $\hat{\boldsymbol{x}}_{LS}$ to this problem is unique, and is given by

$$\hat{\boldsymbol{x}}_{LS} = (\boldsymbol{A}^*\boldsymbol{A})^{-1}\boldsymbol{A}^*\boldsymbol{y}. \tag{A.6.5}$$

We sometimes write $\boldsymbol{A}^\dagger = (\boldsymbol{A}^*\boldsymbol{A})^{-1}\boldsymbol{A}^*$, and call this matrix the *pseudo-inverse* of $\boldsymbol{A}$. Notice that

$$\boldsymbol{A}\hat{\boldsymbol{x}}_{LS} = \boldsymbol{A}(\boldsymbol{A}^*\boldsymbol{A})^{-1}\boldsymbol{A}^*\boldsymbol{y} \tag{A.6.6}$$

$$= \boldsymbol{P}_{\text{range}(\boldsymbol{A})}\boldsymbol{y} \tag{A.6.7}$$

is the orthogonal projection of $\boldsymbol{y}$ onto range($\boldsymbol{A}$); the matrix

$$\boldsymbol{P}_{\text{range}(\boldsymbol{A})} = \boldsymbol{A}(\boldsymbol{A}^*\boldsymbol{A})^{-1}\boldsymbol{A}^*$$

is the projection matrix onto this space. The optimal value of the least squares problem is

$$\|\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}_{LS}\|_2^2 = \left\|(\boldsymbol{I} - \boldsymbol{P}_{\text{range}(\boldsymbol{A})})\boldsymbol{y}\right\|_2^2 \tag{A.6.8}$$

$$= \left\|\boldsymbol{P}_{\text{range}(\boldsymbol{A})^\perp}\boldsymbol{y}\right\|_2^2. \tag{A.6.9}$$

This is just the squared (Euclidean) distance from the observation $\boldsymbol{y}$ to range($\boldsymbol{A}$).

Underdetermined Systems.
If on the other hand $m < n$, as discussed above, the solution is not unique – if any solution $\boldsymbol{x}_o$ exists, then there is an entire affine space $\boldsymbol{x}_o + \text{null}(\boldsymbol{A})$ of solutions. A classical approach to handling such underdetermined systems is to look for the $\boldsymbol{x}$ of smallest length that is consistent with the system. Formally,

$$\min \quad \|\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}. \tag{A.6.10}$$

If $\boldsymbol{A}$ has full row rank (i.e., $\text{rank}(\boldsymbol{A}) = m$), this problem has a unique solution:

THEOREM A.25. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ have full row rank (i.e., $\text{rank}\,(\boldsymbol{A}) = m$). Then for any $\boldsymbol{y} \in \mathbb{R}^m$, the optimization problem*

$$\min \quad \|\boldsymbol{x}\|_2^2 \quad \textit{subject to} \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{A.6.11}$$

has a unique optimal solution,

$$\hat{\boldsymbol{x}}_{\ell^2} = \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1}\boldsymbol{y}. \tag{A.6.12}$$

Proof The following inequality can be checked by directly expanding the right hand side:

$$\begin{aligned}\forall\, \boldsymbol{x},\, \boldsymbol{x}' \in \mathbb{R}^n,\ \|\boldsymbol{x}'\|_2^2 &= \|\boldsymbol{x} + (\boldsymbol{x}' - \boldsymbol{x})\|_2^2 \\ &= \|\boldsymbol{x}\|_2^2 + \langle 2\boldsymbol{x}, \boldsymbol{x}' - \boldsymbol{x}\rangle + \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2. \end{aligned}\tag{A.6.13}$$

If $\boldsymbol{x}$ and $\boldsymbol{x}'$ are feasible for our problem, then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}' = \boldsymbol{y}$, and so $\boldsymbol{x}' - \boldsymbol{x} \in \text{null}(\boldsymbol{A})$. For any feasible $\boldsymbol{x}' \neq \hat{\boldsymbol{x}}_{\ell^2}$, we have

$$\begin{aligned}\|\boldsymbol{x}'\|_2^2 &\geq \|\hat{\boldsymbol{x}}_{\ell^2}\|_2^2 + 2\,\langle \hat{\boldsymbol{x}}_{\ell^2}, \boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2}\rangle + \|\boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2}\|_2^2 \\ &= \|\hat{\boldsymbol{x}}_{\ell^2}\|_2^2 + 2\left\langle \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1}\boldsymbol{y}, \boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2}\right\rangle + \|\boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2}\|_2^2 \\ &= \|\hat{\boldsymbol{x}}_{\ell^2}\|_2^2 + \underbrace{2\left\langle (\boldsymbol{A}\boldsymbol{A}^*)^{-1}\boldsymbol{y}, \boldsymbol{A}(\boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2})\right\rangle}_{=0} + \|\boldsymbol{x}' - \hat{\boldsymbol{x}}_{\ell^2}\|_2^2 \\ &> \|\hat{\boldsymbol{x}}_{\ell^2}\|_2^2. \end{aligned}\tag{A.6.14}$$

□

The matrix $\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1}$ is also called a pseudo-inverse of $\boldsymbol{A}$, and also denoted by $\boldsymbol{A}^\dagger$.[3]

In the above, we have assumed that the matrix $\boldsymbol{A}$ is of full column or row rank. In practice, the system of equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ can be ill-posed or the equations are corrupted by some random (Gaussian) noise $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$. In this case, we may consider solving a regularized version, say the popular *ridge regression*:

$$\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2, \tag{A.6.15}$$

We leave as an exercise for the reader to find the optimal solution to the above problem (see Exercise 1.8).

A.7 Eigenvectors and Eigenvalues

Definition A.26 (Eigenvalue and Eigenvector). *Let $\boldsymbol{A} \in \mathbb{C}^{n\times n}$. We say that $\lambda \in \mathbb{C}$ is an eigenvalue of $\boldsymbol{A}$ if there exists some nonzero vector $\boldsymbol{v} \in \mathbb{C}^n \setminus \{\boldsymbol{0}\}$ such that*

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}. \tag{A.7.1}$$

If we view $\boldsymbol{A}$ as corresponding to a linear map $\mathcal{L} : \mathbb{C}^n \to \mathbb{C}^n$, the definition says that $\mathcal{L}$ preserves the direction of the vector $\boldsymbol{v}$. If λ is an eigenvalue of $\boldsymbol{A}$, with corresponding eigenvector $\boldsymbol{v}$, then $\boldsymbol{v} \in \text{null}(\boldsymbol{A} - \lambda \boldsymbol{I})$, and hence rank $(\boldsymbol{A} - \lambda \boldsymbol{I}) < n$. Using the determinant criterion for singularity, we obtain

Theorem A.27. *$\lambda \in \mathbb{C}$ is an eigenvalue of $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ if and only if it is a root of the characteristic polynomial*

$$\chi(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I}), \tag{A.7.2}$$

i.e., $\chi(\lambda) = 0$.

This implies that every matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ has n complex eigenvalues, counted with multiplicity. Often we are interested in real matrices $\boldsymbol{A} \in \mathbb{R}^{n\times n}$.

Real Symmetric Matrices

It is important to note that *the eigenvalues of a real matrix are not necessarily real.* There is one important special case in which the eigenvalues are guaranteed to be real: symmetric matrices. A matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is *symmetric* if

$$\boldsymbol{A} = \boldsymbol{A}^*. \tag{A.7.3}$$

The eigenvalues of a symmetric matrix are necessarily real, with corresponding real eigenvectors. Moreover, it is not difficult to prove that if $\boldsymbol{v}$ and $\boldsymbol{v}'$ are eigenvectors of a symmetric matrix corresponding to *distinct* eigenvalues $\lambda \neq \lambda'$, then

[3] The fact that we have apparently used the notation $\boldsymbol{A}^\dagger$ for two different things is resolved if we consider the general form of the pseudo-inverse, which is written in terms of the singular value decomposition (SVD). We will do this after reviewing the SVD in Section A.8.

they are orthogonal: $\boldsymbol{v} \perp \boldsymbol{v}'$. From this, we obtain the eigenvector decomposition of a symmetric matrix:

THEOREM A.28 (Eigenvector Decomposition). *Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be symmetric. Then there exist orthonormal vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$ and real scalars $\lambda_1 \geq \cdots \geq \lambda_n$, such that if we write*

$$\boldsymbol{V} = [\boldsymbol{v}_1 \mid \cdots \mid \boldsymbol{v}_n] \in \mathsf{O}(n), \qquad \boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{A.7.4}$$

we have

$$\boldsymbol{A} = \boldsymbol{V} \boldsymbol{\Lambda} \boldsymbol{V}^*, \tag{A.7.5}$$

The expression $\boldsymbol{A} = \boldsymbol{V} \boldsymbol{\Lambda} \boldsymbol{V}^*$ is sometimes written as $\boldsymbol{A} = \sum_{i=1}^n \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^*$. Theorem A.28 leads to the following variational characterization of the eigenvalues, which is useful both for analytical purposes and for identifying optimization problems that can be solved directly via eigenvector decomposition:

THEOREM A.29 (Variational Characterization of Eigenvalues). *The first eigenvalue λ_1 of a symmetric matrix $\boldsymbol{A}$ is the optimal value of the problem*

$$\begin{array}{ll} \max & \boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} \\ \text{subject to} & \|\boldsymbol{x}\|_2^2 = 1. \end{array} \tag{A.7.6}$$

Moreover, every optimizer $\boldsymbol{v}_1$ is an eigenvector corresponding to λ_1. Similarly, the optimal value of

$$\begin{array}{ll} \min & \boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} \\ \text{subject to} & \|\boldsymbol{x}\|_2^2 = 1 \end{array} \tag{A.7.7}$$

is λ_n. For the intermediate eigenvalues, if $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}$ are any mutually orthogonal eigenvectors corresponding to $\lambda_1, \ldots, \lambda_{k-1}$, we have that λ_k is the optimal value for

$$\begin{array}{ll} \max & \boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} \\ \text{subject to} & \|\boldsymbol{x}\|_2^2 = 1,\ \boldsymbol{x} \perp \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}. \end{array} \tag{A.7.8}$$

From the previous result, it seems the eigenvector decomposition is a very useful tool for studying quadratic forms $q(\boldsymbol{x}) = \boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x}$. Matrices $\boldsymbol{A}$ for which $q(\boldsymbol{x})$ is always positive are especially important:

DEFINITION A.30 (Positive Definiteness). *A symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is* positive definite *if for all nonzero $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} > 0$. It is* positive semidefinite *if for all $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} \geq 0$.*

If $\boldsymbol{A}$ is positive definite, we write

$$\boldsymbol{A} \succ \boldsymbol{0}. \tag{A.7.9}$$

If $\boldsymbol{A}$ is positive semidefiniite, we write

$$\boldsymbol{A} \succeq \boldsymbol{0}. \tag{A.7.10}$$

More generally, for symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we write $\boldsymbol{A} \succeq \boldsymbol{B}$ if $\boldsymbol{A} - \boldsymbol{B}$ is semidefinite, i.e., $\boldsymbol{A} - \boldsymbol{B} \succeq \boldsymbol{0}$. This defines a *partial order* on the symmetric matrices, which we call the *semidefinite order*.

THEOREM A.31. *A symmetric matrix $\boldsymbol{A}$ is positive definite (resp. semidefinite) if and only if all of its eigenvalues are positive (resp. nonnegative).*

Circulant Matrix and Convolution.
Given a vector $\boldsymbol{a} = [a_0, a_1, \ldots, a_{n-1}]^* \in \mathbb{R}^n$, we may arrange all its circularly shifted versions in a circulant matrix form as

$$\boldsymbol{A} \doteq \mathsf{circ}(\boldsymbol{a}) = \begin{bmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & a_{n-1} & \cdots & a_2 \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ a_{n-2} & \vdots & \ddots & \ddots & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{A.7.11}$$

It is easy to see that the multiplication of such a circulant matrix $\boldsymbol{A}$ with a vector $\boldsymbol{x}$ gives a (circular) convolution $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{a} \circledast \boldsymbol{x}$ with:

$$(\boldsymbol{a} \circledast \boldsymbol{x})_i = \sum_{j=0}^{n-1} x_j a_{i+n-j \bmod n}. \tag{A.7.12}$$

One remarkable property of circulant matrices is that *they all share the same set of eigenvectors that form a unitary matrix.* Let $\mathfrak{i} = \sqrt{-1}$ and $\omega_n := \exp(-\frac{2\pi\mathfrak{i}}{n})$ be the roots of unit (as $\omega^n = 1$) and we define the matrix:

$$\boldsymbol{F}_n \doteq \frac{1}{\sqrt{n}} \begin{bmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-2} & \omega_n^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \omega_n^0 & \omega_n^{n-2} & \cdots & \omega_n^{(n-2)^2} & \omega_n^{(n-2)(n-1)} \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-2)(n-1)} & \omega_n^{(n-1)^2} \end{bmatrix} \in \mathbb{C}^{n\times n}. \tag{A.7.13}$$

The matrix $\boldsymbol{F}_n$ is a unitary matrix: $\boldsymbol{F}_n\boldsymbol{F}_n^* = \boldsymbol{I}$ and is the well known *Vandermonde matrix*. Multiplying a vector with $\boldsymbol{F}_n$ is known as the *discrete Fourier transform* (DFT). More precisely, we have the following well-known fact [KS12]:

THEOREM A.32 (Eigenvectors of Circulant Matrix). *An $n \times n$ matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ is a circulant matrix if and only if it is diagonalizable by the unitary matrix:*

$$\boldsymbol{F}_n^*\boldsymbol{A}\boldsymbol{F}_n = \boldsymbol{D}_{\boldsymbol{a}} \quad \textit{or} \quad \boldsymbol{A} = \boldsymbol{F}_n\boldsymbol{D}_{\boldsymbol{a}}\boldsymbol{F}_n^*, \tag{A.7.14}$$

where $\boldsymbol{D}_{\boldsymbol{a}}$ is a diagonal matrix of eigenvalues.[4]

[4] The eigenvalues can be complex even for real circulant matrices.

From the above fact, we can easily derive the following properties of circulant matrices:

- Transpose of a circulant matrix is circulant;
- Multiplication of two circulant matrices is circulant;
- For a non-singular circulant matrix, its inverse is also circulant (hence representing a circular convolution).
- Since all circulant matrices can be simultaneously diagonalized by the same unitary matrix $\boldsymbol{F}_n$, their summation and inversion can be reduced to the same operations on their diagonal forms, hence much faster and scalable.

Location of Eigenvalues.
It is often useful to be able to characterize, in terms of the properties of $\boldsymbol{A}$, where the eigenvalues $\lambda \in \mathbb{C}$ are located. For example, we saw that if $\boldsymbol{A}$ is a symmetric matrix, the eigenvalues lie on the real axis. For general $\boldsymbol{A}$, the situation is more complicated. However, we do have the following result of Gershgorin, which states that the eigenvalues must live in a union of discs, centered about the diagonal elements A_{ii} of $\boldsymbol{A}$:

THEOREM A.33 (Gershgorin Disc Theorem). *Let $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, and let $\lambda \in \mathbb{C}$ and $\boldsymbol{v} \in \mathbb{C}^n$ be an eigenvalue-eigenvector pair. Then there exists some $i \in \{1, \dots, n\}$ such that*

$$|\lambda - A_{ii}| \leq \sum_{j\neq i} |A_{ij}|. \tag{A.7.15}$$

This result is called the Gershgorin disc theorem, because it implies that in the complex plane $\mathbb{C}$, each eigenvalue λ lies in a union of discs D_i with centers A_{ii} and radii $r_i = \sum_{j\neq i} |A_{ij}|$. It is most powerful when the off-diagonal elements of $\boldsymbol{A}$ are small. Numerous variants and refinements are known.

A.8 The Singular Value Decomposition (SVD)

Definitions.
The eigenvector decomposition $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^*$ defined in Theorem A.28 provides an essential tool for studying symmetric matrices $\boldsymbol{S}$. In particular, it shows that with an appropriate rotation of the space, a symmetric matrix acts like a diagonal matrix. It would be very useful to have a similar representation for general matrices, including non-symmetric square matrices, and rectangular matrices. The *singular value decomposition* goes much of the way, allowing us to find bases for the domain and range of a linear map with respect to which it becomes quite simple:

THEOREM A.34 (Compact SVD, Existence). *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, with* $\operatorname{rank}(\boldsymbol{A}) = r$.

There exist scalars $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ *and matrices* $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ *and* $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ *with orthonormal columns* ($\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{I}$, $\boldsymbol{V}^*\boldsymbol{V} = \boldsymbol{I}$) *such that if we set*

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r}, \tag{A.8.1}$$

we have

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*. \tag{A.8.2}$$

The σ_i *are called* singular values *of* $\boldsymbol{A}$, *while the columns of* $\boldsymbol{U}$ *and* $\boldsymbol{V}$ *are called the (left and right, respectively) singular vectors.*

The expression in Theorem A.34 can be used to express $\boldsymbol{A}$ as a sum of r orthogonal rank-one matrices:

$$\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*. \tag{A.8.3}$$

The compact SVD immediately reveals several important properties of $\boldsymbol{A}$:

THEOREM A.35 (Properties of Compact SVD). *Let* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, *with compact SVD* $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$. *Then*

- $\operatorname{range}(\boldsymbol{A}) = \operatorname{range}(\boldsymbol{U})$. *The columns of* $\boldsymbol{U}$ *are an orthonormal basis for the range of* $\boldsymbol{A}$.
- $\operatorname{range}(\boldsymbol{A}^*) = \operatorname{range}(\boldsymbol{V})$. *The columns of* $\boldsymbol{V}$ *are an orthonormal basis for the row space of* $\boldsymbol{A}$.

Occasionally it is useful to extend $\boldsymbol{U}$ and $\boldsymbol{V}$ to orthogonal matrices, giving the full singular value decomposition:

THEOREM A.36 (Full SVD). *Let* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. *Then there exist* $\boldsymbol{U} \in \mathsf{O}(m)$, $\boldsymbol{V} \in \mathsf{O}(n)$, *and* $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ *such that*

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*, \tag{A.8.4}$$

$\boldsymbol{\Sigma}$ *is diagonal (i.e.,* $\Sigma_{ij} = 0$ *for* $i \neq j$), *and*

$$\Sigma_{11} \geq \Sigma_{22} \geq \cdots \geq \Sigma_{\min\{m,n\},\min\{m,n\}} \geq 0.$$

With a full SVD, we may write the pseudo-inverse of a matrix, introduced in Section A.6, in a unified form:

$$\boldsymbol{A}^\dagger = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^*, \tag{A.8.5}$$

where $\boldsymbol{\Sigma}^\dagger \in \mathbb{R}^{n\times m}$ is the pseudo-inverse of the diagonal matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$.[5]

It is sometimes a point of confusion that the notation for the full SVD and the compact SVD coincide. In this course, we will mostly work with the compact SVD, unless stated otherwise.

[5] That is, $\boldsymbol{\Sigma}^\dagger$ is a diagonal matrix with the diagonal entries Σ_{ii}^{-1} for all $\Sigma_{ii} > 0$.

Approximation Properties.
The SVD provides an immediate solution to several approximation problems. Most fundamentally, it gives a way of forming a best rank-r approximation to $\boldsymbol{A}$:

THEOREM A.37 (Best Rank-r Approximation). *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ have singular value decomposition*

$$\boldsymbol{A} = \sum_{i=1}^{\min\{m,n\}} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*. \tag{A.8.6}$$

Then an *optimal solution to the rank-r approximation problem*

$$\begin{array}{ll} \min & \|\boldsymbol{X} - \boldsymbol{A}\|_F \\ \text{subject to} & \operatorname{rank}(\boldsymbol{X}) \leq r \end{array} \tag{A.8.7}$$

is the truncated SVD

$$\widehat{\boldsymbol{A}}_r = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*. \tag{A.8.8}$$

If $\sigma_r(\boldsymbol{A}) > \sigma_{r+1}(\boldsymbol{A})$, then the solution is unique.

Interestingly, if we change $\|\cdot\|_F$ to other unitary invariant matrix norms (such as the operator norm), the above result remains unchanged. The SVD also gives a way of optimally approximating a given square matrix with an orthogonal matrix:

THEOREM A.38 (Best Orthogonal Approximation). *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, and let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ be any full singular value decomposition of $\boldsymbol{A}$. Then an optimal solution to the problem*

$$\begin{array}{ll} \min & \|\boldsymbol{X} - \boldsymbol{A}\|_F \\ \text{subject to} & \boldsymbol{X} \in \mathsf{O}(n) \end{array} \tag{A.8.9}$$

is given by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}^*$.

A.9 Vector and Matrix Norms

Norms on Vector Spaces.
A *norm* on a vector space $\mathbb{V}$ gives a way of measuring lengths of vectors, that conforms in important ways to our intuition from lengths in $\mathbb{R}^3$. Formally:

DEFINITION A.39 (Norm). *A norm on a real vector space $\mathbb{V}$ is a function $\|\cdot\| : \mathbb{V} \to \mathbb{R}$ that is*

1 **Nonnegatively homogeneous**: *$\|\alpha\boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$ for all vectors $\boldsymbol{x} \in \mathbb{V}$, scalars $\alpha \in \mathbb{R}$,*
2 **Positive definite**: *$\|\boldsymbol{x}\| \geq 0$, and $\|\boldsymbol{x}\| = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$,*