Project - Image Denoising via Sparse Coding

1 Introduction

In the area of signal/image processing, the tasks often related with recovering a target $x^\dagger$ from its **measurements**, which can be modeled as

$$f = Ax^\dagger + \epsilon$$

where A is the observation operator and ϵ stands for noise. One simple example of (1) is **signal sampling and reconstruction**: suppose we have a analog signal from which are sampling at the rate of F samples per second without noise, the classical Nyquist sampling theorem tells us that, in order to obtain a perfect reconstruction of the signal, the sampling rate must be at least *twice* its max frequency. Sampling theorem poses a hard constrain onto the signal we can recover, or challenges to hardware design of the target we wish to recover is not band-limited, or its max frequency is too high.

However, when the signal $x^\dagger$ has certain structure, the Nyquist theorem can by bypassed and the signal can be recovered from a very smaller amount of observations. One example of such "certain structure" is *sparsity* or *sparse representation*, which in general means that though the dimension of the signal can be very large (for example high sampling frequency means huge dimension), its *representation* requires much less number of points.

Example 1.1 (Staircase function). For the staircase function if Figure 1, it is clear that representing it using **Fourier Transform** requires the whole spectrum due to the *singularity*. However, representing it using only *position + jump* requires only two numbers.

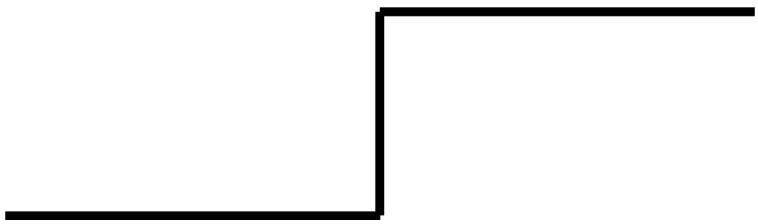

Figure 1: Staircase function.

Hence in sparse representation, the term "sparse" implies that most of the coefficients in the linear combination are zero, or in other words, each signal or image can be represented as a combination of a small number of basic elements. Sparse coding was initially developed in the context of neural coding, where it was proposed as a model for how the mammalian visual cortex represents visual information efficiently. Theoretical breakthrough of sparse coding was around 2005 due to E. Candès, T. Tao, J. Romberg and D. Donoho. In the work *Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information* by Candès, Romberg and Tao, they show that for the problem (1), if the number of non-zero elements, say κ, in $x^\dagger$ is small enough, and A satisfies certain properties, then $x^\dagger$ can be recovered from $O(\kappa)$ number of measurement. Later on, Donoho proposed the term **compressed sensing** in a paper with the same name.

2 Sparse Representation and Dictionary Learning

The basic elements used in sparse coding come from a so-called "dictionary" of atoms. Each atom is a small signal or image patch, and a complete signal or image can be approximated by adding up the atoms, each multiplied by a corresponding coefficient. The goal of sparse representation is to find the coefficients that give the best approximation, under the constraint that the coefficients are sparse.

Dictionary Learning, on the other hand, is the process of learning the dictionary of atoms used in sparse coding. Instead of using a predefined dictionary, such as a Fourier basis or a wavelet basis, the idea in dictionary learning is to learn the dictionary from the data itself. This has the advantage that the dictionary can adapt to the specific structures in the data, leading to more efficient representations.

In the process of dictionary learning, an initial dictionary is typically chosen at random, and then iteratively updated to better fit the data. This process involves two alternating steps: a sparse coding step, where the coefficients for the current dictionary are found, and a dictionary update step, where the dictionary atoms are updated given the current coefficients. This is a non-convex optimization problem, and various algorithms have been proposed to solve it, such as the K-SVD algorithm, the method of optimal directions (MOD), and online dictionary learning.

Sparse coding and dictionary learning together form the basis for a class of models known as sparse coding models. These models have been successful in a variety of tasks, such as image denoising, inpainting, and compression, audio source separation, and more. They have also been used as a tool for feature extraction in machine learning, where the sparse coefficients are used as features for classification or regression tasks.

2.1 Sparse presentation

Before the era of deep learning, sparse representation was of the most successful technique in signal/image processing and computer vision. As a matter of fact, sparse coding and dictionary learning provide a framework for understanding deep learning models. In particular, convolutional neural networks can be seen as a generalization of sparse coding models, where the dictionary is learned in a hierarchical fashion; see for instance *Visualizing and understanding convolutional networks* by Zeiler and Fergus.

Given an image x of size $m \times n$. We first decompose it into non-overlapping patches, then for each patch, we concatenating the columns of the patch into a single column vector, then lastly putting the columns together we get an matrix representation of the image. More precisely, let p_i be a patch of the image, and $\vec{p}_i$ be the column of concatenating the columns of p_i, then

$$X = \left[\vec{p}_1, \vec{p}_2, \cdots, \vec{p}_N\right] \in \mathbb{R}^{P \times N}, \tag{2.1}$$

where P is the size of each patch, and N is the total number of patches. Now let the set $\{d_1, d_2, ..., d_K\}$ be a bank of **atoms**, i.e. our dictionary, each atom has the same size at the image patches above. Vectorizing each atom (noted as $\vec{d}_i$) and putting them together, we obtain the matrix representation of the

$$D = \left[\vec{d}_1, \vec{d}_2, \cdots, \vec{d}_K\right] \in \mathbb{R}^{P \times K}. \tag{2.2}$$

Given a patch p, it can be written as the linear combination of the atoms, that is

$$p = \sum_{i=1}^{K} a_i d_i + e \tag{2.3}$$

where a_i is the cofficient of each atom, and e is the error (meaning that p cannot be perfecty represented by the atoms). In the vectorized form, we have

$$\vec{p} = D\mathbf{a} + \mathbf{e} \tag{2.4}$$

where $\mathbf{a} = (a_1, a_2, ..., a_K)^T \in \mathbb{R}^K$ is the coefficient vector. With the above setup, now we can represent X with D, which reads

$$X = DA + E \tag{2.5}$$

where $A = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_N]$ is the coefficient of each patch and $E = [\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_N]$ is the error in representing each patch.

When we are given a perfect dictionary, the representation of X under D is sparse, that is the

number of non-zeros in A is the smallest. However, in practice it is impossible to have the perfect dictionary, hence we need to learning it, from given image(s).

2.2 Dictionary learning

Given an image in patched form X, to learn dictionary from it, we need to solve the following optimization problem

$$\min_{D,A} \frac{1}{2}\|X - DA\|_F^2 + \lambda\|A\|_0, \tag{2.6}$$

where

- $\|\cdot\|_F^2$ is the sum of the squares of all the elements;
- $\|\cdot\|_0$ returns the number of non-zeros elements;
- $\lambda > 0$ is a balancing weight.

Apparently, the above problem is non-convex due to $\|\cdot\|_0$. To circumvent this difficulty, we can replace it with the ℓ_1-norm, leading to

$$\min_{D,A} \frac{1}{2}\|X - DA\|_F^2 + \lambda\|A\|_1. \tag{2.7}$$

Note that in both models, their best λ's have different values. By "best", means that the resulting A is sparest.

Though we have made the problem convex, there remains two problems

- The first problem is that (2.7) is non-smooth, making it impossible to use gradient descent schemes. As a result, we need to further smooth the ℓ_1-norm, for example the **Huber function** which is denoted as $h(A)$, which results in

$$\min_{D,A} \Phi(A, D) = \frac{1}{2}\|X - DA\|_F^2 + \lambda h(A). \tag{2.8}$$

- The second problem is that (2.8) is non-convex, as we have the bilinear term DA. However, when either term is fixed, the problem is convex. More precisely, when A is fixed, (2.8) is a convex problem of D. As a consequence, we can use an **alternating strategy** to solve (2.8)

$$\begin{cases} A^{(k+1)} = A^{(k)} - \gamma_A \nabla_A \Phi(A, D^{(k)}), \\ D^{(k+1)} = D^{(k)} - \gamma_D \nabla_D \Phi(A^{(k+1)}, D), \end{cases} \tag{2.9}$$

  where γ_A, γ_D are step-sizes.

See Figure 3 below for an image and dictionaries learned from it.

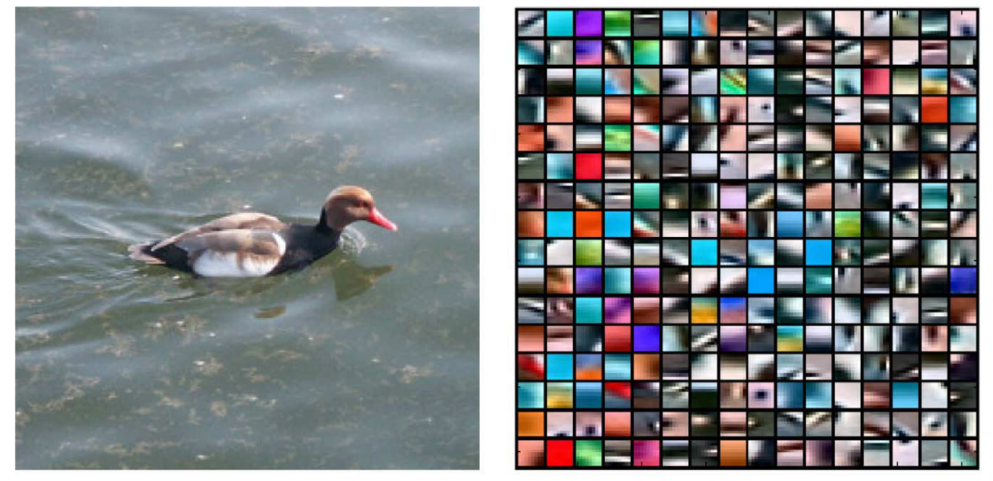

Figure 2: Given image and learned dictionary from the image.

Remark 2.1. Take fixed A for example, note that the gradient of Φ respect to D is Lipschitz, however

the Lipschitz constant depends on A. Same for A when fixing D. So a proper strategy of choosing step-size should be devised.

Remark 2.2. In the above discussion, we use Huber function to smooth, you can also consider other strategies for smoothing. For the algorithm, you can also consider others instead of (2.9).

Remark 2.3. The weight parameter λ here is **scalar**, you can also consider other types of λ, such as a vector where each λ_i is for each row of A. Another option of λ is **reweighted** λ, and please search for references for this.

3 Numerical Experiments

In this project, we provide 18 images for numerical experiments, see below.

Figure 3: McM dataset.

These images can be found in the `data.zip` file, including clean images and noised images.

3.1 Settings of the project

We fix the following settings for the dictionary learning

- Patch size is $8 \times 8 \times 3$, so is the dictionary atom size.
- The number of atoms is 128.

Note that in patch size, $\times 3$ means three color channels.

3.2 Tasks of the project

Remark 3.1. It should be noted that the model introduced above, e.g. (2.8), is to learn a dictionary from a **single** image. We can also put the patches of several images together to learn a dictionary from them.

3.2.1 Sparse representation and denoising

Task 1 For the *top left* image, find the dictionary D such that the A is the **sparest**. By **sparest**, means the *least number* of non-zero elements, not the *smallest* ℓ_1-norm. Parameters choices should be included in the report.

Task 2 For the *top left* image, with the dictionary learned from **Task 1**, design an approach to denoise, i.e. removing the noise of its noised image (also given in the `data.zip`).

3.2.2 Image denoising for all images

Task 3 For each image, learn a dictionary from it and apply the dictionary to remove noise from the corresponding noisy image. Output the PSNR of the denoised image (see blow), which is the higher the better

$$\texttt{MSE} = \frac{\sum_{i,j}(x^{\dagger}_{i,j} - \bar{x}_{i,j})^2}{N} \quad \text{and} \quad \texttt{PSNR} = 10 \times \log_{10} \frac{255^2}{\texttt{MSE}}$$

where $x^{\dagger}$ is the ground truth image, $\bar{x}$ is the denoised image, and N is the total number of pixels. For color image, the PSNR is computed for each channel.

Task 4 Task 3 apparently will take quite long time as we need to solve (2.8) for every image. If we put all image together, and solve (2.8) only once, the scale of this configuration will also be very large. Moreover, 64 atoms probably will be not enough to capture the **features** of the combined images. There are two possible tricks to make the overcome these two difficulties - To reduce the scale of problem, from each image we can choose a subset of patches and combine them together. The question then becomes how to choose patches from each image. - To capture features from the images, we can increase the number of atoms, for example 256 as shown in Figure 2.

3.3 Unknown ground truth

In the above tasks, the dictionary is learned from ground truth images, which in most practical scenarios is impossible. Therefore, we have the following "Task 5".

Task 5 With a given noisy image, design a model which can perform dictionary learning and imaging denoising in the same time. Then apply this model to the given noisy images, and output the PSNR values.

4 Assessments

Below we describe the format and assessment criteria of the project.

- **Project format** It is a group project, maximum number of team members is **5**.
- **Grades** The grade for the project include three parts: *report, result ranking* (in terms of PSNR values), with percentage 50%, 50%, respectively.
  For the result ranking, the scores are based on **percentile**, which is

Percentile	Scores
0 - 10%	100
11 - 30%	90
31 - 60%	80
61 - 90%	70
91 - 100%	60

- **Submission** Project report, source code. The source code should be **executable**, and all the results should be **reproducible**. As a result, in the submitted codes, you should include a **readme** file explaining the parameters choices of every result, so that our TA can check them.
- **Deadline** 23:59, 14 January, 2024.
- **Warnings**
  - **Cheating** is not allowed, if found, all involved will receive a penalty.