Machine Learning for Signal Processing
Homework 3 - Part A

Bhiksha Raj - Abelino Jimenez, Anurag Kumar, Yixuan Zhang, Davy Uwizera

This is Part A of homework 3. This part contains 2 questions.

1 Sparse recovery

We have previously noted in class that given a dictionary D and a data X that can be composed as a *sparse* combination of dictionary entries, i.e. we can write $X = DY$, where $|Y|_0 \leq k$ for some small value of k, then Y can be recovered from X and D using sparse recovery techniques. Specifically, we noted that this can be estimtated through the following minimization:

$$\hat{Y} = \arg\min_{Y} \|X - DY\|_2^2 + \lambda |Y|_1$$

We will now see how this simple principle can be used to *subsample* data and still recover it. This principle is known as "compressive sensing".

We will use an example to explain the concept. The example will also be your homework problem.

A little story: Cassini-Huygens

The Cassini spacecraft was launched on 15 Oct 1997 and entered into orbit around Saturn on 1 July 2004. Ever since it has been orbiting the ringed planet, taking thousands of pictures of Saturn, its rings, and its moons, and beaming them back to Earth. Here are some of the gorgeous pictures sent by Cassini to earth. Cassini is powered by about 33 kg of Plutonium-238, which will continue to provide power to the spacecraft until the end of its mission.

On Christmas day 2004 Cassini launched the Huygens probe towards Saturn's moon, Titan. Huygens landed on Titan on 14 Jan 2005. Unlike Cassini, Huygens was powered by a battery. By design, the module had no more than three hours of battery life, most of which was planned to be used during the descent onto Titan. Engineers expected to get at most only 30 minutes of data from the surface, much of which was in the form of pictures of Titan's surface and topography. Here are the incredible pictures sent by Huygens.

Our "story" ends with this note: Huygens could only send 350 pictures before its batteries ran out of power. (An additional 350 pictures were lost because of a software bug due to which the channel they were sent on was ignored).

Compressive sensing to the rescue

One of the main consumptions of on-board battery is for the *transmission* of the pictures. Each image must be adequately "wrapped" with error-correcting code and transmitted to the recepient (Huygens transmitted to both Cassini and the Earth). The amount of energy consumed to transmit a picture directly relates to the number of pixels captured – more pixels require more energy both to capture and to transmit.

Let us now consider an alternate scheme.

1. A note on image sparsity in the wavelet domain

   It is well known that when considered in the wavelet transform domain, images are sparse. A wavelet transform can be viewed as a matrix operation (just as a discrete Fourier transform can). Let I be any image (expressed as a vector). The wavelet transform $\mathcal{F}$ of the image can be computed as

   $$\mathcal{F} = WI$$

   where W is the transform matrix. (In reality, the transform is viewed as a *filterbank*, but we will assume the simpler model above for our discussion).

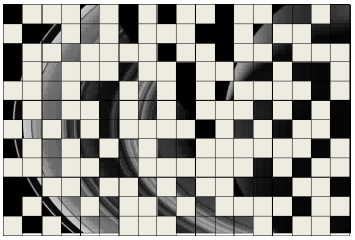
   The image itself can be recovered from the transform as

   $$I = W^{-1}\mathcal{F}$$

   For images, $\mathcal{F}$ is generally sparse, i.e. most of the components of $\mathcal{F}$ are zero or close to zero. **We will use this to our benefit**.

2. Capturing *one-pixel* masked integral images

   Instead of simply snapping a picture directly, consider a scheme where Huygens would instead apply a *random mask* to its camera and computes an *integral* instead. So, for instance, instead of capturing the entire image, Huygens applies a random mask to get the following picture

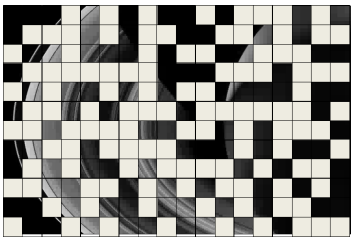
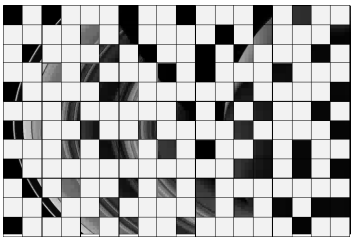
   and computes a **single** integral value

   $$P_1 = \sum_{i,j} m_{i,j} I_{i,j}$$

   where $m_{i,j}$ is the value of the mask at pixel (i, j), and $I_{i,j}$ is the actual image value. Representing the mask as the vector $\mathbf{m}_1$, where the subscript 1 is to indicate that this is the first such mask applied, Huygens' first measurement would be

   $$P_1 = \mathbf{m}_1^\top I$$

   Simiarly, applying a series of other such masks, e.g.

   Huygens can now get a *series of measurements* $P_1,\ P_2,\ P_3, \cdots, P_K$ of (scalar) measurements, where P_i has been obtained using mask $\mathbf{m}_i$, and K is the total number of measurements obtained in this manner. Representing all of the masks as a single matrix $\mathbf{M} = [\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3 \cdots \mathbf{m}_K]^\top$, and the measurements P_i collectively as a vector $\mathbf{P} = [P_1\ P_2\ P_3 \cdots P_K]^\top$, we can write

   $$\mathbf{P} = \mathbf{M}I$$

Note that K may be far fewer than the number of pixels in the image. Huygens can now simply transmit $P_1 \cdots P_K$, and save on power. If $K < N$, where N is the number of pixels in the image, Huygens could use the saved energy to take more pictures and transmit them. Since we are sending far fewer numbers than we would if we were to actually send the entire image, we will call these *compressed* measurements.

3. Recovering the full image from the compressed measurements

   But how then can the N-pixel pictures themselves be recovered from this sequence of K compressed measurements? For this we exploit the sparsity of images in the wavelet domain.

   Recall from our earlier discussion that we can represent $I = W^{-1}\mathcal{F}$, where $\mathcal{F}$ is sparse. Hence we can write

   $$\mathbf{P} = \mathbf{M}W^{-1}\mathcal{F}$$

   Let $\mathbf{R} = \mathbf{M}W^{-1}$. Remember that the random masks applied to the camera are *known*, i.e. $\mathbf{M}$ is known. The inverse wavelet transform matrix W^{-1} of course known. Hence $\mathbf{R}$ is known. We can therefore write

   $$\mathbf{P} = \mathbf{R}\mathcal{F}$$

   In other words, we are expressing the compressed measurements $\mathbf{P}$ as a sparse combination of the columns in a dictionary matrix $\mathbf{R}$.

   To recover the image I from the $K \times 1$ compressed measurement vector $\mathbf{P}$, it is sufficient to recover the $N \times 1$ sparse vector $\mathcal{F}$, since $I = W^{-1}\mathcal{F}$. We can do so using the sparse recovery technique we discussed in the context of dictionary-based representations, which we recalled above:

   $$\hat{\mathcal{F}} = \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 + \lambda|\mathcal{F}|_1$$

   The complete image can now be recovered as $I = W^{-1}\hat{\mathcal{F}}$. This is the basic concept behind compressive sensing.

   We have managed to come up with a solution that enables Huygens to send far fewer numbers (only K) than the size of the image itself (N pixels) and still recover the image. Huygens can now use the conserved battery to take more pictures. We have thus rescued NASA's space exploration program!!

   Having achieved these lofty goals, let us now return to a more mundane issue: homework problems.

Problem 1a

In this problem we give you compressed measurements from an image taken by Cassini (not Huygens). We also give you the "measurement matrix" $\mathbf{R}$. You must recover the full image. (Note: we are referring to $\mathbf{R}$ in the above equations as the measurement matrix; more conventionally $\mathbf{M}$ would be called the measurement matrix).

You have been provided of the original image (`hw3material/problem1`). This is only for reference, to compute error; we will not use it in recovery computations. It is a 128×128 image with a total of 16384 pixels. Note that the sparse wavelet transform $\mathcal{F}$ of the image will also have 16384 components, although most of them will be very small or 0.

You also have been provided of three sets of compressed measurements `P1639.mat`, `P2048.mat` and `P4096.mat`, and their corresponding measurement matrices `R1639.mat`, `R2048.mat` and `R4096.mat` (`hw3material/problem1`). In each case the numbers represent the number of measurements. Thus `P1639.mat` contains a 1639×1 vector, representing 1639 one-pixel measurements, and `R1639.mat` is a 1639×16384 matrix that gives the linear transform from the 16384-component sparse transform $\mathcal{F}$ to the measurement vector in `P1639`. In the directory you will also find files name `SXXXX` (where XXXX is 1639, 2048 or 4096). This file is required by matlab for image reconstruction, and is not part of the actual recovery algorithm.

You are required to use the sparse recovery procedure to recover $\mathcal{F}$ for each of these three measurements, and to reconstruct the image.

i. For each of the three measurements, solve

$$\hat{\mathcal{F}} = \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 + \lambda|\mathcal{F}|_1$$

We recommend setting $\lambda = 100$, although you may also try other values. You may use any solver for the above problem. As a default, we include a matlab implementation of a least-squares ℓ_1 (`l1_ls.m`) solver from Stephen Boyd (`YourAndrewID/problem1`). Note however that this is not the best algorithm; there are other better algorithms for ℓ_1 minimization. The Stanford sparselab page has several nice solvers under the *downloads* link. You may try these; some of them should give you better results than l1_ls. Submissions that report lower error (as explained below) will obtain better scores for this problem, so it is in your benefit to explore. You will be required to submit your code.

ii. For each of the three measurements, reconstruct the original image using the recovered $\mathcal{F}$. You can do so using matlab through the following commands:

```
load SXXXX.mat;
S = SXXXX;
Irec = waverec2(reshape(F,128,128),S,'db1');
```

Compute the error $E = \sum_{i,j}(I(i,j) - I_{rec}(i,j))^2$, where $I(i,j)$ and $I_{rec}(i,j)$ are the (i,j)th pixel value in the original and recovered image respectively. Report this error.

Problem 1b

In part Ia we solved the problem of sparse recovery as one of ℓ_1 minimization. We do so since it is well known that minimizing the ℓ_1 norm also often minimizes the ℓ_0 norm of the solution; however our actual objective in sparse recovery is to find a solution that has minimal ℓ_0 norm, i.e. the smallest number of non-zero elements! Often, the optimal ℓ_1 minimized solution will differ significantly (and can be significantly worse) than the optimal ℓ_0 minimized solution.

In this part of the problem we will solve the alternate optimization problem:

$$\hat{\mathcal{F}} = \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 \quad \text{such that} \quad \|\mathcal{F}\|_0 \leq K$$

where K is the level of sparsity desired in $\mathcal{F}$. This is clearly an ℓ_0 *constrained* optimization problem. $\|\mathcal{F}\|_0 \leq K$ specifies a *feasible set*, which is the set of all vectors that have no more than K non-zero elements (note that this is a non-convex set). Recall from our lecture on optimization that we can use the method of projected gradients for this problem.

The gradient of $\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2$ w.r.t $\mathcal{F}$ is given by

$$\nabla_{\mathcal{F}}\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 = -\mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F})$$

Representing the n^{th} iterate in an iterative estimate of $\mathcal{F}$ by $\mathcal{F}^n$, the projected gradient descent algorithm to estimate $\mathcal{F}$ would be given by

$$\begin{aligned} \mathcal{F}^{n+1}_{unconstrained} &= \mathcal{F}^n + \eta\mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F}^n) \\ \mathcal{F}^{n+1} &= P_K(\mathcal{F}^{n+1}_{unconstrained}) \end{aligned}$$

where η is a step size. The operator P_K in the second step of the algorithm simply sets the $N - K$ smallest elements of $\mathcal{F}^{n+1}_{unconstrained}$ to zero, forcing it to be a $K - sparse$ vector.

The second step in the above iteration is the "projection" step of the projected gradient algorithm. It provably *projects* $\mathcal{F}^{n+1}_{unconstrained}$ onto the feasible set for $\mathcal{F}$, namely the set of all K-sparse vectors.

The above iteration is guaranteed to converge under specific conditions on $\mathbf{R}$, K and η. This is captured nicely in the following algorithm known as "**Iterative Hard Thresholding**" (IHT), proposed by Tom Blumensath.

Iterative Hard Thresholding

As a first step, the IHT algorithm assumes that the individual columns of the measurement matrix $\mathbf{R}$ are vectors of length ≤ 1.0. So, as a first step, confirm that this is true for the provided $\mathbf{R}$.

The IHT algorithm then performs iterations of the following computation:

1. Choose a maximum sparsity level K for your solution.
2. Initialize $\mathcal{F}^0_{unconstrained} = 0$. (Alternately, you could set it to any random value).
3. Iterate:
$$\mathcal{F}^{n+1} = P_K(\mathcal{F}^n + \mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F}^n))$$
until the error $\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2$ converges, or the maximum number of iterations is achieved.
4. The final output is $\hat{\mathcal{F}} = \mathcal{F}_{conv}$, where $\mathcal{F}_{conv}$ is the final estimate derived from the previous step.

The actual homework problem is the following

i. Implement the IHT algorithm.

ii. Find the estimate $\hat{\mathcal{F}}$ for each of the three measurements in problem Ia. Use $K = max(\frac{M}{4}, \frac{N}{10})$, where M is the number of measurements, and N is the size of the image. Note that these are far from optimal settings. Set the maximum number of iterations to 200.

iii. For each of the three measurements, reconstruct the original image using the recovered $\mathcal{F}$, as you did in problem 1a. Compute the error $E = \sum_{i,j}(I(i,j) - I_{rec}(i,j))^2$, where $I(i,j)$ and $I_{rec}(i,j)$ are the (i,j)th pixel value in the original and recovered image respectively. Report this error.

iv. You should find that the IHT solution is considerably worse than the ℓ_1 minimization result. Can you give any intuition as to why this is so?

Submission Instructions

Problem 1a

You are going to recover images for 3 different measures of P and R. In the `data` folder for Problem 1, you will find three folders 1639, 2048, 4096. Each of these folders have three variable P, R and S which you need for solving the problem.

1. Write a function `p1a_getF(P, R, lambda)` which returns the recovered $\mathcal{F}$. It takes P, R and λ as inputs.

2. Write a main script `p1a_Main.m`, which loads matrices, pass them to `p1_getF()` function and then reconstruct the image using the recovered $\mathcal{F}$. Running this script should get the job done. Show the recovered image, compute errors and any other necessary thing.

3. In your report, report the reconstruction error.

4. Submit the reconstructed image in `result` folder of problem 1. You should submit both an image file as well as the actual recovered image matrix, *Irec*. Name them as `I_1639a.png` and `I_1639a.mat` respectively.

5. The image recovery and reconstruction error needs to be done for all three measurements, 1639, 2048 and 4096. Name your recovered image file accordingly.

Problem 1b

You are going to recover images for 3 different measures of P and R. In the `data` folder for Problem 1, you will find three folders 1639, 2048, 4096. Each of these folders have three variable P, R and S which you need for solving the problem.

1. Write a function `iht(P, R, K, niter )` which implements IHT algorithm. It takes as input P, R, K (sparsity level), n_{iter} = number of iterations as inputs.

2. Write a main script `p1b_Main.m`, which does rest of the stuff, similar to `p1a_Main.m`.

3. Use $n_{iter} = 200$.

4. In your report, report the reconstruction error for all 3 measurements.

5. Submit the recovered image for all three measurements in `result` folder of problem 1. Name your recovered image file properly, `I_1639b.png` etc.

2 EM and Shift-Invariant Models

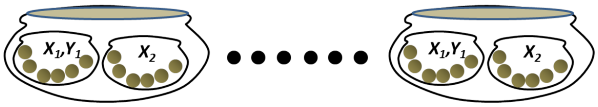
In this problem we will consider shift-invariant mixtures of multi-variate multinomial distributions.

Consider data that have multiple discrete attributes. "Discrete" attributes are attributes that can take only one of a countable set of values. We will consider discrete attributes of a particular kind – integers that have not only a natural rank ordering, but also a definite notion of distance.

Let (X, Y) be the pair of discrete attributes defining any data instance. Since both X and Y are discrete, the probability distribution of (X, Y) is a bi-variate multinomial.

We describe (X, Y) as the outcome of generation by the following process:

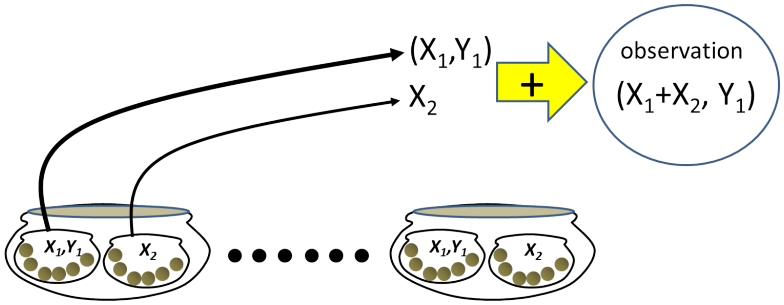
The process has at its disposal several urns. Each urn has **two** sub-urns inside it. The first sub-urn represents a bi-variate multinomial: it contains balls, such that each ball has an (X_1, Y_1) value marked on it. The second sub-urn represents a uni-variate multinomial – it contains balls, such that each ball has a X_2 value marked on it.



In the following explanation we will use the notation $P_x(X)$ to indicate the probability that the *Random Variable* x takes the value X.

We represent the content of the larger sub-urn within each urn as (x_1, y_1). The smaller sub-urn generates the random variable x_2.
Drawing procedure: At each draw the drawing process performs the following operations.

- It first randomly selects one of the larger urns according to a probability distribution $P_z(Z)$. Here Z represents the urn selected.
- Then it selects one ball each from each of the sub-urns in the selected urn. The probability of balls in the (x_1, y_1) sub-urn of the Z^{th} urn is $P_{x_1,y_1|z}(X_1, Y_1|Z)$. The probability of balls in the (x_2) sub-urn of the Z^{th} urn is $P_{x_2|z}(X_2|Z)$. Drawing from these distributions, the process obtains a (X_1, Y_1) pair from the (x_1, y_1) sub-urn, and X_2 from the x_2 sub-urn
- It finally output $(X, Y) = (X_1 + X_2, Y_1)$.



Thus, the final observation is:

$$(X, Y) = (X_1 + X_2, Y_1)$$

Representing the output random variable as (x, y), the probability that it takes a value (X, Y) is given by $P_{x,y}(X, Y)$.

Problem 2.1

Give the expression for $P_{x,y}(X, Y)$ in terms of $P_z(Z)$, $P_{x_1,y_1}(X_1, Y_1|Z)$ and $P_{x_2}(X_2|Z)$.

Problem 2.2

You are given a histogram of counts $H(X, Y)$ obtained from a large number of observations. $H(X, Y)$ represents the number of times (X, Y) was observed. Give the EM update rules to estimate $P_z(Z)$, $P_{x_1,y_1}(X_1, Y_1|Z)$ and $P_{x_2}(X_2|Z)$.

Problem 2.3

In this problem we will try to deblur a picture that has become blurry due to a slight left-to-right shake of the camera. You can find the actual picture in *hw3materials/problem2*.

We model the picture as a histogram (the value of any pixel at a position (X, Y), which ranges from 0-255, is viewed as the count of "light elements" at that position). We model this distribution as a shift-invariant mixture of one component (i.e. one large urn).

Assuming a very slight 20-pixel strictly-horizontal shake, we model that within the X_2 sub-urn X_2 can take integer values 0-19 (i.e. 20 wide). The X_1 value in the (X_1, Y_1) sub-urn can range from 0 to (width-of-picture - 20). Y_1 can take values in the range 0 to (height-of-picture - 1).

Estimate and plot $P_{x_2}(X_2)$ and $P_{x_1,y_1}(X_1, Y_1)$. You will need the solution to problem 2.2 for this problem. If the solution to problem 2.2 is incorrect, the solution of problem 2.3 will not be considered or given any points.

Submission Instructions

Problem 2.1 and 2.2.

Please submit your answers with the proper explanation as part of "Report_YourAndrewID.pdf".

Problem 2.3

Modify the script `solution2` in the directory `YourAndrewID/problem2`. The output of this script should be the deblurred image. Add to the report the figure you obtained.