RECAP

- Regularizers for images (or image like continuous-valued maps)

\[ \hat{X} = \arg \min_X \Phi(X) + R(X) = \sum \lambda_i |v_i|^p \]

where \( \{v_i\} \) are gradients / wavelet-coefficients of \( X \).

- We talked about heavier-tailed penalties: \( p < 2 \). They correspond to "truer" priors and have a non-linear shrinkage relationship

\[ s_{R,a}(v_0) = \arg \min_v \alpha(v - v_0)^2 + R(v) \]

- Shrinkage functions can be directly applied if \( v_i \) corresponds to a unitary transform (like Wavelets), and we’re talking about denoising.

- Assume \( v_i = (WX)_i \), where \( W \) is a unitary matrix representing the wavelet transform.

\[ \Phi(X) = \|X - Y\|^2 = \|WX - WY\|^2 = \sum (v_i - v_{i0})^2, \quad v_{i0} = (WX)_i \]

\[ \hat{X} = \arg \min_X \sum (v_i - v_{i0})^2 + \sum \lambda_i |v_i|^p \]

- So, we can optimize for each \( v_i \) independently.

- Denoising Algorithm: Take wavelet transform, apply shrinkage function on each coefficient independently, take inverse wavelet transform.

COURSE ADMIN

- Reminder: if you haven’t taken CSE 559A, look at the syllabus for that course and make sure you’re comfortable with the topics taught there.

- Some readings posted to course website. Cover material we’ll discuss today / next-class.
More complex for

Use conjugate gradient

IRLS

\[ \hat{X} = \arg \min_X \|AX - Y\|^2 + \lambda \sum_n |(V_x \star X)(n)|^p + \lambda \sum_n |(V_y \star X)(n)|^p \]

- Needed when gradients don’t correspond to a unitary transform, or application isn’t denoising.
- Know how to solve for \( p = 2 \)
  \[ \hat{X} = Q^{-1}B, \quad Q = A^T A + \lambda (G^T \tilde{G}_x + G^T \tilde{G}_y), \quad B = A^T Y \]
- \( Q \) is a huge matrix but can do this in Fourier domain or with conjugate gradient (recall from 559A).

More complex for \( p < 2 \)

We’ll talk about two simple optimization algorithms:
- Iterative Reweighted Least Squares (IRLS)
- Half-quadratic Splitting

IRLS

\[ \hat{X} = \arg \min_X \|AX - Y\|^2 + \lambda \sum_n |(V_x \star X)(n)|^p + \lambda \sum_n |(V_y \star X)(n)|^p \]

Re-write as

\[ \hat{X} = \arg \min_X \|AX - Y\|^2 + \lambda \sum_n w_{nx}(X)(V_x \star X)(n)^2 + \lambda \sum_n w_{ny}(X)(V_y \star X)(n)^2 \]

\[ w_{nx}(X) = \frac{1}{|(V_x \star X)(n)|^{2-p}}w_{ny}(X) = \frac{1}{|(V_y \star X)(n)|^{2-p}} \]

- Initialize weights at iteration 0.
- At iteration \( t \): you have \( \{w_{nx}^t, w_{ny}^t\} \)
  - Solve for \( X^{t+1} \) as \( \arg \min_X \|AX - Y\|^2 + \lambda \sum_n w_{nx}^t(V_x \star X)(n)^2 + \lambda \sum_n w_{ny}^t(V_y \star X)(n)^2 \)
  - Set \( w_{nx}^{t+1} = \frac{1}{|\langle V_x \star X^{t+1}, X^{t+1}\rangle|^{p-2}}, \) etc.

Repeat until convergence.

- This is a least squares problem.
- But no longer diagonalizable in the Fourier domain because different weights at each location.
- Use conjugate gradient

IRLS

\[ \hat{X} = \arg \min_X \|AX - Y\|^2 + \lambda \sum_n |(V_x \star X)(n)|^p + \lambda \sum_n |(V_y \star X)(n)|^p \]

- Initialize weights at iteration 0.
- At iteration \( t \): you have \( \{w_{nx}^t, w_{ny}^t\} \)
  - Solve for \( X^{t+1} \) as \( \arg \min_X \|AX - Y\|^2 + \lambda \sum_n w_{nx}^t(V_x \star X)(n)^2 + \lambda \sum_n w_{ny}^t(V_y \star X)(n)^2 \)
  - Set \( w_{nx}^{t+1} = \frac{1}{|\langle V_x \star X^{t+1}, X^{t+1}\rangle|^{p-2}}, \) etc.

Repeat until convergence.

- Usually converges in practice for \( 0 < p < 2 \).
  [Proof under some assumptions, from analyzing sequences of \( X^t \)]
- However, each iteration expensive because can’t use Fourier-domain computation.
HALF-QUADRATIC SPLITTING

Basic Idea

\[
\hat{X} = \text{arg min}_X F(X) + G(X)
\]

Create an "auxiliary" variable \( W \) for \( X \) for one of the costs, and re-write as:

\[
\hat{X} = \text{arg min}_X \min_W F(X) + \frac{\beta}{2} \| X - W \|^2 + G(W)
\]

As \( \beta \rightarrow \infty \), the two problems are equivalent (since the quadratic term will force \( W = X \))

Algorithm

- Begin with some initial small value of \( \beta \), and some initial guess for \( X \) or \( W \).
- Minimize the split cost in terms of \( X \) and \( W \).
- Do this by alternatingly minimizing wrt to \( X \) and \( W \).

\[
\begin{align*}
\hat{X}^{\ell+1} &= \text{arg min}_X F(X) + \frac{\beta}{2} \| X - W^{\ell} \|^2 \\
\hat{W}^{\ell+1} &= \text{arg min}_W \frac{\beta}{2} \| X^{\ell+1} - W \|^2 + G(W)
\end{align*}
\]

HALF-QUADRATIC SPLITTING

Let’s look at this for our case: (also reading: Krishnan and Fergus)

\[
\hat{X} = \text{arg min}_X \| AX - Y \|^2 + \lambda \sum_n \| (V_x \ast X)[n] \|^p + \lambda \sum_n \| (V_y \ast X)[n] \|^p
\]

Introduce auxiliary variables \( w_{\text{an}}, w_{\text{yn}} \) for \( (V_x \ast X)[n] \) and \( (V_y \ast X)[n] \).

\[
\| AX - Y \|^2 + \frac{\beta}{2} \left[ \sum_n \| (V_x \ast X)[n] - w_{\text{an}} \|^2 + \sum_n \| (V_y \ast X)[n] - w_{\text{yn}} \|^2 \right] + \lambda \sum_n \| w_{\text{an}} \|^p + \lambda \sum_n \| w_{\text{yn}} \|^p
\]

Now, let’s look at what the alternating minimization looks like:

- Wrt \( w_{\text{an}}, w_{\text{yn}} \) fixed to their current values:

\[
\text{arg min}_X \| AX - Y \|^2 + \frac{\beta}{2} \left[ \sum_n \| (V_x \ast X)[n] - w_{\text{an}} \|^2 + \sum_n \| (V_y \ast X)[n] - w_{\text{yn}} \|^2 \right]
\]

This is a least-squares minimization. Can be done in the Fourier domain. (No per-pixel weights)
We’re going to do some number of iterations for each value of $\beta$
Increase $\beta$ (typically by a constant factor each time).

What happens when $\beta$ is small?

- Minimization wrt $X$ cares more about the data term, will give noisier estimates of $X$.
- Minimization wrt $w_{xn}, w_{yn}$ will care more about the regularizer, apply more shrinkage.
- Greater disparity between actual gradients of $X$ and $w_{xn}, w_{yn}$.

What happens when $\beta$ is large?

- Minimization wrt $X$ cares more about its gradients matching $w_{xn}, w_{yn}$.
- Similarly, less shrinkage for $w_{xn}, w_{yn}$, try to match $X$ better.

\[ \|AX - Y\|^2 + \frac{\beta}{2} \left( \sum_{n} |(V_x * X)(n) - w_{xn}|^2 + \sum_{n} |(V_y * X)(n) - w_{yn}|^2 \right) + \lambda \sum_{n} |w_{xn}|^p + \lambda \sum_{n} |w_{yn}|^p \]

Some results, from applying this to deblurring.


Some final thoughts

- We’ve been using regularizers that are sum of independent penalties applied to individual gradients.

\[ R(X) = \sum_{n} R_i ((V_y * X)(n)) \]

- Prior P.O.V: We’re treating these gradients as being independent.
- Shrinkage P.O.V: How we shrink one gradient doesn’t depend on the other.
- But that may not be optimal. We might want to define a penalty on “groups of gradients”
  - Joint on $x$- and $y$- derivatives at same location
  - Joint on $x$- derivatives at a location in the R, G, and B channels of a color image

Independent Penalties:

\[ R(v_1, v_2, \ldots, v_p) = \lambda \left( |v_1|^p + |v_2|^p + \ldots |v_p|^p \right) \]

Radial Penalties

\[ R(v_1, v_2, \ldots, v_p) = \lambda \left( \sqrt{v_1^2 + v_2^2 + \ldots + v_p^2} \right)^p \]
GRADIENT PENALTY REGULARIZERS

Independent Penalties:

\[ R(v_1, v_2, \ldots, v_n) = \lambda \left( |v_1|^p + |v_2|^p + \ldots + |v_n|^p \right) \]

Radial Penalties

\[ R(v_1, v_2, \ldots, v_n) = \lambda \left( \sqrt{|v_1|^p + |v_2|^p + \ldots + |v_n|^p} \right)^p \]

- What happens when \( p = 2 \)?
  - Both are identical.
  - But not for other values of \( p \).

STORY SO FAR

- Simple regularizers: with gradient penalties.
- Encode notion that gradients should be smooth.
- Heavy tailed / norm \( p < 2 \) is better!
- But requires more complex optimization.
- Gradient penalties are essentially "hand-crafted"
  - Choosing gradient filters
  - Choosing penalty function (value of \( p \))
  - Choosing which gradients to "group together"
- But ideally, we would like to learn these from data.

LEARNED IMAGE PRIORS

- \( p(X) \): \( X \) is a random vector, where each sample of \( X \) is a natural image
- Unfortunately, images themselves are too high dimensional.
- So instead, learn a distribution of image patches \( X \).

Learned Image Priors

\[ R(X) = -\log p(X), \text{ where } p(X) \text{ is a probability distribution learned by fitting to a training set of typical images (or depth maps, flow fields, ...) } \]

- Once we have learned \( p(X) \) (and therefore \( R(X) \)), we will apply it at test time for inference.
LEARNED IMAGE PRIORS

- $p(X)$: $X$ is a random vector, where each sample of $X$ is a natural image patch.
- Let’s say we work with $8 \times 8$ patches. We want $p(X)$ to capture the spatial correlations between neighboring pixels in this patch.
  - We’re again flattening the $8 \times 8$ patch to turn $X$ into a 64-dimensional vector.

Learning a prior

- Pick a parameteric distribution for $p$: $p(X) = f(X; \theta)$
- Here, $\theta$ are some set of parameters based on the chosen form $f$
- Collect a training set of patches $\mathcal{T} = \{X_1, X_2, X_3, \ldots, X_T\}$
- Choose the values of $\theta$ based on $\mathcal{T}$
  - Most logical option: choose $\theta$ so as to maximize the likelihoods of the training set samples under $p(X)$
    $$\theta = \arg \max \prod_t f(X_t; \theta) = \arg \max \sum_t \log f(X_t; \theta)$$

GAUSSIAN PRIORS

$$p(X) = f(X; \theta = \{\mu, \Sigma\}) = \det(2\pi \Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^{-1} (X - \mu)\right)$$

- Simplest choice of parametric form.
- If $X$ is $d$-dimensional ($d = 64$ for 8x8 patches), then $\mu$ is the $d$-dimensional mean vector, and $\Sigma$ is a $d \times d$ symmetric positive definite matrix.
- How do we fit this to a training set?
  $$\theta = \arg \max \prod_t f(X_t; \theta) = \arg \max \sum_t \log f(X_t; \theta)$$
  $$\mu, \Sigma = \arg \max \frac{T}{2} \log \det(\Sigma) - \frac{1}{2} \sum_t (X_t - \mu)^T \Sigma^{-1} (X_t - \mu)$$
  - Taking derivative and setting to 0:
    $$\mu = \frac{1}{T} \sum_t X_t; \quad \Sigma = \frac{1}{T} \sum_t (X_t - \mu)(X_t - \mu)^T$$

- From training set $\{X_t\}$
    $$\mu = \frac{1}{T} \sum_t X_t; \quad \Sigma = \frac{1}{T} \sum_t (X_t - \mu)(X_t - \mu)^T$$

If you train this on randomly cropped patches from a set of natural images:

- You will typically get a mean vector $\mu$ with all elements equal.
  - Kind of makes sense, randomly shifted patches, no reason any location in the patch should have a different mean.
- We can also do an eigen-decomposition of $\Sigma$ and look at the eigenvalues and eigenvectors.
  - $\Sigma = VDV^T$. Columns of $V$ are eigenvectors, and $D$ is diagonal matrix with eigenvalues.
- You will find that the eigenvalues decay very quickly
  - Second largest eigenvalue much smaller than largest. Third much smaller than second, etc.
- Each eigen-vector is a patch-shaped object. So we can visualize them.

GAUSSIAN PRIORS

- Eigenvectors of Covariance matrix

  Looks like a Fourier Basis!

  First eigen-vector also has all equal values. Corresponds to patch mean.

  $$\mu \propto V_1$$

  $$V_i^T \mu = 0 \text{ for } i > 1$$
Ignoring terms that don’t depend on \( X \)

\[
R(X) = \frac{1}{2} (X - \mu)^T \Sigma^{-1} (X - \mu)
\]

\[
R(X) = \frac{1}{2D_{11}} \|V_i^T X - V_i^T \mu\|^2 + \sum_{i=2}^{d} \frac{1}{2D_{ii}} \|V_i^T X\|^2
\]

- The first term corresponds to the patch mean. \( D_{11} \) is really large, so its reciprocal is really small.
- Also, this "DC" component behaves least like a Gaussian. So with Gaussian priors, we often don’t include the patch mean in the regularization.

Vector to Spatial Representation

\[
V_i^T X = \sum_{n} V_i[n] X[n]
\]

- \( V_i \) for \( i > 2 \) correspond to high-frequency components of \( X \). In other words, patch gradients.
- In fact, if we applied \( R(\cdot) \) on all overlapping patches in an image, \( V_i^T X \) would correspond to convolving the image with \( V_i \) (reshaped to 8x8) as a filter.
- So we are back to squared penalty on gradients ...
- But this time, we're "learning" the set of gradient filters and corresponding weights \( \lambda \propto D_{ii}^{-1} \).