RECAP

- Regularizers for images (or image like continuous-valued maps)
  \[ \hat{X} = \arg\min_X \Phi(X) + R(X) \quad R(X) = \sum_i \lambda_i |v_i|^p \]

- We talked about heavier-tailed penalties: \( p < 2 \).

Source: Krishnan & Fergus

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.

RECAP

- Regularizers for images (or image like continuous-valued maps)
  \[ \hat{X} = \arg\min_X \Phi(X) + R(X) \quad R(X) = \sum_i \lambda_i |v_i|^p \]

- We talked about heavier-tailed penalties: \( p < 2 \). Have a non-linear shrinkage relationship

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

- But this is if you were minimizing each \( v \) independently.
This is a least squares problem for minimizing
\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n |(V_x * X)(n)|^p + \lambda \sum_n |(V_y * X)(n)|^p \]

- Recall for the case when \( p = 2 \)
  \[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \|G_x X\|^2 + \lambda \|G_y X\|^2 \]
- \( G_x, G_y \) are matrices corresponding to convolution with \( V_x \) and \( V_y \)
  - Represent a \( W \times H \) image \( X[n] \) as a single vector \( X \) with \( WH \) elements.
  - Do the same for the gradient maps.
- \( G_x \) and \( G_y \) are \( WH \times WH \) matrices which represent convolution.
- Sparse matrices: most elements are 0, and you have the same values in different columns but shifted around.
- This is a least squares problem for minimizing
  \[ X^T Q X - 2X^T B \]
  \[ Q = A^T A + \lambda (G_x^T G_x + G_y^T G_y), \quad B = A^T y \]
  \[ \hat{x} = Q^{-1} B \]
  where \( Q \) is a \( WH \times WH \) matrix and \( B \) is a \( WH \)-dimensional vector.

**IRLS**

\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n |(V_x * X)(n)|^p + \lambda \sum_n |(V_y * X)(n)|^p \]

Re-write as
\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n w_{x,n} |(V_x * X)(n)|^2 + \lambda \sum_n w_{y,n} |(V_y * X)(n)|^2 \]
\[ w_{x,n} = \frac{1}{|(V_x * X)(n)|^{2-p}}, \quad w_{y,n} = \frac{1}{|(V_y * X)(n)|^{2-p}} \]

**Optimization**

\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n |(V_x * X)(n)|^p + \lambda \sum_n |(V_y * X)(n)|^p, \quad p = 2 \]
\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \|G_x X\|^2 + \lambda \|G_y X\|^2 \]
\[ \hat{x} = Q^{-1} B, \quad Q = A^T A + \lambda (G_x^T G_x + G_y^T G_y), \quad B = A^T y \]
- \( Q \) is a huge matrix but sparse, and in this case (because of convolutions) diagonalized by the Fourier transform.
- We talked about how you can compute \( Q^{-1} B \) in the Fourier domain, or by using conjugate gradient (where at each iteration, you only have to multiply by \( Q \)).
- But what about \( p < 2 \)?
  - This is the domain of continuous optimization. A number of algorithms available (ESE 415).
  - But we’ll talk about two simple (yet effective) approaches:
    - Iterative Reweighted Least Squares (IRLS)
    - Half-quadratic Splitting

**IRLS**

\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n |(V_x * X)(n)|^p + \lambda \sum_n |(V_y * X)(n)|^p \]

Re-write as
\[ \hat{x} = \arg \min_x \|Ax - y\|^2 + \lambda \sum_n w_{x}(X) |(V_x * X)(n)|^2 + \lambda \sum_n w_{y}(X) |(V_y * X)(n)|^2 \]
\[ w_{x}(X) = \frac{1}{|(V_x * X)(n)|^{2-p}}, \quad w_{y}(X) = \frac{1}{|(V_y * X)(n)|^{2-p}} \]
- Initialize weights at iteration 0.
- At iteration \( t \): you have \( \{w_{x,t}, w_{y,t}\} \)
  - Solve for \( X^{t+1} \) as \( \arg \min_x \|Ax - y\|^2 + \lambda \sum_n w_{x,t}(X) |(V_x * X)(n)|^2 + \lambda \sum_n w_{y,t}(X) |(V_y * X)(n)|^2 \)
  - Set \( w_{x,t+1} = \frac{1}{\sum_n w_{x,t}(X)|V_x * X(n)|^{2-p}}, \text{etc.} \)
- Repeat until convergence.
IRLS

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

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

HALF-QUADRATIC SPLITTING

Basic Idea

\[ \hat{X} = \arg \min_F(X) + G(X) \]

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

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

As \( \beta \to \infty \), 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 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 \( W \).
    \[ \hat{X}^{t+1} = \arg \min_X F(X) + \frac{\beta}{2} \|X - W^t\|^2 \]
    \[ W^{t+1} = \arg \min_W \frac{\beta}{2} \|X^{t+1} - W\|^2 + G(W) \]
  - Set \( w_{x,t+1}^{t+1} = \frac{1}{\langle V_x * X^t(n), V_x(n) \rangle} \), etc.

Repeat until convergence.

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

HALF-QUADRATIC SPLITTING

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

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

Repeat until convergence.

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

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

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

Introduce auxiliary variables $w_{x\,n}$, $w_{y\,n}$ for $(V_x \ast X)[n]$ and $(V_y \ast X)[n]$.

$$\|AX - Y\|^2 + \frac{\beta}{2} \left[ \sum_n |(V_x \ast X)[n] - w_{x\,n}|^2 + \sum_n |(V_y \ast x)[n] - w_{y\,n}|^2 \right] + \lambda \sum_n |w_{x\,n}|^p + \lambda \sum_n |w_{y\,n}|^p$$

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

- Wrt $X$ with $\{w_{x\,n}, w_{y\,n}\}$ fixed to their current values:

$$\arg\min_{X} \|AX - Y\|^2 + \frac{\beta}{2} \left[ \sum_n |(V_x \ast X)[n] - w_{x\,n}|^2 + \sum_n |(V_y \ast X)[n] - w_{y\,n}|^2 \right]$$

This is a least-squares minimization. Can be done in the Fourier domain. (No per-pixel weights)

HALF-QUADRATIC SPLITTING

$$\|AX - Y\|^2 + \frac{\beta}{2} \left[ \sum_n |(V_x \ast X)[n] - w_{x\,n}|^2 + \sum_n |(V_y \ast X)[n] - w_{y\,n}|^2 \right] + \lambda \sum_n |w_{x\,n}|^p + \lambda \sum_n |w_{y\,n}|^p$$

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

- Wrt $\{w_{x\,n}, w_{y\,n}\}$:

$$\arg\min_{\{w_{x\,n}, w_{y\,n}\}} \frac{\beta}{2} \left[ \sum_n |(V_x \ast X)[n] - w_{x\,n}|^2 + \sum_n |(V_y \ast X)[n] - w_{y\,n}|^2 \right] + \lambda \sum_n |w_{x\,n}|^p + \lambda \sum_n |w_{y\,n}|^p$$

There are no cross-terms between the different gradient. So you can minimize them independently.

$$w_{x\,n} = \arg\min_w \frac{\beta}{2} (w - (V_x \ast X[n]))^2 + \lambda |w|^p$$

- This is our shrinkage function
- 1-D optimization, can pre-compute and build a look-up table (for different values of $\beta$).

So, each inner step is fast: either in the Fourier domain, or independent 1-D optimizations.

HALF-QUADRATIC SPLITTING

Some results, from applying this to deblurring.

HALF-QUADRATIC SPLITTING

Some results, from applying this to deblurring.


GRADIENT PENALTY REGULARIZERS

Some final thoughts

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

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

- 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- derivative at same location
  - Joint on x-derivatives at a location in the R, G, and B channels of a color image

Independent Penalties:

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

Radial Penalties

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

- What happens when \( p = 2 \)?
  
Both are identical.

But not for other values of \( p \).

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

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

- What does the shrinkage function for this look like?

\[ \{v_1, v_2\} = s(\bar{v}_1, \bar{v}_2) = \arg\min_{v_1, v_2} (v_1 - \bar{v}_1)^2 + (v_2 - \bar{v}_2)^2 + \lambda \left( \sqrt{v_1^2 + v_2^2} \right)^p \]

\[ v_1/v_2 = \bar{v}_1/\bar{v}_2 \]

Broadly, you apply a similar 1-D non-linear shrinkage on the “radius”:

\[ \sqrt{v_1^2 + v_2^2} = s \left( \sqrt{\bar{v}_1^2 + \bar{v}_2^2} \right) \]

- If \( \bar{v}_1 \) and \( \bar{v}_2 \) are both small: independent and joint costs will both shrink them to 0.
- If \( \bar{v}_1 \) and \( \bar{v}_2 \) are both large: independent and joint costs will both leave them at their observed values.
- If \( \bar{v}_1 \) is small and \( \bar{v}_2 \) is large:
  
**Old Version:** Shrink \( v_1 \) to 0, leave \( v_2 \) alone.

**New Version:** Make a joint decision to shrink or not-shrink both.

- Can use this with HQS (or even IRLS).