Computer Vision II - Homework Assignment 3

Stefan Roth, Junhwa Hur, Faraz Saeedan
Visual Inference Lab, TU Darmstadt

May 31, 2016

This homework is due on June 15, 2016 at 23:59.
Please read the general instructions from HW1 carefully!

Note:

- Use the provided files to implement your functions. Do not remove any existing lines of code inside those files.
- Please include one file/sheet with all your written solutions.

Problem 1 - Gradient of the Log-Prior - 10 points

In class we derived the gradient of the log-prior with respect to all the pixels in the image. In Assignment 2 we computed the unnormalized density of an MRF prior with Student-t potentials. Now we want to compute the gradient of an MRF log-prior with Student-t potentials. Use the values $\sigma = 10$ and $\alpha = 1$ for this problem.

Tasks:

- Implement the gradient of the log-prior from Assignment 2. In particular, write a function

  ```
  g = mrf_grad_log_prior(x, sigma, alpha)
  ```

  that computes the gradient of the log-prior density w.r.t. every pixel in `x` with the given parameters. Note that the output array `g` should have the same dimensions as the input image. Implement first the function

  ```
  f = dx_studentt(x, sigma, alpha)
  ```

  and use this function to calculate the gradient of the log-prior.

  Test the function `mrf_grad_log_prior` by comparing the analytical gradient with a numerical approximation. Say we want to test whether the gradient for a particular pixel (k, l) of $\mathbf{x}$ is correct. We create two test images $\mathbf{x}^p$ and $\mathbf{x}^m$ with

  $$\mathbf{x}^p_{i,j} = \begin{cases} x_{i,j} + \epsilon & i = k, j = l \\ x_{i,j} & \text{otherwise} \end{cases} \tag{1}$$

  and

  $$\mathbf{x}^m_{i,j} = \begin{cases} x_{i,j} - \epsilon & i = k, j = l \\ x_{i,j} & \text{otherwise} \end{cases} . \tag{2}$$

Then use the fact that the partial derivative can be approximated as

$$\frac{\partial}{\partial x_{k,l}} \log p(\mathbf{x}) \approx \frac{\log p(\mathbf{x}^p) - \log p(\mathbf{x}^m)}{2\epsilon}, \tag{3}$$

where ϵ is small (say 10^{-5}). The difference between the analytical gradient (i.e. `g(k,l)`) and the numerical approximation from Eq. (3) should be small (in absolute value). To help you with this step, we have included `testgrad.jl`.

5 points

- Compute and display (as an image) the gradient of the log-prior density for the image `la.png`.

1 point

- Create a random noise image of the same dimensions as `la.png` by drawing uniform random numbers in $[0, 255]$ independently for each pixel. Compute and display (as an image) the gradient of the log-prior density for this noise image.

1 point

- Create a constant image ($\mathbf{x} \equiv 127$) again of the same dimensions and also compute and report the gradient of the log-prior density.

1 point

- Comment on what kinds of features the gradient brings out for each of the images.

2 points

Notes:

- Make sure that you treat pixels on the boundary appropriately. While all pixels in the interior have 4 potential functions that contribute to the gradient for that pixel, boundary pixels only have 3 potentials associated with it, and corner pixels only 2. The test procedure will help you get this right.
- One common mistake is to flip the signs of the gradients. Pay close attention to the direction in which you compute the gradients of the potentials.

Problem 2 - Image Denoising with MRFs - 20 points

In this problem, you will use the image prior of problem 1 to perform image denoising. In particular, we will assume that images are corrupted by pixelwise independent Gaussian noise with standard deviation σ_N. Hence, we can write an appropriate likelihood model (ignoring any constants) as

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_{i=1}^{N}\prod_{j=1}^{M} \exp\left\{-\frac{1}{2\sigma_N^2}(x_{i,j}-y_{i,j})^2\right\}. \tag{4}$$

We combine this likelihood with our prior from above to get the posterior

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}) \cdot p(\mathbf{x}). \tag{5}$$

For computational convenience we will work with the log of the posterior (ignoring const.):

$$\log p(\mathbf{x}|\mathbf{y}) = \log p(\mathbf{y}|\mathbf{x}) + \log p(\mathbf{x}) + \text{const.} \tag{6}$$

Tasks:

- Implement the Gaussian likelihood model for denoising. In particular write the functions

  ```
  llh = denoising_llh(x, y, sigmaN)
  ```

  to compute the log-likelihood and

  ```
  g = denoising_grad_llh(x, y, sigmaN)
  ```

  to compute the gradient. Use the verification method(comparison between numerical and analytic versions) explained in Problem 1 to make sure that you implemented the gradient correctly. In order to speed up the verification procedure use only a random subset of pixels.

  6 points

- Implement a denoising algorithm

  ```
  xtcurrent = gradientascent(x, sigmaN, sigma, alpha)
  ```

  using gradient ascent on the log posterior, i.e.

  $$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta \cdot \nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{y}). \tag{7}$$

  5 points

- Add Gaussian noise of $\sigma_N = 15$ to `la.png` and remove it using your denoising algorithm. The parameter of the likelihood is given by the standard deviation of the added noise, but you need to experimentally find appropriate parameters σ and α for the prior (try $\sigma = 10$ and $\alpha = 1$ to get started). Make sure you use a small enough step size to stay out of numerical problems ($\eta = 0.1$ or possibly even smaller). Also make sure to use enough iterations to reach approximate convergence. Show the denoising result and explain your observations from tuning the parameters.

  6 points

- Find one other parameter setting that leads to different characteristics of the denoised image (e.g. smoother, but less crisp boundaries), denoise the image from above and show the result. Explain how the two results differ and discuss why neither of the results is optimal.

1 points

- To measure the quality of your denoised image you will implement the *peak signal-to-noise ratio* (PSNR)

  `p = psnr(x_gt, x)`,

  which is defined as

$$\text{PSNR} = 10 * \log_{10}\left(\frac{v_{\text{Max}}^2}{\text{MSE}}\right), \tag{8}$$

  where v_{Max} is the maximally possible pixel intensity value, and

$$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(X(i,j) - Y(i,j)\right)^2 \tag{9}$$

  is the mean squared error between the original (`x_gt`) and the denoised image (`x`). Compute and display the PSNR for both denoised images and comment on your findings.

2 points

Notes:

- You are not allowed to use any optimization tool.
- Since gradient ascent is an iterative procedure, you should implement it with a for loop.
- Command useful for this problem includes `randn`.
- You should start the gradient ascent with the noisy image.
- You may need many iterations to reach approximate convergence, possibly thousands. It is up to you to decide whether to monitor convergence manually or automatically. It might be helpful to inspect the intermediate denoising results by displaying them (e.g. in every 100th iteration).
- It is a good idea to verify that the log-posterior is actually increasing after each iteration. If it is not, your step size is too large.

Problem 3 - Image Inpainting with MRFs - 10 points

In this problem you will implement a simple image inpainting algorithm in which you fill in missing pixels using the prior alone. To do this, the likelihood leaves all the "good" pixels untouched and is uniform for all the "bad" pixels:

$$p(\mathbf{y}|\mathbf{x}) = \begin{cases} \delta(x_{i,j} - y_{i,j}), & (i,j) \notin M \\ \text{const}, & (i,j) \in M \end{cases} \tag{10}$$

Here, M denotes the mask of all "bad" pixels.

Tasks:

- Implement a gradient ascent inpainting algorithm

  ```
  gradientascent(x, M, sigma, alpha)
  ```

  that combines this likelihood with the MRF prior from above (in the log-domain). In particular implement the following gradient ascent scheme:

  $$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta \cdot M * \nabla_{\mathbf{x}} \log p(\mathbf{x}), \tag{11}$$

  where $M*$ is the pixelwise product of the mask with the gradient.

  7 points

- Randomly remove 50% of the pixels of `castle.png` by creating a binary mask M and setting all masked pixels of the image to 127. Run gradient ascent until approximate convergence and show the image before and after inpainting. Also compute and display the PSNR, which you implemented in problem 2. You should use the same parameters of the prior that you used for denoising; in other terms this does not require any parameter tuning.

  2 points

- Try to do the same, but with 80% of the pixels removed. Show the results and comment on them. Also compute and display the PSNR, which you implemented in problem 2.

  1 points

Notes:

- You should start the gradient ascent with the masked image.
- You might need even more iterations than for denoising, but here you can "get away" with using a large step size initially (possibly as big as $\eta = 5$) to quickly get toward a good solution and then use a few hundred clean-up iterations with a small step size like above.

Bonus Problem 4 - Loss-based Training for Image Denoising - 20 points

This problem continues from Problem 2, so make sure to do that first. Note that this problem is somewhat more difficult compared to the previous ones. Here, you will use loss-based training to achieve better denoising results in fewer iterations of gradient ascent. In particular, you will learn the parameters $\boldsymbol{\theta} = \{\sigma, \alpha\}$ of the Student-t potential such that the PSNR of the denoised image is maximized after every iteration of gradient ascent.

Our goal is to obtain optimal parameters $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ via gradient-based minimization of the learning objective function

$$J(\boldsymbol{\theta}) = L(\mathbf{x}_{\mathrm{GT}}, f_{\boldsymbol{\theta}}(\mathbf{y})) \quad \text{with prediction function} \tag{12}$$

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{x} + \eta \cdot \nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{y}; \boldsymbol{\theta}) \quad \text{and loss function} \tag{13}$$

$$L(\mathbf{x}_{\mathrm{GT}}, \mathbf{x}) = -\mathrm{PSNR}(\mathbf{x}_{\mathrm{GT}}, \mathbf{x}). \tag{14}$$

Note that we only do learning with a single training example $\mathcal{D} = \{(\mathbf{x}_{\mathrm{GT}}, \mathbf{y})\}$ where $\mathbf{x}_{\mathrm{GT}}$ denotes the ground truth image and $\mathbf{y}$ the noisy image that we observe in practice. We have written the posterior as $p(\mathbf{x}|\mathbf{y}; \boldsymbol{\theta})$ to make the dependence on parameters $\boldsymbol{\theta}$ explicit. The prediction function $f_{\boldsymbol{\theta}}(\mathbf{x})$ does just one step of gradient ascent on the log posterior.

Tasks:

- Implement $\nabla_{\mathbf{x}} L(\mathbf{x}_{\mathrm{GT}}, \mathbf{x})$ by writing the function

  `g = grad_loss(x_gt, x)`.

  2 points

- Implement $f_{\boldsymbol{\theta}}(\mathbf{x})$ and $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})$ by writing the function

  `[f, dsigma, dalpha] = prediction(x, y, sigmaN, sigma, alpha)`.

  Here, `dsigma` and `dalpha` denote the partial derivatives w. r. t. σ and α, respectively. Use $\eta = 1$ and note that `dsigma` and `dalpha` each have the same size as the image.

  7 points

- Implement $J(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ by writing the function

  `[J, g] = learning_objective(x_gt, x, y, sigmaN, sigma, alpha)`

  where `J` is a scalar value and `g = [dsigma, dalpha]` is a 2×1 column vector. Make use of the two functions that you have written before.

  3 points

- Load the image `la.png` as $\mathbf{x}_{\mathrm{GT}}$ and `la-noisy.png` as $\mathbf{y}$. Find the optimal parameters $\hat{\boldsymbol{\theta}}$ by minimizing the objective function with `optimize()`.

  What are the values of the learned parameters $\hat{\sigma}$ and $\hat{\alpha}$? What is the PSNR when you denoise $\mathbf{y}$ with those parameters? Show the denoised image.

  1 point

- After learning the model parameters $\hat{\boldsymbol{\theta}}$ for the noisy image $\mathbf{y}$, can we expect similarly good results for a new noisy test image $\bar{\mathbf{y}}$? Briefly explain your answer.

1 point

- While the denoised image after 1 step of gradient ascent (with learned parameters) already looks reasonable, your result from Problem 2 is probably still better. To overcome this, you will now learn parameters $\hat{\boldsymbol{\theta}}^{(t)}$ with $t = 1, \ldots, T$ for the first $T = 10$ steps of gradient ascent, one step at a time. At each step t, the learning objective function is

$$J(\boldsymbol{\theta}^{(t)}) = L(\mathbf{x}_{\mathrm{GT}}, f_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(t-1)})) \quad \text{where} \tag{15}$$

$$\mathbf{x}^{(t)} = f_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(t-1)}) \quad \text{and} \tag{16}$$

$$\mathbf{x}^{(0)} = \mathbf{y} \tag{17}$$

  Note that what you have already implemented is a special case of this with $T = 1$. Show the denoised image $\mathbf{x}^{(t)}$ (obtained with learned parameters $\hat{\boldsymbol{\theta}}^{(t)}$) and its PSNR after each iteration t. Is the PSNR of the denoised image after 10 steps better than what you have obtained in Problem 2? Why is the comparison not fair?

4 points

- Assuming that you always find the global minimum of the learning objective function, are the denoised images $\mathbf{x}^{(t)}$ optimal (in terms of PSNR), i.e. there are no other model parameters $\boldsymbol{\theta}^{(t)}$ that lead to better results? Hint: Consider cases $T = 1$ and $T > 1$ separately.

2 points

Note:

- Use the Function optimize() from optim.jl