Computer Vision II - Homework Assignment 3

Stefan Roth, Stephan Richter, Anne Wannenwetsch
Visual Inference Lab, TU Darmstadt

June 14, 2017

This homework is due on June 29th, 2017 at 9:00.
Please read the instructions 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 - Image Denoising with MRFs - 19 points

In this problem, you will use the image prior from the last assignment 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

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

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

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}) \cdot p(\mathbf{x}). \tag{2}$$

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

$$\log p(\mathbf{x}|\mathbf{y}) = \log p(\mathbf{y}|\mathbf{x}) + \log p(\mathbf{x}) + \text{const.} \tag{3}$$

Tasks:

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

  ```
  nllh = mrf_denoising_nllh(x, y, sigma_noise)
  ```

  to compute the negative log-likelihood and

  ```
  g = grad_mrf_denoising_nllh(x, y, sigma_noise)
  ```

  to compute the gradient.

6 points

Notes on optimizers: - LBFGS ≈ 23 → super slow!
BFGS doesn't work!
- Accel. Grad. Desc. ≈ 23,1 → fast
- Mom. Grad. Desc. ≈ 22,5 → medium
- Grad. Des. ≈ 22,3 → slow

- Implement a denoising algorithm

  ```
  y = denoise(x, sigma_noise, sigma, alpha)
  ```

  using gradient descent on the negative log posterior, i.e.

  $$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta \cdot \nabla_{\mathbf{x}} - \log p(\mathbf{x}|\mathbf{y}). \tag{4}$$

  You can again use Julia's `Optim` package to perform the optimization.

  4 points

- Add Gaussian noise of $\sigma_{\texttt{noise}} = 15$ 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). If you are implementing gradient descent yourself, 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.

  5 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.

  2 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

  $$\text{PSNR} = 10 * \log_{10}\left(\frac{v_{\text{Max}}^2}{\text{MSE}}\right), \tag{5}$$

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

  $$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(X(i,j) - Y(i,j))^2 \tag{6}$$

  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

PSNR - SCORE

Notes:

- You should start the gradient descent 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 negative log-posterior is actually decreasing after each iteration. If it is not, your step size is too large.

Problem 2 - Image Inpainting with MRFs - 8 points

In this problem you will implement a simple image inpainting algorithm in which you fill in missing pixels using the prior. Intuitively, we want to leave "good" pixels untouched and estimate the intensities at "bad" pixels. Hence, the likelihood is constant everywhere and we maximize the posterior probability of intensities at "bad" pixels. Selecting bad pixels with mask m, we get:

$$m(\mathbf{y}) \circ p(\mathbf{x}|\mathbf{y}) \propto m(\mathbf{y}) \circ p(\mathbf{x}), \tag{7}$$

where $\circ$ denotes the Hadamard product.

Tasks:

- Implement a gradient descent inpainting algorithm

  ```
  x' = inpaint(x, m, sigma, alpha)
  ```

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

  $$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta \cdot m * \nabla_{\mathbf{x}} \log p(\mathbf{x}), \tag{8}$$

  where $M*$ is the pixelwise product of the mask with the gradient. As before, it is ok to use the `Optim` package.

  5 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 descent until approximate convergence and show the image before and after inpainting. Also compute and display the PSNR, which you implemented in problem 1. 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 descent 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.

Problem 3 - Loss-based Training for Image Denoising - 20 points

This problem continues from Problem 1, 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 descent. 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 descent.

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

$$\text{of the learning objective function} \qquad J(\boldsymbol{\theta}) = L(\mathbf{x}_{\text{GT}}, f_{\boldsymbol{\theta}}(\mathbf{y})) \tag{9}$$

$$\text{with prediction function} \qquad f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{x} + \eta \cdot \nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{y}; \boldsymbol{\theta}) \tag{10}$$

$$\text{and loss function} \qquad L(\mathbf{x}_{\text{GT}}, \mathbf{x}) = -\text{PSNR}(\mathbf{x}_{\text{GT}}, \mathbf{x}). \tag{11}$$

Note that we only do learning with a single training example $\mathcal{D} = \{(\mathbf{x}_{\text{GT}}, \mathbf{y})\}$ where $\mathbf{x}_{\text{GT}}$ 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 $f_{\boldsymbol{\theta}}(\mathbf{x})$ in Eq. (10) does just one step of gradient descent on the negative log posterior.

Tasks:

- Implement $\nabla_{\mathbf{x}} L(\mathbf{x}_{\text{GT}}, \mathbf{x})$ 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.

  6 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.

  2 points

- Load the image `la.png` as $\mathbf{x}_{\text{GT}}$ and `la-noisy.png` as $\mathbf{y}$. Write a function

  ```
  find_optimal_params(x_gt, x, y, sigmaN, theta0)
  ```

  that finds the optimal parameters $\hat{\boldsymbol{\theta}}$ by minimizing the objective function using an appropriate starting value `theta0`. To minimize the objective function apply `optimize()` from Optim.jl with options `method = LBFGS()` and `iterations = 50`.

  2 points

- 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 descent (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 descent, one step at a time. At each step t, the learning objective function is

$$J(\boldsymbol{\theta}^{(t)}) = L(\mathbf{x}_{\mathrm{GT}}, f_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(t-1)})) \quad \text{where} \tag{12}$$

$$\mathbf{x}^{(t)} = f_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(t-1)}) \quad \text{and} \tag{13}$$

$$\mathbf{x}^{(0)} = \mathbf{y} \tag{14}$$

  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.

3 points

- 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?

1 point

- 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