Homework 3

CSC2529 Computational Imaging 2023

Due: Oct. 18, 2023 at 11:59pm

Learning Goals

In this homework, you will learn how to implement image convolutions in Python and how to measure the time it takes for various algorithms to run. You will experiment with spatial-domain and Fourier-domain implementations of convolutions and implement intuitive deconvolution algorithms that are directly derived from the convolution theorem. Finally, you will implement a simple least squares solver using gradient descent and stochastic gradient descent and learn the tradeoffs between these approaches.

Instructions

In this week's problem session, we will walk you through this homework step by step. It may be helpful to watch the problem session before you start working on this homework or ask questions on Piazza.

This is a programming assignment. Students are strongly encouraged to use Python for this assignment. Other programming environments may generally also be acceptable, but will not be supported in the office hours or on Piazza.

You should document all your answers, plots, and derivations in a **single pdf** file containing all requested results and the scripts that generated these results. Submit your solution to Quercus. Solutions to tasks should be on separate pages and include text, images, and code.

Task 1 of 3 (15 points)

Implement image filtering using convolutions implemented with both spatial-domain and Fourier-domain computations. Use the example image 'birds_gray.png'.

Use the commands `scipy.signal.convolve2d`, `pypher.psf2otf`, `numpy.fft.fft2` and `numpy.fft.ifft2`.

1. Implement low-pass filtering for three Gaussian convolution kernels with the following standard deviations: 0.1, 1, and 10. Use a kernel size that is 9 times larger than the standard deviation in each case (i.e. `kernelSize = ceil(9.*[sigma sigma])`, see starter code). Use the function `fspecial_gaussian_2d` to generate the blur kernels.

   Plot the resulting images and also report timings for both spatial and Fourier convolutions. For the timings, submit a bar plot that shows the timings of all 3 convolutions in both spatial and Fourier domain, so 6 bars total. What insights do you draw from looking at these timings in the bar plots?

2. Implement high-pass filtering for three different kernels. The choice of specific kernel parameters (radius etc.)

is up to you, just make sure to test three different kernels. Use the same kernel size of 101×101 pixels for each of them. Plot the resulting images and report timings.

Task 2 of 3 (35 points)

Implement inverse filtering and Wiener deconvolution. Use the example image 'birds_gray.png'. Blur the image with a Gaussian PSF with a standard deviation of 5. Show four examples with the same blur kernel but add random, additive Gaussian noise with the following standard deviations to the blurred image, before it is inversely filtered: 0, 0.001, 0.01, 0.1. Restore the image by:

1. Inverse filtering using division in the Fourier domain.
2. Wiener deconvolution. This is almost the same as inverse filtering, but uses a damping factor in the Fourier domain that depends on the signal-to-noise-ration (SNR).

Plot the resulting images and report the peak signal-to-noise ratio (PSNR) for each result.

Task 3 of 3 (50 points)

Implement gradient descent and stochastic gradient descent to solve a linear inverse problem of the form $Ax = b$. Use the starter code in `hw3_task3.py`.

- First, implement gradient descent (GD) to find $\underset{x}{\operatorname{argmin}} \frac{1}{2}\|Ax - b\|_2^2$. In the starter code we generate values for A and b and you can use the default gradient descent step size and number of iterations. Also, note that we include code to compute the least squares solution to $Ax = b$ using the SVD and a standard matrix factorization approach. Your solution with GD should achieve roughly the same residual as the provided solution.
- Then, implement stochastic gradient descent (SGD). For each iteration of SGD, you will sample a fixed number of random rows from A (and corresponding rows from b) and then use the resulting matrix to calculate the gradient descent update. The number of rows sampled from A is the "batch size" in our SGD optimization. Use the same step size and number of optimization steps as for GD.
- In your writeup, include (1) your code, (2) plots of the residual vs optimization iteration for GD and SGD with batch sizes of 10, 100, and 1000 (show these on the same axes), and (3) another similar set of plots, but with residual vs. execution time. You can use the `time` python package to roughly measure the time it takes to run each iteration.
  - Also include answers to the following questions. How does the performance of GD compare to SGD? Is there a tradeoff with using a smaller or larger batch size in the optimization?

Bonus Task (up to 5 points extra)

In Task 3 we used gradient descent to minimize an L2 objective function. One other popular choice is the L1 objective where we seek to find $\underset{x}{\operatorname{argmin}} \frac{1}{2}\|Ax - b\|_1$. One benefit of the L1 objective is that it is less sensitive to outliers in the data that have a large residual. Here we will investigate optimization with the L1 objective function.

- Derive $\nabla_x \|Ax - b\|_1$ and implement a Python function that calculates this gradient given A, x, and b.

- What is the gradient when one or more of the entries of $Ax - b$ is zero? What could your gradient function return in that case? (*Hint: see this Wikipedia article https://en.wikipedia.org/wiki/Subderivative and the discussion in this PyTorch issue https://github.com/pytorch/pytorch/issues/7172.*)
- Show that your solution works by finding $\underset{x}{\text{argmin}} \|Ax - b\|_1$ using gradient descent and plotting the L1 residual vs. iteration number.

Questions?

First, review the lecture slides and videos as well as the problem session because the answer to your question is likely in there. If you don't find it there, post on Piazza, come to office hours, or email the course staff mailing list (in that order).