Homework 6

CSC2529: Computational Imaging 2023

Due: Nov. 15, 2023 at 11:59 pm

Learning Goals

In this homework, you will learn how to solve inverse problems using iterative optimization with regularizers. For this purpose, we will explore the specific problems of 2D image deconvolution and single-pixel imaging. Single-pixel imaging is a fun problem but really a placeholder for any inverse problem, which you will solve using the tools of this week's lectures and homework. We will explore different solvers, including the Adam solver that is built into PyTorch and the alternating direction method of multipliers (ADMM). We evaluate these solvers using total variation (TV) regularizers as well as pre-trained denoising priors. All of this will give you a good intuition for two really good solvers used in combination with several different regularizers that will prepare you to solve almost any inverse problem.

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: Image deconvolution using Adam and the TV prior (20 points)

Implement an Adam-based solver for image deconvolution with TV regularization. The Adam solver is already implemented in PyTorch, which runs either on a CPU or on a GPU without any changes to your code. You will find starter code in the homework release folder. Specifically, edit the files "hw6_task1.py" and "deconv_adam_tv.py" for this task.

1. In the file "deconv_adam_tv.py", implement the expression for the *anisotropic* TV term. For this purpose, first implement the function `grad_fn` to compute the horizontal and vertical gradients of an image and then implement the aggregation function of the anisotropic TV term to compute a single scalar value that you should assign to the variable `loss_regularizer` in this file. Submit the deconvolved RGB image as well as the PSNR using the default parameters set in the file "hw6_task1.py".

2. In the file “deconv_adam_tv.py”, implement the expression for the *isotropic* TV term. For this purpose, use your implementation of the function `grad_fn` again to compute the horizontal and vertical gradients of an image and then implement the aggregation function of the isotropic TV term to compute a single scalar value that you should assign to the variable `loss_regularizer` in this file.

Submit your code and the deconvolved RGB image as well as the PSNR using the default parameters set in the file “hw6_task1.py”. Compare the results you achieved with the anisotropic and isotropic TV terms – how different are they?

A piece of advice: do not try to use the function `torch.sqrt()` to implement your TV terms, because the gradient of this function at 0 is `nan`. Use the PyTorch functions `torch.abs()`, `torch.norm()`, and `torch.sum()` instead.

Task 2 of 3: Image deconvolution using ADMM (40 points)

Implement ADMM-based solvers for image deconvolution with TV and DnCNN regularization. You will find starter code in the homework release folder. Specifically, edit the files “hw6_task2.py”, “deconv_admm_tv.py”, and “deconv_admm_dncnn.py” for this task.

1. In the file “deconv_admm_tv.py”, copy over your implementation of `grad_fn` from task 1. As discussed in the comments of this file, please make sure that this function takes as input a grayscale 2D image of size [M N] and outputs both horizontal and vertical gradients stacked in a numpy array of size [2 M N].
2. In the file “deconv_admm_tv.py”, implement the x-update. For this purpose, you can compute the denominator of the expression discussed in class, course notes, and problem session once (where indicated) and then compute the nominator in every iteration. Combine these two terms and implement the full x-update (where indicated).
3. In the file “deconv_admm_dncnn.py”, implement the x-update. For this purpose, you can compute the denominator of the expression discussed in class, course notes, and problem session once (where indicated) and then compute the nominator in every iteration. Combine these two terms and implement the full x-update (where indicated).

Submit your code and the deconvolved RGB images as well as the PSNR using the default parameters set in the file “hw6_task2.py”.

Task 3 of 3: Single-pixel imaging using ADMM (40 points)

Implement three solvers for single-pixel imaging using a compression factor of 4×, i.e., where the number of measurements M is four times lower than the number of unknowns N. You will find starter code in the homework release folder. Specifically, edit the files “hw6_task3.py”, “leastnorm.py”, “admm_tv.py”, and “admm_dncnn.py” for this task.

1. In the file “leastnorm.py”, compute the least-norm solution using the conjugate gradient (CG) solver implemented by `scipy.sparse.linalg.cg`. For this purpose, you need to formulate function handles based on the functions `Afun` and `Atfun` and pass them into the call of the `cg` function. Follow the comments in the starter code.
2. In the file “admm_tv.py”, implement the ADMM solver with the TV prior. We already implemented the z-

and u-updates for you, but you still need to implement the x-update. We can't use inverse filtering here, so you need to use the function `scipy.sparse.linalg.cg` to implement an iterative CG-based solver to compute the x-update. Follow the comments in the starter code.

3. In the file "admm_dncnn.py", implement the ADMM solver with the DnCNN prior. Again, we already implemented the z- and u-updates for you, but you still need to implement the x-update. We can't use inverse filtering here, so you need to use the function `scipy.sparse.linalg.cg` to implement an iterative CG-based solver to compute the x-update. Follow the comments in the starter code.

Submit your code and the reconstructed images as well as the PSNR using the default parameters set in the file "hw6_task3.py".

Bonus Task (up to 5 points extra)

Experiment with an unrolled HQS network in the context of the image deconvolution problem of task 2. For this purpose, you can fix the number of HQS iterations K to a small number (whatever you can fit on your GPU) and work with a pre-trained DnCNN network. Learn hyper-parameters $\lambda^{(k)}$ and $\rho^{(k)}$ for each of the $k = 1 \ldots K$ iterations using a suitable training set that does not contain the birds test image. Compare your results with the results from task 2 using the birds test image.

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).