

Homework 5

EE367/CS448I Computational Imaging 2026

Due: 02/13/2026 at 11:59PM

Instructions

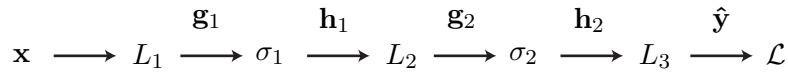
In this week's problem session, we will walk you through this homework step by step. It may be helpful to watch the problem session before you start working on this homework or ask questions on Ed Discussion.

This is mostly 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 Ed Discussion. Please see the course website for additional details on computing resources if you need help installing or using Python.

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 Gradescope. Solutions to tasks should be on separate pages and include text, images, and code.

Task 1 of 3 (60 points)

In this question we will analyze and implement a simple fully connected neural network, depicted below.



Here, \mathbf{x} is an input vector, and L_i and σ_i represent the linear layers and non-linearities of the network, respectively. The intermediate outputs of the network are given by the bold symbols above the arrows. The network outputs a vector, $\hat{\mathbf{y}}$, which is then input into a loss function \mathcal{L} .

Let the outputs of the linear layers, non-linearities, and the loss function be defined as follows, where σ is the ReLU nonlinearity applied elementwise: $\sigma(x) = \max(0, x)$. Note that we will follow the convention from PyTorch of defining inputs and outputs as row vectors (rather than column vectors).

$$\begin{aligned} L_1(\mathbf{x}; \mathbf{W}_1, \mathbf{b}_1) = \mathbf{g}_1 = \mathbf{x}\mathbf{W}_1^T + \mathbf{b}_1 & \quad \mathbf{x} \in \mathbb{R}^{1 \times d_{\text{in}}}, \quad \mathbf{W}_1 \in \mathbb{R}^{n \times d_{\text{in}}}, \quad \mathbf{b}_1 \in \mathbb{R}^{1 \times n}, \quad \mathbf{g}_1 \in \mathbb{R}^{1 \times n} \\ \sigma_1(\mathbf{g}_1) = \mathbf{h}_1 & \quad \mathbf{h}_1 \in \mathbb{R}^{1 \times n} \end{aligned} \quad (1)$$

$$\begin{aligned} L_2(\mathbf{h}_1; \mathbf{W}_2, \mathbf{b}_2) = \mathbf{g}_2 = \mathbf{h}_1\mathbf{W}_2^T + \mathbf{b}_2 & \quad \mathbf{W}_2 \in \mathbb{R}^{n \times n}, \quad \mathbf{b}_2 \in \mathbb{R}^{1 \times n}, \quad \mathbf{g}_2 \in \mathbb{R}^{1 \times n} \\ \sigma_2(\mathbf{g}_2) = \mathbf{h}_2 & \quad \mathbf{h}_2 \in \mathbb{R}^{1 \times n} \end{aligned} \quad (2)$$

$$\begin{aligned} L_3(\mathbf{h}_2; \mathbf{W}_3, \mathbf{b}_3) = \hat{\mathbf{y}} = \mathbf{h}_2\mathbf{W}_3^T + \mathbf{b}_3 & \quad \hat{\mathbf{y}} \in \mathbb{R}^{1 \times d_{\text{out}}}, \quad \mathbf{W}_3 \in \mathbb{R}^{d_{\text{out}} \times n}, \quad \mathbf{b}_3 \in \mathbb{R}^{1 \times d_{\text{out}}} \\ \mathcal{L}(\hat{\mathbf{y}}; \mathbf{y}) = \frac{1}{2} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 & \quad \mathbf{y} \in \mathbb{R}^{1 \times d_{\text{out}}} \end{aligned} \quad (3)$$

1. In order to train the network with gradient descent, we need to know the gradient of the loss function with respect to the trainable parameters. Using the chain rule, write the expression for the gradient of the loss

function with respect to each of the trainable parameters of the first linear layer: \mathbf{W}_1 and \mathbf{b}_1 . That is, derive expressions for $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}$. *Hint:* your expressions should involve partial derivatives of \mathcal{L} , $\hat{\mathbf{y}}$, the terms above the arrows in the diagram above, and \mathbf{W}_1 or \mathbf{b}_1 .

2. Write the analytical expressions for each of the partial derivative terms you derived in (1). For example, suppose one of the terms is $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}}$. Then you should find $\frac{\partial}{\partial \hat{\mathbf{y}}} \frac{1}{2} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$ in terms of $\hat{\mathbf{y}}$. Similarly, if one of the terms is $\frac{\partial g_2}{\partial \mathbf{h}_1}$, then you should find $\frac{\partial}{\partial \mathbf{h}_1} (\mathbf{h}_1 \mathbf{W}_2^T + \mathbf{b}_2)$ in terms of \mathbf{h}_1 . For the case of the derivative of a vector by a matrix, it is easiest to provide an expression for the derivative of the i vector element by the j, k matrix element.
3. Write out the gradient descent update rule for \mathbf{W}_1 and \mathbf{b}_1 in terms of the analytical expressions you derived in (2). *Hint:* make sure that the dimensions of your formula make sense! If $\mathbf{W}_1 \in \mathbb{R}^{n \times d_{\text{in}}}$, then the gradient is the same dimensionality: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} \in \mathbb{R}^{n \times d_{\text{in}}}$.
4. *Programming exercise.* Use the code provided in `hw5_task1.py` to implement the fully connected neural network described in this problem, and use automatic differentiation to calculate the derivatives of the network parameters. Specifically, you will complete the following tasks.
 - (a) Complete the code to calculate derivatives for the backward pass for the provided Python functions, `LinearFunction()` and `ReLUFunction()`. Check your answers by running the `check_part_a()`.
 - (b) Check your analytical gradient solutions from part 3 against the gradients using automatic differentiation. Complete the `check_analytical_gradients()` function and make sure the results match.
 - Is it more efficient to compute the gradients analytically (i.e., as in part 3) or to use automatic differentiation? Why?
 - (c) Use gradient descent to train the network for an image inpainting task. Complete and run the `train_network()` function. Include some inpainted outputs of the network in your writeup (you can use the provided code). Don't forget to include include your code from parts (a), (b), and (c).

Task 2 of 3 (20 points)

Neural networks have shown strong performance for the task of image denoising. Typically a network is trained to denoise images with the same noise variance. After training, the network performs well for denoising images at that noise level; however, performance is very poor for images at other noise levels. Interestingly, Mohan et al. (2020)¹ recently showed that in very deep networks, removing the biases from the convolutional and batch normalization layers greatly improves network denoising performance for unseen noise levels. We would like to see if this same trend holds for shallow denoising networks.

Train a small neural network for image denoising and conduct an ablation study to evaluate the effect removing the biases and adjusting the number of hidden features in the network. Specifically, you will train four different network architectures with or without biases and using 32 or 64 hidden features. In each case, train the networks to denoise images with zero-mean gaussian noise and a single noise level: $\sigma = 0.1$.

- Download the BSDS300 image dataset² and extract it so the BSDS300 folder is in the same directory as `hw5_task2.py`.

¹Robust and Interpretable Blind Image Denoising via Bias-Free Convolutional Neural Networks Sreyas Mohan, Zahra Kadkhodaie, Eero P. Simoncelli, Carlos Fernandez-Granda, Proc. ICLR, 2020.

²<https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300-images.tgz>

- After training, fill in the below table by reporting the average PSNR across all images on the validation dataset for the indicated noise levels.
- Include a denoised example image from the validation set for each model at each noise level (use the provided `evaluate_model` function).
- Comment on the results. Does increasing the number of hidden features improve performance? Does removing the biases improve performance on unseen noise levels?
- Include your code with the submission.

To simplify things, we have implemented a shallow denoising network and starter code in `hw5_task2.py`. Report the metrics after 2 epochs of training, and use the default batch size, learning rate, and default settings for any other parameters. Training all the networks takes roughly 10 minutes on a CPU or 2 minutes on a GPU.

| Uses Bias? | Hidden Channels | PSNR (dB) | | |
|------------|-----------------|-----------------|----------------|----------------|
| | | $\sigma = 0.05$ | $\sigma = 0.1$ | $\sigma = 0.2$ |
| ✓ | 32 | ? | ? | ? |
| ✓ | 64 | ? | ? | ? |
| ✗ | 32 | ? | ? | ? |
| ✗ | 64 | ? | ? | ? |

Task 3 of 3 (20 points)

Object or camera motion cause motion blur, usually resulting a poor-looking image. As we've seen previously, images can be deblurred using Wiener deconvolution, which performs division in the frequency domain. While this approach can perform well at removing image blur, it is sensitive to noise. In this task we will evaluate different methods for using neural networks to perform deblurring and denoising of images.

We will assume a simple image formation model:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}, \quad \mathbf{n} \sim \mathcal{N}(0, \sigma\mathbf{I}),$$

where \mathbf{A} models convolution with a known blur kernel, \mathbf{x} is the clean image, and \mathbf{n} is i.i.d. normal with standard deviation σ .

We will evaluate three different methods:

- (Method 1) Wiener deconvolution
- (Method 2) Using a neural network to learn deconvolution and denoising
- (Method 3) A hybrid approach that combines Wiener deconvolution with a network that learns to denoise the output

We have provided two pretrained neural network models which you will evaluate on a validation set of images. For this task, we select a general-purpose network architecture called a U-Net. The first U-Net is trained to deblur and denoise a set of noisy images that have been blurred with the same blur kernel. The second U-Net is trained to denoise the output of Wiener deconvolution. Both networks are trained on noise levels $\sigma \in [0.005, 0.01]$.

1. Evaluate the performance of each approach by computing the average PSNR across the validation set images for three different noise levels: $\sigma \in \{0.005, 0.01, 0.02\}$. Complete the code as indicated in `hw5_task3.py` and

fill in the below table.

| | PSNR (dB) | | |
|----------|------------------|-----------------|-----------------|
| | $\sigma = 0.005$ | $\sigma = 0.01$ | $\sigma = 0.02$ |
| Method 1 | ? | ? | ? |
| Method 2 | ? | ? | ? |
| Method 3 | ? | ? | ? |

2. For what noise levels does Wiener deconvolution demonstrate the best performance? Show a few image outputs for each method and describe qualitatively why it outperforms the other methods.
3. For what noise levels do the neural networks perform best? Based on these experiments, which learned approach performs better? Why do you think this might be the case?

Bonus Task (up to 10 points extra)

While we looked at using neural networks with supervised learning, there are many methods that make use of neural networks without explicit supervision, or even without conventional training at all. One method uses convolutional neural networks as deep image priors³, achieving strong performance for image denoising, super-resolution, inpainting, and other tasks. Implement the deep image prior and report denoising performance on an image from the BSDS dataset. Report PSNR and include the noisy and reconstructed images in your report.

Hint: adapt the code here to load an image and denoise it. <https://github.com/DmitryUlyanov/deep-image-prior/blob/master/denoising.ipynb>

Questions?

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

³https://dmitryulyanov.github.io/deep_image_prior