CIS 419/519: Applied Machine Learning Spring 2023

Homework 5

Handed Out: March 15 *Due: March 29, 7:59 p.m.*

- You are encouraged to format your solutions using LaTeX. You'll find some pointers to resources for learning LaTeXamong the Canvas primers. Handwritten solutions are permitted, but remember that you bear the risk that we may not be able to read your work and grade it properly — do not count on providing post hoc explanations for illegible work. You will submit your solution manuscript for written HW4 as a single PDF file.
- The homework is **due at 7:59 PM** on the due date. We will be using Gradescope for collecting the homework assignments. Please submit your solution manuscript as a PDF file via Gradescope. Post on Piazza and contact the TAs if you are having technical difficulties in submitting the assignment.

1 Written Questions

Note: You do not need to show work for multiple choice questions. If formatting your answer in LaTeX, use our LaTeX template `hw5_template.tex` (This is a read-only link. You'll need to make a copy before you can edit. Make sure you make only private copies.).

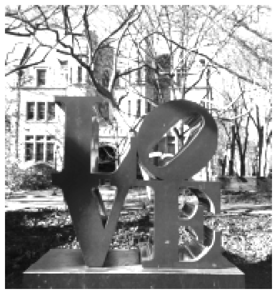
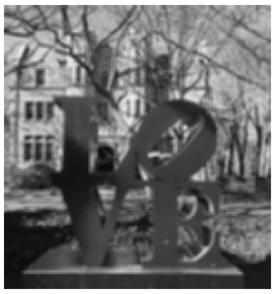
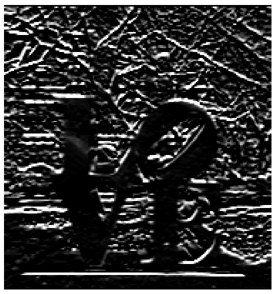
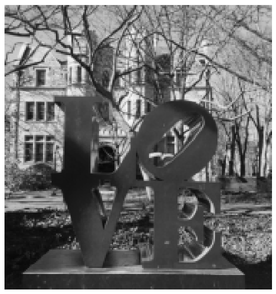
1. [Image Filtering/Convolution] (12 pts) We discussed the use of convolution filters for images briefly in class, mostly in the context of CNN. However, before CNNs became popular, convolution filters had already been an essential part of signal processing and computational photography. You will probably be surprised by how many features in Photoshop or Lightroom can be easily implemented with the correct choice of convolution filter(s). In this question, we will take a look at a few common types of convolution filters for images, and visualize how they would transform the original image. Let's take the following image for example. This is a gray-scale image, where each pixel can be represented by a value between $[0, 1]$, where 0 is black and 1 is white. The gray-scale image itself can be represented by a matrix of shape $width * height$, and we are going to apply 3*3 convolution filters to the matrix. Assume the bias parameter is set to 0 for all these convolution filters.

For the following sub-questions, you will be given a convolution filter, and a few transformed image. Your task is to pick the one that corresponds to the given filter. Your TAs have created a skeleton colab notebook, where you can implement + test out these filters with different images. https://colab.research.google.com/drive/1_mNak1RCxehWBn8Le3LGGHCfbwA2xUTg?usp=sharing.

(a) [4 pts] Consider the following filter:

$$X = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

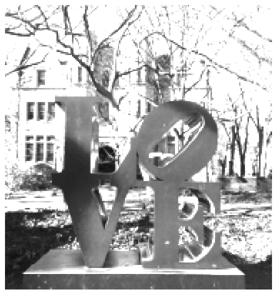
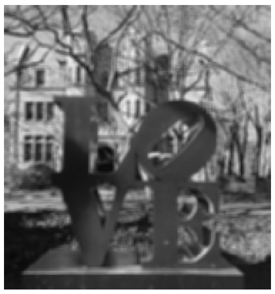
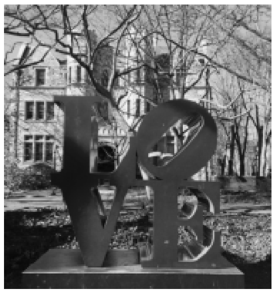
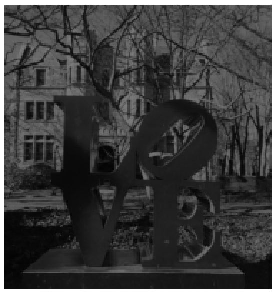
If we apply X as a convolution filter to the original image, which of the following transformed image will we see? **Briefly justify your choice using one or two sentences.**

- A. 
- B. 
- C. 
- D. 

(b) [4 pts] Now consider this following filter:

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

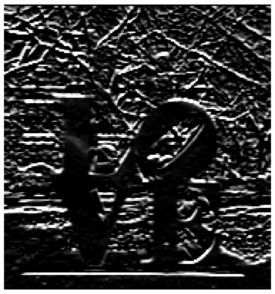
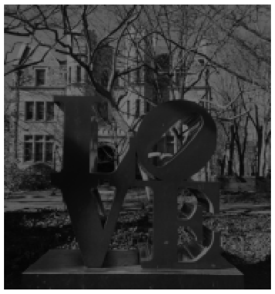
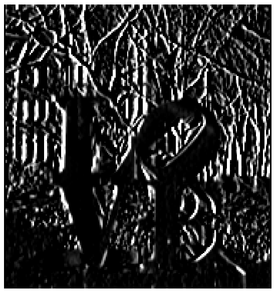
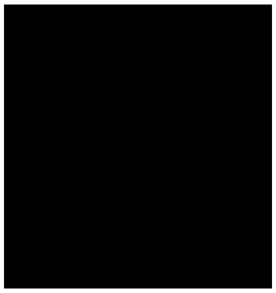
If we apply X as a convolution filter to the original image, which of the following transformed image will we see? **Briefly justify your choice using one or two sentences.**

- A. 
- B. 
- C. 
- D. 

(c) [4 pts] Now let's look at a more challenging example:

$$X = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

If we apply X as a convolution filter to the original image, which of the following transformed image will we see? **Briefly justify your choice using one or two sentences.**

- A. 
- B. 
- C. 
- D. 

2. [Neural Networks] (5 pts) You are performing gradient descent to train a model with 2 learnable parameters w_1 and w_2. Suppose the sequence of gradients over $k = 3$ iterations is (oldest first):

$$dL/dw_1 = [1, 0, 1]$$
$$dL/dw_2 = [2, 6, 3]$$

At the beginning of the k^{th} iteration, the values of w_1 and w_2 are both set to 1. What should the updated value of w_1 be after the k^{th} iteration, if the learning rule being used for gradient descent is momentum gradient descent, defined below.

Momentum Gradient Descent : Set initial momentum ρ to zero before iteration number 1. Set $\alpha = 0.1, \mu = 0.9$. The momentum rule, discussed in class is:

$$\rho = \mu * \rho - \alpha \nabla_{w|w=w_t}$$

$$w_{t+1} = w_t + \rho$$

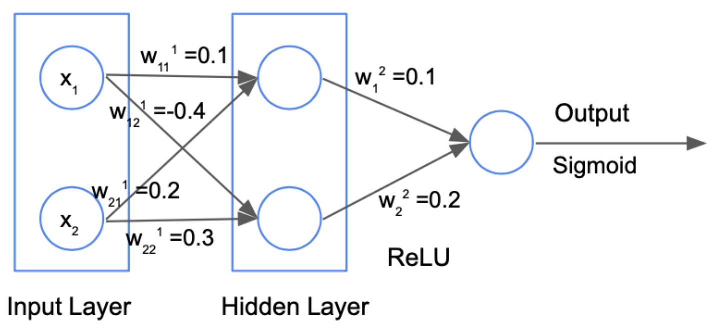
3. [CNNs] (8 pts) Consider the following 3-layer neural network, with layers enumerated starting from the input.

   (a) **Conv2d**: In channels: 3, Out channels: 5, kernel: 5x5, Stride 1, Padding 0
   (b) **ReLU**
   (c) **MaxPool2d**: kernel: 2x2, Stride 2, Padding 0
   (d) **Conv2d**: In channels: 5, Out channels: 10, kernel: 3x3, Stride 1, Padding 0
   (e) **ReLU**
   (f) **MaxPool2d**: kernel: 2x2, Stride 2, Padding 0
   (g) **Conv2d**: In channels: 10, Out channels: 20, kernel: 3x3, Stride 1, Padding 0
   (h) **ReLU**
   (i) **MaxPool2d**: kernel: 2x2, Stride 2, Padding 0

   If the input image is of size 232 (dimension 1) x 232 (dimension 2) x 3 (dimension 3), compute the following:

   (a) Output size dimension 1
   (b) Output size dimension 2
   (c) Output size dimension 3
   (d) Number of learnable parameters

4. [Backpropagation] (15 pts)



Consider the two-layer neural network shown above. You will use the **ReLU** function as the activation function to compute activations of the *hidden layer*, and the **Sigmoid** activation function on the *output*. The input vector is $\mathbf{x} = [10 \quad 8]^{\mathsf{T}}$. All weights are displayed on the image (the superscript denotes the layer information).

Suppose that the loss function is $\mathcal{L}(\mathbf{output}) = \mathbf{output}$. Compute the gradient of this loss with respect to each of the weights. Be sure to show the details of your computational work.
As a reminder, the ReLU function and the derivative of the ReLU function are as follows:

$$ReLU(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\nabla ReLU(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

5. *(5190 mandatory, 4190 optional)* [Neural Network Design/Regularization] (9 pts) In reality, when you are developing a neural network for a machine learning problem, often times you would need to design a learning objective or network architecture that is customized towards the problem. A typical challenge in these cases is to make the forward computation differentiable. An example problem of such would be one that involves first-order logic. e.g. How can we enforce logical constraints to the output of your neural network model?

   Suppose you are designing a multi-label (i.e. The model outputs multiple labels for a single input example) image classification model for animal taxonomy. Among the labels, there are {`hamster`, `cat`, `mammal`}. Ideally, the model should output `mammal` whenever it outputs `hamster` or `cat`, but it should never output `hamster` and `cat` together.

   While conceptually neural networks are able to express such logical constraints, in reality it usually requires many, many training examples before it can learn to do so consistently. When collecting such data becomes expensive, an alternative strategy is to formulate such constraints (as a form of "knowledge") as a regularization term to the learning objective of the model. In this question, we will look at one specific way of doing it using t-norm fuzzy logic. Let's assume that there are three (activated) output neurons C_h, C_c, C_m for the labels {`hamster`, `cat`, `mammal`} respectively, where having a value closer to 1 indicates positive, and closer to 0 indicates negative. Then the aforementioned logical constraint that `hamster` and `cat` should never appear together can be expressed by $C_h \wedge C_c = 0$ (i) , and the `mammal` constraints can be expressed as $C_h \wedge (\neg\, C_m) = 0$ (ii) and $C_c \wedge (\neg\, C_m) = 0$ (iii).

   As none of the constraints can be directly expressed as differentiable loss functions/terms, your goal here is to express the constraints using Łukasiewicz t-norm, a fuzzy, probabilistic way of expressing first-order logic.

$$\begin{aligned} A \wedge B &= max(0, A + B - 1) \\ A \vee B &= min(1, A + B) \\ \neg A &= 1 - A \end{aligned}$$

   (a) [3 pts] Express the three constraints (i), (ii) and (iii) with Łukasiewicz t-norm defined above.

   (b) [2 pts] The next step would be expressing the three constraints as three regularization terms to the loss function. Before we do that, we need to think whether we need an activation functions for the three constraints expressed in t-norm. Why or why not? Briefly justify your answer.

   (c) [4 pts] Now it's time to design the loss functions for three t-norm constraints (as regularization terms). Keep in mind the basic criteria for loss function: loss should be high for mistakes, and low for correct outputs.
   (*Hint: Think of the t-norm constraints as a 0-1 classification or regression task, where you want the output of the t-norm constraints to match the ground truth "labels" in (i), (ii), (iii) respectively.*)