Week 8 Lab

Bookmark this page

Week 8 Lab

(1.0 points possible)

Filters

For this lab:

- You will need to understand the material in the notes on convolutional neural networks.
- We suggest that you write down your answers/explanations
- It will be useful to sketch the networks, input/outputs, and convolution operations to answer the questions in this Lab.

1) Filters

Assume that the input to a CNN is a one-dimensional (1D) image, that is, a vector of values corresponding to light intensity, A common operation that we want to perform on images is to detect where there are "edges" (i.e., rapid changes in the intensity) since these sometimes correlate with changes in material, lighting, depth, etc. The early layers of the visual systems of animals appear to do processing of this type.

Let's start simple and imagine that the input "image" is a vector of three elements, $[x_1, x_2, x_3]$. We want to build a network that has a high output when there is a rapid change of intensity in the image and low otherwise. What this will mean is that we will call x_2 an "edge" location iff x_1 is high and x_3 is low or vice versa. Another way of saying this is that we want to detect locations where $|x_3 - x_1| > 1$. (In general, the choice of 1 is arbitrary and it should be a learned bias. **But here, we will use 1 as the threshold for all of the following problems.**)

Let's consider building an "edge classifier" using a single unit, with three inputs and ReLU activation, whose output should be greater than 0 for an edge location defined as above and 0 otherwise. The weights here are either $-1, 0,$ or 1.

1A. Pick a set of weights and bias for this unit that can do this task (or determine if no such set of weights exist). Be prepared to explain your answer.

Enter a list of four numbers for the unit weights and bias $[w_1, w_2, w_3, b]$ or the string 'None' if no such weights exist: 'None'

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

1B. Now, let's consider a network with two units (each with three inputs) on the first layer and one unit (with two inputs) on the output layer. All units have ReLU activation. Pick a set of weights and biases for these units that can do the edge detection task or determine if no such set of weights exist. Be prepared to explain your answer.

Enter a list of three lists of weights and bias, first for the two units on the first layer (in the order of $[w_1, w_2, w_3, b]$) and then for the output unit (in the order of $[w_1, w_2, b]$), or 'None' in no such weights and biases exist.

[[-1, 0, 1, -1], [1, 0, -1, -1], [1, 1, 0]]

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

1C. Now, let's think about 3x3 "images" and try to detect diagonal "edges." Consider a 3x3 image patch

$$\begin{matrix} x_1, x_2, x_3 \\ x_4, x_5, x_6 \\ x_7, x_8, x_9 \end{matrix}$$

We might care about patterns that have:

- $|(x_1 + x_2 + x_4) - (x_9 + x_8 + x_6)| > 1$ (diagonal edge x_3-x_5-x_7), or
- $|(x_3 + x_2 + x_6) - (x_7 + x_8 + x_4)| > 1$ (diagonal edge x_1-x_5-x_9)

We can do this with four ReLU units in the first layer and one output unit. Pick a set of nine weights (and a bias) for **just one** of the first layer units, that responds to one of the four patterns.

Enter a list of 10 numbers: 9 weights (w_i) and a bias. [1, 1, 0, 1, 0, -1, 0, -1, -1, -1]

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

2) Convolution

The weights for these units that we have constructed are sometimes called a "filter kernel." It's a (usually) small vector (for 1D images) or matrix (for 2D images) that is "tuned" to produce a high value for certain kinds of patterns/features (such as edges). Note that you can think of these filters allowing a **limited receptive field** which only focus on part of the image instead of the whole image. In order to detect the patterns/features anywhere in the (large) input image, we **slide** the filters over the image and produce a new feature map. This process is known as **convolving the filter with the image** and this is where CNNs get their name [*see footnote 1*].

Assume that we have a 1D input x of size 15 (note that the x index starts from 1 here):

$$x = [x_1, x_2, \cdots, x_{15}] = [0, 0, 0, 2, 2, 2, 4, 4, 4, 2, 2, 2, 0, 0, 0]$$

Consider we have a filter $f = [w_1, w_2, w_3]$ of size 3. Now we want to convolve this filter f with the 1D input x to obtain a 1D feature map $\phi(x)$ of the same size 15. When we perform convolution at input location i, the feature map at location i is $\phi_i = w_1 \cdot x_{i-1} + w_2 \cdot x_i + w_3 \cdot x_{i+1}$. We then send the feature maps to non-linear activation functions $\sigma(\cdot)$ which might, for example, be ReLU units.

Note that we will assume zero padding, such that the input value x_i is 0 if some part of the filter falls outside the image, i.e., $x_i = 0, \forall i \notin \{1, 2, \cdots, 15\}$. Also, we slide the filter over the input by the stride, which is 1 here.

2A. Visualize the input in terms of gray-scale plot. (You are free to make assumptions about your gray-scale mapping, etc.)

2B. You are given a ReLU unit that detects $x_3 - x_1 > 1$. Enter a list of indices i where the output of this ReLU unit is positive, i.e., $\sigma(\phi_i) > 0$:

Enter a list of indices in ascending order: [3, 4, 6, 7]

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

2C. You are given a ReLU unit that detects $x_1 - x_3 > 1$. Enter a list of indices where the output of this ReLU unit is positive, i.e., $\sigma(\phi_i) > 0$:

Enter a list of indices in ascending order: [9, 10, 12, 13]

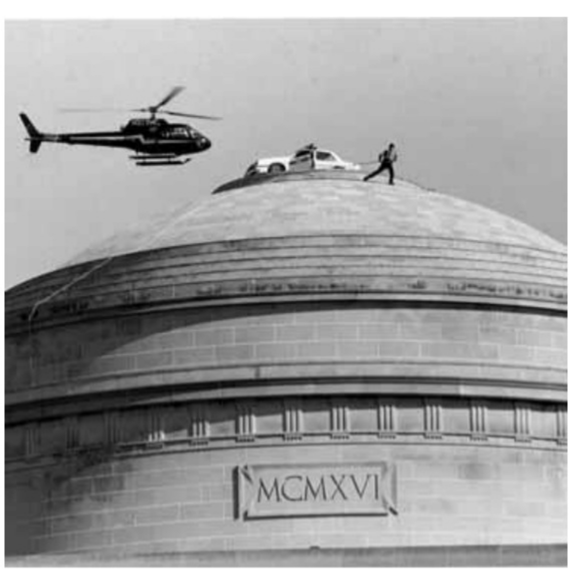
Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

Note that in the above processing, "step" patterns are detected anywhere in the image and independent of the overall "brightness." These are important properties for image processing.

3) Experiment with convolution

Executing the code below will convolve one of the filters f_1, f_2, f_3 (including the bias) with a familiar image and return an image that is white where the output is greater than 0, and black elsewhere. Thus, the image returned and shown in the detailed output is binary. Here's the original image:



3A. Try changing the filters and the bias to get a feeling for what is going on when a CNN learns the weights and biases for the filters. The range of values in the images after filtering is on the order of ± 100; that should help you pick the bias value. Click "Show/Hide Detailed Results" to see the output image.

```python
# Some interesting filters to try:
f1 = ((1./8)*np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])).tolist()
f2 = ((1./8)*np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])).tolist()
# f3 = ((1./10)*np.array([[-1, -1, -1], [-1, 10, -1], [-1, -1, -1]])).tolist()
f3 = ((1./10)*np.array([[-1, -1, -1], [-1, 30, -1], [-1, -1, -1]])).tolist()
# You will need to change the bias to get nice results... think about the sign.
def run():
    # return filter(weights=f3, bias=15)
    return filter(weights=f3, bias=1)
    # return filter(weights=f3, bias=5)

```

Run Code | Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

Your score on your most recent submission was: 100.00%

Test Results:

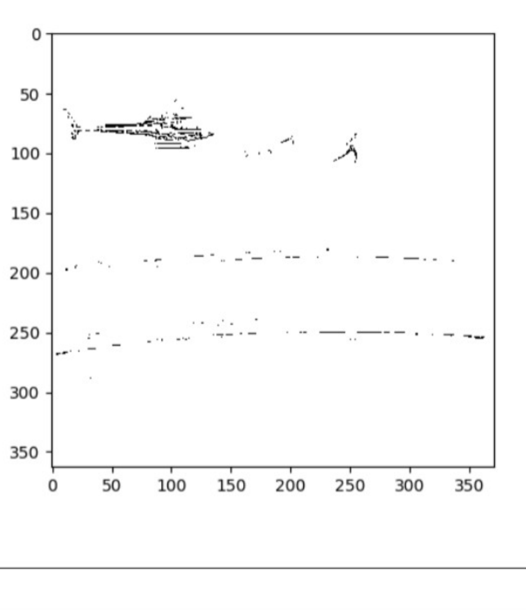
Test 01

The test case was:

```
#Your Code Here
ans=run()
```

Our solution produced the following value for `ans`:

Your submission produced the following value for `ans`:



3B. Consider these "features":

A) Points brighter than neighborhood
B) Vertical edges (right brighter)
C) Horizontal edges (top brighter)

Which filter from the code above best detects each feature?

Enter a python list of numbers (1, 2, 3 corresponding to the filters f1, f2, f3) for features A, B, C:

[3, 1, 2]

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

3C. How is the output image affected if you increase or decrease the bias value? Explain your observations.

3D. Keeping the bias constant, change the coefficients of each filter array and consider how they affect the output image. What happens when the bias is zero? What happens when the bias is non-zero?

3E. Given a 1D black and white "image" (think of this as a sequence of 0's and 1's), we define an object to be a consecutive sequence of black pixels (0's) (e.g., 0101 has two objects, 10110011000 has three objects, etc.). Design a simple CNN that can count the number of objects in a 1D image. Specify your choice of filter. How many hidden layers and filters do you need? What is your choice of activation functions and why? Hint: you can use a fully connected layer at the last layer.

4) Conversion from Convolutional Layer to Fully Connected Layer

Consider a 1D input x of size 4, a 1D feature map $\phi(x)$ of the same size 4, and a filter f of size 3: $(w_1, w_2, w_3) = (5, 6, 7)$ with stride of 1. Assume that when doing convolutions, inputs that fall outside the image indices are treated as zeros, as in Problem 2. The effect of convolving a filter with the input layer x is actually equivalent to constructing a matrix M of weights between two layers of a fully connected network, such as we saw in HW 6, but where the same weight may occur more than once in the matrix, i.e.,

$$\phi(x) = M^\top x ,$$

where M is a 4 by 4 matrix.

4A. What is the convolution-equivalent matrix M? Hint: What is the feature map $\phi(x)$?

Enter the matrix M as a list of rows, where each row is a list of the matrix elements in that row (i.e., in the same format you would use to construct a 2D numpy array):

[[6,5,0,0], [7,6,5,0], [0,7,6,5], [0,0,7,6]]

Submit | View Answer | Ask for Help | 100.00%

You have infinitely many submissions remaining.

By looking at the matrix you can see the "sliding" of the filter during the convolution. In a fully-connected network every entry in the matrix is a different weight value that has to be learned; in contrast, in the convolutional case above you can see that each weight appears several times in the matrix (for essentially every output unit) and many weights are zero. The "re-use" of weights is referred to as "parameter sharing" or "parameter tying." This network can be trained with error back-propagation as we already understand it; care must just be taken to sum over all "error values" that a particular weight can contribute to.

4B. In general, we have a 1D input x of size d, a 1D filter f with size k and with stride of s, and 1D zero-paddings on both sides with size p. In the above problem 4A, we have $d = 4, k = 3, s = 1, p = 1$. Discuss what the size of feature map $\phi(x)$ is, in terms of general d, k, s, and p.

Footnotes

[Footnote at Problem 2] If you were already familiar with what a convolution is, you might have noticed that the definition given in the lab corresponds to a correlation and not to a convolution. Indeed, correlation and convolution refer to different operations in signal processing. However, in Neural Networks literature, most libraries implement the correlation (as described in this lab) but call it convolution. The distinction is not significant; in principle, if convolution is called for the network would learn the same weights, just flipped. For a discussion on the difference between convolution and correlation and the conventions used in the literature you can read section 9.1 (pages 327-329) from the deep learning book.