New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Written Homework 4.
Due Dec 13, 2024, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually.

Problem 1: Convolution for Shifted Images (15pts)

On the last problem set we considered kernel functions for a cartoonish problem of classifying images of digits under shifts. Here we will think about how to do the same thing with a convolutional feature extractor.

Again consider the following images of 0s and 1s. Here we take the convention that white pixels have $+1$ values and black pixels have -1 values.

$$I_1 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & +1 & +1 & +1 & -1 \\ -1 & +1 & -1 & +1 & -1 \\ -1 & +1 & +1 & +1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \quad I_2 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \quad I_3 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$I_4 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \quad I_5 = \begin{bmatrix} -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \quad I_6 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 \end{bmatrix}$$

$$I_7 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 & -1 \end{bmatrix} \quad I_8 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & +1 & +1 \\ -1 & -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 & +1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Your goal is to design a convolutional feature transformation $C(I)$ that would allow $I_3, I_4, I_5, I_6, I_7, I_8$ to all be correctly classified using a simple nearest neighbor classifier trained with I_1, I_2. Specifically, we should have that $\|C(I_i) - C(I_1)\|_2 < \|C(I_i) - C(I_2)\|_2$ for $i \in \{3, 6, 8\}$ and that $\|C(I_i) - C(I_1)\|_2 > \|C(I_i) - C(I_2)\|_2$ for $i \in \{4, 5, 7\}$. Design such a C with the following two-layer form:

$$C(I) = g(I \circledast W_1) \circledast W_2,$$

where W_1 and W_2 are 2×2 convolutional filters and g is any entrywise non-linearity. You can select W_1, W_2, and g to be anything you like.

To present your solution, specify your choice of W_1, W_2, and g, and attach code that implements C and shows that you can correctly classify the images using this transformation. Explain in a few sentences how you came to your choices for W_1, W_2, and g.

Note: There are *many* possible valid transformations, so please come up with your own solution and don't ask a classmate to see their approach. I expect a lot of diversity in the solutions students find. If you aren't able to find a transformation, explain in words what approach you tried, and show your best result.

There are a lot of ways to solve this problem. My approach was based on the observation that zeros will have more horizontal edge response than ones. So I set W_1 to be a horizontal edge detection filter: $W_1 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$. I then set g to be a ReLU non-linearity (a lot of other things work as well) and set W_2 to be a simple blur filter $W_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. Any image of a zero passed through this feature extractor resulted in a final image with relatively large pixel values, while passing an image of a one through resulted in an image with values near zero. So the simple 1-nn classifier gets all the images correct.

Problem 2: The necessity of non-linearities in CNNs (15pts)

Consider a Convolutional Neural Network that processes 1 dimensional vector data (e.g. vectors of length d). Typically such a network has convolutional layers that convolve the input with a set of filters, and then apply an entrywise non-linearity to the result. This is often followed by another convolutional layer.

What if you didn't include the non-linearity? Would the network still be as powerful? In this problem you will prove that the answer is no. In particular, let $W_1 \in \mathbb{R}^\ell$ and $W_2 \in \mathbb{R}^k$ be convolutional filters. Prove that there is always another convolutional filter $\bar{W}$ such that:

$$(x \circledast W_1) \circledast W_2 = x \circledast \bar{W}$$

In other words, if there was no non-linearity, two adjacent convolutional layers could be replaced by a single convolutional layer that uses different filters!

This is a hard problem. I would suggest trying to "guess" the correct value of $\bar{W}$. You can probably figure out its length. If you convolve a length n vector by a length b vector, then you should get out a vector with length $n-b+1$. So if you convolve x with W_1 followed by W_2, you should get a length $(n-\ell+1)-k+1$ vector. This should immediately tell you that $\bar{W}$ has length $\ell + k - 1$.

You might also be able to guess the actually values in $\bar{W}$. One hint would be as follows: what happens if you convolve a filter W with the vector:

$$x = \big[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0\big]?$$

If you pad with enough 0's, you should get that the output looks exactly like W padded with zeros. So one way to figure out W is to imagine what you get if you convolve the vector above with W_1 first, then with W_2.

Using the definition of discrete convolution, we can write:

$$\begin{aligned}
[(x \circledast W_1) \circledast W_2]_i &= \sum_{j=1}^{\ell} (x \circledast W_1)[j+i-1] W_2[j] \\
&= \sum_{j=1}^{\ell} \sum_{z=1}^{k} x[z + (j+i-1) - 1] W_1[z] W_2[j] \\
&= \sum_{j=1}^{\ell} \sum_{z=1}^{k} x[z+j+i-2] W_1[z] W_2[j]
\end{aligned}$$

Let $\bar{W}$ be a length $\ell + k - 1$ filter with

$$\bar{W}[m] = \sum_{\substack{j \in 1,\ldots,\ell \\ z \in 1,\ldots,k \\ j+z-1=m}} W_1[z] W_2[j]$$

. Then we have that:

$$\begin{aligned}
\left[x \circledast \bar{W}\right]_i &= \sum_{m=1}^{\ell+k-1} x[m+i-1]\bar{W}[m] \\
&= \sum_{m=1}^{\ell+k-1} x[m+i-1] \sum_{\substack{j\in 1,\ldots,\ell \\ z\in 1,\ldots,k \\ j+z-1=m}} W_1[z]W_2[j] \\
&= \sum_{m=1}^{\ell+k-1} \sum_{\substack{j\in 1,\ldots,\ell \\ z\in 1,\ldots,k \\ j+z-1=m}} x[m+i-1]W_1[z]W_2[j] \\
&= \sum_{m=1}^{\ell+k-1} \sum_{\substack{j\in 1,\ldots,\ell \\ z\in 1,\ldots,k \\ j+z-1=m}} x[j+z+i-1]W_1[z]W_2[j]
\end{aligned}$$

This sum is exactly equivalent to $\sum_{j=1}^{\ell}\sum_{z=1}^{k} x[z+j+i-2]W_1[z]W_2[j]$: both have the same term inside the sum, and you can check that the summation just ranges over every value of j from 1 to ℓ and of z from 1 to k. So we conclude that $[(x \circledast W_1) \circledast W_2]_i = \left[x \circledast \bar{W}\right]_i$.

Problem 3: Triangulating Points via Principal Component Analysis (20pts)

(15 pts) Consider a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ arranged as rows in an $n \times d$ matrix $\mathbf{X}$. Assume $d \leq n$. As discussed in class, PCA can be used to find a set of shorter "code vectors" $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$ that are useful for data visualization. Specifically, if we choose these code vectors to be the loading vectors of PCA applied to $\mathbf{X}$ we should have that for each i, j:

$$\langle \mathbf{z}_i, \mathbf{z}_j \rangle \approx \langle \mathbf{x}_i, \mathbf{x}_j \rangle \qquad \text{and} \qquad \|\mathbf{z}_i - \mathbf{z}_j\|_2 \approx \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

What if we don't have access to $\mathbf{X}$ itself, but only to the *distances* between each pair of data points? I.e. you are given symmetric $n \times n$ matrix $\mathbf{D}$ with $\mathbf{D}_{i,j} = \mathbf{D}_{j,i} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$.

(a) From the information in $\mathbf{D}$ show how to recover $\mathbf{X}\mathbf{X}^T$ for a data set whose distances match those in $\mathbf{D}$. **Hint:** Since distances are only unique up to rotation and translation, you can assume that one of the points is centered at the origin. E.g. that $\mathbf{x}_1 = \mathbf{0}$. The i, j entry of $\mathbf{D}$ is equal to $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. If we assume that $\mathbf{x}_1 = \mathbf{0}$ as suggested, then $\|\mathbf{x}_i\|_2^2$ and $\|\mathbf{x}_j\|_2^2$ are exactly equal to the distances between $\mathbf{x}_i$ and $\mathbf{x}_j$ and $\mathbf{x}_1$, so they can be found using the first row of the matrix. Overall, we can compute $(\mathbf{X}\mathbf{X}^T)_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = -\frac{1}{2}\left(\mathbf{D}_{ij} - \mathbf{D}_{1i} - \mathbf{D}_{1j}\right)$.

(b) Explain how to use the SVD of $\mathbf{X}\mathbf{X}^T$ to recover the rank k loading vectors of $\mathbf{X}$, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$. Let $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. As discuss in class, the loading vectors are equal to $\mathbf{U}_k\mathbf{\Sigma}_k$: the first columns of $\mathbf{U}$ multiplied by the first $k \times k$ block of $\mathbf{\Sigma}$. Note that $\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{U}^T$. Here we are using that $\mathbf{V}$ is orthogonal so $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. $\mathbf{\Sigma}\mathbf{\Sigma}$ is itself a diagonal matrix with all entries equal to the squares of those in $\mathbf{\Sigma}$. So this is actually and SVD for $\mathbf{X}\mathbf{X}^T$. So one option is to compute the SVD of $\mathbf{X}\mathbf{X}^T$, set $\mathbf{U}$ equal to its left singular vectors, and set $\mathbf{\Sigma}$ to be a diagonal matrix containing the squareroots of its singular values.

Alternatively, the students are free to appeal directly to Slide 11 here. Which says that they can get $\mathbf{U}$ and $\mathbf{\Sigma}$ by computing and eigendecomposition of $\mathbf{X}\mathbf{X}^T$.

(c) When we set $k = d$, prove that for all i, j we will obtain vectors that *exactly* capture the geometry of $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$. I.e. that:

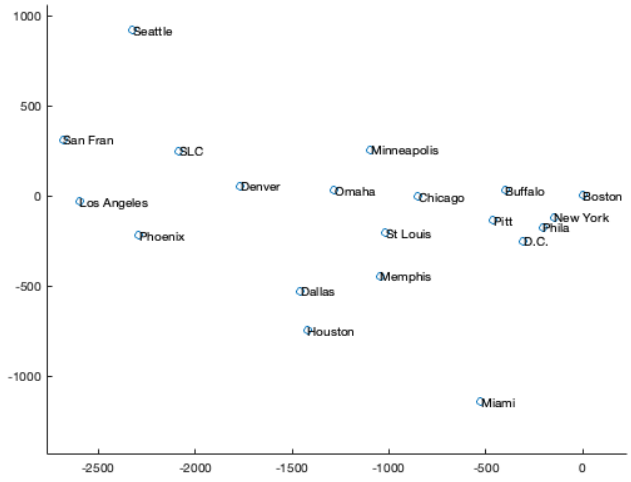
$$\|\mathbf{z}_i - \mathbf{z}_j\|_2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

From class have that

$$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2 = \|\mathbf{x}_i^T\mathbf{V}^T - \mathbf{x}_j^T\mathbf{V}^T\|_2^2 = \|\mathbf{V}(\mathbf{x}_i - \mathbf{x}_j)\|_2^2 = (\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{V}^T\mathbf{V}(\mathbf{x}_i - \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2.$$

In the second to last step we used that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ This is a really useful property – it means that if you have pairwise distances between points that you know come from a low-dimensional space, you can back out that arrangement using PCA. This problem is related to that of using triangulation to locate points based on distances.

(d) Implement your method from part (a) and (b) and run it on the U.S. cities dataset provided in `UScities.txt` for rank $k = d = 2$. This data set contains unsquared Euclidean distances between 20 cities, so you need to square the distances to obtain $\mathbf{D}$. Plot your estimated city locations $\mathbf{z}_1, \ldots, \mathbf{z}_n$ on a 2D plot and label the cities to make it clear how the plot is oriented. Submit these images and your code with the problem set (attached to the same PDF). I'm expecting a result similar to the one below (note that the locations look correct up to rotation).



Chris's code can be found at the end of this PDF. It's in MATLAB, but the implementation in numpy would be essentially identical.

(e) A cool property of this technique is that it's very robust to inaccuracies in the distance matrix $\mathbf{D}$. Rerun your code, but where you first perturb the distances in $\mathbf{D}$. Specifically, replace both $\mathbf{D}_{i,j}$ and $\mathbf{D}_{j,i}$ with a random number between $(1-r)\mathbf{D}_{i,j}$ and $(1+r)\mathbf{D}_{i,j}$ (i.e., keep the matrix symmetric). Plot the city locations recovered for a few different values of r. How high can you set the value and still get a good approximation to the city locations? This last answer doesn't need to be a quantitative.

```
UScities = csvread('UScities.csv')
D = UScities.^2;


for i = 1:20
   for j = i:20
      if(rand()<.1)
         D(i,j) = D(i,j) + .5*(rand()-.5)*D(i,j)
         D(j,i) = D(i,j);
      end
   end
end

norms = D(1,:);
F = -.5*(D - ones(n,1)*norms - norms'*ones(1,n));
[U,S] = svds(F,2)
X = U*sqrt(S);

scatter(-X(:,1),X(:,2))
L = {'Boston','Buffalo','Chicago','Dallas','Denver','Houston','Los
Angeles','Memphis','Miami','Minneapolis', 'New
York','Omaha','Phila','Phoenix','Pitt','St Louis', 'SLC', 'San Fran','Seattle','D.C.'};
dx = 0.1; dy = 0.1;
text(-X(:,1)+dx, X(:,2)+dy, L);
% xlim([-2000,2000])
% ylim([-2000,2000])

X = rand(n,2)
for i = 1:n
   for j = 1:n
      D(i,j) = norm(X(i,:) - X(j,:))^2;
   end
end
```