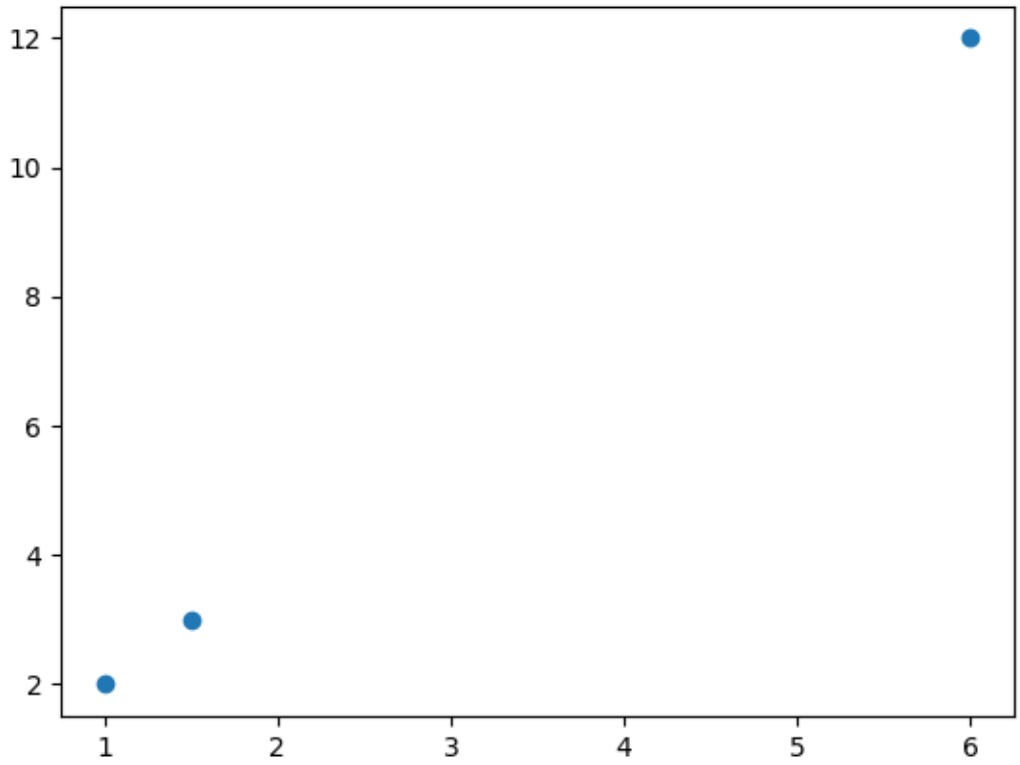
Section 09: PCA, SVD, Convolutional Neural Networks

1. Principal Component

Consider the following dataset, which is represented as three points in $\mathbb{R}^2$. Note that in this problem we will **not** demean the dataset.

$$\begin{bmatrix} 1 & 2 \\ 1.5 & 3 \\ 6 & 12 \end{bmatrix}$$



(a) What is the first principal component vector, v_1?

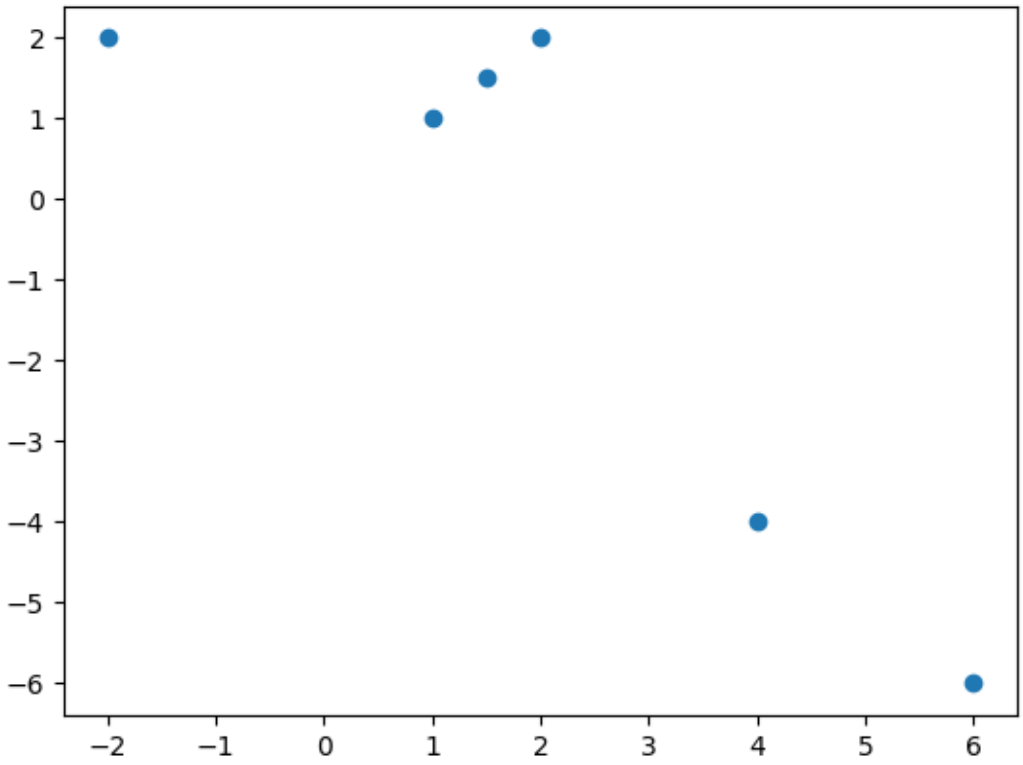
(b) What is the second principal component, v_2?

(c) If we use only the first principal component to compress the dataset, what will the representation of each point be?

(d) Will this representation be lossy, or perfectly preserve the dataset?

Answer the same questions for the following, slightly larger dataset:

$$\begin{bmatrix} 1 & 1 \\ 1.5 & 1.5 \\ -2 & 2 \\ 4 & -4 \\ 6 & -6 \\ 2 & 2 \end{bmatrix}$$



(a) What is the first principal component vector, v_1?

(b) What is the second principal component, v_2?

(c) If we use only the first principal component to compress the dataset, what will the representation of each point be?

(d) Will this representation be lossy, or perfectly preserve the data?

2. Using the Eigenbasis

It's a very useful fact that for any symmetric n×n matrix A you can find a set of eigenvectors u_1, ..., u_n for A such that:

- $\|u_i\|_2 = 1$
- $u_i^T u_j = 0, \forall i \neq j$
- $u_1, \ldots, u_n$ form a basis of $\mathbb{R}^n$

One of the reasons this fact is useful is that facts about these matrices are easier to prove if you think about the vectors in terms of their "eigenbasis" components, instead of their components in the standard basis. As a trivial example, we'll show that you can calculate Ax for a vector x without having to do the matrix multiplication.

(a) Consider the matrix $A = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}$. Verify that $u_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ and $u_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ are eigenvectors and meet the definitions. Find the eigenvalues associated with u_1 and u_2

(b) since $\{u_1, u_2\}$ are a basis, we can write any vector as a linear combination of them. Write $x = \begin{bmatrix} -1/\sqrt{2} \\ 3/\sqrt{2} \end{bmatrix}$ in this basis.

(c) Based on the eigenvectors and eigenvalues your found in part a, diagonalize A, i.e, find matrix U and D such that $UDU^T = A$ where all entries of D are 0 except for the ones on the diagonal.

(d) Use the decomposition and the eigenvalues you calculated in the previous parts to calculate Ax without doing matrix-vector multiplication.

This method of calculating a matrix vector product won't actually be more computationally efficient – but it's what's "really" happening when you do the multiplication, so this will be useful intuition under certain circumstances. Expressing vectors in an eigenbasis is also a useful proof technique, as we'll see in some later problems.

3. Singular Value Decomposition - Proofs

Recall that if we have a symmetric, square matrix A $\in \mathbb{R}^{n \times n}$, we can eigen-decompose it in the form of $A = USU^T$, where the columns of U are eigenvectors of A with lengths of 1, and the diagonal of S is the list of eigenvalues corresponding to those eigenvectors.
Now, for a more general case, where A is a data matrix with the dimension of $\mathbb{R}^{n \times d}$, there is still a way to decompose it: $A = USV^T$, where $U \in \mathbb{R}^{n \times n}$, S is a rectangular diagonal matrix and $S \in \mathbb{R}^{n \times d}$, and $V \in \mathbb{R}^{d \times d}$.
It is called Singular Value Decomposition (SVD).

(a) Let A have SVD USV^T. Show AA^T has the columns of U as eigenvectors with associated eigenvalues S^2.

(b) Let A have SVD USV^T. Show A^TA has the columns of V as eigenvectors with associated eigenvalues S^2.

(c) For the matrix A, suppose we are given that $AA^T = US^2U^T$ and $A^TA = VS^2V^T$. Show that $A = USV^T$. I.e., show that for any vector $x \in \mathbb{R}^d$, we have $Ax = USV^Tx$

4. Convolutional Neural Networks

(a) Discuss the advantages of a convolutional layer compared to a fully connected one.

(b) Discuss the advantages of maxpooling in CNN.

5. Shapes in Convolutional Neural Networks

When designing a convolutional neural network, it's important to think about the shape of the data flowing through the network. In this problem you will get gain experience with thinking about the shapes in a neural network and a better intuition for why convolutional neural networks require so few parameters compared to fully connected layers.

Shape of a convolutional layer / maxpooling output: For a $n \times n$ input, $f \times f$ filter, padding p and stride s, the output size is $o \times o$ where:

$$o = \frac{n - f + 2p}{s} + 1$$

We will use Pytorch Conv2d to represent a 2D convolution, and Pytorch MaxPool2d to represent a 2D max pooling. Take a look at the documentation on Conv2d and MaxPool2d.

(a) Assume your input is a batch of N 64×64 RGB images. The input tensor your neural network receives will have shape $(N, 3, 64, 64)$. For each of the following operations, determine the new shape of the tensor as it flows through the network. Note that activations are omitted since they don't change the shape of the data as they act coordinate-wise.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

7. `Flatten`

8. `Linear(in_features=576, out_features=10)`

(b) Again assume your input is a batch of N 64×64 RGB images. Now compute the number of parameters that each layers has. For the convolutional layers, also compute the number of parameters a fully connected layer mapping from the flattened input channels to the flattened output channels would have. It is okay to leave the number of parameters as products and additions such as $64 \cdot 32 + 16$.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

7. `Flatten`

8. `Linear(in_features=576, out_features=10)`