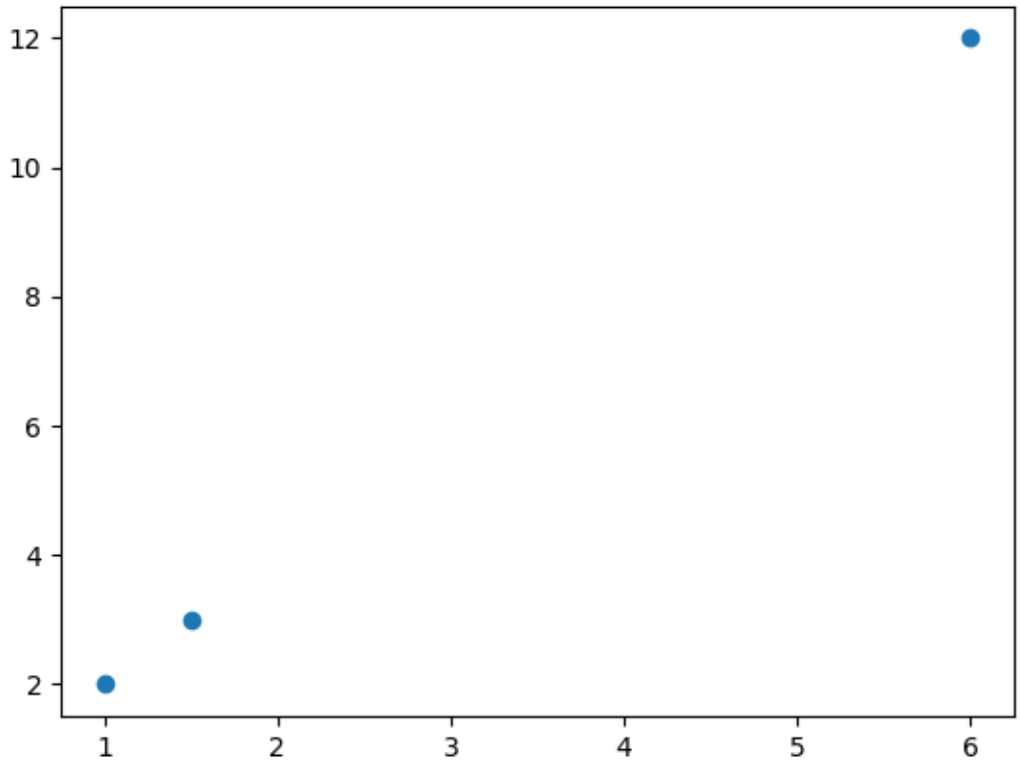
Section 09: Solutions

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?

Solution:

Each point has second coordinate twice the first, so every point is on the line $y = 2x$, or equivalently is a multiple of the vector $[1, 2]$.

That direction, normalized, is the first principal component, so $v_1 = [1/\sqrt{5}, 2/\sqrt{5}] \approx [0.45, 0.89]$.

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

Solution:

Since every data is in the span of the first principal component, any unit norm vector perpendicular to v_1 is an acceptable choice. One such vector is $[-2\sqrt{5}, 1/\sqrt{5}] \approx [-0.89, 0.45]$.

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

Solution:

The first point is $\sqrt{5}v_1$, the second one is $1.5 \cdot \sqrt{5}v_1$, and the third one is $6 \cdot \sqrt{5}v_1$.

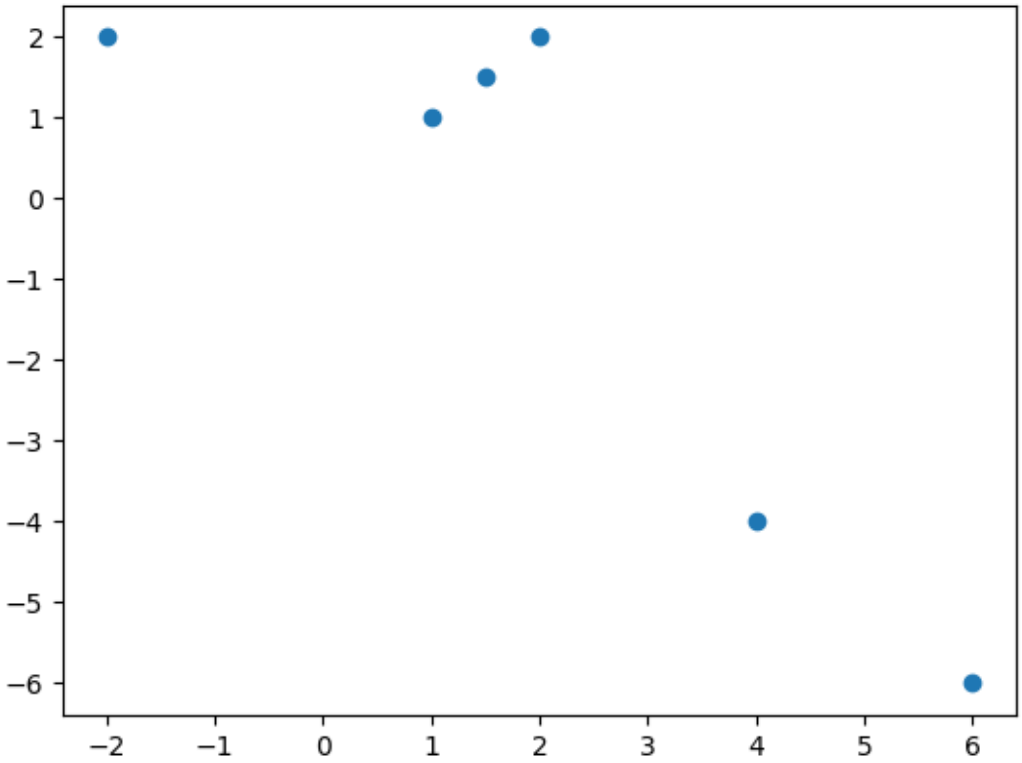
(d) Will this representation be lossy, or perfectly preserve the dataset?

Solution:

In this particular dataset, we perfectly preserve this dataset (the points are all multiples of v_1).

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?

Solution:

Notice that every point is either a multiple of $[1, 1]$ or $[1, -1]$, so some of those must be our principal component. The norms of the multiples of $[1, -1]$ are much larger, so $[1/\sqrt{2}, -1/\sqrt{2}]$ is v_1.

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

Solution:

We need a vector perpendicular to v_1, which can best describe our remaining data. Since we're in two dimensions, we don't have choices after we chose the first principal component. Therefore, the second principal component is $[1/\sqrt{2}, 1/\sqrt{2}]$.

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

Solution:

Data points 1, 2, and 6 are all perpendicular to v_1, so are represented as $[0, 0]$ (i.e. $0 \cdot v_1$). The other points are multiples of v_1, which are $-2\sqrt{2}v_1$, $4\sqrt{2}v_1$, and $6\sqrt{2}v_1$, respectively.

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

Solution:

The data representation is lossy. Points 1, 2, and 6 have lost information.

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

Solution:

They are eigenvectors: $Au_1 = \begin{bmatrix} 4/\sqrt{2} - 1/\sqrt{2} \\ -1/\sqrt{2} + 4/\sqrt{2} \end{bmatrix} = 3u_1$ and $Au_2 = 5u_2$ by a similar calculation.
They are unit norm: $u_1^T u_1 = (1/\sqrt{2})^2 + (1/\sqrt{2})^2 = 1/2 + 1/2 = 1$. The calculation for u_2 is similar.
They are orthogonal: $u_1^T u_2 = (1/\sqrt{2})(-1/\sqrt{2}) + (1/\sqrt{2})(1/\sqrt{2}) = -1/2 + 1/2 = 0$
They form a basis (since they're 2 linearly independent vectors in $\mathbb{R}^2$)

(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.

Solution:

$x^T u_1 = -1/2 + 3/2 = 1. x^T u_2 = 1/2 + 3/2 = 2$. So $x = u_1 + 2u_2$

(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.

Solution:

$U = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$ $D = \begin{bmatrix} 3 & 0 \\ 0 & 5 \end{bmatrix}$

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

Solution:

$$Ax = A\left(u_1 + 2u_2\right) = Au_1 + 2Au_2 = 3u_1 + 10u_2 = \begin{bmatrix} -7/\sqrt{2} \\ 13/\sqrt{2} \end{bmatrix}$$

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.

Solution:

We have $A = USV^T$ then:

$$\begin{aligned} AA^T &= USV^T(USV^T)^T \\ &= USV^T((V^T)^T S^T U^T) \\ &= USV^T VSU^T \\ &= USISU^T \\ &= US^2U^T \end{aligned}$$

Since we can diagonalize AA^T into US^2U^T, it has eigenvectors that are columns of U and 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.

Solution:

We have $A = USV^T$ then:

$$\begin{aligned} A^TA &= (USV^T)^T USV^T \\ &= VSU^T USV^T \\ &= VSISV^T \\ &= VS^2V^T \end{aligned}$$

Since we can diagonalize A^TA into VS^2V^T, it has eigenvectors that are columns of V and 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 $Ax = USV^Tx$

Solution:

Let $\{v_1, v_2, \ldots, v_n\}$ be the rows of V^T. They are orthogonal to each other and unit norm. For any $x \in \mathbb{R}^d$ we can write $x = \sum_{i=1}^{d} \alpha_i v_i$. Then we have:

$$\begin{aligned} USV^Tx &= USV^T \sum_{i=1}^{d} \alpha_i v_i \\ &= \sum_{i=1}^{d} \alpha_i USV^T \ v_i \\ &= \sum_{i=1}^{d} \alpha_i USe_i \\ &= \sum_{i=1}^{d} \alpha_i U\lambda_i e_i \\ &= \sum_{i=1}^{d} \alpha_i \lambda_i u_i \end{aligned}$$

In the meantime, since v_i is an eigenvector of A^TA, we have $A^TAv_i = \lambda_i^2 v_i$ (1)
Multiply A on both sides of (1), we get $(AA^T)Av_i = \lambda_i^2 Av_i$
Therefore, Av_i is an eigenvector of AA^T
If we multiply v_i^T on both sides of (1), we get $v_i^TA^TAv_i = \lambda_i^2 v_i^Tv_i$, which is equivalent to $||Av_i||^2 = \lambda_i^2||v_i||^2$
Therefore, we know that the length of vector Av_i is λ_i
Normalize the vector: $\frac{Av_i}{\lambda_i} = u_i$
Hence, $\lambda_i u_i = Av_i$
Plug it back into the formula:

$$\begin{aligned} USV^Tx &= \sum_{i=1}^{d} \alpha_i \lambda_i u_i \\ &= \sum_{i=1}^{d} \alpha_i Av_i \\ &= A\sum_{i=1}^{d} \alpha_i v_i \\ &= Ax \end{aligned}$$

4. Convolutional Neural Networks

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

Convolutional layers are more flexible than fully connected ones since not all input neurons affect all output neurons. In addition, the number of weights per layer is smaller than that of linear layers, which would ease computation with high-dimensional data.

(b) Discuss the advantages of maxpooling in CNN. **Solution:**

Pooling layers are used to downsample feature maps, which make processing more efficient by reducing the number of parameters to learn.

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)`

   Solution:

   $(N, 16, 64, 64)$

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

   Solution:

   $(N, 16, 32, 32)$

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

   Solution:

   $(N, 32, 30, 30)$

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

   Solution:

   $(N, 32, 16, 16)$

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

Solution:

$(N, 8, 16, 16)$

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

Solution:

$(N, 4, 12, 12)$

7. `Flatten`

Solution:

$(N, 576)$

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

Solution:

$(N, 10)$

(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)`

Solution:

Conv: $3 \cdot 16 \cdot 3 \cdot 3 + 16 = 448$

Fully connected: $3 \cdot 64 \cdot 64 \cdot 16 \cdot 64 \cdot 64 + 16 \cdot 64 \cdot 64 = 805306512$

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

Solution:

0

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

Solution:

Conv: $16 \cdot 32 \cdot 3 \cdot 3 + 32 = 4640$

Fully connected: $16 \cdot 32 \cdot 32 \cdot 32 \cdot 30 \cdot 30 + 32 \cdot 30 \cdot 30 = 471888000$

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

Solution:

0

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

Solution:

Conv: $32 \cdot 8 + 8 = 264$

Fully connected: $32 \cdot 16 \cdot 16 \cdot 8 \cdot 16 \cdot 16 + 8 \cdot 16 \cdot 16 = 16779264$

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

Solution:

Conv: $8 \cdot 4 \cdot 5 \cdot 5 + 4 = 804$

Fully connected: $8 \cdot 16 \cdot 16 \cdot 4 \cdot 12 \cdot 12 + 4 \cdot 12 \cdot 12 = 1180224$

7. `Flatten`

Solution:

0

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

Solution:

$576 \cdot 10 + 10 = 5770$