

## Exercise 1 - Eigenvalues and Eigenvectors

You are given the following set of eigenvalues and eigenvectors. Compute the corresponding matrix.

$$\lambda_1 = 1, \lambda_2 = 2, \mathbf{v}_1 = (\sqrt{0.5}, \sqrt{0.5})^\top, \mathbf{v}_2 = (\sqrt{0.5}, -\sqrt{0.5})^\top.$$

### Solution

First, remember that the normalized eigenvectors of a symmetric matrix are orthogonal. Thus, we have

$$\mathbf{e}_i^\top \mathbf{e}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

Second, for symmetric  $\mathbf{A}$ , its spectral decomposition is given by  $\mathbf{A} = \mathbf{Q} \mathbf{Q}^\top$ , where  $\mathbf{Q}$  is a matrix where each column is an (orthogonal) eigenvector of unit length.

In our case, the eigenvectors are already normalized and orthogonal, so we can simply write  $\mathbf{Q} = (\mathbf{e}_1, \mathbf{e}_2)$  and  $\Lambda = \text{diag}(\lambda_1, \lambda_2)$ . Then, we have

$$\mathbf{A} = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 1.5 \end{pmatrix}$$

## Exercise 2 - Parameter Counting

Use PyTorch to load the `alexnet` model and automatically compute its number of parameters. Output the number of parameters for each layer and the total number of parameters in the model.

### Solution

First, we have to load the `alexnet` model which is part of `torchvision`:

```
import torchvision
alexnet = torchvision.models.alexnet()
```

The number of parameters for the entire model, is the easier part: We can simply use the `parameters()` iterator which returns the set of parameters for each module. Those can then be counted using the `numel()` method resulting in

```
sum(p.numel() for p in alexnet.parameters())
```

Obtaining the number of parameters for each of the layer requires looking into the source code of the `alexnet` model. The structure is similar to `vgg` and the forward pass looks like this:

```
x = self.features(x)
x = self.avgpool(x)
x = torch.flatten(x, 1)
x = self.classifier(x)
```

A closer look at the implementation reveals that we can obtain the individual layers parameter by simply iterating over the `self.features` and `self.classifier` modules. The `self.avgpool` module does not have any parameters. The following code snippet shows how to obtain the number of parameters for each layer of the convolutional backbone:

```
for layer in alexnet.features:
    print(layer, sum(p.numel() for p in layer.parameters()))
```

To get the number of paramers in the cnn head, simply update the code snippet to iterate over `alexnet.classifier` instead of `alexnet.features`.

### Exercise 3 - Convolutional Layers

Consider the following  $4 \times 4 \times 1$  input  $X$  and a  $2 \times 2 \times 1$  convolutional kernel  $K$  with no bias term

$$X = \begin{pmatrix} 1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \\ 1 & -1 & 0 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

- (a) What is the output of the convolutional layer for the case of stride 1 and no padding?
- (b) What if we have stride 2 and no padding?
- (c) What if we have stride 2 and zero-padding of size 1?

#### Solution

- (a) Here, we simply apply the convolutional kernel over each  $2 \times 2$  patch of the input. There are 9 such patches. The output  $Y$  is then

$$Y = \begin{pmatrix} 1 & 3 & -1 \\ 4 & 2 & 2 \\ 5 & 3 & 3 \end{pmatrix}$$

- (b) Same idea except that we skip every other patch resulting in only 4 patches. The output  $Y$  is then

$$Y = \begin{pmatrix} 1 & -1 \\ 5 & 3 \end{pmatrix}$$

- (c) Now, we have added zeros on each side of the input. The resulting  $6 \times 6$  padded input  $X_{\text{padded}}$  and corresponding output  $Y$  are

$$X_{\text{padded}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 1 & 1 & 0 \\ 2 & 2 & 0 \\ 2 & -1 & 1 \end{pmatrix}$$

## Exercise 4 - Scaled Dot-Product Attention

Consider the matrices  $Q$ ,  $K$ ,  $V$  given by

$$Q = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 0 & 1 \end{pmatrix}, \quad V = \begin{pmatrix} 1 & 0 & -2 \\ 2 & 1 & 2 \\ 0 & 3 & -1 \end{pmatrix}.$$

Compute the context matrix  $C$  using the scaled dot product attention.

### Solution

The resulting context matrix is given by:

$$C \approx \begin{pmatrix} 1.80 & 1.00 & 1.44 \\ 1.26 & 1.25 & 0.26 \end{pmatrix}$$

A simple implementation would look as follows:

```
import torch
Q = torch.tensor([[1, 2], [3, 1]]).float()
K = torch.tensor([[2, 1], [1, 1], [0, 1]]).float()
V = torch.tensor([[1, 2, -2], [1, 1, 2], [0, 1, -1]]).float()
d_k = torch.tensor(K.shape[1])
M = torch.matmul(Q, K.transpose(0, 1)) / torch.sqrt(d_k)
S = torch.exp(M) / torch.sum(torch.exp(M), dim=1).view(-1,1)
torch.matmul(S, V)
```

Pytorch also provides a function for scaled dot product attention:

```
import torch.nn.functional as F
F.scaled_dot_product_attention(Q, K, V)
```