

The exercises this week involve some old material so you can check your learning and understanding.

### Exercise 1 - Maximum Likelihood Estimator

Assume you are given datapoints  $(x_i)_{i=1}^N$  with  $x_i \in \mathbb{R}$  coming from a Exponential distribution. The probability density function of a exponential distribution is given by  $f(x) = \lambda \exp(-\lambda x)$  with  $x \in \mathbb{R}$ . Derive the maximum likelihood estimator of the parameter  $\lambda$ .

#### Solution

First, let's quickly remember that the maximum likelihood estimator (MLE) of a probability distribution from datapoints  $\mathbf{x}_1, \dots, \mathbf{x}_N$  is given by

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta \in \Theta} \prod_{i=1}^N f(\mathbf{x}_i | \theta),$$

where  $f$  is the probability density function of the considered probability distribution family,  $\theta$  are the parameters of the distribution, and  $\Theta$  is the parameter space (a set containing all possible parameters).

As mentioned in our previous exercise, we usually work with the log-likelihood in practice. In this particular case, the log likelihood is given by

$$\begin{aligned} l(\lambda | x_1, \dots, x_N) &:= \sum_{i=1}^N \ln f(\mathbf{x}_i | \lambda) \\ &= \sum_{i=1}^N \ln (\lambda \exp(-\lambda x_i)) \\ &= \sum_{i=1}^N \ln(\lambda) + \ln(\exp(-\lambda x_i)) \\ &= N \ln(\lambda) - \sum_{i=1}^N \lambda x_i. \end{aligned}$$

The derivative with respect to  $\lambda$  is

$$\frac{\partial l(\lambda | x_1, \dots, x_N)}{\partial \lambda} = \frac{N}{\lambda} - \sum_{i=1}^N x_i.$$

The MLE is obtained by setting it to 0 and solving for  $\lambda$  as

$$\hat{\lambda}_{MLE} = N \left( \sum_{i=1}^N x_i \right)^{-1}.$$

### Exercise 2 - 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

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

- (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

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

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

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

- (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

$$X_{\text{padded}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 2 & 4 \\ 0 & 1 & -1 \end{pmatrix}$$

### Exercise 3 - Computational Parameter Counting

Use PyTorch to load the `vgg11` 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 `vgg11` model which is part of `torchvision` as has been shown in the lecture:

```
import torchvision
vgg11 = torchvision.models.vgg.vgg11(pretrained=False)
```

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 vgg11.parameters())
```

Obtaining the number of parameters for each of the layer requires looking into the source code of the vgg11 model. All VGG models are ultimately instantiated by using the `VGG` class. Its 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 vgg11.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 `vgg11.classifier` instead of `vgg11.features`.

#### Exercise 4 - Influence Functions

Let  $\hat{\theta}$  and  $\hat{\theta}(\epsilon)$  be as defined in class. Show that the first order Taylor expansion of  $\hat{\theta}(\epsilon)$  around  $\epsilon = 0$  is given by the equation given in class, i.e. by

$$\hat{\theta}(\epsilon) \approx \hat{\theta} + \epsilon \left. \frac{d\hat{\theta}(\epsilon)}{d\epsilon} \right|_{\epsilon=0}.$$

#### Solution

First, let's recall the definitions of  $\theta$  and  $\hat{\theta}(\epsilon)$ :

$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{N} \left[ \sum_{i=1}^N L(x_i, y_i; \theta) \right]$$

$$\hat{\theta}(\epsilon) = \operatorname{argmin}_{\theta} \frac{1}{N} \left[ \sum_{i=1}^N L(x_i, y_i; \theta) \right] + \epsilon L(x, y; \theta)$$

The first order taylor series expansion around  $\epsilon = 0$  is given by

$$\hat{\theta}(0) + \epsilon \left. \frac{d\hat{\theta}(\epsilon)}{d\epsilon} \right|_{\epsilon=0}$$

From the definitions above, it can directly be seen that  $\hat{\theta}(0) = \hat{\theta}$  which completes the proof.