DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

WI4635 LINEAR ALGEBRA AND OPTIMIZATION FOR MACHINE LEARNING – PROJECT 2

Instructions

- The ideal solution (grade 10) includes both the implementation as well as a discussion of the training, validation, and test results. The implementation has to be commented/documented as well as algorithmic choices explained and justified.
- Suboptimal performance or unexpected behavior of the model/implementation should be discussed.
- A submission should contain a PDF file of the report as well as a zipped folder containing the code, data, as well as instructions on how to execute the code. The code may be provided in form of a Python script or a Jupyter notebook. If during the grading, it is not clear how to run the code, we may assume that the code does not work.
- **The deadline is December 22, 2023, 23:59 CET. Please submit one project per group through Brightspace.**

Motivation

As discussed in the lectures, convolutional neural networks (CNNs) are a class of neural network architectures which are well-suited for image processing tasks. In this project, you will implement your own CNN model in several incremental steps and verify the individual building blocks of your implementation. Finally, you will test it on image classification for the famous MNIST [1] data set, which contains images of handwritten digits (see Figure 1).

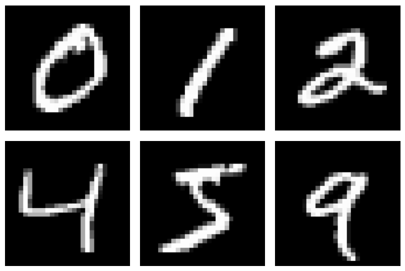
Figure 1: Exemplary images from the MNIST dataset [1].

The data sets can be easily downloaded, for instance, using the TensorFlow module `tf.keras.datasets`. The data set can be downloaded via the code:

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

1 Discrete Convolution

(a) Implement a Python function that performs a discerte convolution on a gray-scale input image. The function should additionally take a kernel matrix of arbitrary size as input. In order to test your implementation, employ exemplary images from the MNIST data set as well as the following kernels:

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
5×5 Gaussian blur	$\frac{1}{9}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 1 \\ 6 & 24 & 36 & 24 & 1 \\ 4 & 16 & 24 & 16 & 1 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$

Note: *The implementation may be simplified by first extracting patches of the size of the kernel matrix from the input image. Then, each coefficient of the output image can be computed using only this patch and the kernel matrix.*

(b) Allow for using multiple kernels of the same size as input. Given a single gray-scale input image and n kernels, the function should produce n output images, where each image results from the application of a different kernel.

2 Trainable Kernel

When implementing a CNN, we need to be able to learn the coefficients of the kernel matrices. In this exercise:

(a) For a small but arbitrary input image and kernel (for instance, Example 5.1 from the lectures), derive the computational graph as well as one step of forward and backward propagation by hand.

(b) Write a Python class for a single convolutional layer of a CNN. The class should at least store the weights of the trainable kernel matrices; in the constructor, these should be initialized randomly. Moreover, the class should at least contain the functions `forward_propagation()` and `backward_propagation()` for performing the forward respectively backward propagation.

(c) Test your implementation in the following way:

- Choose three different 3×3 kernels from the previous Exercise 1 and apply them to all MNIST images

$$X = x_1, \dots, x_N,$$

resulting in

$$Y = y_1, \dots, y_N.$$

Note that each x_i has a single channel, and due to the three kernels, the resulting y_i have three channels.

- Implement a “simple” mini-batch stochastic gradient descent (SGD) optimizer to reproduce the kernels that have been used to compute the reference data Y in the previous step. In particular, solve the optimization problem with mean-squared error (MSE) loss

  $$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \|\mathrm{Conv}_\theta(x_i) - y_i\|^2,$$

  where Conv_θ is your convolutional layer and θ are the coefficients for two 3×3 kernel matrices. If you are successful, after training via SGD, the learned coefficients should be close to those used to generate the reference data. If necessary, you may tune the optimization, for instance, by adjusting the learning rate of SGD. Discuss your results.

3 Convolutional Neural Network

The final part deals with implementing a small CNN for image classification. Therefore:

(a) Implement Python classes for max-pooling and softmax layers (with backward and forward propagation):

- A max-pooling layer is a nonlinear layer without trainable weights; see the lectures and [2] for more details.
- A softmax layer is a layer of the form

  $$\sigma(Wx + b),$$

  where W and b are a weight matrix and bias vector, respectively. Moreover,

  $$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

  is the softmax acivation function for some input vector $x \in \mathbb{R}^n$. We have that

  $$0 \le \sigma(x)_i \le 1, \quad 1 \le i \le n, \quad \text{and} \quad \sum_{i=1}^{n} \sigma(x)_i = 1,$$

  and hence, the outputs can represent probabilities for the input being in classes $1, \dots, n$. The idea is then that the output is associated with the class with the highest probability.

(b) Test your CNN implementation for the classification of the MNIST images. Therefore, employ a CNN with the architecture

$$CNN(x) = \mathrm{SoftMax}(\mathrm{MaxPooling}(\mathrm{Conv}(x))),$$

and optimize the cross-entropy loss function

$$-\sum_{x_i}^{N} y_i \log(CNN(x_i))$$

using a mini-batch SGD as in Exercise 2; here, x_i is an input image and y_i is the corresponding reference classification, which is of the form

$$y_i = \begin{pmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{pmatrix}^\top,$$

where the 1 is in the row corresponding to the correct class.

In order to train and test the neural network, proceed as follows:

- Split the data into training and test data.
- In order to find a good set of hyper parameters of your CNN model, perform k-fold cross validation; see, for instance, [4, Section 11.10] for details.
- Finally, test the performance of your model on the test data.

References

[1] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* Adaptive Computation and Machine Learning series. MIT Press, 2016.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[4] Jeremy Watt, Reza Borhani, and Aggelos K Katsaggelos. *Machine learning refined: Foundations, algorithms, and applications.* Cambridge University Press, 2020.