Due date: November 12, 2024
Please submit a pdf with answers (add figures if appropriate)
as well as the notebook used to generate them.

In this exercise, we want to use neural networks (NNs) for non-linear classification, i.e., given some features we want to predict to which class the sample belongs. We compare two different architectures: A simple multilayer perceptron (MLP) and a *convolutional* neural network (CNN). As an example, we use a subset of the classic `MNIST` dataset of handwritten digits and we want to train a model to predict the number between 0 and 9 shown on the image. To use an already existing dataset, we need to install `torchvision` by typing `conda install -c pytorch torchvision`.

1 MLP (4 pt)

1.1 MLP classifier for images

We want to predict the probability of a given sample to belong to a certain class, rather than predicting the value of a function as done in the regression problem. In principle, this task is very similar to the regression problem from Exercise 03. However, to use a NN as classifier, we need to apply a few modifications:

1. Our simple MLP can only work with vectors of rank 1, but not with rank-2 objects such as images. We therefore need to *flatten* any input image to a vector which can be done using a flattening layer (`nn.Flatten(start_dim=1, end_dim=-1)`). Define a new class `MLPClassifier` (based on the `MLP` class from Exercise 03) and add a flattening operation as first layer to the `MLPClassifier` class.

2. The MSE is not a good loss function for a classification problem. Instead, use the cross entropy loss (`torch.nn.CrossEntropyLoss`).

3. Also for evaluating the model performance, the MSE is only of limited usefulness. Instead, we are interested in the number of correctly classified images (classification accuracy). This number can be obtained using the function `evaluate_classification` which is provided in the notebook.

1.2 Fitting

Similar to the regression problem in Exercise 03, we first want to find a suitable architecture. Feel free to play around with the number of layers and the number of nodes per layer and experiment yourself - otherwise we suggest to use 2 hidden layers with 40 nodes per layer in addition to the input layer of 784 nodes (corresponding to the flattened input of an 28×28 pixels image) and the output layer of 10 nodes (corresponding to the number of classes).
For your chosen NN architecture, visualize the classification accuracy as a function of the number of training epochs.

2 Convolutional NN (6 pt)

As we have seen in the MNIST part, our simple MLP needs to flatten rank-2 tensors such as images because it is only able to deal with vectors. In this exercise we want to use a CNN, which is able to deal with images directly, as classifier. What are other advantages of using a CNN over an MLP for image classification?

2.1 Class definition

Define a new class `CNN` which implements a CNN. Your CNN should consist of 2 blocks each implementing (in this order!)

1. a 2D convolution (`nn.Conv2d`)
2. a ReLU activation function (`nn.ReLU`)
3. a 2D max pooling (`nn.MaxPool2d`)

Each block can be built using `nn.Sequential`. Check the pytorch documentation for `nn.Conv2d` and `nn.MaxPool2d` to find how the shapes of inputs and outputs relate. Finally, use a linear layer to map the output of the last convolutional block to a vector of size 10 to represent the 10 different classes. Don't forget to flatten data before feeding it to the linear layer – it can only handle vectors!

2.2 Fitting

Train your CNN for image classification using the MNIST dataset. You can again experiment with the parameters of the model or use the following parameters for the convolutions:

- Number of input channels of first block: 1 (since we only have grayscale images and not RGB)
- Number channels: 16 (this is the number of output channels of the first block and the number of input channels of the second block)
- Kernel size: 5
- Kernel size max pooling: 2
- Stride: 1
- Padding: 2

For the chosen architecture, how many epochs do you need to train this network until convergence (have a look at how the accuracy evolves over time)? How many trainable parameters does your model have? Compare the accuracy and the number of parameters to the corresponding quantities of the MLP part.