Exercise 5
Introduction to PyTorch

The deep learning framework PyTorch will form the basis of our implementations from this exercise onward. This week, we want you to get to know the framework on a simple supervised learning task. If you're already familiar with PyTorch, this exercise should be mostly familiar.

1 Setting up PyTorch

Install PyTorch to your local machine by following the instructions at `https://pytorch.org/get-started/locally/`. We recommend using Anaconda for the installation. You do not need to install CUDA, or use a GPU for this exercise.

Test your installation by creating a PyTorch `tensor` (equivalent to a numpy array) and printing it:

```
import torch

a = torch.arange(25)
a = a.reshape(5, 5)
print(a)
b = torch.tensor([1, 2, 3, 4, 5])

# Example 1
print(a * b)

# Example 2
print(a @ b)
```

What is the difference between examples 1 and 2?

2 Supervised Learning with Neural Networks in PyTorch

In this task, you will learn to train a simple classifier on the MNIST dataset. You can find documentation for the `torch.nn` module at `https://pytorch.org/docs/stable/nn.html`.

1. In the class 'Net', create a network with the following architecture:
   - Conv 2D (32 Channels, 3x3 Kernel, Stride 1)
   - ReLU
   - Conv 2D (64 Channels, 3x3 Kernel, Stride 1)
   - ReLU
   - Fully Connected Layer ($9216 \Longrightarrow 128$)
   - ReLU
   - Fully Connected Layer ($128 \Longrightarrow 10$)
   - Softmax

   You'll need to create the appropriate members for your layers in `__init__` and perform the calculations in the `forward`-method. PyTorch will perform any needed gradient calculations for you.
2. Create an optimizer (e.g., `th.optim.Adam`) for your model.
3. We already provide the basic outer training loop that will iterate over epochs and batches in `main`. Implement one step of prediction, gradient calculation, and optimization in the `train` and `test` methods.
4. Run the training. Your training and evaluation losses should decrease substantially.