Heidelberg University
Institute of Computer Engineering
HAWAII Lab

Hendrik Borras
Xiao Wang

Neural Networks from Scratch
Summer Term 2026

Exercise 2

- **Return electronically until Wednesday, April 29, 2026, 09:00**
- **Include your names on the top sheet. Hand in only one PDF.**
- **A maximum of three students are allowed to work jointly in a group.**
- **When you include plots add a short explanation of what you expected and observed.**
- **Hand in source code if the exercise required programming. You can bundle the source code along with the PDF in a .zip file.**

2.1 Neural network from scratch

In this exercise we are going to implement a small three layer MLP with sigmoid activations from scratch and then train it on the MNIST dataset.
For this we provide two template scripts, one uses *numpy* and the other uses *jax*, either of which you can use as a starting point. If you choose to explore both, it would be beneficial to include a comparison of their performance, for example the runtime . The templates already contain a lot of the structure of the network, dataset loading and basic training routine, so the main task boils down to implementing the forward and backward paths and update routines for different components.

- **Note.** If you follow implementation in the *numpy* template, then the script won't use any GPU capability. Given the size of the MLP and task, this is fine–we will investigate GPU acceleration in the next exercise. Make sure to use only *numpy* or native Python for computations. PyTorch is only to be used for the dataloaders.
- **Note.** *Jax* is relatively new Python library for machine learning. It provides a NumPy-style API and is designed to accelerate computing for Neural Networks in particular. For more information, please refer to the official documentation. If you follow the implementation in the jax template, then several validation functions are already provided to help you verify the correctness of your backward computations.
- **Setup.** First, set up your environment. Again, you can use `uv` as described in the previous exercise and with the reference `pyproject.toml`. For this exercise make sure that in particular PyTorch and JAX are installed.
- **Building the MLP.** Implement the missing forward pass and backward pass and update routines for the three-layer MLP, as outlined in the `ex21_template.py` file. These are in particular the central components of the Linear/Dense layer, Sigmoid activation function and softmax function.
- **Training.** Train the network with the given default parameters for 30 epochs. Then plot how the test and train loss develop with the number of epochs. Additionally create a plot which shows how the test accuracy develops.

  Note: If you would like to monitor training progress from the CLI in a convenient way, you may want to look into `tqdm` progress bars (or a shiny alternative like `alive`). Add the dependency, modify the first line of the loop, and you are good to go.
- **Investigate learning rate.** Run the training again for 30 epochs, but this time with varying learning rates. Choose at least five different learning rates between 1. and 0.001. Create plots of the test accuracy and loss over epochs to compare the different learning rates.
- **Discussion.** Discuss the results obtained in the previous plots.