Heidelberg University
Institute of Computer Engineering
Computing Systems Group

Holger Fröning
Hendrik Borras

Embedded Machine Learning
Summer Term 2023

Exercise 3

- **Return electronically until Wednesday, May 24, 2023, 09:00**
- **Include your names on the top sheet. Hand in only one PDF.**
- **A maximum of nine students are allowed to work jointly on the exercises.**
- **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.**
- **Programming exercises can only be graded if they run on the cluster in the provided conda enviroment. Make sure to document additional steps, which you might have taken to run the exercises.**

3.1 PyTorch: Automatic differentiation

In this exercise we will get to know the automatic differentiation capabilities of PyTorch. In particular we will see how this makes building neural networks easier and how we can thus more easily make use of GPU acceleration.
Note: If you would like to complete this exercise on your own machine you must have an NVidia GPU present and accessible to PyTorch.

- First, setup your conda environment on the cluster or on your own machine, as explained in the brook user manual. Please use the included `conda_env_ex03.yaml` file to create the environment.
- For this exercise we provide a source code template called `exercise03_template.py` you can use this as a starting point for all exercises on this sheet.
- Implement the MLP from the previous exercise ("2.2: Neural network from scratch") with automatic differentiation in Pytorch.
- Run a training run with the new network implementation once on the CPU and once on the GPU (with the default parameters, but for 30 epochs) and plot the test accuracy over time, not over the number of epochs.
- Discuss the results obtained in the previous plot.

(20 points)

3.2 CNN and MLP on SVHN

Next we will move on to a more complex neural network architecture and dataset. The network will be adapted to a convolutional architecture, which is more fitting for image classification. And as a dataset we will use the Street View House Numbers (SVHN)[1] In particular we will still be looking at a number classification task, however this one with images taken from the outside world and in RGB, so much more complex, when compared to MNIST.

- Implement a small convolutional network with PyTorch's automatic differentiation feature. The network architecture should be as following:
  - Layer 1: Convolution: Input channels: 3, Output channels: 32, Kernel size: 3, Stride: 1
  - Layer 2: Convolution: Input channels: 32, Output channels: 64, Kernel size: 3, Stride: 2
  - Layer 3: Convolution: Input channels: 64, Output channels: 128, Kernel size: 3, Stride: 1

[1]`http://ufldl.stanford.edu/housenumbers/`

  - Layer 4: Flatten layer
  - Layer 5: Linear: Input channels: 18432, Output channels: 128
  - Layer 6: Linear: Input channels: 128, Output channels: 10 Linear(128, 10)
  - Activation function after computational layers: ReLu
  - Output activation function: LogSoftmax
- Switch the dataloader from MNIST to SVHN[2]. Make sure to remove the dataset normalization from the dataset transformation, which is specific to MNIST.
- Train both the MLP and CNN on SVHN for 30 epochs.
- Note: You may need to reduce the learning rate for the CNN, compared to the MLP to get more stable results.
- Plot the test accuracy curves for both the MLP and CNN on SVHN over the number of epochs in one plot.
- Plot the test accuracy curves for both the MLP and CNN on SVHN over training time in one plot.
- Discuss the results obtained in the previous plots.

(30 points)

3.3 Optimizers

Next we will move on to more complex optimizers to hopefully find better optima.

- Run the CNN on SVHN for 30 epochs and with different optimizers. Compare how the test accuracy evolves for different optimizers over the epochs in a plot.
- Make sure to test at least three optimizers and include at least the following ones: SGD and ADAM. A complete list of built-in optimizers can be found in the PyTorch documentation[3].
- Furthermore make sure to test different learning rates for each optimizer. A range between 0.1 and 0.001 is advisable. This search dosen't have to be exhaustive, but for the final plot you should choose an learning rate that works reasonably well for each optimizer. Make sure to note down which optimizer used which learning rate in the final result.
- Discuss the results obtained in the previous plot.

(10 points)

3.4 Willingness to present

Please declare whether you are willing to present any of the exercises from this sheet. The declaration should happen on the PDF which you hand in.
The declaration can be made on a per-exercise basis. Each declaration corresponds to 50% of the exercise points. You can only declare your willingness to present exercises for which you also hand in a solution. If no willingness to present is declared you may still be required to present an exercise for which your group has handed in a solution. This may happen if as example nobody else has declared their willingness to present.

- PyTorch: automatic differentiation (Section: 3.1)
- CNN and MLP on SVHN (Section: 3.2)
- Optimizers (Section: 3.3)

(30 points)

Total: 90 points

[2]`https://pytorch.org/vision/stable/generated/torchvision.datasets.SVHN.html`
[3]`https://pytorch.org/docs/stable/optim.html#algorithms`