Machine Learning and Physics
Winter Semester 2022/2023

Exercise Sheet #9

Due: 16.01.23 11h00

Roman Remme

General Regulations.

- Please hand in your solutions in groups of three people. A mix of attendees from Monday and Tuesday tutorials is fine.
- Your solutions to theoretical exercises can be either handwritten notes (scanned), or typeset using LaTeX. In case you hand in handwritten notes, please make sure that they are legible and not too blurred or low resolution, otherwise we might not correct your submission.
- For the practical exercises, the data and a skeleton for your jupyter notebook are available at https://github.com/hci-unihd/mlph_sheet09. Always provide the (commented) code as well as the output, and don't forget to explain/interpret the latter. Please hand in both the notebook (`.ipynb`), as well as an exported pdf.
- Submit all your files in the Übungsgruppenverwaltung, only once for your group of three.

1 Jet-Tagging with PyTorch

In this task, you will use PyTorch to train a neural network on the jet-tagging data. If you do not yet know how to do something, you can find many good tutorials for PyTorch online, for instance at https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html.

(a) Load the data and split it into train, validation and test set. Convert each split to a `torch.utils.data.TensorDataset`. The validation and test set should each contain $N = 200$ data points with the rest making up the training set. (1 pt)

(b) Instantiate an MLP for the classification problem with the appropriate number of in- and outputs, using `torchvision.ops.MLP`. Use ReLU activations and four hidden layers with 128 neurons each. (1 pt)

(c) Instantiate the optimizer and criterion: Use Adam with the default learning rate of 10^{-4}, and cross entropy as the loss function. (2 pts)

(d) Implement the training loop and train the neural network on the jet tagging data, for at least 30 epochs. Plot the training loss over the training iterations. After each epoch, compute the training accuracy of that epoch and evaluate the model on the validation set to compute the validation loss and accuracy. Plot both accuracies versus the training iterations and discuss the results. (4 pts)

(e) Add functionality to your training loop that saves the model after each epoch, if the validation loss is the lowest encountered so far. After training, load this model and evaluate the performance on the test set. (2 pts)

(f) Bonus: Tune some hyperparmeters: Try different activation functions, e.g. ELU, tanh, and shallower / deeper as well as wider and less wide networks. Which combination gives the minimal validation loss? Discuss your results and evaluate the test accuracy of your best model. (2 pts)

2 CNNs for Galaxy Classification

The Galaxy10 SDSS[1] dataset consists of 21785 colored (green, red and near-infrared band) images of galaxies with a resolution of 69 by 69 pixels, taken from the Sloan Digital Sky Survey (SDSS). They have been

[1] https://astronn.readthedocs.io/en/latest/galaxy10sdss.html

annotated by human volunteers: Each image is assigned to one of nine classes which describe the morphology of the depicted galaxy. In this task, you will train Convolutional Neural Networks (CNNs) to classify the images into those classes.

(a) Implement a small CNN for the classification task: It should consist of three consecutive blocks:

1. A convolutional layer with kernel size 5 and 8 output features, ReLU activation and max-pooling,
2. A convolutional layer with kernel size 5 and 16 output features, ReLU activation and max-pooling,
3. A flattening layer that reshapes the channel, width and height axes into a single axis, followed by an MLP with two hidden layers of size 64 and 32.

(2 pts)

(b) Train the model and make the same corresponding plots as in 1(d). Create and plot a confusion matrix of the trained model on the validation set. (4 pts)

(c) Repeat the above with a more powerful architecture: Modify a Resnet34 (as implemented in torchvision) to the correct number of output channels. Note: For this, training on a GPU using google-colab[2] should result in a significant speedup compared to most CPUs. (2 pts)

(d) In contrast to natural images, which oftentimes have a preferred orientation, the galaxies in this dataset are oriented arbitrarily. Hence, it would be desirable to adapt the model such that its prediction is invariant with respect to at least some rotations, e.g. that rotating the input image by a multiple of 90° does not change the outputs. How could this be achieved? Can you also come up with a method that would work for a larger set of rotations? (2 pts)

3 Bonus: Receptive Field of the U-Net

Using pen and paper, compute the field of view of the original U-Net, i.e. the set of input pixels that the prediction in a certain output pixel depend upon. How would you tackle this computationally? (5 pts)

[2] https://colab.research.google.com/