A.B. Meyer (DESY), 24 February 2022

Statistical Methods in Data Analysis, KSETA

code by Thomas Keck, DA lecture, KIT 2017, exercise 9

Keras and Tensorflow on MNIST Dataset

The classification of handwritten digits is the standard problem for image classification. In this exercise, we will process 70000 labeled images of handwritten digits from the MNIST[1] dataset in order to train and test (deep) neural networks on this task.

Following these instructions, you will gain experience with state-of-the-art software packages used for large scale deep learning and you can experiment with your own neural network designs and compare the results with modern setups.

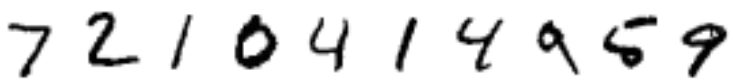

Figure 1: Example images from the MNIST dataset

- **Setup and training**

  Read the script `install_dependencies.sh` and run it. It will set up a Python virtual environment that holds all needed software such as `TensorFlow` (`www.tensorflow.org`) and `Keras` (`www.keras.io`). Activate the virtual environment in every new terminal: `source py_venv/bin/activate`.

  `TensorFlow` is one of the most popular and powerful tools in the machine learning community and can be used to build, train and execute large scale machine learning models. The core concept of `TensorFlow` is the representation of the information flow as tensors in a graph. The wrapper `Keras` hides this concept in large parts to make the library much easier to use.

  The MNIST dataset (`http://yann.lecun.com/exdb/mnist/`) contains in total 70000 images of handwritten digits. The images are in greyscale with 28×28 pixels each (see Figure 1 for some examples).

  This exercise is shipped with the script `download_dataset.py` that downloads and extracts the dataset as binary and converts for you some example images from the binary dataset as `png` files (the inverted images of Figure 1). Execute the script, have a look at the example images and feel free to read the code.

  In preparation of the training, read the code of the script `train.py` and identify the part, where the machine learning model is defined. The model contains all relevant components, i.e. convolutional, dense and maximum pooling layers. It specifies activation functions and optimizer. Full documentation can be found on the `Keras` (`www.keras.io`) webpage.

  Execute the script `train.py` and determine whether you are in the underfitting or overfitting regime. How long (wallclock) does one epoch take? Explain how the number of trainable parameters, 11022, comes about. What would be the number of parameters for a conventional (dense) neural network with one hidden layer and 10 outputs?

- **Improved training and testing**

  Now, modify the model defined in the file `train.py` and try to achieve the best global accuracy. Hint: You will need to increase the model capacity, e.g. number of convolution

[1] (Modified) National Institute of Standards and Technology

filters and possibly add additional dense layers. Optimization settings are usually not so critical, but you are free to experiment with them, too, of course. Be careful with the increase of model capacity, as the CPU requirements take off quickly.

For your trained model, produce an estimate of your achieved accuracy by running the script `apply.py` manually on twenty images, i.e. `./apply.py example_input_*.png`. How does your estimate compare with your expectation? Can you explain the result?

Have a look at the website http://yann.lecun.com/exdb/mnist/, which holds a leaderboard for the test dataset with the performance of modern (deep) machine learning models. You might want to compare your setup with popular models such as `LeNet-5`, `VGG-16`, `AlexNet` or `Inception-v4` to get an idea of the complexity of modern neural network architectures.

- **Application on other data**

  Use `GIMP` (or any other graphics editor) to create images of your own handwritten digits and evaluate whether the performance you experience with your own example images matches the performance achieved during training on the MNIST dataset. You need to create greyscale `png` images with 28×28 pixels, the background of the image has to be black and the digits have to be written in white color. Does the model classify your images correctly? If not, what can be possible reasons that it does not work as expected?