Machine learning II
Master Data Science
Winter Semester 2019/20

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Prof. Tim Downie

Workshop 10
Neural Networks for image data

In Exercise 2 you will fit a neural network to the MNIST zip code digits using a convolutional neural network as covered in the lecture. To do this, you need to install the `mxnet` package using the code at the top of the source file `ML2_Workshop10.R`. Note that site where up to date package is located has to be specified in `install.packages()`. Older versions can lead to comparability problems.

MX-NET[1] is a project providing executable code as a back end, which can be called from the R, Python or C++ environments. This has the advantage that the code is not platform specific, but the disadvantage that the interface is quite tricky and the documentation is often too brief. Because the interface has to be exact, I have provided complete code (except for directory path names).

Download the `MNIST` data in moodle and the R source code. There are plenty of comments in the source file, but read through this overview first.

In the last few Workshops you have used several different packages for using Neural networks in R. There are advantages and disadvantages between these packages. A comparison and summary of these and other R-packages can be found at
`www.is.uni-freiburg.de/ressourcen/r-oeffentlicher-zugriff/deep-learning-in-r`
Link

Exercise 1 A fully connected neural network using MX-NET
To learn the concepts and syntax of MX-NET we will start by fitting a simple fully connected neural network.

The `Sonar` data consist of 60 variables corresponding to specific frequencies and an outcome variable which is either “rock” or “metal” according to what surface the sound signals was bounced off.

After splitting the data into training and test sets run the standard NN using one function `mx.mlp()`.

[1] `https://mxnet.incubator.apache.org/`

The strength of MX-NET comes from being able to define each layer of a NN using a separate function and these are joined together in a chain. So the function

```
chain.data <- mx.symbol.Variable("data")
```

initiates the chain by defines the input layer and gives the resulting R-object the name `chain.data`. The object `chain.data` is the passed into the fully connected layer, whose output is stored into `chain.fc1`, which is in turn passed as an argument to the "softmax" output layer.

Once the structure of the NN has been set up the training is done using the function

```
mx.model.FeedForward.create()
```

Exercise 2 Image classifier

There are many sources for the `MNIST` handwritten digit image data. The data in Moodle come from the website of LeCun et al.[2], who originally provided the data and have done plenty of research in this field, e.g. developing the feature detectors mentioned in the lecture.

The data file contains a list object which contains elements `train$x`, `train$y`, `test$x` and `test$y`. The `x` data are the 28×28 pixel images are stored as vectors, each vector is one row in a matrix. The vector `y` contains the labels, the digits 0 to 9. The full training data contains 60 000 handwritten and labelled images and the training data 10 000.

The following NNs test the limits of standard PCs/Laptops, and your computer might run out of memory when training the full data, or run very slowly. The source code takes a training set of 10 000 images and a test set of 5 000 images, which is adequate to get good results, while still converging within a couple of minutes.

The first model is a fully connected model with two hidden layers. This is the type of neural network which doesn't take advantage of the spatial information in the image. The NN gets about a 94% correct classification rate.

The second model uses two convolutional layers each time followed by max-pooling, the second pooling layer feeds into a fully connected hidden layer. Note that the local receptive field has size 5×5, but these weights are optimised, no feature detector is used.

Once you have run the code, try increasing the size of data. You should find that the accuracy rate gets even better, but takes considerably longer to run. On my home laptop the convolutional NN took 11 minutes to train the full data set.

[2] `http://yann.lecun.com/exdb/mnist/`