UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing


Prof. Björn Ommer
ommer@uni-heidelberg.de

**Exercise sheet 7: Learning and Neural Networks**

Due on *15/12/2017*, 2pm.

In this exercise we will move from the concepts of learning to training neural networks.

**Question 1: Types of Learning** **(2P)**

Please describe in your words the following types of learning: supervised, semi-supervised, unsupervised and reinforcement learning. For each type of learning also state the data required and provide examples.

**Question 2: Learning as Optimization Problem** **(2P)**

Assume that we want to learn a parameterized function

$$y = f(x, W),$$

where $W$ are the parameters to learn. Moreover, assume that we have given training samples $(x_i, y_i)$, $i = 1, \ldots, m$. We would like to determine $W$ such that the error $e_i := f(x_i, W) - y_i$ in absolute value becomes as small as possible.

a) Set up an appropriate optimization problem

$$\min_W F(W, \{(x_i, y_i\}_{i=1}^m). \tag{1}$$

Of course, $F$ should depend on $e_i$. It should be chosen in a way that $\frac{\partial}{\partial e_i} F$ is differentiable.

b) Assuming that $f$ is differentiable with respect to $W$, what would be a possibility to find a solution for (1)?

c) An example for $f(x, W)$ would be a neural network with parameters $W$. The size of a network and the number of its parameters are usually chosen according to the complexity of the problem at hand. Looking at the 'triple junction' dataset in Fig. 1, what is the minimal number of layers needed to learn a good classifier? If you do not know, make a guess. In any case provide some written reasoning.

**Question 3: Implementing a simple Classification example (4P)**
We consider two classification problems with labeled data points in two dimensions:

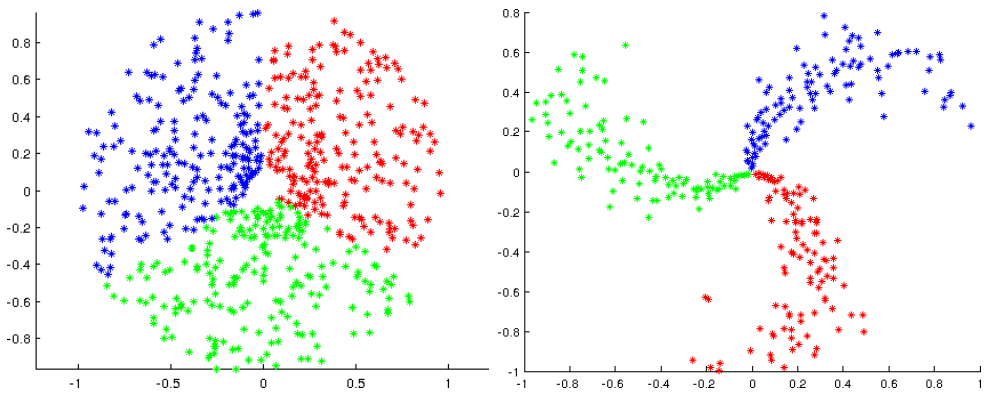

Figure 1: Two 2D classification problems with three classes: 'triple junction' on the left and 'spiral' on the right.

Implement a neural network to solve *both* classification tasks. Base your code on PyTorch[1]. The data will be provided as text files in the supplementary material (actual data might be slightly different to Fig. 1). There will be a training and a test-set for both datasets.

In the following you will go through the basic steps of preparing you data, defining you network models, optimizing the model while keeping track of loss and accuracy, and finally reporting you results on the test-set. Fridays (08.12.) tutorial provides a good introduction and should get you going. You find additional examples on the tutorial page of PyTorch: `http://pytorch.org/tutorials/`.

a) In the supplementary material you find Python starter-code for loading and visualizing the data from the text files. Moreover, you find an unfinished ToyDataset class, that you are supposed to implement following to the guidelines of the PyTorch documentation[2]. Also provide plots of the visualized datasets (train-sets are sufficient).

b) Define a network architecture. To make life easier you should use the torch.nn package and keep the network architecture as simple as possible. You should not have to worry about GPU training in this question! Provide at least two clearly different network architectures in your code.

[1] PyTorch v0.3.0 has just been released, so in case you are still using v0.2.0 you might want to update

[2] `http://pytorch.org/docs/0.3.0/data.html`

c) Setup the optimization problem and the corresponding solver making use of torch.nn.loss and torch.optim. Since this is a classification problem you should use an appropriate loss function. State clearly what criterion you are using!

d) Use your implementation of the ToyDataset class from a) together with the torch.utils.data.DataLoader in order to feed the data the optimizer of your choice. Use mini batch gradient descent with an appropriate batch size (tune yourself). Run your training for about 100 epochs (one epoch equals one pass over the training dataset). A learning rate of 0.01 is a good starting point. If unsure, check tutorials to guide you in your choice of hyperparameters. Report loss and accuracy(percentage of correct labels) for every 10th epoch of your training on train- *and* test-set.

**Notes**

- Your code should contain sufficient documentation and your answers should be clearly referenced.
- Jupyter Notebooks[3] provide a convenient interface to work on the coding assignment, document and submit your final solution.
- The 'triple junction' dataset is probably easier to learn than the 'spiral' one.

**Question 4: A CNN for MNIST** **(2P)**

Run the PyTorch example in order to train a convolutional network to recognizing handwritten digits from the MNIST dataset: `https://github.com/pytorch/examples/blob/master/mnist/main.py`

We want to learn what kind of mistakes the convolutional network is making. Plot a confusion matrix of the results on the test-set. You may have a look at the function *confusion_matrix*[4] from the *scikit-learn* library.

**Optional** The number of wrongly classified samples on the test-set is pretty small. Lets plot all of them in a single figure using multiple subplots. Moreover, annotate each sample inside the figure with the predicted label as well as the softmax probability of the predicted label.

[3] `http://jupyter.org/`

[4] `http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html`

---

*Note: Submit exactly one ZIP file and one PDF file via Moodle before the deadline. The ZIP file should contain your executable code. Make sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends and captions.*