Prof. Dr. Carsten Meyer
Faculty of Engineering
Christian-Albrechts-Universität Kiel

Neural Networks and Deep Learning – Summer Term 2019

Exercise sheet 4

Submission due: Tuesday, June 04, 13:15 sharp

Remark:

Some of the following experiments are performed on the MNIST data set. The data are contained in the file **`mnist.pkl.gz`** and are loaded using the module **`mnist_loader.py`**. Note that the data are normalized to the range [0, 1].

Generally, the data are divided into a training set (first 60000 samples) and a test set (last 10000 samples). Here, however, we additionally use a validation set of 10000 samples, so we use the first 50000 data samples for training, the next 10000 data samples for validation and the last 10000 samples for testing.

In order to verify the distribution of patterns to classes, the following python code can be executed after loading the data, i.e. after defining the variables **`training_target`**, **`validation_target`** and **`test_target`**:

```
for i in range(10):
    print("%d: %d" % (i, (training_target==i).sum()))

for i in range(10):
    print("%d: %d" % (i, (validation_target==i).sum()))

for i in range(10):
    print("%d: %d" % (i, (test_target==i).sum()))
```

The output is summarized in the following table:

	0	1	2	3	4	5	6	7	8	9
train	4932	5678	4968	5101	4859	4506	4951	5175	4842	4988
valid.	991	1064	990	1030	983	915	967	1090	1009	961
test	980	1135	1032	1010	982	892	958	1028	974	1009

Regarding this training / validation / test split, we see that the data are not fully homogeneously distributed over the splits; however, each digit is sufficiently well represented.

Exercise 1 (Learning in neural networks):

a) Explain the following terms related to learning in neural networks:

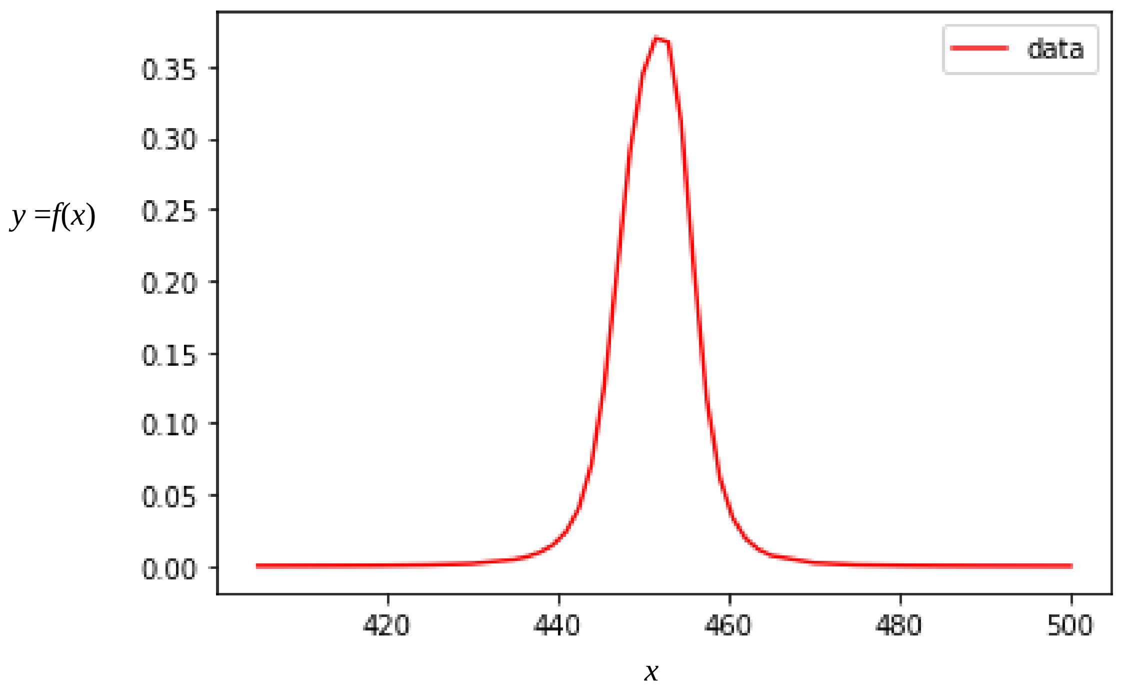
- Loss function
- Stochastic gradient descent
- Mini-batch
- Regularization
- Dropout
- Batch normalization
- Learning with momentum
- Data augmentation
- Unsupervised pre-training / supervised fine-tuning
- Deep learning

b) Name the most important output activation functions $f(z)$, i.e., activation function of the output neuron(s), together with a corresponding suitable loss function L (in both cases, give the mathematical equation). Indicate whether such a perceptron is used for a classification or a regression task.

Exercise 2 (Multi-layer perceptron – regression problem):

The goal of this exercise is to train a multi-layer perceptron to solve a high difficulty level nonlinear regression problem. The data has been generated using an exponential function with the following shape:

Eckerle 4 Dataset



This graph corresponds to the values of a dataset that can be downloaded from the Statistical Reference Dataset of the Information Technology Laboratory of the United States on this link: http://www.itl.nist.gov/div898/strd/nls/data/eckerle4.shtml

This dataset is provided in the file **`Eckerle4.csv`**. Note that this dataset is divided into a training and test corpus comprising 60% and 40% of the data samples, respectively. Moreover, the input and output values are normalized to the interval [0, 1]. Basic code to load the dataset and divide it into a training and test corpus, normalizing the data and to apply a multi-layer perceptron is provided in the python file **`exercise2.py`**.

Choose a suitable network topology (number of hidden layers and hidden neurons) and use it for the multi-layer perceptron defined in the file **`exercise2.py`** (see the lines marked with **`# FIX!!!`**). Try to avoid underfitting and overfitting. Furthermore, define a suitable multi-layer perceptron (again see the lines marked with **`# FIX!!!`**). Define and vary important parameters of the multi-layer perceptron in order to achieve a network performance as good as possible. Report on the found network configuration and on your conclusions.

Remark: You may also use any deep learning framework to solve the exercise, e.g. the “Keras” python library with the “Tensorflow” python deep learning libaray, as suggested in http://gonzalopla.com/deep-learning-nonlinear-regression. This allows e.g. to apply stochastic dropout in order to reduce the risk of overfitting. It is however recommended to use a rather small number of hidden layers.

(Source of exercise: http://gonzalopla.com/deep-learning-nonlinear-regression)

Exercise 3 (Parameters of a multi-layer perceptron – digit recognition):

In the following exercises, we use the neural network library of **`scikit-learn`** to configure, train and apply a multi-layer perceptron to the problem of recognizing handwritten digits (the famous "MNIST" problem). The MNIST data are provided in **`mnist.pkl.gz`**; they can be loaded using the script **`mnist_loader.py`**.

Perform experiments on this pattern recognition problem trying to investigate the influence of a number of parameters on the classification performance, namely

a) the learning rate,
b) the number of hidden neurons (in a network with a single hidden layer),
c) the number of hidden layers,
d) the solver and learning rate schedule used in optimization,
e) the activation function.

(Further aspects as the loss function and the influence of regularization are addressed in Exercise 4). The script **`exercise3.py`** can serve as a basis or starting point.

Report on and summarize your findings.

Further investigations and experiments as well as code extensions and modifications are welcome!

Exercise 4 (Vanishing gradient):

a) The script **`exercise4.py`** implements a multi-layer perceptron for use on the MNIST digit classification problem. Apart from the training loss, it also displays a histogram of the weights (between the input and the first hidden layer) after 1 training epoch and at the end of the training, and visualizes the weights (between the input layer and 16 hidden neurons of the first hidden layer). Using a logistic activation function, compare the output for a single hidden layer, four and five hidden layers. Then change to a ReLU activation function and inspect the results for five hidden layers. Discuss your findings.

b) Starting from your analysis for the multi-layer perceptron with five hidden layers and ReLU activation function, modify parameters such that the training becomes successful.

c) Give a theoretical justification, why the weights and biases of neurons in the first hidden layers in a multi-layer perceptron with many hidden layers are modified only slowly when using a sigmoid activation function and gradient descent. To this end, consider – as an example – a simplified network with three hidden layers (and a single neuron per layer), compute and analyse the change Δb_1 of the bias of the first hidden neuron with respect to a change in the cost function C. What changes in your analysis when using a ReLU activation function instead of a sigmoid?

(In case of an errror message just re-run the script.)