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

Neural Networks and Deep Learning – Summer Term 2020

Exercise sheet 4

Submission due: Wednesday, June 03, 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):

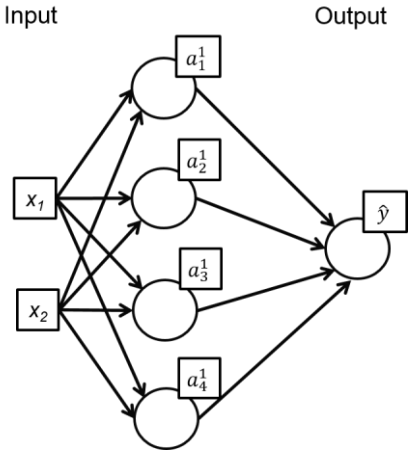
a) Explain the following terms related to neural networks as short and precise as possible:

- 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: Backprogagation, regression problem):

a) Consider the multi-layer perceptron in the following figure:



The activation function at all hidden nodes is ReLU and at the output node linear.

Perform one iteration of plain backpropagation (without momentum, regularization etc.), based on a mini-batch composed of two input samples $\mathbf{x}^{(\mu)}$ with corresponding target values $y^{(\mu)}$, learning rate $\eta = 0.5$ and SSE loss:

$\mathbf{x}^{(1)} = (-1, 1)^T$ with target $y^{(1)} = 1$ and $\mathbf{x}^{(2)} = (2, -1)^T$ with target $y^{(2)} = -1$

The initial weights and biases are given as (t is the iteration index):

$$\mathbf{W}^1(t=0) = \begin{pmatrix} 1 & 2 \\ 0 & -1 \\ -1 & -3 \\ -2 & 2 \end{pmatrix}; \; \mathbf{W}^2(t=0) = (1 \quad 0 \quad -1 \quad 2)$$

$$\boldsymbol{b}^1(t=0) = \begin{pmatrix} -2 \\ 2 \\ 0 \\ -2 \end{pmatrix}; \; b^2(t=0) = -2$$

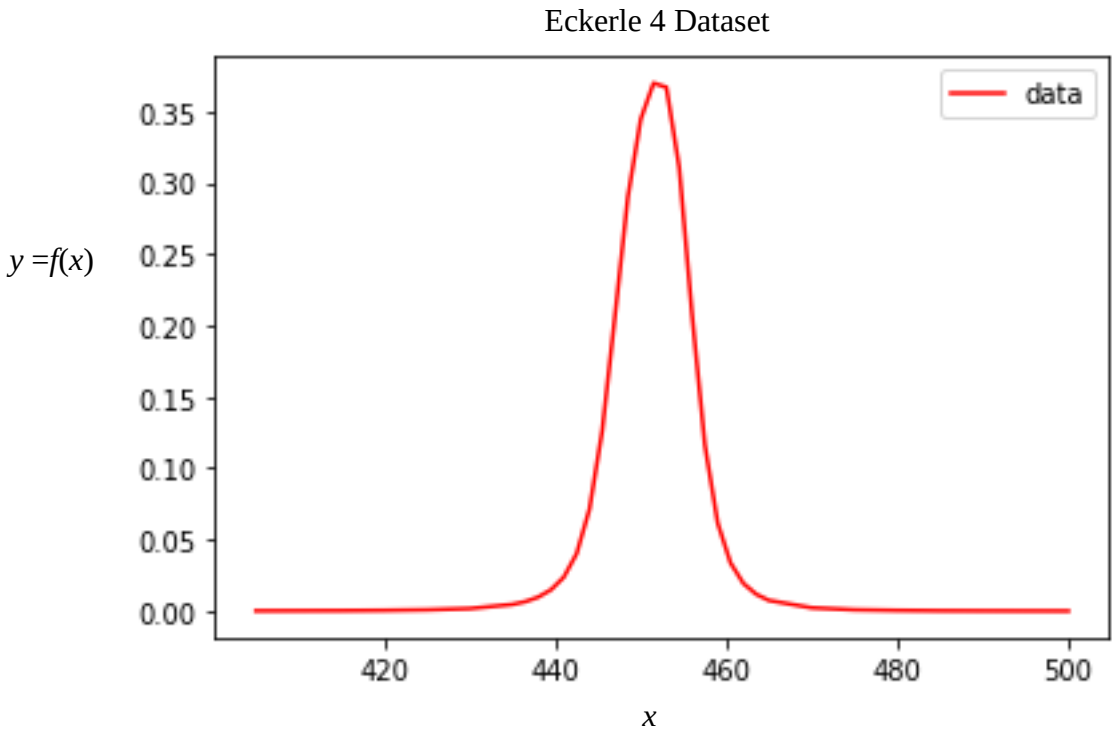
For the forward path, calculate the postsynaptic potential (PSP), the activations and outputs and insert them into the following table:

Input $\boldsymbol{x} = (x_1, x_2)^T = \boldsymbol{a}^0$	PSP $\boldsymbol{z}^1$	Activation $\boldsymbol{a}^1$	Output $\hat{y} = a^2$
$(-1, 1)^T$			
$(2, -1)^T$			

For the backward path, calculate the updated weights and biases for the hidden and output layer and insert them into the following table:

Weights $\mathbf{W}^1(t=1)$	Bias $\boldsymbol{b}^1(t=1)$	Weights $\mathbf{W}^2(t=1)$	Bias $\boldsymbol{b}^2(t=1)$

b) 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:



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 Jupyter notebook.

Choose a suitable network topology (number of hidden layers and hidden neurons, potentially include dropout, activation function of hidden layers) and use it for the multi-layer perceptron defined in the Jupyter notebook. Set further parameters (learning rate, loss function, optimizer, number of epochs, batch size; see the lines marked with **`# FIX!!!`** in the Jupyter notebook). Try to avoid underfitting and overfitting. Vary the network and parameter configuration in order to achieve a network performance as optimal as possible. For each network configuration, due to the random components in the experiment, perform (at least) 4 different training and evaluation runs and report the mean and standard deviation of the training and evaluation results. Report on your results and conclusions.

(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 Tensorflow and Keras to configure, train and apply a multi-layer perceptron to the problem of recognizing handwritten digits (the famous "MNIST" problem). The MNIST data are loaded using a Tensorflow Keras built-in function.

Perform experiments on this pattern recognition problem trying to investigate the influence of a number of parameters on the classification performance. This may refer to

- the learning rate and potentially learning schedule,
- the number of hidden neurons (in a network with a single hidden layer),
- the number of hidden layers as well as applying dropout and / or batch normalization,
- the solver (including momentum),
- the activation function at hidden layers,
- regularization.

The script in the Jupyter notebook can serve as a basis or starting point.

Report your findings and conclusions.

Note: These experiments may require a lot of computation time!

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

Exercise 4 (Vanishing gradient):

a) The Jupyter notebook implements a multi-layer perceptron for use on the MNIST digit classification problem. Apart from the training loss and accuracy, it also displays a histogram of the weights (between the input and the first hidden layer) after initialization 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 sigmoid activation function, compare the output for a single hidden layer, five and six hidden layers. Then change to a ReLU activation function and inspect the results for six hidden layers. Discuss your findings.

b) 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?

c) Starting from your analysis for the multi-layer perceptron with six hidden layers and sigmoid activation function in part a), try to find other model configurations which lead to a successful training. You may modify e.g. the learning rate and batch size, the weight and bias initialization, apply batch normalization and / or dropout, and add regularization.