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

Neural Networks and Deep Learning – Summer Term 2020

Exercise sheet 3

Submission due: Wednesday, May 20, 13:15 sharp

<u>Note:</u>

Some of the following exercises are based on the Keras and TensorFlow libraries. TensorFlow is an open source platform for machine learning, supporting matrix (tensor) multiplications on graphical processing units (GPUs) and therefore ideally suited for (deep) neural networks; for more information see https://www.tensorflow.org/. Keras is a high-level neural networks application programming interface (API) running on top of TensorFlow or other libraries like Theano or CNTK with the aim to facilitate using those libraries; see https://keras.io/.

Python code to configure neural networks using Keras and Tensorflow has been provided in form of a Jupyter notebook, **`Sheet_3.ipynb`**; for documentation on Jupyter and Jupyter notebooks see https://jupyter.org/. **`Sheet_3.ipynb`** needs additional resources like images etc.; all necessary files are contained in a git repository in the directory **`Lab3`**. In order to copy those files to your computer, you have to install **`git`** on your computer; see e.g. https://git-scm.com/download/win.

One way to copy the required lab files to your computer is to use a git bash shell: On Windows, open an explorer and navigate to a folder where you want to copy all relevant lab files. Then (after installing an appropriate git package), right click the mouse in that directory and select the option “Git Bash Here”. A git bash shell opens; in that shell, enter the following command:

```
git clone https://gitlab+deploy-token-26:XBza882znMmexaQSpjad@git.informatik.uni-kiel.de/las/nndl.git <your target directory>
```

where **`<your target directory>`** has to be replaced by the path of the directory on your local computer that you want to generate and which should contain the downloaded lab files. After cloning, this directory then should contain a folder **`Lab3`** with the file **`Sheet_3.ipynb`** and other text files.

For the python code involving Keras and TensorFlow to work, there are two possibilities:

a) (recommended) Execute the code in Google Colab (https://colab.research.google.com/notebooks/intro.ipynb). To this end, you need to create a Google account. Log in and open https://colab.research.google.com/. In the upper right corner, select the "Upload" tab and then "Browse". Locate the cloned **`Lab3`** directory and select the file **`Sheet_3.ipynb`**. Note that you have to run the first cell (containing **`! git clone`** ...) before running any other cell; otherwise you may get an error message from subsequent cells like
`OSError: nndl/Lab3/exercise3b_input.txt not found.`

   Note that with this option external images are not displayed correctly; instead, you only see e.g. the following line:
   `IMAGE: perceptron`
   For external images to be displayed correctly, option b) has to be invoked (or refer to the pdf exercise sheet).

b) Run the notebook locally. In order for the Jupyter notebook to work properly, also the files and the subdirectory of **`Lab3`** have to be loaded into Jupyter notebook. A possibility for that is to invoke Jupyter notebook from an Anaconda installation (see https://www.anaconda.com/) via an Anaconda Prompt (NOT directly selecting the Jupyter Notebook option): Open an Anaconda prompt, navigate to the **`Lab3`** directory just cloned, and then enter
`jupyter-notebook`
into the Anaconda prompt. This opens Jupyter notebook *including* all necessary files. Then, click on **`Sheet_3.ipynb`**; now you can start to work with the Jupyter notebook (and should see external images, which are contained in the **`images`** subdirectory, properly). If you want to execute the cells involving Keras and TensorFlow locally on your computer, you have to install Keras and TensorFlow locally and to configure your environment appropriately.

Run one cell after the other; this should finally provide you with some plots being generated.

Note that changes in **`Sheet_3.ipynb`** which are saved in the colab environment are *not* saved to **`Lab3/Sheet_3.ipynb`**!

Further documentation how to install Jupyter notebooks can be found at:

https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html
https://jupyter.readthedocs.io/en/latest/install.html#install

Exercise 1 (Learning in neural networks):

a) Explain the following terms related to neural networks:

- Learning in neural networks
- Training set
- Supervised learning
- Unsupervised learning
- Online (incremental) learning
- Offline (batch) learning
- Training error
- Generalisation error
- Overfitting
- Cross-validation

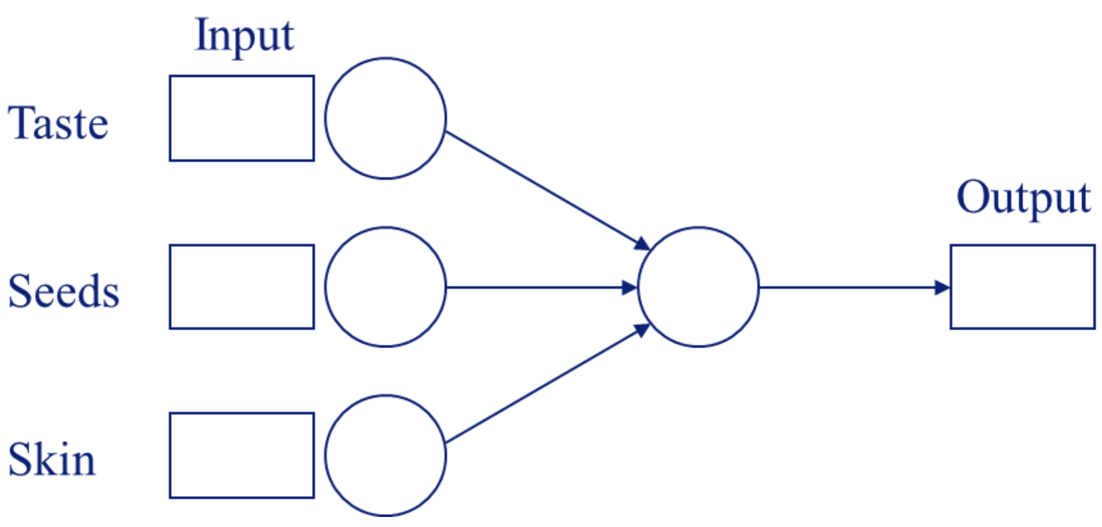
b) Name and briefly describe some methods to indicate or avoid overfitting when training neural networks.

Exercise 2 (Perceptron learning – analytical calculation):

The goal of this exercise is to train a single-layer perceptron (threshold element) to classify whether a fruit presented to the perceptron is going to be liked by a certain person or not, based on three features attributed to the presented fruit: its taste (whether it is sweet or not), its seeds (whether they are edible or not) and its skin (whether it is edible or not). This generates the following table for the inputs and the target output of the perceptron:

Fruit	Perceptron input (features of the fruit)			Target output person likes = 1, doesn't like = 0
	Taste sweet = 1, not sweet = 0	Seeds edible = 1, not edible = 0	Skin edible = 1, not edible = 0	
Banana	1	1	0	1
Pear	1	0	1	1
Lemon	0	0	0	0
Strawberry	1	1	1	1
Green apple	0	0	1	0

Since there are three (binary) input values (taste, seeds and skin) and one (binary) target output, we will construct a single-layer perceptron with three inputs and one output.



Since the target output is binary, we will use the perceptron learning algorithm to construct the weights.

To start the perceptron learning algorithm, we have to initialize the weights and the threshold. Since we have no prior knowledge on the solution, we will assume that all weights are 0 ($w_1 = w_2 = w_3 = 0$) and that the threshold is $\theta = 1$ (i.e. $w_0 = -\theta = -1$). Furthermore, we have to specify the learning rate η. Since we want it to be large enough that learning happens in a reasonable amount of time, but small enough so that it doesn't go too fast, we set $\eta = 0.25$.

Apply the perceptron learning algorithm – in the incremental mode – analytically to this problem, i.e. calculate the new weights and threshold after successively presenting a banana, pear, lemon, strawberry and a green apple to the network (in this order).

Draw a diagram of the final perceptron indicating the weight and threshold parameters and verify that the final perceptron classifies all training examples correctly.

Note: The iteration of the perceptron learning algorithm is easily accomplished by filling in the following table for each iteration of the learning algorithm:

$\mu = 1$; current training sample: banana						
Input $\mathbf{x}^{(\mu)}$	Current weights $\mathbf{w}(t)$	Network Output $y^{(\mu)}$	Target output $d^{(\mu)}$	Learning rate η	Weight update $\Delta\mathbf{w}(t)$	New Weights $\mathbf{w}(t+1)$
$x_0 = 1$	$w_0 =$					
$x_1 =$	$w_1 =$			0.25		
$x_2 =$	$w_2 =$					
$x_3 =$	$w_3 =$					

(Source of exercise: Langston, Cognitive Psychology)

Exercise 3 (Single-layer perceptron, gradient learning, 2dim. classification):

The goal of this exercise is to solve a two-dimensional binary classification problem with gradient learning, using TensorFlow. Since the problem is two-dimensional, the perceptron has 2 inputs. Since the classification problem is binary, there is one output. The (two-dimensional) inputs for training are provided in the file **`exercise3b_input.txt`**, the corresponding (1-dimensional) targets in the file **`exercise3b_target.txt`**. To visualize the results, the training samples corresponding to class 1 (output label "0") have separately been saved in the file **`exercise3b_class1.txt`**, the training samples corresponding to class 2 (output label "1") in the file **`exercise3b_class2.txt`**. The gradient learning algorithm – using the **`sigmoid`** activation function – shall be used to provide a solution to this classification problem. Note that due to the **`sigmoid`** activation function, the output of the perceptron is a real value in [0,1]:

$$sigmoid(h) = \frac{1}{1+e^{-h}}$$

To assign a binary class label (either 0 or 1) to an input example, the perceptron output y can be passed through the Heaviside function $\Theta[\, y - 0.5 \,]$ to yield a binary output y^{binary}. Then, any perceptron output between 0.5 and 1 is closer to 1 than to 0 and will be assigned the class label "1". Conversely, any perceptron output between 0 and <0.5 is closer to 0 than to 1 and will be assigned the class label "0". As usual, denote the weights of the perceptron w_1 and w_2 and the bias $w_0 = -\theta$.

a) Using the above-mentioned post-processing step $\Theta[\, y - 0.5 \,]$ applied to the perceptron output y, show that the decision boundary separating the inputs $\mathbf{x}=(x_1, x_2)$ assigned to class label "1" from those inputs assigned to class label "0" is given by a straight line in two-dimensional space corresponding to the equation (see in python code at # *plot last decision boundary*):

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

b) The classification problem (defined by the training data provided in **`exercise3b_input.txt`** and the targets provided in **`exercise3b_target.txt`**) shall now be solved using the TensorFlow and Keras libraries. The source code is given below and can be executed by clicking the play button (see remarks above).

1. Train the model at least three times and report on your findings.
2. Change appropriate parameters (e.g. the learning rate, the batch size, the choice of the solver, potentially the number of epochs etc.) and again report on your findings.

c) Repeat exercise b) with the training set **`exercise3c_input.txt`** and the targets **`exercise3c_target.txt`**. Those points have been generated from the input points of exercise b) by removing points from class 1 (i.e. those points the x-coordinate of which is below 0.35). Do not forget to modify the variables **`class1`** and **`class2`** to load the files **`exercise3c_class1.txt`** and **`exercise3c_class1.txt`**, respectively! Discuss the output of the training algorithm in terms of the resulting decision boundary and the final training error.

d) Divide the input samples from part b) into separate training and validation sets, where the latter shall comprise 30% of the data. You may use available Keras functionality for this purpose. Run the script at least two times, plot the training and validation loss and accuracy as a function of the epoch number and report on your findings.

e) Modify the script to handle the XOR-problem, i.e. set
```
input = np.array([[0,0],[0,1],[1,0],[1,1]])
target = np.array([0, 1, 1, 0])
```
and plot the final decision boundary and the loss function. Report on your findings.

Useful information about training and evaluation with Tensorflow and Keras can be found at https://www.tensorflow.org/guide/keras/train_and_evaluate

Exercise 4 (Multi-layer perceptron and backpropagation – small datasets):

The goal of this exercise is to apply a multi-layer perceptron (MLP), trained with the backpropagation algorithm as provided by Tensorflow Keras library, to four classification problems provided by the UCI repository (and contained in the scikit learn package; i.e. iris, digits, wine, breast_cancer) and two artificially generated classification problems (circles, moon). In particular, the influence of the backpropagation solver and of the network topology shall be investigated in parts a) and b) of the exercise, respectively.

a) In this part of the exercise, a number of solvers (stochastic gradient descent, Adam, Adam with Nesterov momentum, AdaDelta, AdaGrad or RMSProp) shall be applied to the six datasets. An (incomplete) python script for this experiment is provided the Jupyter notebook. Complete the code (model definition, selection and configuration of an optimizer and model "compilation" including selection of an appropriate loss function ; see # TO BE ADAPTED in the Jupyter notebook); consult the Tensorflow Keras documentation if needed. Furthermore, select suitable values of the most important parameters (e.g. learning rate, batch size...). Then, apply the script for at least three different optimizers, for a suitable baseline model configuration. Report the final training and validation loss and accuracy values and provide plots for the training and validation loss and accuracy curves as a function of the number of epochs (see script). What are your conclusions regarding the comparison of the optimization strategies? Also report on the database statistics.

   The optimizer is selected e.g. with
   ```
   opt = SGD(learning_rate=lr) # SGD or Adam, Nadam, Adadelta, Adagrad,
   RMSProp
   ```
   Note that additional parameters of the optimizers can be set if desired (see the Tensorflow Keras documentation).

b) Using the most successful optimizer from part a), in this part of the exercise different network topologies shall be investigated, i.e. the number of hidden layers and of hidden neurons shall be varied. To this end, modify the python script accordingly and systematically test the network performance. Provide the final training and validation loss and accuracy and provide the loss and accuracy curves as function of the number of epochs. You may also test further parameter settings. What are your conclusions regarding the network topology?