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

Neural Networks and Deep Learning – Summer Term 2018

Exercise sheet 3

Submission due: Tuesday, May 22, 11:30 sharp

Note:

Some of the following exercises are based on the "scikit learn" python library, a set of python modules for machine learning and data mining (see www.scikits.appspot.com/learn).

This can easily be installed using the Anaconda package (scikit-learn is already included in the Anaconda distribution):

1.) Download Anaconda from (caution: download takes quite a while!)
https://www.continuum.io/downloads
(Python 3.5) → Anaconda3-4.0.0-Windows-x86_64.exe

2.) Install Anaconda by double-clicking the executable

Remark:
A **S**cientific **PY**thon **D**evelopment **E**nvi**R**onment ("Spyder") is included in the Anaconda package.

3.) From the start menu, start an Jupyter QTconsole (Anaconda3 → Jupyter QTConsole)

Further documentation on SciKit can be found at:

http://scikit-learn.org/stable/install.html

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

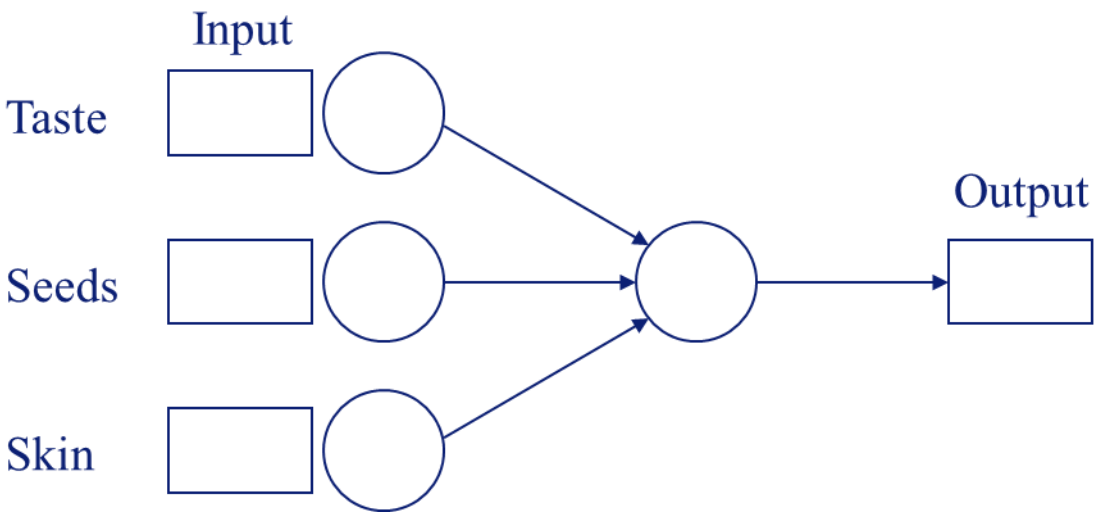
b) Name and briefly describe some methods to indicate or avoid overfitting when training neural networks.

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:

μ =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 $\mathbf{w}(t+1)$
$x_0 = 1$	$w_0 =$			0.25		
$x_1 =$	$w_1 =$					
$x_2 =$	$w_2 =$					
$x_3 =$	$w_3 =$					

(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 the sklearn python library. 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`**. 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 **`logistic()`** activation function – shall be used to provide a solution to this classification problem. Note that due to the **`logistic()`** activation function, the output of the perceptron is a real value in [0,1]:

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

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" (see also the scikit learn documentation, "Neural network models (supervised), Mathematical formulation)". 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 lines 83 and 108 in file **`exercise3b.py`**):

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

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 scikit learn python library. Corresponding software has been provided in the file **`exercise3b.py`**. Run the file by typing
`run exercise4b.py`
in a python console at least three times and report on your findings. 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.

Note: By setting
`hidden_layer_sizes=()`
no hidden layers are generated, realising a single-layer perceptron. The number of layers is 2 (scikit learn counts input and output as two layers, instead of counting the number of learnable weight layers). Training is invoked with the "**`fit`**" method.

The weights can be accessed by **net.coefs_**, the biases by **net.intercepts_** (both are lists of numpy arrays, so several indices are needed to access individual weights).
In scikit learn, a single-layer perceptron can also be realized as perceptron linear model:
```
from sklearn.linear_model import perceptron
```
Here, however, the activation function is always linear.

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 into a separate training and a test set. To this end, you may use the **train_test_split** function of sklearn. To use this function, the following line has to be included in the python file:
```
from sklearn.model_selection import train_test_split
```
Adapt the script from part b) of this exercise to perform a training on the training corpus and a test on the training and the test corpus created by **train_test_split**. The evaluation of the model can be performed using the **score** function, e.g.
```
net.score(X_train, y_train)
net.score(X_test, y_test)
```
Plot the training and test score (accuracy) as a function of the iteration number. Run the script at least two times and report on your findings.

e) Modify the script **exercise3b.py** 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.

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 the scikit learn python library, to four classification problems provided by the UCI repository (and contained in the scikit learn package; i.e. iris, digits, wine, diabetes) 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, seven solvers (stochastic gradient descent with constant or adapted learning rate and / or momentum or not, adam optimizer) shall be applied to the six datasets. A python script for this experiment is provided in **`exercise4a.py`**. Find out the values of the most important parameters (number of hidden layers and of hidden neurons, activation function, batch size, learning rate, momentum ...) as provided by the script. Apply the script and record the values of the training loss, training score and test score. You may also test different values for important parameters. What are your conclusions regarding the comparison of the solver strategies and the learning success? Report on the database statistics.

b) Using the most successful solvers from part a), namely the "constant with Nesterov's momentum" and the "adam" solver, 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. A corresponding python script for this experiment is provided in **`exercise4b.py`**. Run the script, record the training loss, training score and test score and report on your conclusions regarding the network topology. You may also test further parameter settings.