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

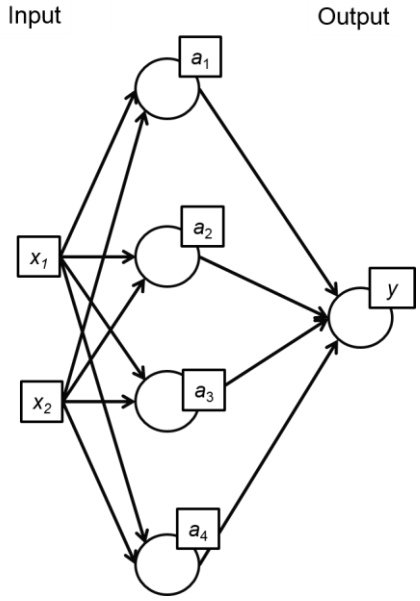
Neural Networks and Deep Learning – Summer Term 2019

Exercise sheet 6

Submission due: Wednesday, July 03, 13:15 sharp

Exercise 1 (Radial basis functions):

For this exercise, the following network architecture shall be given:



a) Consider a radial basis function network with the architecture above. The 4 radial basis functions ϕ_1, ϕ_2, ϕ_3 and ϕ_4 are defined as follows: The centers $\mathbf{c}_j = (c_{j,(1)}, c_{j,(2)})^T$ of the radial basis functions are given by (T denotes the transpose of a matrix or vector):
$\mathbf{c}_1 = (0.5, 0.5)^T$
$\mathbf{c}_2 = (-0.5, 0.5)^T$
$\mathbf{c}_3 = (-0.5, -0.5)^T$
$\mathbf{c}_4 = (0.5, -0.5)^T$
The radial basis functions ϕ_j themselves are Heaviside-functions with radius $r_j = 0.5$ for j = 1,2,3,4. In particular, we define (where ||…|| denotes the Euclidean distance):
$a_j = \phi_j(r_j) = \Theta[0.5 - r_j]$ with $r_j = \|\mathbf{x} - \mathbf{c}_j\|$ for j = 1,2,3,4 ; $\mathbf{x} = (x_1, x_2)^T$.

This means that the radial basis function ϕ_j at node a_i is a circle around the corresponding center $\mathbf{c}_j$ with radius 0.5 and value 1 within the circle and value 0 outside of the circle.
The synaptic weights between the input node x_i and the hidden node a_j are $w_{ji} = c_{j,(i)}$ for all i and all j, since they correspond to the centers of the radial basis functions. Let the synaptic weights v_j between hidden node a_i and output node y be given as:

$$\mathbf{V} = (v_j) = (1 \quad 2 \quad 3 \quad 4)^T$$

The threshold of the output unit is given by $\theta = 0$ (note that the output unit is a *linear* element!).

i) Calculate the activations a_i at the hidden nodes and the output y for the following four input pairs and insert the results into the table:

Inputs	distance r_i at hidden nodes	RBF values a_i	PSP z at output node	Output y
$x_1 = 0.5$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.8$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		
$x_1 = -0.2$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.5$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		

ii) For real input values x_1 and x_2, draw a plot of the two-dimensional plane indicating the output y for each input pair (x_1, x_2) and discuss how the radial basis function calculates the output for each input in the two-dimensional plane.

b) For comparison, now consider a multi-layer perceptron with the architecture above. Let the units a_1, a_2, a_3 and a_4 be threshold elements, i.e. their output is given by $a_j = \Theta[z_j]$, where Θ denotes the Heaviside function and z_j $(j = 1, \ldots, 4)$ denotes the postsynaptic potential (computed from the inputs x_i as usual, i.e. including the threshold). Let the threshold of all hidden units be $\theta = 0.5$. Let also the output node be a threshold element, i.e. the output y is given by $y = \Theta[z]$ with the postsynaptic potential z and threshold $\theta = 0.5$. Denote the synaptic weights between the input node x_j and the hidden node z_i with w_{ij}, and the synaptic weights between the hidden node z_i and the output node y with v_i. The synaptic weight matrix $\mathbf{W}$ between the inputs and the hidden nodes and the synaptic weight vector $\mathbf{V}$ between the hidden nodes and the output have the same values as given in the previous part of the exercise.

i) Calculate the values a_j at the hidden nodes and the output y for the following two input pairs and insert the results into the table

Inputs	PSP z_i at hidden nodes	activations a_j at hidden nodes	PSP z at output node	Output y
$x_1 = 0.5$	$z_1 =$	$a_1 =$	$z =$	$y =$
	$z_2 =$	$a_2 =$		
$x_2 = 0.8$	$z_3 =$	$a_3 =$		
	$z_4 =$	$a_4 =$		
$x_1 = -0.2$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.5$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		

ii) For real input values x_1 and x_2, draw a plot of the two-dimensional plane indicating the output y for each input pair (x_1, x_2) and discuss how the multi-layer perceptron calculates the output for each input in the two-dimensional plane.

Exercise 2 (Simple recurrent neural network for character sequences):

The python script **`exercise2.py`**, which is based on the script **`min-char-rnn.py`** written by Andrej Karpathy (see https://gist.github.com/karpathy/d4dee566867f8291f086), implements a simple recurrent neural network applied to predict the next character in a sequence of characters, based on the current character (see also the lecture slides).

a) Depict a diagram of the recurrent neural network (including the number of neurons in each layer, the activation functions, the update equations and the dimensions of the matrices and vectors involved). Describe in detail how the network processes a sequence of characters in training (what is input, what is output, unfolding) and how the network generates a sequence of characters. Run the script (on the input file **`input_short.txt`**) and describe the output.

b) Run the script 10 times (you may modify the python script correspondingly) and report on your finding. Vary the configuration of the network and important parameters and assess the network performance based on 10 runs of each modified configuration for the input file **`input_short.txt`**. Report on your findings and conclusions.

Exercise 3 (Long Short Term Memory LSTM, time series prediction):

a) Consider an LSTM network with a single hidden layer composed of two LSTM units, i.e. #hidden = 2, which receives three-dimensional inputs $\mathbf{x} = (x_1, x_2, x_3)^T$, i.e. the number of input features is #features = 3.

The synaptic weight matrices are given as follows:

Input gate:

$\mathbf{W}_{xi} = \begin{pmatrix} 2 & 0 & 3 \\ 4 & 1 & 0 \end{pmatrix}$ weights from input to input gate (#hidden × #features)

$\mathbf{W}_{hi} = \begin{pmatrix} -3 & 0 \\ -5 & 1 \end{pmatrix}$ weights from hidden units to input gate (#hidden × #hidden)

$\boldsymbol{b}_i = \begin{pmatrix} 0 \\ -2 \end{pmatrix}$ bias of input gate (vector of dimension #hidden)

Forget gate:

$\mathbf{W}_{xf} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{pmatrix}$ weights from input to forget gate (#hidden × #features)

$\mathbf{W}_{hf} = \begin{pmatrix} -2 & 0 \\ 2 & 1 \end{pmatrix}$ weights from hidden units to forget gate (#hidden × #hidden)

$\boldsymbol{b}_f = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ bias of forget gate (vector of dimension #hidden)

Cell ("gate gate"):

$\mathbf{W}_{xg} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 2 & 1 \end{pmatrix}$ weights from input to "gate" gate (#hidden × #features)

$\mathbf{W}_{hg} = \begin{pmatrix} 2 & 0 \\ -1 & 1 \end{pmatrix}$ weights from hidden units to "gate" gate (#hidden × #hidden)

$\boldsymbol{b}_g = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ bias of "gate" gate (vector of dimension #hidden)

Output gate:

$\mathbf{W}_{xo} = \begin{pmatrix} -1 & 2 & -2 \\ 0 & 1 & 1 \end{pmatrix}$ weights from input to output gate (#hidden × #features)

$\mathbf{W}_{ho} = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$ weights from hidden units to output gate (#hidden × #hidden)

$\boldsymbol{b}_o = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ bias of output gate (vector of dimension #hidden)

Both the hidden layer activations and the cell activations shall be initialized with 0:

$$\boldsymbol{a}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} .$$

Further, there are four output units, i.e. #output = 4. The synaptic weight matrix between the hidden units and the outputs is given by

$$\mathbf{W}_{yh} = \begin{pmatrix} 2 & -1 \\ 0 & 3 \\ 1 & -1 \\ 4 & 0 \end{pmatrix}$$ weights from hidden unit activations to output (#output × #hidden)

$$\boldsymbol{b}_y = \begin{pmatrix} 2 \\ 0 \\ -1 \\ 1 \end{pmatrix}$$ bias of outputs (vector of dimension #output)

For the output units, a ReLU activation function is used.

Calculate the outputs in a many-to-many scenario, i.e. an output is calculated for each input vector, at the time steps $t = 1$ and $t = 2$ for the input sequence

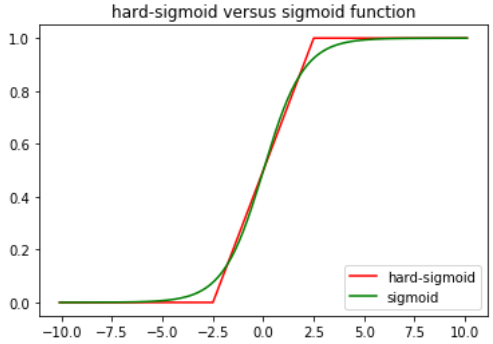
$$\boldsymbol{x}_1 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \qquad \boldsymbol{x}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

Optional: Draw a diagram of the LSTM network.

Note: Instead of the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ the so-called "hard-sigmoid" function σ_{hard} is being used at the input, forget and output gate, which is defined as follows:

$$\sigma_{hard}(z) = max\{0, min\{1, 0.2 * x + 0.5\}\}$$

$$\text{i.e. } \sigma_{hard}(z) = \begin{cases} 0 & for\ x < -2.5 \\ 0.2 * x + 0.5 & for -2.5 \leq x \leq 2.5 \\ 1 & for\ x > 2.5 \end{cases}$$



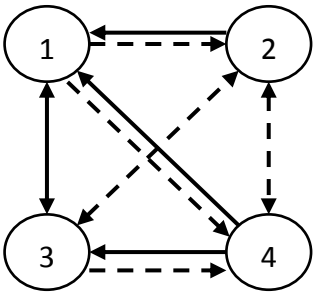
The script **`exercise3a.py`** demonstrates how this network can be implemented in Keras. You may use the script to check your solution.

b) The file **`exercise3b.py`** (downloaded from https://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/) implements a simple LSTM network to predict shampoo sales over a period of three years. Describe the LSTM network, the way how the network processes the time sequence to predict future shampoo sales (how many inputs of which dimension, how many outputs of what dimension?), and how training is performed (how many samples, what is the length of the sequence considered in truncated backpropagation through time?). Vary important parameters (repeating over several training runs) and comment on your findings.

Note that to modify other parameters, certain conditions must hold, e.g. the batch size must be a factor of the number of training samples; but then, other architectural changes of the LSTM network have to be applied. This is beyond this small introductory exercise...

Exercise 4 (Output calculation in a fully connected recurrent network):

In this exercise, the dynamics in a simple fully connected recurrent neural network shall be calculated analytically. Consider the following recurrent neural network with asymmetric connections:



Asymmetric connections;
Solid line: $w_{ij} = +1$
Broken line: $w_{ij} = -1$

weight matrix and thresholds:

$$W_{ij} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix} \; ; \quad \theta_i = \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}$$

The neurons are assumed to have binary states $s \in \{0; 1\}$ and initial state $\mathbf{s}^0 = \mathbf{s}(t=0) = (s_1, s_2, s_3, s_4) = (0, 0, 0, 0)$.

a) Calculate the network output up to time step $t = 5$ in parallel (synchronous) dynamics. Give an interpretation of your result.

b) Calculate the network output up to time step $t = 8$ in asynchronous dynamics, assuming fixed order neuron update $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Give an interpretation of your result.

Exercise 5 (Competitive network, Winner takes all algorithm, classification, digit recognition):

a) Consider a competitive network (using Euclidean distance) with two input neurons x_1 and x_2, three output neurons y_1, y_2 and y_3 and weight matrix

$$\mathbf{W} = \begin{pmatrix} 1/2 & 2/3 \\ 1/3 & 1/6 \\ 2/3 & 1/3 \end{pmatrix}$$

Compute the output for the following inputs:

$\mathbf{x} = (x_1, x_2) = (1/2, 1/3)$

$\mathbf{x} = (x_1, x_2) = (1/3, 2/3)$

(Source of exercise: Schwarz, FH Zwickau)

b) In the remaining parts of this exercise, a competitive network – trained with the "winner takes all" algorithm, is used to solve a classification problem. The competitive network is defined by an input and an output layer. Note that there is no bias (threshold) in the network. Further note that the training is unsupervised, so no target values are provided to the algorithm. There is also no learning rate.

Generate input samples by running the python script **`exercise5b_generate_input_samples.py`**. Then define the network and perform network training by running the script **`exercise5b.py`** (source of script: internet neurolab documentation). Provide a screenshot of the training and the values of the final weights of the output neurons. Give an interpretation of the training process by answering the following questions:

- What is plotted in the figure, i.e. what does "real centers" and "train centers" mean?
- Give an interpretation of the training outcome, i.e. of the final network weights.
- Compare the final weight vectors to the "real centers" of the data clouds (see the script **`exercise5b_generate_input_samples.py`**).

Note:

This exercise is based on the neural network library "neurolab" for python. In case neurolab is not installed on your computer, please install it as follows (version 3, using Anaconda package):

1.) In an Anaconda Prompt, type

    pip install neurolab

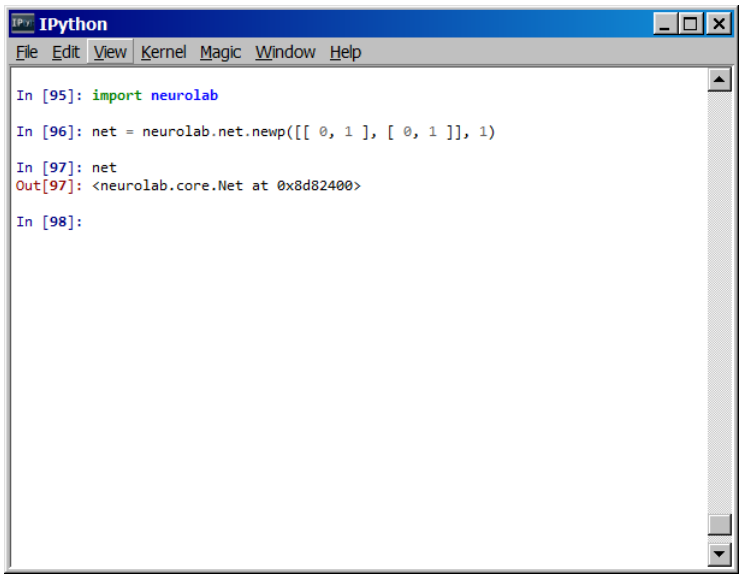
2.) From the start menu, start an Jupyter QTconsole (Anaconda3 → Jupyter QTConsole)

    Test:

    After that, you should be able to issue the following commands in the JupyterQTConsole without an error message (see figure below):

    import neurolab
    net = neurolab.net.newp([[0, 1], [0, 1]], 1)

    If there is no error message and you can access the net object, the neurolab installation was successful.



Further documentation on Neurolab installation can be found at:
http://pythonhosted.org/neurolab/install.html

Note:

Neurolab is similar to the Matlab neural network toolbox (but seems to have restricted functionality compared to the Matlab neural network toolbox – so not everything from Matlab is available in Neurolab, and the syntax is a bit different).

A tutorial on the Matlab neural network toolbox is available at
http://www.znu.ac.ir/data/members/fazli_saeid/ANN/matlab4.pdf

c) In this part of the exercise, the task is to teach a competitive network to classify digits between 0 and 9. The digit training data are provided in the file **`digits.pat`** as 7×6 matrix of binary pixel values. The file also contains the correct classification result in a 1-out-of-10 coding. The input data has been transformed to row format for a 1-dimensional arrangement of input neurons in the file **`digits_input.txt`** (since the training of the competitive network is unsupervised, the target results are not needed for training). (Data **`digits.pat`** taken from Schwenker, Ulm University).

i. Train a competitive network to solve the classification task (starting from the script **`exercise5c.py`**). How many input neurons and how many output neurons are needed? Provide the correct numbers in the script (see question marks **`???`** and keyword **`# FIX!!!`** in the script), run the script and provide a screenshot of the training error. In addition, provide the training results (written to output by the script below "**`test on training set:`**") and give an interpretation of the results.

ii. From the training set **`digits.pat`**, some noisy samples have been generated by flipping 4 bits in each input sample, see file **`digits_noisy.pat`**. The idea is now to test the trained network also on these noisy samples. To this end, outcomment the lines below "**`## noisy test set`**" in the file **`exercise5c.py`**. Run the script and provide the classification results on both the training and the noisy test set. Give an interpretation whether the network has correctly classified also the noisy digit examples.

Note: Each training run may result in a different order of the trained patterns (see part a). Therefore, be careful to perform the test on the training and the noisy test set with the *same* network weights. This is achieved by either performing both tests directly after training, or by performing successive tests with *fixed* network weights without a new training (where the network weights are fixed to the results of the previous training).