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

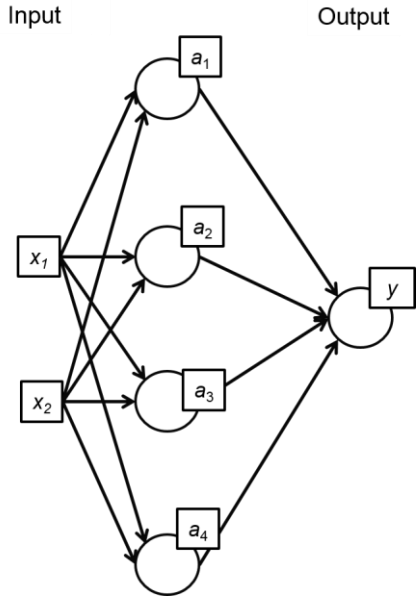
Neural Networks and Deep Learning – Summer Term 2018

Exercise sheet 6

Submission due: Tuesday, July 03, 11:30 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 1 and value 1 within the circle and value 0 outside of the circle.
The synaptic weights between the input node x_j and the hidden node a_i are $w_{ij} = c_{j,(i)}$ 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
$x_1 = 0.5$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.8$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		
$x_1 = -0.2$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.5$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		

Solution:

1) We have

$a_j = \phi_j(r_j) = \Theta[0.5 - r_j]$ with $r_j = \|\mathbf{x} - \mathbf{c}_j\|$ for $j = 1,2,3,4$; $\mathbf{x} = (x_1, x_2)^T = (0.5, 0.8)^T$

$\mathbf{c}_1 = (0.5, 0.5)^T \quad \Rightarrow r_1 = \|\mathbf{x} - \mathbf{c}_1\| = \left\|\begin{pmatrix}0.5\\0.8\end{pmatrix} - \begin{pmatrix}0.5\\0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}0.0\\0.3\end{pmatrix}\right\| = 0.3$

$\mathbf{c}_2 = (-0.5, 0.5)^T \quad \Rightarrow r_2 = \|\mathbf{x} - \mathbf{c}_2\| = \left\|\begin{pmatrix}0.5\\0.8\end{pmatrix} - \begin{pmatrix}-0.5\\0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}1.0\\0.3\end{pmatrix}\right\| = \sqrt{1.09} = 1.044$

$\mathbf{c}_3 = (-0.5, -0.5)^T \quad \Rightarrow r_3 = \|\mathbf{x} - \mathbf{c}_3\| = \left\|\begin{pmatrix}0.5\\0.8\end{pmatrix} - \begin{pmatrix}-0.5\\-0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}1.0\\1.3\end{pmatrix}\right\| = \sqrt{2.69} = 1.640$

$\mathbf{c}_4 = (0.5, -0.5)^T \quad \Rightarrow r_4 = \|\mathbf{x} - \mathbf{c}_4\| = \left\|\begin{pmatrix}0.5\\0.8\end{pmatrix} - \begin{pmatrix}0.5\\-0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}0.0\\1.3\end{pmatrix}\right\| = 1.3$

Therefore we get for the hidden layer activations:

$a_1 = \phi_1(r_1) = \Theta[0.5 - r_1] = \Theta[0.2] = 1$

$a_2 = \phi_2(r_2) = \Theta[0.5 - r_2] = \Theta[-0.544] = 0$

$a_1 = \phi_3(r_3) = \Theta[0.5 - r_3] = \Theta[-1.14] = 0$

$a_1 = \phi_4(r_4) = \Theta[0.5 - r_4] = \Theta[-0.8] = 0$

For the linear layer we obtain

$$z = \mathbf{V} \cdot \mathbf{a} - \theta = \mathbf{V} \cdot \mathbf{a} = (1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 1$$

The final result is $y = z = 1$, since it is a *linear* element.

2) We have

$a_j = \phi_j(r_j) = \Theta[0.5 - r_j]$ with $r_j = \|\mathbf{x} - \mathbf{c}_j\|$ for $j = 1,2,3,4$; $\mathbf{x} = (x_1, x_2)^T = (-0.2, 0.5)^T$

$\mathbf{c}_1 = (0.5, 0.5)^T \quad \Rightarrow r_1 = \|\mathbf{x} - \mathbf{c}_1\| = \left\|\begin{pmatrix} -0.2 \\ 0.5 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\right\| = \left\|\begin{pmatrix} -0.7 \\ 0 \end{pmatrix}\right\| = 0.7$

$\mathbf{c}_2 = (-0.5, 0.5)^T \quad \Rightarrow r_2 = \|\mathbf{x} - \mathbf{c}_2\| = \left\|\begin{pmatrix} -0.2 \\ 0.5 \end{pmatrix} - \begin{pmatrix} -0.5 \\ 0.5 \end{pmatrix}\right\| = \left\|\begin{pmatrix} 0.3 \\ 0 \end{pmatrix}\right\| = 0.3$

$\mathbf{c}_3 = (-0.5, -0.5)^T \quad \Rightarrow r_3 = \|\mathbf{x} - \mathbf{c}_3\| = \left\|\begin{pmatrix} -0.2 \\ 0.5 \end{pmatrix} - \begin{pmatrix} -0.5 \\ -0.5 \end{pmatrix}\right\| = \left\|\begin{pmatrix} 0.3 \\ 1.0 \end{pmatrix}\right\| = \sqrt{1.09} = 1.044$

$\mathbf{c}_4 = (0.5, -0.5)^T \quad \Rightarrow r_4 = \|\mathbf{x} - \mathbf{c}_4\| = \left\|\begin{pmatrix} -0.2 \\ 0.5 \end{pmatrix} - \begin{pmatrix} 0.5 \\ -0.5 \end{pmatrix}\right\| = \left\|\begin{pmatrix} -0.7 \\ 1.0 \end{pmatrix}\right\| = \sqrt{1.49} = 1.221$

Therefore we get for the hidden layer activations:

$a_1 = \phi_1(r_1) = \Theta[0.5 - r_1] = \Theta[-0.2] = 0$

$a_2 = \phi_2(r_2) = \Theta[0.5 - r_2] = \Theta[0.2] = 1$

$a_1 = \phi_3(r_3) = \Theta[0.5 - r_3] = \Theta[-0.544] = 0$

$a_1 = \phi_4(r_4) = \Theta[0.5 - r_4] = \Theta[-0.721] = 0$

For the linear layer we obtain

$$z = \mathbf{V} \cdot \mathbf{a} - \theta = \mathbf{V} \cdot \mathbf{a} = (1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = 2$$

The final result is $y = z = 2$, since it is a *linear* element.

This can be summarized as follows:

Inputs	distance r_i at hidden nodes	RBF values a_i	PSP z at output node	Output y
$x_1 = 0.5$	$r_1 = 0.3$	$a_1 = 1$	$z = 1$	$y = 1$
	$r_2 = 1.044$	$a_2 = 0$		
$x_2 = 0.8$	$r_3 = 1.640$	$a_3 = 0$		
	$r_4 = 1.3$	$a_4 = 0$		
$x_1 = -0.2$	$r_1 = 0.7$	$a_1 = 0$	$z = 2$	$y = 2$
	$r_2 = 0.3$	$a_2 = 1$		
$x_2 = 0.5$	$r_3 = 1.044$	$a_3 = 0$		
	$r_4 = 1.221$	$a_4 = 0$		

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.

Solution:

The four radial basis functions are circles with radius 0.5 around the four points $\mathbf{c}_1 = (0.5, 0.5)^T$, $\mathbf{c}_2 = (-0.5, 0.5)^T$, $\mathbf{c}_3 = (-0.5, -0.5)^T$ and $\mathbf{c}_4 = (0.5, -0.5)^T$. Two of these circles have at most one point in common namely the four points $(0.5, 0)^T$, $(0, 0.5)^T$, $(-0.5, 0)^T$, $(0, -0.5)^T$; otherwise, the circles do not intersect. Therefore, except for these four points, any input point $\mathbf{x} = (x_1, x_2)^T$ will activate at most a single radial basis function, namely the one into which circle the input falls (or the activation of all radial basis functions will be 0). The four weights

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

can be seen to indicate which of the radial basis functions is activated. This leads to the following figure depicting the output for any input pattern (except the four points mentioned above). For these four points, the output is calculated as follows:

$\mathbf{x} = (x_1, x_2)^T = (0.5, 0)^T$

$\mathbf{c}_1 = (0.5, 0.5)^T \quad \Rightarrow r_1 = \|\mathbf{x} - \mathbf{c}_1\| = \left\|\begin{pmatrix}0.5\\0\end{pmatrix} - \begin{pmatrix}0.5\\0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}0\\-0.5\end{pmatrix}\right\| = 0.5$

$\mathbf{c}_2 = (-0.5, 0.5)^T \quad \Rightarrow r_2 = \|\mathbf{x} - \mathbf{c}_2\| = \left\|\begin{pmatrix}0.5\\0\end{pmatrix} - \begin{pmatrix}-0.5\\0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}1.0\\-0.5\end{pmatrix}\right\| = \sqrt{1.25} = 1.118$

$\mathbf{c}_3 = (-0.5, -0.5)^T \quad \Rightarrow r_3 = \|\mathbf{x} - \mathbf{c}_3\| = \left\|\begin{pmatrix}0.5\\0\end{pmatrix} - \begin{pmatrix}-0.5\\-0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}1.0\\0.5\end{pmatrix}\right\| = \sqrt{1.25} = 1.118$

$\mathbf{c}_4 = (0.5, -0.5)^T \quad \Rightarrow r_4 = \|\mathbf{x} - \mathbf{c}_4\| = \left\|\begin{pmatrix}0.5\\0\end{pmatrix} - \begin{pmatrix}0.5\\-0.5\end{pmatrix}\right\| = \left\|\begin{pmatrix}0\\0.5\end{pmatrix}\right\| = 0.5$

Therefore we get for the hidden layer activations:

$$a_1 = \phi_1(r_1) = \Theta[0.5 - r_1] = \Theta[0] = 1$$

$$a_2 = \phi_2(r_2) = \Theta[0.5 - r_2] = \Theta[-0.618] = 0$$

$$a_1 = \phi_3(r_3) = \Theta[0.5 - r_3] = \Theta[-0.618] = 0$$

$$a_1 = \phi_4(r_4) = \Theta[0.5 - r_4] = \Theta[0] = 1$$

$$z = \mathbf{V} \cdot \mathbf{a} - \theta = \mathbf{V} \cdot \mathbf{a} = (1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix}1\\0\\0\\1\end{pmatrix} = 5$$

Correspondingly, we get

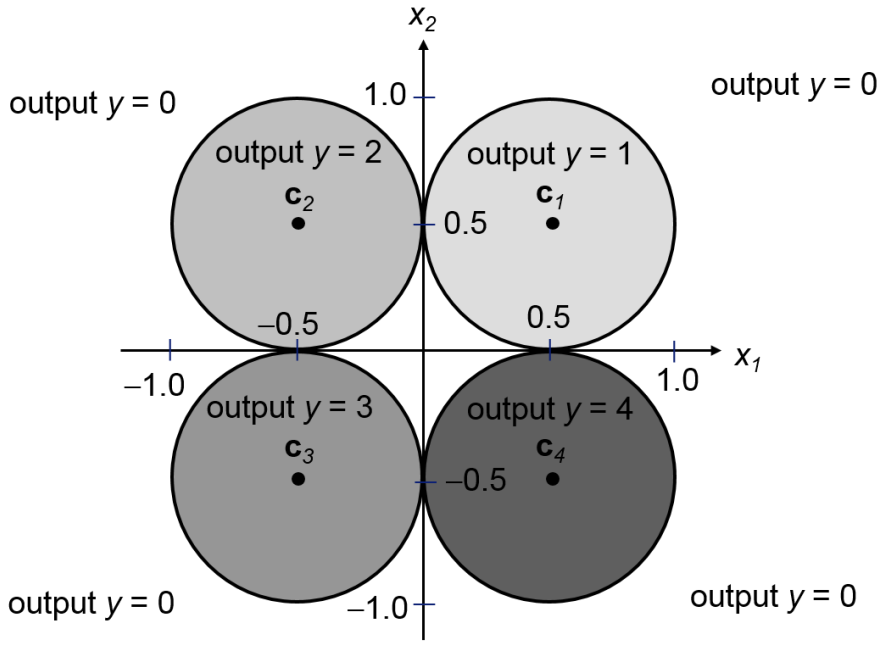
$\mathbf{x} = (x_1, x_2)^T = (0, 0.5)^T \quad \Rightarrow y = 1 + 2 = 3$

$\mathbf{x} = (x_1, x_2)^T = (-0.5, 0)^T \Rightarrow y = 2 + 3 = 5$

$\mathbf{x} = (x_1, x_2)^T = (0, -0.5)^T \Rightarrow y = 3 + 4 = 7$

The output is not unique; the inputs $(0.5, 0)^T$ and $(-0.5, 0)^T$ yield the same output 5.

To summarize, we get the following partition of the input space:



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, … , 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 **W** between the inputs and the hidden nodes and the synaptic weight vector **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
$x_1 = 0.5$	$z_1 =$	$a_1 =$	$z =$	$y =$
	$z_2 =$	$a_2 =$		
$x_2 = 0.8$	$z_3 =$	$a_3 =$		
	$z_4 =$	$a_4 =$		
$x_1 = –0.2$	$r_1 =$	$a_1 =$	$z =$	$y =$
	$r_2 =$	$a_2 =$		
$x_2 = 0.5$	$r_3 =$	$a_3 =$		
	$r_4 =$	$a_4 =$		

Solution:

The synaptic weight matrix from two input units to 4 hidden units is a 4 × 2 matrix, the threshold $\boldsymbol{\theta}$ is a 4-dimensional vector and both are given by (see above)

$$\mathbf{W} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \\ \mathbf{c}_4 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & -0.5 \end{pmatrix}; \quad \boldsymbol{\theta} = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

The postsynaptic potential z is given by $\mathbf{z} = \mathbf{W} \cdot \mathbf{x} - \boldsymbol{\theta}$ and the activation $\mathbf{a}$ by $\mathbf{a} = \Theta[\mathbf{z}]$. Thus we get:

1) For $\mathbf{x} = (x_1, x_2)^T = (0.5, 0.8)^T$:

$$\mathbf{z} = \mathbf{W} \cdot \mathbf{x} - \boldsymbol{\theta} = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & -0.5 \end{pmatrix} \cdot \begin{pmatrix} 0.5 \\ 0.8 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.65 \\ 0.15 \\ -0.65 \\ -0.15 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.15 \\ -0.35 \\ -1.15 \\ -0.65 \end{pmatrix}$$

For the activations at the hidden layer we obtain:

$$\mathbf{a} = \Theta[\mathbf{z}] = \Theta\left[\begin{pmatrix} 0.15 \\ -0.35 \\ -1.15 \\ -0.65 \end{pmatrix}\right] = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

For the final layer we obtain with $\theta = 0.5$:

$$z = \mathbf{V} \cdot \mathbf{a} - \theta = \mathbf{V} \cdot \mathbf{a} = (1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 0.5 = 1 - 0.5 = 0.5 \text{ and } y = \Theta[z] = 1$$

2) For $\mathbf{x} = (x_1, x_2)^T = (-0.2, 0.5)^T$:

$$\mathbf{z} = \mathbf{W} \cdot \mathbf{x} - \boldsymbol{\theta} = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & -0.5 \end{pmatrix} \cdot \begin{pmatrix} -0.2 \\ 0.5 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.15 \\ 0.35 \\ -0.15 \\ -0.35 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} -0.35 \\ -0.15 \\ -0.65 \\ -0.85 \end{pmatrix}$$

For the activations at the hidden layer we obtain:

$$\mathbf{a} = \Theta[\mathbf{z}] = \Theta\left[\begin{pmatrix} -0.35 \\ -0.15 \\ -0.65 \\ -0.85 \end{pmatrix}\right] = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

For the final layer we obtain with $\theta = 0.5$:

$$z = \mathbf{V} \cdot \mathbf{a} - \theta = \mathbf{V} \cdot \mathbf{a} = (1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 0.5 = -0.5 \text{ and } y = \Theta[z] = 0$$

This can be summarized as follows:

Inputs	PSP z_i at hidden nodes	activations a_j at hidden nodes	PSP z at output node	Output y
$x_1 = 0.5$	$z_1 = 0.15$	$a_1 = 1$	$z = 0.5$	$y = 1$
	$z_2 = -0.35$	$a_2 = 0$		
$x_2 = 0.8$	$z_3 = -1.15$	$a_3 = 0$		
	$z_4 = -0.65$	$a_4 = 0$		
$x_1 = -0.2$	$r_1 = -0.35$	$a_1 = 0$	$z = -0.5$	$y = 0$
	$r_2 = -0.15$	$a_2 = 0$		
$x_2 = 0.5$	$r_3 = -0.65$	$a_3 = 0$		
	$r_4 = -0.85$	$a_4 = 0$		

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.

Solution:

The equation for the final layer reads:

$$y = \Theta[\mathbf{V} \cdot \mathbf{a} - \theta] = \Theta\left[(1 \quad 2 \quad 3 \quad 4) \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} - 0.5\right] = \Theta[a_1 + 2a_2 + 3a_3 + 4a_4 - 0.5]$$

Since the activations at the hidden layer are binary, $\mathbf{a} = \Theta[\mathbf{z}]$, we see that if one or more of the hidden units is activated, the output y of the multi-layer perceptron is 1. In other words, the output of the multi-layer perceptron is 0 only if none of the hidden units is activated: $a_j = 0$ for all $j = 1, ..., 4$. Since $a_j = \Theta[z_j]$, this amounts to stating that all postsynaptic potentials z_j must be negative:

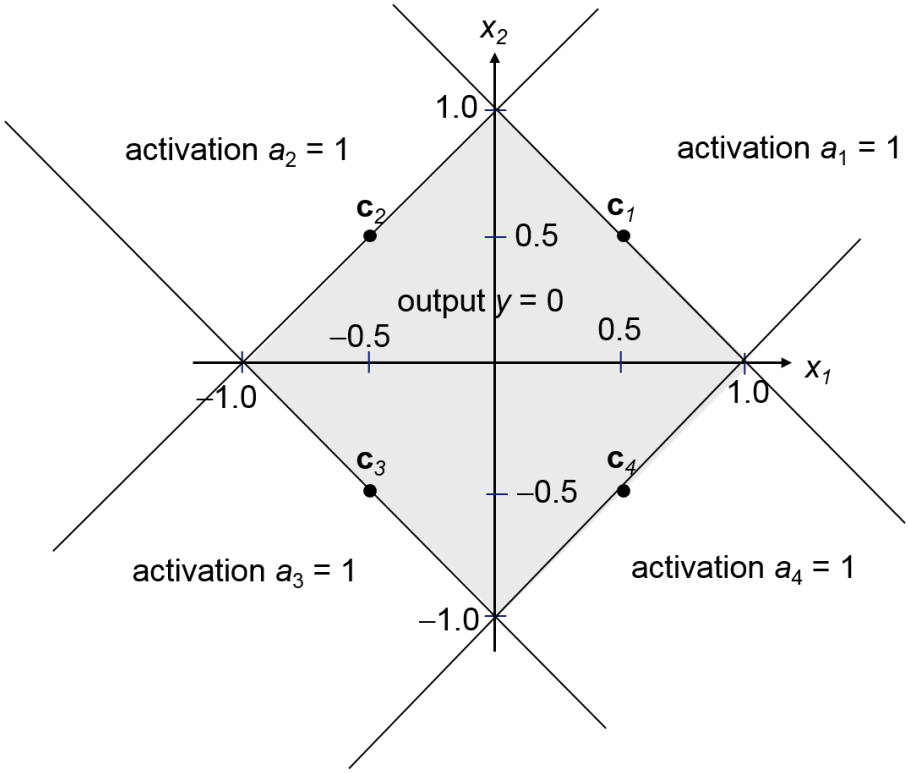
$$\mathbf{z} = \mathbf{W} \cdot \mathbf{x} - \boldsymbol{\theta} = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & -0.5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.5x_1 + 0.5x_2 - 0.5 \\ -0.5x_1 + 0.5x_2 - 0.5 \\ -0.5x_1 - 0.5x_2 - 0.5 \\ 0.5x_1 - 0.5x_2 - 0.5 \end{pmatrix} \overset{!}{<} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Thus we have

$$x_1 + x_2 - 1 < 0 \quad \Leftrightarrow \; x_2 < 1 - x_1$$
$$-x_1 + x_2 - 1 < 0 \quad \Leftrightarrow \; x_2 < 1 + x_1$$
$$-x_1 - x_2 - 1 < 0 \quad \Leftrightarrow \; x_2 > -1 - x_1$$
$$x_1 - x_2 - 1 < 0 \quad \Leftrightarrow \; x_2 > -1 + x_1$$

These four lines are plotted in the figure below; the four conditions mean that an input must be within the shaded square in order to yield the output $y = 0$; otherwise the output is 1.

Therefore, we obtain the following partition of the input space:


x_2
1.0
activation a_2 = 1
activation a_1 = 1
$\mathbf{c}_2$
$\mathbf{c}_1$
0.5
output y = 0
–0.5
0.5
x_1
–1.0
1.0
$\mathbf{c}_3$
$\mathbf{c}_4$
–0.5
activation a_3 = 1
activation a_4 = 1
–1.0

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).

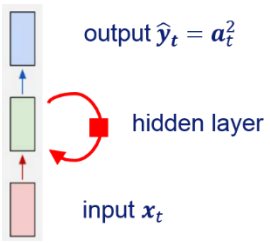
a) Depict a diagram of the recurrent neural network (including the number of neurons in each layer, the activation functions, the update equations and the dimensions of the matrices and vectors involved). Describe in detail how the network processes a sequence of characters in training (what is input, 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.

Solution:

The recurrent network has the following topology:

- Number of input units: As many units as there are different characters in the input file **`input.txt`**; in the example provided, there are 34 different characters (where upper case and lower case are distinguished, i.e. 'W' and 'w' are two different characters, and also punctuation symbols like '.', ',' and ' ' count as individual characters), so there are 34 input units.
- Number of hidden units: The network has **`hidden_size`** hidden units; in the example provided, there are per default 100 hidden units. There is only a single hidden layer in the network. The hidden layer uses a "tanh" activation function.
- Number of output units: There are as many output units as input units; in the example provided, there are 34 different output units. The output layer uses a softmax activation function.

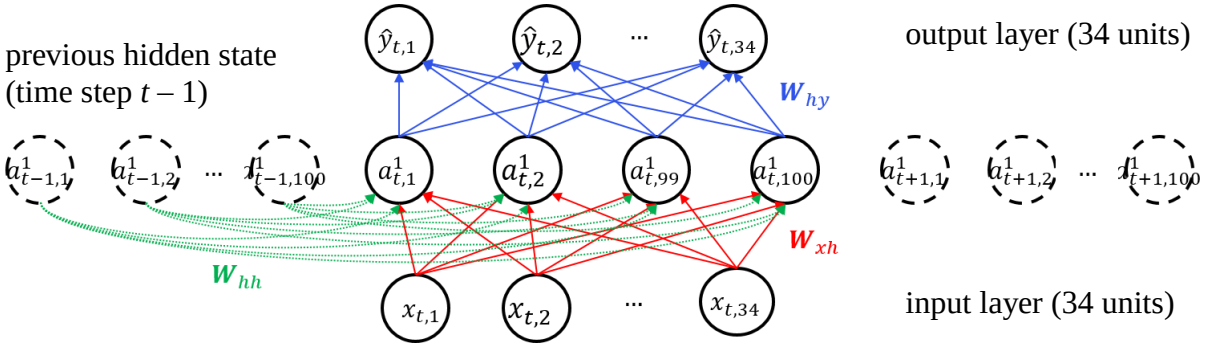
Simple and extended diagram of the network:



output layer (34 units)

hidden layer (100 units) with recurrent connections

input layer (34 units)



Notation:

$x_{t,2}$: input at time t (first subscript), second component (second subscript)
$a^1_{t,100}$: activation in first hidden layer (superscript) at time t (1st subscript), component 100 (2nd subscript)
$\hat{y}_{t,34}$: output at time t (first subscript), component 34 (second subscript)

Remarks:

- The input-hidden connections W_{xh} connect the 34 input units to the 100 hidden units (100 × 34 matrix) and are shown in red
- The recurrent connections W_{hh} connect the 100 hidden units, i.e. their values e.g. at time $t - 1$ to their value at time t (100 × 100 matrix); they are shown dashed green; only the connections from time step $t - 1$ to t are shown for clarity, indicating that the hidden state at time $t - 1$ influences the hidden state at time t; similarly, the hidden state at time t influences the hidden state at time $t + 1$ (shown to the right of the diagram above, but without synaptic connections for clarity of presentation)
- The hidden-output connections W_{hy} connect the 100 hidden units to the 34 output units (34 × 100 matrix) and are shown in blue

The forward update equations are given as follows (see lecture slides and the code in **`exercise2.py`**, lines 78 – 80):

$$\boldsymbol{z}_t^1 = \mathbf{W}_{xh}\boldsymbol{x}_t + \mathbf{W}_{hh}\boldsymbol{a}_{t-1}^1 + \boldsymbol{b}^1 \quad (1)$$
$$\boldsymbol{a}_t^1 = \tanh(\boldsymbol{z}_t^1) \quad (2)$$
$$\boldsymbol{z}_t^2 = \mathbf{W}_{hy}\boldsymbol{a}_t^1 + \boldsymbol{b}^2 \quad (3)$$
$$\hat{\boldsymbol{y}}_t = \mathrm{softmax}(\boldsymbol{z}_t^2) \quad (4)$$

$\boldsymbol{a}_{t-1}^1, \boldsymbol{a}_t^1, \boldsymbol{b}^1, \boldsymbol{z}_t^1$: 100-dim.
$\boldsymbol{x}_t$: 34-dim.
$\mathbf{W}_{xh}$: 100 × 34, $\mathbf{W}_{hh}$: 100 × 100, $\mathbf{W}_{hy}$: 34 × 100
$\boldsymbol{b}^2, \boldsymbol{z}_t^2$: 34-dim.
$\hat{\boldsymbol{y}}$: 34-dim.

Training proceeds as follows:

- At the beginning of training or after each complete pass over the input data, the hidden layer activations are initialised to $\boldsymbol{a}_0^1 = 0$ ($t = 0$) and the algorithm starts (again) at the beginning of the input file ($p = 0$).
- A sequence of **`seq_length`** input characters is constructed (in the example per default 25), here the first 25 characters of the file **`input.txt`**. For each input character, the corresponding target is the letter immediately following the input character. Therefore, the targets are a sequence of **`seq_length`** characters, which are shifted by 1 character with respect to the inputs. Both inputs and targets are converted to a one-hot-encoding (here 34-dimensional vectors, see above), so the (current) inputs are a sequence of **`seq_length`** 34-dimensional vectors, and similarly the (current) targets are a sequence of **`seq_length`** 34-dimensional vectors, shifted by 1 character. The goal is to predict a single output character (the next character in a sequence) from a single input character at each time step.
- Starting with the initial value $\boldsymbol{a}_0^1 = 0$, a forward pass is performed sequentially for the **`seq_length`** input characters (one after the other, propagating the hidden layer activations), i.e., the network is unfolded for **`seq_length`** time steps. For each input character $\boldsymbol{x}_t$ (in one-hot-encoding) at time t, the network output $\hat{\boldsymbol{y}}_t$ is computed (as

probability distribution, 34-dimensional vector) and compared to the target (34-dimensional vector, one-hot-encoding). The log-likelihood loss is calculated over the sequence of examples.

- Based on the calculated log-likelihood loss, the backward equations are applied to the network which is unfolded for **`seq_length`** time steps and the network parameters are updated (truncated backpropagation through time) using "AdaGrad". Clipping is applied to the network parameters in order to prevent too small or too large values of the parameters.
- Then, the sequence is shifted by **`seq_length`** characters. For the forward equations, the last hidden layer activations are used (instead of setting the hidden layer activations to 0, which only occurs when iterations start at the beginning of the file).
- This is performed for a number of iterations (in the example, maximally 10000 iterations).

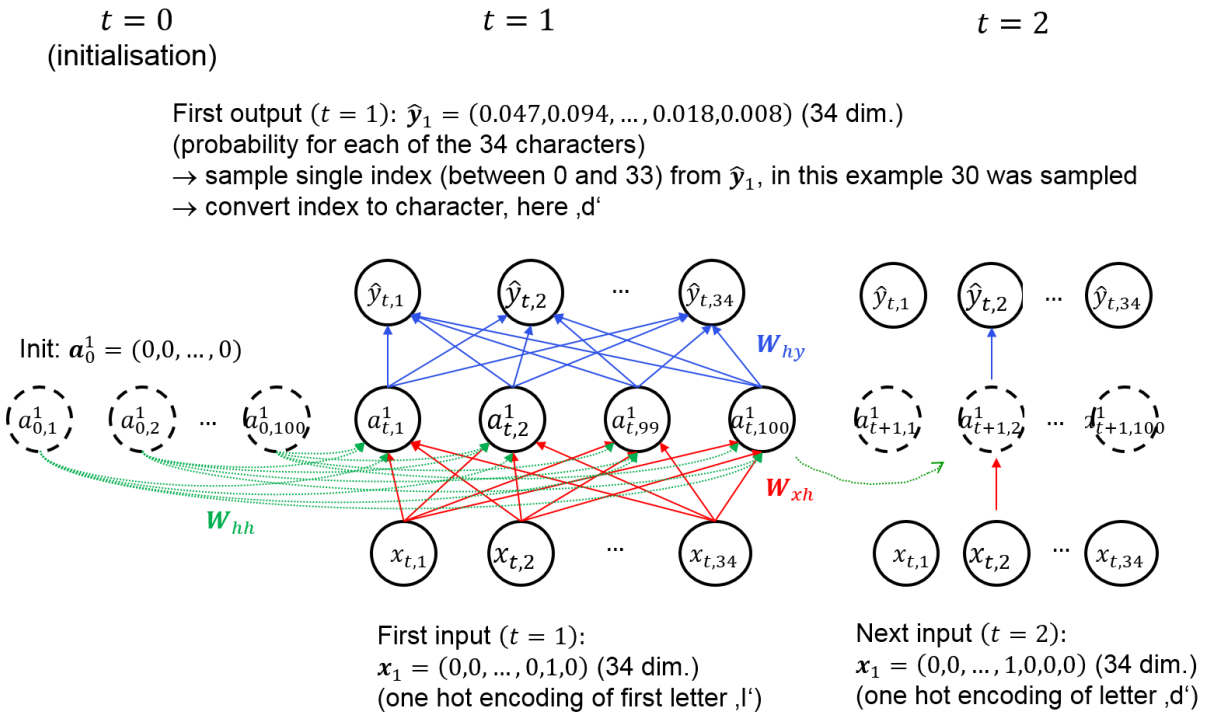
Since **`seq_length`** time steps are considered in a single backpropagation through time update step, the network has a "memory" of at most **`seq_length`** time steps. The next parameter update is then performed on the sequence shifted by **`seq_length`** time steps, where in the forward propagation the hidden layer activations are propagated, but in the backward equations, the individual sequences of **`seq_length`** time steps are treated independently from each other. To improve prediction performance, the **`seq_length`** should be made large; this, however, increases training complexity and slows down training.

Note that at each time, a *single* character is input to the network, *not* a sequence of characters. The latter would be the case if we would use a feedforward neural network, e.g. a multi-layer perceptron, to predict the sequence; in this case, we have to input a "history" of characters to the network, because otherwise the feedforward network had no chance of representing "long-range dependencies" in the character sequence beyond a single character. In case of a recurrent network, the history of characters is "remembered" within the hidden layer activations, which are passed via the recurrent connections from time step to time step in order to store the long-range dependencies. Together with the current input (the current character), the hidden layer activations must guide the network to predict the correct subsequent letter, depending on the history of characters "stored" in the hidden layer activations. Therefore, the hidden layer activations must be "rich" enough (i.e. there must be enough hidden neurons) to "store" the history of characters.

The network generates a sequence as follows (see also the diagram below):

- First, the hidden layer activations are initialised to $\boldsymbol{a}_0^1 = 0$ ($t = 0$).
- Then, at each time step, a single letter in form of a one-hot-encoding is input to the network. Here, there are 34 different characters (see above), so the input vector is 34-dimensional. The first input $\boldsymbol{x}_1$ ($t = 1$) corresponds to the first letter of either the current sequence, or of the file **`input.txt`**; in the latter case, the first letter is 'I'.
- Using the initial value $\boldsymbol{a}_0^1$ and the first input $\boldsymbol{x}_1$ as well as the current parameters for $\mathbf{W}_{xh}, \mathbf{W}_{hh}$ and $\boldsymbol{b}^1$, the postsynaptic potential $\boldsymbol{z}_1^1$ and the activation $\boldsymbol{a}_1^1$ at time $t = 1$ are calculated according to equations (1) and (2).
- With this values and the current parameters for $\mathbf{W}_{hy}$ and $\boldsymbol{b}^2$, the postsynaptic potential $\boldsymbol{z}_1^2$ and the output $\hat{\boldsymbol{y}}_1$ at time $t = 1$ are calculated according to equations (3) and (4). The output is a 34-dimensional vector containing probabilities for each of the 34 characters.

- From the output probabilities $\hat{\boldsymbol{y}}_1$, a single index is sampled and converted to the corresponding character, which is the first generated output, i.e. the character following the first character 'I'; in the example, the next character is 'd'.
- The predicted character (in form of a one-hot encoding) is then used as input at the next time step and the process is iterated until the end of the loop (in the example either 200 characters for the first letter of the current sequence and 100 characters for the first letter of the input file).



Note that the output $\hat{\boldsymbol{y}}_1$ has been (re-)calculated with the fully trained network.
The synaptic connections at time step t = 2 have only roughly been indicated; they correspond to the synaptic connections at time step t = 1.

Running the script **`exercise2.py`**, the following output was obtained (only the beginning and end of the output are shown; note that a random component is involved in the output):

```
data has 184 characters, 34 unique.
----
sample 200 letters from current position:
hukWVFldhb?bngVha
aGemfblijdzbIWkjVf           zoGwwD           t,bcinemVhdecDwm,,acGB
wbn.blHjBdbudzFwk, ?uHemzaWr.F dVt.?DkkrjumsdIFjmBcG
ltfankk
WnV f VWronogbIindl,kbkDno.s  rGBGz?DVBocD?F
BFFmFtjestWhawn
gFfodtBftbe
----
sample 100 letters from beginning of text:
I BWgnlglchFoH
fWtwmzFbozhF drFgdgfBeFahezWhoVnIejmsgasfVBDwm?Frk?GWob
```

```
fWuu?D?
fkteDrdcogsokdklt,oD?w
----

iter 0, loss: 88.159010
----
sample 200 letters from current position:
taenneiiz
 Dtjene           ,ansaeizeeaht,eazmener                          afaet
jtahcneneeGeneleeiascetrhnecercezeDenektes
awt,feasned,zD,ihs,ee,Gan iini
 rniee t tseekdiuD aieseetiseekaetlldeeaD,t teeeeeeeDueetdeihedten
seGreruaWf
----
sample 100 letters from beginning of text:
Ihr nbet,lt,
aeieateatseeteahac ea  eneettzt eee,lnDeen,rsatt
hnnnoa,aecaen eedutzfeeileica,ereneifie
----

iter 100, loss: 88.917367


...

sample 200 letters from current position:
hr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn genderuekeiderz noch jenem
----
sample 100 letters from beginning of text:
Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch
----

iter 9800, loss: 0.075354
----
sample 200 letters from current position:
ueh sich einst dem trueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gen Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mei
----
sample 100 letters from beginning of text:
Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch
----

iter 9900, loss: 0.073697
Training duration (s) :  26.78624725341797

Final test: Trying to reproduce training sequence; predicted text:
```

```
Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gesder, sch

[[   0.    1.    2. ...,  182.  183.  184.]
 [   1.    0.    1. ...,  181.  182.  183.]
 [   2.    1.    0. ...,  180.  181.  182.]
 ...,
 [ 183.  182.  181. ...,    6.    7.    8.]
 [ 184.  183.  182. ...,    7.    7.    8.]
 [ 185.  184.  183. ...,    8.    8.    8.]]

there are 8 errors
```

I.e., after 10000 iterations the network has learned the sequence quite well (with only 8 errors).

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.

Solution:

Running the script 10 times on **`input_short.txt`**, the following output was obtained (excerpt):

```
data has 184 characters, 34 unique.

Iteration number 1
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gestalten,

there are 7 errors

Iteration number 2
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch ich mein Her, such ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jen

there are 37 errors

Iteration number 3
Final test: Trying to reproduce training sequence; predicted text:
```

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gez noch wo

there are 9 errors

Iteration number 4
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich aedieschn gez noch jenem W, ni.
Versuch wohl, ich Her, dezenem Wahn gezeigt.
Versuch wohl, euch diesmal festzuhalten?
Fuehl i

there are 95 errors

Iteration number 5
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frues sich einst dem trueben Blick gezeiltnsHerz noch jenem Wahn gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein He

there are 60 errors

Iteration number 6
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Geaeben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gezeigt.
Versuch ich wohl, euch diesmal festzuh

there are 76 errors

Iteration number 7
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestesten,
Dioch jenen Her, schwankende Gestalten,
Die frueh sich einst dem trueben Blick gezeigt.
Veruch mhloch jeDich Gestalten,
Die frueh sich eVert

there are 97 errors

Iteration number 8
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gessin?
Fuehl ich mein Herz noch jenem Wahn geueben Blick gezeigt.
Versuch ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn geu

```
there are 41 errors

Iteration number 9
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich einst dem trswal ich wohl, euch diesmal festzuhalten?
Fuehl ich mein Her, schwankende Gestaht dem trsuch ich wohl, euch
diesm

there are 70 errors

Iteration number 10
Final test: Trying to reproduce training sequence; predicted text:

Ihr naht euch wieder, schwankende Gestalten,
Die frueh sich eich wohl, euch diesmal festzuhalten?
Fuehl ich mein Herz noch jenem Wahn gezeigt.
Versuch ich wohl, euch diesmal festzuhalte

there are 84 errors

Error summary:
[7.0, 37.0, 9.0, 95.0, 60.0, 76.0, 97.0, 41.0, 70.0, 84.0]
Mean error: 57.600000
Standard deviation: 32.945072

Training time summary:
[24.24244260787964,     23.89924192428589,     23.836841821670532,
23.852441787719727,     23.883641958236694,     23.774441719055176,
23.774441719055176,     23.94604206085205,     23.914842128753662,
25.833645582199097]
Mean error: 24.095802
Standard deviation: 0.624789
```

Note that the number of errors varies strongly between the different iterations. The training time is, however, quite comparable.

The results of testing different configurations are summarized in the following table:

hidden_size	seq_length	learning_rate	mean error	mean train time
100	10	0.1	63.50 ± 43.572	12.91 ± 1.310
100	25	0.1	57.60 ± 32.945	24.10 ± 0.625
100	50	0.1	31.90 ± 8.621	53.49 ± 2.674
100	100	0.1	63.80 ± 3.795	116.48 ± 4.858
50	50	0.1	53.10 ± 40.037	33.43 ± 4.272
200	50	0.1	115.50 ± 22.29	118.29 ± 3.74
100	50	0.2	77.50 ± 38.925	65.68 ± 2.790
100	50	0.05	37.50 ± 27.646	62.35 ± 4.93

The results are interpreted as follows:

- A shorter sequence length increases the mean error (while reducing mean training time), a larger sequence length improves results at the expense of a larger training time. If the sequence length is too large (here 100), results are getting worse again. This is potentially due to the short input file length, i.e. the sequence length should be much smaller than the length of the input file.
- Reducing the number of hidden neurons increases the mean error (while reducing mean training time), a larger number of hidden neurons improves results at the expense of a larger training time. Increasing the number of hidden neurons drastically increases the mean error, potentially due to the fact that the large number of parameters cannot be trained reliably anymore. Surprisingly, the training time increases only slightly.
- A larger learning rate deteriorates convergence, yielding a larger mean error at a larger training time. Reducing the learning rate slightly increased the mean error as well as the training time.
- Therefore, the optimal configuration (out of those tested) seem to be 100 hidden neurons, a sequence length of 50 and a learning rate of 0.1.

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"):

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

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

$\boldsymbol{b}_c = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ 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

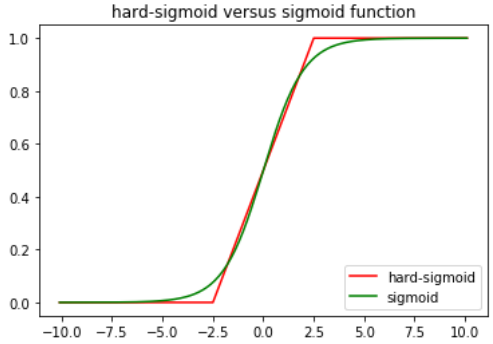
$$\boldsymbol{x}_1 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \qquad \boldsymbol{x}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

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.

Solution:

The update equations of the hidden LSTM units (with hard-sigmoid activation function at the input, forget and output gate) are given as follows:

$$\boldsymbol{i}_t = \sigma_{hard}(\mathbf{W}_{xi}\boldsymbol{x}_t + \mathbf{W}_{hi}\boldsymbol{a}_{t-1} + \boldsymbol{b}_i)$$
$$\boldsymbol{f}_t = \sigma_{hard}(\mathbf{W}_{xf}\boldsymbol{x}_t + \mathbf{W}_{hf}\boldsymbol{a}_{t-1} + \boldsymbol{b}_f)$$
$$\boldsymbol{o}_t = \sigma_{hard}(\mathbf{W}_{xo}\boldsymbol{x}_t + \mathbf{W}_{ho}\boldsymbol{a}_{t-1} + \boldsymbol{b}_o)$$
$$\boldsymbol{g}_t = \tanh(\mathbf{W}_{xc}\boldsymbol{x}_t + \mathbf{W}_{hc}\boldsymbol{a}_{t-1} + \boldsymbol{b}_c)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t$$
$$\boldsymbol{a}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

Note that the hidden and the cell state have the same dimension (namely the number of hidden units, since each LSTM has a cell state in addition to the hidden state). By virtue of the update equations, all gates then also have the same dimension (there is one input gate, one forget gate, one "gate" gate and one output gate per hidden LSTM unit).

The equations to calculate the network output $\hat{\boldsymbol{y}}_t$ is given by

$\boldsymbol{z}_t = \mathbf{W}_{yh}\boldsymbol{a}_t + \boldsymbol{b}_y$ postsynaptic potential at output units from hidden unit activations
$\hat{\boldsymbol{y}}_t = \mathrm{ReLU}(\boldsymbol{z}_t)$ network output

Time step 1:

$$\boldsymbol{i}_1 = \sigma_{hard}(\mathbf{W}_{xi}\boldsymbol{x}_1 + \mathbf{W}_{hi}\boldsymbol{a}_0 + \boldsymbol{b}_i) = \sigma_{hard}\left(\begin{pmatrix}2 & 0 & 3\\ 4 & 1 & 0\end{pmatrix}\begin{pmatrix}1\\0\\-1\end{pmatrix} + \begin{pmatrix}-3 & 0\\ -5 & 1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\-2\end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix}-1\\4\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\-2\end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix}-1\\2\end{pmatrix}\right) = \begin{pmatrix}-0.2+0.5\\0.2*2+0.5\end{pmatrix} = \begin{pmatrix}0.3\\0.9\end{pmatrix}$$

$$\boldsymbol{f}_1 = \sigma_{hard}(\mathbf{W}_{xf}\boldsymbol{x}_1 + \mathbf{W}_{hf}\boldsymbol{a}_0 + \boldsymbol{b}_f) = \sigma_{hard}\left(\begin{pmatrix}1 & 0 & -1\\ 2 & 1 & 2\end{pmatrix}\begin{pmatrix}1\\0\\-1\end{pmatrix} + \begin{pmatrix}-2 & 0\\ 2 & 1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}1\\-1\end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix}2\\0\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}1\\-1\end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix}3\\-1\end{pmatrix}\right) = \begin{pmatrix}1\\0.3\end{pmatrix}$$

$$\boldsymbol{o}_1 = \sigma_{hard}(\mathbf{W}_{xo}\boldsymbol{x}_1 + \mathbf{W}_{ho}\boldsymbol{a}_0 + \boldsymbol{b}_o) = \sigma_{hard}\left(\begin{pmatrix}-1 & 2 & -2\\ 0 & 1 & 1\end{pmatrix}\begin{pmatrix}1\\0\\-1\end{pmatrix} + \begin{pmatrix}1 & 1\\ 0 & 2\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}-1\\-1\end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix}1\\-1\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}-1\\-1\end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix}0\\-2\end{pmatrix}\right) = \begin{pmatrix}0\\0.1\end{pmatrix}$$

$$\boldsymbol{g}_1 = \tanh(\mathbf{W}_{xc}\boldsymbol{x}_1 + \mathbf{W}_{hc}\boldsymbol{a}_0 + \boldsymbol{b}_c) = \tanh(\begin{pmatrix}1 & -1 & 0\\ 0 & 2 & 1\end{pmatrix}\begin{pmatrix}1\\0\\-1\end{pmatrix} + \begin{pmatrix}2 & 0\\ -1 & 1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}-1\\2\end{pmatrix}))$$
$$= \tanh(\begin{pmatrix}1\\-1\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}-1\\2\end{pmatrix}) = \tanh(\begin{pmatrix}0\\1\end{pmatrix}) = \begin{pmatrix}0\\\tanh(1)\end{pmatrix} = \begin{pmatrix}0\\0.7616\end{pmatrix}$$

$$\boldsymbol{c}_1 = \boldsymbol{f}_1 \odot \boldsymbol{c}_0 + \boldsymbol{i}_1 \odot \boldsymbol{g}_1 = \begin{pmatrix}1\\0.3\end{pmatrix} \odot \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0.3\\0.9\end{pmatrix} \odot \begin{pmatrix}0\\0.7616\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\0.6854\end{pmatrix} = \begin{pmatrix}0\\0.6854\end{pmatrix}$$

$$\boldsymbol{a}_1 = \boldsymbol{o}_1 \odot \tanh(\boldsymbol{c}_1) = \begin{pmatrix} 0 \\ 0.1 \end{pmatrix} \odot \tanh\left(\begin{pmatrix} 0 \\ 0.6854 \end{pmatrix}\right) = \begin{pmatrix} 0 \\ 0.1 * \tanh(0.6854) \end{pmatrix} = \begin{pmatrix} 0 \\ 0.0595 \end{pmatrix}$$

The hidden layer activation at time $t = 1$ is calculated to be $\boldsymbol{a}_1 = \begin{pmatrix} 0 \\ 0.0595 \end{pmatrix}$.

The output at time time $t = 1$ is calculated as follows:

$$\hat{\boldsymbol{y}}_1 = \mathrm{ReLU}\left(\mathbf{W}_{yh}\boldsymbol{a}_1 + \boldsymbol{b}_y\right) = \mathrm{ReLU}\left(\begin{pmatrix} 2 & -1 \\ 0 & 3 \\ 1 & -1 \\ 4 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ -1 \\ 1 \end{pmatrix}\right) =$$

$$\mathrm{ReLU}\left(\begin{pmatrix} -0.0595 \\ 0.1785 \\ -0.0595 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ -1 \\ 1 \end{pmatrix}\right) = \mathrm{ReLU}\left(\begin{pmatrix} 1.9405 \\ 0.1785 \\ -1.0595 \\ 1 \end{pmatrix}\right) = \begin{pmatrix} 1.9405 \\ 0.1785 \\ 0 \\ 1 \end{pmatrix}$$

Therefore, the network output at time $t = 1$ is $\hat{\boldsymbol{y}}_1 = \begin{pmatrix} 1.9405 \\ 0.1785 \\ 0 \\ 1 \end{pmatrix}$.

Time step 2:

$$\boldsymbol{i}_2 = \sigma_{hard}(\mathbf{W}_{xi}\boldsymbol{x}_2 + \mathbf{W}_{hi}\boldsymbol{a}_1 + \boldsymbol{b}_i) = \sigma_{hard}\left(\begin{pmatrix} 2 & 0 & 3 \\ 4 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -3 & 0 \\ -5 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} 0 \\ -2 \end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix} 3 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} 0 \\ -2 \end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix} 3 \\ -0.9405 \end{pmatrix}\right) = \begin{pmatrix} 1 \\ 0.3119 \end{pmatrix}$$

$$\boldsymbol{f}_2 = \sigma_{hard}\left(\mathbf{W}_{xf}\boldsymbol{x}_2 + \mathbf{W}_{hf}\boldsymbol{a}_1 + \boldsymbol{b}_f\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -2 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix} -1 \\ 3 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix} 0 \\ 2.0595 \end{pmatrix}\right) = \begin{pmatrix} 0.5 \\ 0.9119 \end{pmatrix}$$

$$\boldsymbol{o}_2 = \sigma_{hard}(\mathbf{W}_{xo}\boldsymbol{x}_2 + \mathbf{W}_{ho}\boldsymbol{a}_1 + \boldsymbol{b}_o)$$
$$= \sigma_{hard}\left(\begin{pmatrix} -1 & 2 & -2 \\ 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} 0 \\ 0.0595 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \end{pmatrix}\right)$$
$$= \sigma_{hard}\left(\begin{pmatrix} 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 0.0595 \\ 0.1190 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \end{pmatrix}\right) = \sigma_{hard}\left(\begin{pmatrix} -0.9405 \\ 1.1190 \end{pmatrix}\right) = \begin{pmatrix} 0.3119 \\ 0.7238 \end{pmatrix}$$

$$\boldsymbol{g}_2 = \tanh(\mathbf{W}_{xc}\boldsymbol{x}_2 + \mathbf{W}_{hc}\boldsymbol{a}_1 + \boldsymbol{b}_c) = \tanh(\begin{pmatrix}1 & -1 & 0\\ 0 & 2 & 1\end{pmatrix}\begin{pmatrix}0\\1\\1\end{pmatrix} + \begin{pmatrix}2 & 0\\ -1 & 1\end{pmatrix}\begin{pmatrix}0\\0.0595\end{pmatrix} + \begin{pmatrix}-1\\2\end{pmatrix})$$

$$= \tanh(\begin{pmatrix}-1\\3\end{pmatrix} + \begin{pmatrix}0\\0.0595\end{pmatrix} + \begin{pmatrix}-1\\2\end{pmatrix}) = \begin{pmatrix}\tanh(-2)\\ \tanh(5.0595)\end{pmatrix} = \begin{pmatrix}-0.9640\\0.9999\end{pmatrix}$$

$$\boldsymbol{c}_2 = \boldsymbol{f}_2 \odot \boldsymbol{c}_1 + \boldsymbol{i}_2 \odot \boldsymbol{g}_2 = \begin{pmatrix}0.5\\0.9119\end{pmatrix} \odot \begin{pmatrix}0\\0.6854\end{pmatrix} + \begin{pmatrix}1\\0.3119\end{pmatrix} \odot \begin{pmatrix}-0.9640\\0.9999\end{pmatrix} = \begin{pmatrix}-0.9640\\0.9369\end{pmatrix}$$

$$\boldsymbol{a}_2 = \boldsymbol{o}_2 \odot \tanh(\boldsymbol{c}_2) = \begin{pmatrix}0.3119\\0.7238\end{pmatrix} \odot \tanh\left(\begin{pmatrix}-0.9640\\0.9369\end{pmatrix}\right) = \begin{pmatrix}-0.2327\\0.5311\end{pmatrix}$$

The hidden layer activation at time $t = 2$ is calculated to be $\boldsymbol{a}_2 = \begin{pmatrix}-0.2327\\0.5311\end{pmatrix}$.

The output at time time $t = 1$ is calculated as follows:

$$\hat{\boldsymbol{y}}_2 = \mathrm{ReLU}(\mathbf{W}_{yh}\boldsymbol{a}_2 + \boldsymbol{b}_y) = \mathrm{ReLU}\left(\begin{pmatrix}2 & -1\\0 & 3\\1 & -1\\4 & 0\end{pmatrix}\begin{pmatrix}-0.2327\\0.5311\end{pmatrix} + \begin{pmatrix}2\\0\\-1\\1\end{pmatrix}\right) =$$

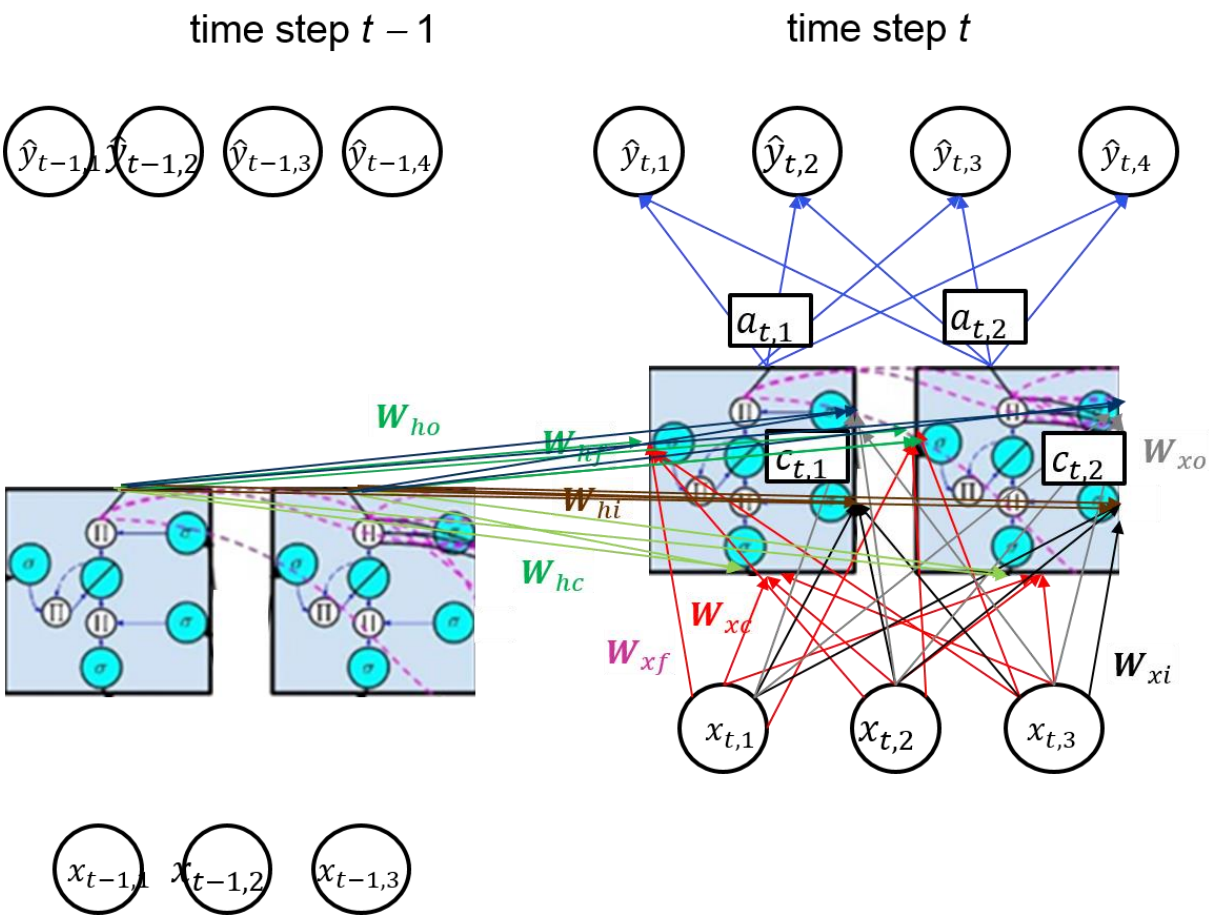
$$\mathrm{ReLU}\left(\begin{pmatrix}-0.9965\\1.5933\\-0.7638\\-0.9308\end{pmatrix} + \begin{pmatrix}2\\0\\-1\\1\end{pmatrix}\right) = \mathrm{ReLU}\left(\begin{pmatrix}1.0035\\1.5933\\-1.7638\\0.0692\end{pmatrix}\right) = \begin{pmatrix}1.0035\\1.5933\\0\\0.0692\end{pmatrix}$$

Therefore, the network output at time $t = 2$ is $\hat{\boldsymbol{y}}_2 = \begin{pmatrix}1.0035\\1.5933\\0\\0.0692\end{pmatrix}$.

To summarize, the following results are obtained:

time	LSTM cell state $\boldsymbol{c}_t$	Hidden LSTM activation $\boldsymbol{a}_t$	Network output $\hat{\boldsymbol{y}}_t$
$t = 1$	$\begin{pmatrix}0\\0.6854\end{pmatrix}$	$\begin{pmatrix}0\\0.0595\end{pmatrix}$	$\begin{pmatrix}1.9405\\0.1785\\0\\1\end{pmatrix}$
$t = 2$	$\begin{pmatrix}-0.9640\\0.9369\end{pmatrix}$	$\begin{pmatrix}-0.2327\\0.5311\end{pmatrix}$	$\begin{pmatrix}1.0035\\1.5933\\0\\0.0692\end{pmatrix}$

A diagram of the LSTM network is shown on the next page. Please note that the synaptic connections have only been shown for time t, together with the recurrent connections from time $t - 1$ to t.



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.

Solution:

The LSTM consists of a single input unit, i.e. input at time *t* is a single number, the pre-processed shampoo sales (see below) at time *t*, a single hidden layer consisting of 4 LSTM units and a single output unit predicting a single number, for which the pre-processing steps are then inverted to give the predicted shampoo sales at time *t* + 1.
Denote the original shampoo sales at time *t* as *s*(*t*).

Preprocessing:

First, instead of the raw values, differences (in this case between two consecutive time steps, i.e., interval = 1) are considered (function **`difference`** in **`exercise3b.py`**):
$d(t) = s(t) - s(t-1)$

Then, supervised data are created by assigning to each difference value $d(t)$ the subsequent difference value $d(t)$ (function **`timeseries_to_supervised`** with lag = 1). The last 12 values (i.e., the sales of the last year of the data) are used as test data, the remaining values (i.e., the first two years) as training data.
The difference values of the training data (both input and output) are scaled to the interval $[-1, 1]$; therefore, the smallest training difference value is -1, the largest difference value is +1. Note that the scaling function is estimated on the training data and then applied to the test data, i.e., the smallest test difference value can be smaller than -1 and the largest test difference value can be larger than +1 (as is the case for this data set). Since the LSTM output activation $\boldsymbol{a}_t$ is defined by a tanh with values in $[-1, 1]$ (and the dense output weights are trained on data which are limited to that range), it cannot be expected that input values outside $[-1, 1]$ can be predicted correctly; a potential solution could be to use a more clever scaling of the data.

LSTM training:

As stated above, the LSTM network consists of a single input unit (which receives the scaled difference value at time *t*), a single hidden layer consisting of 4 LSTM units and a single output unit which predicts the scaled difference value at time *t* + 1. The activation functions of the LSTM units are the tanh and hard-sigmoid functions (see above). The activations of the 4 hidden LSTM units are fully connected ("dense") to the output unit, which has a linear activation function (see the **`get_config()`** method in Keras). This LSTM network is trained on the scaled training data (see the **`fit_LSTM`** function in **`exercise3b.py`**).

Keras expects the training data in the format of a 3-dimensional tensor with shape (batch_size, #timesteps, #features). Here, # features is 1, and the batch size is also set to 1. Interestingly, the number of time steps is also set to 1, meaning that the network is unfolded to a single time step only. However, the "**`stateful`**" parameter is set to "true", such that the activation at the end of a batch (i.e. after processing each element from the input time sequence) is used to initialize the next input. This makes only sense if the sequence of input samples is ordered (i.e., not shuffled). Therefore, the entire training sequence is used in forward propagation. Backward propagation, however, seems to use a single time step only (which may not make too much sense).

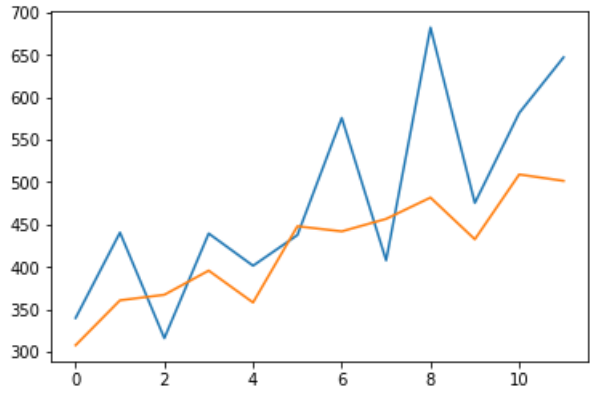
Predictions:

After training, the LSTM network can be used to predict the sale of the next month based on the sale of the current month. In particular, the scaled difference sale is input to the model, and a scaled difference sale is predicted (please note the remark regarding preprocessing above). The predicted scaled difference sale is then scaled using the inverse scaling transformation estimated in training. Afterwards, the absolute sale is computed from this difference sale, which is the output of the network. Note that the prediction of the LSTM network for the subsequent month is not the current prediction (for the current month), but the actual true prediction for the current month. A different evaluation scenario (where a prediction is used to calculate the next prediction) could be constructed.

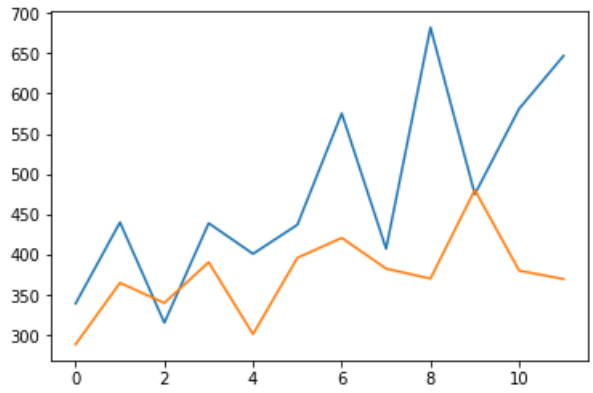
Due to different random initializations of network weights, different training runs result in different outcomes. To investigate the influence of parameters, we therefore average over four training runs.

Here are the outcomes for four training runs with standard parameters:

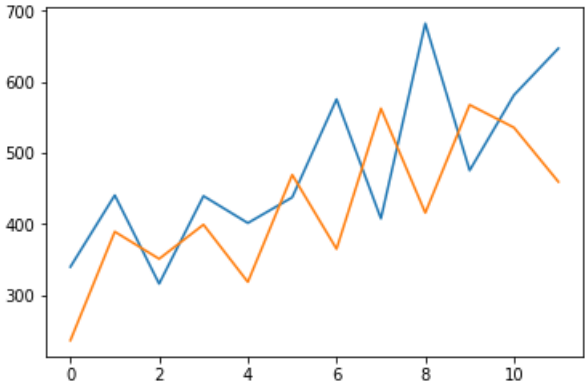
```
Month=1, Predicted=307.617829, Expected=339.700000
Month=2, Predicted=360.529975, Expected=440.400000
Month=3, Predicted=367.070183, Expected=315.900000
Month=4, Predicted=395.553830, Expected=439.300000
Month=5, Predicted=357.947693, Expected=401.300000
Month=6, Predicted=447.805232, Expected=437.400000
Month=7, Predicted=441.785465, Expected=575.500000
Month=8, Predicted=456.486775, Expected=407.600000
Month=9, Predicted=481.610897, Expected=682.000000
Month=10, Predicted=432.425494, Expected=475.300000
Month=11, Predicted=508.856209, Expected=581.300000
Month=12, Predicted=501.295139, Expected=646.900000
Test RMSE: 92.485
```



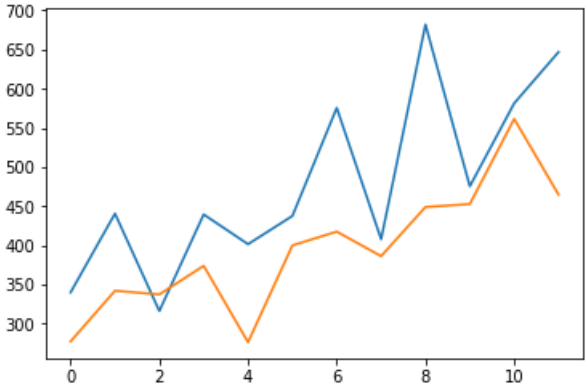

```
Month=1, Predicted=289.103660, Expected=339.700000
Month=2, Predicted=365.371617, Expected=440.400000
Month=3, Predicted=340.406122, Expected=315.900000
Month=4, Predicted=390.896310, Expected=439.300000
Month=5, Predicted=301.760642, Expected=401.300000
Month=6, Predicted=396.340573, Expected=437.400000
Month=7, Predicted=421.090896, Expected=575.500000
Month=8, Predicted=382.941611, Expected=407.600000
Month=9, Predicted=370.604889, Expected=682.000000
Month=10, Predicted=480.514345, Expected=475.300000
Month=11, Predicted=380.399994, Expected=581.300000
Month=12, Predicted=370.142176, Expected=646.900000
Test RMSE: 147.515
```



```
Month=1, Predicted=236.283877, Expected=339.700000
Month=2, Predicted=389.110425, Expected=440.400000
Month=3, Predicted=350.902507, Expected=315.900000
Month=4, Predicted=399.031481, Expected=439.300000
Month=5, Predicted=318.349944, Expected=401.300000
Month=6, Predicted=469.121649, Expected=437.400000
Month=7, Predicted=364.832112, Expected=575.500000
Month=8, Predicted=562.343578, Expected=407.600000
Month=9, Predicted=415.667641, Expected=682.000000
Month=10, Predicted=567.542148, Expected=475.300000
Month=11, Predicted=535.546724, Expected=581.300000
Month=12, Predicted=459.327775, Expected=646.900000
Test RMSE: 132.005
```

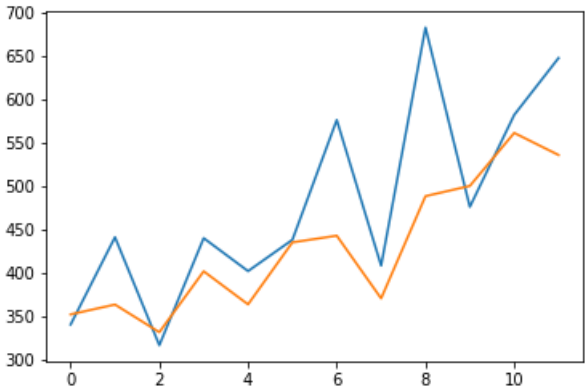



```
Month=1, Predicted=277.048288, Expected=339.700000
Month=2, Predicted=341.801691, Expected=440.400000
Month=3, Predicted=337.055216, Expected=315.900000
Month=4, Predicted=373.649205, Expected=439.300000
Month=5, Predicted=276.008283, Expected=401.300000
Month=6, Predicted=399.720811, Expected=437.400000
Month=7, Predicted=417.356965, Expected=575.500000
Month=8, Predicted=386.014490, Expected=407.600000
Month=9, Predicted=448.771867, Expected=682.000000
Month=10, Predicted=452.534953, Expected=475.300000
Month=11, Predicted=561.388796, Expected=581.300000
Month=12, Predicted=464.643055, Expected=646.900000
Test RMSE: 111.626
```

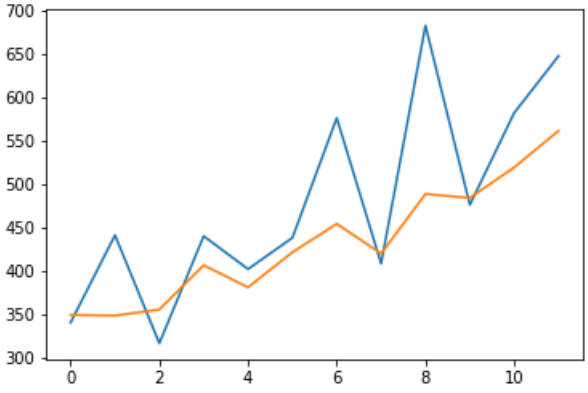


Here are another 4 runs, used to create the following table:

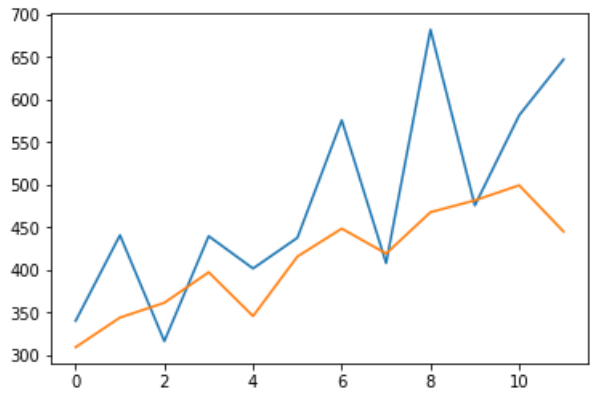
```
Month=1, Predicted=351.431687, Expected=339.700000
Month=2, Predicted=362.759781, Expected=440.400000
Month=3, Predicted=330.887229, Expected=315.900000
Month=4, Predicted=401.083292, Expected=439.300000
Month=5, Predicted=362.951946, Expected=401.300000
Month=6, Predicted=434.429900, Expected=437.400000
Month=7, Predicted=442.113748, Expected=575.500000
Month=8, Predicted=369.824232, Expected=407.600000
Month=9, Predicted=487.678792, Expected=682.000000
Month=10, Predicted=499.459529, Expected=475.300000
Month=11, Predicted=560.556339, Expected=581.300000
Month=12, Predicted=535.086074, Expected=646.900000
Test RMSE: 81.561
```



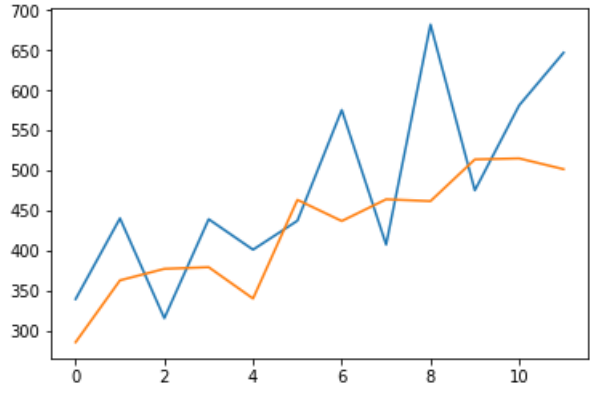

```
Month=1, Predicted=348.491192, Expected=339.700000
Month=2, Predicted=347.754788, Expected=440.400000
Month=3, Predicted=354.691124, Expected=315.900000
Month=4, Predicted=405.780926, Expected=439.300000
Month=5, Predicted=380.301420, Expected=401.300000
Month=6, Predicted=420.735112, Expected=437.400000
Month=7, Predicted=453.497750, Expected=575.500000
Month=8, Predicted=419.172899, Expected=407.600000
Month=9, Predicted=487.863057, Expected=682.000000
Month=10, Predicted=483.316202, Expected=475.300000
Month=11, Predicted=518.541183, Expected=581.300000
Month=12, Predicted=560.717394, Expected=646.900000
Test RMSE: 79.658
```



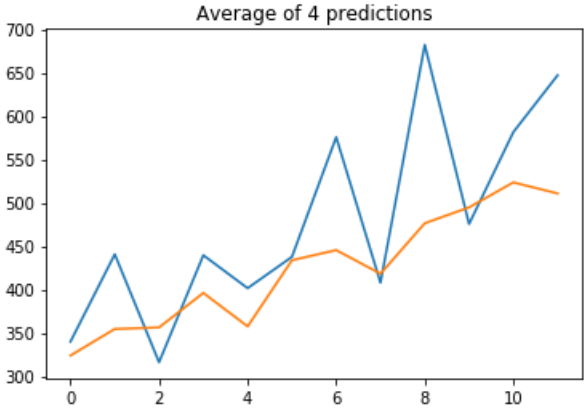
```
Month=1, Predicted=308.805498, Expected=339.700000
Month=2, Predicted=343.609009, Expected=440.400000
Month=3, Predicted=361.021203, Expected=315.900000
Month=4, Predicted=396.901144, Expected=439.300000
Month=5, Predicted=345.326456, Expected=401.300000
Month=6, Predicted=415.351005, Expected=437.400000
Month=7, Predicted=448.169191, Expected=575.500000
Month=8, Predicted=418.816764, Expected=407.600000
Month=9, Predicted=467.431479, Expected=682.000000
Month=10, Predicted=481.306621, Expected=475.300000
Month=11, Predicted=499.042685, Expected=581.300000
Month=12, Predicted=444.637444, Expected=646.900000
Test RMSE: 103.226
```




```
Month=1, Predicted=285.934202, Expected=339.700000
Month=2, Predicted=363.068711, Expected=440.400000
Month=3, Predicted=377.418197, Expected=315.900000
Month=4, Predicted=379.487674, Expected=439.300000
Month=5, Predicted=340.364450, Expected=401.300000
Month=6, Predicted=463.448448, Expected=437.400000
Month=7, Predicted=437.032611, Expected=575.500000
Month=8, Predicted=464.087331, Expected=407.600000
Month=9, Predicted=461.820576, Expected=682.000000
Month=10, Predicted=513.834426, Expected=475.300000
Month=11, Predicted=515.101704, Expected=581.300000
Month=12, Predicted=501.684119, Expected=646.900000
Test RMSE: 99.340
```



Average results:



Average RMSE: 90.946

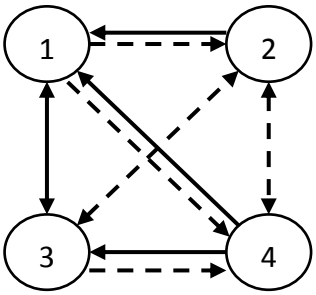
With regard to a few parameter modifications, we obtain the following results:

Configuration	Average RMSE (over 4 runs)	Remark
Number of LSTM units = 4	90.946	Baseline
Number of LSTM units = 8	133.407	... getting worse
Number of LSTM units = 2	108.952	No improvement

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 Hopfield network):

In this exercise, the dynamics in a simple Hopfield 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:

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

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.

Solution:

In synchronous dynamics, all network neurons are updated simultaneously according to

$$s_i(t+1) = \Theta(h_i(t)) = \Theta\left(\sum_{j=1}^{4} w_{ij} \cdot s_j(t) - \theta_i \right)$$

$$\text{or in matrix notation:} \quad \mathbf{s}(t+1) = \Theta\left(\mathbf{W} \cdot \mathbf{s}(t) - \boldsymbol{\theta}\right)$$

Inserting the weight matrix $\mathbf{W} = (w_{ij})$, threshold $\boldsymbol{\theta} = (\theta_i)$ and initial state $\mathbf{s}(t=0) = (0, 0, 0, 0)$ we get:

$$\mathbf{s}(1) = \Theta\left(\mathbf{W}\cdot\mathbf{s}(0) - \boldsymbol{\theta}\right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}\cdot\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} -1.5 \\ 1.5 \\ -0.5 \\ 1.5 \end{pmatrix}\right] = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathbf{s}(2) = \Theta\left(\mathbf{W}\cdot\mathbf{s}(1) - \boldsymbol{\theta}\right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}\cdot\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 2 \\ -1 \\ 0 \\ -1 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 0.5 \\ 0.5 \\ -0.5 \\ 0.5 \end{pmatrix}\right] = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathbf{s}(3) = \Theta\left(\mathbf{W}\cdot\mathbf{s}(2) - \boldsymbol{\theta}\right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 2 \\ -2 \\ 1 \\ -2 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{pmatrix}\right] = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\mathbf{s}(4) = \Theta\left(\mathbf{W}\cdot\mathbf{s}(3) - \boldsymbol{\theta}\right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 1 \\ -2 \\ 1 \\ -2 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} -0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{pmatrix}\right] = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\mathbf{s}(5) = \Theta\left(\mathbf{W}\cdot\mathbf{s}(4) - \boldsymbol{\theta}\right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}\cdot\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} 1 \\ -1 \\ 0 \\ -1 \end{pmatrix} - \begin{pmatrix} 1.5 \\ -1.5 \\ 0.5 \\ -1.5 \end{pmatrix}\right] = \Theta\left[\begin{pmatrix} -0.5 \\ 0.5 \\ -0.5 \\ 0.5 \end{pmatrix}\right] = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

We see that the network assumes assumes different network states in the first 4 time steps until it reaches network state $\mathbf{s}(1)$ after at $t = 5$; thus, the same cycle repeats itself and so the network is in an oscillatory state (4-cycle).

Note that the synaptic matrix is asymmetric; if it was symmetric, the network must have reached a stationary state or a 2-cycle!

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

Solution:

In asynchronous dynamics, only a single network neuron is updated according to

$$s_i(t+1) = \Theta(h_i(t)) = \Theta\left(\sum_{j=1}^{4} w_{ij} \cdot s_j(t) - \theta_i \right)$$

while the other neurons keep their state.

Inserting the weight matrix $\mathbf{W} = (w_{ij})$, threshold $\boldsymbol{\theta} = (\theta_i)$ and initial state $\mathbf{s}(t=0) = (0, 0, 0, 0)$ we get:

Time step $t = 1$: only neuron 1 is updated:

$$s_1(1) = \Theta(h_1(0)) = \Theta\left(\sum_{j=1}^{4} w_{1j} \cdot s_j(0) - \theta_1 \right) = \Theta\left[\begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 1.5 \right] = \Theta[0 - 1.5] = \Theta[-1.5] = 0$$

Network state at $t = 1$: $\mathbf{s}(t=1) = (0, 0, 0, 0)$

Time step $t = 2$: only neuron 2 is updated, using network state $\mathbf{s}(t=1)$

$$s_2(2) = \Theta(h_2(1)) = \Theta\left(\sum_{j=1}^{4} w_{2j} \cdot s_j(1) - \theta_2 \right) = \Theta\left[\begin{pmatrix} -1 & 0 & -1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 1.5 \right] = \Theta[0 + 1.5] = \Theta[1.5] = 1$$

Network state at $t = 2$: $\mathbf{s}(t=2) = (0, 1, 0, 0)$

Time step $t = 3$: only neuron 3 is updated, using network state $\mathbf{s}(t=2)$

$$s_3(3) = \Theta(h_3(2)) = \Theta\left(\sum_{j=1}^{4} w_{3j} \cdot s_j(2) - \theta_3 \right) = \Theta\left[\begin{pmatrix} 1 & -1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} - 0.5 \right] = \Theta[-1 - 0.5] = \Theta[-1.5] = 0$$

Network state at $t = 3$: $\mathbf{s}(t=3) = (0, 1, 0, 0)$

Time step $t = 4$: only neuron 4 is updated, using network state $\mathbf{s}(t=3)$

$$s_4(4) = \Theta(h_4(3)) = \Theta\left(\sum_{j=1}^{4} w_{4j} \cdot s_j(3) - \theta_4 \right) = \Theta\left[\begin{pmatrix} -1 & -1 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + 1.5 \right] = \Theta[-1 + 1.5] = \Theta[0.5] = 1$$

Network state at $t = 4$: $\mathbf{s}(t=4) = (0, 1, 0, 1)$

Time step $t = 5$: only neuron 1 is updated:

$$s_1(5) = \Theta(h_1(4)) = \Theta\left(\sum_{j=1}^{4} w_{1j} \cdot s_j(4) - \theta_1\right) = \Theta\left[\begin{pmatrix}0 & 1 & 1 & 1\end{pmatrix} \cdot \begin{pmatrix}0\\1\\0\\1\end{pmatrix} - 1.5\right] = \Theta[2 - 1.5] = \Theta[0.5] = 1$$

Network state at $t = 5$: $\mathbf{s}(t=5) = (1, 1, 0, 1)$

Time step $t = 6$: only neuron 2 is updated, using network state $\mathbf{s}(t=5)$

$$s_2(6) = \Theta(h_2(5)) = \Theta\left(\sum_{j=1}^{4} w_{2j} \cdot s_j(5) - \theta_2\right) = \Theta\left[\begin{pmatrix}-1 & 0 & -1 & -1\end{pmatrix} \cdot \begin{pmatrix}1\\1\\0\\1\end{pmatrix} + 1.5\right] = \Theta[-2 + 1.5] = \Theta[-0.5] = 0$$

Network state at $t = 6$: $\mathbf{s}(t=6) = (1, 0, 0, 1)$

Time step $t = 7$: only neuron 3 is updated, using network state $\mathbf{s}(t=6)$

$$s_3(3) = \Theta(h_3(2)) = \Theta\left(\sum_{j=1}^{4} w_{3j} \cdot s_j(2) - \theta_3\right) = \Theta\left[\begin{pmatrix}1 & -1 & 0 & 1\end{pmatrix} \cdot \begin{pmatrix}1\\0\\0\\1\end{pmatrix} - 0.5\right] = \Theta[2 - 0.5] = \Theta[1.5] = 1$$

Network state at $t = 7$: $\mathbf{s}(t=7) = (1, 0, 1, 1)$

Time step $t = 8$: only neuron 4 is updated, using network state $\mathbf{s}(t=7)$

$$s_4(4) = \Theta(h_4(3)) = \Theta\left(\sum_{j=1}^{4} w_{4j} \cdot s_j(3) - \theta_4\right) = \Theta\left[\begin{pmatrix}-1 & -1 & -1 & 0\end{pmatrix} \cdot \begin{pmatrix}1\\0\\1\\1\end{pmatrix} + 1.5\right] = \Theta[-2 + 1.5] = \Theta[-0.5] = 0$$

Network state at $t = 8$: $\mathbf{s}(t=8) = (1, 0, 1, 0)$

So far, no stationary state is reached.

Note: This is again due to the asymmetric weight matrix. If the weight matrix was symmetric, a stationary state must be reached in asynchronous dynamics!

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)

Solution:

From the weight matrix **W**, we can assign the three weight vectors corresponding to the three output neurons as follows:

Weight vector of output neuron 1: $\mathbf{w}_1 = (w_{11}, w_{12}) = (1/2, 2/3)$

Weight vector of output neuron 2: $\mathbf{w}_2 = (w_{21}, w_{22}) = (1/3, 1/6)$

Weight vector of output neuron 3: $\mathbf{w}_3 = (w_{31}, w_{32}) = (2/3, 1/3)$

Each output neuron computes the Euclidean distance between the input pattern and the weight vector:

$$d(\mathbf{w}_i, \mathbf{x}) = \|\mathbf{w}_i - \mathbf{x}\| = \sqrt{\sum_{j=1}^{n} (w_{ij} - x_j)^2}$$

Then, it assigns the winner neuron according to

$$i^* = \arg\min_{i=1,\ldots,m} d(\mathbf{w}_i, \mathbf{x}) \;\Rightarrow\; y_i = \begin{cases} 1 & \text{if } i = i^* \\ 0 & \text{else} \end{cases}$$

Consider the first input $\mathbf{x} = (x_1, x_2) = (1/2, 1/3)$.

We have

$$d(\mathbf{w}_1, \mathbf{x}) = \left\| \begin{pmatrix} 1/2 \\ 2/3 \end{pmatrix} - \begin{pmatrix} 1/2 \\ 1/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 0 \\ 1/3 \end{pmatrix} \right\| = \sqrt{0 + (1/3)^2} = 1/3$$

$$d(\mathbf{w}_2, \mathbf{x}) = \left\| \begin{pmatrix} 1/3 \\ 1/6 \end{pmatrix} - \begin{pmatrix} 1/2 \\ 1/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} -1/6 \\ -1/6 \end{pmatrix} \right\| = \sqrt{(1/6)^2 + (1/6)^2} = \sqrt{2}/6$$

$$d(\mathbf{w}_3, \mathbf{x}) = \left\| \begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix} - \begin{pmatrix} 1/2 \\ 1/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 1/6 \\ 0 \end{pmatrix} \right\| = \sqrt{(1/6)^2 + 0} = 1/6$$

From $i^* = \arg\min_{i=1,\ldots,3} d(\mathbf{w}_i, \mathbf{x})$ and $d(\mathbf{w}_3, \mathbf{x}) < d(\mathbf{w}_2, \mathbf{x}) < d(\mathbf{w}_1, \mathbf{x})$ we see that

the winner neuron for input $\mathbf{x} = (x_1, x_2) = (1/2, 1/3)$ is output neuron 3.

Now consider the second input $\mathbf{x} = (x_1, x_2) = (1/3, 2/3)$.

We have

$$d(\mathbf{w}_1, \mathbf{x}) = \left\| \begin{pmatrix} 1/2 \\ 2/3 \end{pmatrix} - \begin{pmatrix} 1/3 \\ 2/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 1/6 \\ 0 \end{pmatrix} \right\| = \sqrt{(1/6)^2 + 0} = 1/6$$

$$d(\mathbf{w}_2, \mathbf{x}) = \left\| \begin{pmatrix} 1/3 \\ 1/6 \end{pmatrix} - \begin{pmatrix} 1/3 \\ 2/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 0 \\ -1/2 \end{pmatrix} \right\| = \sqrt{(0 + (1/2)^2} = 1/2$$

$$d(\mathbf{w}_3, \mathbf{x}) = \left\| \begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix} - \begin{pmatrix} 1/3 \\ 2/3 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 1/3 \\ -1/3 \end{pmatrix} \right\| = \sqrt{(1/3)^2 + (1/3)^2} = \sqrt{2}/3$$

From $i^* = \arg\min_{i=1,\ldots,3} d(\mathbf{w}_i, \mathbf{x})$ and $d(\mathbf{w}_1, \mathbf{x}) < d(\mathbf{w}_3, \mathbf{x}) < d(\mathbf{w}_2, \mathbf{x})$ we see that

the winner neuron for input $\mathbf{x} = (x_1, x_2) = (1/3, 2/3)$ is output neuron 1.

Thus, the competitive network assigns the first input to output neuron 3 and the second input to output neuron 1.

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

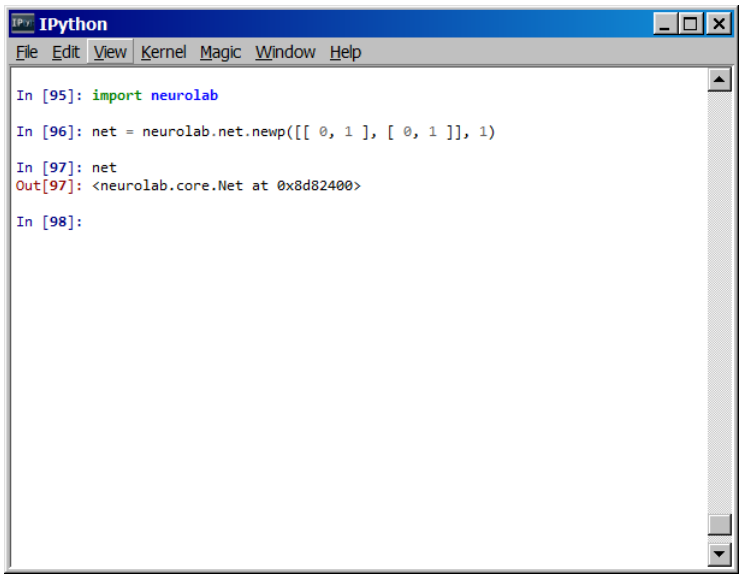
2.) From the start menu, start an Jupyter QTconsole (Anaconda3 → Jupyter QTConsole)

    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

Solution:

"Real centers" are the centers from which data "clouds" are generated in the python script **`exercise5b_generate_input_samples.py`**; i.e. they are the "real" cluster centers. As such they are the "unknown" targets of the (unsupervised) training algorithm.
"Train centers" are the outcome of the network training, which are the vectors of weights from the input to the output layer. There are 4 output neurons – corresponding to the 4 data clusters –, and each output neuron has two weight values corresponding to the connection to the two input neurons. Since each output neuron computes the distance between the input and the weight vector, the weight vector can be interpreted as a "prototype" or center vector of this neuron: if an input corresponding to the weight vector is applied, the output (distance) of the corresponding neuron to this input is 0 and this neuron is the "winner" neuron. Thus, the training is successful if it generates 4 prototype vectors close to the real data centers.

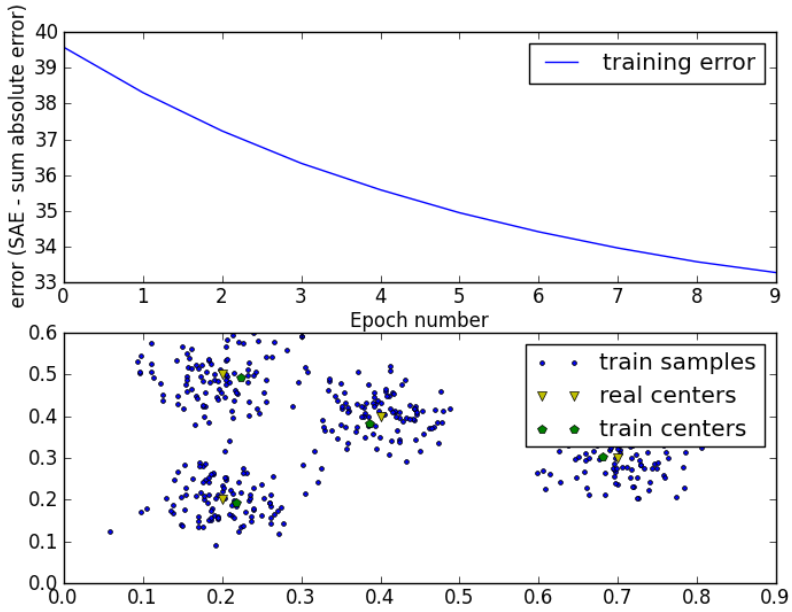
Real data centers:
Class 1: (0.2, 0.2)
Class 2: (0.4, 0.4)
Class 3: (0.7, 0.3)
Class 4: (0.2, 0.5)

Final weights after training:

```
Weights of output neuron 0: [ 0.21796751  0.19130351]
Weights of output neuron 1: [ 0.38616744  0.38297298]
Weights of output neuron 2: [ 0.222978    0.49103367]
Weights of output neuron 3: [ 0.68049574  0.30269889]
```

Thus, there is a reasonable correspondence between the final weight vectors and the real data centers. Note, however, that the order of the weight vectors does not correspond to the order of the real data centers (the training is unsupervised).
Training is very fast (note the low number of iterations).
It even works for homogenous weight initialization (initially, all weights are set to 0.5). This is due to a random component in the training algorithm, where a random "epsilon" is added to the initial weights. If you run the algorithm several times, you can notice that the outcome of the training – especially the order of the resulting weight vectors – is generally different from training run to training run.

A screenshot of the training run is provided below.



What is the meaning of the training error in an *unsupervised* learning task?

The answer is obtained by inspecting the relevant implementation of the error function in the corresponding neurolab module **`wta.py`**: Since the goal of competitive learning is to move the weight vector into the direction of the input vector **x**, the training error of any input sample **x** is defined as the difference between **x** and the weight vector of the winning neuron for this input vector **x**. The SAE (sum absolute error) is then the sum of the absolute value of this difference over all training samples.

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.

Solution:

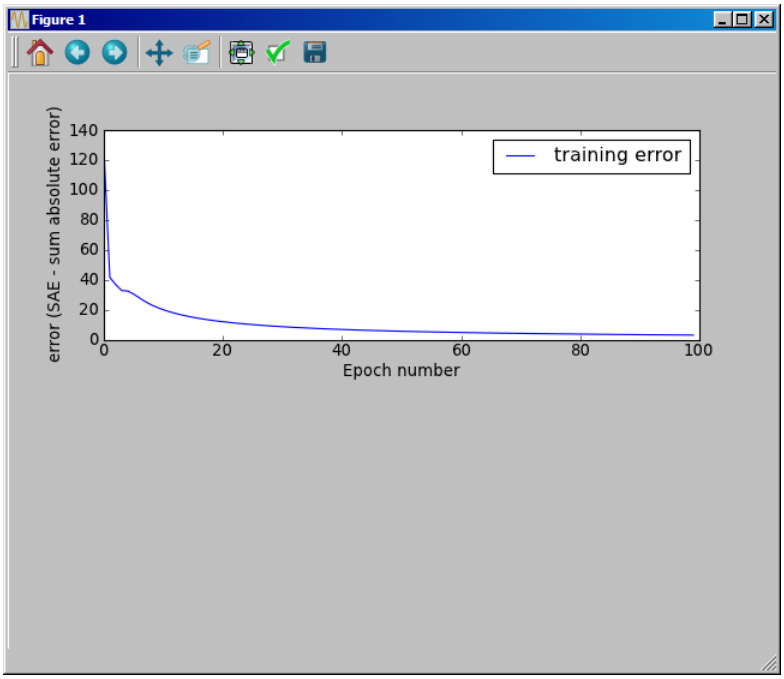
Since the input signals are provided as 7 × 6 matrix, transformed to a single row of input values, we have 42 input values and consequently need 42 input neurons. Since we have to classify 10 digits, we need 10 output neurons. Providing these numbers in the script **`exercise5c.py`** and running the script leads e.g. to the following output:

```
test on training set:
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]]
```

This is a matrix of 10 lines – corresponding to the 10 input patterns – and 10 columns – corresponding to the 10 output neurons. Since we have a competitive network, there must be a single 1 in each output line. The 1 corresponds to the classification result: If the 1 happens to be in column 5, it means that output neuron 5 is the winner neuron and the input pattern is classified as the prototype (weight vector) corresponding to output neuron 5. Note that we do not have influence on the order of the learned digits, i.e. it can happen that the prototype of output neuron 5 actually corresponds to the digit "3". Then, output neuron 5 has learned to classify the digit "3". Therefore, we cannot count the training errors just from the results on the training set. However, we can see whether two input samples have been assigned the same output value: this would be the case if two output lines would be identical (i.e. if there is more than a single 1 in a *column*). Since we know that each input pattern is different, this would definitely be a mistake.

In the above example, the training was successful, since in each column (and in each row) of the output matrix there is exactly a single 1.

Plot of training run:



However, not every training run is successful:

Here is the result of a second run:

```
test on training set:
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]]
```

In this training run, the last two lines are identical and so the last two input patterns are assigned the same output result. Since the input patterns were different, however, there is a misclassification.

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).

Solution:

Here is an example for the output of a training run:

```
test on training set:
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]]

test on independent, noisy test set:
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]]
```

We see that the output of the test on the noisy test set is exactly identical to the output of the test on the training set. Therefore, the noisy digits have been classified exactly as the training digits. Since we know that the order of the input digits has not changed, this means that there is no classification error in this example.