technische universität
dortmund

Dept. of Computer Science
Chair of Algorithm Engineering

Tutorial for

Introduction to Computational Intelligence in Winter 2015/16

Günter Rudolph, Vanessa Volz
Lecture website: `https://tinyurl.com/CI-WS2015-16`

Sheet 1, Block I

22 October 2015

Due date: 11 November 2015, 2pm
Discussion: 12/13 November 2015

Note

These are only shortened versions of the expected answers for future reference and to help recall the tutorial. There may be alternative solutions for many tasks.

Exercise 1.1: McCulloch-Pitts Neural Network (3 Points)

Consider the McCulloch-Pitts ANN depicted in figure 1. Complete the truth table by specifying, for all possible input combinations of x_1, x_2 and x_3, the output of each neuron(A-F) and the output of the network f.

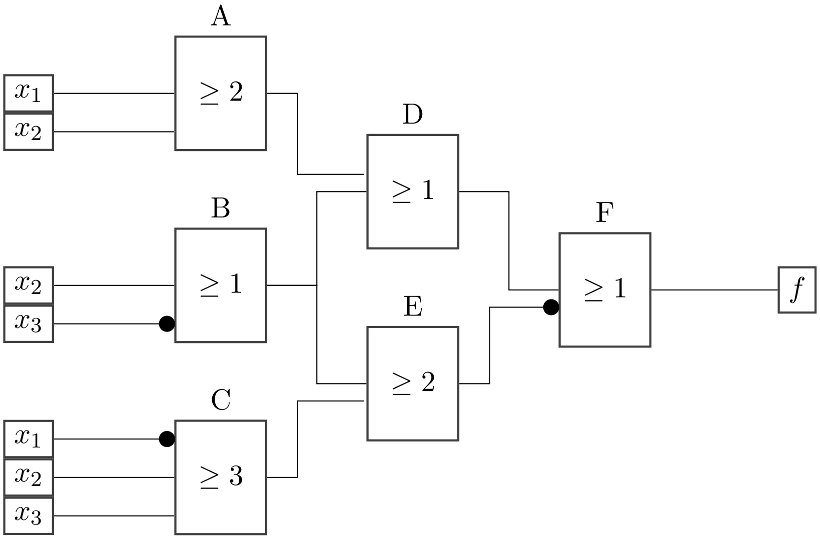

Figure 1: McCulloch-Pitts ANN (Exercise 1.1)

Solution

x1	x2	x3	A	B	C	D	E	F	f
0	0	0	0	1	0	1	0	1	1
1	0	0	0	1	0	1	0	1	1
0	1	0	0	1	0	1	0	1	1
0	0	1	0	0	0	0	0	1	1
1	1	0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1	1	1
1	0	1	0	0	0	0	0	1	1
1	1	1	1	1	0	1	0	1	1

Table 1: truth table

Exercise 1.2: Perceptron Classification (3 Points)
A class $C \subset \mathbb{R}^2$ contains all points marked yellow in figure 2. Please note that the area is only bounded at the black lines.

Manually construct an ANN of perceptrons that decides whether an input vector $(x, y)^T \in \mathbb{R}^2$ belongs to C (i.e. is within the marked area) or not. State the reasons for your choice of weights and network structure.

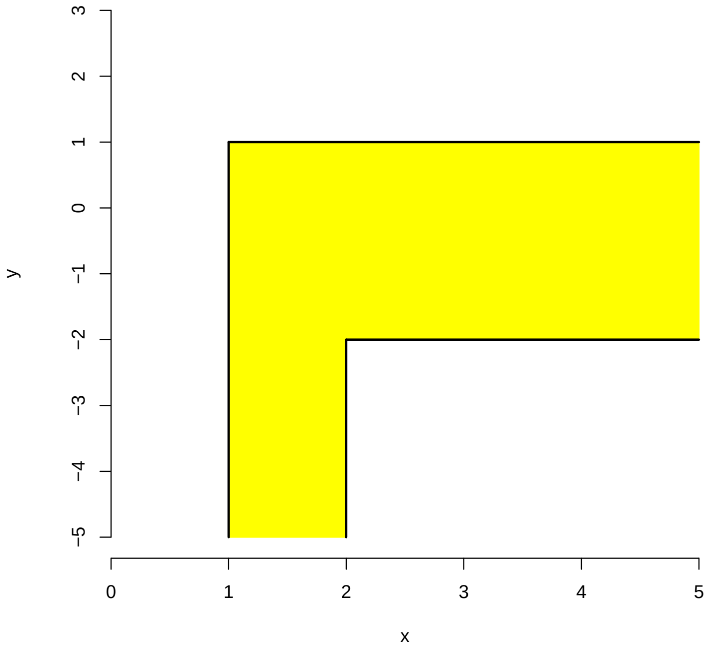

Figure 2: Construct ANN that identifies the marked area (Exercise 1.2)

Solution
Possible logical expression for network: $((x \geq 1) \wedge (-y \geq -1)) \wedge ((-x \geq -2) \wedge (y \geq -2))$

Exercise 1.3: Single-Layer Perceptron - Gradient method (8 Points)
Script ci1516-3.R implements the gradient method for batch learning in a Single-Layer Perceptron (compare lecture 2). Implement the functions `gradient(w,x,y)` and `err(w,x,y)` that are missing in the code. Function `err(w,x,y)` returns the number of classification errors made for input `x` with label vector `y` and the perceptron weights `w`. Using the same parameters, function `gradient(w,x,y)` calculates the gradient to update the weights of the perceptron.

Additionally:

1. Test the implementation on the data in files `set1.txt` and `set2.txt` a few times and with different learning rates g. Describe and explain any trends in the graph that plots errors in each iteration (titled Errors).
2. Lines 65 and 66 in the code calculate the values passed to the function `panel.abline(a,b ,...)` that adds a line of the form $y = a + b * x$ to the plot titled Perceptron. This line represents the separation threshold defined by the learned weights of the perceptron. Explain why.
3. In line 23 in the code, the weights of the perceptron are normalised. Explain the effect of the normalisation on the number of errors and necessary iterations.

Solution

1. Short description of observations

- set1
  - $g = 0.1$: Downward trend, non-monotonous
  - $g = 1$: Oscillation. Reason: step size too large
  - $g = 0.01$: Downward trend, but more iterations than $g = 0.1$. Reason: step-size could be larger
- set2
  - Number of errors jump for all g, only different size of jumps
  - Reason: data in set2 not linearly seperable

2. For the seperation threshold, it holds that $w_1 + w_2x_1 + w_3x_2 = 0$, with $w_1 = -\theta$. Thus:

$$\begin{aligned} & w_1 + w_2x_1 + w_3x_2 = 0 \\ \Leftrightarrow & x_2 = \frac{-w_1}{w_3} + \frac{-w_2}{w_3}x_1 \\ \Leftrightarrow & a = \frac{-w_1}{w_3} \wedge b = \frac{-w_2}{w_3} \end{aligned}$$

3. Normalisation prevents a weight vector with high absolute values, which would slow down learning because the update is the sum of (comparably) low absolute values (true for $g = 0.1$). Not normalising can reverse the effect of a learning rate that is too low ($g = 0.001$).

Exercise 1.4: Multi-Layer Perceptron - Backpropagation (6 Points)
Script ci1516-4.R implements the backpropagation algorithm introduced in the lecture (compare lecture 2) for the ANN depicted in figure 3. Assume $\theta = 0$ for all neurons.

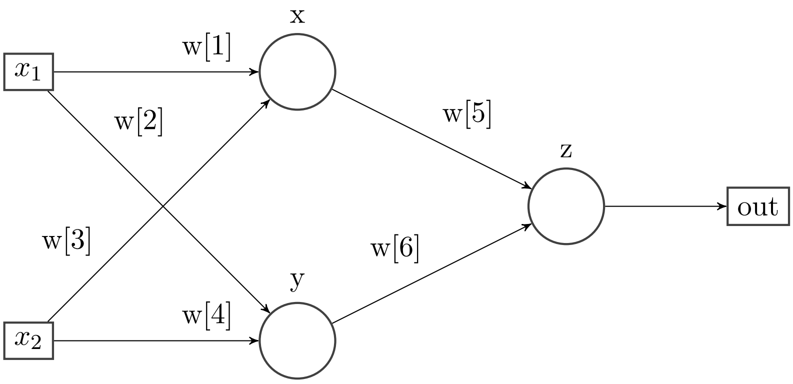

Figure 3: ANN structure (Exercise 1.4)

1. Implement the weight updates as suggested in line 22 in the script.
2. How would the computation of δ_z change if, instead of the function $a(x) = \frac{1}{1+e^{-x}}$, the function $a(x) = \tanh(x)$ was used?
3. Describe and explain the trend in the graph that plots errors in each iteration (titled Errors).

Important: Note that the implementation is only supposed to work for the specific number of neurons and network structure in figure 3. There is no need for a generalised implementation.

Solution

1. See script

2. The gradient changes. It holds that if $a(x) = \tanh(x)$, $a'(x) = (1 - a^2(x))$. Therefore, the derivative on slide 25, lecture 2 changes such that:

$$\begin{aligned}\frac{\delta f(w, u; x, z^*)}{\delta u_{jk}} &= 2[a(u_k' y) - z_k^*] \cdot a'(u_k' y) \cdot y_j \\ &= 2[a(u_k' y) - z_k^*] \cdot (1 - a^2(u_k' y)) \cdot y_j \\ &= 2[z_k - z_k^*] \cdot (1 - z_k^2) \cdot y_j = \delta_z \cdot y_j\end{aligned}$$

3. Error decreases, this is expected due to the combination of gradient descent and online learning. Derivative decreases, too, due to smaller error signals.