**Institute for Computer Science VI, Autonomous Intelligent Systems, University of Bonn**

**Dr. N. Goerke**
**Friedrich-Ebert-Allee 144, 53113 Bonn, Tel: +49 228 73-4167**
**E-Mail: goerke_at_ais.uni-bonn.de**
**http://www.ais.uni-bonn.de/WS1718/4204_L_NN.html**

# Exercises for module Technical Neural Networks (MA-INF 4204), WS1718

# Exercises sheet 3, due: Monday 30.10.2017

23.10.2017

| Group | Name | 15 | 16 | 17 | 18 | 19 | $\sum$ Sheet 3 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |

## Assignment 15 (3 Points)

For some reason, a special computer is not capable of calulating the exponential function, the hyperbolic tangent or the trigonometric functions (sin, cos, ...) in acceptable time. Your task is to develop an alternative transferfunction $g(z)$ which shape is almost like the hyperbolic tangent $tanh(z)$, (maximal deviation below 0.05) but uses less computing power.
Give a formula for $g(z)$ and draw it together with $tanh(z)$.

## Assignment 16 (3 Points)

A N-H1-M MLP with tanh as transferfunction, in hidden-layer and ouput-layer is given, the input values are $x_n = -1.0$, all weights are $w_{i,j} = -10.0$.
How is the result $y_m$ for this network changing, if you include a second hidden layer H2 with transferfunction tanh and all new weights $w_{h,k} = -10.0$ ? (with $N, H1, H2, M > 4$).

## Assignment 17 (2 Points)

Develop and describe an algorithm that shuffles a fixed, given set of training patterns into a new random sequence (as necessary for BP training).
The algorithm shall be efficient with respect to memory and time consumption.

## Assignment 18 (3 Points)

Explain the training algorithm Backpropagation-of-Error in your own words.
Please describe how BP is used to make a neural network learn a desired task.
You are not intended to describe the mathematical derivation of the algorithm, but describe how this algorithm is structured into different phases (five to seven) with respect to a programmer who wants to implement it.

# Assignment 19 (4 Points)

Derive a new learning rule ** for a Multi-Layer-Perceptron.
Start from the new objective function (cost function, error function) $E^{**}$ and derive the new learning rule in analogy to Backpropagation of Error. The new cost function has been extended by a term that depends on all weights $w_{ij}$.
Write down all calculation steps, and give the formulas for calculating the $\delta^{**}$ in output- and hidden layer.

$$ {}^{p}E^{**} = \frac{1}{2}\sum_{m=1}^{M}({}^{p}\hat{y}_m \; - \; {}^{p}y_m)^2 + \beta\frac{1}{2}\sum_{i,j}(w_{ij})^2 $$

# Programming-Assignment PA-B (10 Points, due 6.11.2017, 2 weeks)

Implement (in C, C++, Java or Python ) a Multi-Layer-Perzeptron including the learning rule Backpropagation of error.

**Implement a Multi Layer Perzeptron**
Your program must be capable of setting the structure of the network, number of layers (maximum 4), number of neurons per layer (maximum 1000), transferfunction separately for each layer (*tanh* or *logistic* or *identity*). It is O.K. to set these parameters directly within the sourcecode, you should not implement a user interface for that.
Initialize the weights to random values between $-2.0$ and $+2.0$, make sure, that the random number generator is under your control, and that you can reproduce your results. Set/initialize the random number generator explicitly (random seed).

**Implement Backpropagation of Error**
Implement the 7 steps of the Backpropagation of Error Algorithm (from the lecture). Your program shall read training patterns (input ${}^{p}x_n$, teacher ${}^{p}\hat{y}_m$) from a file `training.dat` with up to $P = 1000$ patterns. Use the sum over quadratic differences as error function. Allow to set different learning rates $\eta$ for the different layers.
Calculate the error in every training step, and print it as a learning curve into a file `learning.curve` during the training process. Choose a format that can easily be depicted using the freely available program *gnuplot*.

End the training if a predefined number of training steps has been performed, and then test the performance of the network (no further weight changes) with respect to a second set of data, the test set `test.dat` (same file format as the training data).

The `training.dat` file starts with two lines of header, followed by $P$ lines of data.
Each header line starts with a `#` character followed by some characters and strings that you can ignore (if you want to).
Each of the P data lines contains the data for one pattern $p$: N-input values ${}^{p}x_1, \; \ldots, \; {}^{p}x_N$, separated by one or more blanks, M-teacher values ${}^{p}\hat{y}_1, \; \ldots, \; {}^{p}\hat{y}_M$, separated by blanks.
Examples will be provided on the web page.

Extra:
If you are experienced with neural networks and MLPs, you can implement Rectified Linear Units (ReLUs) using the *ramp function*.