Universität Hamburg
Department Informatik, ABe NATS, WTM & TAMS
Wolfgang Menzel, Stefan Wermter, Jianwei Zhang

Contact: Cornelius Weber
Office F-233
Tel 42883-2537

(for groups who focus on the MLP)

Exercises to the module: Algorithmisches Lernen

SS 2012 Sheet 7 (for groups who focus on the MLP)

Due: 23.05.2012

Task 7.1 Multi-Layer Perceptron with programming errors

Given data $\{\vec{x}^\alpha, y^\alpha\}$, $\alpha = 1, \ldots, 10000$. Let the components of the input vectors $\vec{x}^\alpha \in \mathbb{R}^5$ be from within the interval $[0, 100]$. The outputs $y^\alpha \in \mathbb{R}$ are within an interval $[0, 10]$ and have been produced by an unknown function with noise. The task is to train an MLP that approximates the function as well as possible.

Correct the errors in the following sketch of a solution and do necessary completions.

Proposed solution:

Definition of the network architecture: 5 inputs (layer 0), 3 hidden neurons (layer 1), 1 output neuron (layer 2). One bias neuron with activation "-1", which is connected with all neurons. Initialisation of all weights $w_{ij}^{\lambda\,\lambda-1}$ with random values in the interval [0,1]

```
BEGIN {iterate}
    all temporary weights: $\Delta w_{ij}^{\lambda\,\lambda-1} = 0$
    training error: $E^{train} = 0$
    BEGIN {for each data point}
        activations of input neurons $i$: $S_i^0 = x_i^\alpha$
        for layers $\lambda=1$ and then $\lambda=2$: $S_i^\lambda = 1/(1+\exp(-\sum_j w_{ij}^{\lambda\,\lambda-1} S_j^{\lambda-1}))$
        in layer $\lambda=2$ for neuron $i$: $\delta_i^\lambda = S_i^\lambda \cdot (1-S_i^\lambda) \cdot (y^\alpha - S_i^\lambda)^2$
        for layer $\lambda=1$: $\delta_j^{\lambda-1} = S_j^{\lambda-1} \cdot (1-S_j^{\lambda-1}) \cdot \sum_i \delta_i^\lambda w_{ij}^{\lambda\,\lambda-1}$
        for layers $\lambda=1$ und $\lambda=2$: $\Delta w_{ij}^{\lambda\,\lambda-1} = \Delta w_{ij}^{\lambda\,\lambda-1} + \delta_i^\lambda S_j^{\lambda-1}$
        with $i=1$, $\lambda=2$: $E^{train} = E^{train} + (y^\alpha - S_i^\lambda)$
    END {for each data point}
    for layers $\lambda=1$ and $\lambda=2$: $w_{ij}^{\lambda\,\lambda-1} = w_{ij}^{\lambda\,\lambda-1} + \Delta w_{ij}^{\lambda\,\lambda-1}$
END {iterate} STOP IF $\{E^{train} = 0\}$
```

— see reverse —

Wolfgang Menzel (menzel@informatik. . .)
Stefan Wermter (wermter@informatik. . .)
Jianwei Zhang (zhang@informatik. . .)

Vorlesung: Mi 8:15-11:45, F-009
Übungen: Mi 12:15-13:45, F-009

Dienstgebäude
Universität Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg

Öffentliche Verkehrsmittel
Buslinien 181, 281
Haltestelle: Informatikum

Task 7.2 Derivation of the transfer function

The logistic function is:

$$f(x) \; = \; \frac{1}{1+e^{-\beta x}}$$

Show that $\frac{\partial f(x)}{\partial x} = \beta f(x) \cdot (1 - f(x))$. How does the implementation of the backpropagation algorithm benefit from this differential equation?

Consider the following transfer function:

$$g(x) \; = \; \frac{1}{2}(\tanh(\frac{\beta}{2}x) + 1)$$

Plot both functions, $f(x)$ and $g(x)$. How do they differ? Does $g(x)$ have the same differential equation as $f(x)$?

Task 7.3 MLP and XOR-problem

Given a multi-layer perceptron with input, hidden and output layer. How many neurons in the hidden layer are needed to classify the XOR data correctly, and how many are required for the AND data?

To solve the exercise practically, there is the Python code of an MLP (`mlp.py`) on the website of the Marsland book in "Chapter 3":
`http://seat.massey.ac.nz/personal/s.r.marsland/MLBook.html`
Use the demo of the MLP on logistic functions (`logic.py`).