# Übungsblatt 9
# „Mustererkennung“

J. Cavojska, N. Lehmann, R. Toudic

14.07.2015

## Inhaltsverzeichnis

# 1 Aufgabe 1a - Trainingsmenge vs. Validierungsmenge

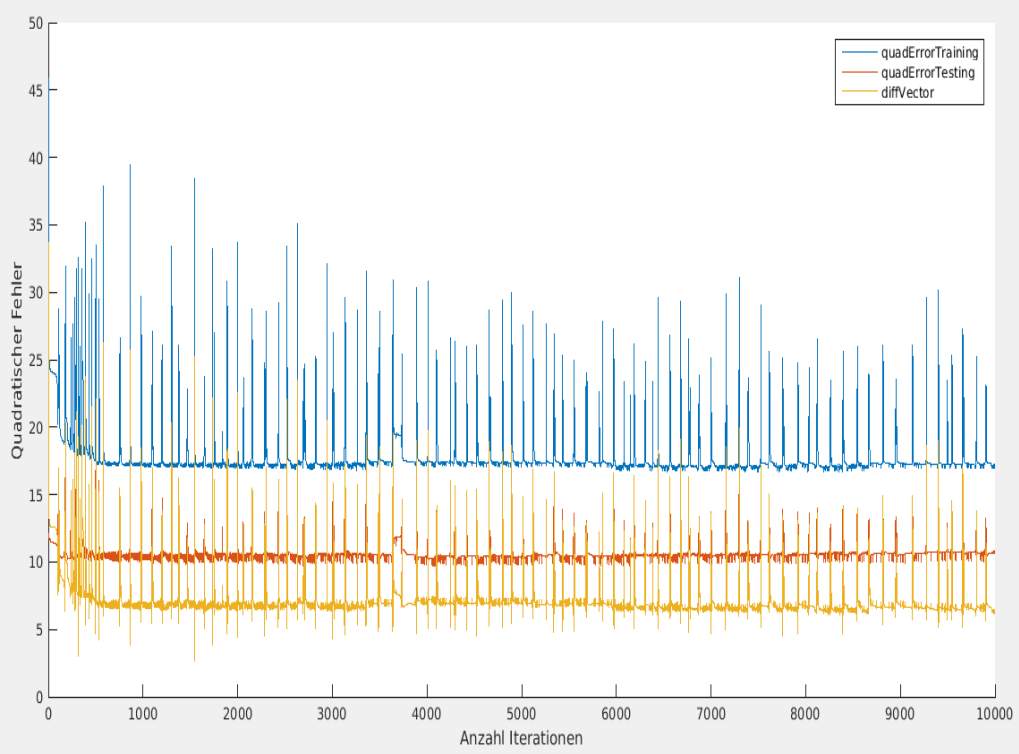
## 1.1 Code

```
Data   = load('pendigits-training.txt');

% prepare data for network
LData1 = horzcat(Data(1:60,1:16)./100, Data(1:60,17));
AData1 = horzcat(Data(1:60,1:16)./100,ones(60,1));
Label1 = Data(1:60,17);

LData2 = horzcat(Data(61:90,1:16)./100, Data(61:90,17));
AData2 = horzcat(Data(61:90,1:16)./100,ones(30,1));
Label2 = Data(61:90,17);

% weights
W1 = ones(17,16)*(-0.5);
W2 = ones(17,10)*(-0.5);

% learning rate
alpha = 1;

% training
quadErrorTraining = 0;
quadErrorTesting = 0;
numIter = 0
while quadErrorTraining >= quadErrorTesting
    clc
    [quadErrorTraining,quadErrorTesting, quadErrorTesting - ↩
        quadErrorTraining]
    numIter = numIter + 1

    % start training
    dW1 = zeros(16,17);
    dW2 = zeros(10,17);
    quadErrorTraining = 0;
    quadErrorTesting = 0;
    % start training batch
    for i = 1:60
        d = AData1(i,:);
        l = Label1(i,:);

        % forward pass - layer 1
        t1 = d * W1;
        out_layer1 = 1 ./ (1 + exp(-t1));

        % forward pass - layer 2
        t2 = [out_layer1, 1]*W2;
        out_layer2 = 1 ./ (1 + exp(-t2));

        % error calculation
```

```
        lv = zeros(1,10);
        for j = 1:10
            if l == j
                lv(1,j+1) = 1;
            end
        end
        error = (out_layer2 - lv);
        quadErrorTraining = quadErrorTraining + 0.5*(error * error');

        % backward pass - layer 1
        s1_der = out_layer1 .* (1 - out_layer1);
        D1 = diag(s1_der);

        % backward pass - layer 2
        s2_der = out_layer2 .* (1 - out_layer2);
        D2 = diag(s2_der);

        W2_                 = W2(1:16,:);
        delta2              = D2*error';
        delta1              = D1*W2_*delta2;
        dW1                 = dW1 + -alpha*delta1*d;
        dW2                 = dW2 + -alpha*delta2*[out_layer1, 1];
    end
    W1                = W1 + dW1';
    W2                = W2 + dW2';


    % start testing

    for runs = 1:length(AData2)

        d     = AData2(runs,:);
        l     = Label2(runs);

        % forward pass
        % layer 1
        t1          = d * W1;
        out_layer1 = 1 ./ (1 + exp(-t1));

        % layer 2
        t2          = [out_layer1, 1]*W2;
        out_layer2 = 1 ./ (1 + exp(-t2));

        % error calculation
        lv = zeros(1,10);
        for j = 1:10
            if l == j
                lv(1,j+1) = 1;
            end
        end
        error = (out_layer2 - lv);
        quadErrorTesting = quadErrorTesting + 0.5*(error * error');
    end
end % end of while quadErrorTraining >= quadErrorTesting
```

## 1.2 Resultate

Unsere Loesung terminierte selbst nach mehreren Tagen und 30330600 Iterationen nicht. Sowohl der quadratische Fehler der Trainingsmenge als der der Validierungsmenge sinken beim Training, jedoch bleibt der quad. Fehler der Trainingsmenge immer groesser als der quad. Fehler der Validierungsmenge, wie der folgende Plot veranschaulicht:



# 2 Aufgabe 1b - Rprop

## 2.1 Code

```
Data   = load('pendigits-training.txt');

% prepare data for network
LData1 = horzcat(Data(1:60,1:16)./100, Data(1:60,17));
AData1 = horzcat(Data(1:60,1:16)./100,ones(60,1));
Label1 = Data(1:60,17);

LData2 = horzcat(Data(61:90,1:16)./100, Data(61:90,17));
AData2 = horzcat(Data(61:90,1:16)./100,ones(30,1));
Label2 = Data(61:90,17);

% weights
disp('original weights:')
W1 = ones(17,16)*(-0.5);
W2 = ones(17,10)*(-0.5);

% Initialisierung der Rprop-Parameter
alpha = 0.0001; % learning rate
up = 1.5;
down = 0.2;
amax = 1;
amin = 0.01;
dE1_old = zeros(16,17);
dE2_old = zeros(10,17);
alphasdW1 = ones(16,17) * alpha;
alphasdW2 = ones(10,17) * alpha;

% training
quadErrorTraining = 0;
quadErrorTesting = 0;
numIter = 0

while quadErrorTraining >= quadErrorTesting
    clc
    numIter = numIter + 1
    disp('[quadErrorTraining, quadErrorTesting, difference]');
    [quadErrorTraining, quadErrorTesting, quadErrorTesting - ↩
        quadErrorTraining]

    % start training
    dW1 = zeros(16,17);
    dW2 = zeros(10,17);
    quadErrorTraining = 0;
    quadErrorTesting = 0;
    dE1_acc = zeros(16,17);
    dE2_acc = zeros(10,17);
    for i = 1:60
        d = AData1(i,:);
        l = Label1(i,:);

```

```
50          % forward pass - layer 1
51          t1 = d * W1;
52          out_layer1 = 1 ./ (1 + exp(-t1));
53
54          % forward pass - layer 2
55          t2 = [out_layer1, 1]*W2;
56          out_layer2 = 1 ./ (1 + exp(-t2));
57
58          % error calculation
59          lv = zeros(1,10);
60          for j = 1:10
61              if l == j
62                  lv(1,j+1) = 1;
63              end
64          end
65          error = (out_layer2 - lv);
66          quadErrorTraining = quadErrorTraining + 0.5*(error * error');
67
68          % backward pass - layer 1
69          s1_der = out_layer1 .* (1 - out_layer1);
70          D1 = diag(s1_der);
71
72          % backward pass - layer 2
73          s2_der = out_layer2 .* (1 - out_layer2);
74          D2 = diag(s2_der);
75
76          W2_                 = W2(1:16,:);
77          delta2              = D2*error';        % 10x1
78          delta1              = D1*W2_*delta2;  % 16x1
79          dW1                 = dW1 + -alphasdW1 .* sign(delta1*d);
80          dW2                 = dW2 + -alphasdW2 .* sign(delta2*[out_layer1, ↩
                1]);
81
82          % accumulate error function gradient to use for backprop later:
83          dE1_acc = dE1_acc + delta1*d;
84          dE2_acc = dE2_acc + delta2*[out_layer1, 1];
85      end % end of training batch
86      W1                  = W1 + dW1';
87      W2                  = W2 + dW2';
88
89
90      % Lernraten mit Rprop anpassen:
91      if numIter == 1
92          dE1_old             = dE1_acc;
93          dE2_old             = dE2_acc;
94      else
95          dE1                 = dE1_acc; % Matrix der partiellen Ableitungen ↩
                von E1 nach dem i-ten Gewicht
96          dE2                 = dE2_acc;
97          dE1_new_old         = dE1 .* dE1_old;
98          dE2_new_old         = dE2 .* dE2_old;
99          dE1_old             = dE1;
100         dE2_old             = dE2;
101
102         % neue Lernraten fuer die Gewichte der 2. Schicht berechnen:
103         for wi=1:size(dE2, 1)
104             for wj=1:size(dE2, 2)
```

```
                if (dE2(wi, wj) * dE2_old(wi, wj)) > 0  % beschleunigen
                    alphasdW2(wi, wj) = min(alphasdW2(wi, wj) * up, amax);
                elseif (dE2(wi, wj) * dE2_old(wi, wj)) < 0  % bremsen
                    alphasdW2(wi, wj) = max(alphasdW2(wi, wj) * down, amin);
                end
            end
        end

        % neue Lernraten fuer die Gewichte der 1. Schicht berechnen:
        for wi=1:size(dE1, 1)
            for wj=1:size(dE1, 2)
                if dE1(wi, wj) * dE1_old(wi, wj) > 0  % beschleunigen
                    alphasdW1(wi, wj) = min(alphasdW1(wi, wj) * up, amax);
                elseif dE1(wi, wj) * dE1_old(wi, wj) < 0  % bremsen
                    alphasdW1(wi, wj) = max(alphasdW1(wi, wj) * down, amin);
                end
            end
        end
    end % end of rprop calculations



    % start testing

    for runs = 1:length(AData2)

        d     = AData2(runs,:);
        l     = Label2(runs);

        % forward pass
        t1          = d * W1;
        out_layer1 = 1 ./ (1 + exp(-t1));

        t2          = [out_layer1, 1]*W2;
        out_layer2 = 1 ./ (1 + exp(-t2));

        % error calculation
        lv = zeros(1,10);
        for j = 1:10
            if l == j
                lv(1,j+1) = 1;
            end
        end
        error = (out_layer2 - lv);
        quadErrorTesting = quadErrorTesting + 0.5*(error * error');
    end
end % end of while quadErrorTraining >= quadErrorTesting
```

## 2.2 Resultate

Diese auf der Aufg. 1a basierende Implementierung von Rprop terminiert leider auch nicht.
Die Gewichte wachsen, bis fuer die Gewichtsmatrix $W1$ Werte erreicht werden, die nach Multiplikation mit dem Input-Vektor $d$ so hohe Eingabewerte $x$ fuer die Sigmoid-Funktion liefern, dass matlab den Ausdruck $\frac{1}{1+e^{-x}}$ zu 0 abrundet. Infolgedessen ist dann auch die Matrix $dW1$ zur Anpassung der Gewichtsmatrix $W1$ eine reine Nullmatrix und der Lernprozess kommt bei den folgenden Werten zum Stillstand:

$quadErrorTraining = 27.0000$
$quadErrorTesting = 12.5000$