

Übungsblatt 9

„Mustererkennung“

J. Cavojska, N. Lehmann, R. Toudic

14.07.2015

Inhaltsverzeichnis

1	Aufgabe 1a - Trainingsmenge vs. Validierungsmenge	2
1.1	Code	2
1.2	Resultate	4
2	Aufgabe 1b - Rprop	5
2.1	Code	5
2.2	Resultate	8

1 Aufgabe 1a - Trainingsmenge vs. Validierungsmenge

1.1 Code

```
1 Data = load('pendigits-training.txt');
2
3 % prepare data for network
4 LData1 = horzcat(Data(1:60,1:16)./100, Data(1:60,17));
5 AData1 = horzcat(Data(1:60,1:16)./100, ones(60,1));
6 Label1 = Data(1:60,17);
7
8 LData2 = horzcat(Data(61:90,1:16)./100, Data(61:90,17));
9 AData2 = horzcat(Data(61:90,1:16)./100, ones(30,1));
10 Label2 = Data(61:90,17);
11
12 % weights
13 W1 = ones(17,16)*(-0.5);
14 W2 = ones(17,10)*(-0.5);
15
16 % learning rate
17 alpha = 1;
18
19 % training
20 quadErrorTraining = 0;
21 quadErrorTesting = 0;
22 numIter = 0
23 while quadErrorTraining >= quadErrorTesting
24     clc
25     [quadErrorTraining, quadErrorTesting, quadErrorTesting - \
        quadErrorTraining]
26     numIter = numIter + 1
27
28     % start training
29     dW1 = zeros(16,17);
30     dW2 = zeros(10,17);
31     quadErrorTraining = 0;
32     quadErrorTesting = 0;
33     % start training batch
34     for i = 1:60
35         d = AData1(i,:);
36         l = Label1(i,:);
37
38         % forward pass - layer 1
39         t1 = d * W1;
40         out_layer1 = 1 ./ (1 + exp(-t1));
41
42         % forward pass - layer 2
43         t2 = [out_layer1, 1]*W2;
44         out_layer2 = 1 ./ (1 + exp(-t2));
45
46         % error calculation
```

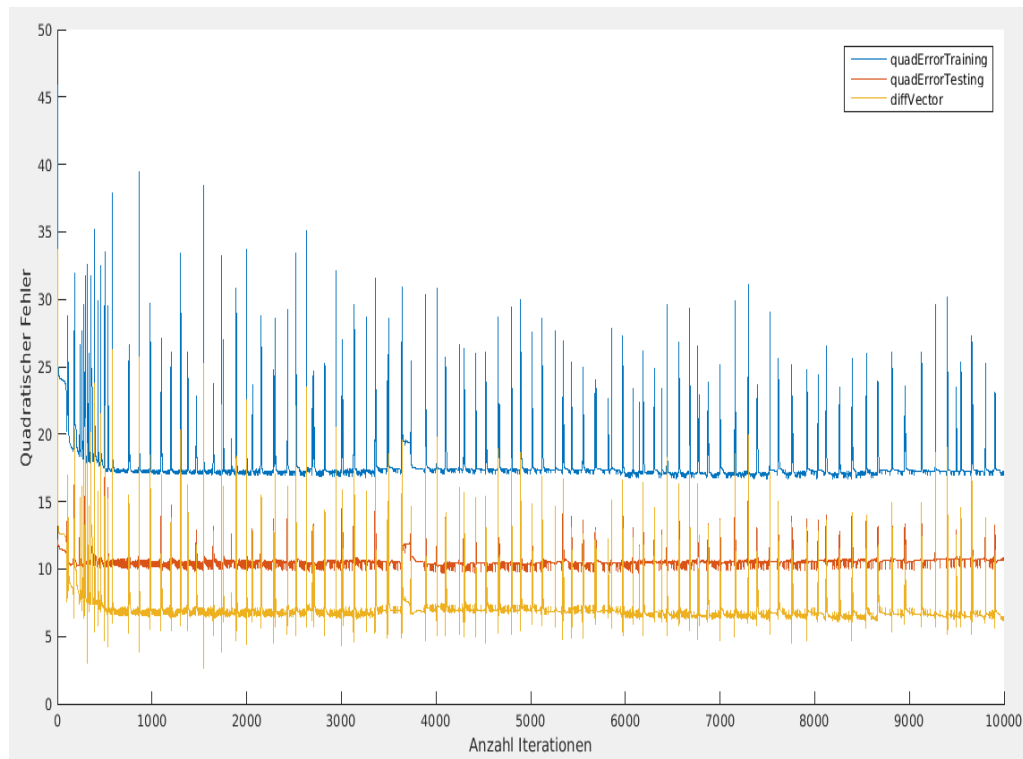
```

47     lv = zeros(1,10);
48     for j = 1:10
49         if l == j
50             lv(1,j+1) = 1;
51         end
52     end
53     error = (out_layer2 - lv);
54     quadErrorTraining = quadErrorTraining + 0.5*(error * error');
55
56     % backward pass - layer 1
57     s1_der = out_layer1 .* (1 - out_layer1);
58     D1 = diag(s1_der);
59
60     % backward pass - layer 2
61     s2_der = out_layer2 .* (1 - out_layer2);
62     D2 = diag(s2_der);
63
64     W2_          = W2(1:16,:);
65     delta2       = D2*error';
66     delta1       = D1*W2_*delta2;
67     dW1          = dW1 + -alpha*delta1*d;
68     dW2          = dW2 + -alpha*delta2*[out_layer1, 1];
69 end
70 W1              = W1 + dW1';
71 W2              = W2 + dW2';
72
73
74 % start testing
75
76 for runs = 1:length(AData2)
77
78     d      = AData2(runs,:);
79     l      = Label2(runs);
80
81     % forward pass
82     % layer 1
83     t1      = d * W1;
84     out_layer1 = 1 ./ (1 + exp(-t1));
85
86     % layer 2
87     t2      = [out_layer1, 1]*W2;
88     out_layer2 = 1 ./ (1 + exp(-t2));
89
90     % error calculation
91     lv = zeros(1,10);
92     for j = 1:10
93         if l == j
94             lv(1,j+1) = 1;
95         end
96     end
97     error = (out_layer2 - lv);
98     quadErrorTesting = quadErrorTesting + 0.5*(error * error');
99 end
100 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

```
1 Data = load('pendigits-training.txt');
2
3 % prepare data for network
4 LData1 = horzcat(Data(1:60,1:16)./100, Data(1:60,17));
5 AData1 = horzcat(Data(1:60,1:16)./100, ones(60,1));
6 Label1 = Data(1:60,17);
7
8 LData2 = horzcat(Data(61:90,1:16)./100, Data(61:90,17));
9 AData2 = horzcat(Data(61:90,1:16)./100, ones(30,1));
10 Label2 = Data(61:90,17);
11
12 % weights
13 disp('original weights:')
14 W1 = ones(17,16)*(-0.5);
15 W2 = ones(17,10)*(-0.5);
16
17 % Initialisierung der Rprop-Parameter
18 alpha = 0.0001; % learning rate
19 up = 1.5;
20 down = 0.2;
21 amax = 1;
22 amin = 0.01;
23 dE1_old = zeros(16,17);
24 dE2_old = zeros(10,17);
25 alphasdW1 = ones(16,17) * alpha;
26 alphasdW2 = ones(10,17) * alpha;
27
28 % training
29 quadErrorTraining = 0;
30 quadErrorTesting = 0;
31 numIter = 0
32
33 while quadErrorTraining >= quadErrorTesting
34     clc
35     numIter = numIter + 1
36     disp(['quadErrorTraining, quadErrorTesting, difference']);
37     [quadErrorTraining, quadErrorTesting, quadErrorTesting - ←
        quadErrorTraining]
38
39     % start training
40     dW1 = zeros(16,17);
41     dW2 = zeros(10,17);
42     quadErrorTraining = 0;
43     quadErrorTesting = 0;
44     dE1_acc = zeros(16,17);
45     dE2_acc = zeros(10,17);
46     for i = 1:60
47         d = AData1(i,:);
48         l = Label1(i,:);
49
```

```

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)

```

```

105         if (dE2(wi, wj) * dE2_old(wi, wj)) > 0 % beschleunigen
106             alphasdW2(wi, wj) = min(alphasdW2(wi, wj) * up, amax);
107         elseif (dE2(wi, wj) * dE2_old(wi, wj)) < 0 % bremsen
108             alphasdW2(wi, wj) = max(alphasdW2(wi, wj) * down, amin);
109         end
110     end
111 end
112
113 % neue Lernraten fuer die Gewichte der 1. Schicht berechnen:
114 for wi=1:size(dE1, 1)
115     for wj=1:size(dE1, 2)
116         if dE1(wi, wj) * dE1_old(wi, wj) > 0 % beschleunigen
117             alphasdW1(wi, wj) = min(alphasdW1(wi, wj) * up, amax);
118         elseif dE1(wi, wj) * dE1_old(wi, wj) < 0 % bremsen
119             alphasdW1(wi, wj) = max(alphasdW1(wi, wj) * down, amin);
120         end
121     end
122 end
123 end % end of rprop calculations
124
125
126
127 % start testing
128
129 for runs = 1:length(AData2)
130
131     d      = AData2(runs,:);
132     l      = Label2(runs);
133
134     % forward pass
135     t1      = d * W1;
136     out_layer1 = 1 ./ (1 + exp(-t1));
137
138     t2      = [out_layer1, 1]*W2;
139     out_layer2 = 1 ./ (1 + exp(-t2));
140
141     % error calculation
142     lv = zeros(1,10);
143     for j = 1:10
144         if l == j
145             lv(1,j+1) = 1;
146         end
147     end
148     error = (out_layer2 - lv);
149     quadErrorTesting = quadErrorTesting + 0.5*(error * error');
150 end
151 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