Mustererkennung WiSe 12/13
Übung 7

Lutz Freitag, Sebastian Kürten

1 Aufgabe 1: XOR

Wir haben ein neuronales Netz wie in der Vorlesung mit Matritzen implementiert, nutzen dabei das online Lernverfahren und lassen es an den Testdaten lernen.

Interessant ist, dass bei manchen initialen Gewichten das Lernen bei einem Fehler von 0.5 konvergiert. Hier muss sich ein lokales Minimum befinden.

In der grafischen Darstellung sieht man den Fehler gegen die Iterationen aufgetragen.

Nach Ende des Lernens werden die Eingaben zur Verifikation in das Netz gegeben und es ergeben sich Fehler von wenigen Promill.

1.0.1 Code

```
inputs  = [ [0, 0, 0]; ...
            [0, 1, 1]; ...
            [1, 0, 1]; ...
            [1, 1, 0]]

W1 = 2 * rand(3,2)
W2 = 2 * rand(3,1)

gamma = 1;

error = inf
errorOverTime = [];
while (error > 0.0005)
    error = 0;
    correction1 = zeros(size(W1));
    correction2 = zeros(size(W2));

    for input = inputs'
        % seperate the input from expected output
        output = input(end);
        input = augmentWithOnes(input(1:end - 1)')';

        % forward propagation
        o1 = sigmoid(input' * W1);
        o2 = sigmoid(augmentWithOnes(o1) * W2);

        e = (o2 .- output);
        error += 0.5 * e^2;

        D1 = diag((o1 .* (1 .- o1)));
        D2 = diag((o2 .* (1 .- o2)));

        % the small W2 matrix
        W2s = W2(2:end,:);

        delta2 = D2 * e;
        delta1 = D1 * W2s * delta2;
```

```
38
39          correction1 -= (gamma * delta1 * input')';
40          correction2 -= (gamma * delta2 * augmentWithOnes(o1))';
41      end
42
43      W1 += correction1;
44      W2 += correction2;
45      error
46      errorOverTime = [errorOverTime, error];
47  end
48
49  W1
50  W2
51
52  for input = inputs'
53      % seperate the input from expected output
54      output = input(end)
55      input = augmentWithOnes(input(1:end - 1)')';
56      o1 = sigmoid(input' * W1);
57      o2 = sigmoid(augmentWithOnes(o1) * W2)
58
59  end
60
61  semilogy(errorOverTime);
62
63  print("a1.png");
64  pause
```

1.0.2 Ausgabe

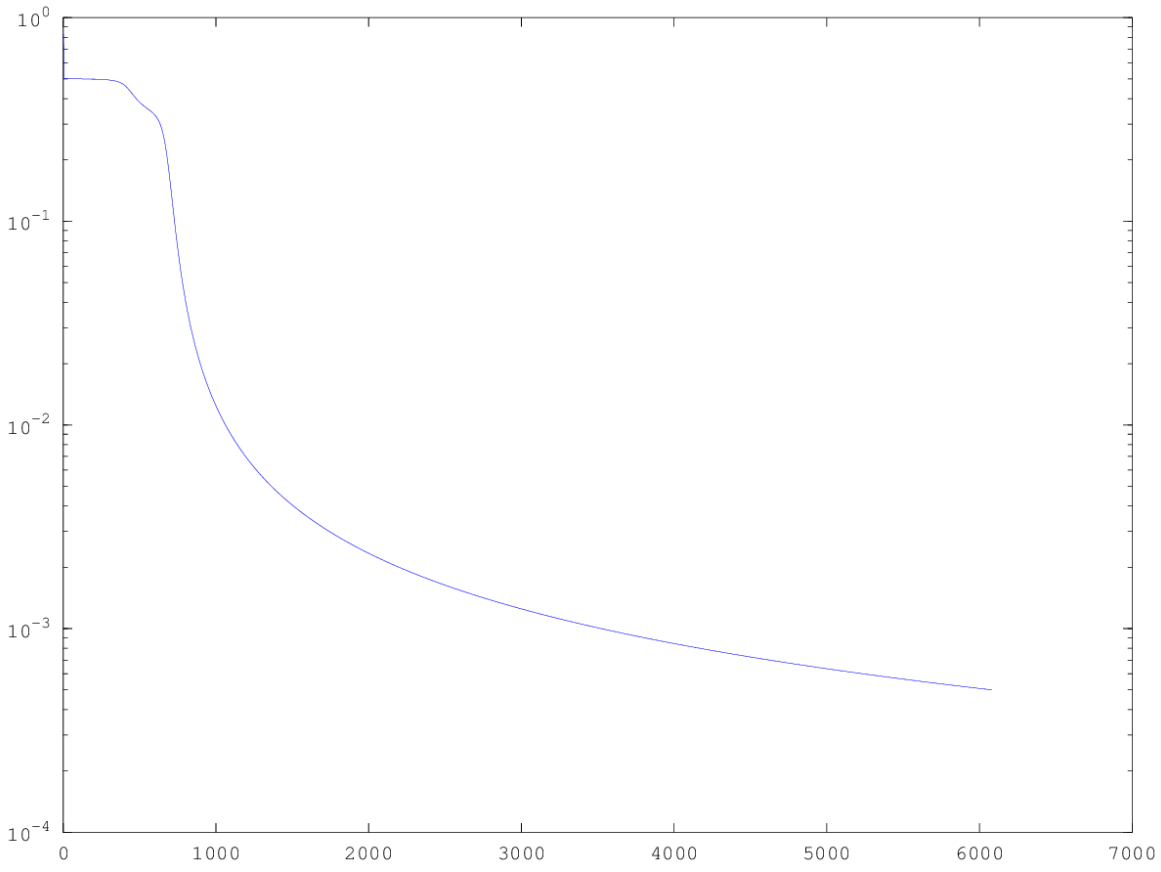

Figure 1: Fehler des neuronalen Netzes vs. Iteration

2 Aufgabe 2: Neuronale Netze auf Pendigits

Wir haben die wichtigsten Hauptkomponenten für die Aufgabe (siehe letzten Zettel) genutzt und damit das Netz trainiert. Dafür haben wir für jede Iteration für ein Gewichtsupdate ein Subset zufällig aus den Trainingsdaten gewählt und mit diesem gelernt.

Als zweiten Teil wird das so eingelerne Netz auf den Testdaten angewand und überprüft, wie gut das Netz generalisieren kann. Dabei muss zwingend die gleiche PCA für die Testdaten genutzt werden, die auch für die Trainingsdaten genommen wurde (fieser Fallstrick).

Wir haben in mehreren Versuchen die Anzahl der Knoten im hidden layer zwischen 13 und 60 variieren lassen, wobei die Gewichte bei 35 am schnellsten konvergierten und die Erfolgsrate gut war.

Die Erkennungsraten liegen dabei zwischen 90% und 95%

2.0.3 Code - Generierung des neuronalen Netzes

```
1  trainingData = load("-ascii", "pendigits-training.txt");
2  testingData = load("-ascii", "pendigits-testing.txt");
3
4  % Mittelwert der Daten berechnden
5  move = [mean(trainingData(:,1:end-1)) 0];
6  % und Daten zum Ursprung verschieben
7  trainingData = trainingData - repmat(move, size(trainingData, 1), 1);
8  testingData = testingData - repmat(move, size(testingData, 1), 1);
9
10 % features von den labels trennen
11 featuresTraining = trainingData(:,1:end - 1);
12 labelsTraining = trainingData(:, end);
13
14 featuresTesting = testingData(:,1:end - 1);
15 labelsTesting = testingData(:, end);
16
17 % compute mu and covanriance matrix
18 [mu, cov] = gauss(featuresTraining);
19
20 % eigenvektoren und eigenwerte berechnen
21 % eigenvektoren stehen in den spalten
22 % die wichtigste komponente steht ganz rechts
23 [u,s,v] = svd(cov);
24
25 % die labelmatrix zum Trainieren so aufbauen, dass an der entsprechenden ...
       Stelle eine 1 steht
26 labels = [];
27 for label = labelsTraining'
28     row = zeros(1,10);
29     row(label + 1) = 1;
30     labels = [labels; row];
31 end
32 labelsTraining = labels;
33
34 % Daten auf die Eigenvektoren projizieren
35 trainingPCA = featuresTraining * u;
36
37 % die unwichtigsten Hauptkomponenten wegwerfen
38 trainingPCAn = trainingPCA(:, 1:14);
39
40 trainingPCAn = [trainingPCAn, labelsTraining];
41
42 inputs = trainingPCAn;
43
```

```
W1 = 1 * rand(15,35);
W2 = 1 * rand(36,10);

gamma = 0.05;
subsetSize = 100
noise = 0.00

error = inf
while (error > 0.5)
    error = 0;
    correction1 = zeros(size(W1));
    correction2 = zeros(size(W2));

    % add some noise
    W1 += noise * (rand(size(W1)) -0.5);
    W2 += noise * (rand(size(W2)) -0.5);

    % generate a subset of inputs to fasten up the computation
    size(inputs, 1);
    r = ceil((size(inputs, 1) - subsetSize) * rand());
    subset = inputs(r:(r + subsetSize),:);

    for input = subset'
        % seperate the input from expected output
        output = input(end - 9: end);
        input = augmentWithOnes(input(1:end - 10)')';

        % forward propagation
        o1 = sigmoid(input' * W1);
        o2 = sigmoid(augmentWithOnes(o1) * W2)';

        e = (o2 .- output);
        error += sum(0.5 .* e.^2);

        D1 = diag((o1 .* (1 .- o1)));
        D2 = diag((o2 .* (1 .- o2)));

        % the small W2 matrix
        W2s = W2(2:end,:);

        delta2 = D2 * e;
        delta1 = D1 * W2s * delta2;

        correction1 -= (gamma * delta1 * input')';
        correction2 -= (gamma * delta2 * augmentWithOnes(o1))';
    end

    W1 += correction1;
    W2 += correction2;

    error'
end

W1
W2

% save computed results
save weights.mat W1 W2
```

2.0.4 Code - Auswertung mit den Testdaten

```

```

```
% load computed results
load weights.mat W1 W2

testingData = load("-ascii", "pendigits-testing.txt");
trainingData = load("-ascii", "pendigits-training.txt");

% Mittelwert der Daten berechnden
move = [mean(trainingData(:,1:end-1)) 0];

% und Daten zum Ursprung verschieben
trainingData = trainingData - repmat(move, size(trainingData, 1), 1);
testingData = testingData - repmat(move, size(testingData, 1), 1);

% features von den labels trennen
featuresTraining = trainingData(:,1:end - 1);
featuresTesting = testingData(:,1:end - 1);
labelsTesting = testingData(:, end);

% compute mu and covanriance matrix
[mu, cov] = gauss(featuresTraining);

% eigenvektoren und eigenwerte berechnen
% eigenvektoren stehen in den spalten
% die wichtigste komponente steht ganz rechts
[u,s,v] = svd(cov);

% die labelmatrix zum Trainieren so aufbauen, dass an der entsprechenden ...
    Stelle eine 1 steht
labels = [];
for label = labelsTesting'
    row = zeros(1,10);
    row(label + 1) = 1;
    labels = [labels; row];
end

labelsTesting = labels;

% Daten auf die Eigenvektoren projizieren
testingPCA = featuresTesting * u;

% die unwichtigsten Hauptkomponenten wegwerfen
testingPCAn = testingPCA(:, 1:14);

testingPCAn = [testingPCAn, labelsTesting];

inputs = testingPCAn;

hits = 0
misses = 0

for input = inputs'
    % seperate the input from expected output
    output = input(end - 9: end);
    input = augmentWithOnes(input(1:end - 10)')';

    o1 = sigmoid(input' * W1);
    o2 = sigmoid(augmentWithOnes(o1) * W2);
    [bla, positive] = max(output);
    [bla, guess]    = max(o2);
    positive
    guess

    if positive == guess
        hits += 1;
    else
        misses += 1;
```

```
68      end
69  end
70
71  hits
72  misses
73  coverage = hits / (hits + misses)
```