Mustererkennung WiSe 12/13
Übung 8

Lutz Freitag, Sebastian Kürten

1 Aufgabe 1: Online vs. Batch

1.1 Code

1.1.1 a1.m

Durchführung der Algorithmen und abspeichern der Resultate.

```
1  penTrainingData = load("-ascii", "pendigits-training.txt");
2  ionTrainingData = load("-ascii", "ionosphere.data");
3
4
5  penFeaturesTraining = penTrainingData(:,1:end - 1);
6  penLabelsTraining = penTrainingData(:, end);
7
8  ionFeaturesTraining = ionTrainingData(:,1:end - 1);
9  ionLabelsTraining = ionTrainingData(:, end);
10
11 [penTrainingPCA, penMove, penCov, penU] = normalize(penFeaturesTraining);
12 [ionTrainingPCA, ionMove, ionCov, ionU] = normalize(ionFeaturesTraining);
13
14 % die labelmatrix zum Trainieren so aufbauen, dass an der entsprechenden ...
       Stelle eine 1 steht
15 penLabels = [];
16 for label = penLabelsTraining'
17     row = zeros(1,10);
18     row(label + 1) = 1;
19     penLabels = [penLabels; row];
20 end
21 penLabelsTraining = penLabels;
22
23
24 % die unwichtigsten Hauptkomponenten wegwerfen
25 penTrainingPCAn = penTrainingPCA(:, 1:14);
26
27 % die unwichtigsten Hauptkomponenten wegwerfen
28 ionTrainingPCAn = ionTrainingPCA %(:, 1:14);
29
30 % arguments for online()/batch(): input, outputs, #hidden, maxError, ...
       gamma, subsetSize
31 [ionDigitsWeightsOnline, ionDigitsErrorsOnline] = ...
       online(ionTrainingPCAn, ionLabelsTraining, 35, 0.5, 0.05, 100)
32 save ionOnline.mat ionDigitsWeightsOnline ionDigitsErrorsOnline
33
34 [ionDigitsWeightsBatch, ionDigitsErrorsBatch] = batch(ionTrainingPCAn, ...
       ionLabelsTraining, 35, 0.5, 0.05, 100)
35 save ionBatch.mat ionDigitsWeightsBatch ionDigitsErrorsBatch
36
37 [penDigitsWeightsOnline, penDigitsErrorsOnline] = ...
       online(penTrainingPCAn, penLabelsTraining, 35, 0.8, 0.05, 100)
38 save penOnline.mat penDigitsWeightsOnline penDigitsErrorsOnline
39
```

```
40 [penDigitsWeightsBatch, penDigitsErrorsBatch] = batch(penTrainingPCAn, ...
       penLabelsTraining, 35, 0.8, 0.05, 100)
41 save penBatch.mat penDigitsWeightsBatch penDigitsErrorsBatch
```

1.1.2 a1disp.m

Plotten der Resultate.

```
1
2  load penOnline.mat
3  load penBatch.mat
4
5  load ionOnline.mat
6  load ionBatch.mat
7
8  hold on
9  figure
10
11 subplot(2, 2, 1)
12 semilogy(penDigitsErrorsOnline);
13 grid on
14 title("pendigits errors online");
15
16
17 subplot(2, 2, 2)
18 semilogy(penDigitsErrorsBatch);
19 grid on
20 title("pendigits errors batch");
21
22
23 subplot(2, 2, 3)
24 semilogy(ionDigitsErrorsOnline);
25 grid on
26 title("ionosphere errors online");
27
28
29 subplot(2, 2, 4)
30 semilogy(ionDigitsErrorsBatch);
31 grid on
32 title("ionosphere errors batch");
33
34 hold off
35
36 print("a1.png");
37
38 pause
```

1.2 Plots

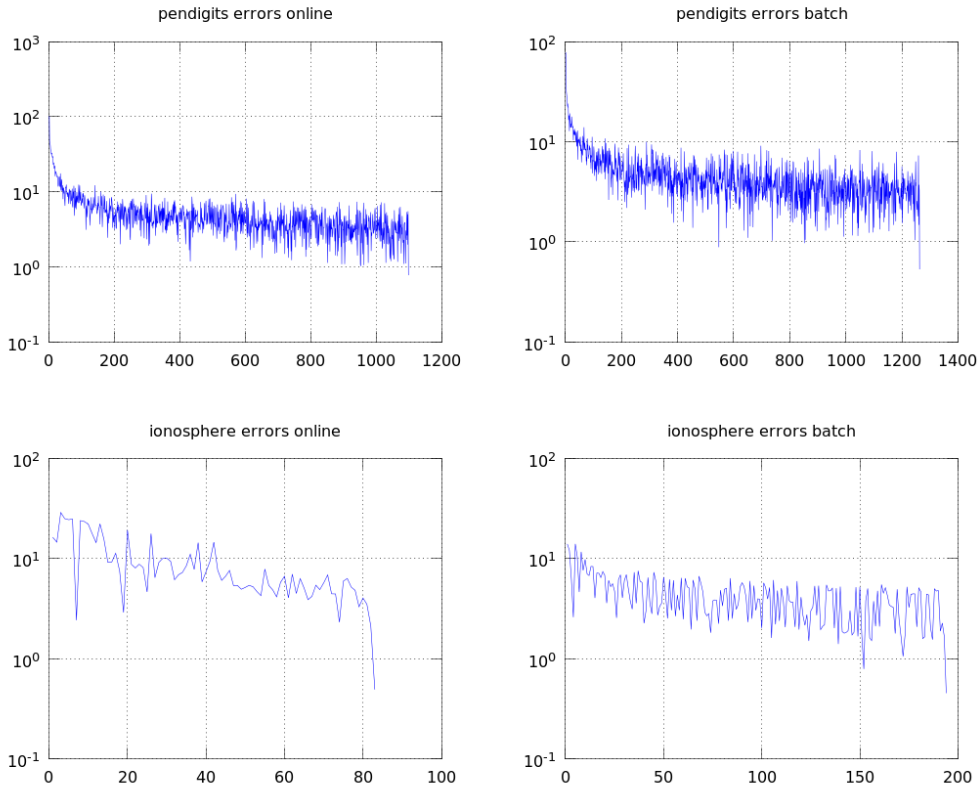

Figure 1: Iterationen vs. Fehler

1.3 Weiterer Code

1.3.1 online.m

```
function [weights, errors] = online(inputData, outputs, hiddenCnt, ...
    maxError, gamma, subsetSize)
    inputWidth = size(inputData, 2)
    outputWidth = size(outputs, 2)
    W1 = rand(inputWidth + 1, hiddenCnt) .- 0.5;
    W2 = rand(hiddenCnt + 1, outputWidth) .- 0.5;

    errors = [];

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

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

        subset = [subsetInputs, subsetOutputs];

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

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

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

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

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

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

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

        W1 += correction1;
        W2 += correction2;

        error'
        errors = [errors, error];
    end

    weights = {W1, W2};
end
```

1.3.2 batch.m

```
function [weights, errors] = batch(inputData, outputs, hiddenCnt, ...
    maxError, gamma, subsetSize)
    inputWidth = size(inputData, 2)
    outputWidth = size(outputs, 2)
    W1 = rand(inputWidth + 1, hiddenCnt) .- 0.5;
    W2 = rand(hiddenCnt + 1, outputWidth) .- 0.5;

    errors = [];

    error = inf
    while (error > maxError)
        error = 0;

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

        subset = [subsetInputs, subsetOutputs];

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

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

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

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

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

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

            W1 -= (gamma * delta1 * input')';
            W2 -= (gamma * delta2 * augmentWithOnes(o1))';

        end

        error'
        errors = [errors, error];

    end

    weights = {W1, W2};
end
```

1.3.3 normalize.m

```
1  function [ret, mean, cov, u] = normalize(inputData)
2      % Mittelwert der Daten berechnden
3      mean = [mean(inputData(:,1:end-1)) 0];
4      inputData = inputData - repmat(mean, size(inputData, 1), 1);
5
6      % compute mu and covanriance matrix
7      [mu, cov] = gauss(inputData);
8
9      [u,s,v] = svd(cov);
10
11     ret = inputData * u;
12 end
```

2 Aufgabe 2: RPROP

2.1 Code

2.1.1 a2.m

Durchführung der Algorithmen und abspeichern der Resultate.

```
1  penTrainingData = load("-ascii", "pendigits-training.txt");
2  ionTrainingData = load("-ascii", "ionosphere.data");
3
4
5  penFeaturesTraining = penTrainingData(:,1:end - 1);
6  penLabelsTraining = penTrainingData(:, end);
7
8  ionFeaturesTraining = ionTrainingData(:,1:end - 1);
9  ionLabelsTraining = ionTrainingData(:, end);
10
11 [penTrainingPCA, penMove, penCov, penU] = normalize(penFeaturesTraining);
12 [ionTrainingPCA, ionMove, ionCov, ionU] = normalize(ionFeaturesTraining);
13
14 % die labelmatrix zum Trainieren so aufbauen, dass an der entsprechenden ...
       Stelle eine 1 steht
15 penLabels = [];
16 for label = penLabelsTraining'
17     row = zeros(1,10);
18     row(label + 1) = 1;
19     penLabels = [penLabels; row];
20 end
21 penLabelsTraining = penLabels;
22
23
24 % die unwichtigsten Hauptkomponenten wegwerfen
25 penTrainingPCAn = penTrainingPCA(:, 1:14);
26
27 % die unwichtigsten Hauptkomponenten wegwerfen
28 ionTrainingPCAn = ionTrainingPCA %(:, 1:14);
29
30
31 [ionWeightsRPROP, ionErrorsRPROP] = rprop(ionTrainingPCAn, ...
       ionLabelsTraining, 35, 0.05, 0.0001, 0.5, 1.2, 0.5, 100)
32 [penWeightsRPROP, penErrorsRPROP] = rprop(penTrainingPCAn, ...
       penLabelsTraining, 35, 0.505, 0.0001, 0.5, 1.2, 0.5, 100)
33 save rprop.mat ionErrorsRPROP penErrorsRPROP
```

2.1.2 a2disp.m

Plotten der Resultate.

```
1
2  load rprop.mat
3
4  hold on
5  figure
6
7  subplot(2,1,1)
8  semilogy(ionErrorsRPROP);
9  grid on
10 title("ionospere errors rprop");
11
12 subplot(2,1,2)
13 semilogy(penErrorsRPROP);
14 grid on
15 title("pendigits errors rprop");
16
17
18 hold off
19
20 print("a2.png");
21
22 pause
```

2.2 Plots

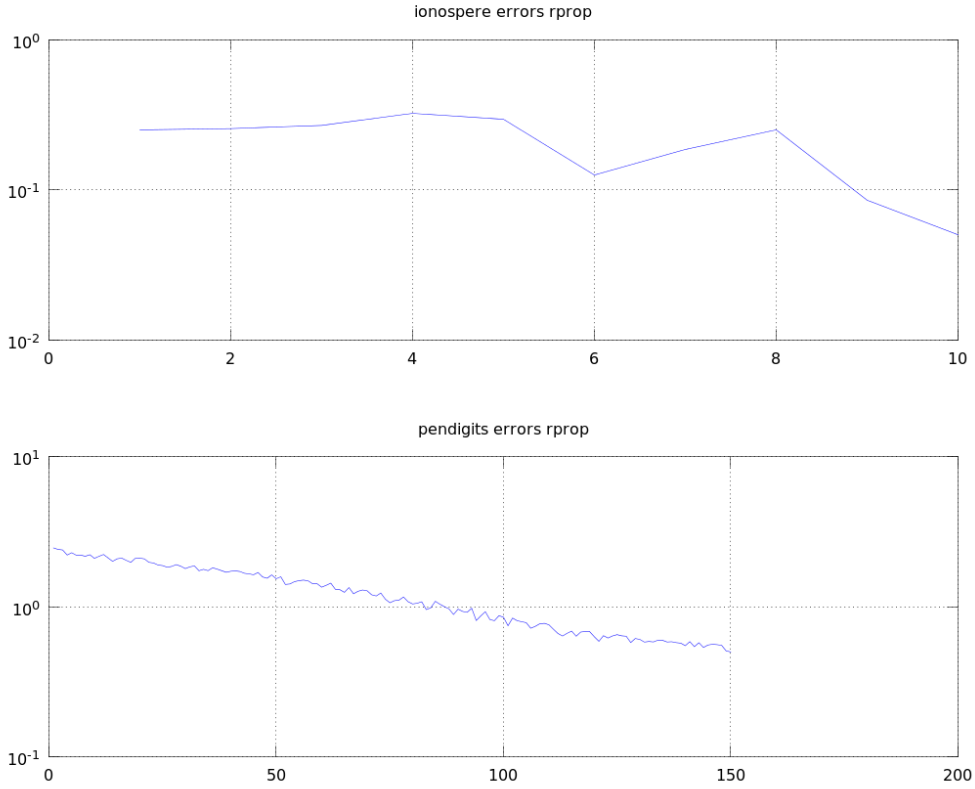

Figure 2: Iterationen vs. Fehler

2.3 Weiterer Code

2.3.1 rprop.m

```
function [weights, errors] = rprop(inputData, outputs, hiddenCnt, ...
    maxError, gammaMin, gammaMax, u, d, subsetSize)
    inputWidth = size(inputData, 2)
    outputWidth = size(outputs, 2)
    W1 = (rand(inputWidth + 1, hiddenCnt) .- 0.5) .* 10;
    W2 = (rand(hiddenCnt + 1, outputWidth) .- 0.5) .* 10;

    gammaMax1 = repmat(gammaMax, size(W1));
    gammaMax2 = repmat(gammaMax, size(W2));

    gammaMin1 = repmat(gammaMin, size(W1));
    gammaMin2 = repmat(gammaMin, size(W2));

    gamma1 = gammaMin1;
    gamma2 = gammaMin2;

    errors = [];

    error = 999
    while (error > maxError)
        error = 0;

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

        subset = [subsetInputs, subsetOutputs];

        last_correction11 = zeros(size(W1));
        last_correction21 = zeros(size(W2));

        last_correction12 = zeros(size(W1));
        last_correction22 = zeros(size(W2));

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

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

            err = (o2 .- output);
            error += (sum(0.5 .* err .^ 2) ./ subsetSize);

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

            W2s = W2(2:end,:);

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

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

            gamma1 =    (min(gamma1 * u, gammaMax1) .* ...
                ((last_correction11 .* last_correction12) < 0)) .+ ...
```

```
59                          (max(gamma1 * d, gammaMin1) .* ...
                                ((last_correction11 .* last_correction12) > ...
                                0)) .+ ...
60                          (gamma1 .* ((last_correction11 .* ...
                                last_correction12) == 0));
61
62              gamma2 =    (min(gamma2 * u, gammaMax2) .* ...
                    ((last_correction21 .* last_correction22) < 0)) .+ ...
63                          (max(gamma2 * d, gammaMin2) .* ...
                                ((last_correction21 .* last_correction22) > ...
                                0)) .+ ...
64                          (gamma2 .* ((last_correction21 .* ...
                                last_correction22) == 0));
65
66              gamma1';
67              gamma2';
68
69
70
71              last_correction12 = last_correction11;
72              last_correction22 = last_correction21;
73
74              last_correction11 = correction1;
75              last_correction21 = correction2;
76
77              W1 -= gamma1 .* sign(correction1);
78              W2 -= gamma2 .* sign(correction2);
79          end
80
81          error'
82          errors = [errors, error];
83
84      end
85
86      weights = {W1, W2};
87  end
```