HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Neuronale Netze und Fuzzy-Logik, SS 07

Aufgabe 1 - Schilddrüsenfunktion

Jan Tammen, Matrikel-Nr. 277143

4. Juli 2007

Prof. Dr. Bittel, HTWG Konstanz

Inhaltsverzeichnis

1. Beschreibung der Aufgabenstellung

Es soll ein neuronales Netz entworfen werden, welches in der Lage ist zu entscheiden, ob bei einem Patienten eine Fehlfunktion der Schilddrüse vorliegt. Dazu müssen die Werte 21 verschiedener Attribute ausgewertet werden. Anhand der hohen Anzahl der Attribute lässt sich erkennen, dass eine solche *Klassifizierungsaufgabe* mit herkömmlichen Methoden eher schwierig lösbar wäre. Daher wird hier auf die Verwendung eines künstlichen neuronalen Netzes zurückgegriffen, welche sich gut zur Klassifikation und Mustererkennung nutzen lassen.

In der vorliegenden Arbeit sollen zunächst die zur Verfügung stehenden Daten analysiert, in Test- und Trainingsdaten aufgeteilt und anschließend ein entsprechendes neuronales Netz mithilfe der *Neural Networks Toolbox* der Mathematiksoftware Matlab implementiert werden. Ziel ist es, eine Klassifizierungsrate zu erreichen, die deutlich über 92% liegt.

2. Lösungsansatz

2.1. Analyse und Aufbereitung der Daten

Die vorliegende Datei `thyroid.dat` enthält die Datensätze von 7200 Patienten und soll als Basis für das Training und den Test des zu erstellenden neuronalen Netzes dienen. Ein Datensatz enthält dabei jeweils 21 verschiedene Attribute, 15 davon in binärer Form und 6 reellwertig. Folgender Auszug aus der Datei zeigt den Aufbau der Datensätze:

```
1 0.35000 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.00260 0.02500 0.16100 0.12600 0.12800 3
2 0.78000 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0.00280 0.02010 0.06800 0.10000 0.06800 3
3 0.18000 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0.00020 0.02080 0.12000 0.11200 0.10700 3
4 0.55000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.00419 0.01600 0.06900 0.09900 0.07000 3
5 0.77000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.00007 0.00900 0.06100 0.09900 0.06100 3
```

Listing 2.1: Auszug aus der Datei `thyroid.dat`

Es ist hier nicht bekannt, welche Attribute welchen Messwerten entsprechen - dieses Wissen wird für die Lösung der Aufgabe auch nicht benötigt. Wie bereits eingangs erwähnt, stellt die hohe Anzahl von Attributen eine erste Schwierigkeit dar. Weiterhin bereiten die extrem ungleich verteilten Klassen Probleme für herkömmliche statistische Methoden. Die letzte Spalte eines Datensatz enthält dabei die zum Datensatz gehörige Klasse, derer es insgesamt drei gibt:

- 1: „hypothyroid“ (Schilddrüsenunterfunktion),
- 2: „hyperthyroid“ (Schilddrüsenüberfunktion),
- 3: „normal“ (normale Schilddrüsenfunktion).

Um zunächst einen groben Überblick über die Daten zu erhalten, wird zunächst die Verteilung der Datensätze auf die einzelnen Klassen untersucht. Die Situation stellt sich wie in Tabelle 2.1 aufgezeigt dar.

Klasse Nr.	Anzahl	Anteil [%]
1	166	2,3
2	368	5,1
3	6666	92,6

Tabelle 2.1.: Verteilung der Datensätze auf die drei Klassen

2.1.1. Aufteilung der Daten in Trainings- und Testdaten

Üblicherweise werden die verfügbaren Daten zufällig aufgeteilt in eine Trainings- und eine Testdatenmenge. Mithilfe der Trainingsmenge wird das neuronale Netz erstellt und trainiert. Die Testdatenmenge dient dazu, die Qualität des Netzes (in Bezug auf die Vorhersagefähigkeiten) abschließend zu überprüfen. Es muss doch gerade in dieser Aufgabe darauf geachtet werden, dass die Aufteilung der Daten korrekt erfolgt. Denn betrachtet man die Verteilung der Datensätze auf die einzelnen Klassen, so erkennt man, dass in der Klasse 3 die weitaus meisten Datensätze vorhanden sind (s. auch Abbildung 2.1). Würde man nun die Daten blindlings zufällig aufteilen, so erhielte man mit großer Wahrscheinlichkeit verfälschte Ergebnisse, da ja die meisten Datensätze in der Klasse 3 lägen. Daher wird hier so vorgegangen, dass die Daten zunächst in die Klassen aufgeteilt werden und anschließend aus jeder dieser drei Untermengen ein gleicher Prozentsatz an Datensätzen in den Trainigsdatenpool übernommen wird. Die verbleibenden Daten werden anschließend zur Simulation des Netzes benutzt, um dessen Generalisierungsfähigkeit zu ermitteln.

2.2. Entwurf eines neuronalen Netzes

Für die folgenden Versuche wird ein künstliches neuronales Netz vom Typ *feed-forward backpropagation* verwendet. Auf die Funktionsweise eines Feed-Forward-Netzes wird hier nicht näher eingegangen; diese ist u.a. beschrieben in [DB98]. Erzeugt wird ein solches Netz mithilfe der Neural Network Toolbox in Matlab über den folgenden Funktionsaufruf:

```
[net, tr, yTrain, eTrain] = newff(PR, [S_1 S_2 ... S_N], {TF_1 TF_2 ... TF_N},
    BTF, BLF, PF)
```

Den einzelnen Parametern kommt dabei die folgende Bedeutung zu:

PR Eine $R \times 2$ Matrix mit den Minimal- und Maximalwerten der R Eingabewerte in der Input-Schicht.

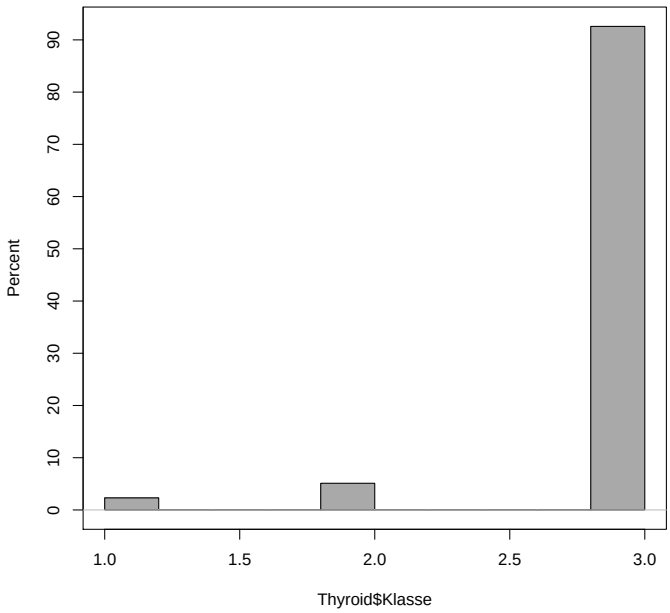


Abbildung 2.1.: Häufigkeitsverteilung der Klassen

S_i Größe der i-ten Schicht bei insgesamt N Schichten.

TF_i Aktivierungsfunktion der i-ten Schicht, Default: `tansig`.

BTF Trainingsfunktion, Default: `traingdx`.

BLF Gewichts-/Schwellwertlernfunktion, Default: `learngdm`.

PF Performancefunktion, Default: `mse`.

Für den Aufbau des Netzes wird zunächst eine **Eingabeschicht** benötigt, in der sich in diesem Falle 21 Neuronen befinden - eines für jedes auszuwertende Attribut. Die Anzahl der **verdeckten Schichten** wird zunächst auf 1 festgelegt. In dieser verdeckten Schicht werden 5 Neuronen eingesetzt. Die Anzahl der Neuronen und der verdeckten Schichten wird in weiteren Versuchen variiert werden, um eine möglichst optimale Konfiguration zu finden (s. Listing 3.2 im Anhang). Für die **Ausgabeschicht** schließlich werden drei binäre Neuronen verwendet. Die Werte der einzelnen Neuronen sind dabei entweder 0 oder 1, je nach gewünschter Klasse:

- Klasse 1: 1-0-0

- Klasse 2: 0-1-0
- Klasse 3: 0-0-1

2.2.1. Trainingsverfahren und -parameter

Für das initiale Training des Netzes werden die folgenden Trainingsparameter verwendet[1]:

- Prozentsatz von Trainingsdaten aus jeder Klasse: 10. Insgesamt 794 Datensätze, davon 17 aus Klasse 1, 37 aus Klasse 2 und 667 aus Klasse 3.
- Maximale Anzahl Epochen: 2000.
- Aktivierungsfunktion TF_1: `logsig`, TF_2: `logsig`.
- Trainingsfunktion BTF: `trainrp`.
- Performancefunktion PF: `mse`.
- Fehlerziel (`trainParam.goal`): 0.0001.

Als Trainingsfunktion wird hier die *Resilient Backpropagation*-Methode eingesetzt, da es nach [DB98] am besten geeignet ist für Klassifizierungsaufgaben und auch sehr performant ist. Als Fehlerziel wird MSE = 0.0001 verwendet. Bei MSE handelt es sich um den *Mean Sum Squared Error* und dieser ist definiert wie folgt: [Bit07, Kapitel 5].

$$MSE = \frac{1}{N} \sum_{(p,t) \in L} (t_i - y_i)^2$$

In den nachfolgenden Abschnitten sind die Ergebnisse der Testläufe mit der initialen Konfiguration sowie Versuche mit variierenden Netzkonfigurationen beschrieben.

[1] Nicht aufgeführte Parameter wurden in der Standardeinstellung belassen

3. Ergebnisse

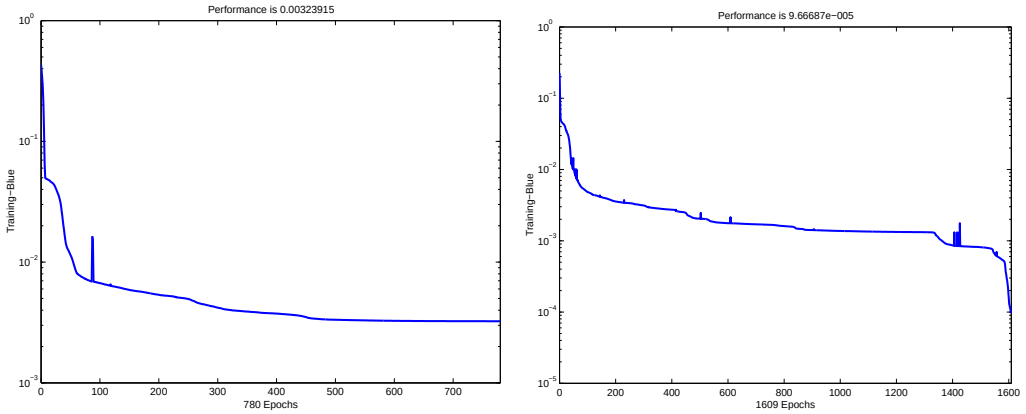
3.1. Testläufe mit initialer Konfiguration

Ziel dieser ersten Testläufe war es, herauszufinden, inwieweit das Fehlerziel mit den gewählten Parametern erreicht werden kann. Es wurden dabei jeweils 10 Testläufe mit identischen Parametern durchgeführt, um „Zufallstreffer“ (Gewichts- und Biaswerte werden bei Initialisierung des Netzes zufällig vergeben) auszuschließen. Tabelle 3.1 zeigt einen Überblick über diesen Testlauf.

Nr.	Epoche	MSE	Ziel erreicht
1	781	3.239147e-003	—
2	268	3.237180e-003	—
3	2000	1.390480e-003	—
4	887	2.027749e-002	—
5	2000	1.708763e-003	—
6	664	1.756893e-002	—
7	1610	9.666874e-005	✓
8	655	2.314696e-003	—
9	1737	1.851531e-003	—
10	338	3.238135e-003	—

Tabelle 3.1.: Ergebnisse der Testreihe mit 21-5-3 Netz

Lediglich ein Testlauf erreichte hier das Fehlerziel - die anderen brachen bei Erreichen der maximalen Epochenanzahl bzw. des minimalen Gradienten ab. Abbildung 3.1a zeigt beispielhaft den Verlauf der Performance (also des MSE) für Testlauf Nr. 1, Abbildung 3.1b zeigt den erfolgreichen Testlauf Nr. 7.



(a) Testlauf Nr. 1: minimaler Gradient erreicht

(b) Testlauf Nr. 7: erfolgreich

Abbildung 3.1.: Verlauf der Performance zweier Testläufe

3.2. Variation der Netzkonfiguration

Wie im vorherigen Abschnitt zu sehen, erreichte das Netz mit nur 5 Neuronen in einer verdeckten Schicht das vorgegebene Fehlerziel (MSE) von $1 \cdot 10^{-4}$ lediglich in einem der Testläufe. Daher werden nun in einem ersten Schritt die Anzahl der Neuronen in der verdeckten Schicht erhöht. Tabelle 3.2 zeigt die Ergebnisse für diesen Versuch; in der letzten Spalte ist aufgeführt, wie viele der 10 Testläufe das Fehlerziel erreicht haben.

Wie sich zeigt, wird das Fehlerziel in dem Netz mit *einer* verdeckten Schicht maximal in 50% der Testläufe erreicht. Es wird nun in einem weiteren Versuch mit einem Netz mit *zwei* verdeckten Schichten gearbeitet. Auch hier wird wieder wie zuvor die Anzahl der Neuronen variiert und mit jeweils 10 Testläufen gearbeitet. Es werden hier nur die vielversprechendsten Kombinationen getestet; Tabelle 3.3 zeigt die Ergebnisse.

Wie man den Testläufen entnehmen kann, scheint eine Netztopologie 21-45-45-3 am besten mit dieser Auswahl von Trainigsdaten trainierbar. Daher wird für die folgenden Simulationsversuche zunächst auf diese Konfiguration zurückgegriffen.

3.3. Simulation des Netzes mit ermittelter Konfiguration

Im ersten Versuch wird hier das Netz mit den verbleibenden Datensätzen simuliert, die nicht für das Training verwendet wurden. Dazu wird der folgende Funktionsaufruf verwendet:

Neuronen 1. verdeckte Schicht	Epoche Durchschnitt	MSE Durchschnitt	Ziel erreicht
10	690	1.129535e-003	2/10
15	473	1.222659e-003	2/10
20	463	9.913476e-004	2/10
25	391	8.526345e-004	2/10
30	382	6.968107e-004	5/10
35	382	6.957020e-004	5/10
40	379	9.908886e-004	2/10
45	380	1.195249e-003	4/10
50	315	1.176323e-003	2/10

Tabelle 3.2.: Ergebnisse der Testreihe mit variierender Neuronenzahl

Neuronen 1. verdeckte Schicht	Neuronen 2. verdeckte Schicht	Epoche Durchschnitt	MSE Durchschnitt	Ziel erreicht
10	10	328	1.581422e-003	1/10
15	15	327	5.940588e-004	4/10
30	30	210	4.081598e-004	5/10
35	35	215	4.459380e-004	4/10
45	45	174	3.408588e-004	7/10

Tabelle 3.3.: Ergebnisse der Testreihe mit zwei verdeckten Schichten

```
[ySim, pf, af, eSim, perf] = sim(net, simulationData');
```

Dabei wird zum einen das trainierte Netz `net` sowie die zur Simulation zu verwendenden Daten übergeben. Die Simulation mit einem erfolgreich trainierten Netz ergab die folgenden Ergebnise:

Fehler Klasse 1: 36/149, d.h. 75,84% korrekt erkannt.

Fehler Klasse 2: 68/331, d.h. 79,46% korrekt erkannt.

Fehler Klasse 3: 55/5999, d.h. 99,10% korrekt erkannt.

Gesamtfehler: 159/6479, d.h. 97,55% korrekt erkannt.

Wie man sieht, wird in den Klassen 1 und 2 eine weitaus niedrigere Klassifikationsgüte erreicht als in Klasse 3. Dies ist auf die geringe Anzahl der Trainingsdaten zurückzuführen. Insgesamt erreichte das Netz mit 97,67% jedoch schon ein recht gutes Ergebnis. Um die Generalisierungsfähigkeit des Netzes weiter zu steigern, wird nun versucht, die Anzahl der Trainingsdaten zu erhöhen. Es werden nicht mehr 10, sondern 50% der Datensätze für das Training verwendet:

Fehler Klasse 1: 21/83, d.h. 74,70% korrekt erkannt.

Fehler Klasse 2: 22/184, d.h. 88,04% korrekt erkannt.

Fehler Klasse 3: 19/3333, d.h. 99,43% korrekt erkannt.

Gesamtfehler: 58/3600, d.h. 98,23% korrekt erkannt.

Wie man sieht, kann die Klassifikationsgüte insgesamt durch die Erhöhung der Anzahl der Trainingsdaten leicht verbessert werden.

3.4. Fazit

Wie die vorangegangenen Versuche gezeigt haben, ist ein künstliches neuronales Netz eine fragiles Gebilde, welches von vielen Einflussfaktoren wie z.B. der Anzahl und Auswahl der Trainings- und Simulationsdaten beeinflusst wird. Auch die Bestimmung der „optimalen“ Netzkonfiguration ist nicht wirklich eindeutig, sondern wurde hier hauptsächlich durch Testläufe und Intuition bestimmt.

Insgesamt konnten im besten Fall ca. 98% der Datensätze korrekt zugeordnet werden. Allerdings bleibt die Klassifikationsgüte für die Klassen 1 und 2 recht gering, was auf die relativ kleine Anzahl an zur Verfügung stehenden Datensätzen zurückzuführen ist. Für die Klassifikation in Schilddrüsenüberfunktion bzw. -unterfunktion ist das künstliche neuronale Netz also nur bedingt zu gebrauchen - bei der Entscheidung jedoch, ob ein Patient *keine* Fehlfunktion hat, erzielt das Netz recht zuverlässige Ergebnisse.

A. Matlab-Programme

```
oneHiddenLayerRuns = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50];
twoHiddenLayerRuns = [10, 15, 30, 35, 45];

for numHiddenNeurons = 1 : size(twoHiddenLayerRuns, 2)
    for run = 1 : 10
        % Neuronales Netz definieren
        % Alternative Aktivierungsfkt.: tansig, purelin
        % Alternative Trainingsfkt.: trainlm, traingdx, trainrp
        net = newff(minmax(p), [twoHiddenLayerRuns(numHiddenNeurons),
            twoHiddenLayerRuns(numHiddenNeurons), 3], ...
                {'logsig', 'logsig', 'logsig'}, 'trainrp');
        net = init(net);

        % Trainingsparameter
        %net.trainParam.mem_reduc = 2; % Fuer trainlm
        %net.trainParam.lr = 0.01;
        %net.trainParam.mc = 0.9;

        net.performFcn = 'mse';    % Performance-Funktion
        net.trainParam.show = 10; % Ausgabe alle x Epochen
        net.trainParam.epochs = 2000; % Epochen
        net.trainParam.goal = 0.0001; % Fehlerziel

        [net, tr, y, e] = train(net, p, t);
        goalMet = 0;
        if (mse(e) <= net.trainParam.goal), goalMet = 1; end
        logToLogFile('thyroid.log', ...
            sprintf('Neurons: %d @ Run: %d ### BTF: %s\nTF: logsig\nPF: %s, Goal
                : %d\nEpoche: %d\nMSE: %d\nGoal met: %d\n', ...
            twoHiddenLayerRuns(numHiddenNeurons), run, net.trainFcn, net.
                performFcn, net.trainParam.goal, ...
            size(tr.epoch, 2), mse(e), goalMet));
        % Plot speichern
        global savefile;
```

```
32          savefile = sprintf('21-%d-%d-3_%d.eps', twoHiddenLayerRuns(
                numHiddenNeurons), ...
33              twoHiddenLayerRuns(numHiddenNeurons), run);
34          plotperf(tr);
35      end
36      logToLogFile('thyroid.log', sprintf('================================\n'));
37  end
```

Listing A.1: Variation der Netzkonfiguration zur Ermittlung eines optimalen Netzaufbaus

```
1   %%% Neuronale Netze, SS 07
2   %%% Jan Tammen, 277143
3
4   clear;
5
6   % Einlesen der Daten
7   thyroid = load('thyroid.dat');
8
9   % Aufteilen der Daten in die drei Klassen
10  % In der letzten Spalte (22) steht die Klassennummer
11  class1Samples = thyroid(find(thyroid(:,22) == 1), :);
12  class2Samples = thyroid(find(thyroid(:,22) == 2), :);
13  class3Samples = thyroid(find(thyroid(:,22) == 3), :);
14
15  % Nun X Prozent jeder Klasse als Trainingdatensaetze auswaehlen
16  percentTrainData = 0.50;
17  numClass1 = ceil(percentTrainData*size(class1Samples,1));
18  numClass2 = ceil(percentTrainData*size(class2Samples,1));
19  numClass3 = ceil(percentTrainData*size(class3Samples,1));
20  class1TrainData = class1Samples(1:numClass1, 1:21);
21  class2TrainData = class2Samples(1:numClass2, 1:21);
22  class3TrainData = class3Samples(1:numClass3, 1:21);
23
24  % Komplette Trainingsdaten (p) zusammenfuegen. Matrix muss dazu
25  % transponiert werden, da jeder Datensatz in einer _Spalte_ stehen muss
26  completeTrainData = [class1TrainData; class2TrainData; class3TrainData];
27  p = completeTrainData';
28
29  % Matrix mit den Zielwerten (target) erstellen, 3 Ausgabeneuronen
30  % Klasse 1: 1 0 0
31  % Klasse 2: 0 1 0
32  % Klasse 3: 0 0 1
33  tClass1 = repmat([1; 0; 0], 1, numClass1);
```

```
34 tClass2 = repmat([0; 1; 0], 1, numClass2);
35 tClass3 = repmat([0; 0; 1], 1, numClass3);
36 t = horzcat(tClass1, tClass2, tClass3);
37
38 % Das neuronale Netz mit den ermittelten Parametern erstellen
39 net = newff(minmax(p), [45, 45, 3], {'logsig', 'logsig', 'logsig'}, 'trainrp');
40 net = init(net);
41
42 net.performFcn = 'mse';    % Performance-Funktion
43 net.trainParam.show = 10;  % Ausgabe alle x Epochen
44 net.trainParam.epochs = 2000; % Epochen
45 net.trainParam.goal = 0.0001; % Fehlerziel
46
47 [net, tr, yTrain, eTrain] = train(net, p, t);
48
49 % Netz simulieren
50 disp('Training abgeschlossen');
51 pause; clc;
52
53 % Simulation mit den verbleibenden, nicht zum Training verwendeten Daten
54 class1SimData = class1Samples(numClass1+1:size(class1Samples, 1), 1:21);
55 class2SimData = class2Samples(numClass2+1:size(class2Samples, 1), 1:21);
56 class3SimData = class3Samples(numClass3+1:size(class3Samples, 1), 1:21);
57 simulationData = [class1SimData; class2SimData; class3SimData];
58 tSimClass1 = repmat([1; 0; 0], 1, length(class1SimData));
59 tSimClass2 = repmat([0; 1; 0], 1, length(class2SimData));
60 tSimClass3 = repmat([0; 0; 1], 1, length(class3SimData));
61 % In tSim stehen die SOLL-Werte
62 tSim = horzcat(tSimClass1, tSimClass2, tSimClass3);
63 [ySim, pf, af, eSim, perf] = sim(net, simulationData');
64 [errors, success, totalSuccess] = calculatePerformance(ySim, tSim, ...
65     {class1SimData; class2SimData; class3SimData});
66
67 disp('Simulation mit Komplementaermenge abgeschlossen');
68 disp(sprintf('Fehler: C1: %d/%d (OK: %f), C2: %d/%d (OK: %f), C3: %d/%d (OK: %f
      )',...
69     errors(1), length(class1SimData), success(1),...
70     errors(2), length(class2SimData), success(2),...
71     errors(3), length(class3SimData), success(3)));
72 disp(sprintf('Klassifikationsguete gesamt: %f', totalSuccess));
```

Listing A.2: Training und Simulation des Netzes

```
% Ueberpruefen der Simulationsergebnisse - in jeder Spalte von ySim
% steht ein Ergebnis. In den ersten X Spalten muss Klasse 1 erkannt
% worden sein, dann Klasse 2 usw.
function [errors, success, totalSuccess] = calculatePerformance(ySim, tSim,
    simData)
errors      = [0 0 0];
success     = [0 0 0];
totalSuccess = 0;

numCols = length(ySim);
for col = 1:numCols
    class = -1;
    classValue = -1;
    for row = 1:3
        if ySim(row, col) > classValue,
            class = row;
            classValue = ySim(row, col);
        end
    end

    % Konnte keiner Klasse zugeordnet werden -> Fehler!
    if classValue < 0 | tSim(class, col) ~= 1
        if col <= length(simData{1}),
            errors(1) = errors(1) + 1;
        elseif col <= length(simData{1}) + length(simData{2}),
            errors(2) = errors(2) + 1;
        else
            errors(3) = errors(3) + 1;
        end
    end
end

success(1) = ( 1 - errors(1)/length(simData{1}) ) * 100;
success(2) = ( 1 - errors(2)/length(simData{2}) ) * 100;
success(3) = ( 1 - errors(3)/length(simData{3}) ) * 100;
totalSuccess = ( 1 - sum(errors)/numCols ) * 100;
```

Listing A.3: Berechnen der Fehlerquote bei der Simulation

Quellen

[Bit07] Bittel, Prof. Dr. O.: *Neuronale Netze.* Vorlesungsunterlagen an der HTWG Konstanz, 2007

[DB98] Demuth, Howard ; Beale, Mark: *Neural Network Toolbox User's Guide.* Januar 1998