FACHHOCHSCHULE MANNHEIM
HOCHSCHULE FÜR TECHNIK UND GESTALTUNG
FACHBEREICH NACHRICHTENTECHNIK
68163 Mannheim Windeckstraße 110
Telefon (0621) 292-0 Telefax (0621) 292-6350

Diplomarbeit

Mustererkennung und Musterklassifikation biologischer Signale unter Anwendung Neuronaler Netze

Verfasser: Alexander Scheib
Matrikelnummer: 81061
Adresse: Josefstraße 2
76846 Hauenstein

Studiengang: Nachrichtentechnik/Elektronik
Vertiefungsrichtung: Übertragungstechnik

Bearbeitungszeitraum: 1.Oktober 1994 bis 31.März 1995

Betreuung: Prof. Dipl.-Ing. K. Abel
Zweitkorrektor: Prof. Dr. B. Wirnitzer

erstellt am Institut für Digitale Signalverarbeitung

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit allein und ohne fremde Hilfe ausgearbeitet zu haben.
Es wurden keine anderen als die von mir angegeben Hilfsmittel verwendet.

Diese Arbeit hat meines Wissens in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, 31.März 1995

_____

0 EINLEITUNG 1

1 NEURONALE NETZE 3

1.1 Signalaufbereitung 3

1.1.1 Normierung von Signalen 3

1.1.2 Frequenzselektive Filter 4

1.2 Aufbau Neuronaler Netze 10

1.2.1 Ableitung aus der Neurophysiologie 10

1.2.2 Modell eines Neuron 12

1.2.3 Übertragungsfunktionen 14

1.2.3.1 Schwellwert Übertragungsfunktion 15

1.2.3.2 Lineare Übertragungsfunktion 15

1.2.3.3 Logistik Übertragungsfunktion 15

1.2.3.4 Tangents hyperbolikus Übertragungsfunktion 16

1.2.4 Beispiele verschiedener Unit-Typen 16

1.2.5 Allgemeines Neuronales Netz 18

1.2.6 Feed Forward Netzwerk 18

1.2.7 Feedback Netzwerk 19

1.3 Anwendung Neuronaler Netzwerke 20

1.3.1 Klassifikation neuronaler Netzwerke 20

1.3.2 Lernen in Neuronalen Netzen 20

1.3.2.1 Lernphasen beim überwachten Lernen 22

1.3.3 Das Perceptron 23

1.3.3.1 Anwendungsbeispiel für ein Perceptron 24

1.3.4 Anwendungsbeispiel: Sprecheridentifikation 27

1.3.4.1 Kohonen Netzwerk und Lernregel 28

1.3.4.2 Auswertung der Phonemklassifizierung 30

1.3.5 Backpropagation Netzwerke 34

1.3.5.1 Prinzip des Gradientenverfahrens 34

1.3.5.2 Die Delta Lernregel 36

1.3.5.3 Die Backpropagation-Regel 38

1.3.5.4 Der Momentum-Term 41

1.3.5.5 Adaptive Lernrate 43

1.3.6 Applikation EKG-Erkennung 44

1.3.6.1 Elektrokardiogramm 44

1.3.6.2 EKG Datenakquisition- Vorverarbeitung 46

1.3.6.3 Auswahl und Training des Neuronalen Netzes zur EKG-Erkennung 46

1.3.6.4 Auswertung der EKG-Erkennung mit einem Neuronalen Netz 51

2 KORRELATIONSANALYSE 56

2.1 Korrelationskoeffizient 56

2.2 Korrelationsfunktion 58

2.3 Durchführung einer linearen Korrelation 60

2.4 Auswertung der Kreuzkorrelationsanalyse zur EKG-Erkennung 61

2.5 Kreuzkorrelationsanalyse und Neuronale Netze im Vergleich 66

3 MUSTERERKENNUNG UND KLASSIFIKATION AM ELEKTROMYOGRAMM DER GLATTEN MUSKULATUR 69

3.1 Glatte Muskulatur 69

3.2 Aktionspotentiale 71

3.3 Aufnahme der EMG - Signale 73

3.4 Analyse der EMG - Signale 76

3.4.1 Analyse der Peaks eines EMG 77

3.4.1.1 Maximale Spannung, Standardabweichung und Varianz der EMG - Peaks 79

3.4.1.2 Anstiegszeit, Abfallzeit und Dauer der EMG - Peaks 81

3.4.2 Auswirkung von Noradrenalin und Acetylcholin auf das EMG - Signal 81

3.5 Mustererkennung an EMG - Signalen 84

3.5.1 Motivation für die Anwendung von Mustererkennungsmethoden auf EMG - Signale 84

3.5.2 Auslöschung der EKG-Komplexe aus dem Meßdatensatz 85

3.5.3 Auswahl und Training des Neuronalen Netzes zur EMG-Verifizierung 89

3.5.4 Auswertung der Mustererkennung an EMG-Signalen 92

4 SCHLUßBETRACHTUNG 95

5 LITERATURVERZEICHNIS 97

6 LISTINGS 99

6.1 Erläuterungen zu den Listings und zu MATLAB® 99

6.2 Anmerkungen zur Neuronalen Netze Toolbox von MATLAB® 100

6.3 Script_File: „LOWPASS.M“ 102

6.4 Script_File: „HIGHPASS.M“ 105

6.5 Script_File: „BANDSTOP.M“ 107

6.6 Script_File: „BANDPASS.M“ 110

6.7 Script_File: „PERCEPT.M“ 114

6.8 Script_File: „START-SP.M“ 116

6.9 Scrip-File: „SPRACH11.M“ 116

6.10 Script-File: „SPR-TE11.M“ 119

6.11 Script-File: „SPR_AUS.M“ 121

6.12 Script-File: „BP-EKG3.M“ 123

6.13 Script-File: „BP-EKGT3.M“ 126

6.14 Script-File: „KORRKOEF.M“ 129

6.15 Script-File: „BP-EKG3x.M 131

6.16 Script-File: „BP-EKT3X.M“ 134

6.17 Scrip-File: „EKG-NULL“ 137

6.18 Script-File: „BP-EMG3B.M“ 138

6.19 Script-File: „EMG-PEAK.M“ 141

6.20 Script-File: „START-EM.M“ 144

6.21 Script_File: „TR_TF.M“ 145

6.22 Script-File: „STDVAR.M“ 146

6.23 Script-File: „MEDICA.M“ 148

Einleitung

Während der Entwicklung des Harnblasenstimulators am Institut für Analogtechnik an der Fachhochschule für Technik in Mannheim, entstand die Idee den Blasenfüllungszustand anhand der Muskelsignale abzuleiten.
Die Muskulatur der Harnblase gehört zu der Gruppe der glatten Muskeln. Die Problematik bei der Ableitung des Elektromyogrammes (EMG) der glatten Muskulatur besteht in der Auswertung der sehr kleinen Muskelsignale. Einerseits müssen zur Meßwertaufnahme die entsprechenden Sensoren (z.B. Nadelsysteme), Vorverstärker und Endverstärker zur Verfügung stehen. Andererseits ist eine nachträgliche Bearbeitung der gemessenen Signale mit Signalverarbeitungsmethoden unumgänglich, da die Elektromyogramme durch viele Störsignale überlagert sind.
Bisher hat sich nur eine geringe Anzahl wissenschaftlicher Arbeitsgruppen mit der Messung von EMG-Signalen der glatten Muskulatur befaßt, weshalb in der medizinischen und physiologischen Fachliteratur nur wenige Hinweise zur benötigten Hardware und zur Vorgehensweise bei der EMG-Messung zu finden sind. Desweiteren ist noch nicht endgültig geklärt, welche der bisher klassifizierten Komponenten in den Elektromyogrammen glatter Muskeln für eine Tonusänderung (Spannungsänderung) bzw. für eine Dauerkontraktion ausschlaggebend sind.
Infolge der Vielzahl der Probleme bei der Blasenfüllstandsmessung unter Zuhilfenahme von EMG-Signalen hat man für den Blasenstimulator diesen Lösungsansatz nicht weiter verfolgt. Das Interesse der Mediziner und Physiologen, die Elektromyogramme glatter Muskeln abzuleiten, zu klassifizieren und Korrelationen zu bestimmten Muskelzuständen und Medikamentenapplikationen herzustellen, ist jedoch ungebrochen. So besteht immer noch die Aufgabe, die Sensoren, Verstärker und die Software zur Auswertung der gemessenen EMG-Signale weiter zu entwickeln.
Parallel zu dieser Diplomarbeit wurde der Sensor und der Vorverstärker im Zusammenhang mit der Diplomarbeit von Markus Hauck am Institut für Analogtechnik an der Fachhochschule für Technik in Mannheim optimiert.
Durch Gespräche mit Herrn Prof. Fink und Herrn Prof. Rüegg vom physiologischen Institut der Universität Heidelberg, hat sich herausgestellt, daß mit großer Wahrscheinlichkeit die Innervation der glatten Muskulatur, wie z.B. des Taenia Coli (Dickdarmmuskels), durch Bursts von Aktionspotentialen hervorgerufen wird.
Ein Ziel dieser Diplomarbeit bestand nun darin, diese Peaks (Aktionspotentiale) aus dem abgeleiteten EMG-Signal eines Kaninchendarmes zu extrahieren und zu klassifizieren. Zur Bewältigung dieser Aufgabe kamen künstliche Neuronale Netze, welche mit der Neuronale Netze Toolbox des Programmes MATLAB® simuliert wurden, zum Einsatz.
Mit diesem Neuronale-Netze-Simulator wurden am Institut für digitale Signalverarbeitung der Fachhochschule für Technik in Mannheim bis dahin noch keine Erfahrungen gesammelt. Deshalb sind im ersten Teil dieser Diplomarbeit drei Anwendungsbeispiele für Neuronale Netze, mit unterschiedlich komplexer Aufgabenstellung aufgezeigt. Die Beispiele und Listings sollen unter anderem weiteren Anwendern der Neuronale Netze Toolbox von MATLAB® den Einstieg in dieses Themengebiet erleichtern.
In einem weiteren Kapitel folgt die eigentliche Auswertung der am Institut für digitale Signalverarbeitung der Fachhochschule für Technik in Mannheim aufgenommenen Elektromyogramme des Kaninchendarmes. Zur Automatisierung der Signalauswertung wurde eine Mustererkennung (Pattern Recognition) unter Anwendung Neuronaler Netze realisiert, mit der die Aktionspotentiale des Kaninchendarmes in den aufgenommenen Signalen lokalisiert wurden.

Neuronale Netze

Signalaufbereitung

Für die Signalvorverarbeitung werden herkömmliche Filterungsmethoden benutzt, mit deren Hilfe das Neuronale Netzwerk „entlastet“ wird. Auf diese Weise kann man die Anzahl der Neuronen und somit die Größe der verwendeten Matrizen zur Berechnung des Ausgangssignals minimieren. Bei geschickter Zusammenstellung von konventionellen Filterungsmethoden und künstlichen Neuronalen Netzen kann sich die Bearbeitungszeit zur Ermittlung des Ausgangssignals erheblich verkürzen.

Normierung von Signalen

Bei der Mustererkennung ist die Energie, d.h. die Amplitude des Signals, oft unerheblich. Ist diese Voraussetzung erfüllt, so kann man von Anfang an durch geeignete Normierung des Signals die Information „Energie des Signals“ herausfiltern.

v = Signalvektor
v’ = Normierter Signalvektor

$$v' = \frac{(v_1^2, v_2^2, v_3^2, \ldots, v_n^2)}{|v|^2}$$

sodaß:

$$1 = \sum_{i=1}^{n} v'_i$$

Diese Art der Normierung wird auch von dem verwendeten Mathematikprogramm MATLAB® direkt unterstützt.

Quellen: [4], [7]

Frequenzselektive Filter

Beim Messen von Strömen und/oder Spannungen in elektronischen Schaltkreisen und vor allem an biologischen Organismen können störende Fremdsignale oft nicht durch methodische Optimierung der Meßanordnung eliminiert werden. Deshalb muß eine Trennung der informationstragenden Signale von den Rauschanteilen angestrebt werden. Hierfür stehen die bekannten Filter als FIR Typ (finit impuls response filter) bzw. als IIR Typ (infinite impuls response filter) zur Verfügung. Für die Erstellung dieser Diplomarbeit wurden FIR Filter als Hoch-, Tief- und Bandpaß, sowie als Bandsperre entwickelt. Alle diese Filter wurden als M-Files unter MATLAB® erstellt. Zur Anpassung der Filter auf Pattern-Recognition-Aufgaben (Mustererkennung), wurde der Phasengang der Filter kompensiert. Der Imaginärteil der Übertragungsfunktion der Filter ist somit identisch Null. Dies hat den Vorteil, daß nach der Fourierrücktransformation in den Zeitbereich der Realteil der Rücktransformierten gleich dem Betrag ist (da die Phase = 0° beträgt). So kann das gefilterte Signal ohne weitere Verarbeitung zur Mustererkennung verwendet werden.

Prinzip des Filterentwurfs

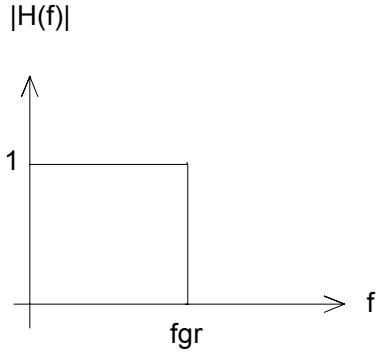
Es wird die unter MATLAB® übliche Vorgehensweise aufgezeigt.
Zunächst wird die gewünschte ideale Übertragungsfunktion im Frequenzbereich vorgegeben. Danach transformiert man diese zurück in den Zeitbereich, womit die Koeffizienten der Diracstoßantwort vorliegen. Anschließend wird die tatsächliche Übertragungsfunktion zurückgerechnet.

$$T_a = \frac{1}{f_a} \quad ; \quad t_{gr} = \frac{1}{f_{gr}}$$

N = Anzahl der Abtastwerte

$$f_0 = \frac{1}{N \cdot T_a} \quad \text{(Frequenzlinienabstand)}$$

Vorgegebene Übertragungsfunktion:



$$H(f) = 1 \cdot e^{-2\pi j f_0 t_{gr}} \quad \text{für} \quad -f_{gr} \le f \le f_{gr} \quad \text{und} \quad 0 \text{ sonst}$$

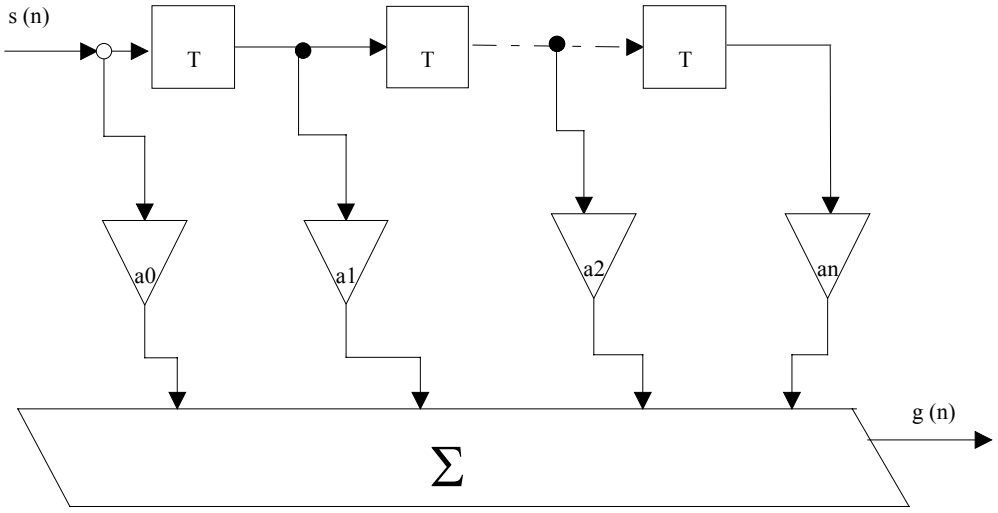
$$H(f) = \left\{ rect(\frac{f}{2 \cdot f_{gr}}) e^{-j2\pi f t_{gr}} \right\} * \sum_{n=-\infty}^{\infty} \delta(f - nf_a)$$

Rücktransformation in den Zeitbereich:

$$h(nT_a) = \left\{ 2 f_{gr} \frac{\sin(2\pi f_{gr} t)}{2\pi f_{gr} t} * \delta(t - t_{gr}) \right\} \frac{1}{f_a} \sum_{n=-\infty}^{\infty} \delta(t - nT_a)$$

$$h(nT_a) = \frac{2 f_{gr}}{f_a} \cdot \frac{\sin\left[2\pi f_{gr}(nT_a - t_{gr})\right]}{2\pi f_{gr}(nT_a - t_{gr})}$$

Abb. 1.1: Blockschaltbild des Filters



Fensterung des Koeffizientensatzes:
Fensterung der Übertragungsfunktion:

H(n) = H(n) W(n)

Berechnung der Übertragungsfunktion:

g(n) = h(n) * s(n)

$$g(n) = \sum_{k=0}^{N-1} h(k) \cdot s(n-k)$$

$$G(z) = \sum_{k=0}^{N-1} h(k) \cdot S(z) z^{-k}$$

$$H(z) = \frac{G(z)}{S(z)} = \sum_{k=0}^{N-1} h(k) z^{-k}$$

$z = e^{j2\pi f T_a}$ mit $f = nf_0 = \frac{n}{NT_a}$ ergibt $z = e^{j2\pi \frac{n}{N}}$

Es ergibt sich die Übertragungsfunktion:

$$\boxed{H(n) = \sum_{k=0}^{N-1} h(k) e^{-j2\pi \frac{nk}{N}}}$$

$$G(n) = H(n) \cdot S(n)$$ (mit Phase des Filters)

$$G(n) = \frac{H(n) \cdot H^*(n)}{|H(N)|} \cdot S(n)$$ (mit Filterphase φ = 0°)

$$\boxed{g(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} G(n) \cdot e^{-j2\pi \frac{kn}{N}}}$$

Die folgenden Darstellungen zeigen eine Anwendung anhand eines PQRS-Komplexes eines EKG-Signals, der mit einem 50 Hz Rauschen überlagert ist. Eine Abbildung wurde mit einem Filter mit linearer Phase erzeugt, die zweite Abbildung zeigt das gefilterte Signal als Ausgangsgröße eines phasenkompensierten Filters.

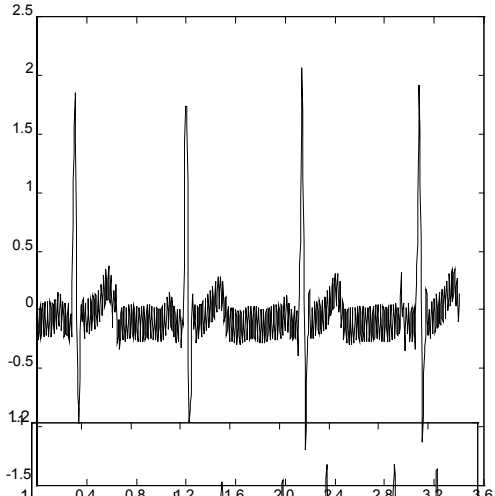


Abb. 1.2: Ungefiltertes EKG-Signal

-Mit überlagertem 50 Hz Rauschen
(Abtastfrequenz = 250 Hz)

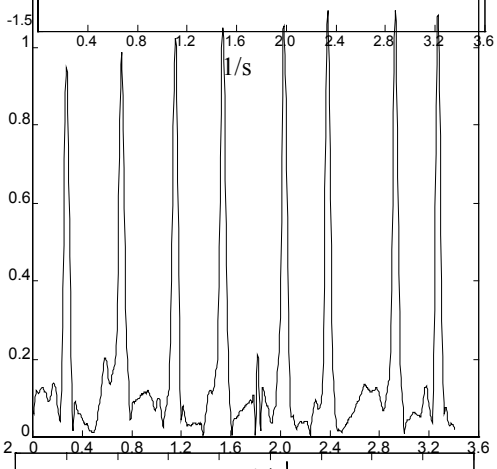


Abb. 1.3: Gefiltertes EKG-Signal

Gefiltert mit einem FIR Tiefpaß
mit linearem Phasengang φ = 45°,
sodaß nach der inversen
Fouriertransformation der Betrag zur
Weiterverarbeitung benutzt werden muß

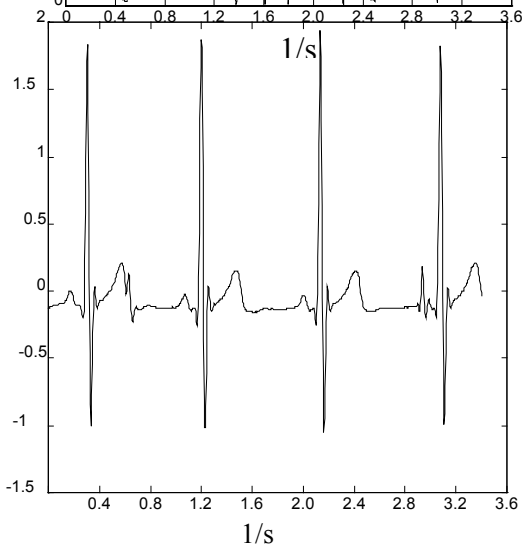


Abb. 1.4: Gefiltertes EKG-Signal

Gefiltert mit einem FIR Tiefpaß
ohne Phasengang φ = 0°,

Programme: LOWPASS.M, HIGHPASS.M, BANDPASS.M, BANDSTOP.M
Quellen: [1], [2], [12], [15]

Aufbau Neuronaler Netze

Ableitung aus der Neurophysiologie

Arbeitsvorgänge zu automatisieren, sowie Maschinen zu steuern und zu regeln ist Grundlage für die Entlastung des Menschen bei den verschiedenartigsten Arbeiten. Um diese Aufgaben zu bewältigen, entstanden unter anderem die heute üblichen Rechner. Deren Aufbau entspricht im allgemeinen der sogenannten Neumann-Architektur, welche für viele Anwendungszwecke sehr gute Ergebnisse liefert. Möchte man allerdings verschiedene Funktionen des menschlichen Gehirns, wie z.B. Assoziationsfähigkeit oder Intelligenz (in irgend einer Form) auf einem Rechner mit Neumann-Architektur nachbilden, so wird man schnell an die Grenzen der Leistungsfähigkeit dieser Maschinen stoßen.
Dies ist einer der Gründe, warum sich schon immer Wissenschaftler mit der Erforschung des menschlichen Gehirns befassen. Man sucht nach dem logischen Aufbau, dem Merkvermögen und den assoziativen Vorgängen im menschlichen Gehirn. Auf anderer Ebene versucht man den chemischen und biologischen Aufbau zu analysieren. Ganz gleich mit welcher Intention die Erforschung des Gehirns heute betrieben wird, sie ist immer auch mit der Idee bzw. Motivation verknüpft, die Ergebnisse dieser Arbeiten auf technische Anwendungen zu übertragen.
Mc Culloch und Pitts entwickelten 1943 das Neuronen-Modell, welches immer noch Grundlage der meisten Neuronalen Netzwerkknoten ist. Dieses Modell wurde von der Physiologie des Menschen abgeleitet.

Das Neuron als kleinstes Bauelement in einem neuronalen Gefüge ist im Grunde genommen eine Zelle mit ganz spezieller Funktion. Es besteht aus einem Zelleib und einem mehr oder weniger langen Ausläufer, dem sogenannten Axon, dem Leitungskabel.
Eine weitere Struktur von Nervenzellen sind die Synapsen. Sie „haften“ an nachgeschalteten Neuronen oder aber auch an Muskelzellen. Über diese Synapsen erfolgt dann die eigentliche Nachrichtenübermittlung von Zelle zu Zelle. Dabei kann ein Neuron mit nur einem anderen oder aber auch mit tausenden anderer Neuronen in Kontakt stehen. Die Synapse fungiert hierbei wie eine Diode als Gleichrichter und verhindert, daß eine Nachricht (Erregung) nicht auf die Ursprungszelle zurückgreifen kann.
Um ein Neuron zu aktivieren, reicht das Eintreffen einer Erregung eines übergeordneten Neurons normal nicht aus. Vielmehr konvergieren viele Neuronen zu diesem einen, so daß durch Summation der eintreffenden einzelnen Informationen ein sogenanntes exzitatorisches (erregendes) Potential zunehmend größer wird, bis ein Schwellwert, bei dem diese Zelle ihrerseits ein Signal aussendet, erreicht wird. Gleichermaßen existieren Informationen inhibitorischer (hemmender) Synapsen, die das Potential zur Erregung des Neurons verringern. Erst die Summe aus exzitatorischen und inhibitorischen Informationen entscheidet, ob in dem entsprechenden Neuron ein Aktionspotential (Erregung) ausgelöst wird oder nicht. Diese Art Neuronen bezeichnet man auch als Schwellwertneuron.
Dieser Neuronentyp ist hier nur exemplarisch für viele Neuronentypen, aus denen das menschliche Gehirn aufgebaut ist, beschrieben.
Für detailgetreuere Darstellungen und Beschreibungen des Aufbaus, sowie der Abläufe innerhalb und außerhalb von Zellen, Neuronen und Synapsen in Organismen sei auf die entsprechende medizinische (physiologische) Literatur verwiesen.

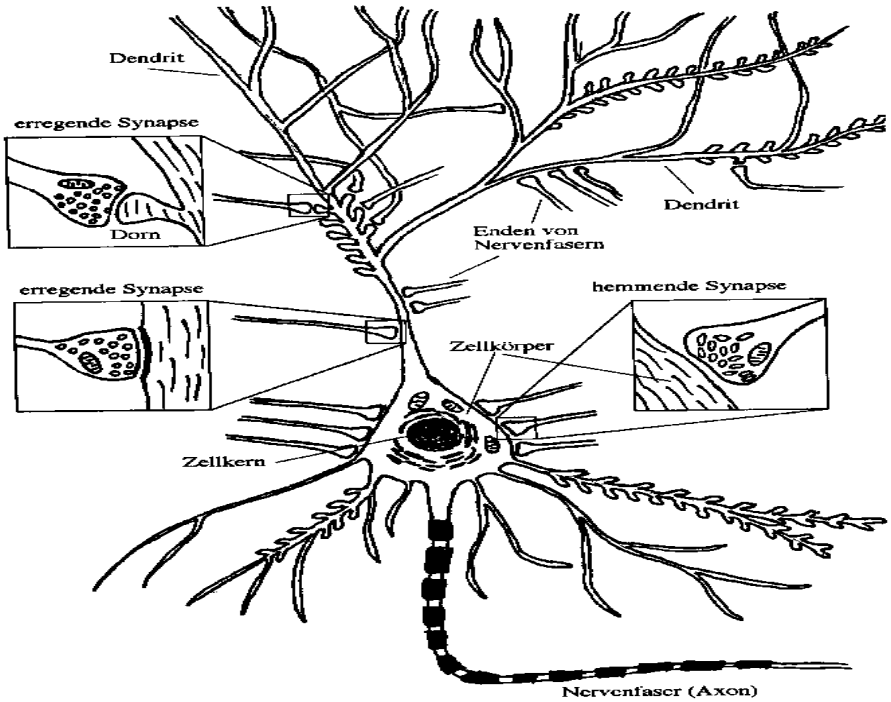


Abb. 1.5: Nervenzelle mit Synapsen (Quelle [22] Seite 38)

Quellen: [8], [10], [11], [19], [22]

Modell eines Neuron

Die kleinsten Elemente Neuronaler Netzwerke bilden die Einheiten oder Units („Neuronen“). Ein Neuron ist in sich in drei Bereiche aufgeteilt:
- ein Eingangsbereich
- ein Aktivierungsbereich
- ein Ausgangsbereich

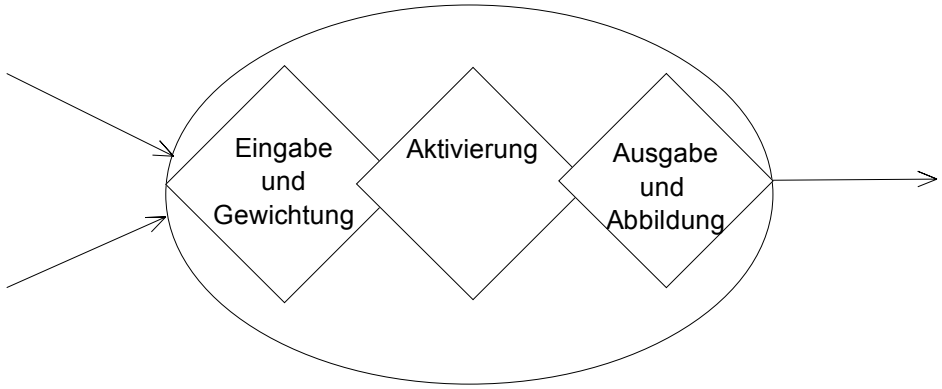


Abb. 1.6: Modell eines Neuron

Eingangsbereich

Im Eingangsbereich wird der sogenannte Netto-Input gebildet. Es werden alle Inputs mit einer Gewichtung multipliziert und aufsummiert. Die Herkunft der einzelnen Inputs hängt vom Aufbau des Netzes ab.

Netto Input: $net[i] = \sum_{j=0}^{N-1} W[i][j] * O_{alt}[j]$

Aktivierungsbereich

Der Aktivierungswert einer Unit wird über die Funktion $F_i(t)$ bestimmt. Oft ist diese Abbildung gleich dem Netto Input (a[i]=net[i]).

Aktivierung bei Neuronen
ohne Gedächtnis : $a_{neu}[i] = F(net[i])$

Aktivierung bei Neuronen
mit Gedächtnis : $a_{neu}[i] = F(a_{alt}[i], net[i])$

Ausgangsbereich

Bei der Bildung des Outputs werden Übertragungsfunktionen angewendet, die z.B. zur Schwellwertbildung dienen. Im trivialen Fall ist f(x)=x. Normalerweise werden sogenannte „sigmoide“ Funktionen (s.u.) verwendet.

Output : $O_{neu}[i] = f(a_{neu}[i])$

Quellen: [11], [19]

Übertragungsfunktionen

Eine Übertragungsfunktion, auch Outputfunktion oder Transferfunktion genannt, definiert das Ausgangssignal einer Unit (eines Neurons), das weitergeleitet werden soll. Die Abbildungsvorschriften dieser Funktionen sind spezifisch auf bestimmte Anwendungen bezogen. Innerhalb eines Neuronalen Netzes mit Hidden Layers, wie sie z.B. in Backpropagation-Netzwerken vorkommen, werden oftmals auch unterschiedliche Übertragungsfunktionen verwendet.
Bei Pattern-Recognition- und Vektor-Recognition-Aufgaben ist z.B. die Hard Limit Funktion oft unangebracht, da hiermit keine Zwischenwerte und somit keine Ähnlichkeitsentscheidungen getroffen werden können.
Zusätzlich ist bei der Auswahl der Übertragungsfunktion immer auch der benutzte Lernalgorithmus und die Aufgabenstellung zu berücksichtigen.
Im Normalfall besitzt jede Einheit ein „Gedächtnis“ . Somit hängt die neue Aktivierung vom Netto-Input und von ihrem alten Aktivierungswert ab. Der Outputvektor einer Einheit stellt also die assoziierte Gedächtnisantwort derselben auf ihren Input dar.
Der Aufbau eines Neuronalen Netzes in Bezug auf die Anzahl und die Topologie der verwendeten Neuronen sowie deren Übertragungsfunktionen, erfordert offensichtlich sehr viel Erfahrung und empirisches Vorgehen. Um diese Arbeit effizienter zu gestalten, wurde im Zusammenhang mit dieser Diplomarbeit mit dem Mathematikprogramm MATLAB® gearbeitet.
Die Neural-Network-Toolbox in MATLAB® stellt unter anderem auch die unten aufgeführten Übertragungsfunktionen zur Verfügung. Sollte man dennoch weitere Übertragungsfunktionen benötigen, können diese selbst erstellt und in den Befehlssatz von MATLAB® aufgenommen werden.

Quellen: [11], [19]

Schwellwert Übertragungsfunktion

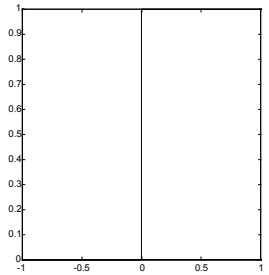


Abb. 1.7: Funktion: hardlim(x+b)

Die Hard-Limit-Funktion entspricht einem Schwellwertschalter oder Komparator, wobei der Bias ‘b’ der Verschiebung der Funktion auf der X-Achse entspricht.

Lineare Übertragungsfunktion

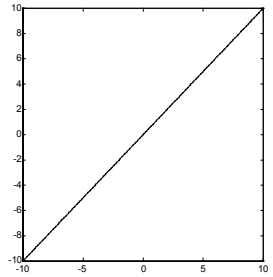


Abb. 1.8: Funktion: purelin(n+b)

Mit der linearen Übertragungsfunktion können die Ausgangswerte des entsprechenden Neuronalen Netzes 1:1 abgebildet werden. Addiert man noch einen Bias hinzu, verschiebt sich die
Übertragungsfunktion nach links bzw. nach rechts.

Logistik Übertragungsfunktion

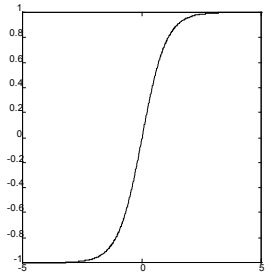


Abb. 1.9: Funktion: logsig(n+b)

Für spezielle Anwendungen bzw. Entscheidungen steht z.B. die Logistik-Übertragungsfunktion zur Verfügung.
Sie wird nach dem Algorithmus a = 1 / (1+exp(-n+b)) gebildet, wobei auch hier der Bias im Prinzip ein Offset bewirkt.

Tangents hyperbolikus Übertragungsfunktion

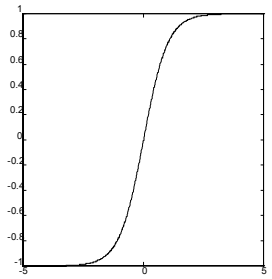


Abb. 1.10: Funktion: tansig(n+b)

Dies ist ebenfalls eine sigmoide Übertragungsfunktion mit nur geringen Unterschieden zur Logistik-Funktion.
Der Algorithmus zur Berechnung lautet:
a = 2 / (1 + exp(-2*(n+b))) - 1.
'b' ist der bekannte Bias mit der gleichen Wirkungsweise wie oben.

Beispiele verschiedener Unit-Typen

Unit-Typ	Aktivierung	Outputfunktion	Netto Input	Lernregel
Sigmoid	keine	sigmoid	$\sum_j W_{ij} O_j$	beliebig, vor allem Backpropagation
Linear	keine	linear	$\sum_j W_{ij} O_j$	Hebb-Regel
Schwellwert	keine	schwellwert	$\sum_j W_{ij} O_j$	Delta-Regel
BSB Brain State in a Box	eigene	keine	$\sum_j W_{ij} O_j$	Hebb-Regel
Thermodynamisch	keine	stochastisch sigmoid	$\sum_j W_{ij} O_j$	Bolzmann-Maschinen
Grossberg	eigene	s-förmig	$\sum_j W_{ij} O_j$	eigene

(Fortsetzung auf der nächsten Seite)

Unit-Typ	Aktivierung	Outputfunktion	Netto Input	Lernregel
IAC Interactive Activation Units	eigene	linearer Schwellwert	$\sum_j W_{ij} O_j$	beliebig
Feldmann Ballard	eigene	schwellwert	$\sum_j W_{ij} O_j$	eigene
Sigma-PI	beliebig	beliebig	$\sum_j W_{ij} \prod_k O_k$	beliebig

Für erläuternde Beschreibung der nicht in dieser Diplomarbeit benutzten und beschriebenen Unit-Typen sei auf die einschlägige Fachliteratur verwiesen. Diese Tabelle erhebt keinen Anspruch auf Vollständigkeit, sondern sie will lediglich eine kurze Übersicht bieten.

Quelle: [11]

Allgemeines Neuronales Netz

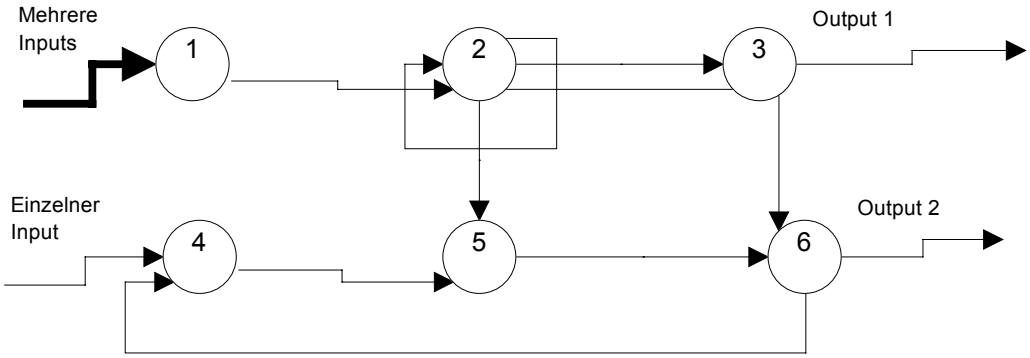


Abb. 1.11: Allgemeines Neuronales Netz

In einem allgemeinen Neuronalen Netz sind alle denkbaren Verbindungen der Neuronen untereinander zulässig, wodurch die Komplexität eines solchen Netzes beliebig ansteigen kann. Dies entspricht den natürlichen Gegebenheiten des zentralen Nervensystems eines Menschen, weshalb die Funktionsweise des menschlichen Gehirns nur sehr schwer zu analysieren ist. Für technische Anwendungen, vor allem auf seriell arbeitenden Rechnern, werden deshalb verschiedene Netzwerk-Topologien spezifiziert.
Quellen: [11],[22]

Feed-Forward-Netzwerk

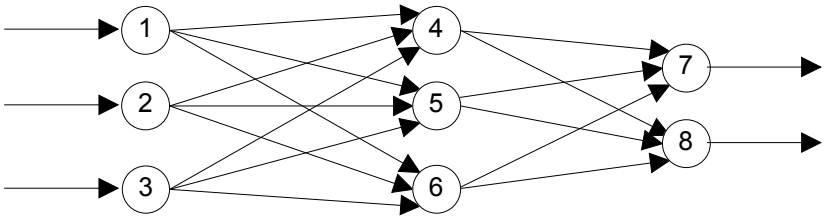


Abb. 1.12: Feed-Foreword-Netzwerk

Die Darstellung zeigt ein Feed-Forward-Netzwerk, das aus acht Neuronen besteht. Die Neuronen sind in drei Schichten („Layers“) angeordnet. Da der mittlere Layer keine direkte Verbindung nach außen besitzt, wird er als Hidden Layer bezeichnet. Charakterisiert wird das Netzwerk, wie schon der Name ausdrückt, durch Neuronenverbindungen, die nur in Richtung des Ausgangs zeigen. Ein solches Netz hat den Vorteil, daß es immer einen stabilen Ausgangszustand erreichen wird.
Quellen: [11], [22]

Feedback-Netzwerk

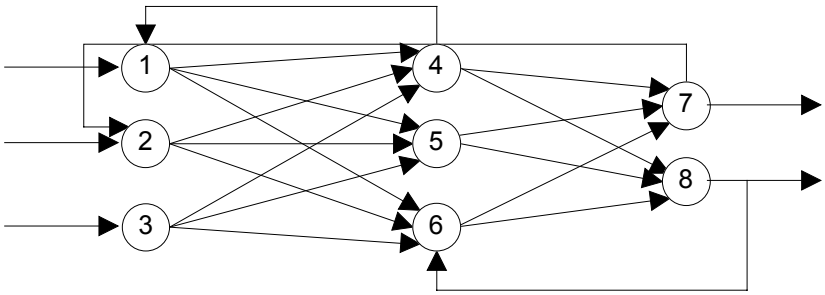


Abb. 1.13: Allgemeines Feedback-Netzwerk

Feedback-Systeme besitzen im Gegensatz zu Feed-Forward-Netzen keine Hierarchie und sind praktisch ohne Ordnung. Durch das Auftreten von Rekursionen ergeben sich interessante Möglichkeiten. Allerdings muß berücksichtigt werden, daß solche Systeme zum Schwingen neigen. Die Meinungen über die Lernfähigkeit hierarchischer gegenüber nicht hierarchischer Systeme gehen auseinander.
Minsky und Papert haben gezeigt, daß man jedes rekursive Netzwerk in ein Feed-Forward- Netzwerk umwandeln kann. Ein weiteres Problem stellt der Zeitpunkt der Aktivierungsregel, d.h. das Update der Units dar. Besteht etwas ähnliches wie ein gemeinsamer Takt, so spricht man von einem synchronen System (gleichzeitiges Update aller Units).
Eine erste Anwendung von Feedback-Netzwerken zeigte Fukushima mit seinem erweiterten Neocognitron. Hierbei verwendete er hemmende (inhibitorische) und erregende (exzitatorische) Rückverbindungen, um die Erkennung von relevanten Merkmalen in Mustern zu beschleunigen. Dieses Verfahren nannte er „Associative Recall“.

Quellen: [11], [22]

Anwendung Neuronaler Netzwerke

Klassifikation neuronaler Netzwerke

Die Einteilung neuronaler Netzwerke in unterschiedliche Typen wird im Normalfall nach der verwendeten Lernmethode vorgenommen. Diese bestimmt auch automatisch, ob ein Netzwerk einen rückgekoppelten Zweig oder mehrere enthalten darf bzw. ganz ohne Rückkopplungen auskommen muß. Außerdem wird durch den ausgewählten Lernalgorithmus die Anzahl der Layer eines Netzwerkes festgelegt, d.h. ob nur eine Schicht, zwei oder mehrere zulässig sind. Dementsprechend spricht man z.B. von einem „Hebb-Netzwerk“, wenn zum Trainieren der Musterdatensätze die Hebb-Regel verwendet wurde.
Quellen: [7], [11]

Lernen in Neuronalen Netzen

Lernen in Neuronalen Netzen bedeutet üblicherweise die Veränderung der Verbindungsgewichte, so daß die Ausgabe des Netzwerkes auf bestimmte Eingabemuster der gewünschten bzw. vorgegebenen Ausgabe entspricht. In vielen Fällen, z.B. bei der Klassifizierung unbekannter Signale, ist die Vorgabe der Netzwerkantwort allerdings nicht möglich. Dies führt zu den drei Arten des Lernens in Neuronalen Netzen.

- Überwachtes Lernen (supervised learning)

- Bestärkendes Lernen (reinforcement learning)

- Unüberwachtes Lernen (unsupervised learning, self-organized learning)

Beim überwachten Lernen wird zu jedem Eingabemuster das gewünschte Ausgabemuster vorgegeben. Dieser externe „Lehrer“ zwingt das Neuronale Netz, die Einstellung aller seiner Verbindungsgewichte so zu verändern, daß es möglichst genau die vorgegebenen Ausgangsvektoren für die entsprechenden Eingangsvektoren berechnen kann.

Durch wiederholtes Repräsentieren vieler Paare von Eingabe- und Ausgabemustern kann eine gewisse Generalisierung erreicht werden, d.h. das Neuronale Netz assoziiert selbständig auf ähnliche aber nicht gelernte Eingabemuster mit einem entsprechenden (richtigen) Ausgabemuster. Diese Art, ein Neuronales Netzwerk zu trainieren, ist üblicherweise die schnellste Methode. Allerdings ist hierbei die Vorgabe der gewünschten Aktivierungen aller Ausgangsneuronen notwendig.

Das bestärkende Lernen benötigt im Prinzip ebenfalls alle Paare der Eingabe- und Ausgabemuster. Hier teilt der „Lehrer“ diese dem Neuronalen Netz aber nicht mit. Nach jedem Trainingsdurchlauf wird nur angegeben, ob das Eingabemuster richtig oder falsch klassifiziert wurde. Das Netz hat nun die Aufgabe, selbst die optimale bzw. richtige Lösung zu finden. Diese Art des Lernens ist schon deutlich langsamer als überwachtes Lernen.

Unüberwachtes Lernen benötigt überhaupt keinen externen Lehrer. Dem Netz werden nur Eingabemuster angeboten. Dieses versucht dann die Muster nach Gemeinsamkeiten oder Ähnlichkeiten zu klassifizieren, indem es Aktivierung gleicher oder räumlich benachbarter Neuronen identifiziert.

Auf diese Weise extrahiert das Neuronale Netzwerk statistische Parameter aus den Eingabemustern, um mit Hilfe derer die Muster zu klassifizieren. Das bekannteste Beispiel für unüberwachtes Lernen sind die Feature Maps von Kohonen (auch SOM: self-organized Maps).

Quellen: [7], [11], [22]

Lernphasen beim überwachten Lernen

Da Neuronale Netze aus der Natur abgeleitet wurden, stellt sich auch immer wieder die Frage nach der biologischen Plausibilität. Nach diesem Kriterium würde das unüberwachte Lernen die erste Wahl bedeuten. In technischen Problemstellungen sind aber Implementierbarkeit und Lerngeschwindigkeit ausschlaggebend, weshalb überwiegend überwachtes Lernen zur Anwendung kommt. Deshalb sollen hier nur für diese Methode die Phasen während des Lernvorgangs beschrieben werden. Die Vorgehensweisen beim unüberwachten Lernen und beim bestärkenden Lernen sind jedoch sehr ähnlich.

a) Präsentation des Eingabemusters durch entsprechende Aktivierung der Eingabeneuronen (input units) ,
b) Vorwärtspropagierung der angelegten Eingabe durch das Netz (bei mehrschichtigen Netzen); Ausgabemuster für die aktuelle Eingabe erzeugen,
c) Vergleich der Ausgabe mit der erwünschten Ausgabe (teaching input) liefert einen Fehlervektor (Differenz, Delta),
d) Rückwärtspropagierung der Fehler von der Ausgabeschicht zur Eingabe führt zu Änderungen der Verbindungsgewichte, um den Fehlervektor zu verringern,
e) Anpassung der Gewichte aller Neuronen entsprechend der berechneten Änderungen.

Dieses Schema für die Vorgehensweise innerhalb der Lernphase ist Grundlage aller überwachter Lernverfahren für nicht rekurrente Netze.

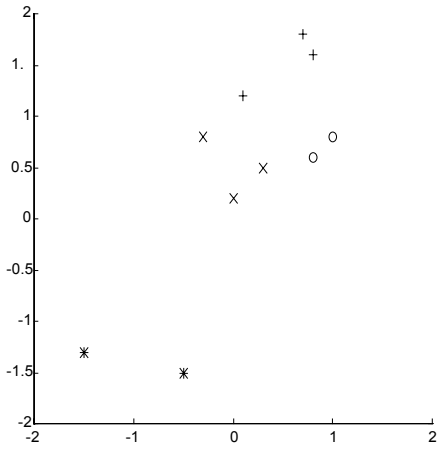
Quellen: [7], [19], [22]

Das Perceptron

F. Rosenblatt entwickelte einige verschiedene Variationen eines einfachen Neuronalen Netzwerkes, welches er Perceptron nannte. Später wurde dieses von Minsky und Pappert rigoros mathematisch analysiert, die ihrerseits noch weitere Varianten des Perceptrons definierten. Man kann also sagen, daß es „das Perceptron“ nicht gibt. In der Literatur findet man dementsprechend verschiedene Definitionen des Perceptrons. Im allgemeinen stellt ein Perceptron ein Feed-Forward-Netzwerk mit ebenenweiser Verbindung dar, d.h. jedes Neuron einer Schicht ist mit allen Neuronen der Vorgängerschicht verbunden.
Perceptrons werden üblicherweise zur Musterklassifizierung verwendet. Dies ist allerdings nur dann möglich, wenn die Eingangsvektoren linear separierbar sind. Daraus resultiert auch die Ursache, warum einschichtige Perceptrons nicht in der Lage sind, das XOR-Problem zu lösen (s. Bsp.).

Quellen: [7], [11], [19]

Anwendungsbeispiel für ein Perceptron



Ein einfaches Beispiel soll die Funktion eines Perceptrons zeigen. Es werden zehn Eingangsvektoren vorgegeben, wobei diese in vier Klassen einzuordnen sind.
Das Perceptron besitzt zwei Eingänge (einer für alle x-Werte, einer für alle y-Werte) und zwei Neuronen in einer Schicht. Als Übertragungsfunktion wurde, wie beim binären Modell des Perceptrons üblich, die Schwellwertfunktion ausgewählt.
Abb. 1.14: Anordnung der Muster

Um die Gewichte einzustellen, wurde die in MATLAB® vorgesehene Perceptron Lernregel verwendet, wobei folgende Definitionen gelten:

P = RxQ Matrix des Eingangsvektors
A = SxQ Matrix des Ausgangsvektors
T = SxQ Matrix des Ergebnisvektors
S = Anzahl der Neuronen
R = Anzahl der Werte eines Musters
Q = Anzahl der Paare von Muster und Ausgabevektoren

<u>Die Perceptron-Lernregel:</u>
Für alle i gilt...
wenn A(i) = T(i) findet keine Veränderung der Gewichte statt,
wenn A(i) = 0 & T(i) = 1,
wird dW(i,j) = P(j) und
W = W + dW gesetzt (für alle j),
wenn A(i) = 1 & T(i) = 0,
wird dW(i,j) = -P(j) und
W = W + dW gesetzt (für alle j).

Ablauf des Netzwerktrainings

Die Effektivität des Trainings eines Neuronalen Netzwerkes hängt sehr stark von der Vorgehensweise während der Trainingsphase ab. In diesem Beispiel wurden folgende Schritte nacheinander durchgeführt:

- Präsentationsphase: Es werden dem Netzwerk alle Eingangsvektoren angeboten und die entsprechenden Ausgangsvektoren berechnet;

- Vergleichsphase : Vergleich der berechneten Ausgangsvektoren mit den vorgegebenen Targetvektoren (soll - Ausgangsvektoren), Abbruch der Trainingsphase falls Übereinstimmung besteht oder falls eine festgelegte Anzahl der Epochen (Trainingsdurchläufe) überschritten wurde;

- Lernphase : Gewichte und Bias (Verschiebung des Schwellwertes der Transferfunktion) nach der Perceptronlernregel verändern;

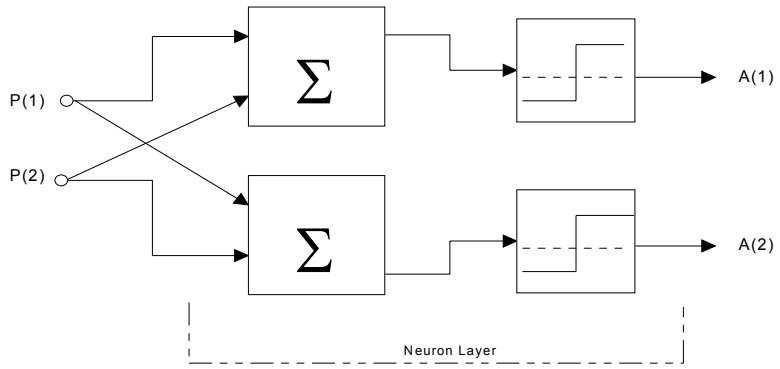
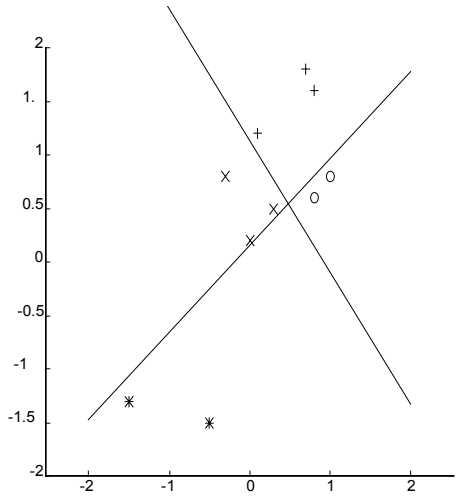


Abb. 1.15: Topologie des verwendeten Perceptrons:

Nach einer kurzen Trainingsphase erhält man folgendes Ergebnis:



Wie man sieht, konnte das Perceptron eine eindeutige Klassifizierung der Mustervektoren vornehmen. Dies war allerdings nur möglich, da eine lineare Separation durchgeführt werden konnte.
Das Netzwerk kann nun angebotene Eingangsvektoren eindeutig einer der vier Klassen zuordnen.

Abb. 1.16: Klassifizierte Muster

Soll ein Netzwerk das XOR Problem lösen, muß es die Punkte P1(0;0), P2(1;1) mit dem Merkmal ‘1’ und die Punkte P3(0;1), P4(1;0) mit dem Merkmal ‘0’ separieren. Nun ist leicht einsehbar, daß diese Vektoren nicht linear separierbar sind. Folglich kann ein einschichtiges Perceptron dieses Problem nicht lösen.
Dieses einfache Beispiel soll die grundsätzliche Arbeitsweise eines Neuronalen Netzwerkes verdeutlichen. Im folgenden wird anhand eines weiteren Beispiels ein Multilayer-Netzwerk vorgestellt, welches eine Weiterentwicklung des Perceptrons darstellt. Grundsätzlich können fast alle Netzwerktypen und Lernregeln als Weiterentwicklung bzw. Spezialisierung des Perceptrons bzw. der Perceptron-Lernregel angesehen werden.

Programme: PERCEPT.M
Quellen: [7], [13], [14]

Anwendungsbeispiel: Sprecheridentifikation

Ein klassisches Problem der Signalverarbeitung ist die Phonemerkennung und die Spracherkennung. Die Schwierigkeit hierbei besteht darin, den Aufwand für die verwendete Hardware gering zu halten und gleichzeitig eine hohe Verarbeitungsgeschwindigkeit zu erreichen. Zusätzlich ist eine große Erkennsicherheit Voraussetzung für eine industrielle Nutzung. Mit Methoden der klassischen Signalverarbeitung kann man diese Anforderungen auch heute noch nur mit größtem finanziellen und technischem Aufwand erfüllen. Deshalb werden immer neue Ansätze zur Lösung dieses Problems gesucht.
Ein Ansatz ist die Anwendung Neuronaler Netze. Teuvo Kohonen entwickelte speziell hierfür bestimmte Neuronale Netzwerke und die dazu benötigten Lernalgorithmen. Das von ihm entworfene System erkennt Texte über Phoneme aus Leistungsdichtespektren gesprochener Sprache. Das für Finnisch und Japanisch implementierte System „Phonetic Typewriter“ erzielt für unbeschränkten Text eine bis zu 97-prozentig richtige Ausgabe in Echtzeit.
Die Idee zur Verwendung der Phonemerkennung als Beispiel für die Anwendung Neuronaler Netze gab die Betreuung des Signalprozessoren-Projektlabors während der Erstellung der Diplomarbeit. Eine Aufgabe bestand darin, die Leistungsfähigkeit einfacher Neuronaler Netze, z.B. für den Aufbau einer Zugangskontrolle, mittels Phonemerkennung zu ermitteln. Einfache Neuronale Netze sollten es deshalb sein, um bei der Realisierung den Aufwand für Hardware und Software gering zu halten. Abweichend von der Vorgabe für die Laborteilnehmer wurde ein einstufiges Feed-Forward-Netzwerk mit der Kohonen-Lernregel trainiert.

Datenakquisition-Vorverarbeitung

Zur Aufnahme der Sprache (Phoneme) wurde ein Kondensatormikrophon an eine AD-Wandlerkarte mit einer Auflösung von 16 Bit angeschlossen. Der dazwischengeschaltete Verstärker diente einerseits zur Impendanzanpassung und andererseits zur Verstärkung des Sprachsignals, um den Aussteuerungsbereich der AD-Wandlerkarte optimal auszunutzen. Parallel hierzu diente ein Oszilloskop zur permanenten Überwachung des Zeitsignals während der Aufnahme. Die Aufnahmen bestanden aus den jeweils sechs mal gesprochenen Buchstaben „a“ und „b“ sowie aus den gesprochenen Worten „eins“ und „zwei“ von insgesamt sechs verschiedenen Sprechern.
Anschließend wurden die binären Daten mit MATLAB® in Dezimalformat konvertiert und, wie bei der Sprachverarbeitung üblich, die Leistungsdichtespektren ermittelt. Das Leistungsdichtespektrum errechnet sich aus dem Betragsquadrat der Fouriertransformierten. Es bot sich an, die Transformation mit 8192 Stützstellen durchzuführen, da in MATLAB® für eine 2^n fache FFT ein in der Berechnungszeit optimierter Algorithmus zur Verfügung steht.

Berechnung des Leistungsdichtespektrums:

$$L_n(f) = \left| \sum_{k=0}^{N-1} s(k) e^{-j2\pi \frac{nk}{N}} \right|^2$$

Da das Leistungsdichtespektrum periodisch ist, wurden zur Datenreduzierung (ohne Informationsverlust) nur die ersten 4096 Werte zur weiteren Verarbeitung abgespeichert.

Quellen: [2], [6], [8]

Kohonen-Netzwerk und -Lernregel

Die Kohonen-Lernregeln basieren auf Ähnlichkeits - bzw. Nachbarschaftsbeziehungen. Kohonen entwickelte bisher mehrere Lernregeln wie z.B.:
LVQ1 (Lernende Vektorquantisierung), LVQ2.1, LVQ3, sowie die Selbstorganisierenden Karten (self - organizing maps, Kohonen maps oder auch Kohonen feature maps genannt). Für die Phonemzuordnung wurde die in MATLAB® zur Verfügung stehende einfache Form der Kohonen Lernregel verwendet. Es ist zu beachten, daß zwischen der Neural-Network-Toolbox für MATLAB® Version 4.0 und Version 4.2c geringfügige Unterschiede bei verschiedenen Lernalgorithmen bestehen.

Kohonen-Lernregel:

$$\Delta W(i,j) = lr \cdot (P(j) - W(i,j)) \forall i$$
$$W = W + \Delta W$$

lr = Lernrate
P = Matrix der zu lernenden Eingangsvektoren
W = Matrix der Gewichte
Bei dieser Lernregel werden diejenigen Gewichte am stärksten verändert, welche sich am meisten vom Output ihres Aktivierungsneurons unterscheiden. Dabei ist eine Gewichtsänderung in positiver wie auch in negativer Richtung möglich, sodaß die Gewichte eines vollständig trainierten Netzes eine Abbildung ihres Trainingsvektors darstellen.

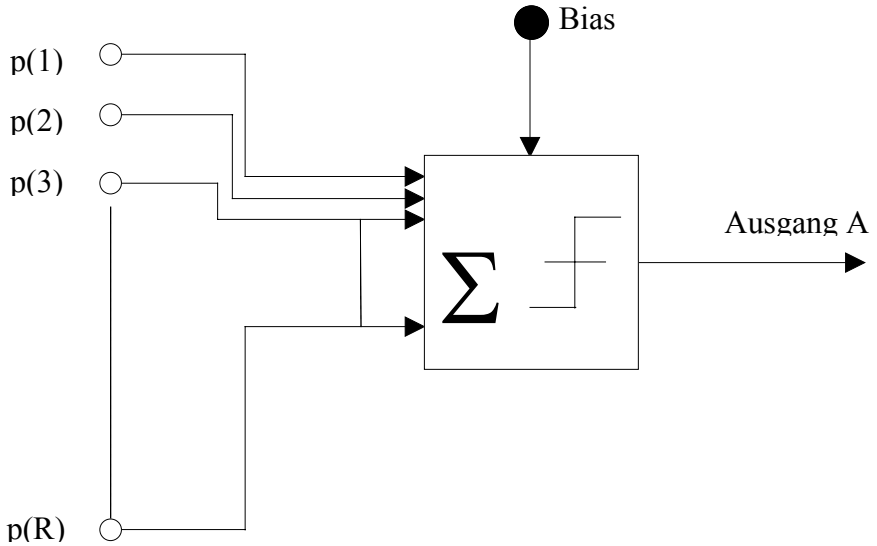


Abb. 1.17: Topologie des verwendeten einfachen Netzwerkes

Die Eingangsneuronen besitzen keine besondere Übertragungsfunktion. Sofwareseitig wurden sie überhaupt nicht realisiert, da sie durch einfaches Anlegen des Eingabevektors an das Ausgangsneuron ersetzt werden können.

Die Leistungsdichtespektren, welche als Eingangsvektoren an das Netz angelegt werden, unterliegen außer einer Standardnormierung (s.o.) keiner weiteren Vorverarbeitung. Auf diese Weise erhält man beim Anlegen des Trainingsmusters an das vollständig trainierte Netz einen Zahlenwert nahe eins. Dieser Netzwerkoutput konvergiert immer näher zu eins, je kleiner die Lernrate gewählt wird.
Der Bias bestimmt die Toleranz der Phonemklassifizierung, d.h. ein Bias von 0.95 setzt eine sehr gute Übereinstimmung der trainierten Muster mit den zu erkennenden Mustern voraus. Durch den Versuchsaufbau, sowie kleinen Variationen der Sprecherstimme während der mehrmaligen Aufnahme gleicher Phoneme, ist dies jedoch nicht gewährleistet. Für die Einstellung des Bias gibt es grundsätzlich keine Vorgabe. Vielmehr muß anhand empirischer Auswertung der Testläufe die Größe des Bias für jedes Phonem neu eingestellt werden.

Quellen: [2], [7], [8], [12], [15]

Auswertung der Phonemklassifizierung

Es wurden insgesamt acht Kohonen-Netzwerke trainiert. Als Trainingsmuster dienten jeweils drei Leistungsdichtespektren als Vorlage für zu erkennende Muster und jeweils sechs Leistungsdichtespektren als Vorlage für falsche Muster. So wurde z.B. ein Netzwerk für die Erkennung gesprochener „a“ von Sprecher eins trainiert. Von Sprecher eins sind hierfür sechs gesprochene „a“ aufgenommen worden. Mit den ersten drei aufgenommenen „a“ und mit sechs davon unterschiedlich aufgenommenen Phonemen wurde das Netzwerk so trainiert, daß es auf gesprochene „a“ von Sprecher eins mit einer „1“ am Ausgang reagiert. Auf alle andere Leistungsdichtespektren sollte das Outputneuron mit „0“ antworten.
Auf diese Weise ist eine einfache Sprecheridentifikation realisiert worden. Die Ergebnisse der Testläufe sind in den folgenden Diagrammen dargestellt.

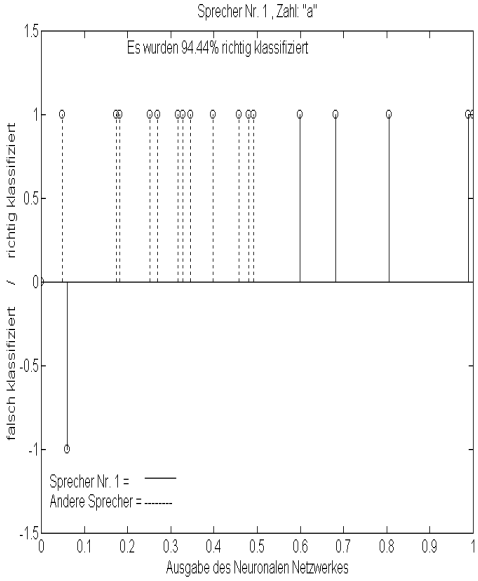


Abb. 1.18

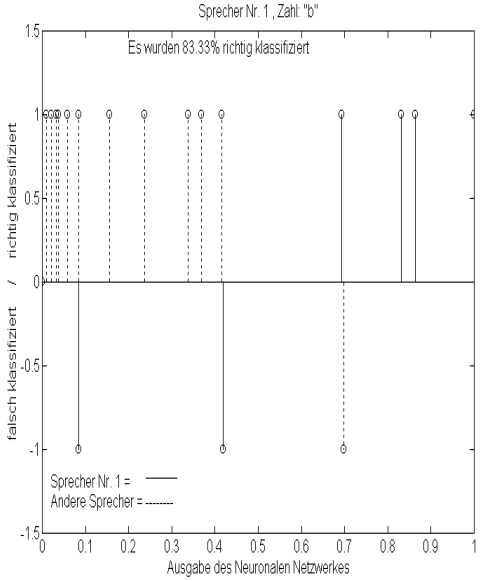


Abb. 1.19

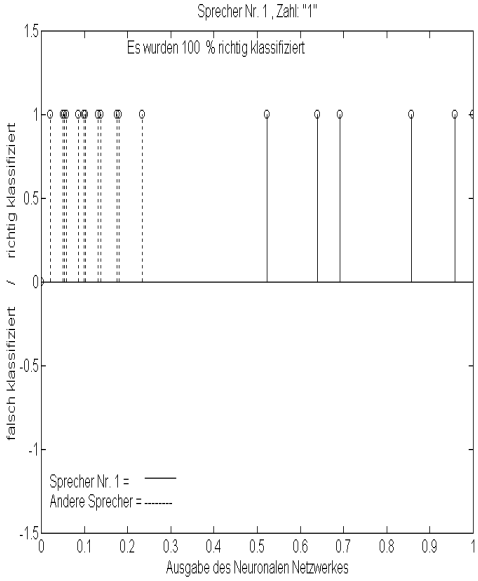


Abb. 1.20

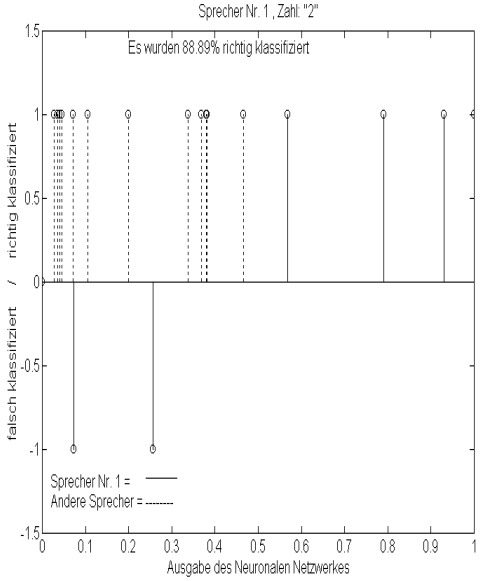


Abb. 1.21

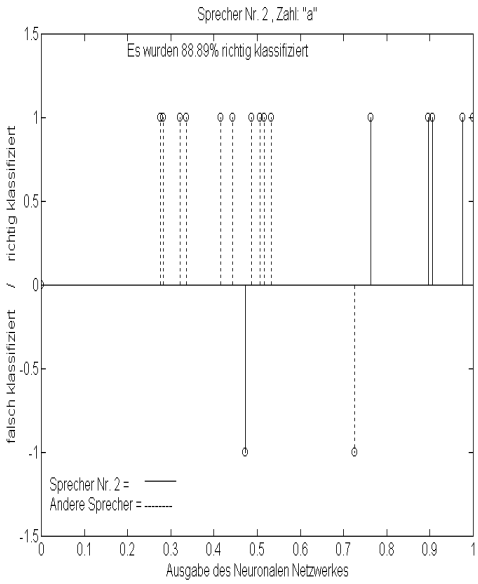


Abb. 1.22

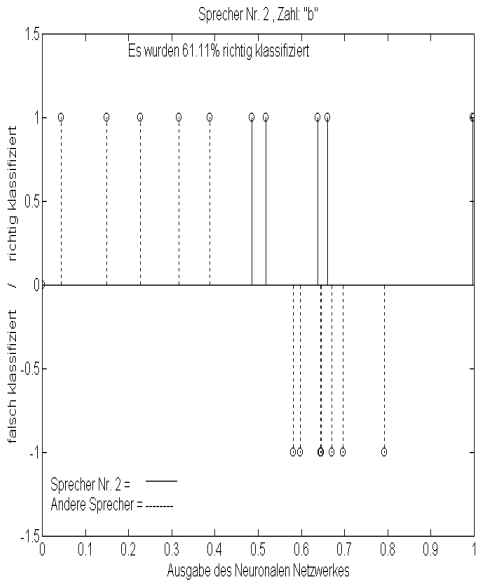


Abb. 1.23

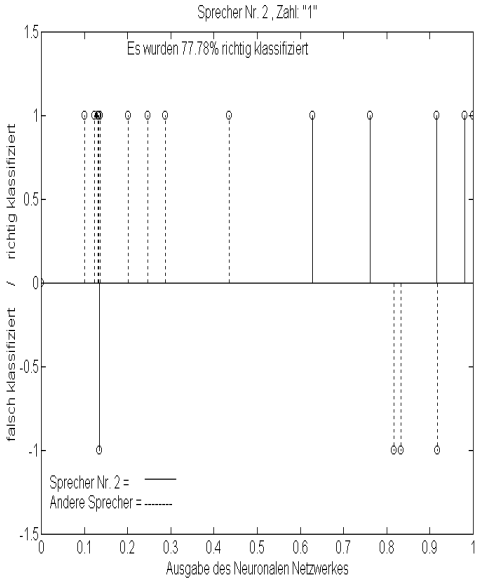


Abb. 1.24

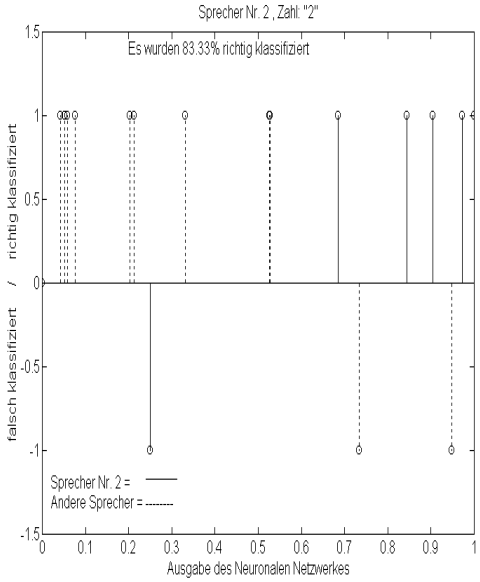


Abb. 1.25

Nach oben mit dem Wert eins sind die richtig klassifizierten Muster aufgetragen. Nach unten mit dem Wert minus eins sind die falsch klassifizierten Muster aufgetragen. Dabei kennzeichnen die mit durchgezogener Linie gezeichneten Einträge die vom zu erkennenden Sprecher gesprochenen Phoneme. Die mit unterbrochener Linie gezeichneten Einträge sind gleiche Phoneme anderer Sprecher. Der Schwellwert (Bias) für die endgültige Zuordnung eines angelegten Musters, wurde für jedes der acht Netzwerke einzeln festgelegt. Die Werte befinden sich im Bereich: $0{,}55 < BIAS < 0{,}7$

Bildet man das arithmetische Mittel der richtig klassifizierten Muster, so erhält man eine mittlere Erkennsicherheit von 84,72 %. Dies ist ein beachtlicher Wert, wenn man die Einfachheit des verwendeten Netzwerkes in Betracht zieht.

Für eine kommerzielle Nutzung reicht diese Erkennsicherheit sicherlich nicht aus. Allerdings ist zu erwarten, daß ein mehrschichtiges Neuronales Netz wesentlich mehr Merkmale aus den angebotenen Signalen extrahieren kann, womit man eine gesteigerte Erkennsicherheit erhalten würde. Des weiteren wäre die Anzahl der Stützstellen der Fouriertransformation zu optimieren, um nur die relevanten Frequenzanteile zur Mustererkennung zu nutzen. Dies hätte eine Datenreduktion und (begrenzt) eine Reduktion von hochfrequenten Rauschanteilen zur Folge. Kohonen benutzte für seinen „Phonetic Typewriter“ eine 256-Punkt-Fast-Fourier- Transformation, um die Frequenzspektren von Phonemen zu berechnen.

In dieser Diplomarbeit sind keine weiteren Untersuchungen zur Sprecheridentifikation über Phonemklassifizierung angestellt worden, da dies nur ein Anwendungsbeispiel Neuronaler Netze unter Verwendung von MATLAB® sein sollte.

Programme: START_SP.M, SPRACH11.M, SPRACH12.M, SPRACH1A.M, SPRACH1B.M
SPRACH21.M, SPRACH22.M, SPRACH2A.M, SPRACH2B.M, SPR11-TE.M
SPR12-TE.M, SPR1A-TE.M, SPR1B-TE.M, SPR21-TE.M, SPR22-TE.M
SPR2A-TE.M, SPR2B-TE.M, SPR_AUS.M
Quellen: [11], [22]

Backpropagation-Netzwerke

Eine weitere wichtige Lernregel für Neuronale Netzwerke stellt die Backpropagation Lernregel dar. Ihr gravierender Vorteil besteht darin, theoretisch beliebig viele Neuronenlayer seriell zu verknüpfen und entsprechend zu trainieren. Dadurch erhält man Netzwerke, welche im Vergleich zu einstufigen Netzwerken mehr Merkmale aus angelegten Mustern extrahieren können. Diese sind auch in der Lage, Aufgaben wie das XOR-Problem zu lösen. Um das Lernen in einem Backpropagation Netzwerk verstehen zu können, müssen zunächst das Prinzip des Gradientenabstiegverfahrens und die Delta-Lernregel erläutert werden.

Prinzip des Gradientenverfahrens

Wie bei allen Gradientenverfahren wird auch hier ein Gradient berechnet, der die Stelle des steilsten Anstiegs bzw. Abfalls einer Funktion angibt. Bei Neuronalen Netzen wird dafür die Fehlerfunktion E(W,B) des Netzwerkes zu grunde gelegt. Die Fehlerfunktion bildet sich aus der Differenz der dem Netzwerk angebotenen Trainingsvektoren und der daraus resultierenden Netzwerkantwort. Steigt man nun orthogonal zum Gradienten nach oben, wird ein Maximum erreicht. In der entgegengesetzten Richtung wird man ein Minimum der Funktion erhalten.

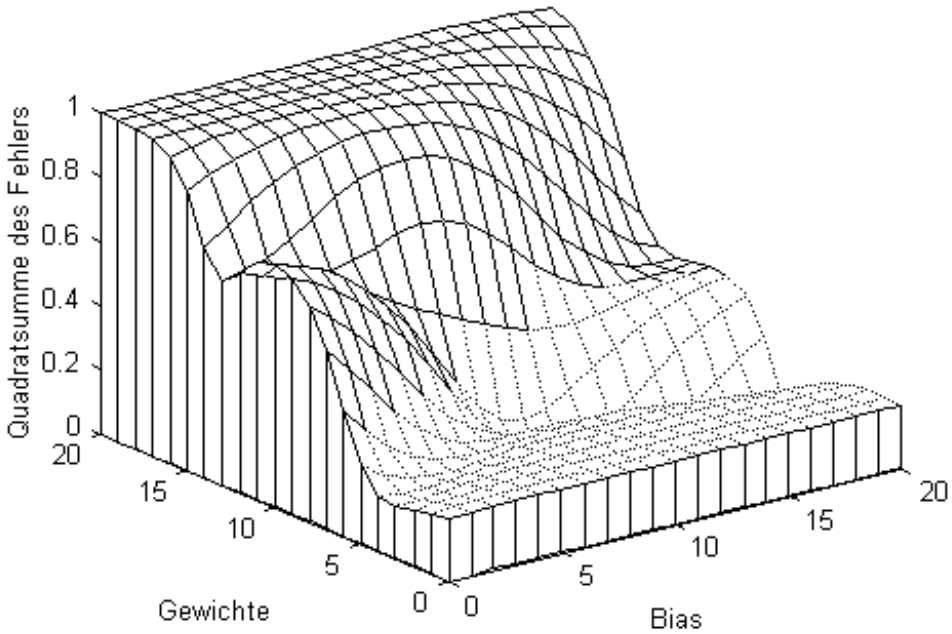


Abb. 1.25: Beispiel einer Fehlerfläche eines Neuronalen Netzes mit einem Neuron in einer Schicht

Das Ziel bei der Anwendung von Gradientenverfahren in Neuronalen Netzen ist das globale Minimum der Fehlerfunktion zu erreichen. Deshalb wird versucht, alle Gewichte um einen Bruchteil des negativen Gradienten der Fehlerfunktion so zu verändern, daß der Fehler minimiert wird.

Bestimmung der Fehlerfunktion (Summe der Fehler über alle Muster p):

E_p = Fehler für Muster p

t_{pj} = Lerneingabe

o_{pj} = Ausgabe von Neuron j bei Muster p

$$E_p = \sum_j (t_{pj} - o_{pj})^2$$

$$E = \sum_p E_p$$

<u>Berechnung der Gewichtsänderung:</u>

$$\nabla = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z} \qquad \text{(Nablaoperator)}$$

$$grad(E) = \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} + \frac{\partial E_z}{\partial z} = \nabla E$$

Mit dem Lernfaktor lr erhält man:

$$\Delta W = -lr \cdot \nabla E$$

Für die Änderung eines einzelnen Gewichtes gilt somit:

$$\Delta W_{ij} = -lr \frac{\partial}{\partial W_{ij}} E$$

Quellen: [7], [22]

Die Delta-Lernregel

Mit der Delta-Regel ist es möglich, zu einem bestimmten Eingabevektor einen gewünschten Ausgabevektor zu assoziieren. Das Teaching muß allerdings im sogenannten Batch-Verfahren (Offline-Verfahren) geschehen, d.h. es müssen dem System erst alle zu trainierenden Muster präsentiert werden, bevor die Veränderung der Gewichte vorgenommen werden kann. Dadurch ist ein Nachtrainieren eines weiteren Musters nicht möglich. Um diese Einschränkung zu beheben wird die Delta-Regel etwas modifiziert, so daß Online-Training verwendet werden kann. Hierbei wird nach dem Anlegen jedes Musters die Änderung der Gewichte neu berechnet.

E_p = Fehlerfunktion des Musters p
o_{pj} = Output von Neuron j bei Muster p
W_{ij} = Verbindungsgewicht von Neuron i zu Neuron j
t_{pj} = Lerneingabe des Musters p an Neuron j

<u>Herleitung der Delta-Regel</u>

$$\Delta W_{ij} = -lr \frac{\partial}{\partial W_{ij}} E = \sum_p -lr \frac{\partial}{\partial w_{ij}} E_p \qquad \text{(Gradientenabstiegverfahren)}$$

Da t_{pj} konstant ist folgt mit der Kettenregel:

$$\frac{\partial E_p}{\partial W_{ij}} = \frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial W_{ij}}$$

Es ist für die Modifikation der Gewichte unerheblich, ob man den Fehler oder den halben Fehler minimiert, deshalb wird mit dem halben Fehler weiter gerechnet:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2$$

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) = -\delta_{pj}$$

Mit der Voraussetzung, daß eine lineare Übertragungsfunktion verwendet wird ergibt sich:

$$o_{pj} = \sum_i o_{pi} W_{ij}$$

$$\frac{\partial o_{pj}}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_i o_{pi} W_{ij} = o_{pi}$$

Nach dem Einsetzen der Ergebnisse erhält man:

$$\frac{\partial E_p}{\partial W_{ij}} = o_{pi}(-\delta_{pj})$$

$$\boxed{\Delta W_{ij} = lr \sum_p o_{pi} \delta_{pj}}$$

Um die Gewichtsänderung nach jedem Trainingsmuster sofort zu berechnen, wird in vielen Neuronalen-Netze-Simulatoren die folgende Gewichtsmodifikation verwendet:

<u>Die Delta-Regel:</u>

$$\Delta_p W_{ij} = lr \cdot o_{pi} \cdot \delta_{pj} = lr \cdot o_{pi}(t_{pj} - o_{pj})$$

Diese Form der Delta-Regel wird auch in MATLAB® zur Verfügung gestellt. Hier wird diese, wie öfters in der Literatur, als Widrow-Hoff-Regel (1960) bezeichnet.

Wie bereits erwähnt, ist die Delta-Regel nur für zweischichtige Netzwerke definiert, d.h. für Neuronale Netze ohne Hidden Layer. Dieses Problem führte zur Verallgemeinerung der Delta-Regel (generalized delta rule), dem sogenannten Backpropagation-Algorithmus. Mit ihm hat man ein Lernverfahren entwickelt, welches beliebig viele Hidden Layer in einem Neuronalen Netzwerk zuläßt.

Quellen: [7], [22]

Die Backpropagation-Regel

In Backpropagation-Netzwerken wird der Fehler am Ausgang sukzessive über alle Layer zum Eingang hin zurückpropagiert. Die Topologie der Backpropagation-Netze ist auf Feed- Foreward-Netzstrukturen beschränkt. Bei einem vollständig trainierten Netzwerk sollten die Hidden-Layer eine interne Repräsentation der Musterassoziation erlernt haben. Diese komplexe Form des Konnektionismus kann in aller Regel mehr Merkmale aus Mustern extrahieren, sodaß dadurch die Erkennsicherheit gegenüber einschichtigen Netzwerken verbessert ist. Zusätzlich sind Backpropagation-Netzwerke in der Lage, nichtlineare Funktionen zu berechnen und abzubilden.

<u>Herleitung des Backpropagationalgorithmus</u>

net_{pj} = Nettoinput der Zelle j bei Muster p

f_{act} = Aktivierungsfunktion

δ_{pj} = Fehlersignal der Zelle j für Muster p

$$net_{pj} = \sum_i o_{pi} W_{ij}$$

$$o_j = f_{act}(net_{pj})$$

Mit der Definition des Fehlers aus dem Gradientenverfahren:

$$\Delta W_{ij} = -lr \frac{\partial}{\partial W_{ij}} E = \sum_p -lr \frac{\partial}{\partial w_{ij}} E_p$$

$$\mathrm{E_p} = \sum_{\mathrm{j}} (\mathrm{net_{pj}} - \mathrm{o_{pj}})^2$$

$$\mathrm{E_p} = \sum_{\mathrm{j}} (\mathrm{net_{pj}} - \mathrm{fact}(\mathrm{net_{pj}}))^2$$

Die Differentiation unter Anwendung der Kettenregel ergibt:

$$\frac{\partial E_p}{\partial W_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial W_{ij}}$$

$$\frac{\partial net_{pj}}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_i o_{pi} W_{ij} = o_{pi}$$

Definition des Fehlersignals:

$$\delta_{pj} = \left(-\frac{\partial E_p}{\partial net_{pj}} \right) \; ; \quad \delta_{pj} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}}$$

$$\frac{\partial o_{pj}}{\partial net_{pj}} = \frac{\partial}{\partial net_{pj}} \cdot f_{act}(net_{pj}) = f'(net_{pj})$$

Durch Einsetzen der bisherigen Ergebnisse erhält man folgende Gleichungen:

$$\frac{\partial E_p}{\partial W_{ij}} = -\delta_{pj} o_{pi}$$

$$\Delta W_{ij} = lr \cdot \sum_p o_{pi} \delta_{pj}$$

Für das Online-Verfahren ergibt sich demnach:

$$\Delta_p W_{ij} = lr \cdot o_{pi} \delta_{pj}$$

Um das Fehlersignal δpj zu berechnen, ist es notwendig, eine Unterscheidung der Units j vorzunehmen. Fehlersignal für den Fall, daß j eine Ausgabezelle ist:

$$-\left(\frac{\partial E_p}{\partial o_{pj}} \right) = (t_{pj} - o_{pj})$$

$$\delta_{pj} = f'_{act}(net_{pj}) \cdot (t_{pj} - o_{pj})$$

Wegen der Rückwärtspropagierung des Fehlers, muß sich der Fehler einer Hidden Unit j aus der Summe aller gewichteten Fehler der Nachfolgezellen k, multipliziert mit den Gewichten der Verbindung von Zelle j zu Zelle k zusammensetzen.

Rückwärtspropagiertes Fehlersignal:

$$\Delta_{pk} = \sum_k \frac{\partial E_p}{\partial net_{pk}}$$

Fehler der Zelle j („Rückwärts“ betrachtet):

$$\Delta_{pj} = \frac{\partial net_{pk}}{\partial o_{pj}}$$

Die Zusammenfassung der Gleichungen ergibt das Fehlersignal für den Fall, daß j eine Hidden Unit ist:

$$-\frac{\partial E_p}{\partial o_{pj}} = -\sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_k (\delta_{pk} \frac{\partial}{\partial o_{pj}} \sum_i o_{pi} W_{ik}) = \sum_k \delta_{pk} W_{jk}$$

$$\delta_{pj} = f'_{act}(net_{pj}) \cdot \sum_k \delta_{pk} W_{jk}$$

<u>Die Backpropagation-Regel</u>

$$\boxed{\Delta_p W_{ij} = lr \cdot o_{pi} \delta_{pj}}$$

$$\delta_{pj} = \begin{cases} f'_{act}(net_{pj}) \cdot (t_{pj} - o_{pj}) & falls \quad j \quad Ausgabezelle \quad ist \\ f'_{act}(net_{pj}) \cdot \sum_k \delta_{pk} W_{jk} & falls \quad j \quad versteckte \quad Zelle \quad ist \end{cases}$$

In dem verwendeten Neuronale-Netze-Simulator von MATLAB® wird keine Aktivierungsfunktion angeboten (sie wird hier durch die Idendität-Funktion ersetzt). Vielmehr wird die Propagierungsregel ($net_j(t) = \Sigma o_i(t) w_{ij}$) mit einer Übertragungsfunktion zusammengefaßt. Bei der Anwendung ergeben sich dadurch aber keine Nachteile. Grundsätzlich ist jedoch darauf zu achten, daß die verwendeten Übertragungsfunktionen monoton und differenzierbar sind, wie aus der Modifikation der Gewichte mit der Backpropagation-Regel ersichtlich ist.

Es sei noch darauf hingewiesen, daß sich diese Herleitungen, sowie die des im folgenden beschriebenen Momentum-Termes, an den Ausführungen in [7] orientiert, welche sich auf [Rummelhart, Mc Clelland. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. The MIT Press, 1986] begründen.

Quellen: [7], [11], [22]

Der Momentum-Term

Bei der Suche nach dem Minimum in einer Fehlerfunktion besteht immer die Gefahr, nicht das globale Minimum, sondern ein lokales Minimum zu finden. Gleichermaßen sind Oszillationen zu verhindern, innerhalb derer einige Muster andere bereits trainierte Muster zum Teil zerstören. Diese Fehlfunktionen können das Training eines Backpropagation-Netzwerkes erheblich stören oder sogar völlig blockieren. Durch das Einfügen eines Terms in den Lernalgorithmus, welcher die Änderung der Gewichte von der vorangegangenen Änderung abhängig macht, können diese Fehlfunktionen verhindert werden. Backpropagation mit Momentum-Term wird auch als konjugierter Gradientenabstieg (conjungate gradient descent) bezeichnet. Dieses Verfahren wurde von Rummelhart, Hinton und Williams entwickelt.

<u>Momentum-Term:</u>

$$\mu \Delta_p W_{ij}(t-1)$$

<u>Backpropagation mit Momentum-Term:</u>

$$\Delta_p W_{ij}(t) = lr \cdot o_{pi} \delta_{pj} + \mu \Delta_p W_{ij}(t-1)$$

In MATLAB® wird Backpropagation mit Momentum-Term ebenfalls mit diesem Algorithmus unterstützt. Der Momentumfaktor μ hat üblicherweise Werte von 0,6 bis 0,99. In MATLAB® wird $\mu = 0{,}9$ als Voreinstellung angegeben. Von Sejnowski und Rosenberg gibt es eine weitere Variante des Momentum-Terms. Diese wird hier nicht vorgestellt, da sie bei der Erstellung der Diplomarbeit keine Anwendung fand.
Quellen: [7], [11], [22]

Adaptive Lernrate

Ein effizientes Mittel zur Verkürzung der benötigten Zeit für den Lernvorgang ist die Anpassung der Lernrate lr nach jeder Trainingsepoche. Innerhalb einer Epoche wird dem Neuronalen Netz ein kompletter Trainingsmustersatz vorgegeben.
Die Realisierung der adaptiven Lernrate in MATLAB® basiert auf einfachen IF-THEN-Entscheidungen während der Lernphase. Dabei müssen folgende Parameter voreingestellt werden:
err_ratio = Maximale Fehlerrate (Default = 1,04)
lr_inc = Faktor für die Vergrößerung der Lernrate (Default = 1,05)
lr_dec = Faktor für die Verringerung der Lernrate (Default = 0,7)
Abweichend von anderen Verfahren zur Adaption der Lernrate wird hier nicht mit einer sehr großen Lernrate begonnen. Solche Verfahren verringern die Lernrate solange schrittweise, bis der Lernvorgang stabil wird. Üblicherweise initialisiert man die Lernrate in MATLAB® mit Werten von 0,01 bis 0,8. Dadurch versucht man anfängliche Oszillationen und somit verlängerte Trainingszeiten zu vermeiden.
<u>Vorgehensweise bei der Adaption der Lernrate</u>
Nach jeder Epoche wird die aktuelle Quadratsumme des Fehlers am Ausgang des Netzwerkes neu berechnet. Diese wird dann mit der err_ratio verglichen. Übersteigt die Fehlerrate den maximalen Wert, wird die Lernrate um den Faktor lr_dec verringert. Unterschreitet die Fehlerrate den vorgegebenen Schwellenwert, wird die Lernrate um den Faktor lr_inc vergrößert. Bei der Anpassung der Adaptionsfaktoren (lr_dec, lr_inc, err_ratio) sollten die Defaultwerte nur in kleinen Schritten verändert werden, da sehr schnell Oszillationen im Lernvorgang auftreten können. Im „worst case“ kann der Fehler des Netzwerkes divergieren, d.h. das Training bricht nach der vorgegeben maximalen Anzahl von Epochen, ohne einen Lerneffekt erzielt zu haben, ab.
Quellen: [7], [22]

Applikation EKG-Erkennung

Die Anwendung von Pattern-Recognition-Algorithmen auf Elektrokardiogramme (EKG) eignet sich besonders gut. Zum Einen sind die Muster der EKG schon bis ins Detail klassifiziert und zum anderen sind EKG-Muster periodische Signale. Wegen der großen Amplitude stellt die Aufnahme von EKG-Signale auch im Laborversuch keine Schwierigkeit dar. Um die Leistungsfähigkeit des Neuronale-Netze-Simulator in MATLAB® zur Mustererkennung und Musterklassifikation an biologischen Signalen zu untersuchen, wurden deshalb EKG-Signale verwendet.

Elektrokardiogramm

In der systolischen Phase (Erregungsphase) einer Muskelzelle tritt eine elektrische Spannung auf, das Aktionspotential. Das Aktionspotential des Herzmuskels entsteht beim Austausch von Natrium-Ionen und Kalium-Ionen. Das außerhalb der Zellen liegende Natrium tritt dabei in die Zellen ein, während das innerhalb der Zellen befindliche Kalium austritt. Bei der folgenden diastolischen Phase (Entspannungsphase) erfolgt die Rückwanderung der Natrium-Ionen und der Kalium-Ionen. Die durch den Konzentrationsunterschied der beiden Elemente bedingte elektrische Spannung ist das sogenannte Aktionspotential des Myokard (Herzmuskel). Die chemischen Vorgänge hierbei sind mit denen eines Primärelementes vergleichbar.

<u>Grundform des EKG</u>

Die Grundform des Elektrokardiogramms ist von dem gesetzmäßigen Erregungsablauf im Herzen und von den jeweils gewählten Ableitformen abhängig. Die Ableitformen geben die Anordnung der Elektroden am Körper an. Den charakteristischen Spannungsverlauf des EKG kann man am besten für die Ableitung zwischen Sternum (Brustbein) und Apex (Herzspitze) erläutern. Diese Grundform (s.u.) gilt für den Normaltyp mit $30° < \alpha < 60°$, wobei α der Winkel der Herzachse mit der Horizontalen beschreibt. Bei Steiltypen ($60° < \alpha < 90°$), Rechtstypen ($90° < \alpha < 120°$), Horizontaltypen ($0° < \alpha < 30°$) oder Linkstypen ($-30° < \alpha < 0°$) erhält man einen entsprechend anderen Spannungsverlauf des EKG.

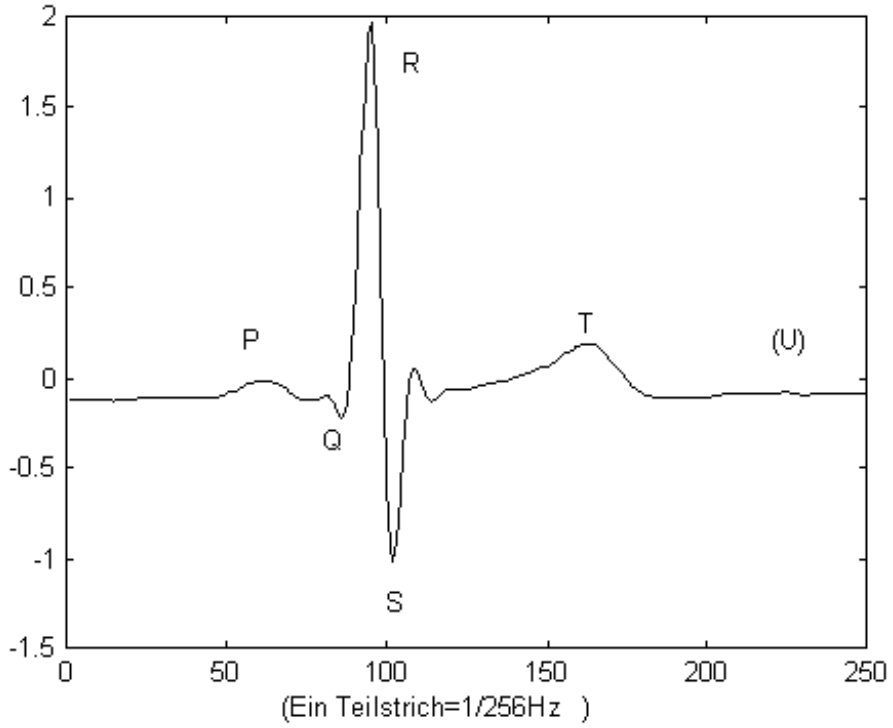


Abb. 1.26: Normalform des EKG bei bipolarer Ableitung

Bedeutung der einzelnen EKG-Abschnitte

Die Erregungsausbreitung in den Vorhöfen (Atrium) wird durch die P-Welle gekennzeichnet. Im Bereich der PQ-Strecke ist die gesamte Vorhofmuskulatur erregt. Dadurch liegen keine Potentialdifferenzen innerhalb des Vorhofmyokards vor. Die PQ-Strecke ist ein Maß für die Dauer vom Beginn der Vorhoferregung bis zum Beginn der Kammererregung. Dies ist die sogenannte Überleitungszeit. Der QRS-Komplex entsteht durch die Erregungsausbreitung in den beiden Ventrikeln. Bevor sich die Erregung in den Ventrikeln zurückbildet, was durch die T-Welle gekennzeichnet ist, zeigt der Nullinienverlauf der ST-Strecke die gleichmäßige Erregung des Ventrikelmyokards an. Die U-Welle tritt nur in seltenen Fällen auf und ist noch nicht eindeutig erklärbar.

Es ist auch möglich, eine vektorielle Deutung des EKG-Verlaufs vorzunehmen. Dabei werden die Potentialunterschiede zwischen erregten und noch nicht erregten Muskelfasern als Vektoren dargestellt. Addiert man zu einem bestimmten Zeitpunkt die Vektoren auf, so erhält man einen Integralvektor. Dieser charakterisiert die vom Herzen ausgehende Spannung, welche sich mit dem Erregungsablauf verändert.

Quellen: [5], [16], [18], [21]

EKG-Datenakquisition- Vorverarbeitung

Die EKG-Daten wurden am Institut für Digitale Signalverarbeitung an der FHT-Mannheim aufgenommen. Für die Aufnahme der Spannungsverläufe sind die Probanden über handelsübliche Klebeelektroden mit dem Meßverstärker verbunden worden. Die Klebeelektroden haben einen mit Gel gefüllten Kern, welches einen optimalen Übergangswiderstand sicherstellt. Zur Ableitung des EKG-Signals sind die Elektroden am Sternum (Brustbein) und oberhalb des Apex (Herzspitze) aufgeklebt worden, was zur Aufnahme der Normalform des EKG führte. Dieses Signal wurde zunächst über einen Vorverstärker mit Filterfunktionen der Firma Stanford auf 10kHz bandbegrenzt. Zusätzlich konnte mit diesem Vorverstärker eine Impendanzanpassung an den Endverstärker vorgenommen werden. Als Endverstärker kam der an der FHT-Mannheim (am Institut für Analogtechnik) für die Aufnahme von Elektromyogrammen entwickelte Verstärker zum Einsatz.

Die Herzfrequenz des Menschen schwankt um 2,5 Hz, sodaß mit der gewählten Abtastfrequenz von 250 Hz eine Unterabtastung ausgeschlossen ist. Die anschließende Analog- Digitalwandlung wurde mittels einer kommerziellen PC-AD-Einsteckkarte mit 16 Bit Auflösung durchgeführt. Infolge der Umgebungsbedingungen im Labor und des Versuchsaufbaus waren die Signale von einem 50Hz Netzrauschen überlagert. Deshalb war es notwendig, die EKG-Signale zunächst mit einer schmalbandigen 50Hz Bandsperre zu filtern. Diese Filterung wurde mit dem oben beschriebenen frequenzselektiven Softwarefilter durchgeführt.

Auswahl und Training des Neuronalen Netzes zur EKG-Erkennung

Nach einigen Voruntersuchungen ist die Auswahl auf ein dreistufiges Neuronales Netz gefallen. Dadurch wurde sichergestellt, daß zur Mustererkennung genügend Merkmale aus dem EKG-Muster extrahiert werden können. Infolge der Netzwerktopologie wird die Backpropagation-Lernregel angewendet, welche es gleichzeitig ermöglicht, linear nicht trennbare Muster voneinander zu unterscheiden.

Für das Training des Neuronalen Netzes werden verschiedene den Meßdaten entnommene Muster zugrunde gelegt. Dabei muß darauf geachtet werden, daß dieser Musterdatensatz aus als richtig zu erkennende wie auch aus als falsch zu klassifizierenden Mustern besteht. Die Auswahl der Trainingsmuster erfolgt üblicherweise empirisch und entscheidet maßgeblich über den Erfolg oder Mißerfolg des Lernvorganges.

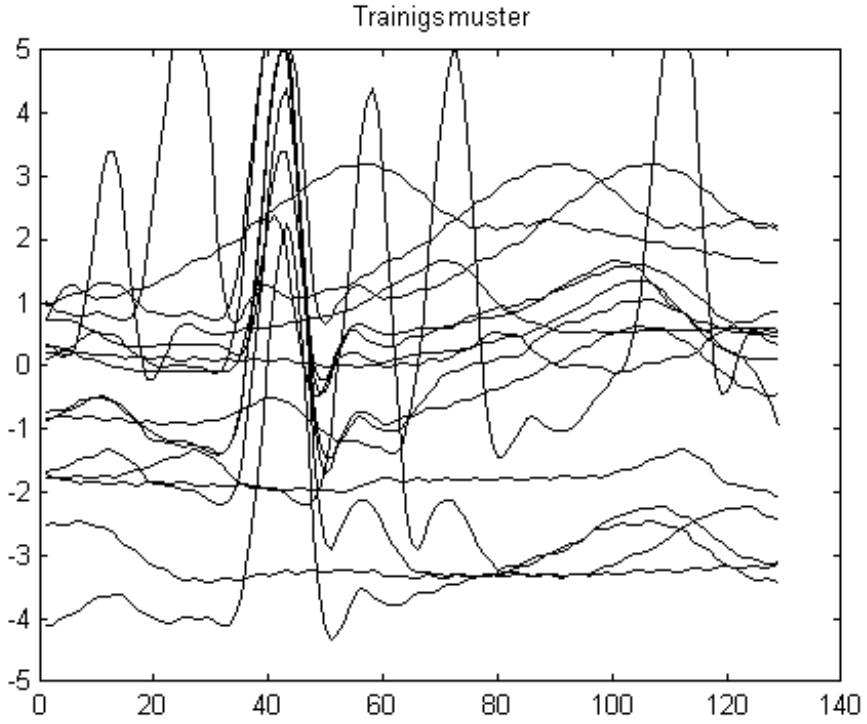


Abb. 1.27: Verwendete Trainingsmuster
Auf der X-Achse sind die Abtastwerte im Abstand von 4ms aufgetragen. Die Y-Achse gibt die Spannungswerte am Ausgang des Endverstärkers in Volt an.
Wie in MATLAB® üblich, werden alle Muster gleichzeitig als Matrix an das Netzwerk angelegt. Gleichermaßen muß entsprechend der Trainingsmuster das zu assoziierende Ausgangsmuster in Form eines Vektors bzw. einer Matrix vorgegeben werden. Da bei dieser EKG-Klassifizierung auf die Erkennung von ventrikulären Extrasystolen, Herzkammer-flattern, Herzkammerflimmern und anderen Herzrythmusstörungen verzichtet wird, kann die Soll-Ausgabe des Neuronalen Netzes als Vektor dargestellt werden. Die Einsen geben die Soll-Ausgabe des Netzes bei erkannten EKG-Signalen an. Bei allen anderen Mustern soll das Netzwerk mit der Ausgabe einer Null antworten.

Ausgabevektor:
targets=[0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
Mit dem Ausgabevektor bzw. der Ausgabematrix, werden beim Neuronalen Netzwerkdesign in MATLAB® gleichzeitig die Anzahl der Neuronen in der letzten Layer-Schicht vorgegeben. In diesem Fall besteht somit der Ausgabe-Layer aus einem Neuron, da die Ausgabematrix zwar aus mehreren Spalten aber nur aus einer Reihe besteht. Daraus resultiert die unten dargestellte Netzwerktopologie.

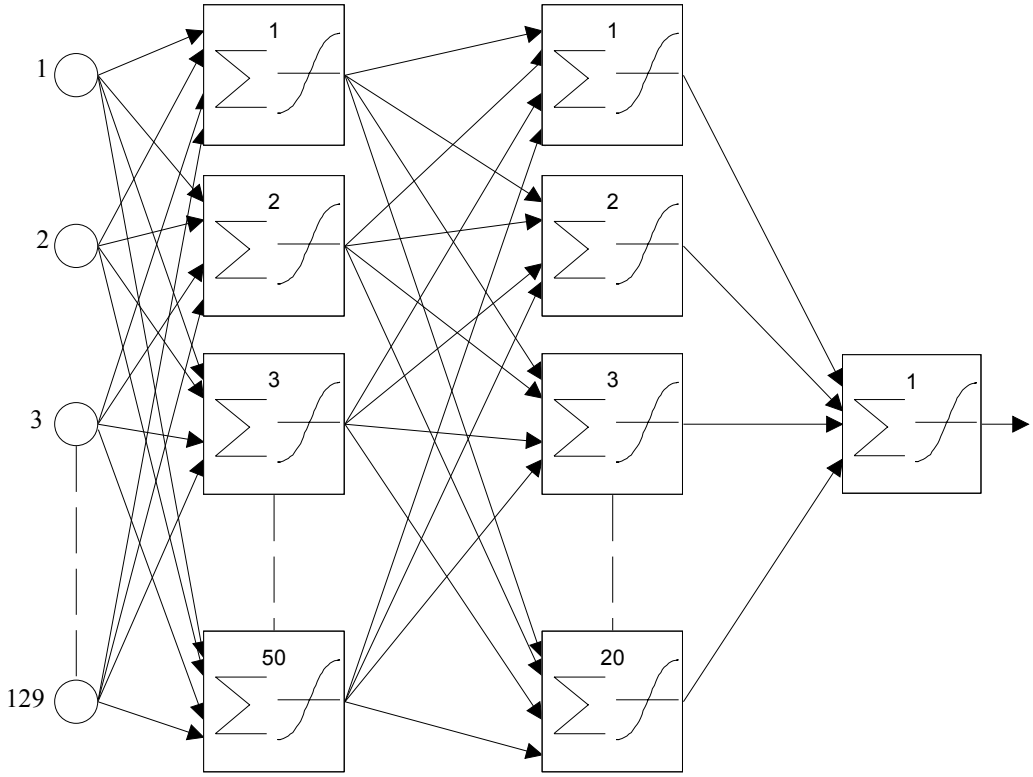


Abb. 1.28: Feed-Foreward-Netzwerk
Der Eingangsunit-Layer hat eine Länge von 129 Sensoren. Mit ihm wird die Länge der angelegten Muster bestimmt. Bei der EKG-Erkennung genügen die 129 Abtastwerte auch für einen ganzen PQRS-Komplex mit niedriger Frequenz. Das Design der ersten und zweiten Hidden-Layer-Schicht ist dem Anwender frei überlassen. Er muß darauf achten, daß die Tiefe der Hidden-Layer groß genug ist, um alle Trainingsmuster zu klassifizieren.

Andererseits wird üblicherweise zur Reduzierung der Trainingszeit und der Bearbeitungszeit zur Klassifizierung der Muster versucht, die Anzahl der Neuronen in den Hidden-Layern soweit wie möglich zu reduzieren. Vor allem bei Systemen, die in Echtzeit arbeiten sollen, wird hierfür sehr viel Aufwand betrieben. In dieser Applikation konnte auf die zeitaufwendige Minimierung des Neuronalen Netzes verzichtet werden, da die Bearbeitungszeit für die EKG-Erkennung infolge der gespeicherten Daten keine maßgebliche Rolle spielt.
Das Training des Neuronalen Netzes wurde mit der Funktion trainbpx der Neural-Network-Toolbox in MATLAB® durchgeführt. Diese Funktion beinhaltet gleichzeitig den Momentum-Term sowie die Adaption der Lernrate (s.o.). Die folgende Darstellung zeigt den Verlauf der Summe des quadratischen Fehlers (SSE) während des Netzwerktrainings sowie den entsprechenden Wert der Lernrate. Mit der unterbrochenen Linie wird die zu erreichende Summe des quadratischen Fehlers gekennzeichnet.

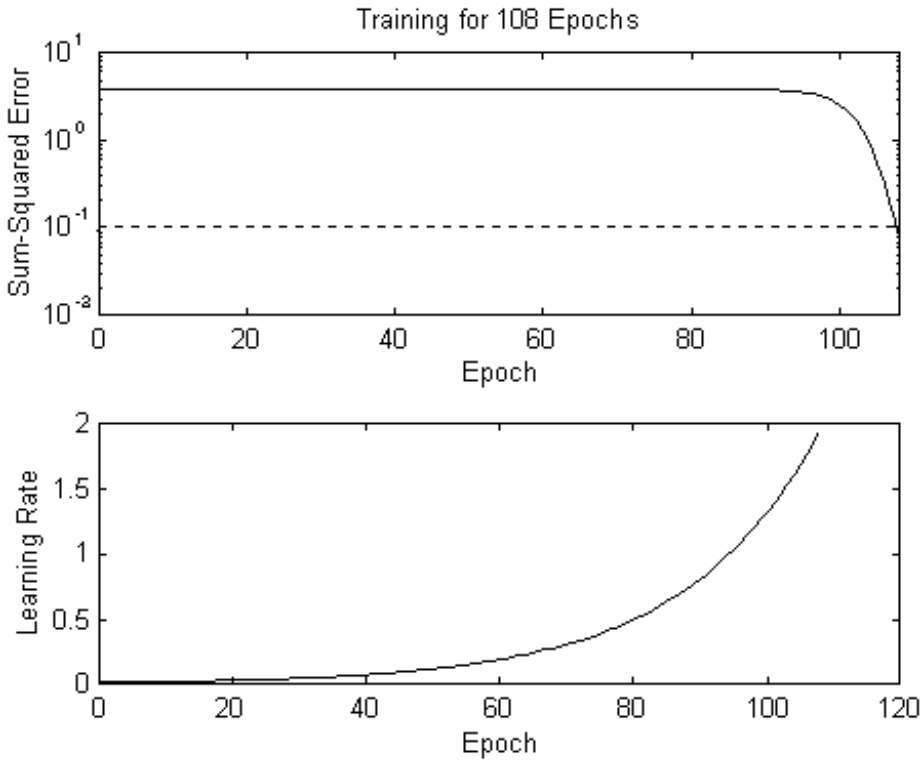


Abb. 1.29: Trainingsverlauf bei der EKG-Erkennung

Der Fehler des Netzwerkes wurde nach 120 Trainingsepochen auf das vorgegebene Maß reduziert. Wie man sieht, bleibt der Lernerfolg bei kleiner Lernrate sehr gering und wächst dann sehr schnell mit derselben an. In diesem Fall steigerte somit die Adaption der Lernrate die Effizienz des Trainings erheblich.
Bei einschichtigen Neuronalen Netzen mit nur einem Neuron kann der Lernerfolg auch durch die Kontrolle des Gewichtsvektors des Neurons überprüft werden. Der Gewichtsvektor sollte dann ein Abbild des zu erkennenden Musters darstellen. Im Falle der EKG-Erkennung mit einem mehrschichtigen Netz können hierfür die Verbindungsgewichte zwischen dem ersten und dem zweiten Hidden-Layer herangezogen werden.

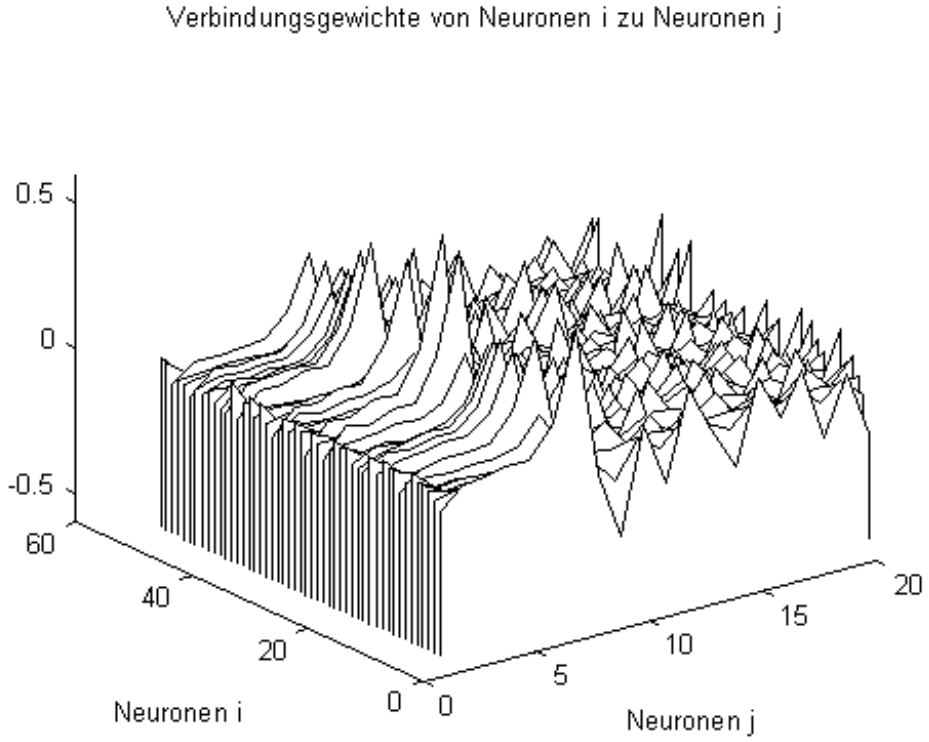


Abb. 1.30: Verbindungsgewichte vom ersten zum zweiten Hidden-Layer

Die Darstellung zeigt alle Verbindungsgewichte von den 50 Neuronen des ersten Hidden-Layer zu den 20 Neuronen des zweiten Hidden-Layer. Es ist grob schematisch die Abbildung eines PQRS-Komplexes zu erkennen.
Da die Gewichte mit den Sensorwerten multipliziert werden, erfolgt eine nur geringe Verstärkung im Bereich der P-Welle, aber eine stärkere Anhebung der Signale im QRS-Bereich. Diese Vorgehensweise hat sehr viel Ähnlichkeit mit der Kreuzkorrelationsanalyse (KKF), weshalb im folgenden noch ein Vergleich zwischen der KKF und Neuronalen Netzen zur Mustererkennung erfolgt.

Auswertung der EKG-Erkennung mit einem Neuronalen Netz

Für das Training des Netzes werden die aufgenommenen EKG-Signale der Probanden ARM, RIND und CARL verwendet. Anschließend folgt ein Testdurchlauf, bei dem das Neuronale Netz die gelernten Muster erkennen soll, um den Lernvorgang zu überprüfen. Hierbei werden erst die ersten 129 Bytes des zu untersuchenden Signals vorgegeben, dann das zweite bis 130. Byte u.s.w. Dieser Vorgang wird solange wiederholt, bis das ganze EKG-Signal „durchgeshiftet“ wurde.
Für die Mustererkennung ist die Skalierung der Achsen der Signaldiagramme ohne Bedeutung, weshalb bei den folgenden Darstellungen darauf verzichtet wird. Es muß nur darauf geachtet werden, daß die Trainingsmuster und die zu erkennenden Datenmuster in ihrer Laufzeit übereinstimmen. Dies wird durch eine fest eingestellte Abtastfrequenz bei der Datenakquisition gewährleistet.
Grundsätzlich kann man davon ausgehen, daß die EKG-Signale von verschiedenen Probanden sich in Frequenz, Amplitude und Form unterscheiden. Deshalb wird für die Ermittlung der Generalisierungsleistung des Neuronalen Netzes das EKG-Signal des vierten Probanden ALEX herangezogen. In einem letzen Testdurchlauf werden nachträglich verrauschte EKG-Signale verwendet, die das Netz erkennen soll. Hier sei

noch angemerkt, daß für die Auswertung eines 1000 Byte langen Datensatzes ca. 8,5s benötigt werden. Dieser Wert wurde mit dem Neuronale-Netze-Simulator von MATLAB® auf einem PC mit dem INTEL® 80486 Prozessor und 8 MByte RAM ermittelt.

Testdurchlauf mit einem der Trainingsmuster

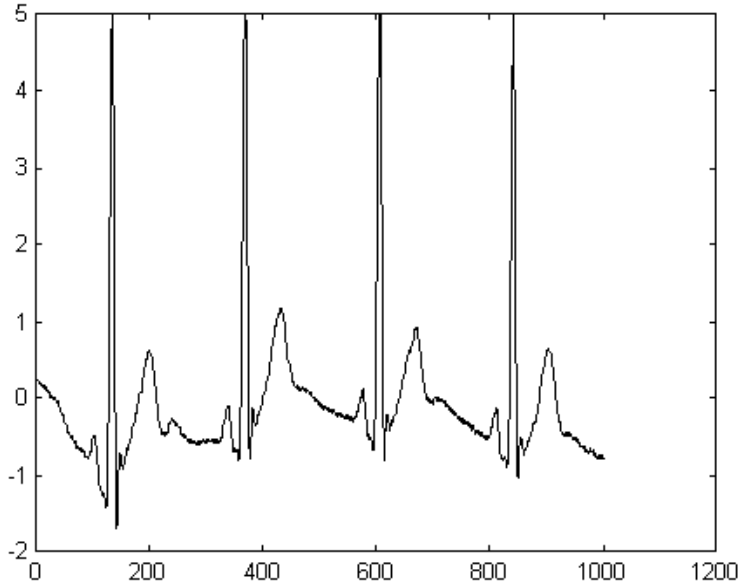


Abb. 1.31: Ausschnitt aus dem EKG des Probanden ARM:

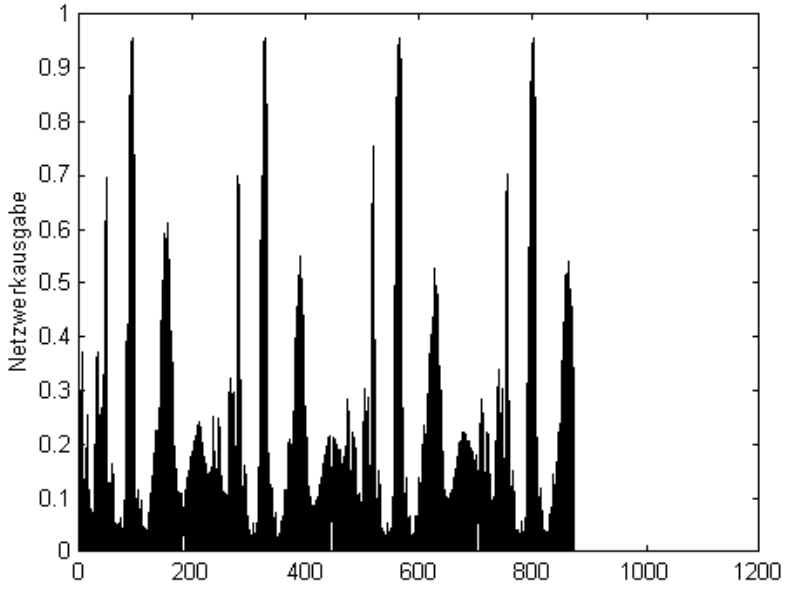


Abb. 1.32: Ausgabe des Neuronalen Netzes:
Setzt man einen Schwellwert von 0,8 an, ergeben sich genau die Stellen, an denen sich die EKG-Komplexe befinden. Somit können alle EKG-Komplexe genau lokalisiert werden.

Testdurchlauf mit einem nichttrainierten Mustersatz

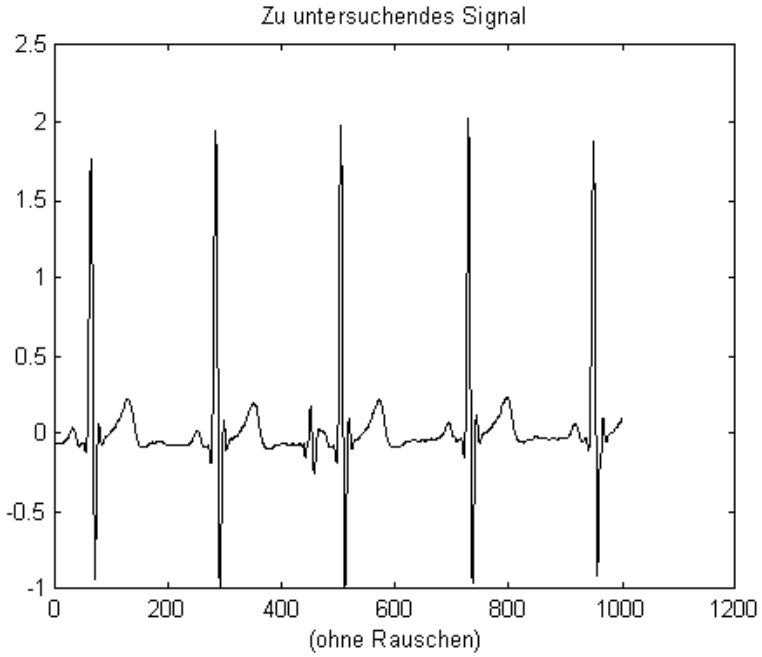


Abb. 1.33: Ausschnitt aus dem EKG des Probanden ALEX

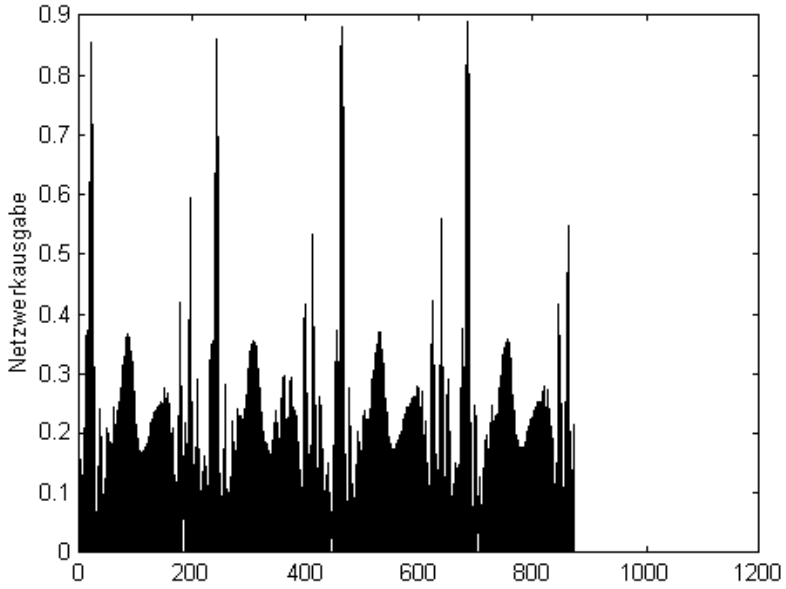


Abb. 1.33: Reaktion des Neuronalen Netzes

Auch hier erhält man mit einem Scwellwert von 0,8 die lokalen Plazierungen der EKG-Komplexe. Weiterhin kann man deutlich erkennen, daß der fünfte PQRS-Komplex nicht erkannt wurde, da die T-Welle vollständig abgeschnitten ist.

Testdurchlauf mit einem nichttrainierten und verrauschten EKG-Signal

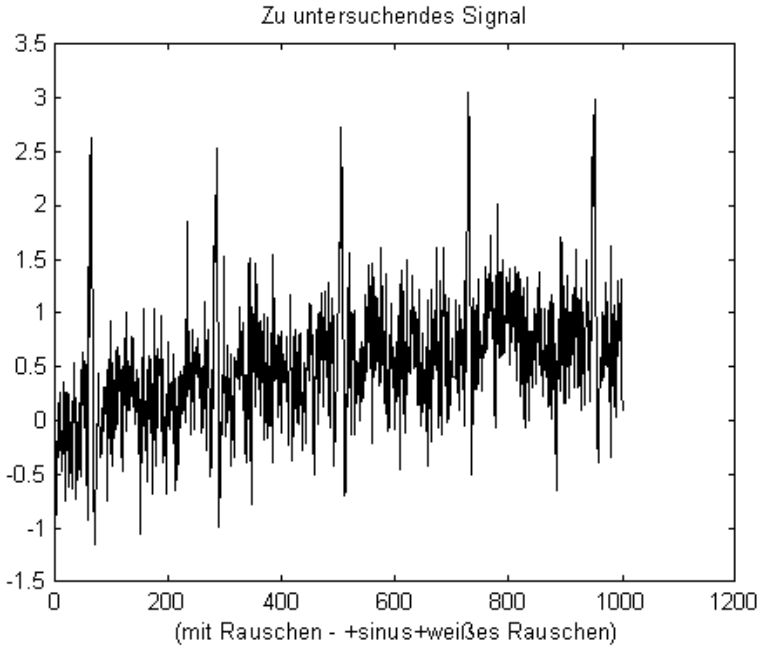


Abb. 1.34: Ausschnitt aus dem EKG des Probanden ALEX (nachträglich verrauscht)

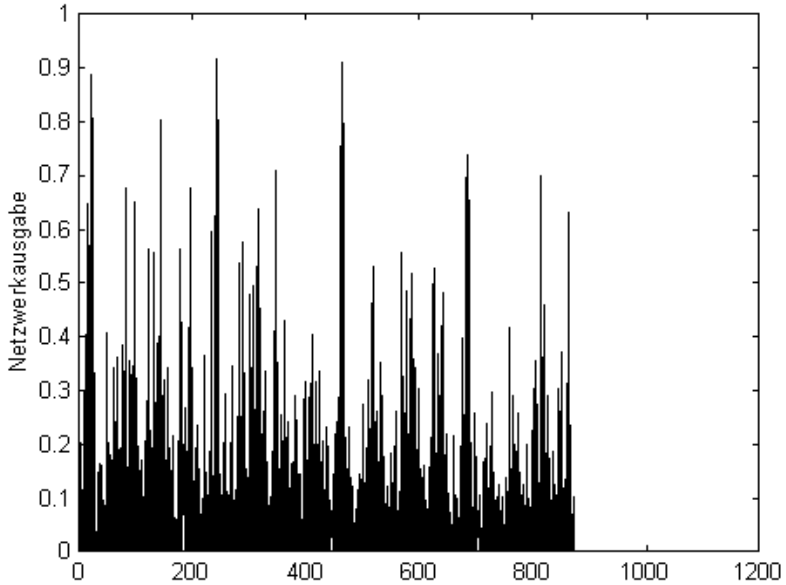


Abb. 1.35: Reaktion am Ausgang des Neuronalen Netzes
Selbst nach der Addition weißen Rauschens (Amplitude 40% der maximalen Signalamplitude) erhält man noch mit dem Schwellwert 0,8 die örtliche Lage von drei EKG-Komplexen.

Dieses Beispiel veranschaulicht die Leistungsfähigkeit Neuronaler Netze für die Anwendung bei Mustererkennungsaufgaben. Ein weiterer interessanter Aspekt ist die Erweiterung des Umfangs der zu erkennenden Muster. Diese Aufgabe könnte einfach durch Hinzufügen eines weiteren Neurons im Output-Layer bewältigt werden. Dadurch wäre es möglich, auch Arrhythmien eindeutig zu klassifizieren. Da für die hierfür benötigten Trainingsmuster ein entsprechender EKG-Simulator benötigt werden würde, sind im Rahmen dieser Diplomarbeit hierzu keine weitergehenden Untersuchungen angestellt worden. Sollte die EKG-Erkennung unter Anwendung Neuronaler Netze weitergeführt werden, empfiehlt der Autor die Verwendung einer Workstation oder den Einsatz sogenannter „Neurocomputer"-Einsteckkarten für den PC. Einsteckkarten, wie z.B. die N6400 (Adaptive Solution Inc.), bewältigen ca. 1,6 G CPS (Connections per Second) und ca. 256 M CUPS (Connection updates per Second). Im Vergleich hierzu arbeitet ein INTEL 80386 Prozessor nur mit ca. 150k CPS und ca. 50k CUPS. Durch die um den Faktor 10^6 fache Performance würde eine Echtzeitverarbeitung der EKG-Signale ohne weiteres ermöglicht werden.
Die Ermittlung der Pulsfrequenz aus EKG-Signalen stellt keine einfache Aufgabe dar und muß für die intensivmedizinische Überwachung von Patienten automatisch und sicher gelöst werden. Aus diesem Grund hat man sich auch am Institut für Mikroelektronik der Universität Stuttgart (IMS) mit der Lösung dieses Problems beschäftigt. Auch hier wurde zur Problemlösung ein Neuronales Netzwerk realisiert. Die Software bildete ein am IMS entwickelter Neuronale-Netze-Simulator (NNSIM), sowie ein für die Generierung der Trainingsmusterdatensätze entwickelter EKG-Simulator. Die Verarbeitung der EKG-Signale in Echtzeit ist durch den Einsatz einer Workstation und durch die Optimierung (Verringerung der Anzahl der Neuronen des Neuronalen Netzes) ermöglicht worden.

Quellen: [10]

Korrelationsanalyse

Korrelationskoeffizient

Mustererkennung (Pattern Recognition) beruht, wie beim menschlichen Vorgehen, im allgemeinen auf Vergleichsoperationen. Bei der Untersuchung von Zufallsprozessen, wie z.B. Rauschen, verwendet man verschiedene Korrelationstechniken. Diese sind Hilfsmittel um unter Anwendung dieser Vergleichsoperationen irgendeine statistische Regelmäßigkeit oder Korrelation in gleichen oder verschiedenen Datensätzen zu finden. Für die Analyse experimenteller Daten kann die Darstellung in einem Diagramm helfen, eine gegenseitige Abhängigkeit oder eine Korrelation innerhalb der Daten zu finden. Beispielsweise möge x die Übertragungsgeschwindigkeit eines Datenkanals und y den entsprechenden Übertragungsfehler darstellen. Man könnte sich nun vorstellen, daß in einem bestimmten Bereich der Übertragungsfehler mit der Übertragungsgeschwindigkeit anwächst. Um nun den Grad des Zusammenhangs zwischen solchen Größen zu messen, benutzt man den Korrelationskoeffizienten ρ.

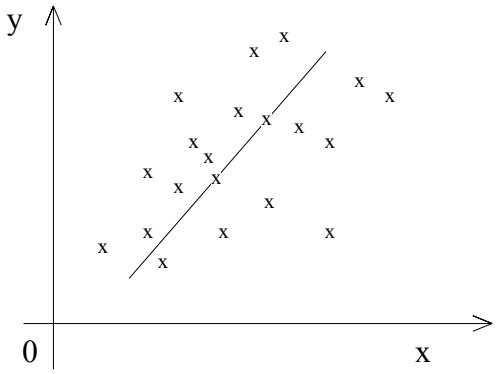


Abb. 2.1: Wertepaare x und y und Ausgleichsgerade

Durch die Berechnung des Korrelationskoeffizienten erhält man ein Maß für die Nähe der Wertepaare x,y an der Geraden y = mx + b .

In einem ersten Schritt zur Berechnung des Korrelationskoeffizienten muß der arithmetische Mittelwert der Meßreihen gebildet werden. Dieser wird für die Bestimmung der reinen Wechselanteile (Meßwertschwankungen) verwendet.

Arithmetischer Mittelwert der Meßwerte:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \qquad \overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

Berechnung der Wechselanteile:

$$x_i = X_i - \overline{X} \qquad y_i = Y_i - \overline{Y}$$

Definition des Momentenmittelwertes (nicht normierter Korrelationsfaktor):

$$\boxed{\mathrm{P} = \frac{1}{n}\sum_{i=1}^{n} x_i \cdot y_i} = \overline{xy}$$

Damit sich die Werte des Korrelationsfaktors im Intervall [-1<P<1] bewegen, wird für die Normierung folgender Ausdruck verwendet:

$$normf = \sqrt{\overline{x^2} \cdot \overline{y^2}}$$

Dieser Ausdruck entspricht genau dem Produkt der beiden Standardabweichungen σ, welche sich einfach aus der Quadratwurzel der empirischen Varianzen bestimmen lassen.

Empirische Varianz:

$$\sigma_x{}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 \qquad \sigma_y{}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

$$normf = \sigma_x \sigma_y$$

Somit ergibt sich insgesamt für den Korrelationsfaktor:

$$\boxed{\rho = \frac{\frac{1}{n}\sum_{i=1}^{n} x_i y_i}{\sqrt{\overline{x^2} \cdot \overline{y^2}}} = \frac{\overline{xy}}{\sigma_x \sigma_y}}$$

Beträgt der Korrelationsfaktor ρ = -1, so besteht vollständige Abhängigkeit oder Korrelation. Das gleiche gilt für den Fall, daß ρ = 1 ist. Bei ρ = 0 ist keine Korrelation vorhanden, d.h. die Punkte sind auf der ganzen Ebene verstreut. In diesem Fall spricht man auch von orthogonalen Variablen bzw. Meßreihen.

Quellen: [1], [4], [6]

Korrelationsfunktion

Der oben beschriebene Korrelationsfaktor kann nur für diskrete Meßreihen angewendet werden. Um Abhängigkeiten innerhalb kontinuierlicher Signale beschreiben zu können, wird dieser Korrelationsfaktor leicht modifiziert.

Korrelationsfaktor für kontinuierliche Signale (nicht normiert):

$$\mathrm{P} = \lim_{T \to \infty} \left(\frac{1}{2T} \int_{-T}^{+T} x(t) y(t) dt \right)$$

In der praktischen Anwendung wird die Korrelationszeit T selbstverständlich begrenzt. Der Normierungsfaktor beim Korrelationsfaktor für kontinuierliche Signale berechnet sich aus dem Produkt der Effektivwerte beider Signale (quadratischer Mittelwert).

Effektivwerte:

$$X = \sqrt{\frac{1}{2T} \int_{-T}^{+T} x^2(t) dt}$$

$$Y = \sqrt{\frac{1}{2T} \int_{-T}^{+T} y^2(t) dt}$$

Normierter Korrelationsfaktor für kontinuierliche Signale:

$$\rho = \frac{\lim_{T \to \infty} \left(\frac{1}{2T} \int_{-T}^{+T} x(t) y(t) dt \right)}{X \cdot Y}$$

Erweitert man den Korrelationsfaktor um die Verzögerungsvariable τ, so kann man die mögliche Korrelation zweier verschiedener Signale x(t) und y(t) durch die Kreuzkorrelationsfunktion $R_{xy}(\tau)$ darstellen.

Kreuzkorrelationsfunktion:

$$\boxed{\rho(\tau) = \lim_{T\to\infty}\left(\frac{1}{2T}\int_{-T}^{+T} x(t)y(t+\tau)dt\right)}$$

Bei der Kreuzkorrelationsfunktion werden Abhängigkeiten zweier verschiedener Signale untersucht. Gleichermaßen ist eine Funktion definiert, welche die Korrelation eines Signals mit sich selbst bestimmt. Die sogenannte Autokorrelationsfunktion ist unter anderem ein sehr gutes Hilfsmittel zur Analyse stark verrauschter Signale.

Autokorrelationsfunktion:

$$\boxed{\mathrm{A}(\tau) = \lim_{T\to\infty}\left(\frac{1}{2T}\int_{-T}^{+T} x(t)x(t+\tau)dt\right)}$$

Quellen: [1], [4], [6]

Durchführung einer linearen Korrelation

Zur bildlichen Veranschaulichung der Vorgehensweise bei der Durchführung einer Korrelationsanalyse wird ein einfaches Beispiel herangezogen, das sich in drei Schritte gliedert. Im ersten Schritt substituiert man die Laufvariable t durch die Verzögerungsvariable τ. Dann verschiebt man eine der beiden Funktionen $x(\tau)$ oder $y(\tau)$ nach links. Im Gegensatz zur Faltung wird bei der Kreuzkorrelationsanalyse keine Spiegelung einer Funktion vorgenommen. Im letzten Schritt variiert man t und schiebt somit eine Funktion über die andere. Die Integration erfolgt dann über den gesamten Überlappungsbereich der beiden Funktionen.

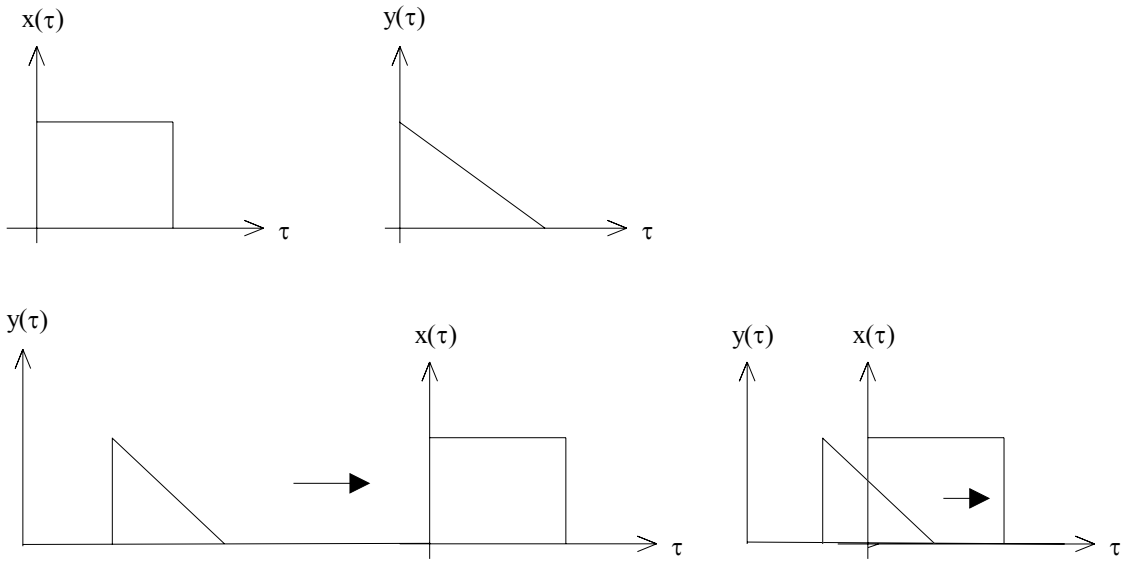


Abb. 2.2: Durchführung einer linearen Kreuzkorrelation

Die Berechnung der Kreuzkorrelationsfunktion mit dem aufgeführten Beispiel ergibt mit MATLAB® folgenden Verlauf.

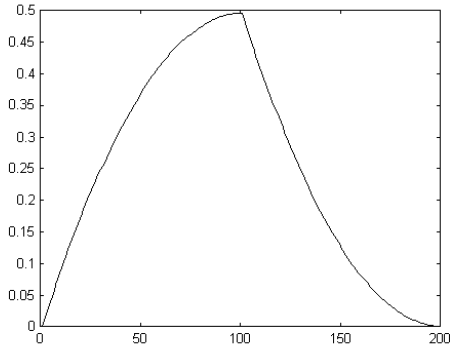


Abb. 2.3: Verlauf der Kreuzkorrelationsfunktion

Das Maximum der Kreuzkorrelation befindet sich an der Stelle maximaler Überlappung der beiden Funktionen x(t) und y(t). Für den Einsatz der Korrelationsfunktion zur Mustererkennung, wie z.B. bei der Pulsfrequenz-Extraktion, bedeutet dies, daß nur die Stelle des Maximums gesucht werden muß.

Auswertung der Kreuzkorrelationsanalyse zur EKG-Erkennung

Entsprechend der Applikation zur EKG-Erkennung mit Neuronalen Netzen wird auch hier ein Musterelektrokadiogramm den aufgenommenen Datensätzen entnommen. Im Gegensatz zu den Neuronalen Netzen kann hier nur ein Muster zum Vergleich herangezogen werden. Das Muster-EKG wird schrittweise über den zu untersuchenden Meßdatensatz „gezogen“. An jeder Stelle findet eine Berechnung der Kreuzkorrelationsfunktion statt mit anschließender Detektion des Maximums. Es ist zu erwarten, daß die so ermittelten Werte für die Korrelationskoeffizienten an den Auftrittstellen der EKG-Komplexe im Meßdatensatz, Zahlenwerte nahe eins annehmen. Zusätzlich sind somit die Auftrittsstellen der EKG-Komplexe bekannt.

<u>Korrelationsanalyse mit einem Muster des auszuwertenden Meßdatensatzes</u>

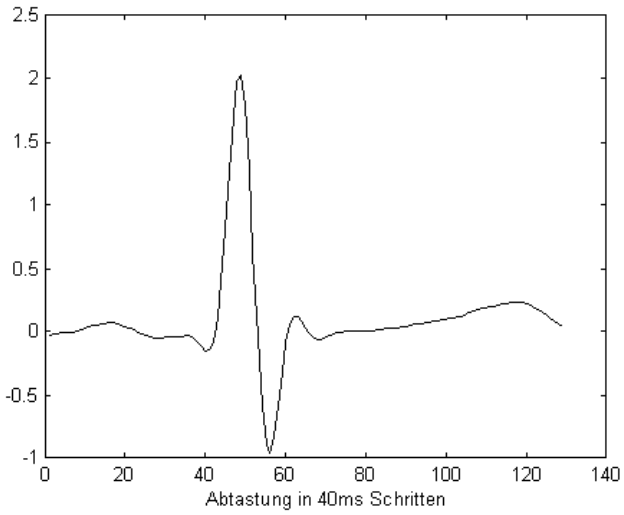


Abb. 2.4: Vergleichsmuster:

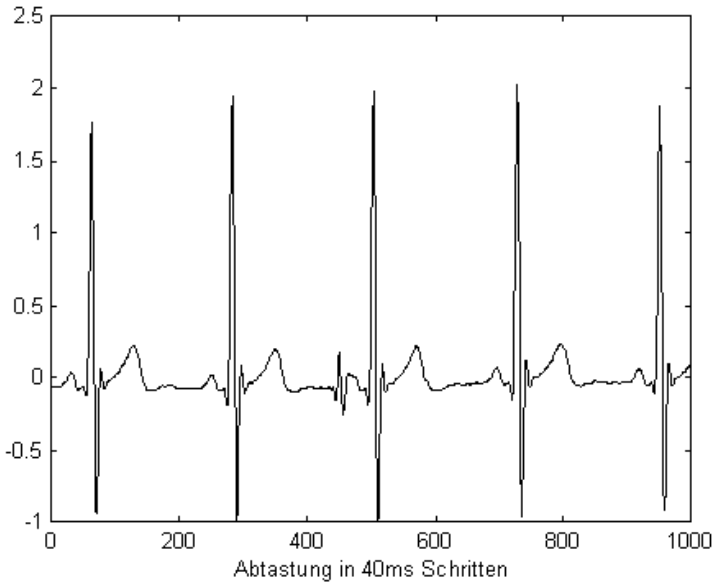


Abb. 2.5: Meßdatensatz:

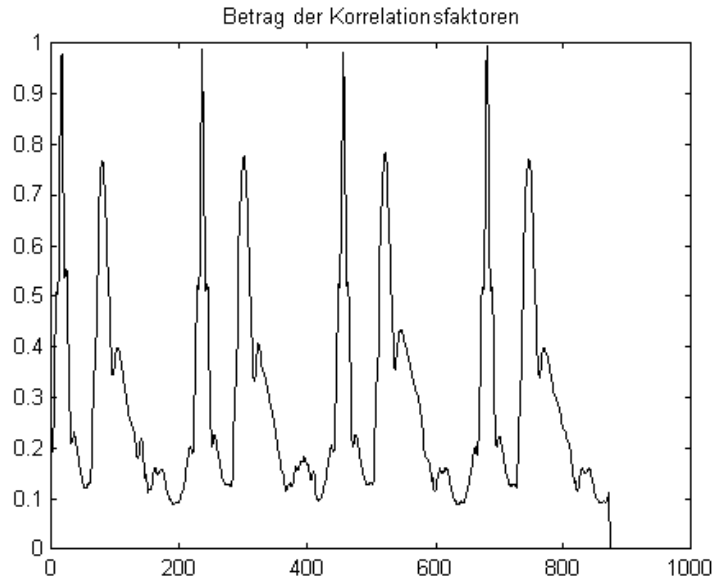


Abb. 2.6: Ergebnis der Kreuzkorrelationsanalyse:

Korrelationsanalyse mit gleichem Muster aber einem EKG-Signal eines anderen Probanden

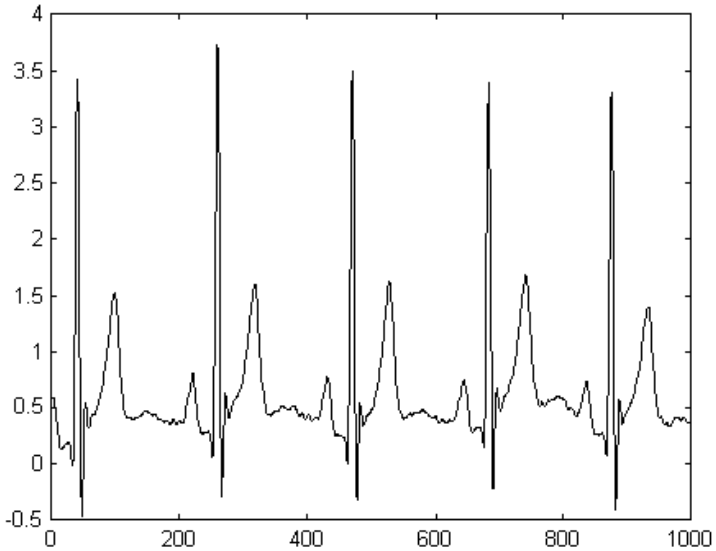


Abb. 2.7: EKG-Signal:

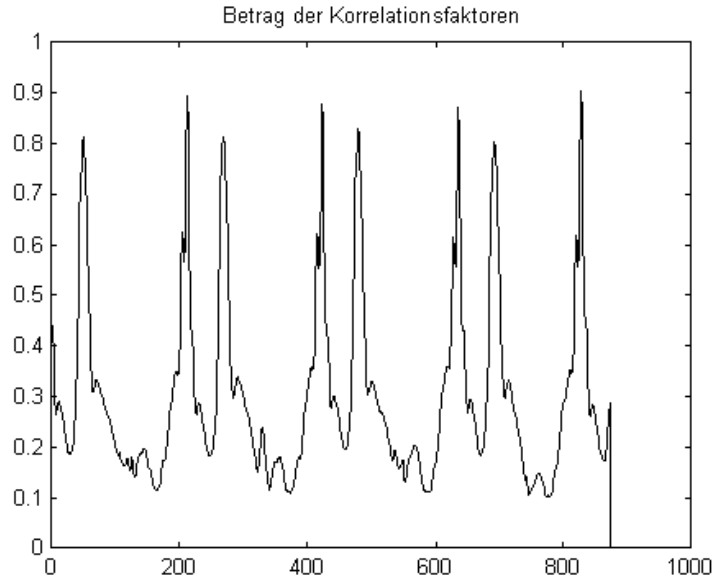


Abb. 2.8: Ergebnis der Korrelationsanalyse:

Korrelationsanalyse mit dem gleichen Muster wie bisher und verrauschtem EKG-Signal

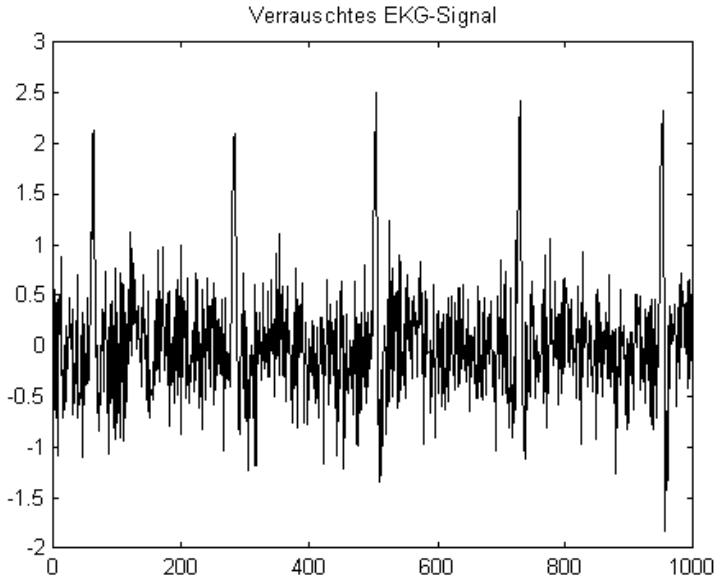


Abb. 2.9: Verrauschtes EKG-Signal
(Additives Rauschen mit einer Amplitude von 40% des Nutzsignal)

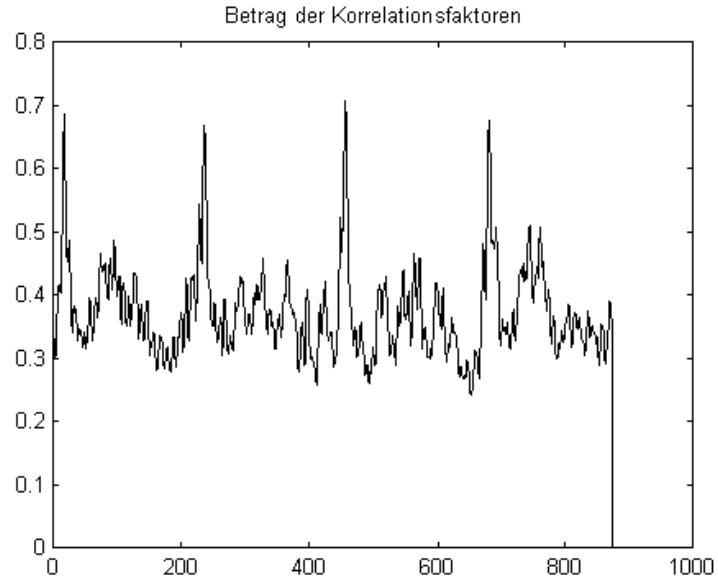


Abb. 2.10: Darstellung der Korrelationsfaktoren

Wie man aus den obigen Abbildungen erkennt, stellt auch die Korrelationsanalyse ein probates Mittel zur Mustererkennung dar. Hier muß allerdings ein variabler Schwellwert zur EKG-Komplex-Detektion eingesetzt werden. Die Adaption dieses Schwellwertes bildet einen weiteren Berechnungsschritt, welcher die Bearbeitungszeit für den gesamten Erkennvorgang nochmals verlängert. Außerdem können hierdurch Fehler bei der Detektion eintreten. Sämtliche Korrelationsanalysen wurden mit dem dafür programmierten Script-File „KORRKOEF“ unter MATLAB® durchgeführt.

Kreuzkorrelationsanalyse und Neuronale Netze im Vergleich

Die Erkennung von EKG-Signalen wurde unter Anwendung Neuronaler Netze als auch mit der Korrelationsanalyse durchgeführt. Dies ermöglicht den direkten Vergleich der beiden Verfahren im Bezug auf Mustererkennungsaufgaben.
Im ersten Schritt wurde das zu erkennende Muster aus dem Meßdatensatz extrahiert. Bei der Anwendung von Neuronalen Netzen nutzt man hier natürlich die Option aus, mehrere Muster zu trainieren. Das Neuronale Netzwerk konnte alle EKG-Komplexe eindeutig lokalisieren. Die Netzwerkausgabe für die erkannten Muster betrug Werte über 0,8. Dadurch wurde es ermöglicht, einen konstanten Schwellwert einzustellen, um die kontinuierlichen Werte am Ausgang des Netzes in binäre umzuwandeln. Bei der Vorgabe eines Musters aus dem Meßdatensatz erhält man mittels der Korrelationsanalyse dem Neuronalen Netz äquivalente Ergebnisse. Auch die Korrelationsanalyse lieferte für erkannte Muster Werte größer 0,8. Somit ist auch hier die Möglichkeit gegeben, die erhaltenen Werte in binäre Ausgaben umzuwandeln und mit Kenntnis der Abtastfrequenz, die Pulsfrequenz eindeutig zu berechnen.
Ein zweites Kriterium für den Vergleich der beiden Verfahren ist die sogenannte Generalisierungsfähigkeit. Dieser Begriff stammt eigentlich aus dem Bereich der Neuronalen Netze. Er beschreibt die Fähigkeit eines Netzwerkes, auch nichttrainierte, aber den trainierten Mustern ähnlichen Eingabemustern zu erkennen. Die Generalisierungsfähigkeit eines Neuronalen Netzes hängt somit nur von der Auswahl der Trainingsmuster ab. Im Gegensatz dazu besteht bei der Korrelationsanalyse nicht die Möglichkeit, gleichzeitig ein Vergleich mit mehreren Mustern durchzuführen. Dieser Nachteil zeigt sich direkt an den Ergebnissen der zweiten Musterekennungsaufgabe.
Hier wurde ein Muster aus einem ersten Datensatz als Referenzmuster herangezogen und ein zweiter Datensatz danach durchsucht.
Wie zu erwarten war, brachte die Korrelationsanalyse nicht die gleichen eindeutigen Ergebnisse entsprechend dem ersten Beispiel hervor. So ergeben vor allem die Stellen der T-Wellen einen großen Korrelationsfaktor. Dieses Ergebnis ist eindeutig dadurch zu begründen, daß bei der Korrelationsanalyse nur ein Muster zum Vergleich berücksichtigt wird. Bei der Anwendung Neuronaler Netze treten diese Schwierigkeiten nicht auf, da dem Netz viele verschiedene Muster eingelernt wurden. Stellt man die unterschiedlichen Trainingsmuster richtig zusammen, so können mit Neuronalen Netzen auch Muster, die den trainierten ähnlich sind, eindeutig klassifiziert werden. In der Generalisierungsfähigkeit der Neuronalen Netze liegt der hauptsächliche Vorteil gegenüber der Korrelationsanalyse bei der Anwendung für Pattern-Recognition-Aufgaben.
In einem letzten Test wird die Erkennsicherheit für Muster aus verrauschten Signalen überprüft. Dazu wurden unverrauschte Muster aus einem Datensatz als Referenzmuster entnommen. Der gleiche EKG-Meßdatensatz ist im Anschluß daran nachträglich verrauscht worden, d.h. es wurden Zufallszahlen (verteilt nach der Gauß-Verteilungsfunktion) in einem Vektor erzeugt, der die gleiche Länge wie der Vektor der Meßdaten besitzt. Die beiden Vektoren wurden im Anschluß daran elementenweise addiert, sodaß ein mit weißem Rauschen überlagerter Signalvektor entstand. Die Amplitude des Rauschsignals wurde auf 40% der maximalen Amplitude des Signalvektors begrenzt. Mit beiden Verfahren erhält man für diese Aufgabe eine gute Erkennsicherheit. Allerdings ist auch in diesem Vergleich die Korrelationsanalyse gegenüber Neuronaler Netze, wegen der fehlenden Generalisierungs-fähigkeit, unterlegen.
Zum Schluß soll noch erwähnt werden, daß die Auswertung der Signale mit der Korrelationsanalyse ca. dreimal soviel Zeit in Anspruch genommen hat wie die Mustererkennung mittels Neuronaler Netze. Dieser Faktor ist aber durch Optimierung der Software sicherlich noch zu verringern.
In der Summe der Vergleichsfaktoren ist die Anwendung Neuronaler Netze der Anwendung der Korrelationsanalyse für Mustererkennungsaufgaben vorzuziehen. Dies ist vor allem durch die schon erwähnte Generalisierungsfähigkeit der Neuronalen Netze zu begründen. Ein weiterer Vorteil Neuronaler Netze ist die unkomplizierte Anpassung eines Netzwerkes für die Klassifikation mehrerer verschiedener Muster. Solche Aufgaben würden mit der Korrelationsanalyse eine iterative Bearbeitung des Problems voraussetzen, sodaß auf jeden Fall die dann benötigte Bearbeitungszeit sehr weit über der eines Neuronalen Netzes liegen würde.

Mustererkennung und Klassifikation am Elektromyogramm der glatten Muskulatur

Glatte Muskulatur

Die Muskeln sind die „Maschinen“ eines Organismus, die für alle Bewegungsabläufe verantwortlich sind. Ihr Anteil am Gesamtkörpergewicht des Menschen ist mit ca. 40% - 50% entsprechend groß.
Der Aufbau der Muskeln kann in vier Untereinheiten aufgeteilt werden. Ein Muskel besteht aus vielen Muskelfaserbündeln. Die Muskelfaserbündel setzen sich ihrerseits aus vielen Muskelfasern zusammen, welche aus einer Vielzahl von Muskelfibrillen bestehen. Die Muskelfibrillen enthalten als aktive Kontraktionselemente die Muskelfilamente. Diese Aktin und Myosin genannten Filamente ziehen sich unter Energieverbrauch reißverschlußartig ineinander (Muskelkontraktion) und lösen sich - ebenfalls unter Energieverbrauch - nach beendeter Muskelanpassung wieder.

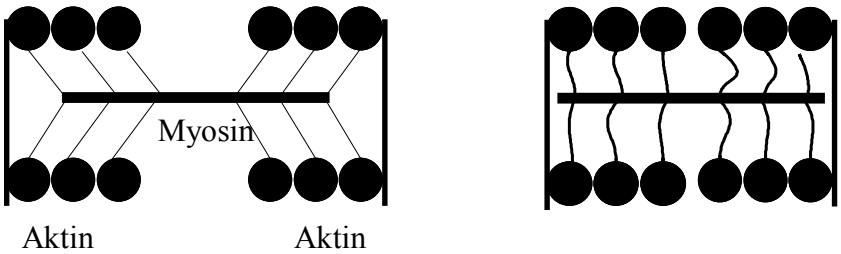


Abb. 3.1: Muskelfilamente im entspannten und angespannten Muskel
Die gesamte Muskulatur wird in drei Obergruppen aufgeteilt:
- Herzmuskulatur
- gestreifte Muskulatur
- glatte Muskulatur

Die Herzmuskulatur ist von ihrem Aufbau und ihren Eigenschaften her eine Mischform von glatter und quergestreifter Muskulatur. Die Aktivität der Herzmuskulatur wird wie bei allen Muskeln, durch elektrische Ströme ausgelöst, welche als Elektromyogramm bezeichnet werden. Da die Amplitude des Herzmuskelelektromyogrammes sehr groß ist, kann eine Ableitung einfach mittels Klebeelektroden auf bestimmten Hautpartien vorgenommen werden. Dieses Elektromyogramm ist allgemein unter der Bezeichnung Elektrokardiogramm (EKG) bekannt und ist entsprechend seiner Wichtigkeit sehr genau analysiert.
Die gestreifte Muskulatur wird auch Skelettmuskulatur genannt, da sie die Muskulatur des aktiven Bewegungsapparates des Körpers stellt. Ihr Name stammt von ihrem mikroskopischen Bild her, daß eine charakteristische Querstreifung erkennen läßt. Das Elektromyogramm der Skelettmuskulatur ist im Vergleich zu dem der glatten Muskulatur wegen der größeren Amplitude relativ einfach abzuleiten und wird auch entsprechend oft in der Diagnostik von Muskelerkrankungen eingesetzt.
Die glatte Muskulatur ist die Muskulatur der inneren Organe und Blutgefäße. Sie bildet einen Teil der Wand des Magen-Darm-Traktes, die Wände der Blase, des Uterus (Gebärmutter) sowie der Bronchien und der Blutgefäße. Die glatte Muskulatur zeichnet sich vor allem durch ihre Leistungsfähigkeit beim Verharren in einem Spannungszustand, der als Tonus bezeichnet wird, aus. Befindet sich die glatte Muskulatur in diesem Spannungszustand, treten im Gegensatz zur gestreiften Muskulatur keine Ermüdungserscheinungen ein und es findet auch kein erhöhter Stoffwechsel statt. Dieser Mechanismus läßt sich etwa mit einem gespannten elastischen Band vergleichen, welches sich in jedem Spannungszustand gleichsam in einem Ruhezustand befindet.
Die Einzelzuckungen der glatten Muskulatur, welche durch neurogene (nervale) oder myogene (Schrittmacherzellen) Innervation ausgelöst werden, können mehrere Sekunden andauern. Deshalb können Einzelzuckungen, die in weniger als einer Sekunde ausgelöst werden, sich zu einem vollständigen Tonus überlagern. Die zur Aktivierung der glatten Muskulatur notwendigen elektrischen Ströme bzw. elektrischen Spannungen sind in ihrer Amplitude wesentlich kleiner wie die der Skelett- und Herzmuskulatur. Bei der Aufnahme des Elektromyogrammes (EMG) eines glatten Muskels am lebenden Organismus wird dieses von den EMG der gestreiften Muskulatur und dem EMG des Herzmuskels überlagert. Dadurch ist eine Ableitung nur durch eine nachträgliche Signalverarbeitung möglich. Um diese Signalüberlagerungen zu vermeiden, wurde zunächst eine Klassifikation des EMG-Signals der glatten Muskulatur und besonders der darin enthaltenen Aktionspotentiale an einem isolierten Stück des Taenia coli (Dickdarmmuskel) vorgenommen.
Quellen: [3], [9]

Aktionspotentiale

Das Elektrolyt des Muskelgewebes enthält bestimmte Konzentrationen von Kalium-, Kalzium-, Natruim- und Chlorionen. Da die Zellmembranen der Muskelzellen für unterschiedliche Ionen unterschiedlich leitfähig sind, ist die Ionenkonzentrationen innerhalb und außerhalb der Zellen verschieden. Dadurch stellt sich, entsprechend der elektrochemischen Spannungsreihe, ein Potentialunterschied ein. Innerhalb der Zellen herrscht eine große Kaliumionenkonzentration bei gleichzeitig geringer Anzahl an Natrium- und Chlorionen. Außerhalb der Zellen sind die Ionenkonzentrationen genau umgekehrt, so daß im Ruhezustand ein Gleichgewichtspotential oder Ruhepotential besteht. Die Ionen sind bestrebt, durch die Zellwand zu diffundieren, um einen Konzentrationsausgleich zwischen intra- und extrazellulärem Raum zu schaffen. Für Natrium- und Chlorionen ist dies nur unter bestimmten Bedingungen möglich, bei denen sich die Permeabilität der Zellmembran kurzzeitig verändert. Im Gegensatz hierzu können Kaliumionen leicht durch die Membran diffundieren. Die Zelle ist nun mit der sogenannten Ionenpumpe durch einen aktiven Transportvorgang in der Lage, diesen Ionenaustausch wieder umzukehren.
Trifft ein Erregungsreiz die Zelle, so verändert sich die Permeabilität der Zellmembran und Natriumionen können in die Zelle eindringen. Diesen Vorgang nennt man Depolarisation, der bis zum Abblocken (bei einem bestimmten Schwellenpotential) der eindringenden Natriumionen andauert.
Die Ionenpumpe transportiert danach die Natriumionen wieder aus der Zelle heraus, infolge-dessen eine Repolarisation stattfindet. Die Kalium- und Chloridionen verhalten sich entsprechend. Der gesamte Potentialverlauf während dieser Ionenströme nennt man Aktionspotential.
Aktionspotentiale übertragen Informationen innerhalb eines Organismuses. So werden Signale von Nervenzellen als Aktionspotentiale durch die Nervenleitungen geschickt. Treffen diese auf Muskelzellen, bewirkt dies eine Innervierung der Muskelzellen. Einzelne Aktionspotentiale unterscheiden sich in Form und Amplitude kaum voneinander, so daß ihr Informationsgehalt gering ist.
Die eigentlichen Informationen sind in der Frequenz der Aktionspotentiale innerhalb der Bursts, der Dauer der Bursts und der Häufigkeit des Auftretens der Bursts selber kodiert. Diese Frequenzkodierung erlaubt es, daß Informationen über lange Strecken unverändert übertragen werden können.
Die folgenden Abbildungen zeigen Aktionspotentiale verschiedenen Ursprungs. Die Ableitungen wurden intracellulär mittels Microelektroden durchgeführt. Im Gegensatz zu den EMG-Ableitungen unter Anwendung der extracellulären Differenzspannungsmessung für die Durchführung dieser Diplomarbeit können somit das Ruhepotential und die tatsächliche Amplitude der Aktionspotentiale gemessen werden. In der Abbildung 3.3 sind die einzelnen Phasen eines Nervenaktionspotentials beschrieben, wobei die Amplitude und die Dauer des Aktionspotentials bei glatten und gestreiften Muskeln einen anderen Wert besitzen.

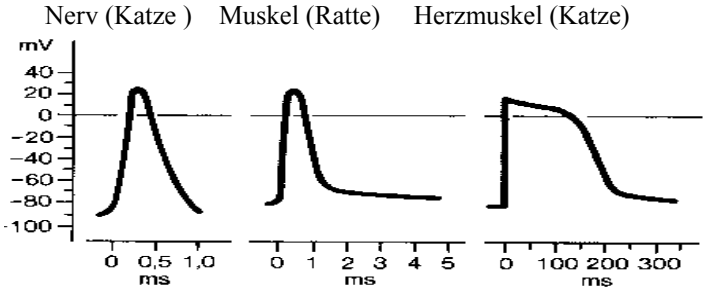


Abb. 3.2: Verschiedene Aktionspotentiale (Quelle [18] Seite 23):

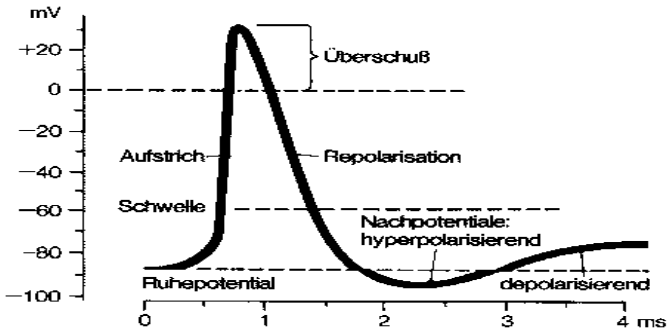


Abb. 3.3: Klassifizierung der einzelnen Phasen des Nervenaktionspotentials
(Quelle [18] Seite 24)

Quellen: [5], [18], [20], [21]

Aufnahme der EMG - Signale

Die Messungen wurden am Institut für digitale Signalverarbeitung der Fachhochschule für Technik in Mannheim durchgeführt, um effektiv verschiedene Meßnadelsysteme und Modifikationen des Vorverstärkers im Versuch zu erproben. Zusätzlich bestand somit die Möglichkeit, alle Meßergebnisse direkt mit den Signalverarbeitungs - Tools von MATLAB® auszuwerten. Die für die Messungen benötigten Kaninchendarmstücke wurden freundlicherweise vom physiologischen Institut der Universität Heidelberg zur Verfügung gestellt.

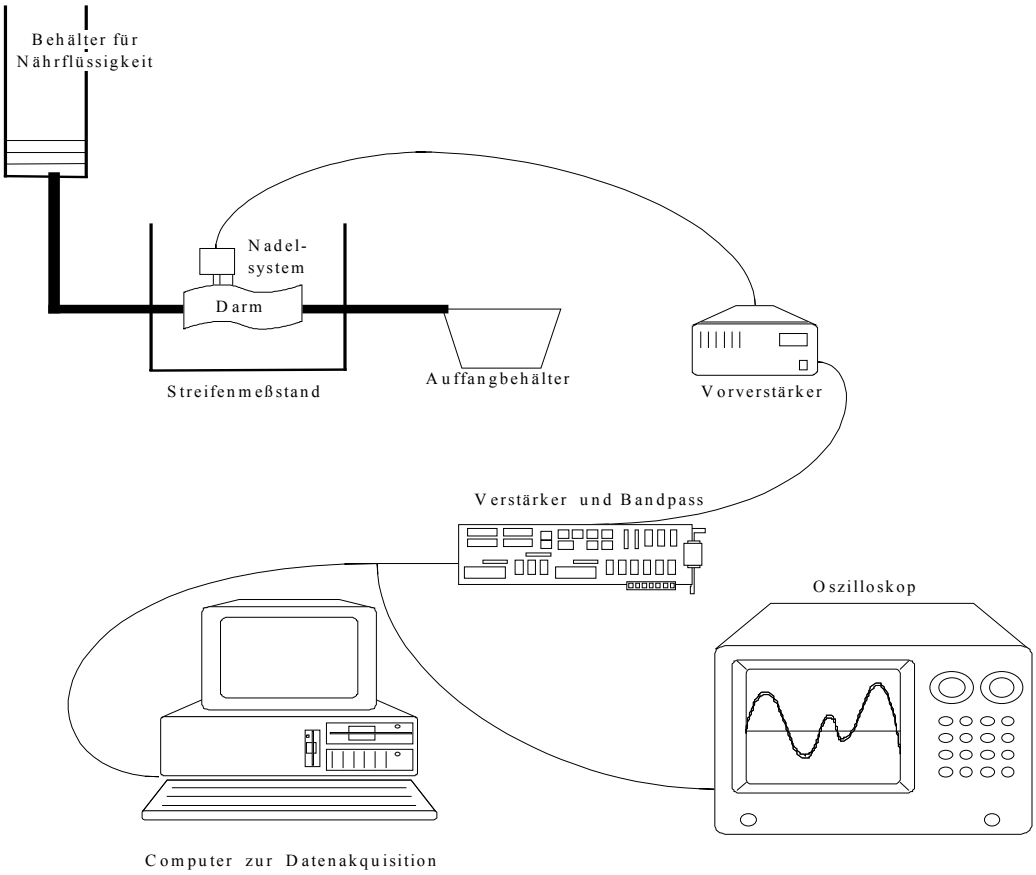
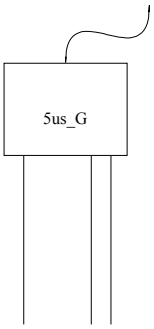


Abb. 3.4: Schematische Darstellung des Meßaufbaus

Das Nadelsystem und der Vorverstärker wurden am Institut für Analogtechnik an der Fachhochschule für Technik in Mannheim entwickelt. Die Optimierung beider Komponenten waren Gegenstand der Diplomarbeit von Markus Hauck, die zeitgleich mit dieser Diplomarbeit angefertigt wurde.
Der Vorverstärker arbeitet als Differenzspannungsverstärker. Mit einem nach außen geführten Schalter kann der Verstärkungsfaktor zwischen v = 10 oder v = 100 ausgewählt werden. Ein zweiter Schalter ermöglicht die Auswahl der Grenzfrequenz des Eingangshochpasses mit f_{grenz} = 10 mHz oder f_{grenz} = 2,4 Hz. Mit dem Eingangshochpass können somit die Slow Waves unterdrückt werden, damit nur noch die Bursts der Aktionspotentiale abgeleitet werden.
Für die ausgewerteten Messungen wurden vergoldete Stahlnadeln mit unsymmetrischem Abstand zum Bezugspotential verwendet. Die Auswahl dieses Nadelsystems erfolgte empirisch, da bisher keine extrazellulären EMG-Messungen in der Literatur verzeichnet sind, auf deren Ergebnisse aufgebaut werden konnte.
Aufbau des Sensors:



Nadeltyp: 5us_G (Nadelabstand insgesamt 5mm, unsymmetrische Anordnung, Material Gold)
Bezugspotential: mittlere Nadel
Differenzspannung 1: linke Nadel
Differenzspannung 2: rechte Nadel
Abb. 3.5: Meßnadelsystem
Die von dem Nadelsystem aufgenommen EMG-Signale werden zunächst mit dem Eingangshochpass des Vorverstärkers gefiltert und mit dem Vorverstärker verstärkt. Im Anschluß daran folgt eine zweite Verstärkung des Signals mit einem kommerziellen Verstärker, der zusätzlich eine Hoch-, Tief- oder Bandpassfilterung bereitstellt. Mit diesem integrierten frequenzselektiven Filter kann auf einfache Weise die Auswirkung einer Bandbegrenzung auf das EMG-Signal schon während der Messungen untersucht werden. Das verstärkte und gefilterte EMG-Signal wird dann parallel an ein Oszilloskop und einen Computer mit einer Datenakquisitionskarte weitergeleitet.
Um eine erste Datenreduzierung vorzunehmen, wird mit dem Oszilloskop das Meßergebnis permanent überprüft, so daß nur relevante Signalausschnitte aufgezeichnet werden. Die Datenerfassung erfolgt über einen Analog-Digitalwandler mit 16 Bit Auflösung, wobei eine Abtastrate von 1000 Hz eingestellt wurde. Bei Signalfrequenzen von bis zu 20 Hz kann daher eine Unterabtastung und damit verbundene Aliasing-Effekte ausgeschlossen werden.
Ein Hauptproblem bei den EMG-Messungen war die Reduzierung von Störsignalen, welche vor allem aus 50Hz Brummspannungen und höherfrequenten, von den Meßgeräten abgestrahlten Störungen bestand. Deshalb wurde eine Abschirmvorrichtung in Form eines Faradayschen Käfigs um den Streifenmeßstand gebaut. Dieser wurde mit der gemeinsamen Masse aller am Meßaufbau beteiligten Apparate verbunden. Als besonders wirkungsvoll hat sich der Massebezug der Nährflüssigkeitssäule, sowie die Abschirmung sämtlicher signalführender Leitungen herausgestellt, die ansonsten wie eine Antenne für Störsignale wirkten. Selbstverständlich muß bei der Verbindung der Masseleitungen darauf geachtet werden, daß keine Masseschleifen entstehen, da diese wiederum elektromagnetische Einkopplungen von Störsignalen zur Folge haben können.
Wegen der großen Verstärkung ($v \geq 1000$) und der Breitbandigkeit des Verstärkers ($10\ mHz \leq f \leq 100\ kHz$) werden mit diesem System auch Spannungen aufgenommen, die durch die Bewegung des Darmes entstehen. Diese Artefakte im Signal können in der Amplitude ein Vielfaches der Aktionspotentiale betragen. So bestand ein weiteres Kriterium darin, einen erschütterungsfreien Standort für den Streifenmeßstand zu gewährleisten. Für EMG-Messungen an nicht isolierten Organen sind hierdurch große Störsignale zu erwarten. Da diese Störsignale sich in der Frequenz nur wenig von Aktionspotentialen unterscheiden, wird eine Separation der Bewegungsartefakte von den EMG-Peaks nicht immer möglich sein.

Die Aktivität der Kaninchendärme hängt immer auch von der Funktionsfähigkeit ihres Stoffwechsels ab, weshalb in den durchgeführten Messungen nach zwei bis drei Stunden die Därme keine Aktionspotentiale mehr lieferten. Deshalb sollten zur Optimierung des Streifenmeßstandes für zukünftige EMG-Messungen Glucoselösungen, calcium- und natriumhaltige Elektrolyte sowie eine Anreicherung der Nährflüssigkeit mit Sauerstoff vorgesehen werden. Gleichermaßen wird auch mit sinkender Temperatur der Stoffwechsel und somit die Aktivität von Organen reduziert, so daß eine konstante Darm- und Nährflüssigkeitstemperatur von ca. 36°C-37°C eine weitere Verbesserung der Meßergebnisse ergeben würde.

Quellen: [3], [17]

Analyse der EMG - Signale

Zunächst müssen die aufgenommenen EMG-Signale klassifiziert werden. Eine erste Unterscheidung findet in der Trennung der niederfrequenten Slow Waves und der höherfrequenteneren Aktionspotentiale statt. In dieser Diplomarbeit sollten jedoch hauptsächlich die Korrelation zwischen Aktionspotentialen der glatten Muskulatur und der Bewegung des Muskels, sowie die Auswirkung verschiedener Medikamente auf das Feuern von Aktionspotentialen untersucht werden, weshalb die Slow Waves unberücksichtigt blieben.
Die immer im EMG-Signal auftretenden Slow Waves wurden bereits in vorhergehenden Diplomarbeiten untersucht. Der Gegenstand dieser Untersuchungen war eine Korrelation zwischen Slow Waves und dem Füllungszustand von Rattenblasen zu finden.

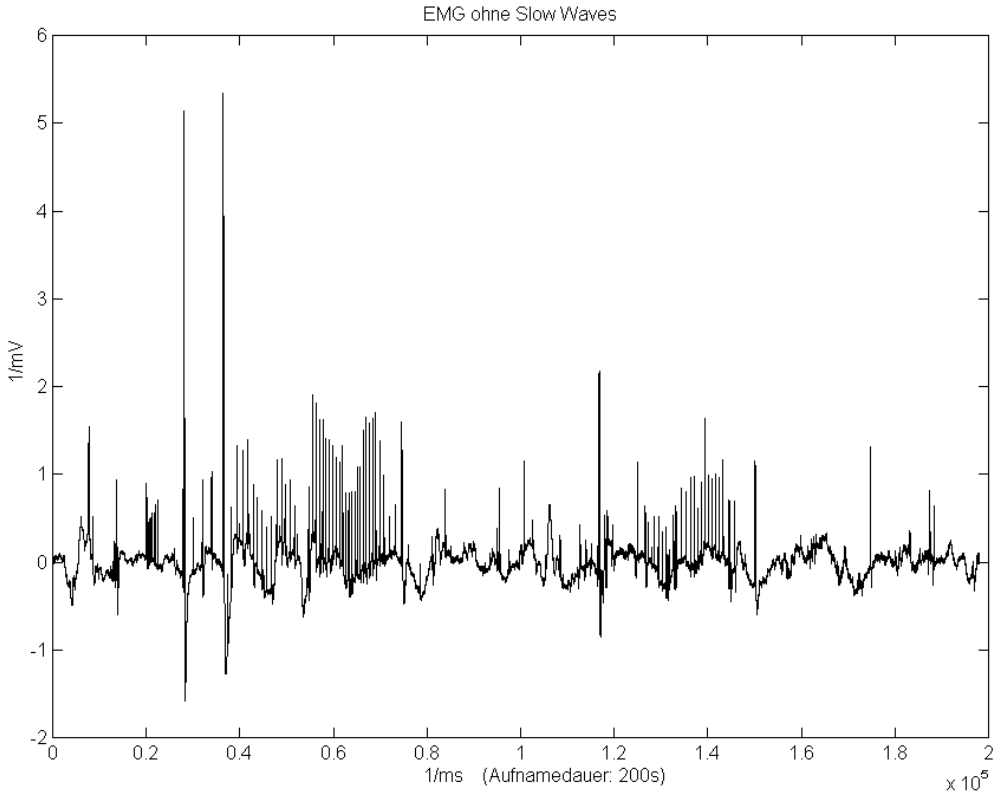


Abb. 3.6: EMG - Ableitung eines Kaninchendarmes
(Messung 9 vom 13.12.1994)

Analyse der Peaks eines EMG

Die Anregung, d.h. die Kontraktion bzw. Entspannung glatter Muskeln, kann bei verschiedenen mit glatter Muskulatur ausgestatteten Organen auf unterschiedliche Weise geschehen. Wie sich bei Fachgesprächen mit Herrn Prof. Rüegg und Herrn Prof. Fink vom Physiologischen Institut der Universität Heidelberg ergab, ist von der glatten Muskulatur des Magens bekannt, daß Slow Waves für eine Aktivierung der Magenmuskulatur ausreichen. Andererseits ist bei der glatten Muskulatur des Darmes noch nicht endgültig geklärt, ob die Bewegungen des Darmes (Peristaltik) nur durch Bursts von Aktionspotentialen oder auch nur durch Slow Waves ausgelöst werden können. Grundsätzlich sollte aber eine Korrelation zwischen der Frequenz der Aktionspotentiale innerhalb eines Burst und der Peristaltik des Darmes bestehen.
Bevor nun diese Zusammenhänge überprüft werden konnten, mußte zunächst eine erste Klassifikation der aufgenommenen Signale durchgeführt werden. Zu diesem Zweck wurden insgesamt acht Parameter aus den aufgenommenen Aktionspotentialen und den Bursts von Aktionspotentialen berechnet. Die Auswahl der Parameter wurde entsprechend der in der Literatur üblichen Klassifikation von Aktionspotentialen (von Nerven) vorgenommen. Alle Parameter wurden durch eine Mittelung über mehrere Werte innerhalb einer Meßreihe ermittelt. Die angegebenen Parameter ergeben sich dann aus dem Mittelwert der Parameter aus mehreren Meßreihen.

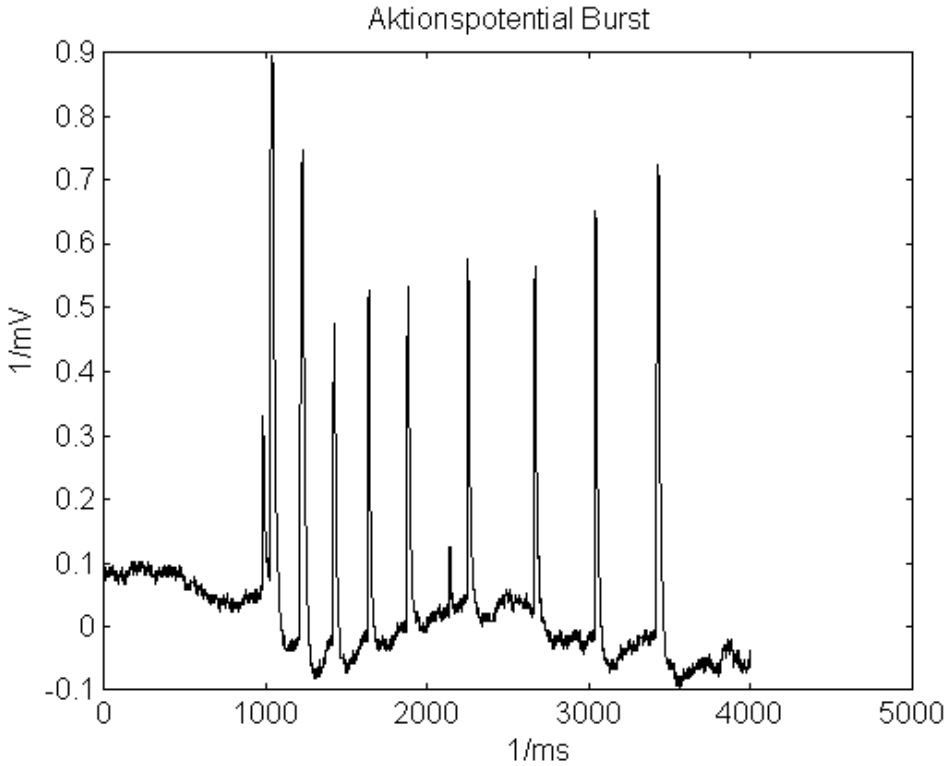


Abb. 3.7: Folge von Aktionspotentialen (Burst)

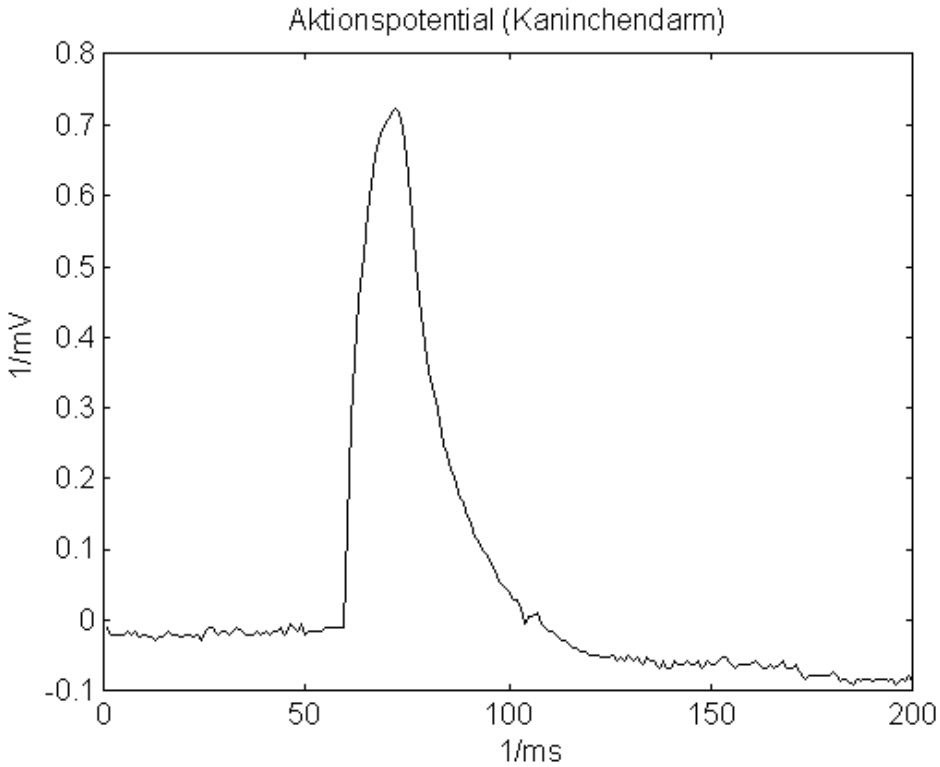


Abb. 3.8: Aus dem Meßsignal extrahiertes Aktionspotential

Maximale Spannung, Standardabweichung und Varianz der EMG - Peaks

Die gemessene Amplitude der EMG - Peaks hängt von verschiedenen Faktoren ab. Einer dieser Faktoren ist der Übergangswiderstand zwischen dem Nadelsystem und dem Darmgewebe. Während einer Messung (über 100s oder 200s) können sich die Umgebungsbedingungen, wie z.B. Leitfähigkeit des Elektrolytes oder die Verbindung des Nadelsystems, mit dem Gewebe verändern. Ein weiterer Faktor ist die Überlagerung von Aktionspotentialen verschiedener Schrittmacherzellen, die sich in der Amplitude der Peaks zeigt. Diese Einflüsse führen folglich zu sehr unterschiedlichen Maxima der EMG - Peaks, wodurch auch ein Hauptproblem der extrazellulären Ableitung von EMG-Signalen charakterisiert ist.

Berechnung der Varianz σ^2:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Die Standardabweichung σ berechnet sich entsprechend aus der Wurzel der Varianz. Die nachfolgende Tabelle gibt eine Übersicht der gemessenen Werte für die Standardabweichung, Varianz und die Maxima der EMG - Peaks an.

Burstfolge	Standardabweichung	Varianz	Maximum/mV
1	0,134	0,017	0,89
2	0,338	0,114	1,645
3	0,214	0,045	1,7
4	0,3	0,09	1,6
5	0,38	0,14	1,54

Im Durchschnitt ergibt sich:

Standardabweichung: 0,27
Varianz: 0,14
Maximum: 1,54 mV

Quellen: [4]

Anstiegszeit, Abfallzeit und Dauer der EMG - Peaks

Der zeitliche Verlauf der Aktionspotentiale stellt das Hauptunterscheidungsmerkmal der Aktionspotentiale unterschiedlichen Ursprungs dar. So dauert ein Aktionspotential am Nerv ca. 1ms, am Skelettmuskel etwa 10ms und am Herzen mehr als 200ms. Die Schwellwerte für den Anfang und das Ende der Flanken bei der Messung der Anstiegs- und Abfallzeit (Rise- und Fall -Time) wurden, wie in der Meßtechnik üblich, bei 10% bzw. 90% der maximalen Amplitude festgelegt. Die Dauer der Aktionspotentiale wurde dann beginnend mit der 10%-Marke der ansteigenden Flanke bis zur 10%-Marke der abfallenden Flanke bestimmt. Dadurch werden auch Fehler bei der Messung der verschiedenen Laufzeitparameter, die durch Rauschen verursacht werden können, ausgeschlossen.

Gemessene bzw. berechnete Parameter

Anstiegswinkel: $\varnothing\ \alpha_{an} = 87{,}1°$
Abfallwinkel: $\varnothing\ \alpha_{ab} = 78{,}9°$
Anstiegszeit: $\varnothing\ t_{an} = 11{,}3$ ms
Abfallzeit: $\varnothing\ t_{ab} = 23{,}2$ ms
Gesamtdauer (eines Peaks): $\varnothing\ t_{ges} = 34{,}5$ ms

Auswirkung von Noradrenalin und Acetylcholin auf das EMG - Signal

Muskeln werden gewöhnlich durch Aktionspotentiale der innervierenden Motoneuronen oder der Schrittmacherzellen erregt. Bei myogener Erregung durch Schrittmacherzellen werden präpotentiale bzw. Schrittmacherpotentiale erzeugt, welche eine Depolarisation bis zu einer Schwelle bewirken, bei der Aktionspotentiale ausgelöst werden. Die Schrittmacherzellen des Taenia Coli (Dickdarmmuskulatur) können durch Zugabe von Acetylcholin bis nahe der Schwelle depolarisiert werden, wodurch die Frequenz der Aktionspotentiale erhöht wird. Im Gegensatz dazu wirkt die Applikation von Noradrenalin hyperpolarisierend und senkt somit die Frequenz der Aktionspotentiale beim Taenia Coli. Der Tonus der glatten Muskulatur verändert sich mit der Frequenz der Aktionspotentiale, so daß die Verabreichung von Acetylcholin eine Erregung bewirkt und die Zugabe von Noradrenalin eine Entspannung.
Für die Aufnahme und Klassifikation der Aktionspotentiale und der Bursts von Aktionspotentialen bietet es sich an, durch die Gabe von Acetylcholin gezielt Bursts von Aktionspotentialen zu erzeugen. Verschwinden diese Peaks nach der Gabe von Noradrenalin, ist dies ein Nachweis für die Zugehörigkeit des Signals zum EMG der glatten Muskulatur. Dieser Mechanismus konnte auch während der EMG-Messungen verifiziert werden.

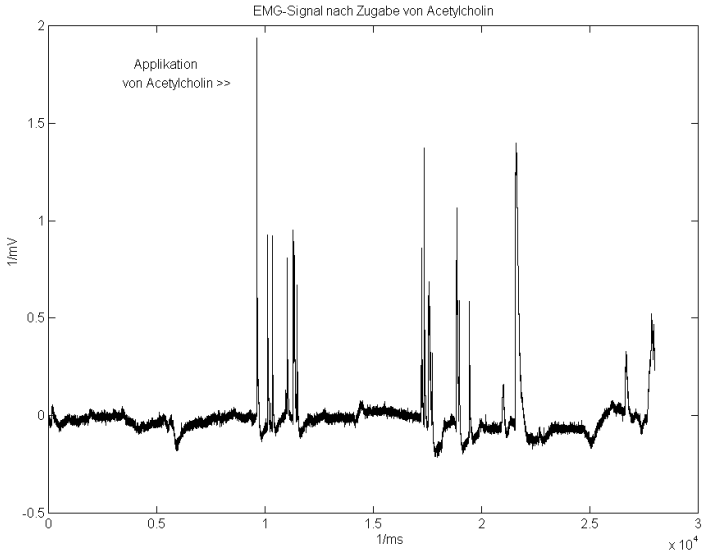


Abb. 3.9: EMG - Signal nach der Zugabe von Acetylcholin
(Messung vom 28.11.1994)

Wie man deutlich erkennen kann, folgen kurz nach der Applikation von Acetylcholin eine Reihe von Aktionspotentialen. Diese treten immer in Bursts unterschiedlicher Länge und Frequenz auf.

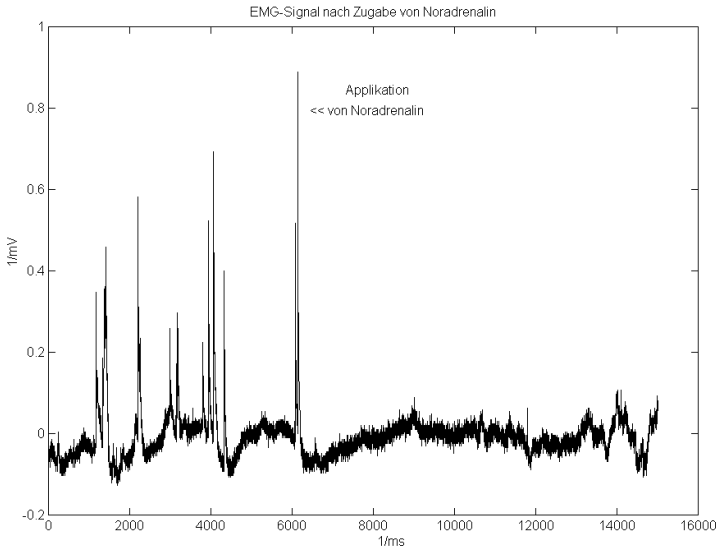


Abb. 3.10: EMG - Signal nach der Zugabe von Noradrenalin
(Messung vom 28.11.1994)

Auch hier erhält man die erwartete Veränderung im EMG-Signal. Kurze Zeit nach der Zugabe von Noradrenalin erlischt die Aktivität der glatten Muskulatur des Kaninchendarmes, was sich in der Nullinie (nur noch Rauschen) des EMG - Signals widerspiegelt.
Die Korrelation der verschiedenen Signalabschnitte mit den entsprechenden Medikamenten stimmen somit sehr gut mit den Angaben in der entsprechenden Fachliteratur überein. Nähere qualitative Zusammenhänge zwischen der Medikamentendosierung und der Frequenz der Aktionspotentiale wurden nicht untersucht. Infolge des einfachen Meßaufbaus konnten leider auch keine Untersuchungen zur Korrelation der Frequenz der Aktionspotentiale innerhalb der Bursts und der Stärke bzw. Dauer des Tonus der glatten Muskulatur angestellt werden.

Mustererkennung an EMG-Signalen

Motivation für die Anwendung von Mustererkennungsmethoden auf EMG-Signale

Das Elektromyogramm glatter Muskeln besitzt, wie oben beschrieben, eine im Vergleich zu anderen biologischen Signalen eine kleine Amplitude. Bei der Messung von Signalen unter realen Bedingungen, d. h. nicht in einer optimalen Laborumgebung, werden auch immer Störsignale verschiedenster Art aufgezeichnet. Dabei sind vor allem elektromagnetische Einkopplungen, z.B. der 50 Hz Netzspannung, zu erkennen, die man allerdings relativ einfach direkt bei der Aufnahme mit Hardwarefiltern oder nach der Aufnahme durch Softwaremethoden herausfiltern kann. Wesentlich problematischer stellt sich die Filterung der durch die Bewegungsartefakte hervorgerufenen Störspannungen dar, da diese sich in ihrer Form und Amplitude häufig nur wenig von Aktionspotentialen unterscheiden. Weitere Störquellen sind alle Organe bzw. Muskeln, vor allem der Herzmuskel, welche zur Innervation elektrische Signale benötigen. Die störenden Elektromyogramme breiten sich mehr oder weniger bedämpft über den gesamten Organismus aus und überlagern sich additiv mit dem zu messenden EMG. Besonders das Elektrokardiogramm des Herzens kann mit seiner vergleichsweise großen Amplitude das gesuchte EMG völlig überdecken.

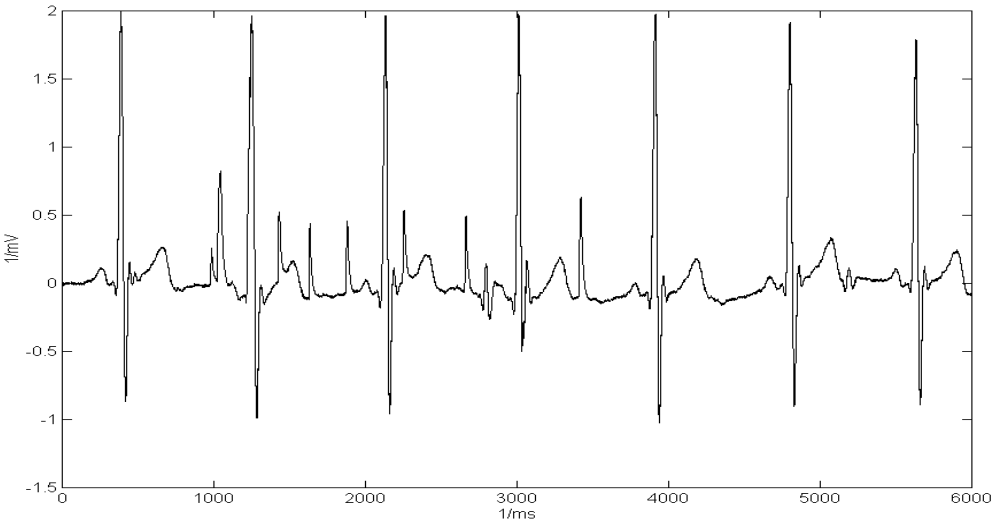
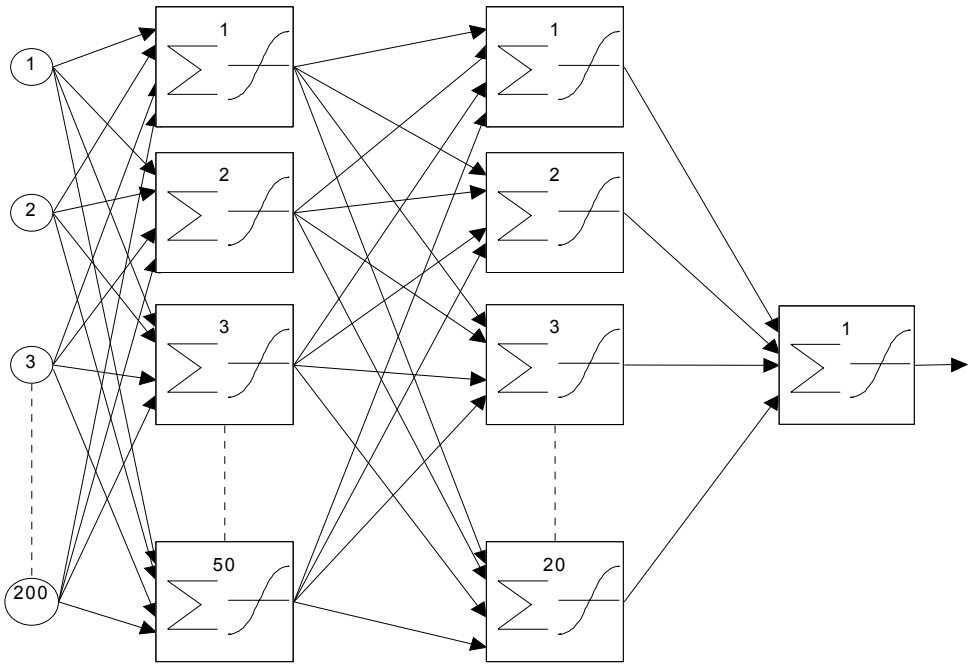


Abb. 3.11: Überlagerung eines EMG-Burst mit einem EKG

Um nun eine automatisierte EMG-Detektion durchzuführen, ist bei den meisten Messungen aus den oben genannten Gründen eine Spitzenwerterkennung unangebracht. Deshalb müssen zunächst unter Anwendung Neuronaler Netze die EKG-Komplexe -falls vorhanden- erkannt und eliminiert werden. Danach können die verbleibenden Peaks selektiert und mit einem Neuronalen Netz auf die Zugehörigkeit zum EMG-Signal untersucht werden. Die Realisierung der EMG-Erkennung wird im folgenden beschrieben.

Auslöschung der EKG-Komplexe aus dem Meßdatensatz

Wie schon in der Applikation „EKG-Erkennung“ unter Kapitel 1.3.6 wird auch hier ein dreistufiges Neuronales Netz verwendet. Dieses mußte allerdings noch einmal neu „aufgebaut“ werden, da die Randbedingungen, wie Abtastfrequenz und Störsignale, hier verändert sind. Nach einer längeren Erprobungsphase mit unterschiedlich tiefen Layerschichten, hat sich ein Feed-Foreward-Netzwerk mit 50 Neuronen im ersten und 20 Neuronen im zweiten Hidden-Layer als besonders geeignet herausgestellt. Als Lernregel wurde auch hier der Backpropagationalgorithmus (siehe Kapitel 1.3.5) mit adaptiver Lernrate und Momentum-Term verwendet.



Sensoren 1. Hidden-Layer 2. Hidden-Layer Output Unit

Abb. 3.12: Netzwerk zur EKG-Erkennung

Die Auswahl der Trainingsmuster konnte auf wenige Datensätze reduziert werden, da sich die EKG-Komplexe in ihrer Form und Amplitude deutlich von den EMG-Signalen unterscheiden. Mit der Verwendung eines dreistufigen Neuronalen Netzes wurde zusätzlich sichergestellt, daß zur Klassifikation beider Muster genügend charakteristische Merkmale aus den Datensätzen extrahiert werden können.

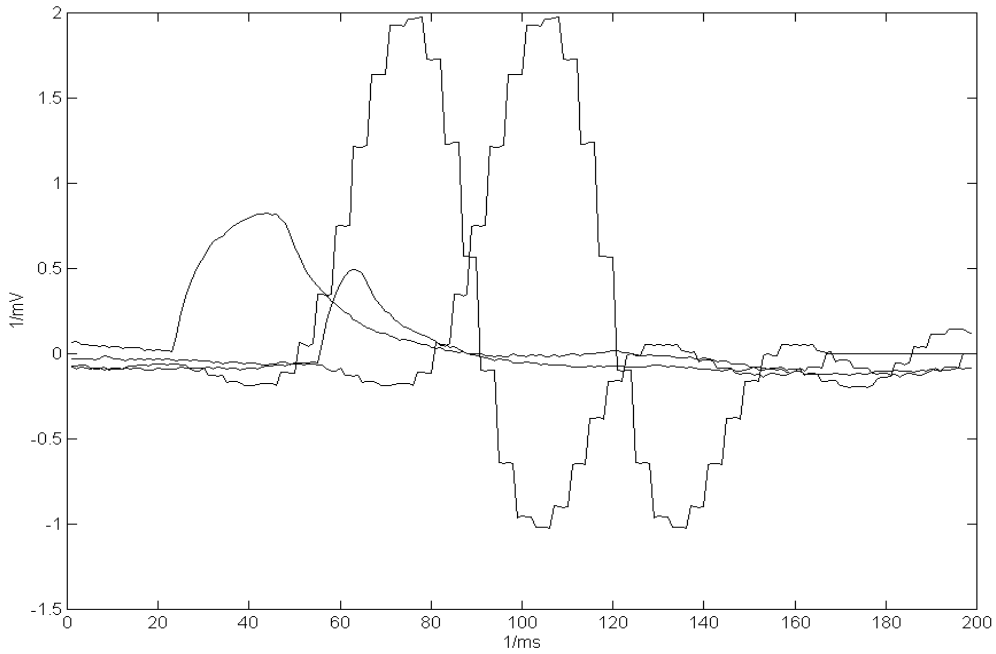


Abb. 3.13: Trainingsmuster zur EKG-Erkennung

Die Stufen in den EKG-Komplexen entstanden durch die nachträgliche „Verlängerung“ des mit 250 Hz abgetasteten EKG-Signals. Dadurch wurde erreicht, daß das EKG-Signal zeitlich mit dem mit 1000 Hz abgetasteten EMG-Signal übereinstimmt. Für die Mustererkennung mit einem Neuronalen Netz hat dies keine weiteren Konsequenzen. Die Simulation des Neuronalen Netzwerkes ist auch bei dieser Applikation mit dem Programm MATLAB® durchgeführt worden.

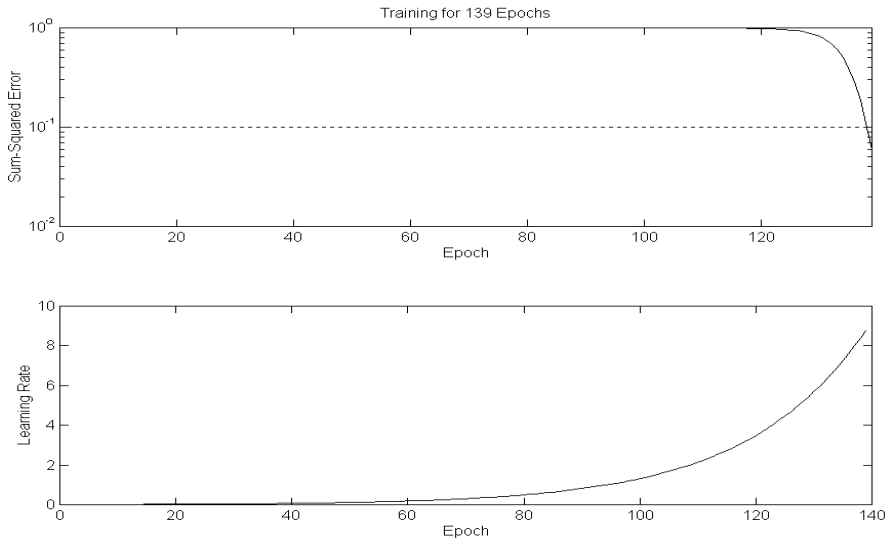


Abb. 3.14: Verlauf des Netzwerktrainings

Am Verlauf des Netzwerktrainings erkennt man die Effizienz des adaptiven Lernfaktors. Nach nur 139 Epochen erreicht die Summe der quadratischen Fehler einen Wert kleiner 0,01 (innerhalb einer Epoche werden alle Trainingsmuster an das Netzwerk angelegt, die Summen der quadratischen Fehler am Ausgang berechnet und alle Gewichte nach der Backpropagationregel modifiziert).
Mit dem trainierten Neuronalen Netz können jetzt die PQRS-Komplexe im Meßdatensatz detektiert werden. Da diese für die weitere Signalverarbeitung nicht mehr benötigt werden, wird an den Stellen ihres Auftretens das Meßsignal identisch Null gesetzt. Die dadurch provozierten Frequenzen im Amplitudendichtespektrum des Meßsignales können außer acht gelassen werden, da nur nach Mustern im Zeitbereich gesucht wird. Das bearbeitete Signal enthält nur noch die Peaks des aufgenommenen EMG's und die verbleibenden Störsignale.

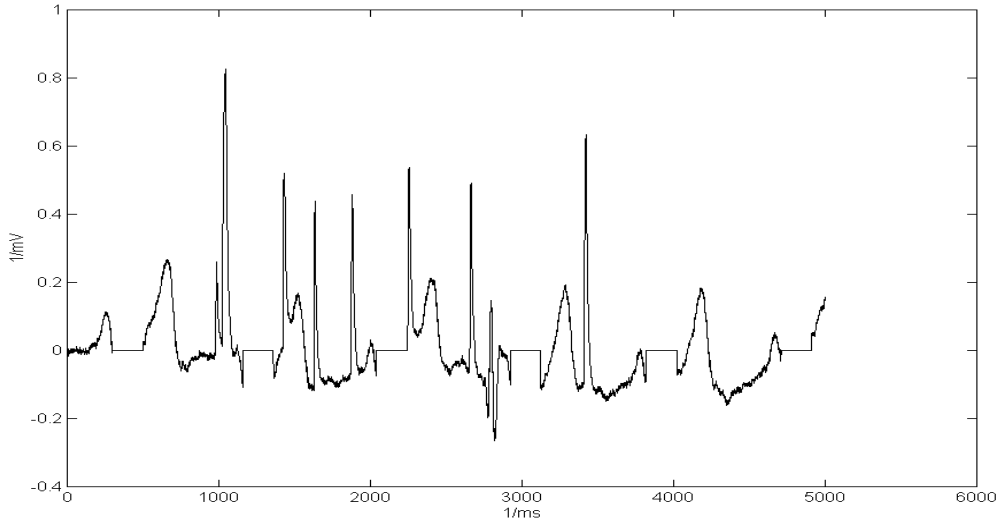


Abb. 3.15: Überlagertes Signal aus EKG und EMG nach der Filterung

Im gefilterten Signal verbleibt nur noch der gesuchte EMG-Burst, die T-Welle des EKG und Rauschanteile. Durch eine einfache Peakdetektion können jetzt die EMG-Aktionspotentiale lokalisiert werden.

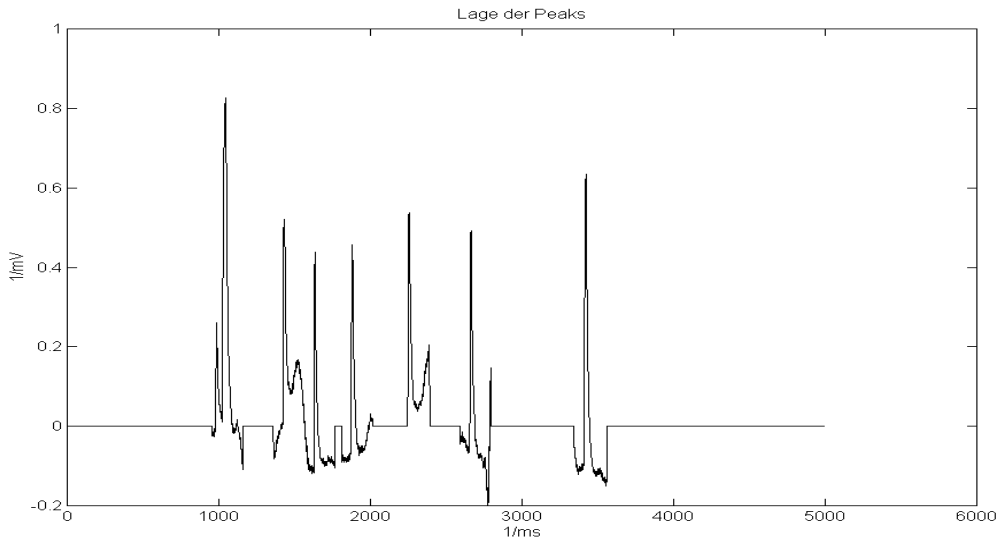


Abb. 3.16: Lokalisierte Peaks

Auswahl und Training des Neuronalen Netzes zur EMG-Verifizierung

Bei den extrahierten Peaks muß es sich nicht zwangsläufig um die gesuchten Aktionspotentiale des EMG des Taenia Coli (Dickdarmmuskel) handeln, da, wie schon erwähnt, die Bewegungsartefakte den Aktionspotentialen sehr ähneln können. Unter Anwendung eines weiteren Neuronalen Netzes werden deshalb die selektierten Peaks mit Mustern von EMG-Aktionspotentialen verglichen. Auch hierfür wurden mehrere Neuronale Netze mit unterschiedlich tiefen Layerschichten erprobt. Das ausgewählte Neuronale Netzwerk besteht aus einer Eingangsschicht mit 200 Neuronen und einer Ausgangsschicht mit 2 Neuronen. Für das Training wurde auch hier die Backpropagation-Lernregel mit adaptiver Lernrate und Momentum-Term angewendet.

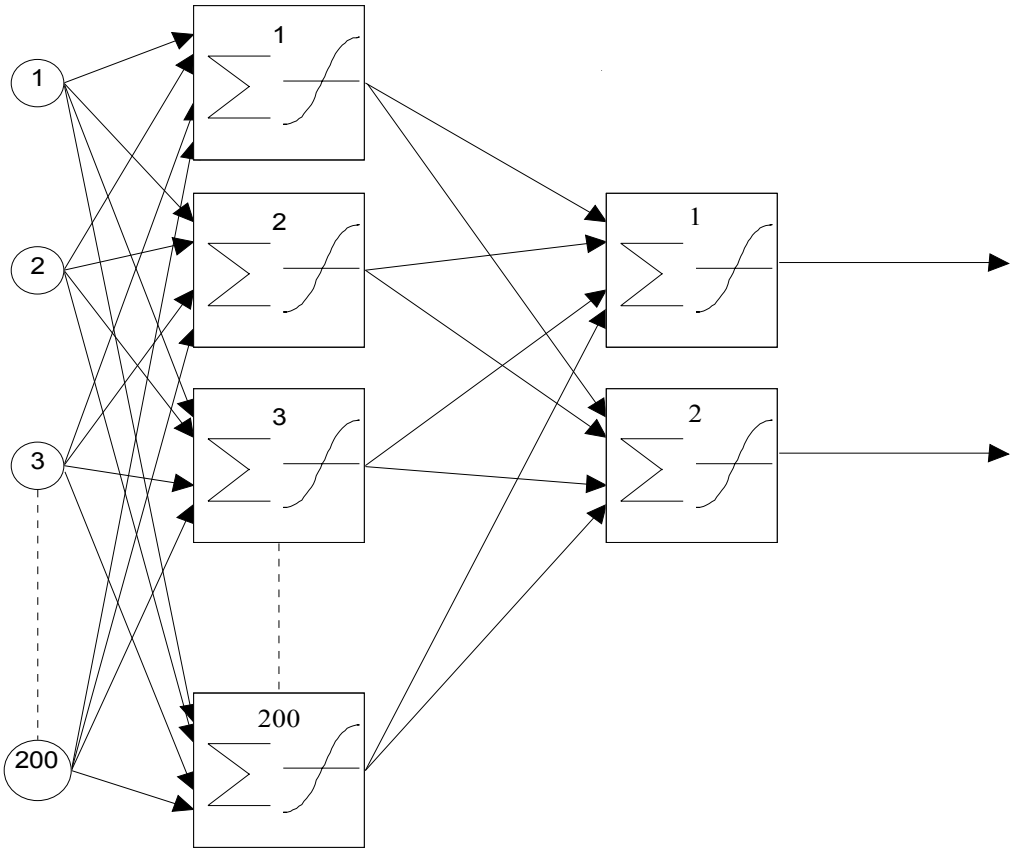


Abb. 3.17: Netzwerktopologie zur Aktionspotential-Verifizierung

Die Fähigkeit eines Neuronalen Netzes, verschiedene Muster eindeutig zu klassifizieren und trotzdem innerhalb einer Klasse zu generalisieren, hängt in erster Linie von der Auswahl der Trainingsmuster ab. Da die gesuchten Aktionspotentiale im EMG glatter Muskeln in der Amplitude und in ihrer Form variieren, muß sich dies also auch in den Trainingsmustern zeigen. Die geeignetsten Muster für das Netzwerktraining wurden hier sorgfältig durch einige Testläufe mit verschiedenen EMG-Signalen ermittelt.

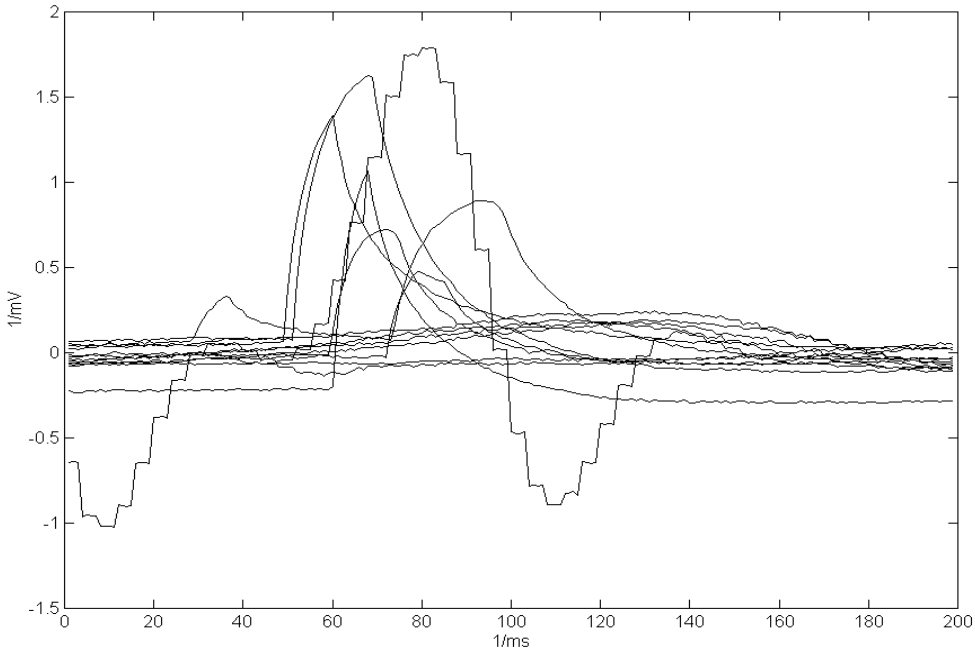


Abb. 3.18: Trainingsmuster

Wegen der stark unterschiedlichen Amplituden erscheinen einige „Falsch-Muster“ in der Abbildung ähnlich einer Grundlinienschwankung. Bei der Mustererkennung unter Anwendung Neuronaler Netze ist jedoch die Form eines Signals entscheidend, so daß das Netzwerk auch für diese als falsch zu erkennenden Muster trainiert werden muß.

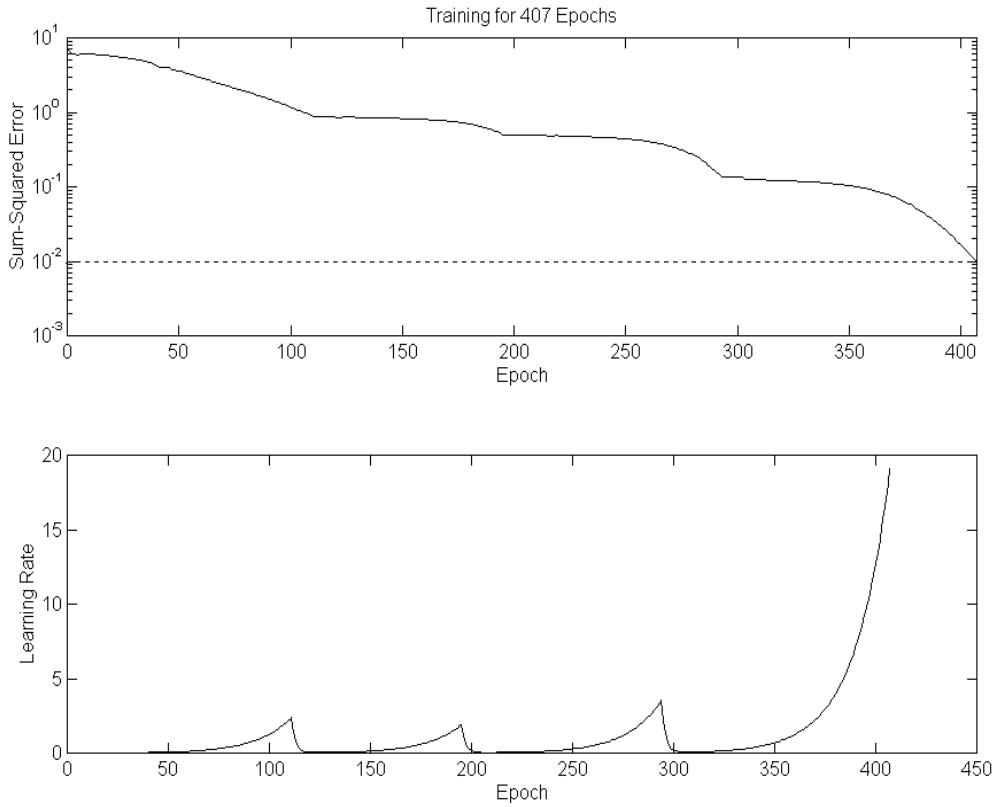


Abb.3.19: Verlauf des Netzwerktrainings

Die Oszillationen des Lernfaktors zeigen, daß ein monotoner Anstieg der Lernrate auch zu einer Vergrößerung der Fehlerquadratsumme führen kann. Der verwendete Algorithmus zur Adaption der Lernrate vergrößert diese nur so lange, wie sich die Summe der quadratischen Fehler am Ausgang des Neuronalen Netzes verkleinern. Sobald sich die Summe der Fehlerquadrate vergrößert wird die Lernrate um einen bestimmten Faktor verringert. Dadurch wird ein „Verlernen“ bereits trainierter Muster verhindert. Im Verlauf des „Sum-Squared-Error“ erkennt man dies an den plateauförmigen Abschnitten.

Auswertung der Mustererkennung an EMG-Signalen

Die extrahierten Peaks können nun mit Hilfe des trainierten zweischichtigen Neuronalen Netzes mit den Trainingsmustern verglichen werden. Durch die Auswahl der Trainingsmuster ist das Netzwerk in der Lage, auch „schnellere“ bzw. „langsamere“ Aktionspotentiale zu erkennen, d. h. es ist generalisierungsfähig. Dies ist vor allem bei veränderlichen Umgebungsbedingungen während der Messungen notwendig. Durch Schwankungen in der Temperatur, sowie bei den extrazellulären und intrazellulären Elektrolytkonzentrationen können sich die Frequenz und die Dauer der EMG-Aktionspotentiale verändern.

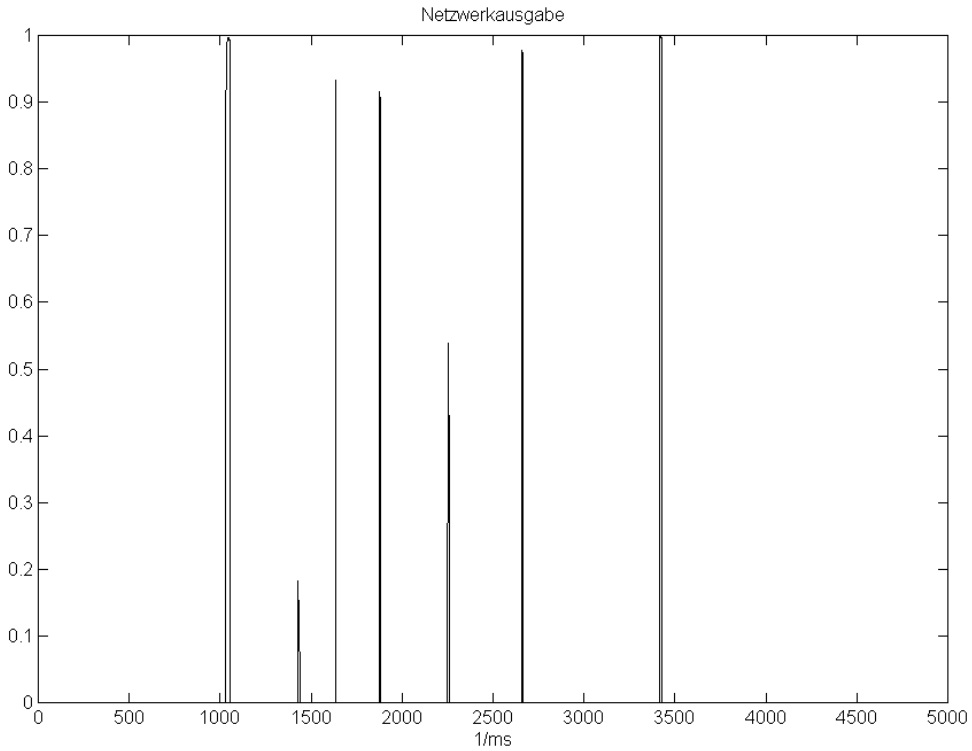


Abb. 3.20: Netzwerkausgabe

Die Antwort des Neuronalen Netzes auf das vorgegebene Signal (Abb. 3.11) zeigt deutlich die Stellen der Aktionspotentiale. Für die weitere Signalverarbeitung wird die Netzwerkantwort mit einer Schwellwertübertragungsfunktion auf die Werte Null und Eins reduziert.

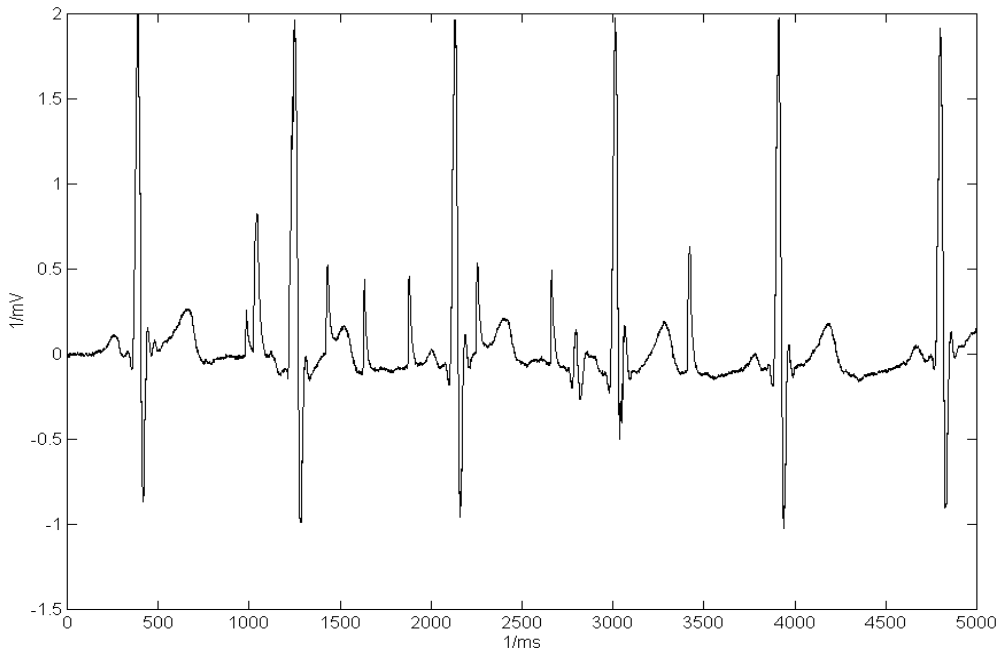


Abb. 3.21: Überlagertes Signal aus EKG und Aktionspotential-Bursts

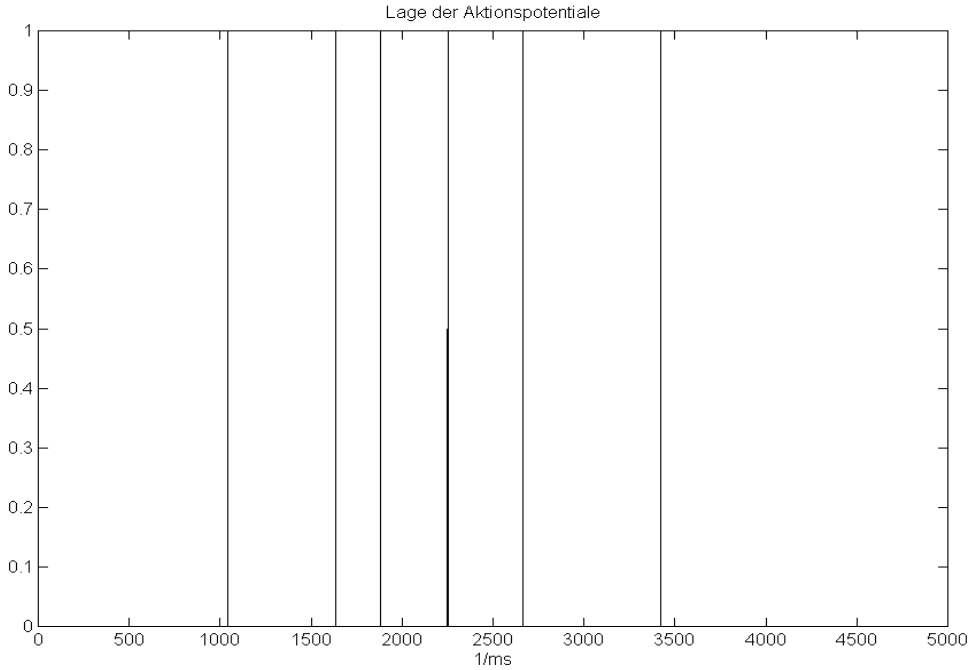


Abb. 3.22: Lage der Aktionspotentiale
Die Gegenüberstellung der Netzwerkantwort und des Mischsignals aus EKG und EMG zeigt, daß an den lokalisierten Stellen tatsächlich Aktionspotentiale auftreten. Ein Aktionspotential zum Zeitpunkt t = 1400 ms wurde nicht detektiert. Offensichtlich wurde dieses durch die Überlagerung mit der T-Welle eines EKG-Komplexes so stark verändert, daß es nicht von dem Neuronalen Netz erkannt werden konnte. Wie durch weitere Untersuchungen an anderen EMG-Signalen festgestellt wurde, tritt dieser Fall nur selten ein. Aktionspotentiale, die additiv mit der QRS-Strecke des EKG überlagert sind, können selbstverständlich nicht erkannt werden. Dies gelingt auch bei persönlicher visueller Auswertung der Signale nicht, da das Störsignal (das EKG) eine mehr als doppelt so große Amplitude wie die Aktionspotentiale des EMG besitzt.

Schlußbetrachtung

An den Beispielen der Sprecheridentifizierung und der Pulsfrequenzmessung (EKG-Erkennung) wurde gezeigt, daß mit Neuronalen Netzen vielfältige Aufgaben aus dem Bereich der Mustererkennung gelöst werden können. Die Leistungsfähigkeit des verwendeten Neuronale-Netze-Simulator von MATLAB® im Bezug auf die Arbeitsgeschwindigkeit und die Größe der Netzwerke wird hierbei hauptsächlich durch die zur Verfügung stehende Hardware begrenzt. Der Simulator unter MATLAB® enthält nur einige der derzeit bekannten Algorithmen zur Einstellung der Gewichte und Bias eines Neuronalen Netzes. Nicht in der Neuronale-Netze-Toolbox implementierte Trainingsalgorithmen können einfach und komfortabel als „M-Script-File“ programmiert werden.

Die Anwendung Neuronaler Netze für die Mustererkennung an EMG-Signalen stellt somit ein probates Werkzeug dar. Durch die Kaskadierung mehrerer Neuronaler Netze können auch komplexe Pattern-Recognition-Aufgaben bewältigt werden, welche durch konventionelle Signalverarbeitungsmethoden nur schwer lösbar sind. Andererseits ist der zeitliche Aufwand für den Aufbau und das Training eines Neuronalen Netzes sehr groß, da die Auswahl einer problemgerechten Netzwerktopologie und dem entsprechenden Trainingsalgorithmus eine umfassende empirische Phase voraussetzt. Gleichermaßen müssen die Pattern-Target-Paare (Trainingsmuster-Netzwerkausgabe-Paare) sorgfältig selektiert werden, um die gewünschte Erkennsicherheit zu erreichen. Ein weiterer wichtiger Aspekt ist, wie schon erwähnt, die Arbeitsgeschwindigkeit des trainierten Neuronalen Netzes und der zeitliche Aufwand für evtl. erforderliches nachträgliches Training des Netzwerkes. Diese Zeiten werden maßgeblich von der Anzahl und der Tiefe der Layerschichten, von der vorhandenen Hardware und dem Neuronale-Netze-Simulator bestimmt. Der in dieser Diplomarbeit eingesetzte Neuronale-Netze-Simulator von MATLAB® ist im Zusammenhang mit einem PC (INTEL® 80486 Prozessor und 8 MByte RAM) für die dargestellten Anwendungen nicht echtzeitfähig. Allerdings steht mit diesem Simulator eine sehr komfortable Entwicklungsumgebung zur Verfügung, deren Ergebnisse (die berechneten Gewichte und Bias) dann auch auf schnellen (Neuro-) Computern implementiert werden können.

Für weiterführende Untersuchungen zu diesem Thema wäre eine Kombination aus verschiedenen Signalverarbeitungsmethoden denkbar, welche dann zu einer kürzeren Verarbeitungszeit führen könnten. Aus diesem Grund wurden schon in dieser Diplomarbeit frequenzselektive Filter in Kombination mit Neuronalen Netzen eingesetzt. Zur Reduzierung, z.B. periodischer Störsignalanteile, welche das zu messende EMG-Signal überlagern, würde sich auch der Einsatz adaptiver Filter eignen.

Die Messung der Elektromyogramme des Kaninchendarmes zeigen die Funktionsfähigkeit der zur Zeit zur Verfügung stehenden Nadelsysteme und Verstärker. Um die Reproduzierbarkeit der Meßergebnisse zu verbessern, ist jedoch eine Optimierung des Meßaufbaus (siehe Kapitel 3.3) für nachfolgende Arbeiten anzustreben.

Anhand der durchgeführten Messungen konnten die Aktionspotentiale des Kaninchendarmes eindeutig klassifiziert und Korrelationen zur Applikation verschiedener Medikamente hergestellt werden. Die Reaktionen des Kaninchendarmes auf die verschiedenen Medikamente und die dabei gemessenen Aktionspotentiale entsprechen genau denen von Herrn Prof. Rüegg (vom physiologischen Institut der Universität Heidelberg) vorhergesagten. Dies ist auch eine Bestätigung für die Relevanz und die Richtigkeit der durchgeführten EMG-Messungen und den daraus gefolgerten Aussagen.

Durch den Vergleich der Kreuzkorrelationsanalyse (KKA) mit Neuronalen Netzen im Bezug auf die Lokalisierung der EMG Aktionspotentiale konnte gezeigt werden, daß die KKA für diese Aufgabe nicht geeignet ist. Im Gegensatz zur KKA können Neuronale Netze bei der Mustererkennung einen Vergleich mit mehreren Mustern durchführen und zwischen ähnlichen Mustern generalisieren (siehe Kapitel 2.5). Dadurch werden auch Aktionspotentiale im EMG erkannt, die sich in ihrer Form und Lauflänge von den Vergleichsmustern geringfügig unterscheiden.

Literaturverzeichnis

[1] Prof. K. Abel: Vorlesungsskriptum zur Vorlesung „Nachrichtentheorie“, 1993

[2] J. J. Achenbach: Analoge und digitale Filter und Systeme, B.I. Wissenschaftsverlag Mannheim, 1992

[3] Arns, Jochheim, Remschmidt: Neurologie und Psychiatrie für Krankenpflegeberufe, Thieme Verlag Stuttgart, 1983

[4] I. N. Bronstein / K. A. Semendjajew: Taschenbuch der Mathematik, B. G. Teubner, 1991

[5] Joseph Bücker: Anatomie und Physiologie, Georg Thieme Verlag, 1985

[6] Horst Czichos: HÜTTE, Springer-Verlag, 1991

[7] Howard Demuth / Mark Beale: Neural Network Toolbox for use with MATLAB®, Januar 1994

[8] J. A. Freemann / D. M. Skapura: Neural Networks, Addison-Wesley, 1992

[9] Dr. Ludwig Geiger: Überlastungsschäden beim Klettern, Odyssee-Alpinverlag Germering, 1988

[10] S. Hafner: Neuronale Netze in der Automatisierungstechnik, Oldenburg Verlag, 1994

[11] Monika Köhle: Neuronale Netze, Springer-Verlag, 1990

[12] Martini: Methoden der Signalverarbeitung, Franzis Verlag, 1987

[13] Math Works: MATLAB® Manual, Januar 1994

[14] Math Works: MATLAB® Reference Book, Januar 1994

[15] Openheim, Schafer: Zeitdiskrete Signalverarbeitung, Oldenburg-Verlag,1992

[16] Pöppl, Handels (Hrsg.): DAGM Symposium „Mustererkennung 1993“-Muster im Dienst der Gesundheit, Springer Verlag, 1993

[17] Jeroen Scheepe: Basic experimental studies on smooth muscle electromyography on the urinary bladder: Physiological basics and first experiences, Vrije Universiteit Brussel, 1994

[18] Schmidt Thews: Physiologie des Menschen, Springer Verlag Berlin Heidelberg, 1990

[19] E. D. Schmitter: Neuronale Netze, Hofacker Verlag, 1991

[20] W. Schröter / K. H. Lautenschläger: Chemie, Fakten und Gesetze, Buch- und Zeit- Verlagsgesellschaft, 1987

[21] Thews / Mutschler / Vampel: Anatomie, Physiologie, Pathophysiologie des Menschen, Wissenschaftliche Verlagsgesellschaft mbH Stuttgart

[22] Andreas Zell: Simulation Neuronaler Netze, Addison-Wesley, 1994

Listings

Erläuterungen zu den Listings und zu MATLAB®

Für die Ausarbeitung dieser Diplomarbeit wurde mit dem Programm MATLAB® gearbeitet. Diese Software wurde für mathematisch- technische Anwendungen entwickelt und bietet zusätzlich verschiedenste Visualisierungstechniken. Sie ist in der Programmiersprache C programmiert und ermöglicht die Verwendung der aufrufbaren Anwendungs-Programmier-Schnittstelle (API), die Verbindungen zu anderen Anwendungen und Sprachen unterstützt. Ein weiteres wichtiges Feature ist, die Option MEX-Files zu erstellen (in C oder Fortran programmiert), welche dynamisch während der Laufzeit in MATLAB® eingebunden werden können. Viele MATLAB®-Funktionen und Toolbox-Funktionen sind als M-Dateien implementiert. Die offene Struktur erlaubt die Einsicht in die verwendeten Algorithmen. Dadurch können auf einfache Art Modifikationen der vorhandenen MATLAB®-Funktionen durchgeführt werden.

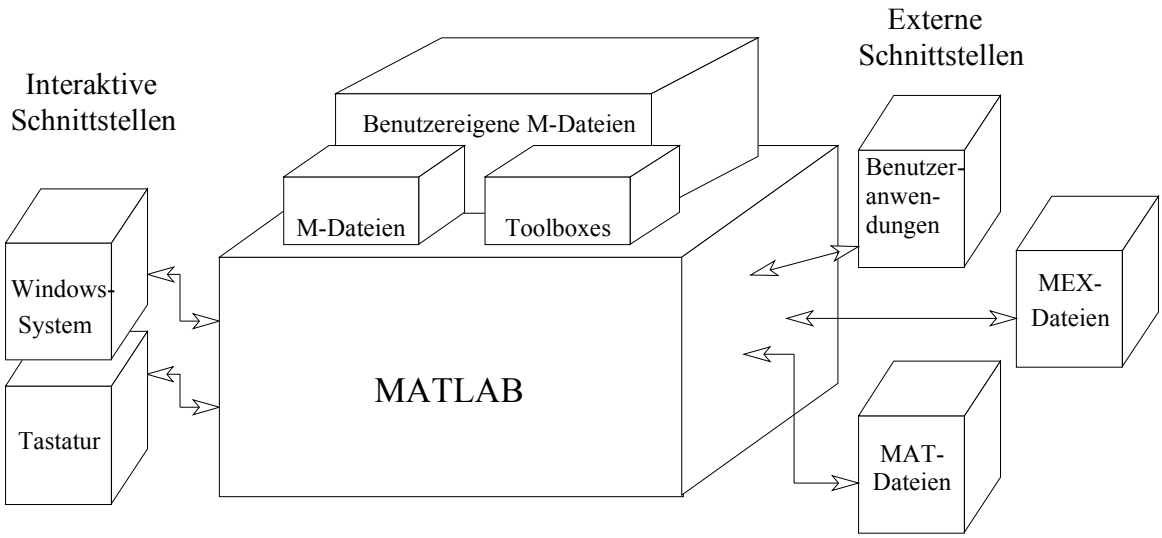


Abb. 5.1: Schematische Darstellung der offenen Architektur von MATLAB®

Im Verlauf der Diplomarbeit wurden folgende Toolboxes eingesetzt:
- Neuronale Netze
- Signalverarbeitung
- Bildverarbeitung
- Optimierung
- Systemidentifikation

Alle programmierten M-Dateien sind unter MATLAB® for WINDOWS Version 4.2c lauffähig. Die Syntax entspricht weitgehend der unter der Programmiersprache C üblichen Syntax (mit geringen Modifikationen). Die Ausführung der Programme (Script-Files und Funktionen) erfolgt in einem interpreter-ähnlichen Modus, wodurch eine längere Programmlaufzeit im Vergleich zu compilierten Programmen zustande kommt. Um die Arbeitsgeschwindigkeit zu steigern, besteht die Option, wie oben bereits erwähnt, sogenannte „MEX"-Files zu erstellen, welche dann in compilierter Form eine Reduzierung der Laufzeit ermöglichen.
Zum Starten der Script-Files wird der Dateiname im MATLAB® - Fenster eingegeben und mit RETURN bestätigt. Bei der Ausführung der FUNCTION-Unterprogramme müssen hier noch zusätzlich die entsprechenden Variablen zur Wertübergabe angegeben werden.

Anmerkungen zur Neuronalen-Netze-Toolbox von MATLAB®

In der Neuronalen-Netze-Toolbox sind einige wichtige Lernalgorithmen bzw. Netzwerktopologien implementiert und in dem mitgelieferten Handbuch sehr gut dokumentiert. Durch die aufgeführten Beispiele wird der Einstieg in die Anwendung der Toolbox erheblich erleichtert. Für weitere Arbeiten im Zusammenhang mit Neuronalen Netzen wäre jedoch eine Erweiterung der implementierten Netzwerktopologien und Trainingsalgorithmen wünschenswert. Die folgende Aufzählung soll hierzu einige Anregungen geben.

Aufzählung einiger Algorithmen bzw. Netzwerktopologien, die bisher noch nicht in der Neuronalen-Netze-Toolbox von MATLAB® implementiert sind:

1. Quickprop
2. Resilient Propagation
3. Backpercolation
4. Jordan Netze
5. Backpropagation Through Time
6. Real Time Recurrent Learning
7. Recurrent Backpropagation
8. Time Dependent Recurrent Backprop.
9. Cascade Correlation
10. Recurrent Cascade Correlation
11. Lernende Vektorquantisierung (LVQ) 2.1
12. LVQ 3
13. Optimized LVQ 1
14. Counterpropagation
15. Bidirektionaler Assoziativspeicher (BAM)
16. Propabilistische Neuronale Netze (PNN)
17. Adaptive Resonance Theory (ART) 1
18. ART 2
19 ART 3
20. Fuzzy ART
21. Neocognitron
22. Time Delay Netze (TDNN)
23. Multi State TDNN
24. Adaptive logische Neuronale Netze

Eine Minimierung von Neuronalen Netzen kann häufig auch verbesserte Ergebnisse liefern und zusätzlich die Bearbeitungszeit verkürzen. Hierfür ist bereits die bekannte Weight-Decay-Technik in der Neuronale Netze-Toolbox-enthalten. Die Verfahren Optimal-Brain-Damage, Optimal-Brain-Surgeon, Unit-Pruring (Reduzierung verdeckter Neuronen) und Input-Unit Pruring könnten hier zur Optimierung der Resultate führen.

Quellen: [8], [10], [19], [22]

Script_File: „LOWPASS.M“

```
% Tiefpassfilter für allgemeine Anwendungen
% ================================
%      g(t) = h(t) * s(t)
% bzw  G(f) = H(f) . S(f)
%      g(t) = F(^-1)(G(f))

% gt    = Ausgangssignal im Zeitbereich
% gt2   = Ausgangssignal im Zeitbereich mit Phasenkompensation (des FIR Filters)
% st    = Eingangssignal im Zeitbereich
% fabt  = Abtastfrequenz
% fgr   = Grenzfrequenz des Filters
%=========================================================
% Es ist darauf zu achten, daß    length(gt)>=2^n    ist, damit die Dauer für die
% Berechnung der FFT möglichst kurz gehalten wird.
%=========================================================
% Stand : 20.09.1994 / 07.11.1994
% Alexander Scheib

function [gt,gt2] = lowpass (st,fabt,fgr);


[m,n]=size(st);
if m>1
  st=st';
end

% mittelwertfreies Eingangssignal bilden:
  st=st-mean(st);
  l=length(st);
  lst=l;
  x=1;

% optimales N fuer FFT finden:
  n=round(log2(l)-0.5);
  N=2^n;


% FFT des Eingangssignal bilden:
  x=round(l/N-0.5);
  Sf=fft(st(1:N-1),N);
    %  for i=2:1:x
    %  sf=fft(st((i-1)*N+1:i*N) , N);
    % Sf=[Sf sf];
    % end


% Frequenzlinienabstand f0
  f0=(1/N)*fabt;

% Zeitsignal st plotten
  bin_number=1:l;
  t=(bin_number-1)*(1/fabt);
  plot(t,abs(st),'-'),title('Betrag des ungefilterten Zeitsignals s(t)'),
  xlabel('1/s')
  pause;

% Plot des Amplitudendichtespektrum Sf (f=Frequenz   a=Amplitudenvektor)
```

```
  a =abs(Sf);
  bin_number=1:length(Sf);
  f=(bin_number-1)*f0;
  plot(f,a,'-'),title('Amplitudendichtespektrum  s(f)'),
  xlabel('f/Hz')
  pause;

% Das Filter
% =======

  Ta=1/fabt;
  Wn=fgr/(fabt/2);   % Normierte cut-off-frequency
                     % Wn=fgr/(fa/2) sein !!!!?

% Filter mit halber Punktzahl der FFT bilden und periodizieren (wegen IFFT)
% NN=round(N/2);

% Mit Bartlett Fenster
  B= fir1(NN-1,Wn,boxcar(NN));
            % Koeffizienten des FIR-Filter jetzt in Vektor B
  A=1;          % Rekursiver Teil
  [h,w]=freqz(B,A,(NN));
            % w = NN-Punkt Frequenzvektor (anstatt f kontinuierlich)
            % h = NN-Punkt Uebertragungsfunktion H(nf)
  mag=abs(h);
  phase=unwrap(angle(h));
  bin_number=1:length(w);
  f=(bin_number-1)*(fabt/2)/NN;       % ist gleich f0/2
  ph=phase*180/pi ;

  plot (f,mag),title('|H(f)| des Filters  (Bartlett)'),
  xlabel('f/Hz')
  pause;
   plot (f,ph,'+'),title('Phase (Bartlett)');      pause

% Ausmultiplizieren (Filterung) und Ruecktransformation in den Zeitbereich
  [m,n] =size(h);
  h=reshape(h,n,m);
% Filter periodizieren
  lh=length(h);
  for i=1:lh
    hh(lh-i+1)=h(i);    % conj(h(i));
  end
  h=[h hh];
  hi=h;
  for i=1:x-1
   h=[h hi];
  end

  Gf = Sf.* h;
   absGf=abs(Gf);

% Kompensation der Filter-Phase,
% so daß der Realteil von ifft(Gf2) gleich dem Betrag von ifft(Gf2) ist
% ==================================================
  Gf2= Sf.* h.*conj(h)./abs(h);

% Plot des Amplitudenspektrum des Ausgangssignales G(f)
  bin_number=1:length(Gf);
  f=(bin_number-1)*(fabt/1)/N;
  % Plot des Amplitudendichtespektrum G(f) (f=Frequenz   a=Amplitudenvektor)
```

```
  plot(f,absGf,'-'),title('Amplitudendichtespektrum |G(f)|'),
  xlabel('f/Hz')
  pause;

% Ruecktransformation des gefilterten Signals
  gt=ifft(Gf(1:N-1),N);
  gt1=gt;
 gt=abs(gt);
 gt2=real(ifft(Gf2(1:N-1),N));

% Betrag des gefilterten Zeitsignals plotten
  bin_number=1:length(gt);
  t=(bin_number-1)*(1/fabt);
% Plot des Amplitudendichtespektrum (f=Frequenz   a=Amplitudenvektor)
% plot(t,gt,'-',t,gt2),title('Betrag des gefilterten Zeitsig. g(t) mit und ohne
%     Phasenkompensation'),
xlabel('t/s')
  pause;

end ;    % End of File
```

Script_File: „HIGHPASS.M“

```
% Hochpassfilter für allgemeine Anwendungen
% ================================
%          g(t) = h(t) * s(t)
% bzw  G(f) = H(f) . S(f)
%          g(t) = F(^-1)(G(f))

% gt    = Ausgangssignal im Zeitbereich
% gt2   = Ausgangssignal im Zeitbereich mit Phasenkompensation (des FIR Filters)
% st    = Eingangssignal im Zeitbereich
% fabt  = Abtastfrequenz
% fgr   = Grenzfrequenz des Filters
% ==========================================================
% Es ist darauf zu achten, daß    length(gt)>=2^n    ist !!!   (aus Zeitgruenden)
% ==========================================================
% Stand :20.09.1994 / 07.11.1994
% Alexander Scheib

function [gt,gt2] = highpass (st,fabt,fgr);

[m,n]=size(st);
if m>1
  st=st';
end

% Mittelwertfreies Eingangssignal bilden
  st=st-mean(st);
  l=length(st);
  lst=l;
  x=1;

% optimales N fuer FFT finden
```

```
  n=round(log2(l)-0.5);
  N=2^n;


% FFT des Eingangssignal bilden
  x=round(l/N-0.5);
  Sf=fft(st(1:N),N);
    %  for i=2:1:x
    %  sf=fft(st((i-1)*N+1:i*N) , N);
    % Sf=[Sf sf];
    % end

% Frequenzlinienabstand f0
  f0=(1/N)*fabt;


% Zeitsignal st plotten
  bin_number=1:l;
  t=(bin_number-1)*(1/fabt);
  plot(t,abs(st),'-'),title('Betrag des ungefilterten Zeitsignals s(t)'),
  xlabel('t/s')
  pause;

% Plot des Amplitudendichtespektrum Sf (f=Frequenz   a=Amplitudenvektor)
  a =abs(Sf);
  bin_number=1:length(Sf);
  f=(bin_number-1)*f0;
  plot(f,a,'-'),title('Amplitudendichtespektrum  s(f)'),xlabel('f/Hz')
  pause;

% Das Filter
% =======

  Ta=1/fabt;
  Wn=fgr/(fabt/2);   % Normierte cut-off-frequency
                 % Wn=fgr/(fa/2) sein !!!!?

% Filter mit halber Punktzahl der FFT bilden und periodizieren (wegen IFFT)
  NN=round(N/2);

% Mit Bartlett Fenster
   B= fir1(NN,Wn,'high',bartlett(NN+1));
              % Koeffizienten des FIR-Filter jetzt in Vektor B
  A=1;            % Rekursiver Teil
  [h,w]=freqz(B,A,(NN));
              % w = NN-Punkt Frequenzvektor (anstatt f kontinuierlich)
              % h = NN-Punkt Uebertragungsfunktion H(nf)
  mag=abs(h);
  % phase=unwrap(angle(h));
  bin_number=1:length(w);
  f=(bin_number-1)*(fabt/2)/NN;        % ist gleich f0/2
  % ph=phase*180/pi ;

  plot (f,mag),title('|H(f)| des Filters  (Bartlett)'),xlabel('f/Hz')  ;    pause
  % plot (f,ph,'+'),title('Phase (Bartlett)');       pause


% Ausmultiplizieren (Filterung) und Ruecktransformation in den Zeitbereich
  [m,n] =size(h);
  h=reshape(h,n,m);
```

```
% Filter periodizieren
  lh=length(h);
  for i=1:lh
    hh(lh-i+1)=h(i);    % conj(h(i));
  end
  h=[h hh];
  hi=h;
  for i=1:x-1
   h=[h hi];
  end

  Gf = Sf.* h;
  absGf=abs(Gf);

% Kompensation der Filter-Phase,
% so daß der Realteil von ifft(Gf2) gleich dem Betrag von ifft(Gf2) ist
% ==================================================
  Gf2= Sf.* h.*conj(h)./abs(h);


% Plot des Amplitudenspektrum des Ausgangssignales G(f)
  bin_number=1:length(Gf);
  f=(bin_number-1)*(fabt/1)/N;
  % Plot des Amplitudendichtespektrum G(f) (f=Frequenz   a=Amplitudenvektor)
  plot(f,absGf,'-'),title('Amplitudendichtespektrum |G(f)|'),xlabel('f/Hz');    pause


% Ruecktransformation des gefilterten Signals
  gt=ifft(Gf(1:N),N);
  gt=abs(gt);
  gt2=real(ifft(Gf2(1:N-1),N));

% Betrag des gefilterten Zeitsignals plotten
  bin_number=1:length(gt);
  t=(bin_number-1)*(1/fabt);
% Plot des Amplitudendichtespektrum (f=Frequenz   a=Amplitudenvektor)
%  plot(t,gt,'-'),title('Betrag des gefilterten (Ausgangs-) Zeitsignals g(t)'),
 plot(t,gt,'-',t,gt2),title('Betrag des gefilterten Zeitsig. g(t) mit und ohne Phasenkompensation'),
  xlabel('t/s')
  pause;

end ;    % End of File
```

Script_File: „BANDSTOP.M“

```
%  Bandsperre für allgemeine Anwendungen
% ==============================
%        g(t) = h(t) * s(t)
% bzw  G(f) = H(f) . S(f)
%         g(t) = F(^-1)(G(f))

% gt    = Ausgangssignal im Zeitbereich
% gt2   = Ausgangssignal im Zeitbereich mit Phasenkompensation (des FIR Filters)
% st    = Eingangssignal im Zeitbereich
% fabt  = Abtastfrequenz
% fgru  = Untere Grenzfrequenz der Bandsperre
```

```
% fgro  = Obere Grenzfrequenz der Bandsperre
% ==========================================================
% Es ist darauf zu achten, daß    length(gt)>=2^n    ist !!!   (aus Zeitgruenden)
% ==========================================================

% Stand :20.09.1994  / 07.11.1994
% Alexander Scheib

function [gt,gt2] = bandstop (st,fabt,fgru,fgro);

[m,n]=size(st);
if m>1
  st=st';
end

% Mittelwertfreies Eingangssignal bilden
  st=st-mean(st);
  l=length(st);
  lst=l;
  x=1;

% optimales N fuer FFT finden
  n=round(log2(l)-0.5);
  N=2^n;

% FFT des Eingangssignal bilden
  x=round(l/N-0.5);
  Sf=fft(st(1:N),N);
    %  for i=2:1:x
    %  sf=fft(st((i-1)*N+1:i*N) , N);
    % Sf=[Sf sf];
    % end

% Frequenzlinienabstand f0
  f0=(1/N)*fabt;

% Zeitsignal st plotten
  bin_number=1:l;
  t=(bin_number-1)*(1/fabt);
  plot(t,abs(st),'-'),title('Betrag des ungefilterten Zeitsignals s(t)')
  pause;


% Plot des Amplitudendichtespektrum Sf (f=Frequenz   a=Amplitudenvektor)
  a =abs(Sf);
  bin_number=1:length(Sf);
  f=(bin_number-1)*f0;
  plot(f,a,'-'),title('Amplitudendichtespektrum  s(f)')
  pause;
% Das Filter
% =======

  Ta=1/fabt;
  Wn=[(fgru/(fabt/2)) (fgro/(fabt/2))];   % Normierte cut-off-frequency
                % Wn=fgr/(fa/2) sein !!!!?


% Filter mit halber Punktzahl der FFT bilden und periodizieren (wegen IFFT)
  NN=round(N/2);

% Mit Bartlett Fenster
```

```
 B= fir1(NN,Wn,'stop',bartlett(NN+1));
            % Koeffizienten des FIR-Filter jetzt in Vektor B
 A=1;          % Rekursiver Teil
 [h,w]=freqz(B,A,(NN));
            % w = NN-Punkt Frequenzvektor (anstatt f kontinuierlich)
            % h = NN-Punkt Uebertragungsfunktion H(nf)
 mag=abs(h);
 % phase=unwrap(angle(h));
 bin_number=1:length(w);
 f=(bin_number-1)*(fabt/2)/NN;       % ist gleich f0/2
 % ph=phase*180/pi ;

 plot (f,mag),title('|H(f)| des Filters  (Bartlett)')  ;    pause
 % plot (f,ph,'+'),title('Phase (Bartlett)');       pause


% Ausmultiplizieren (Filterung) und Ruecktransformation in den Zeitbereich
 [m,n] =size(h);
 h=reshape(h,n,m);


% Filter periodizieren
 lh=length(h);
 for i=1:lh
   hh(lh-i+1)=h(i);    % conj(h(i));
 end
 h=[h hh];
 hi=h;
 for i=1:x-1
  h=[h hi];
 end

 Gf = Sf.* h;
 absGf=abs(Gf);

% Kompensation der Filter-Phase,
% so daß der Realteil von ifft(Gf2) gleich dem Betrag von ifft(Gf2) ist
% ==================================================
 Gf2= Sf.* h.*conj(h)./abs(h);

% Plot des Amplitudenspektrum des Ausgangssignales G(f)
 bin_number=1:length(Gf);
 f=(bin_number-1)*(fabt/1)/N;
 % Plot des Amplitudendichtespektrum G(f) (f=Frequenz   a=Amplitudenvektor)
 plot(f,absGf,'-'),title('Amplitudendichtespektrum |G(f)|');    pause


% Ruecktransformation des gefilterten Signals
 gt=ifft(Gf(1:N),N);
 gt=abs(gt);
 gt2=real(ifft(Gf2(1:N-1),N));

% Betrag des gefilterten Zeitsignals plotten
 bin_number=1:length(gt);
 t=(bin_number-1)*(1/fabt);
 % Plot des Amplitudendichtespektrum (f=Frequenz   a=Amplitudenvektor)
%  plot(t,gt,'-'),title('Betrag des gefilterten (Ausgangs-) Zeitsignals g(t)')
plot(t,gt,'-',t,gt2),title('Betrag des gefilterten Zeitsig. g(t) mit und ohne Phasenkompensation'),
 xlabel('t/s')
 pause;
```

```
end ;    % End of File
```

Script_File: „BANDPASS.M“

```
% Bandpassfilter für allgemeine Anwendungen
% ==================================
%         g(t) = h(t) * s(t)
% bzw  G(f) = H(f) . S(f)
%         g(t) = F(^-1)(G(f))

% gt    = Ausgangssignal im Zeitbereich
% gt2   = Ausgangssignal im Zeitbereich mit Phasenkompensation (des FIR Filters)
% st    = Eingangssignal im Zeitbereich
% fabt  = Abtastfrequenz
% fgru  = Untere Grenzfrequenz des Bandpasses
% fgro  = Obere Grenzfrequenz des Bandpasses

% ==========================================================
% Es ist darauf zu achten, daß    length(gt)>=2^n    ist !!!   (aus Zeitgruenden)
% ==========================================================
% Stand :20.09.1994  / 07.11.1994
% Alexander Scheib

function [gt,gt2] = bandpass (st,fabt,fgru,fgro);

[m,n]=size(st);
if m>1
  st=st';
end

% Mittelwertfreies Eingangssignal bilden
  st=st-mean(st);
  l=length(st);
  lst=l;
  x=1;

% optimales N fuer FFT finden
  n=round(log2(l)-0.5);
  N=2^n;

% FFT des Eingangssignal bilden
  x=round(l/N-0.5);
  Sf=fft(st(1:N),N);
    %  for i=2:1:x
    %  sf=fft(st((i-1)*N+1:i*N) , N);
    % Sf=[Sf sf];
    % end

% Frequenzlinienabstand f0
  f0=(1/N)*fabt;
% Zeitsignal st plotten
  bin_number=1:l;
  t=(bin_number-1)*(1/fabt);
  plot(t,abs(st),'-'),title('Betrag des ungefilterten Zeitsignals s(t)')
  pause;

% Plot des Amplitudendichtespektrum Sf (f=Frequenz   a=Amplitudenvektor)
```

```
  a =abs(Sf);
  bin_number=1:length(Sf);
  f=(bin_number-1)*f0;
  plot(f,a,'-'),title('Amplitudendichtespektrum  s(f)')
  pause;



% Das Filter
% =======

  Ta=1/fabt;
  Wn=[(fgru/(fabt/2)) (fgro/(fabt/2))];   % Normierte cut-off-frequency
                % Wn=fgr/(fa/2) sein !!!!?

% Filter mit halber Punktzahl der FFT bilden und periodizieren (wegen IFFT)
  NN=round(N/2);

% Mit Bartlett Fenster
   B= fir1(NN,Wn,bartlett(NN+1));
              % Koeffizienten des FIR-Filter jetzt in Vektor B
  A=1;            % Rekursiver Teil
  [h,w]=freqz(B,A,(NN));
              % w = NN-Punkt Frequenzvektor (anstatt f kontinuierlich)
              % h = NN-Punkt Uebertragungsfunktion H(nf)
  mag=abs(h);
  % phase=unwrap(angle(h));
  bin_number=1:length(w);
  f=(bin_number-1)*(fabt/2)/NN;        % ist gleich f0/2
  % ph=phase*180/pi ;

  plot (f,mag),title('|H(f)| des Filters  (Bartlett)')  ;    pause
  % plot (f,ph,'+'),title('Phase (Bartlett)');       pause

% Ausmultiplizieren (Filterung) und Ruecktransformation in den Zeitbereich
  [m,n] =size(h);
  h=reshape(h,n,m);

% Filter periodizieren
  lh=length(h);
  for i=1:lh
    hh(lh-i+1)=h(i);    % conj(h(i));
  end
  h=[h hh];
  hi=h;
  for i=1:x-1
   h=[h hi];
  end

  Gf = Sf.* h;
  absGf=abs(Gf);

% Kompensation der Filter-Phase,
% so daß der Realteil von ifft(Gf2) gleich dem Betrag von ifft(Gf2) ist
% ==================================================
  Gf2= Sf.* h.*conj(h)./abs(h);

% Plot des Amplitudenspektrum des Ausgangssignales G(f)
  bin_number=1:length(Gf);
  f=(bin_number-1)*(fabt/1)/N;
```

```
  % Plot des Amplitudendichtespektrum G(f) (f=Frequenz   a=Amplitudenvektor)
  plot(f,absGf,'-'),title('Amplitudendichtespektrum |G(f)|');    pause
% Ruecktransformation des gefilterten Signals
  gt=ifft(Gf(1:N),N);
  gt=abs(gt);
 gt2=real(ifft(Gf2(1:N-1),N));

% Betrag des gefilterten Zeitsignals plotten
  bin_number=1:length(gt);
  t=(bin_number-1)*(1/fabt);
  % Plot des Amplitudendichtespektrum (f=Frequenz   a=Amplitudenvektor)
%  plot(t,gt,'-'),title('Betrag des gefilterten (Ausgangs-) Zeitsignals g(t)')
plot(t,gt,'-',t,gt2),title('Betrag des gefilterten Zeitsig. g(t) mit und ohne Phasenkompensation'),
  xlabel('t/s')
 pause;

end ;    % End of File
```

Script_File: „PERCEPT.M“

```
% Beispiel zur Anwendung eines Perceptron
% ==============================

% ( Das Beispiel entspricht der MATLAB 4.0 NN-Demo: EXAMPLE Percept2 )
% Alexander Scheib
% Stand: 20.10.94

% Das Perceptron soll die zehn Eingangsvektoren in der Matrix P (spaltenweise)
% klassifizieren. Nach der Trainingsphase kann jeder vorgelegte Vektor V klassifiziert
% werden.
% A = hardlim(W*V,B)
% A = [0;0] bedeutet Klasse 'o'
% A = [0;1] bedeutet Klasse '*'
% A = [1;0] bedeutet Klasse '+'
% A = [1;1] bedeutet Klasse 'x'

clf reset

% Mustervektoren
% ===========

P = [+0.1 +0.7 +0.8 +0.8 +1.0 +0.3 +0.0 -0.3 -0.5 -1.5; ...
    +1.2 +1.8 +1.6 +0.6 +0.8 +0.5 +0.2 +0.8 -1.5 -1.3];

% Zu assoziierende Vektoren
%  ==================

T = [1 1 1 0 0 1 1 1 0 0;
    0 0 0 0 0 1 1 1 1 1];

% Plot der Mustervektoren
% =================

V = [-2 2 -2 2];
plotpv1(P,T,V);
axis('equal')
title('Input-Vektoren')
xlabel('Input 1')
ylabel('Input 2')
pause

% Netzwerkstruktur festlegen
% ===================

% Input-Vektor der Größe R, Anzahl der Neuronen = S

[R,Q] = size(P); [S,Q] = size(T);
  [W0,B0] = rands(S,R);

% Plot der Klassifikation mit initialisierten Gewichten
% ====================================
 hold on
% plotpc(W0,B0,'--');

% Netzwerktraining
% ============
W = W0;
B = B0;
```

```
flops(0);

% Presentationsphase
A = hardlim(W*P,B);
for epoch=1:max_epoch

% Vergleichsphase
  if all(A == T), epoch=epoch-1; break, end

% Lernphase
  [dW,dB] = learnp(P,A,T);
  W = W + dW; B = B + dB;

% Presentationsphase
  A = hardlim(W*P,B);

totalflops = flops;
end

% Plot der Klassifikation
% ================
hold on
plotpc1(W,B);
hold off
pause

% Zusammenfassung der Ergebnisse
% ========================

fprintf('Anzahl der Trainingsdurchläufe %.0f epochs.\n',epoch)

fprintf('Benötigte "floating point operations" %.0f flops.\n',totalflops);

fprintf('Durchschnittliche %.0f flops/epoch.\n',round(totalflops/epoch));

end ;    % End of File
```

Script_File: „START-SP.M“

```
% Sprecheridentifikation
% ================

% Startup-Script-File mit dem nacheinander für alle Sprecher ein Kohonen-Netzwerk
% trainiert wird mit anschließendem Test des Neuronalen Netzes:

% Alexander Scheib
% Stand: 02.12.94


sprach11;  spr11_te; sprach12;  spr12_te;

sprach1a;  spr1a_te;  sprach1b;  spr1b_te;

sprach21;  spr21_te;  sprach22;  spr22_te;

sprach2a;  spr2a_te;  sprach2b;  spr2b_te;

end ;    % End of File
```

Die beiden folgenden Script-File’s zeigen exemplarisch die Vorgehensweise für das Netzwerktraining anhand des Wortes „EINS“ und den dazugehörenden Testlauf. Für alle anderen Wörter/Phoneme wurden entsprechende Script-Files programmiert.

Scrip-File: „SPRACH11.M“

```
% Sprecheridentifikation (Trainieren des Netzwerkes)
% ====================================
% Lernregel: Hebb/kohonen

% Alexander Scheib
% Stand: 20.10.1994 / 02.12.94

clf reset
pausetime = 0.1;

% Laden der Muster
%=============
% (Laden der Leistungsdichtespektren verschiedener Sprecher)

load b:\Sfgm2;
Lfgm1x=Sfgm2(:,1:6);


load a:\Sfkv2;
Lfkv1x=Sfkv2(:,1:6);

P1=[Lfgm1x(:,1) Lfkv1x(:,1) Lfgm1x(:,2) Lfkv1x(:,2) Lfgm1x(:,3) Lfkv1x(:,3)];
P1=[P1 Lfgm1x(:,1) Lfgm1x(:,4) Lfgm1x(:,2) Lfgm1x(:,5) Lfgm1x(:,3) Lfgm1x(:,6)];
```

```
% Begrenzung der 4096 Byte langen Datensätze
P=normc(P1(1:4096,:));
figure(1);
subplot(2,2,1)
plot (P1(:,1));
subplot(2,2,2)
plot (P(:,1));

subplot(1,1,1);
pause2(0.5);

T=[1 0 1 0 1 0 1 0 1 0 1 0];

clf;

% Netzwerkstrukture festlegen
% ====================

[R,Q] = size(P); [S,Q] = size(T);

% Bias und Gewichte initialisieren
% =======================

W0 = zeros(S,R);
B0 = -0.95 * ones(S,1);

% Plot der Ausgangsgewichte
% ===================

disp('Taste drücken, um Training zu starten');
%pause, disp('');

figure(1)
  subplot(2,2,1);
  plot(W0(1,:));
  title('Ortogonale Gewichte W(i,j)');


% Training
% ======

disp_freq = 10;
max_epoch = 50;



lr = 0.01;
dr=lr/3

W = W0;
B = B0;

flops(0);
for muster_anz=1:2
 for epoch=1:max_epoch
  for q=1:Q

    % Präsentationsphase
    %==============
    A = T(:,q);
```

```
    % LERNPHASE
    i = find(A == 1);
    dW = learnk(W,P(:,q),i,lr);
   W = W + dW;
    end

  % Ausgabe des Lernerfolgs
  % ==================
  temp = flops;
  if rem(epoch,disp_freq) == 0
    pause2(pausetime);
    subplot(2,2,1);
    plot(W(1,:));
    title('Weights W(i,j)');
    end
  flops(temp);
  end        % end of max_epoch
end

totalflops = flops;

% PLOT

subplot(2,2,1);
plot(W(1,:));
title('Final Weights W(i,j)');
pause

% Ergebnise anzeigen
% ==============
save sprech1 P1;
save lab_w W;
save lab_b B;


fprintf('Anzahl der Durchläufe (Epochen) %.0f pro Epoche\n',epoch)
fprintf('Gesamtzahl der flops %.0f (floating point operations)\n',totalflops);
fprintf('Durchschnittliche Anzahl der flops/Epoche %.0f \n',round(totalflops/epoch));
disp('');
disp('')
disp('Ergebnis der Lernphase (A = hardlim(W*P,B))')
% A = hardlim(W*P,B)
bl=bl.*0-0.6;
A = hardlim(normc(purelin(W*P1(1:3000,:))),bl)

end ;    % End of File
```

Script-File: „SPR-TE11.M“

```
% Sprecheridentifikation
% ================
% Script-File zum Testen des trainierten Neuronalen Netzes

% Alexander Scheib
% Stand: 02.12.94
```

```
load b:\lfgm3;
load b:\lfkv3;
load b:\lfpb3;
load b:\lfsz3;
load b:\lftp3;

load b:\sfgm2;
for i=1:3
A(i)= purelin(W*normc(Sfgm2(1:3000,0+i)));
end;

for i=1:3
A(i+3)= purelin(W*normc(Lfgm46(1:3000,0+i)));
end;

for i=1:3
A(i+6)= purelin(W*normc(Lfkv46(1:3000,0+i)));
end;

for i=1:3
A(i+9)= purelin(W*normc(Lfpb46(1:3000,0+i)));
end;

for i=1:3
A(i+12)= purelin(W*normc(Lfsz46(1:3000,0+i)));
end;

for i=1:3
A(i+15)= purelin(W*normc(Lftp46(1:3000,0+i)));
end;

mx=max(A);A=A./mx;
y=ones(1,18).*(-1);
 for i=1:6
  if A(i) > 0.5
   y(i)=1;
  end;
 end;


for i=7:18
 if A(i) < 0.5
  y(i)=1;
 end;
end;

figure(1);
clf;
stem(A(1:6),y(1:6),'-');
hold on;
stem(A(7:18),y(7:18),':');
hold off;
title('Sprecher Nr. 1 , Zahl: "1"');
xlabel('Ausgabe des Neuronalen Netzwerkes');
 ylabel('falsch erkannt        /        richtig erkannt');
text(0.02,-1.31,'Andere Sprecher = --------');
text(0.02,-1.19,'Sprecher Nr. 1 =');
text(0.27,-1.12,'_____');
axis([0 1 -1.5 1.5]);
xp=0;
```

```
for i=1:18
 if y(i)==1
  xp=xp+1;
 end;

end;
xp=xp/18*100;
text1=('Es wurden      % richtig klassifiziert');
einf=num2str(xp);
text1(11:length(einf)+10)=einf;
text(0.2,1.4,text1);
pause;

% Netzwerkausgabe für die Verifikation des Sprechers Nr.:1 anhand des Wortes „EINS“
% (und zu erkennende „Falschbeispiele“) in Matrix AM Spalte eins speichern
AM(:,1)=A';
YM(:,1)=y';
```

Script-File: „SPR_AUS.M“

```
% Script-File zur Auswertung der Sprechererkennung
% ====================================

% Laden der Matrix AM (Netzwerkausgabe)
load am;
% Laden der Matrix YM (Bewertete Netzwerkausgabe -- richtig erkannt =+1
    falsch erkant =-1)
load ym;

[n,m]=size(YM);
YM(19,:)=zeros(1,m);
AM(19,:)=zeros(1,m);

i1=1;
i2=1;
buchs='12ab';
spr_nr='12';

for i=1:m
figure(i);
clf;
stem(AM(1:6,i),YM(1:6,i),'-');
hold on;
stem(AM(7:19,i),YM(7:19,i),':');
hold off;
text2=('Sprecher Nr.   =');
text2(14)=spr_nr(i2);
text(0.02,-1.19,text2);
text1=('Sprecher Nr.   , Zahl: " "');
text1(14)=spr_nr(i2);
text1(25)=buchs(i1);
title(text1);
i1=i1+1;
if i1>4
 i1=1;
 i2=2;
end;
```

```
xlabel('Ausgabe des Neuronalen Netzwerkes');
 ylabel('falsch klassifiziert     /     richtig klassifiziert');
text(0.02,-1.31,'Andere Sprecher = --------');
text(0.24,-1.12,'_____');
axis([0 1 -1.5 1.5]);

xp=0;
for j=1:18
 if YM(j,i)==1
  xp=xp+1;
 end;
end;
xp=xp/18*100;

text1=('Es wurden      % richtig klassifiziert');
einf=num2str(xp);
text1(11:length(einf)+10)=einf;
text(0.2,1.4,text1);
pause;
end;

end ;    % End of File
```

Script-File: „BP-EKG3.M“

```
% Programm zum Training eines dreischichtigen Neuronalen Netzes zur EKG-Erkennung
% ================================================================

% Alexander Scheib
% Stand: 26.10.94 / 20.11.1994

% Dieses Netz besitzt S1 Eingangs-Neuronen, S2 Hidden-Neuronen
% und S3 Ausgangs-Neuronen. Somit besteht das KNN aus drei Layern.


% Mustervektoren laden
% ================

load ekgarm;
load ekgcarl;
load ekgrind;
figure(1)

p1=ekgarm(323:451);    % yellow
p2=ekgarm(3533:3661);  % magenta
p3=ekgarm(770:898);    % cyan
p4=ekgarm(5578:5706);  % red
p5=ekgarm(19665:19793);% green
p6=ekgarm(2594:2722);  % blue

pr1=ekgrind1(3141:3141+128);
          %pr2=ekgrind1(2367:2367+128);
          %pr3=ekgrind1(33357:33357+128);

pc1=ekgcarl1(2915:2915+128);
          %pc2=ekgcarl1(20287:20287+128);
          %pc3=ekgcarl1(7622:7622+128);


fp1=ekgarm(373:501);
fp2=ekgarm(3503:3631);
fp3=ekgarm(700:828);
fp4=ekgarm(5478:5606);
fp5=ekgarm(19765:19893);
fp6=ekgrind1(3171:3171+128);
fp7=ekgarm(339:467);
fp8=ekgarm(5563:5691);



% Vektor mit Trainingsmustern definieren
% ----------------------------------------------

must_vek=[fp6 p1 fp1 p2 fp2 p3 fp7 pr1 fp3 p4 fp4 p5 fp8 pc1 fp5 p6];

          % must_vek=[p1 pr1 fp1 p2 pc1 fp2 p3 pr2 fp3 p4 pc2 fp4 p5 pr3 fp5 p6 pc3];

% Muster plotten
% ===========
  figure(1)
  plot(must_vek),title('Trainigsmuster');
```

```
  pause;

% Netzwerkaugabe entsprechend den Mustern festlegen
% ------------------------------------------------------------------
targets=[0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];


% Netzwerkstruktur initialisieren
% ========================

 [R,Q] = size(must_vek);
 S1=50;
 S2 = 20;
 [S3,Q] = size(targets);


 [W1,B1] = nwlog(S1,R);
  W2 = rands(S2,S1)*0.01;
  B2 = rands(S2,1)*0.01;
  W3 = rands(S3,S2)*0.01;
  B3 = rands(S3,1)*0.01;


% Training der Muster mit den entsprechenden Targets
% ========================================

P = must_vek;
T = targets;



% Trainingparameter
% ------------------------

disp_freq = 15;
max_epoch = 250;
err_goal = 0.1;
lr = 0.01;
lr_inc = 1.05;
lr_dec = 0.7;

momentum = 0.95;
err_ratio = 1.04;


% Training ohne Rauschen
% -------------------------------

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);

% Trainierte Parameter speichern
% ========================

save pr1_w1.dat W1 /ascii /double
save pr1_b1.dat B1 /ascii /double
save pr1_w2.dat W2 /ascii /double
save pr1_b2.dat B2 /ascii /double
save pr1_w3.dat W3 /ascii /double
save pr1_b3.dat B3 /ascii /double
```

```
% Training mit Rauschen
% ================
pause;
max_epoch = 300;
err_goal = 0.5;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];

for pass = 1:5
  fprintf('Pass = %.0f\n',pass);
  P = [must_vek, must_vek, ...
  (must_vek + randn(R,Q)*0.2), ...
  (must_vek + randn(R,Q)*0.3)];
  T = [targets targets targets targets];
  [W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
end

% Nochmals ohne Rauschen trainieren
% ==========================

P = must_vek;
T = targets;

max_epoch = 250;
err_goal = 0.1;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
% [W1,B1,W2,B2] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);


% Trainingparameter mit Rauschen speichern
% ================================

save pr2_w1.dat W1 /ascii /double
save pr2_b1.dat B1 /ascii /double
save pr2_w2.dat W2 /ascii /double
save pr2_b2.dat B2 /ascii /double
save pr1_w3.dat W3 /ascii /double
save pr1_b3.dat B3 /ascii /double

% Ergebnis anzeigen
% =============

A = logsig(W3*logsig(W2*logsig(W1*P,B1),B2),B3);
SSE = sumsqr(A-T);
fprintf('Fehler der Quadratsumme ohne Rauschen: %g.\n',SSE);

end ;    % End of File
```

Script-File: „BP-EKGT3.M“

```
% Script-File zur EKG-Auswertung (Pulsfrequenzermittlung anhand der PQRS-Komplexe)
% ================================================================

% Alexander Scheib
```

```
% Stand:26.10.1994 / 20.11.1994

% Im Zusammenhang mit dem Trainingsprogramm "BP_EKG3.M" zu verwenden.
% Die Gewichte und Bias werden nicht vom Laufwerk gelesen, sondern müssen
% zuvor unter Anwendung von "BP_EKG3.M" ermittelt werden

% Testdatensätze
% -------------------
% (Es kann immer nur ein Datensatz ausgewählt werden - alle anderen müssen mit
%  „%“ ausdokumentiert werden)
% ( Zur Begrenzung der Auswertezeit werden die Datensätze auf 1000 Zeichen gekürtzt)

%s=ekgcarl1(2500:3500);
%s=ekgrind1(2500:3500);
%s=ekgarm(2500:3500);
s=ekgal1f(2500:3500);

%s=2*randn(size(s));        % Nur Rauschen um auf Fehldetektion zu testen
t=0:2*pi/(3500-2500):2*pi;s1=sin(t*0.3)';

[m,n]=size(s);
if n>1
  s=s';
end
% Signal mit additivem Rauschen überlagern
% -----------------------------------------------------

 sN=s+0.75*s1(1:size(s))+0.4*randn(size(s));


% Main
% ====
titel='(ohne Rauschen)';
for durchl=1:2
 l=length(s);
 AM=zeros(size(1:l));
 AMX=AM;
 AMY=AM;

 px=1:1:length(p1);
 px=px';
 clf;

 figure(1)
 plot(s),title('Zu untersuchendes Signal'),xlabel(titel);
 pause;
 l1=length(p1)-1;
 schwelle=0.65;
 tic
   for i=1:2:(l-l1)
   px=s(i:i+l1);
   A = logsig(W3*logsig(W2*logsig(W1*px,B1),B2),B3);
   AMX(i)=A(1,1);
   Ai=A(1,1).^2;
   if Ai > schwelle
     AM(i)=Ai;
   end;
  end;
 toc

 figure(2)
```

```
 plot(AM),title('Lage der Peaks'),xlabel(titel);
 pause;


 % Ergebnis filtern (Lauflänge der PQRS-Komplexe berücksichtigen)
 % ==============================================
 i1=1;
 anz=0;
 counter=0;
  laufz=round(l1/2);
  i1=find(AM > schwelle);
  l=(length(i1)-1);

  for i=1:1:l
   counter=counter+1;
   if ((i1(i)-i1(i+1)) < -laufz) | (i==l)
    xpos=i1(i-round(counter/2-0.5));
    AMY(xpos)=1;
    anz=anz+1;
    counter=0;
   end
  end
 figure (3)
 plot(AMY),title('Gefiltertes Ergebnis (Auftreten der PQRS-Komplexe)'),xlabel(titel);
 fprintf('Anzahl der Detektierten PQRS-Komplexe: %g\n',anz);
 pause;
 s=sN;
 titel='(mit Rauschen - +sinus+weißes Rauschen)';
end

end ;    % End of File
```

Script-File: „KORRKOEF.M“

```
% EKG-Mustererkennung mittels Kreuzkorrelationsanalyse
% Alexander Scheib
% Stand : 18.12.94

function [korrvek,korr] = korrkoef(vek,mustvek);
% korrvek: Alle Kreuzkorrelationskoeffizienten
% korr: Nur Kreuzkorrelationskoeffizienten die größer ‘schwelle’ sind

m = length(vek);
n=length(mustvek);

% Mittelwert von Signalvektor und Mustervektor bilden
mean2=mean(vek);
mean1=mean(mustvek);

% Gleichanteil von Muster- und Signalvektor subtrahieren
mustvek = mustvek - mean1;
vek     = vek-mean2;

% Initialisieren der Variablen und Rückgabevektoren
korrvek = zeros(size(1:m));
xco = zeros(size(1:m));
korr=zeros(size(1:m));
%xqq=1/n*sum(mustvek.^2);
xqq=std(mustvek)^2;

% Main
% ====
% Berechnung der Kreuzkorrelationsfaktoren nach der Formel:
```

$$\% \; korrvek(i) = \frac{\frac{1}{n}\sum_{i=1}^{n} x_i y_i}{\sqrt{\overline{x^2} \cdot \overline{y^2}}}$$

```
tic
m=m-n;
   for i=1:m
    vek1=vek(i:i+n-1)-mean(vek(i:i+n-1));
    %yqq=1/n*sum(vek1.^2);
     yqq=std(vek1)^2;
     for j=1:n
      xyqh(j)=abs(vek1(j)*mustvek(j));
     end;
    xyq=1/n*sum(xyqh);
    %   A = xcorr(mustvek,vek1,'coeff');
    A= xyq/sqrt(xqq*yqq);
    korrvek(i) = A;
end
toc

% Extraktion der Kreuzkorrelationskoeffizienten größer ‘schwelle’
for i=1:m
  if (korrvek(i) >= 0.65)
     korr(i)=korrvek(i);
  end
end

% Ausgabe der Ergebnisse
```

```
% ------------------------------
figure(1)
plot(korr),title('Nur Korrelationsfaktoren >=0.65');
xlabel(‘1/fabt’);
pause;

figure(1)
plot(korrvek),title('Alle Korrelationsfaktoren');
xlabel(‘1/fabt’);
pause;

end ;    % End of File
```

Script-File: „BP-EKG3x.M

```
% Trainingsprogramm zur EKG-Detektion im Summensignal aus EKG und EMG
% ==========================================================

% Alexander Scheib
% Stand: 06.01.95 / 22.02.1995

% Dieses Netz besitzt S1 Eingangs-Neuronen, S2 Hidden-Neuronen
% und S3 Ausgangs-Neuronen. Somit besteht das KNN aus drei Layern.


% Mustervektoren laden
%================

p1 = matr_ekg2(:,40);
p2 = matr_ekg2(:,35);

% 'Falsch-Muster' definieren
fp1=uebl(11600:11800)';
fp2=uebl(10000:10200)';

must_vek=[fp2 p1 fp1 p2];

%must_vek=[fp6 p1 fp1 p2 fp2 p3 fp7 pr1 fp3 p4 fp4 p5 fp8 pc1 fp5 p6];

          % must_vek=[p1 pr1 fp1 p2 pc1 fp2 p3 pr2 fp3 p4 pc2 fp4 p5 pr3 fp5 p6 pc3];

% Mustervektoren darstellen
% ====================
  figure(1)
  plot(must_vek),title('Trainigsmuster');
  pause;

% Netzwerkausgabe für die entsprechenden Mustervektoren vorgeben
targets=[0 1 0 1];


% Netzwerkstruktur initialisieren
% =======================

 [R,Q] = size(must_vek);
 S1=50;
 S2 = 20;
 [S3,Q] = size(targets);



% Gewichte und Bias initialisieren
 [W1,B1] = nwlog(S1,R);
  W2 = rands(S2,S1)*0.01;
  B2 = rands(S2,1)*0.01;
  W3 = rands(S3,S2)*0.01;
  B3 = rands(S3,1)*0.01;

% Training der Muster mit den entsprechenden Targets
% ====================================

P = must_vek;
```

```
T = targets;


% Trainingparameter festlegen
% -----------------------------------

disp_freq = 15;
max_epoch = 250;
err_goal = 0.1;
lr = 0.01;
lr_inc = 1.05;
lr_dec = 0.7;
momentum = 0.95;
err_ratio = 1.04;

% Training ohne Rauschen
% ------------------------------
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);


% Trainierte Parameter speichern
% ======================

save x1_w1.dat W1 /ascii /double
save x1_b1.dat B1 /ascii /double
save x1_w2.dat W2 /ascii /double
save x1_b2.dat B2 /ascii /double
save x1_w3.dat W3 /ascii /double
save x1_b3.dat B3 /ascii /double


% Training mit Rauschen
% ================
pause;

% Trainingsparameter modifizieren
max_epoch = 300;
err_goal = 0.5;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
for pass = 1:5
  fprintf('Pass = %.0f\n',pass);
  P = [must_vek, must_vek, ...
      (must_vek + randn(R,Q)*0.2), ...
      (must_vek + randn(R,Q)*0.3)];
  T = [targets targets targets targets];
  [W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
end

% Nochmals ohne Rauschen trainieren
% ==========================
P = must_vek;
T = targets;

max_epoch = 250;
err_goal = 0.1;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
% [W1,B1,W2,B2] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);
```

```
% Trainingparameter mit Rauschen speichern
% ================================

save x_w1.dat W1 /ascii /double
save x_b1.dat B1 /ascii /double
save x_w2.dat W2 /ascii /double
save x_b2.dat B2 /ascii /double
save x_w3.dat W3 /ascii /double
save x_b3.dat B3 /ascii /double


% Ergebnis anzeigen
% =============

A = logsig(W3*logsig(W2*logsig(W1*P,B1),B2),B3);
SSE = sumsqr(A-T);
fprintf('Fehler der Quadratsumme ohne Rauschen: %g.\n',SSE);

end ;    % End of File
```

Script-File: „BP-EKT3X.M“

```
% Script-File zur EKG-Detektion aus dem Summensignal von EKG und EMG
% mittels eines dreischichtigen Neuronalen Netzes
% ==========================================================

% Alexander Scheib
% Stand:31.01.1995 / 23.02.1995

% Im Zusammenhang mit dem Trainingsprogramm "BP_EKG3X.M" zu verwenden.
% Die Gewichte werden nicht vom Laufwerk gelesen, sondern müssen
% zuvor von "BP_EKG3X.M" ermittelt werden


% Zu untersuchender Datensatz
% -------------------------------------
% (Um die Auswertedauer zu begrenzen wird die Länge des Datensatzes auf 1000
% Zeichen beschränkt)

% Auswahl eines Datensatzes - alle anderen werden mit „%“ ausdokumentiert
%s=mes9(8000:10500);
%s=mes9(19100:21100);
s=uebl(9000:14000)';
%s=uebl(12800:13000)';
%s=[s; s; s; s; s; s; s];

% Signal mit additivem Rauschen überlagern
%-----------------------------------------------------
 t=0:2*pi/(length(s)):2*pi;s1=sin(t*0.3)';
 sN=s+0.1*s1(1:size(s))+0.1*randn(size(s));

% Main
```

```
% ====
% Initialisierung der Variablen, Vektoren und Matrizen
 l=length(s);
 AM=zeros(size(1:l));
 AMX=AM;
 AMY=AM;

 [m,n]=size(x_W1);
 px=1:1:n;
 px=px';
 clf;

 figure(1)
 plot(s),title('Zu untersuchendes Signal');
 pause;
 l1=n-1;

 schwelle=0.65;


 tic
   for i=1:2:(l-l1)
    px=s(i:i+l1);
    A = logsig(x_W3*logsig(x_W2*logsig(x_W1*px,x_B1),x_B2),x_B3);
     if sum(A)<=1
      AMX(i)=A(1,1);
       Ai=A(1,1);    %.^2;
       AM(i)=Ai;
     end;
   end;
  toc

nul=zeros(1,100);
AMX= [nul AMX(1,1:length(AMX))];
AM=[nul AM(1,1:length(AM))];
 figure(2)
 plot(AM),title('Lage der EKG - Komplexe');
 pause;

 % Ergebnis filtern (Selektion der Netzwerkausgaben die größer ‘schwelle’ sind)
 % ==========================================================
 i1=1;
 anz=0;
 counter=0;
 % Berücksichtigung der Dauer eines PQRS-Komplexes
  laufz=round(l1/4);
  i1=find(AM > schwelle);
  l=(length(i1)-1);
  for i=1:1:l
   counter=counter+1;
   if ((i1(i)-i1(i+1)) < -laufz) | (i==l)
    xpos=i1(i-round(counter/2-0.5));
    AMY(xpos)=1;
    anz=anz+1;
    counter=0;
   end;
  end;

 figure (3);
 plot(AMY),title('Lage der EKG - Komplexe');
 pause;
```

```
 fprintf('Anzahl der Detektierten EKGs: %g\n',anz);
 AMX1=AMX;
 AMY1=AMY;
% Auswertung der Netzwerkantwort mit verrauschtem Signal
% ==========================================

titel='(mit Rauschen - +sinus+weißes Rauschen)';

% Initialisierung der Variablen und Matrizen
l=length(s);
AM=zeros(size(1:l));
AMX=AM;
AMY=AM;
clf;

figure(1)
plot(sN),title('Zu untersuchendes Signal');
pause;

tic
 for i=1:1:(l-l1)
  px=sN(i:i+l1);
  A = logsig(x_W3*logsig(x_W2*logsig(x_W1*px,x_B1),x_B2),x_B3);
  if sum(A)<=1
   AMX(i)=A(1,1);
   Ai=A(1,1);    %.^2;
    if Ai > schwelle
     AM(i)=Ai;
    end;
  end;
 end;
toc

nul=zeros(1,100);
AMX= [nul AMX(1,1:length(AMX))];
AM=[nul AM(1,1:length(AM))];

 figure(2)
 plot(AM),title('Lage der Peaks'),xlabel(titel);
 pause;


 % Ergebnis filtern (Lauflänge und ‘schwelle’)
 % ================================

 i1=1;
 anz=0;
 counter=0;
  laufz=round(l1/4);
  i1=find(AM > schwelle);
  l=(length(i1)-1);

  for i=1:1:l
   counter=counter+1;
   if ((i1(i)-i1(i+1)) < -laufz) | (i==l)
    xpos=i1(i-round(counter/2-0.5));
    AMY(xpos)=1;
    anz=anz+1;
    counter=0;
   end;
```

```
 end;
 figure (3);
 plot(AMY),title('Gefiltertes Ergebnis (Auftreten der Aktionspotentiale)'),xlabel(titel);
 pause;
 fprintf('Anzahl der Detektierten APs: %g\n',anz);
 AMX2=AMX;
 AMY2=AMY;

end ;    % End of File
```

Scrip-File: „EKG-NULL“

```
% EKG-Komplexe im überlagerten Signal von EMG und EKG Null setzen

% Alexander Scheib
% Stand: 07.02.1995

% Positionen der EKG-Komplexe suchen
pos_ekg=find(AMY1==1);
null=zeros(201,1);

sx=s;
for i=1:length(pos_ekg)
  sx(pos_ekg(i)-100:pos_ekg(i)+100,1)=null;
end;

% Ausgabe des resultierenden Signals
figure(4);
plot(sx);

end ;    % End of File
```

Script-File: „BP-EMG3B.M“

```
% Netzwerktrainingsprogramm zur EMG-Verifizierung
% ======================================

% Alexander Scheib
% Stand: 03.02.95 / 16.02.1995

% Dieses Netz besitzt S1 Hidden-Neuronen, S2 Ausgangs-Neuronen
% Somit besteht das KNN aus zwei Layern.


% Trainingsmustervektoren definieren
% ==========================

figure(1)
```

```
p1=mes9(57650:57850);
p2=mes9(59050:59250);
p3=mes9(62550:62750);
p4=mes9(20350:20550);
p5=mes9(22350:22550);

fp1=mes9(22100:22300);
fp2=mes9(19950:20150);
fp3=mes9(9700:9900);
fp4=mes9(9250:9450);

fp5=uebl(14550:14750)';
fp6=uebl(14770:14970)';
fp7=uebl(12800:13000)';
fp8=uebl(12930:13130)';

fp9 = uebl(12150:12350)';
fp10 =uebl(13060:13260)';
fp11= uebl(12170:12370)';

must_vek=([p1 fp9 p1 fp1 p2 fp2 p3 fp10 p3 fp5 p4 fp6 p5 fp11 fp8]);
        %must_vek=([p1 p1 p1 must_ekg p1 p1 p1]);
          % must_vek=[p1 pr1 fp1 p2 pc1 fp2 p3 pr2 fp3 p4 pc2 fp4 p5 pr3 fp5 p6 pc3];

% Mustervektoren anzeigen
% ==================

  figure(1)
  plot(must_vek),title('Trainigsmuster');
  pause;

% Variablen und Vektoren initialisieren
[m,n]=size(matr_ekg2(:,1:2:67));
nu=zeros(1,n);
ei=ones(1,n);


% Netzwerkausgabe festlegen
% ====================
targets=[1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
      0 1 0 0 0 0 0 1 0 0 0 0 0 1 0];


% Netzwerkstruktur initialisieren
% =======================

 [R,Q] = size(must_vek);
 S1=200;
  [S2,Q] = size(targets);

% Bias und Gewichte initialisieren
 [W1,B1] = nwlog(S1,R);
  W2 = rands(S2,S1)*0.01;
  B2 = rands(S2,1)*0.01;


% Training der Muster mit den entsprechenden Targets
% ======================================

P = must_vek;
```

```
T = targets;

% Fehleroberfläche grafisch darstellen
% ==========================

clf;
fprintf('\n\nDie Fehleroberfläche wird berechnet, einen Moment bitte...\n\n');
Wrange = -3:.2:3;
Brange = -3:.2:3;

ES = errsurf(P(6,:),T,Wrange,Brange,'logsig');
mesh(Wrange,Brange,ES);
view([60 30]);
xlabel('Gewichte W');
ylabel('Bias B');
zlabel('Quadratsummenfehler')
title('Fehler Oberflächengraph')
pause


% Trainingparameter einstellen
% ------------------------------------

disp_freq = 20;
max_epoch = 10000;
err_goal = 0.01;
lr = 0.005;
lr_inc = 1.06;
lr_dec = 0.6;
momentum = 0.9;
err_ratio = 1.05;


% Training des Netzwerkes mit unverrauschten Mustern
% ------------------------------------------------------------------

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
% [W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
[W1,B1,W2,B2] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);


% Trainierte Parameter speichern
% ======================

save p1_w1.dat W1 /ascii /double
save p1_b1.dat B1 /ascii /double
save p1_w2.dat W2 /ascii /double
save p1_b2.dat B2 /ascii /double


% Training des Neuronalen Netzes mit verrauschten Mustern
% ==========================================

% Modifikation der Trainingsparameter
max_epoch = 1000;
err_goal = 0.5;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
figure(2);

for pass = 1:0
```

```
  fprintf('Pass = %.0f\n',pass);
  P = [must_vek, must_vek, ...
     (must_vek + randn(R,Q)*0.1), ...
     (must_vek + randn(R,Q)*0.2)];
  T = [targets targets targets targets];
  [W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);  end

% Nochmals ohne Rauschen trainieren
% ==========================

P = must_vek;
T = targets;

max_epoch = 1000;
err_goal = 0.1;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);
% [W1,B1,W2,B2] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);


% Trainingparameter mit Rauschen speichern
% ================================

save p_w1.dat W1 /ascii /double
save p_b1.dat B1 /ascii /double
save p_w2.dat W2 /ascii /double
save p_b2.dat B2 /ascii /double


% Ergebnis anzeigen
% =============

A = logsig(W3*logsig(W2*logsig(W1*P,B1),B2),B3);
SSE = sumsqr(A-T);
fprintf('Fehler der Quadratsumme ohne Rauschen: %g.\n',SSE);

end ;    % End of File
```

Script-File: „EMG-PEAK.M“

```
% Script-File zur Selektion und Verifikation der verbleibenden EMG-Peaks
% im „EKG-bereinigten“ Datensatz

% Alexander Scheib
% Stand: 07.02.1995 / 23.02.1995

% Peaks im Signalvektor lokalisieren
[maxmax,pos]=max(sx);
pos_emg=find(sx>maxmax/2);
schwelle=0.5;


% Initialisierung der Variablen und Matrizen
 l=length(sx);
 AM=zeros(size(1:l));
```

```
 AMX=AM;
 AMY=AM;
 sy=zeros(1,length(sx))';


% Main
% ====

for i=1:length(pos_emg)
  % Signalabschnitt des  i ten Peaks auswählen
  px=sx(pos_emg(i)-70:pos_emg(i)+130);
  sy(pos_emg(i)-70:pos_emg(i)+130)=px;
   % Anwendung des Neuronalen Netzes
  A = logsig(p_W2*logsig(p_W1*px,p_B1),p_B2);
  % Entsprechend der Targetvektoren relevante Netzwerkantworten selektieren
  if sum(A)<=1
   AMX(pos_emg(i))=A(1,1);
   Ai=A(1,1);    %.^2;
   AM(pos_emg(i))=Ai;
  end;
end;
nul=zeros(1,100);

% Auftrittsstellen der gefundene Peaks anzeigen
 figure(5)
 plot(sy),title('Lage der Peaks');
 pause;

 figure(6)
 plot(AM),title('Durch Neuronales Netz detektierte Peaks ');
 pause;


 % Ergebnis filtern (Berücksichtigung der Länge der Peaks)
 % ========================================

% Variablen Initialisieren
 i1=1;
 anz=0;
 counter=0;
  laufz=round(l1/4);
  i1=find(AM > schwelle);
  l=(length(i1)-1);



% Ausgabevektor des Neuronalen Netzes durchsuchen
  for i=1:1:l
   counter=counter+1;
   if ((i1(i)-i1(i+1)) < -laufz) | (i==l)
   xpos=i1(i-round(counter/2-0.5));
   AMY(xpos)=1;
   anz=anz+1;
   counter=0;
   end
  end


% Ergebnis anzeigen
% =============
```

```
 figure (7)
 plot(AMY),title('Lage der Aktionspotentiale (gefiltertes Ergebnis)');

 fprintf('Anzahl der Detektierten APs: %g\n',anz);
 AMX2=AMX;
 AMY2=AMY;
 sy1=sy;

end ;    % End of File
```

Script-File: „START-EM.M“

```
% Start File zur EMG extraktion aus einem überlagerten Signal von EMG und EKG
% (automatisierter Ablauf)

% Alexander Scheib
% Stand: 07.02.1995 / 23.02.1995

% Die Bais und Gewichte müssen vorher mit "bp_emg3b" und "bp_ekg3x" berechnet
% werden

% Laden des Summensignals aus EKG und EMG
load uebl;

% Laden der ermittelten Gewichte und Bias für die EKG-Eliminierung
% ---------------------------------------------------------------------------------

load x_w1;
load x_w2;
load x_w3;
load x_b1;
load x_b2;
load x_b3;

% Laden der ermittelten Gewichte und Bias für die Verifikation der EMG-Peaks
% ---------------------------------------------------------------------------------------------

load p_w1;
load p_w2;
load p_b1;
load p_b2;

% Lokalisierung der EKG-Komplexe
% ------------------------------------------
bp_ekt3x;

% EKG-Komplexe im Signal durch Null ersetzen
% ---------------------------------------------------------

ekg_null;
pause;

% Verifizieren der verbleibenden EMG-Peaks und grafisch darstellen
% ---------------------------------------------------------------------------------

emg_peak;

end ;    % End of File
```

Script_File: „TR_TF.M“

```
% Funktion zur Rise Time und Fall Time Berechnung innerhalb einer Burstfolge
% In ‘mes’ muß der EMG-Burst übergeben werden

% Alexander Scheib
% Stand: 12.1.1995

function [tr_ms,tf_ms,anstiegswinkel,abfallwinkel] = tr_tf(mes,fabt);
```

```
% Abtastfrequenz
% z.B.: fabt= 1000 (Abtastfrequenz)
[maxi,place] =max(mes);
trigger1=maxi*1/2;
trigger2=maxi*1/10;
sw=0; ii=1; r=1;
for i=1:length(mes)
 if (mes(i)>trigger1) & (sw==0)
  while (mes(i-r) > trigger2)
   r=r+1;  end;
  i1(ii)=i-r;  r=1;   sw=1;
 end;
 if (mes(i)<trigger1) & (sw==1)
 while (mes(i+r) > trigger2)
   r=r+1;   end;   i2(ii)=i+r;   r=1;   sw=0;
  [peaks(ii),maxim(ii)]=max(mes(i1(ii):i));  ii=ii+1;
 end;
end;

% tr zwischen i1(ii) und maxim(ii)
% tf zwischen maxim(ii) und i1(ii)

for ii=1:length(peaks)
  tr(1:maxim(ii),ii)=diff(mes(i1(ii):i1(ii)+maxim(ii)).*fabt);
  tr_ms(ii)=(maxim(ii))*1/fabt*1000;
  trmean(ii)=mean(tr(:,ii));
  tf(1:maxim(ii),ii)=diff(mes(i2(ii)-maxim(ii):i2(ii)).*fabt);
  tf_ms(ii)=( i2(ii)-(i1(ii)+maxim(ii)) )*1/fabt*1000;
  tfmean(ii)=mean(tf(:,ii));
end;

if length(tr_ms)>2
  [a,b]=max(tr_ms);
  [a1,b1]=min(tr_ms);
  [a2,b2]=max(tf_ms);
  [a3,b3]=min(tf_ms);
  tr_ms(b)=mean(tr_ms);
  tr_ms(b1)=mean(tr_ms);
  tf_ms(b2)=mean(tf_ms);
  tf_ms(b3)=mean(tf_ms);
end;

tr_mean=mean(tr_ms);
tf_mean=mean(tf_ms);

% Die Spalten in "anstieg " enthalten jeweils den Mittelwert der Steigungen
% einer Flanke (entsprechend "abfall")
anstieg=atan(trmean).*180./pi   % in Grad
abfall=atan(tfmean).*180./pi    % in Grad
anstiegswinkel=mean(anstieg);
abfallwinkel=mean(abfall);
tges=tr_mean+tf_mean;
clc;

figure(1);
plot(mes);
pause;
close(figure(1));

fprintf('\nMittlerer Anstiegswinkel (in Grad) der detektierten Flanken\n');
disp(anstieg);
```

```
fprintf('\n\nMittlerer Abfallwinkel (in Grad) der detektierten Flanken\n');
disp(abfall);

fprintf('\n\n\nMit beliebiger Taste weiter\n');
pause;
clc;

fprintf('\n\nAnstiegswinkel im Durchschnitt:  %4.2f °\n',anstiegswinkel);
fprintf('\n\nAbfalwinkel im Durchschnitt:  %4.2f °\n',abfallwinkel);
fprintf('\n\nMittlere Anstiegszeit: %6.2f ms\n',tr_mean);
fprintf('\n\nMittlere Abfallzeit: %6.2f ms\n',tf_mean);
fprintf('\n\nGesamtlänge des Peaks (10%% bis 90%%): %6.2f ms\n\n\n\n\n\n',tges);
```

Script-File: „STDVAR.M“

```
% Funktion zur std und var Berechnung innerhalb einer Burstfolge
% In mes muß der Burst übergeben werden

% Alexander Scheib
% Stand: 12.1.1995

function [std1,var1,maxi,peaks] = stdvar(mes);

[maxi,place] =max(mes);trigger=maxi*1/3;sw=0;ii=1;
for i=1:length(mes)
 if (mes(i)>trigger) & (sw==0)   i1=i;  sw=1; end;
 if (mes(i)<trigger) & (sw==1)   sw=0;  [peaks(ii),placeii]=max(mes(i1:i));
ii=ii+1; end; end;
std1=std(peaks);var1=var(peaks);
end; % of File
```

Script-File: „MEDICA.M“

% Präsentationsprogramm für die Messe „MEDICA“ im November 1994
% (Düsseldorf)
% In einer Endlosschleife wird phasenweise die Filterung eines EKG dargestellt
%
%
%

```
load ekgal1;
ant=0;
while(ant==0)
fabt=256;
fgr=45;
st=ekgal1(500:1523);
i=0;
 [gt,gt2]= lowpassm(st,fabt,fgr);
clc;
end;

% Tiefpaß-Demo-Version für die Medica
% ============================
% gt    = Ausgangssignal im Zeitbereich
% gt2   = Ausgangssignal im Zeitbereich mit Phasenkompensation (des FIR Filters)
% st    = Eingangssignal im Zeitbereich
% fabt  = Abtastfrequenz
% fgr   = Grenzfrequenz des Filters

% Stand :20.09.1994 / 07.11.1994 / 11.11.94
% Alexander Scheib

function [gt,gt2] = lowpassm (st,fabt,fgr);
[m,n]=size(st);
if m>1
  st=st';
end
ant=0;  % endlos Schleife
while(ant==0)
clf; clc;
% Mittelwertfreies Eingangssignal bilden
  st=st-mean(st);
  l=length(st);
  lst=l;
  x=1;
% optimales N fuer FFT finden
  n=round(log2(l)-0.5);
  N=2^n;
% Zeitsignal st plotten
  % text(0,-9.4,'Plot des Zeitsignals');
  bin_number=1:l;
  t=(bin_number-1)*(1/fabt);
  figure(1)
  subplot(2,2,1);
  plot(t,st,'-'),title('Ungefiltertes Zeitsignal '),
  xlabel('t/s')
%  text(0,-10.4,'Weiter mit beliebiger Taste');
  pause2(1);

x=round(l/N-0.5);
  Sf=fft(st(1:N-1),N);

% Frequenzlinienabstand f0
  f0=(1/N)*fabt;

% Plot des Amplitudendichtespektrum Sf (f=Frequenz   a=Amplitudenvektor)
  a =abs(Sf);
  bin_number=1:length(Sf);
  f=(bin_number-1)*f0;
  subplot(2,2,2);
  plot(f,a,'-'),title('Amplitudendichtespektrum des Eingangssignals '),
```

```
  xlabel('f/Hz')
pause2 (1);

% Das Filter
% ==========
  Ta=1/fabt;
  Wn=fgr/(fabt/2);   % Normierte cut-off-frequency
                % Wn=fgr/(fa/2) sein !!!!?

% Filter mit halber Punktzahl der FFT bilden und Periodizieren (wegen IFFT)
  NN=round(N/2);
B= fir1(NN-1,Wn,boxcar(NN));
A=1;
  [h,w]=freqz(B,A,(NN));
              % w = NN-Punkt Frequenzvektor (anstatt f kontinuierlich)
              % h = NN-Punkt Uebertragungsfunktion H(nf)
  mag=abs(h);
  phase=unwrap(angle(h));
  bin_number=1:length(w);
  f=(bin_number-1)*(fabt/2)/NN;        % ist gleich f0/2
  ph=phase*180/pi ;
  subplot(2,2,3);
  plot (f,mag),title('Betrag der Übertragungsfunktion des Filters '),
  xlabel('f/Hz')
pause2(3);
  [m,n] =size(h);
  h=reshape(h,n,m);

% Filter periodizieren
  lh=length(h);
  for i=1:lh
    hh(lh-i+1)=h(i);    % conj(h(i));
  end
  h=[h hh];
  hi=h;
  for i=1:x-1
   h=[h hi];
  end
  Gf = Sf.* h;
   absGf=abs(Gf);
Gf2= Sf.* h.*conj(h)./abs(h);
% Plot des Amplitudenspektrum des Ausgangssignales G(f)
  bin_number=1:length(Gf);
  f=(bin_number-1)*(fabt/1)/N;

  % Plot des Amplitudendichtespektrum G(f) (f=Frequenz   a=Amplitudenvektor)
  subplot(2,2,4);
  plot(f,absGf,'-'),title('Amplitudendichtespektrum des gefilterten Signals '),
  xlabel('f/Hz')
pause2(1);

gt=ifft(Gf(1:N-1),N);
  gt1=gt;
 gt=abs(gt);
 gt2=real(ifft(Gf2(1:N-1),N));

% Betrag des gefilterten Zeitsignals plotten
  bin_number=1:length(gt);
  t=(bin_number-1)*(1/fabt);
subplot(2,2,3);
  plot(t,gt2,'-'),title('Gefiltertes Zeitsignal  '),
```

```
  xlabel('t/s')
 pause2(10);
end;  % of File
```