Kapitel 7

Neuronale Netze zur Datenklassifikation

7.1 Einleitung

Die Entwicklung der Neuroinformatik hat seit Beginn der 80er Jahre einen großen Aufschwung erfahren. Der wesentliche Grund dafür ist sicherlich die große Leistungssteigerung bei den Computern. Damit wurden Computersimulationen von komplexeren Gehirnmodellen und künstlichen neuronalen Netzen (KNN) erst möglich. Dagegen gehen die ersten aussagekräftigen Theorien über die Informationsverarbeitung im Gehirn und den Nervenzellen bis in die 40er Jahre zurück.

7.1.1 Attraktivität neuronaler Modelle

Es ist offensichtlich, daß v.Neumann-Computer bei kognitiven Aufgaben (Hören, Sehen, Mustererkennen, etc.) und bei unvollständiger, inkonsistenter oder verrauschter Information im Vergleich zum Gehirn versagen. Das Hit-Muster, das z.B. Teilchenspuren in einer Driftkammer hinterlassen (Abb. 7.1), hat unser Auge 'momentan', innerhalb O(0.1s), als stetig aufeinanderfolgende Punkte erkannt und miteinander verbunden. Der Zeitbedarf eines Computers ist nur dank seiner sehr viel größeren Geschwindigkeit pro einzelnem Rechenschritt vergleichbar. Mit künstlichen neuronalen Netzen könnte dieselbe Leistung innerhalb von O(μs) erzielt werden.

Gehirn-Architektur: Die charakteristischen Merkmale der Datenverarbeitung im Gehirn machen den Unterschied zu dem heutigen Standard für Computerarchitekturen klar:

- sehr viele parallele Prozessoren, O(10^{11}), insgesamt kompakt, geringer Energieverbrauch;
- langsame Einzelschritte, O(ms);
- massiv parallele Verarbeitung (O(10^{13}) Synapsen);
- keine Hardware-Software-, Algorithmen-Daten-Trennung;
- lernfähig:
  - evolutionäres, dynamisches Lernen gibt hohe Flexibilität für die Informationsverarbeitung,

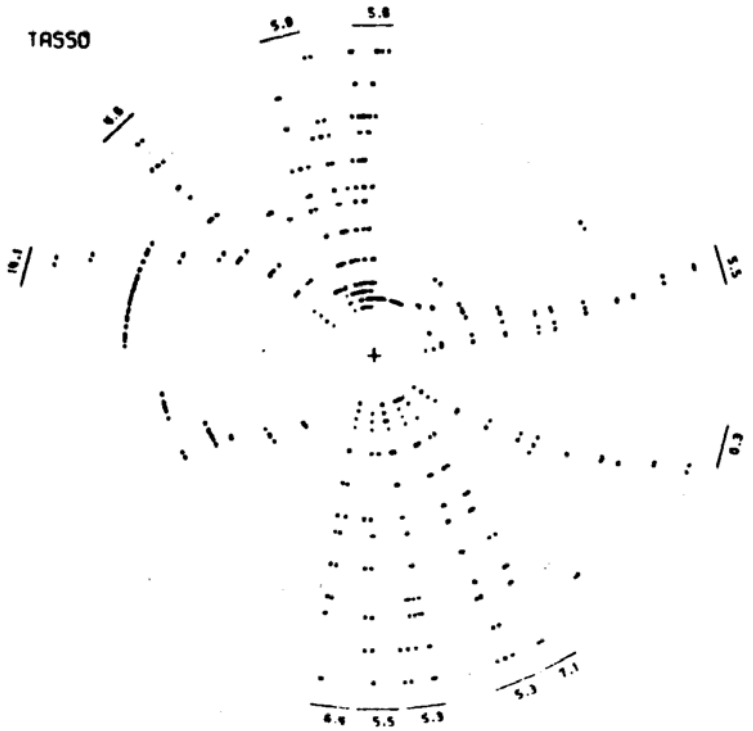


Abbildung 7.1: Hit-Muster, die von Teilchenspuren in einer Driftkammer (TASSO-Experiment) hinterlassen wurden.

  - evolutionäre Selbstorganisation gibt dem Netz eine gewisse Plastizität zur Anpassung an Neues;

- fehlertolerant (Abb. 7.2), Information kann bis zu einem gewissen Grade
  - unvollständig,
  - inkonsistent,
  - verrauscht sein;
- Stärke: schnelle Erfassung komplexer Zusammenhänge, kognitive Aufgaben, Mustererkennung, assoziative Verknüpfungen.

Literatur zu Neuronalen Netzen: Einführende Literatur zu neuronalen Netzen findet man unter [4, 5, 6, 7, 8, 9, 10, 11]. Siehe auch Spektrum der Wissenschaft, Nov.79 und Nov.92, beide Hefte sind dem Gehirn gewidmet [12, 13].

7.2 Natürliche und künstliche neuronale Netze

7.2.1 Natürliche neuronale Netze

Die intellektuellen Leistungen werden in der Hirnrinde (Neokortex) erzielt (Fläche etwa 0.2 m^2, Dicke 2-3 mm). Die Hirnrinde ist in Felder für verschiedene Teilaufgaben organisiert (z.B. visuelle, motorische, somatosensorische, Assoziations-Felder).

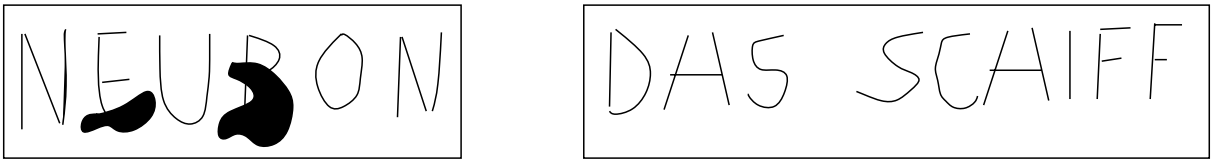


Abbildung 7.2: Beispiele für Fehlertoleranz und Ausgleich von Ungenauigkeiten im Gehirn: auf der linken Seite ist die Information verstümmelt; rechts wird exakt das gleiche Symbol einmal als 'A' und dann als 'H' im Zusammenhang richtig erkannt.

Ein Schnitt durch die Hirnrinde zeigt ein vertikal ausgerichtetes Netz von Neuronen (Nervenzellen) mit ihren Verzweigungen (Abb. 7.3). In einer vertikalen Säule von $1\,\mathrm{mm}^2$ befinden sich etwa 10^5 Neuronen, insgesamt gibt es etwa 10^{11} Neuronen im Gehirn.

Aufbau und Funktion der Neuronen:

Es gibt viele unterschiedliche Neuron-Typen. Um die uns interessierenden wesentlichen Eigenschaften von Neuronen zu beleuchten, konzentrieren wir uns auf die schematische Darstellung eines typischen Neurons in Abb. 7.4. Solch ein Neuron besteht aus

- dem Zellkörper, Durchmesser 5-80 μm,
- den Dendriten, die sich zu Dendritenbäumen mit einer Reichweite von 0.01-3 mm verzweigen,
- den Axons, die bis zu 1 m lang sein können.

Funktionsweise eines Neurons:

- Die Dendriten sammeln in einer Umgebung bis zu etwa 400 μm Signale von benachbarten Neuronen oder von den Axonen weiter entfernter Neuronen.
- Die Signalübertragung auf die Dendriten oder direkt auf den Zellkörper erfolgt über chemische Kontakte (Neurotransmitter) an den Synapsen innerhalb von O(1 ms). In der Hirnrinde hat jedes Neuron O(10^3) Synapsen (allgemein im Gehirn O(1) bis O(10^5)). Die Zeitskala für die Übertragung ist 1 ms, d.h. daß z.B. die visuelle Erkennung eines Bildes mit nicht mehr als O(10) serielle Schritten erfolgen muß.
- Das Summensignal aller Dendriten verändert das elektrische Potential des Zellkörpers.
- Bei Überschreiten einer Schwelle erzeugt diese Potentialänderung einen Nadelpuls (Spike) auf dem Axon (Signalgeschwindigkeit etwa 10 m/s).

Einfaches Modell: das McCulloch-Pitts-Neuron: Abbildung 7.5 zeigt das McCulloch-Pitts-Neuron, das einem logischen Schaltelement entspricht. Die binären Eingangssignale n_i erzeugen ein binäres Ausgangssignal n (n_i, $n = 0$ oder 1) nach der Vorschrift:

$$n(t+1) \;=\; \Theta\left(\sum_j w_j n_j(t) - s\right) \tag{7.1}$$

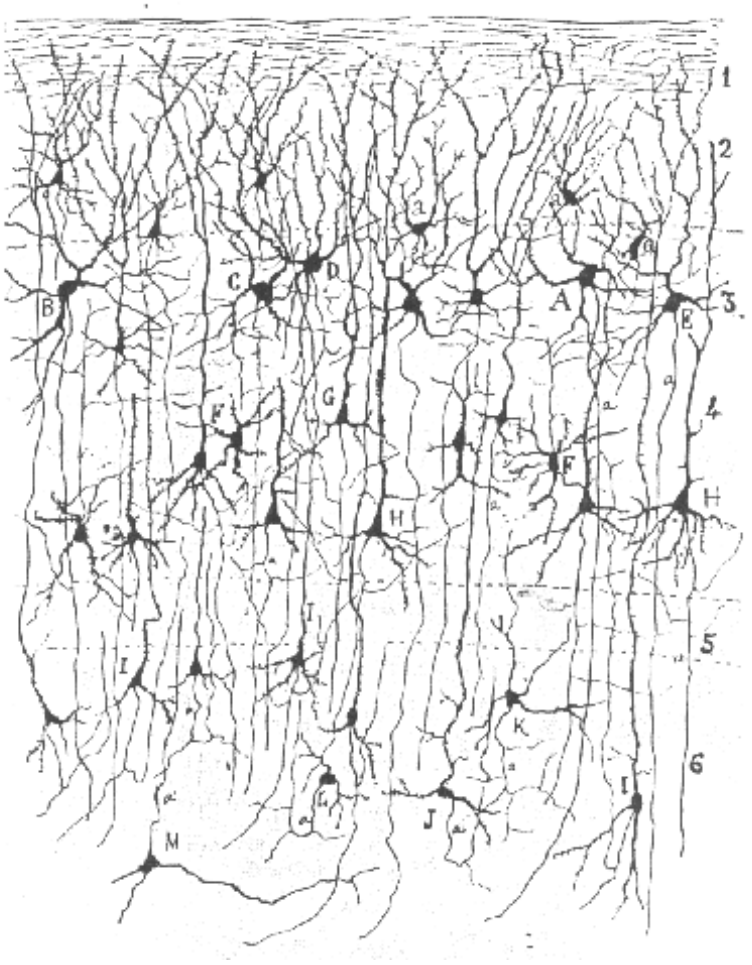


Abbildung 7.3: Vertikaler Schnitt durch die Hirnrinde. Die Dichte der Neuronen ist um einen Faktor 100 untersetzt

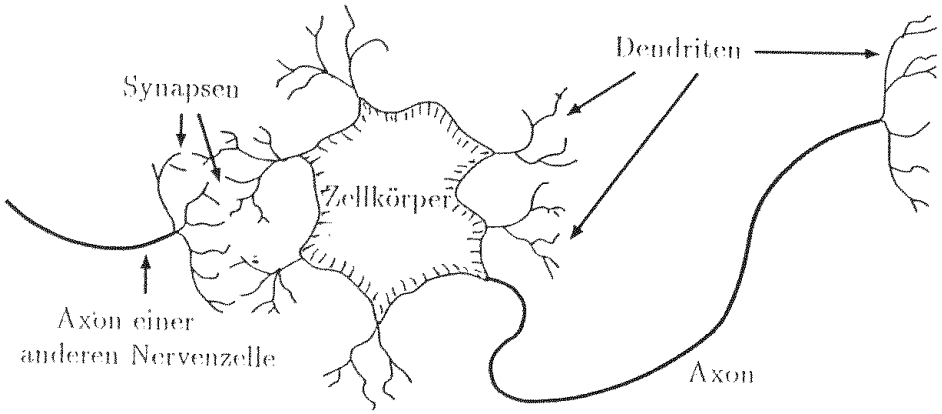


Abbildung 7.4: Schematische Darstellung eines Neurons.

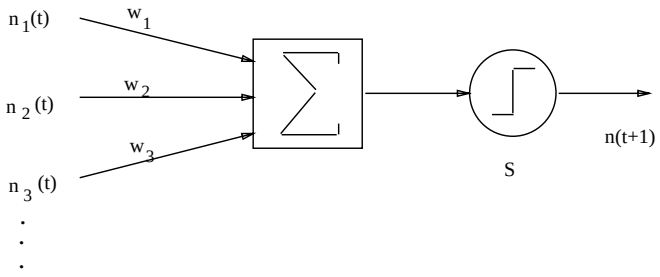


Abbildung 7.5: Neuron als logisches Schaltelement

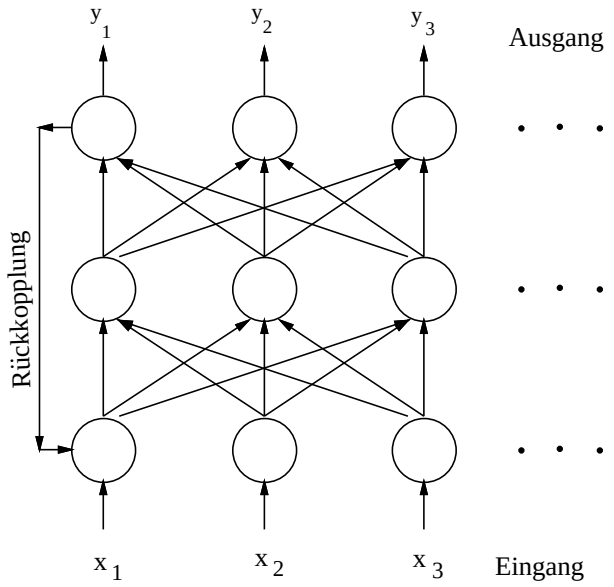


Abbildung 7.6: Beispiel für ein neuronales Netz.

Dabei ist t eine diskrete Zeitvariable. Die Heaviside-Funktion ist definiert als:

$$\Theta(x) = \begin{array}{ll} 1 & x \geq 0 \\ 0 & sonst \end{array}$$

Die Gewichte w_i entsprechen den Synapsenstärken, s ist der Schwellenwert. Das Neuron 'feuert' also, wenn die gewichtete Summe der Eingangssignale die Schwelle s überschreitet. Die Gewichte können > 0 (erregend) oder < 0 (hemmend) sein, wie es auch tatsächlich für Synapsen beobachtet wird.

Neuronale Vernetzung: Wesentlich für die Funktion des Gehirns ist das kollektive Verhalten eines Systems von nichtlinear gekoppelten Neuronen. Im Beispiel Abb. 7.6 werden die Eingangsreize x_i (z.B. visuelle Signale) in Ausgangssignale y_i (z.B. zur Bewegung eines Muskels) transformiert.

Lernen und Selbstorganisation:

Aus eigener Erfahrung wissen wir, daß das Gedächtnis auf unterschiedlichen Zeitskalen arbeitet. Manches ist bereits nach Sekunden verpflogen, wie die dauernd einwirkenden sensorischen Reize,

anderes behalten wir für Minuten oder Tage oder Jahre. Das Behalten im Gedächtnis ist also ähnlich einem evolutionärem Prozess. Generell scheint zu gelten, daß die Stärke und Häufigkeit eines Reizes das Lernen wesentlich beeinflußt. Man beachte, daß wir zum Lernen offensichtlich in der Regel nicht zu wissen brauchen, ob das Gelernte richtig ist ('Lernen ohne Lehrer').

Auf diese Beobachtungen ist die Lernregel von Hebb begründet: Die Synapsenstärke ändert sich proportional zu der Korrelation zwischen prä- und postsynaptischem Signal:

$$\Delta w_i = \eta \cdot y(x_i) \cdot x_i, \text{ mit } 0 < \eta < 1 \tag{7.2}$$

Der Lernparameter η legt die Lerngeschwingigkeit fest. Es ist ein besonders empfindlicher Parameter: einerseits möchte man schnell lernen, andererseits birgt zu schnelles Lernen die Gefahr, daß zuviel Unsinn abgespeichert wird.

Strukturbildung: Mit den etwa 10^{13} Synapsen ergeben sich etwa $10^{10^{14}}$ mögliche Konfigurationen des Gehirns. Das kann nicht alles genetisch festgelegt sein! Genetisch kodiert sind wahrscheinlich nur Organisationsschemata und ein Strukturbildungsmechanismus. Die Verbindungen zwischen den Neuronen werden z.T. evolutionär aufgrund sensorischer Reize gebildet und können meistens auch später noch verändert werden.

Topographische Abbildungen: Der Lernvorgang führt offensichtlich zu Strukturen im Gehirn, die vorgegebene topographische Zusammenhänge bei den einlaufenden Sinnesreizen intakt lassen. Beispielsweise wird im somatosensorischen Kortex der Tastsinn der Hautoberfläche so abgebildet, daß benachbarte Körperbereiche benachbart bleiben. Eine wesentliche Eigenschaft der Abbildung ist die Anpassung der Größe der Bildbereiche entsprechend der Wichtigkeit und das jeweils benötigte Auflösungsvermögen.

7.2.2 Künstliche neuronale Netze (KNN)

Künstliche neuronale Netze und neuronale Algorithmen sind in den letzten Jahren intensiv theoretisch untersucht, auf Computern simuliert und – seltener – als Hardware realisiert worden. Bei der Entwicklung von NN-Modellen wird man sich natürlich von den biologischen Befunden inspirieren lassen. Für die Anwendung ist es aber nicht wichtig, ob ein Modell tatsächlich in der Natur realisiert wird. Hier ist der praktische Erfolg ausschlaggebend.

Ausgehend von den im vorigen Abschnitt entwickelten Vorstellungen über natürliche neuronale Netze definieren wir im folgenden, welches die gemeinsamen Elemente der KNN-Modelle sein sollen. Diese Aufstellung ist nicht strikt, sondern soll eher eine Orientierung sein.

- Prozessorelement: (formales) Neuron, Netzwerk-Knoten (Abb. 7.7).
- Eingabeaktivitäten x_j (Signale auf den Dendriten) sind reelle Zahlen (oder Spannungen, Ströme), eventuell binär (-1,1) oder (0,1).
- Gewichte (entspricht den Synapsen) w_{ij}, > 0 (erregend), < 0 (hemmend)
- Aktivitätsfunktion, z.B.:

$$z_i = \sum_j w_{ij} x_j - s_i$$

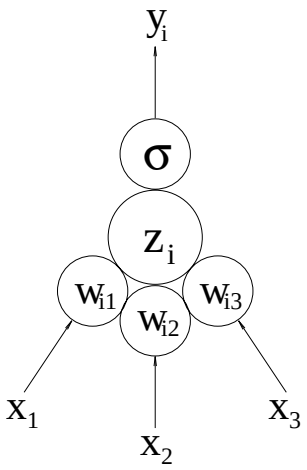


Abbildung 7.7: Struktur eines künstlichen Neurons

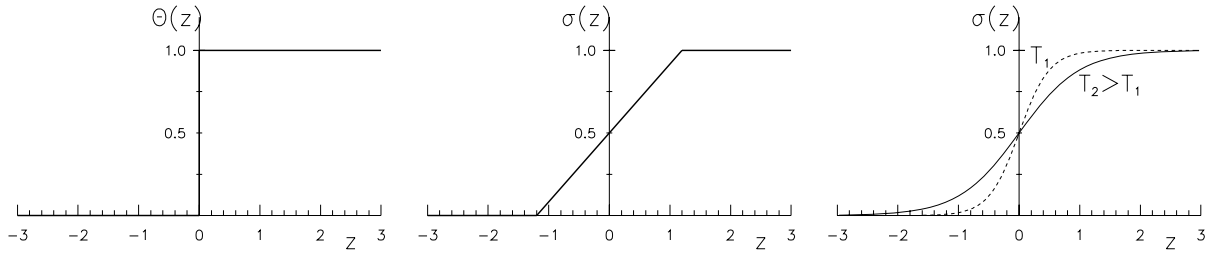


Abbildung 7.8: Beispiele von Schwellenfunktionen

- Ausgabefunktion (oder Transferfunktion) g:

$$y_i = g(z_i)$$

I.a. liegt y_i im Intervall [-1,1] oder [0,1] und hat häufig ein Schwellwertverhalten mit Sättigung an den Intervallgrenzen. Neben der Θ-Funktion werden häufig folgende 'sigmoide' Funktionen gewählt (Abb. 7.8):

$$\sigma(z) = \frac{1}{1+e^{-z/T}} \tag{7.3}$$

$$\sigma(z) = tanh(z/T) \tag{7.4}$$

$$\sigma(z) = 1/2(1+tanh(z/T)) \tag{7.5}$$

Die Funktionen (7.3) und (7.5) haben Werte im Intervall [0,1] und die Funktion (7.4) im Intervall [-1,1]. Sigmoide Funktionen haben den Vorteil im Bereich der Schwelle differenzierbar zu sein. Die 'Temperatur' T bestimmt den Bereich mit variabler Verstärkung:

Für T→ 0 geht σ in die Θ-Funktion über (binäres Neuron).

T groß: weiche Entscheidung.

- Netzwerk-Architektur: Netzwerk mit Knoten und Verbindungen
  - 'jeder mit jedem'

  - Nachbarschaftsverknüpfung
  - uni- oder bi-direktional
  - Schicht-Struktur mit hierarchischer Anordnung (z.B. feed-forward)
  - mit oder ohne Rückkopplung
  - ...

- Lernen:
  - Anpassung der Gewichte
  - Anpassung der Architektur: Erzeugen und Löschen von Neuronen und Verbindungen

- Lernregel:
  - selbständig (ohne Lehrer, unsupervised), z.B. Hebb-Regel
  - angeleitet (mit Lehrer, supervised) Vergleich des Netzwerk-Outputs mit der (vom Lehrer vorgegebenen) Erwartung, Anpassung durch Fehlerminimierung (z.B. Backpropagation- Algorithmus).

- Update-Regel: Neubestimmung eines Netzzustandes kann synchron, sequentiell oder iterativ (wegen nichtlinearer Kopplungen) gemacht werden.

- Netzwerk-Phasen:
  - Trainingsphase (Verwendung eines Trainings-Datensatzes)
  - Generalisierungsphase (Anwendung auf unbekannte Daten)

7.3 Feed-Forward-Netzwerke

In dieser Vorlesung wollen wir uns auf sogenannte Feed-Forward-Netzwerke beschränken, in denen die Neuronen geschichtet angeordnet sind und die Verbindungen streng nur in eine Richtung, jeweils zur nächsthöheren Schicht, von der Eingabeschicht bis zur Ausgabeschicht laufen (Abb. 7.6, ohne Rückkopplung). Feed-Forward-Netze (FFN) werden häufig zur

- Lösung von Klassifikationsaufgaben,
- Mustererkennung und
- Funktionsapproximation

benutzt. Für praktische Anwendungen sind sie wahrscheinlich der wichtigste Netzwerktyp. Ihre Bedeutung haben FFN wohl durch die von herkömmlichen Computern gut ausführbaren, im Prinzip sequentiellen, Algorithmen und insbesondere die Backpropagation-Lernvorschrift erhalten.

Das einfachste Beispiel ist das (einfache) Perzeptron mit nur einer Eingangsschicht und einer Ausgangsschicht. Mit Computersimulationen konnte gezeigt werden, daß ein Perzeptron 'intelligenter' Leistungen fähig ist: Es kann angebotene Muster unterscheiden und kann diese Musterklassifizierung mit Hilfe eines Lehrers lernen (supervised learning).

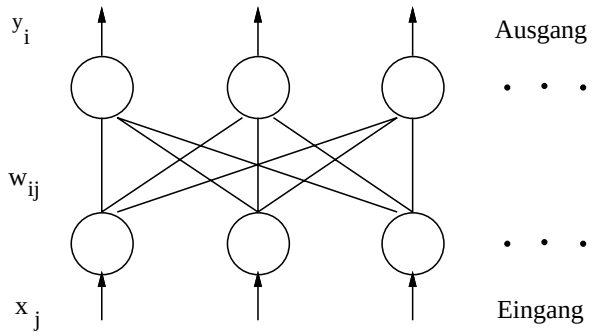


Abbildung 7.9: Perzeptron-Netzwerk

7.3.1 Das einfache Perzeptron

Definition und Eigenschaften des Perzeptrons:

Abbildung 7.9 zeigt das einfache Perzeptron mit einer Eingangsschicht (oder -lage) und einer Ausgangsschicht (wir ordnen den Eingängen eine Schicht zu, ist manchmal auch anders definiert). Jeder der k Eingänge ist mit jedem der l Ausgänge verbunden, den Verbindungen werden die Gewichte w_{ij} $(i = 1, \ldots, k;\ j = 1, \ldots, l)$ zugeordnet. Die Eingänge $x_1,\ x_2, \ldots, x_k$ lassen sich in einem 'Mustervektor' $\vec{x}$ zusammenfassen, der einen Punkt im 'Musterraum' (pattern space) darstellt. Die einzelnen Komponenten sind 'Merkmale' (features). Über die folgende Vorschrift wird einem Mustervektor $\vec{x}$ ein Ausgabevektor $\vec{y}$ zugeordnet:

$$y_i = g\left(\sum_j w_{ij} x_j\right) = g(\vec{w}_i \vec{x}) \tag{7.6}$$

Im letzten Teil wurden die Gewichte zu einem Ausgangsknoten i zu einem Vektor zusammengefaßt. Die Transferfunktion g ist gewöhnlich eine sigmoide Funktion (ursprünglich beim Perzeptron einfach die Θ-Funktion, wir wollen uns hier nicht darauf beschränken). In Gl. (7.6) kommen keine expliziten Schwellen s_i vor wie in der Formel (7.1) für das McCulloch-Pitts-Neuron. Schwellen können durch eine zusätzliche konstante Eingabe $x_0 = 1$ und die Gewichte $w_{i0} = -s_i$ berücksichtigt werden.

Beispiel: Darstellung der Boolschen Funktionen AND und OR: Wir wollen hier binäre Ein-und Ausgabegrößen betrachten mit Werten 0 und 1. Dann muß die Transferfunktion die Θ-Funktion sein, $g = \Theta$. Im folgenden wird gezeigt, daß sich die Funktionen AND und OR entsprechend der Wahrheitstafel in Abb. 7.10 durch ein Netz mit den 2 Eingängen x_1 und x_2 und einem Ausgang y realisieren lassen ('Ja-Nein-Maschine').

Wir wollen an dieser Stelle zunächst nicht der Frage nachgehen, wie das Netz die richtigen Antworten lernt; das wird dann allgemeiner für mehrschichtige FFN gezeigt (siehe Abschnitt 7.3.3). Man kann sich aber leicht davon überzeugen, daß die Gewichte

$$\text{AND}: \ (\mathrm{w}_0, \mathrm{w}_1, \mathrm{w}_2) \ = \ (-1.5, 1.0, 1.0)$$

$$\text{OR} \ : \ (\mathrm{w}_0, \mathrm{w}_1, \mathrm{w}_2) \ = \ (-0.5, 1.0, 1.0)$$

das Problem lösen (Abb. 7.10). Die Bedeutung dieses Resultates ist sehr anschaulich: Nach Gl. (7.6) wird der Raum der Muster (x_1, x_2) in 2 Klassen geteilt, die der Bedingung

$$\vec{w}\vec{x} \ < \ 0 \text{ bzw. } \vec{w}\vec{x} \ < \ 0$$

x_1	x_2	$y(AND)$	$y(OR)$	$w_1x_1 + w_2x_2$
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	2

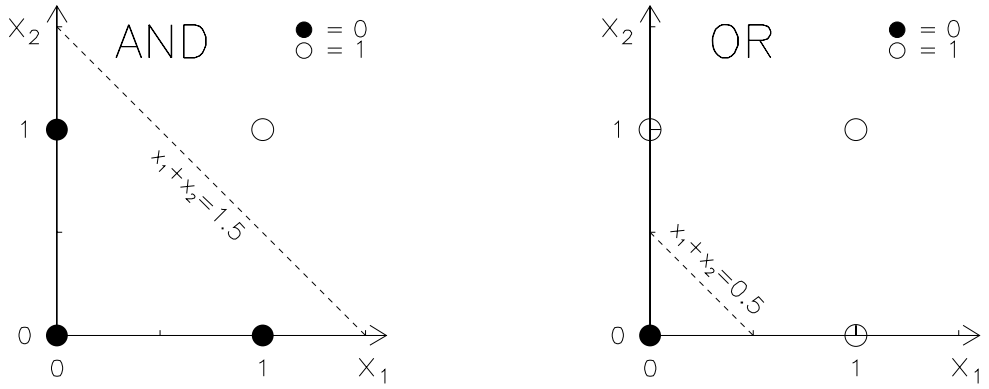


Abbildung 7.10: Oben: Wahrheitstafel für die Boolschen Funktionen AND und OR zusammen mit der Summe der gewichteten Eingänge wie vom Perzeptron berechnet. Unten: Klasseneinteilung im Musterraum für das AND- und OR-Problem. Die gestrichelten Geraden geben die von dem Perzeptron jeweils gefundene Klassentrennung an.

genügen. Die Trennung zwischen beiden Klassen

$$\vec{w}\vec{x} \;=\; 0$$

definiert eine Hyperebene im Musterraum, auf der der Vektor $\vec{w}$ senkrecht steht. In unserem Fall sind die Hyperebenen Geraden, die folgenden Gleichungen genügen:

$$\text{AND} : x_1 + x_2 = 1.5$$

$$\text{OR} \;\; : x_1 + x_2 = 0.5$$

Abbildung 7.10 zeigt die Lage der Geraden in dem Musterraum.

Allgemein gilt, daß durch Gl. (7.6) für jeden Ausgabeknoten eines Perzeptrons eine Hyperebene definiert wird, die jeweils den Musterraum in zwei Klassen einteilt. Die Trennung ist scharf für $g = \Theta$, was für eine Klasse $y = 0$ und für die andere $y = 1$ liefert. Bei einer sigmoiden Funktion ist die Ausgangsaktivität y ein (i.a. nichtlineares) Maß für den Abstand von der Hyperebene, solange man sich noch so nahe an der Hyperebene befindet, daß g noch nicht in Sättigung ist.

Limitierung des einfachen Perzeptrons:

Aus der vorangehenden Diskussion ergibt sich sofort, daß ein Perzeptron nur dann Muster in Klassen einteilen kann, wenn diese durch eine Hyperebene zu trennen sind. Man sagt in diesem Fall: die Klassen sind 'linear separierbar'; die Hyperebenen werden 'lineare Diskriminanten' genannt. Ein bekanntes, einfaches Beispiel, bei dem das einfache Perzeptron keine Lösung findet,

x_1	x_2	$y(XOR)$	$\vec{w}\vec{x}$
0	0	0	$w_0 < 0$
1	0	1	$w_0 + w_1 > 0$
0	1	1	$w_0 + w_2 > 0$
1	1	0	$w_0 + w_1 + w_2 < 0$

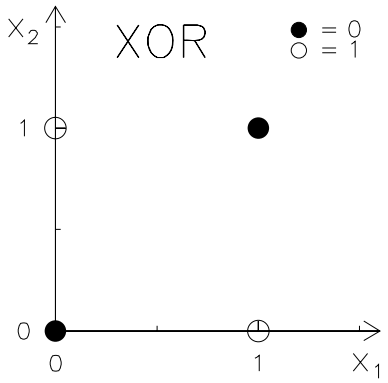


Abbildung 7.11: Links: Wahrheitstafel für die Boolschen Funktionen XOR zusammen mit den Bedingungen an die Gewichte. Rechts: Klasseneinteilung im Musterraum für das XOR-Problem.

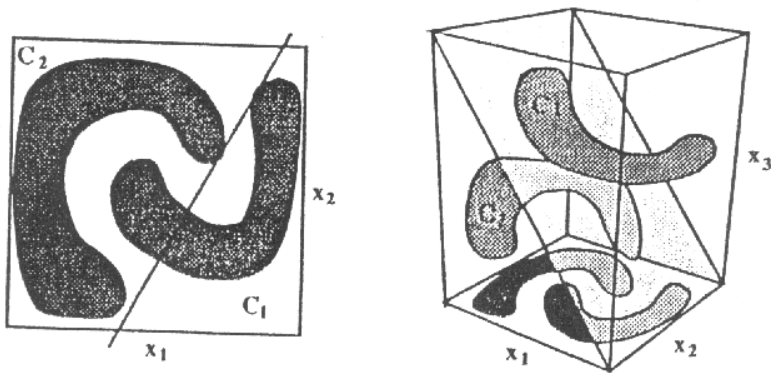


Abbildung 7.12: Lineare Separierbarkeit: a) in 2 Dimensionen nicht separierbar, b) in 3 Dimensionen separierbar.

ist die XOR-Funktion (Exclusive-OR) definiert in der Tabelle in Abb. 7.11. Man erkennt sofort, daß die Bedingungen an die Gewichte nicht gleichzeitig erfüllt werden können. Das entspricht der Tatsache, daß in Abb. 7.11 keine Gerade gefunden werden kann, die die $y = 0$- von der $y = 1$-Klasse trennt.

Ein anderes Beispiel von nicht linear separierbaren Punktemengen ist in Abb. 7.12a gezeigt. In solchen Fällen kann man eventuell doch noch eine Perzeptron-Lösung finden, wenn man ein weiteres Merkmal findet, daß die Klassen diskriminiert. Die trennende Hyperebene läge dann in einem um eine Dimension erweiterten Raum (Abb. 7.12b). Das Problem ließe sich auch mit Hilfe komplizierterer Transferfunktionen lösen, was aber dem grundlegenden Konzept für neuronale Netze (möglichst einfache Einzelschritte) widerspräche.

Eine allgemein anwendbare Lösung findet man durch Erweiterung des Perzeptron-Modells auf mehrschichtige Netze.

7.3.2 Das Mehrlagen-Perzeptron

Lösung des XOR-Problems:

Wir haben gesehen, daß ein einfaches Perzeptron durch

$$\vec{w}\vec{x} \;=\; 0 \tag{7.7}$$

Hyperebenen im Musterraum definiert, die den Raum in die beiden Klassen

$$\begin{array}{lll} \vec{w}\vec{x} & < & 0 \qquad\qquad \text{Klasse 1} \\ \vec{w}\vec{x} & > & 0 \qquad\qquad \text{Klasse 2} \end{array} \tag{7.8}$$

unterteilt. Mit der Kombination von Hyperebenen lassen sich offensichtlich Volumina im Musterraum definieren. Eine solche Kombination gelingt tatsächlich durch die Erweiterung des einfachen Perzeptrons um eine (oder mehrere) Lagen. Dieses Mehrlagen-Perzeptron hat dann neben den Eingangs- und Ausgangslagen auch versteckte Lagen (hidden layers).

Bei dem XOR-Problem (Abb. 7.11) sehen wir, daß die 1-Klasse zwischen den beiden für das AND und das OR gefundenen Hyperebenen (Abb. 7.10) liegt. Das liegt natürlich daran, daß sich das XOR aus einer entsprechenden AND-OR-Kombination ergibt:

$$y(XOR) \;=\; \overline{y(AND)} \;\wedge\; y(OR).$$

Wir definieren also ein dreilagiges Netz mit 2 Knoten in der Eingangslage, 2 Knoten in der versteckten Lage, 1 Knoten in der Ausgangslage (Netz-Konfiguration: 2 - 2 - 1). Die Aktivitäten der Knoten und die Gewichte sind:

$\vec{x}$: Eingangsaktivitäten,

$\vec{x}'$: Aktivitäten der versteckten Knoten,

y : Ausgangsaktivität (i.a. auch ein Vektor),

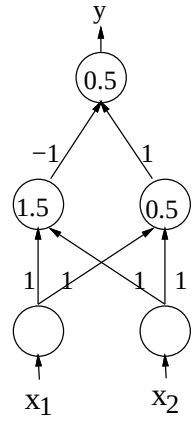
$\vec{w}_i$: Gewichte für die Eingänge ($i = 1, 2$ ist der Index der versteckten Knoten),

$\vec{w}'$: Gewichte für die Ausgänge $\vec{x}'$ der versteckten Knoten.

In Abb. 7.13 sind an die Netz-Verbindungen die Gewichte w_{i1}, w_{i2} bzw. w'_1, w'_2 und an die Knoten die Schwellen $-w_{i0}$ bzw. $-w'_0$ geschrieben. Mit der Tabelle sieht man, daß in diesem Netz die beiden versteckte Knoten jeweils das AND und OR realisieren und die Ausgangslage die logische Verknüpfung von beiden. Die 1-Klasse des Netzes liegt also zwischen den beiden Geraden in Abb. 7.13b, die 0-Klasse außerhalb.

Für das Anlernen von Netzen ist es wichtig zu sehen, daß die Lösungen für die Klassenseparation nicht eindeutig sind. In unserem Beispiel gibt es eine unendliche Schar von Hyperebenen, die kontinuierlich durch Translationen und Rotationen auseinanderhervorgehen und die, solange sie nicht einen der Musterpunkte überspringen, dasselbe leisten. Problematischer für die Kontrolle des Lernens ist allerdings, daß es auch Lösungen geben kann, die nicht kontinuierlich zusammenhängen. Für das XOR-Problem finden wir z.B. die in Abb. 7.14 angegebene Lösung, bei der die zwei Hyperebenen diesmal die 0-Klasse einschließen, während die 1-Klasse außerhalb liegt.

x_1	x_2	x'_1	x'_2	$w'_1x'_1 + w'_2x'_2$	y
0	0	0	0	0	0
1	0	0	1	1	1
0	1	0	1	1	1
1	1	1	1	0	0



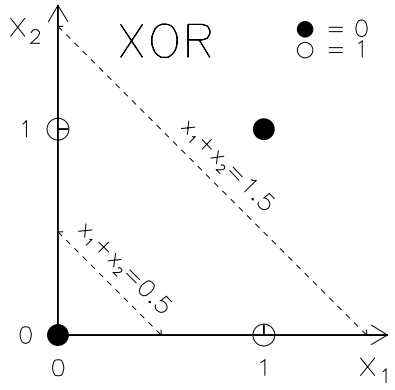
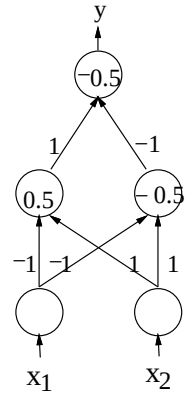


Abbildung 7.13: Links: Wahrheitstafel für das XOR-Netz auf der rechten Seite. Mitte: Netzwerk mit Gewichten und Schwellen zur Lösung des XOR-Problems. Rechts: Musterraum des XOR-Problems mit den durch das Netz bestimmten Hyperebenen.

x_1	x_2	x'_1	x'_2	$w'_1x'_1 + w'_2x'_2$	y
0	0	0	1	-1	0
1	0	0	0	0	1
0	1	1	1	0	1
1	1	0	1	-1	0



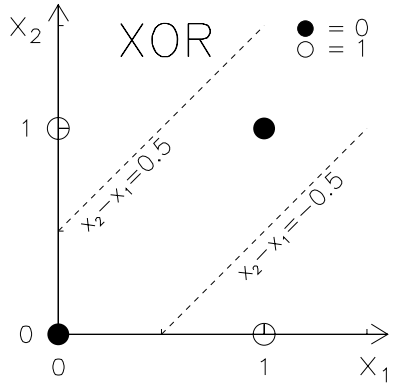


Abbildung 7.14: Links: Wahrheitstafel für das XOR-Netz auf der rechten Seite. Mitte: Netzwerk mit Gewichten und Schwellen zur Lösung des XOR-Problems (alternativ zu Abb. 7.13). Rechts: Musterraum des XOR-Problems mit den durch das Netz bestimmten Hyperebenen.

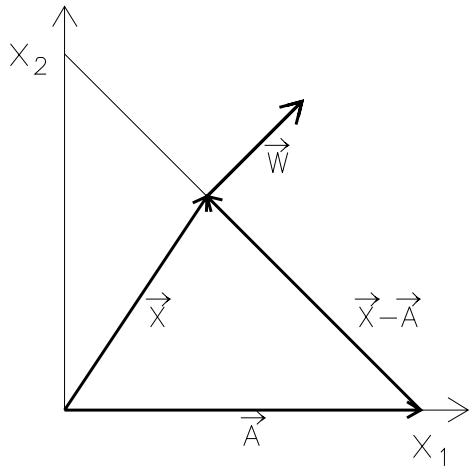


Abbildung 7.15: Zur Darstellung der Hesseschen Normalform der Geradengleichung.

Die Hessesche Normalform für die Hyperebenen:

Die Gleichung einer Hyperebene, $\vec{w}\vec{x} = 0$, ist offensichtlich invariant gegenüber einer Transformation

$$\vec{w} \to -\vec{w} \tag{7.9}$$

Dasselbe gilt aber nicht für die Klasseneinteilung durch $\vec{w}\vec{x} < 0$ und $\vec{w}\vec{x} > 0$, weil durch (7.9) die Klassen gerade vertauscht werden. Wir wollen uns deshalb die Bedeutung der Orientierung von $\vec{w}$ genauer klar machen.

Für die folgenden Überlegungen wollen wir die Gewichte und Vektoren für einen 2-dimensionalen Musterraum betrachten:

$$\begin{aligned} \vec{X} &= (x_1, x_2) \\ \vec{W} &= (w_1, w_2) \end{aligned}$$

(die großen Buchstaben sollen von den Vektoren $\vec{x}$ und $\vec{w}$ unterscheiden, die ja mit den 0-Komponenten die Schwellen enthalten). Dann ist die Gleichung der Hyperebene:

$$\vec{W}\vec{X} = -w_0,$$

sodaß auch für einen festen Ortsvektor $\vec{A}$ eines Punktes auf der Geraden gilt:

$$\vec{W}\vec{A} = -w_0$$

und damit:

$$\vec{W}(\vec{X} - \vec{A}) = 0 \tag{7.10}$$

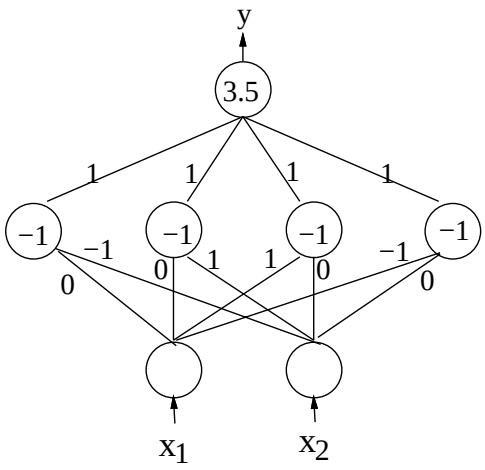
Das heißt, $\vec{W}$ steht senkrecht auf $\vec{X} - \vec{A}$ und damit senkrecht auf der Geraden, weil $\vec{X} - \vec{A}$ die Richtung der Geraden hat (Abb. 7.15). Durch die Wahl des Vorzeichens der Gewichte wird damit eine Orientierung der Normalen auf der Hyperebene festgelegt. Gleichung (7.10) ist die Hessesche Normalform der Geradengleichung (wobei genau genommen $\vec{W}$ zu normieren wäre).

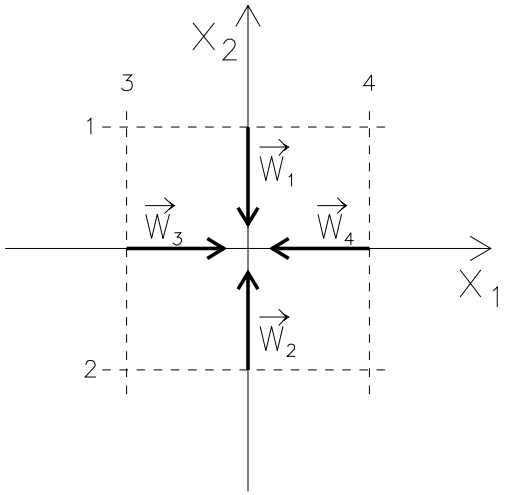
Musterklassifizierung mit einem Dreilagen-Perzeptron:

Die Punkte in dem Quadrat $[-1 < x < +1;\ -1 < y < +1]$ sollen zur Musterklasse A gehören (Abb. 7.16). Um diese Klasse zu separieren, sind dann 4 verdeckte Knoten notwendig, die jeweils eine Begrenzungsgerade festlegen (siehe Tabelle in Abb. 7.16). Wenn man die Vorzeichen so wählt, daß die Gewichtsvektoren alle in das Volumeninnere zeigen (Abb. 7.16), dann lassen sich die Ausgänge der verdeckten Knoten alle mit positiven Gewichten kombinieren, um die Klasse A zu selektieren.

Θ-Funktion als Übertragungsfunktion: Benutzt man die Θ-Funktion als Übertragungsfunktion dann wird mit den Gewichten und Schwellen in Abb. 7.16 das Quadrat exakt herausgeschnitten.

Sigmoide Übertragungsfunktion: Bei Verwendung von sigmoiden Funktionen als Übertragungsfunktion werden in der ersten verdeckten Lage die trennenden Hyperebenen immer noch scharf definiert. Im Gegensatz zu der 0-1-Entscheidung ('links' oder 'rechts' von der Hyperebene) der Θ-Funktion erhält man hier jedoch ein kontinuierliches Maß für den Abstand von der Hyperebene. Erst bei der gewichteten Summe dieser Abstände in der nächsten Stufe





i	Geraden-Gl.	w_{i0}	w_{i1}	w_{i2}	w'_i
1	$-x_2 + 1 = 0$	1	0	-1	1
2	$x_2 + 1 = 0$	1	0	1	1
3	$x_1 + 1 = 0$	1	1	0	1
4	$-x_1 + 1 = 0$	1	-1	0	1

Abbildung 7.16: Oben: a) Netzwerk mit Gewichten und Schwellen zur Selektion der Punkte innerhalb des in b) gezeigten Quadrates. Unten: Definition der Geraden und Gewichtsvektoren für das Netzwerk in der Abbildung. Der Index i steht sowohl für einen versteckten Knoten als auch für die zu diesem Knoten gehörige Gerade.

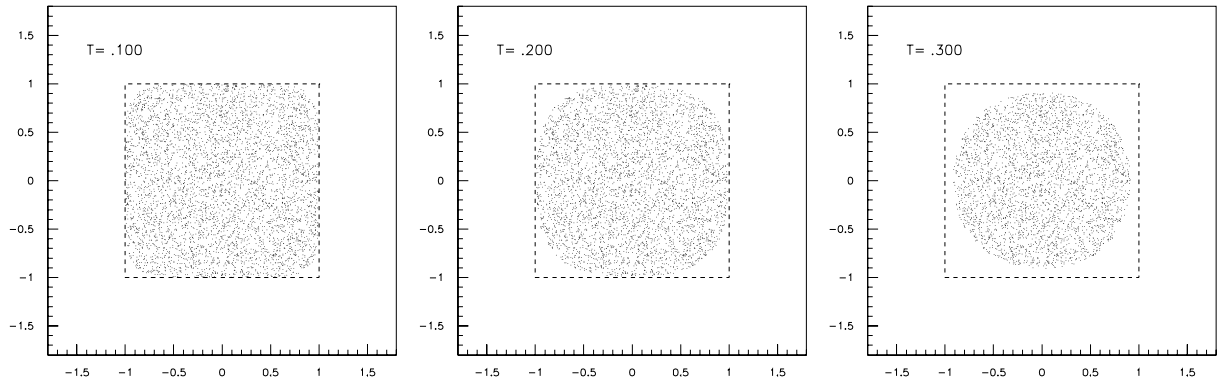


Abbildung 7.17: Durch das Netz in Abb. 7.16 selektierte Punktmenge bei Benutzung einer sigmoiden Schwellenfunktion mit Temperaturparameter a) $T = 0.1$, b) $T = 0.2$, c) $T = 0.3$.

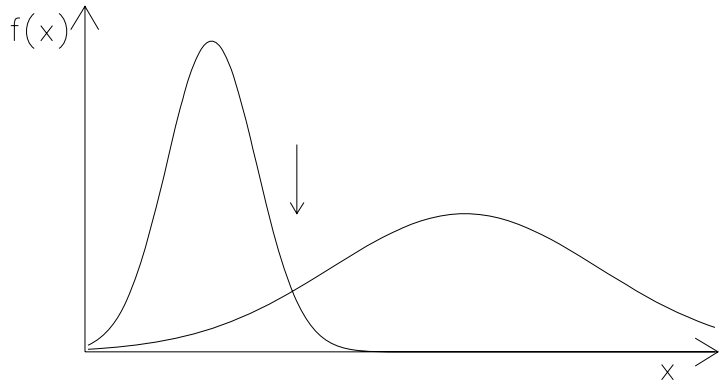


Abbildung 7.18: Beispiel für überlappende Verteilungen im Musterraum.

spielt die relative Größe der Abstände eine Rolle. In dieser Summe kann nämlich ein kleiner Abstand von einer Hyperebene einen großen Abstand von einer anderen Ebene kompensieren. Das führt zu Abrundungen von Ecken bei der Klassifikation und erlaubt i.a. die Konturen des Klassenvolumens besser zu approximieren.

In Abb. 7.17 wird gezeigt, wie sich die Kontur der selektierten Punktmenge verändert, wenn man im obigen Beispiel des Quadrates statt der Θ-Funktion die 'logistische Funktion' (7.3) mit dem Temparaturparameter $T = 1$ benutzt.

An diesem Beispiel läßt sich der Einfluß des Parameters T gut verdeutlichen: Für $T \to 0$ nähert man sich der Θ- Funktion an und damit nähert sich das ausgeschnittene Volumen mehr dem Quadrat; für $T \to \infty$ wird das Volumen abgerundeter. Trotz dieses starken Einflusses ist ein variabler T-Parameter eigentlich überflüssig: die Wirkung von T kann durch geeignete Normierung der Gewichte ebenso erreicht werden (große Gewichte ergeben scharfe Grenzen und umgekehrt). In der Lernphase kann es sich andererseits als nützlich erweisen, mit einem T-Parameter das Lernverhalten zu steuern.

7.3.3 Lernen

Die Lernstrategie:

Für Feed-Forward-Netze sind Lernstrategien entwickelt worden, bei denen das Netz mit Hilfe eines Trainingsdatensatzes lernt, die richtige Antwort zu geben. Während des Trainings kann das Netz seine Antwort mit der richtigen vergleichen; das ist also die Situation 'Lernen mit Lehrer' (supervised learning). Wenn wir Muster in Klassen einteilen wollen, erwarten wir für einen Mustervektor $\vec{x}$ folgende Antworten y_j:

$$\vec{x} \;\to\; y_j \quad \begin{array}{rl} = & 1 \quad \text{wenn } \vec{x} \text{ in Klasse } j \\ = & 0 \quad \text{sonst} \end{array}$$

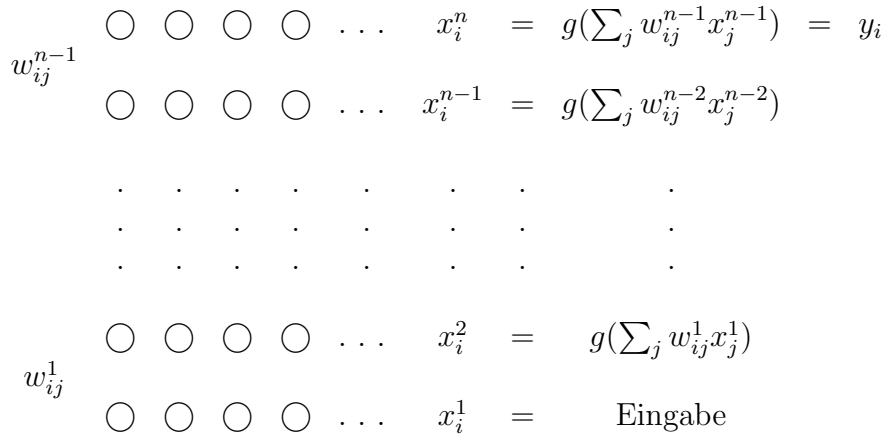
Dieses Lernziel ist sofort einsichtig, wenn die Klassen disjunkt sind. Wir wollen es aber auch beibehalten, wenn die Klassen sich überlappen wie im Fall der beiden Gauß-Verteilungen in Abb. 7.18. Wenn die Fläche unter den Kurven ein Maß für die Häufigkeit des Auftretens von Mustern der jeweiligen Klasse ist, dann ist die optimale Trennung dort, wo beide Wahrscheinlichkeiten gleich sind, d.h. der Schnittpunkt beider Kurven ('Bayes-Limes'). Wir werden sehen, daß ein wohl-trainiertes Netz diesen optimalen Limes erreichen kann.

Wie gut das Netz gelernt hat, wird mit einem dem Netz unbekannten Datensatz getestet, d.h. man prüft, ob das Netz das Gelernte auf unbekannte Daten übertragen, ob es 'generalisieren'

kann.

Lernalgorithmen:

Wir betrachten ein Feed-Forward-Netz mit n Lagen, die Ausgangsaktivitäten der k-ten Lage seien durch den Vektor $\vec{x}^k$ gegeben, die Gewichte zwischen der k-ten Lage und dem i-ten Knoten in der k+1-ten Lage sei $\vec{w}_i^k$. Das Netz hat dann folgende Struktur:



Der Trainingsdatensatz enthalte N Mustervektoren, für jedes Muster p ($p = 1, \ldots, N$) und für jeden Ausgangsknoten i sei die richtige Antwort $\hat{y}_i^{(p)}$ bekannt, die mit der Antwort $y_i^{(p)}$ des Netzes verglichen werden kann. Als Maß für die Optimierung des Netzwerkes definieren wir die Fehlerfunktion (l ist die Zahl der Ausgangsknoten)

$$E = \frac{1}{2} \sum_{p=1}^{N} \sum_{i=1}^{l} (y_i^{(p)} - \hat{y}_i^{(p)})^2 \tag{7.11}$$

Die Fehlerfunktion soll durch Variation der Gewichte w_{ij}^k minimiert werden, es muß also gelten:

$$\frac{\partial E}{\partial w_{ij}^k} = 0 \qquad k = 1, \ldots n-1 \tag{7.12}$$

Da E nicht-linear von den Gewichten abhängt, kann das Gleichungssystem (7.12) im allgemeinen nur iterativ gelöst werden. Wir wählen das für solche Optimierungsprobleme geläufige Gradientenabstiegs-Verfahren (Abb. 7.19) um das (globale) Minimum zu suchen. Es sei hier bemerkt, daß es bei multi- dimensionalen Problemen im allgemeinen sehr schwierig ist, das globale Minimum zu finden. Für unsere Anwendungen ist es aber in der Regel nicht wichtig, ob das Netz tatsächlich das globale Minimum gefunden hat, wenn es nur ein relativ gutes gefunden hat.

Die Fehlerfunktion soll also entlang des negativen Gradienten im Gewichtsraum schrittweise verkleinert werden. Dazu korrigieren wir jedes Gewicht w_{ij}^k entsprechend:

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ij}^k} \tag{7.13}$$

Wenn der Lernparameter η genügend klein ist (damit es keine Oszillationen um das Minimum gibt), kann die Korrektur nach jedem angebotenen Muster p erfolgen:

$$\Delta w_{ij}^k = -\eta \frac{\partial E^{(p)}}{\partial w_{ij}^k}$$

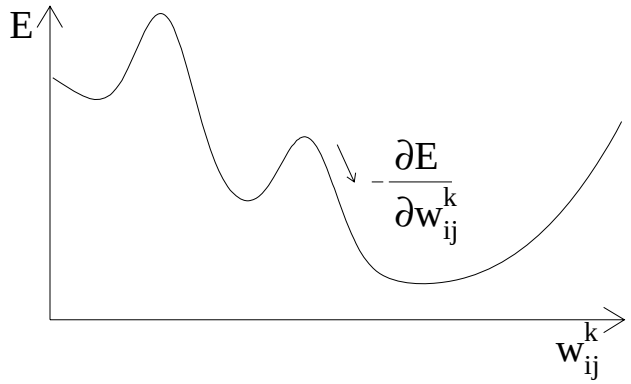


Abbildung 7.19: Beispiel für den Verlauf einer Fehlerfunktion im Gewichtsraum.

Dann stellt jedes Muster bereits einen Iterationsschritt dar; in der Regel ist dieses Verfahren schneller, als wenn man vor jeder Gewichtskorrektur erst über alle N Muster mittelt. Aus Stabilitätsgründen kann es allerdings manchmal vorteilhaft sein über eine kleine Zahl m von Mustern zu mitteln (m≈10).

Eine effiziente Methode, die Gewichtskorrekturen für die verschiedenen Lagen zu berechnen, ist der Backpropagation-Algorithmus, den wir allerdings hier aus Zeitgründen nicht näher besprechen.

Training:

Im folgenden sollen einige Begriffe, die beim Training von FF-Netzen auftreten, erläutert werden:

Trainingsdatensatz: Der Trainingsdatensatz enthält N Muster, die jeweils den Eingabevektor $\vec{x}^{(p)}$ und die erwartete Antwort $\vec{y}^{(p)}$ enthalten:

$$(\vec{x}^{(p)}, \hat{\vec{y}}^{(p)}),\ p = 1, \ldots, N \tag{7.14}$$

Lernzyklen: Im allgemeinen muß das Lernen relativ langsam erfolgen ($\eta < 1$), damit das Minimum sicher gefunden werden kann. Um zum Minimum zu kommen, muß der Trainingsdatensatz in der Regel wiederholt dargeboten werden (Lernzyklen).

Konvergenzkontrolle: Die **Konvergenz** des Verfahrens wird nach jedem Zyklus (oder nach q Zyklen) getestet durch Auswertung der Fehlerfunktion E (oder meistens E/N) oder der **Effizienz** der Selektion für jede Klasse i:

$$\epsilon_i \ = \ \frac{N_i^{net}}{N_i^{in}} \tag{7.15}$$

Dabei ist N_i^{net} die Anzahl der Muster, die vom Netz richtig in die i-te Klasse eingeordnet werden, und N_i^{in} die Anzahl der dem Netz angebotenen Muster der Klasse i. Die Effizienz sollte in einen Sättigungswert übergehen, der je nach Überlapp der Klassen zwischen 50% und 100% liegen sollte (100% kann nur für disjunkte Klassen erwartet werden). Abbildung 7.20 zeigt das erwartete Verhalten der Fehlerfunktion und der Effizienz.

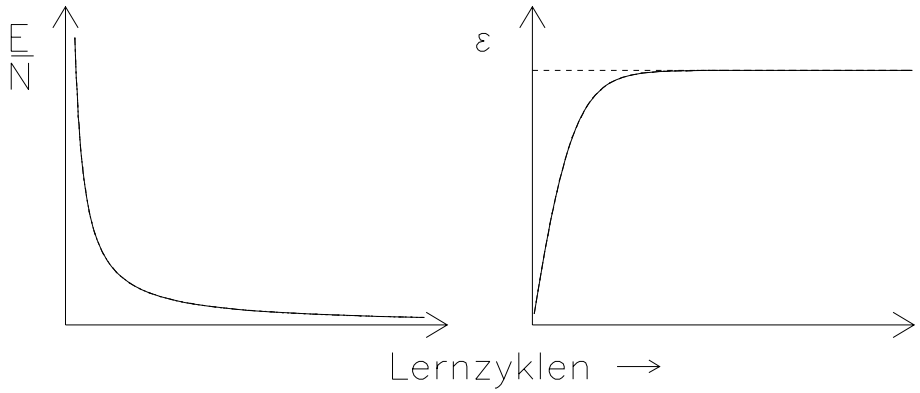


Abbildung 7.20: Kontrolle der Konvergenz: typische Verläufe der Fehlerfunktion (links) und der Effizienz (rechts).

Generalisierung: Die Bewährungsprobe für ein Netz ist schließlich der Nachweis, daß es das Gelernte auf einen ihm unbekannten Testdatensatz anwenden kann. Geprüft wird auch hier die Fehlerfunktion und die Effizienzen für die verschiedenen Klassen. Im allgemeinen sind die Effizienzen etwas niedriger und die Fehlerfunktion etwas größer als für die Trainingsdaten. Bei zu großer Diskrepanz ist zu prüfen, ob das Netz durch 'Overtraining' zu stark an die Trainingsdaten angepaßt ist. Das ist dann auch ein Hinweis, daß das Netz wahrscheinlich zuviele Freiheitsgrade hat.

Praktische Regeln zum Netzwerktraining:

Wahl von 'intelligenten' Variablen: Um gute Resultate mit Neuronalen Netzen zu erzielen, ist es in der Regel wichtig, die benutzten Variablen geschickt auszuwählen und eventuell vorzuverarbeiten.

Kontrolle von Lerngeschwindigkeit und Konvergenzverhalten: Es gibt viele verschiedene Methoden, um das Lernen, das häufig sehr zeitaufwendig sein kann, effektiver zu machen. Dazu gehört die dynamische Anpassung des Lernparameters an die Variation der Fehlerfunktion mit den Gewichten. Statistische Schwankungen im Trainigsdatensatz können durch Hinzufügen eines "Trägheitsterms", der proportional zur Gewichtsänderung im vorhergehenden Schritt ist, gedämpft werden.

Beschränkung der Komplexität eines Netzes:

Wieviele Lagen sind notwendig? Mit 2 Lagen können linear separierbare Probleme behandelt werden (siehe Lösungen der AND-, OR-Probleme mit dem Perzeptron).

Mindestens 3 Lagen werden gebraucht, wenn das Problem nicht linear separierbar ist (z.B., wenn eine Klasse in zwei disjunkten Bereichen, getrennt durch eine andere Klasse, liegen; siehe XOR-Problem). Ohne Beweis sei angegeben: Mit einem 3-Lagen-Netz kann

- jede kontinuierliche Funktion $y = f(\vec{x})$ approximiert werden,
- jede Boolsche Funktion $y = f(x_1, \ldots, x_n)$, mit $y,\ x_i = 1$ oder 0, dargestellt werden.

Wieviele Knoten pro Lage? Ein geschlossenes Volumen in n Dimensionen kann im allgemeinen durch n+1 Hyperebenen (oder weniger, wenn es zu einer oder mehreren Seiten offen ist,) eingeschlossen werden. Mehr als n+1 Hyperebenen pro geschlossenem, zu selektierendem Volumen liefert mehr Freiheit, den Konturen zu folgen (für das Quadrat ist offensichtlich n+2=4 eine bessere Wahl der Anzahl der Hyperebenen). Wir halten also fest:

- In der Regel sind mindestens n+1 Knoten in der ersten versteckten Lage notwendig.
- Die Zahl der Knoten in der zweiten versteckten Lage hängt von der Komplexität des Problems ab, insbesondere von der Anzahl der nicht-zusammenhängenden Volumina. Es ist wahrscheinlich nicht mehr als ein Knoten pro Volumen notwendig.
- Es sollten so wenig Knoten wie möglich definiert werden, um die Generalisierungsfähigkeit des Systems sicherzustellen.

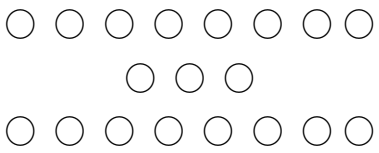
Entfernen und Generieren von Verbindungen und Knoten Um die Komplexität des Netzes so gering wie möglich zu halten, sind Techniken entwickelt worden, die erlauben, unwichtige Verbindungen und Knoten zu erkennen und zu entfernen oder auch notwendige Verbindungen und Knoten zu generieren.

Selbstgenerierung der Netz-Architektur: Bei diesem Vorgehen beginnt man zunächst mit einem sehr einfachen Netz und baut dann sukzessiv neue Verbindungen, Knoten und Lagen auf, deren Notwendigkeit dann wieder durch das Verhalten der Fehlerfunktion, der Konvergenz etc. geprüft werden kann.

7.3.4 Typische Anwendungen für Feed-Forward-Netze

Beispiel für ein binäres Netz: 8-Bit-Encoder:

Wir trainieren ein (8-3-8)-Netz



mit 8 Mustervektoren $\vec{x}^p = (x_1^p, \ldots, x_8^p),\ p = 1, \ldots, 8$, und den erwarteten Netzantworten $\hat{\vec{y}}^p = (\hat{y}_1^p, \ldots, \hat{y}_8^p),\ p = 1, \ldots, 8$, denen folgende Binärwerte zugeordnet werden:

$$\begin{aligned} x_i^p &= \delta_{ip} \\ \hat{y}_i^p &= \delta_{ip} \end{aligned}$$

Wir erwarten also das gleiche Muster am Eingang und Ausgang. Wie schafft es das Netz diese Information durch das Nadelöhr von nur 3 Knoten in der versteckten Lage zu transportieren?

Das Netz wurde mit einem PC-Programm (NNSIMU) trainiert. Die Gewichte in der ersten Schicht ergaben sich alle zu etwa $|w_{ij}| \approx 5$. Das Interessante an den Gewichten ist eigentlich nur ihr Vorzeichen, siehe Tab. 7.1. Das Vorzeichen von w_{ij} gibt in diesem Fall direkt die Aktivität des i-ten versteckten Knotens an, wenn das j-te Muster anliegt. Aus der Tabelle erkennt man sofort, daß das Netz den Binärcode 'entdeckt' hat: die redundanten 8-Bit-Sequenzen sind in 3-Bit-Binärzahlen umgewandelt worden.

Tabelle 7.1: Vorzeichen der für das Encoder-Problem gefundenen Gewichte w_{ij} in der ersten Schicht.

i $j \rightarrow$	1	2	3	4	5	6	7	8
1	-	+	-	+	+	+	-	-
2	+	-	-	-	+	+	+	-
3	+	-	+	+	-	+	-	-

a) b)

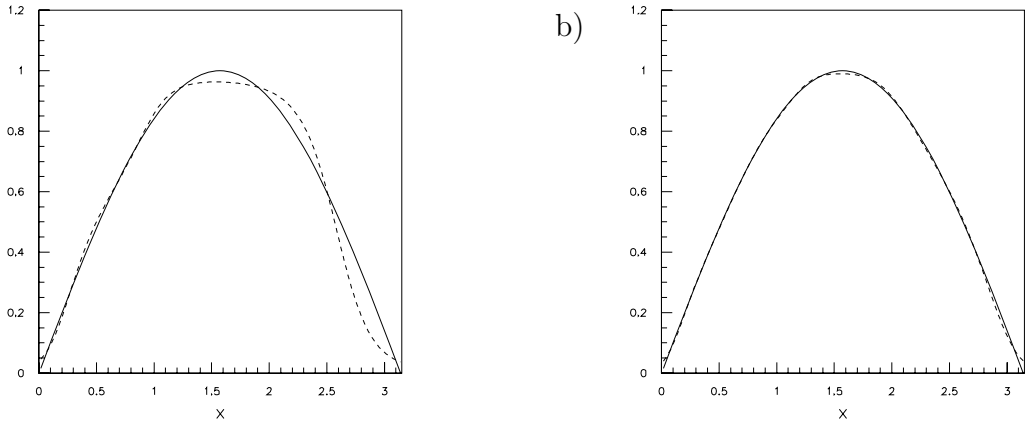


Abbildung 7.21: Approximation einer Sinus-Funktion durch ein (1-8-1)-Netz. Trainingszeiten: a) einige Sekunden, b) etwa 8 Stunden.

Funktionsapproximation:

Wie bereits in Abschnitt 7.3.3 ausgeführt, kann mit einem 3-lagigen Netz kann jede kontinuierliche Funktion,

$$\vec{x} \;=\; (x_1, \ldots, x_n) \;\rightarrow\; y \;=\; f(\vec{x}),$$

approximiert werden.

In Abb. 7.21 ist das Ergebnis eines Trainings der Funktion

$$y \;=\; \sin x, \quad 0 < x < \pi$$

gezeigt. Trainiert wurde ein (1-8-1)-Netz mit 200 Musterpaaren (x, y), äquidistant verteilt auf der x-Achse. Nach einigen Lernzyklen, entsprechend einer Rechenzeit von einigen Sekunden, ergab sich die Approximation in Abb. 7.21a. Erst nach etwa 8 Stunden wurde die ausgezeichnete Reproduktion des Sinus durch das Netz in Abb. 7.21b erzielt (diese extrem lange Zeit für ein doch relativ einfaches Problem zeigt eigentlich nur, dass das benutzte Programm nicht sehr effektiv war).

In Abb. 7.22 sind einige Zwischenwerte des Netzes als Funktion von x dargestellt. Es läßt sich gut erkennen, wie daraus die Sinus-Funktion zusammengebaut wird. Außerdem wird durch einige fast verschwindende Aktivitäten nahegelegt, daß Knoten in der versteckten Lage (z.B. der 6. und 8. Knoten) überflüssig sein könnten, die in einem nächsten Schritt entfernt werden könnten.

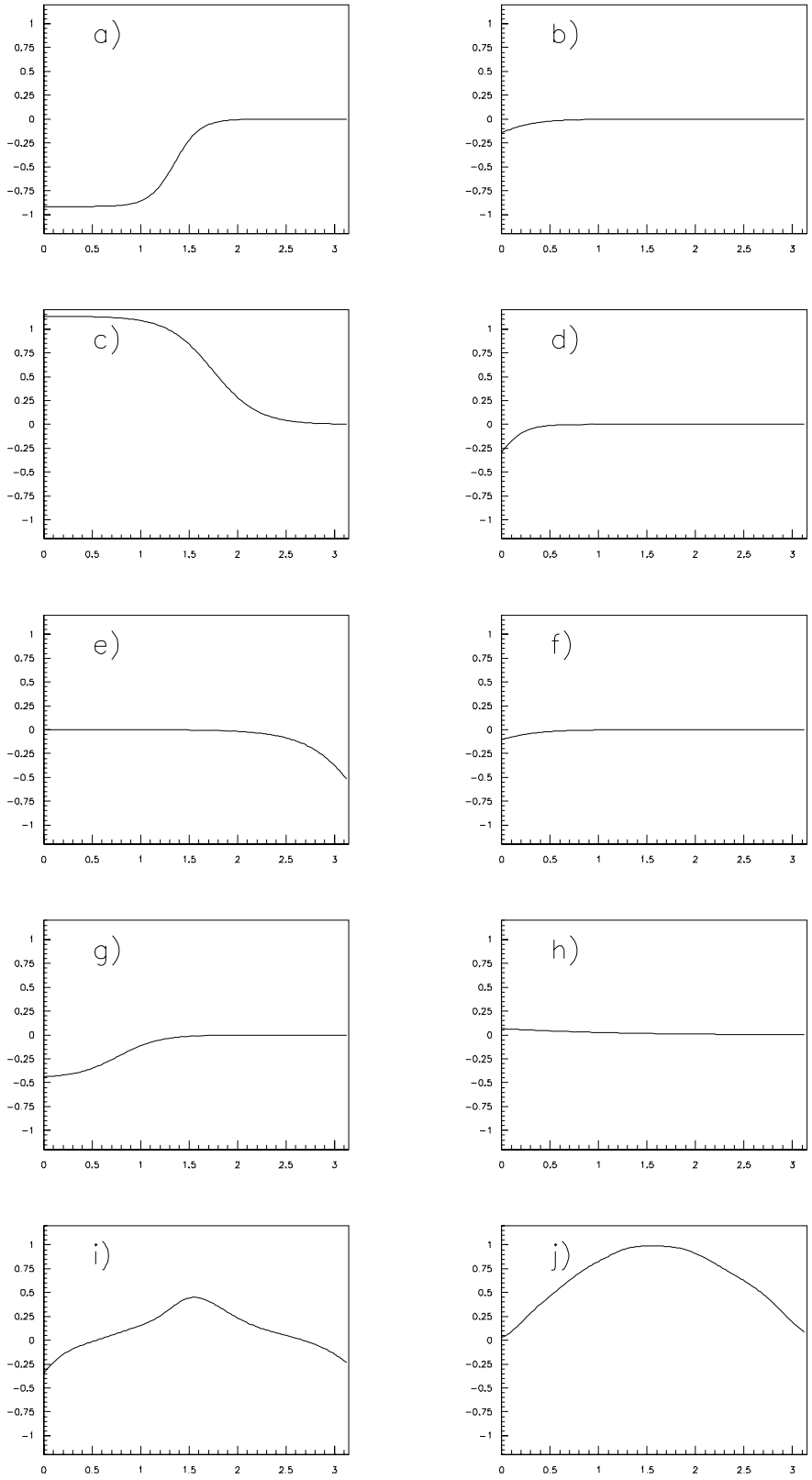


Abbildung 7.22: Für das in Abb. 7.21b benutzte Netz sind als Funktion von x dargestellt: a) bis h) die 8 gewichteten Ausgänge der versteckten Knoten $v_i = w_i' g(z_i)$; i) die Aktivität des Ausgangsknotens $z' = \sum_{i=1,\ldots,8} v_i$, j) das Ausgangssignal $y = g(z')$.

Klassifikationsprobleme:

Das Problem, Muster in verschiedene Klassen einzuordnen, tritt in unterschiedlichsten Zusammenhängen auf, zum Beispiel:

- Einteilung in disjunkte Klassen: als Beispiele mit kontinuierlichen Musterräumen hatten wir das Quadrat behandelt (siehe Abb. 7.16); Beispiele für diskrete Musterräume sind die Boolschen Funktionen (AND, OR, XOR, ...).

- Die Muster verschiedener Klassen können im allgemeinen auch in Verteilungen liegen, die sich überlappen. Ein einfaches Beispiel sind die überlappenden Gauß-Verteilungen in Abb. 7.18 (mehr dazu im nächsten Abschnitt).

  Gemeinsam ist diesen Fragestellungen, daß von einem bestimmten Muster nicht unbedingt gesagt werden kann, in welcher Klasse es liegt. Im allgemeinen kann nur eine Wahrscheinlichkeit angegeben werden, einer bestimmten Klasse anzugehören. Was die optimale Trennung ist und wie ein NN entscheidet, wird im nächsten Abschnitt besprochen.

- Mustererkennung: Eine der großen Herausforderungen für die Neuroinformatik ist die Verarbeitung und das Erkennen von visuellen, auditiven oder anderen kognitiven Mustern. Von den bisherigen Beispielen unterscheidet sich diese Problemstellung im wesentlichen durch ihre sehr viel größere Komplexität. Ein Bild beispielsweise muß in sehr viele Pixel unterteilt werden, die als Eingabe für das Netz dienen; die Netze werden damit sehr umfangreich. Ein besonderes Problem ist auch die Dynamik, durch die neben der räumlichen auch die zeitliche Dimension ins Spiel kommt. Besonders wichtige Eigenschaften der Netze sind Fehlertoleranz und Rauschunterdrückung.

7.3.5 BP-Lernen und der Bayes-Diskriminator

Die Bayes-Diskriminante:

Es seien Musterklassen C_i, $(i = 1, \ldots, m)$, gegeben. Der Bayes-Diskriminator ordnet einen Mustervektor $\vec{x}$ in diejenige Klasse C_i ein, für die die folgende Bayes-Diskriminanten-Funktion maximal ist:

$$P(C_i|\vec{x}) = \frac{p(\vec{x}|C_i)P(C_i)}{\sum_{j=1}^{m} p(\vec{x}|C_j)P(C_j)} \tag{7.16}$$

Dabei ist

$P(C_i\|\vec{x})$	(a posteriori) Wahrscheinlichkeit, daß $\vec{x}$ in Klasse C_i ist,
$P(C_i)$	(a priori) Wahrscheinlichkeit für Klasse C_i,
$p(\vec{x}\|C_i)$	Wahrscheinlichkeitsverteilung für $\vec{x}$, wenn es in Klasse C_i liegt.

Die Wahrscheinlichkeiten sind normiert:

$$\sum_i P(C_i) \;=\; 1; \qquad \int_{\Omega^n} p(\vec{x}|C_i) d^n x$$

Es ist wichtig zu beachten, daß Ω^n das 'beobachtete' Volumen ist, d.h. im allgemeinen ist die tatsächliche Verteilung noch mit einer Akzeptanzfunktion η zu korrigieren:

$$p(\vec{x}|C_i) \;\rightarrow\; p(\vec{x}|C_i)\,\eta(\vec{x}|C_i)$$

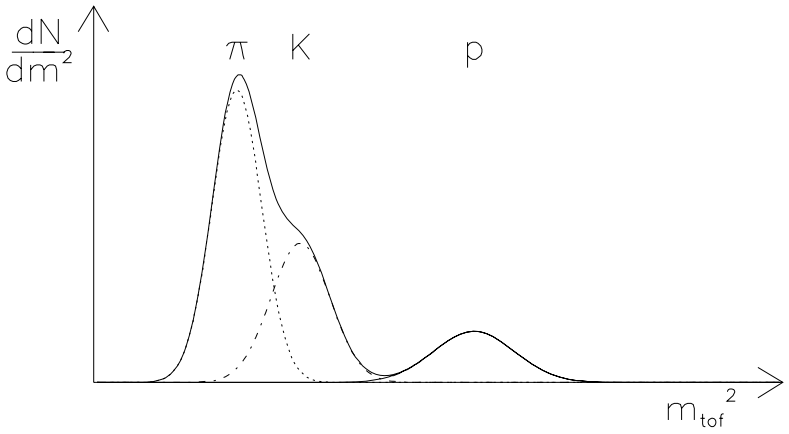


Abbildung 7.23: Typische Verteilung der Massenquadrate, berechnet aus einer Flugzeitmessung für Pionen , Kaonen und Protonen.

Beispiel: Bei der Teilchenidentifikation durch Flugzeitmessung (TOF) wird der Impuls p und die Geschwindigkeit β gemessen. Daraus läßt sich das Quadrat der Masse ('TOF-Masse') bestimmen:

$$m^2_{TOF} \;=\; p^2(\frac{1}{\beta^2} - 1)$$

Die verschiedenen Klassen entsprechen den Teilchensorten Pion, Kaon und Proton ($C_i,\ i = \pi, K, p$), die mit der Häufigkeit $P(C_i)$ auftreten. Unter der Annahme, daß m^2_{TOF} für eine Teilchensorte i Gauß-verteilt ist um die tatsächliche Masse m_i^2 des Teilchens, ergibt sich für die Verteilung von m^2_{TOF} unter der Hypothese i:

$$p(m^2_{TOF}|C_i) \;=\; \frac{1}{\sqrt{2\pi}\sigma_i} \exp \frac{-(m^2_{TOF} - m_i^2)^2}{2\sigma_i^2}$$

Ein typisches Meßergebnis ist in Abb. 7.23 gezeigt. Die Entscheidung wird dann für das Teilchen gefällt, für das die Diskriminanten-Funktion in (7.16) maximal ist.

Approximation des Bayes-Diskriminators mit neuronalen Netzen:

Ein Netz sei auf die Trennung der beiden Klassen C_1 und C_2 trainiert worden, sodaß die erwarteten Netzantworten jeweils sind:

$$\begin{aligned} \hat{y} \;&=\; 1 \quad \text{für } \vec{x} \text{ in } C_1 \\ \hat{y} \;&=\; 0 \quad \text{für } \vec{x} \text{ in } C_2 \end{aligned}$$

Dann berechnet sich der Erwartungswert der mittleren quadratischen Abweichungen der Netzantworten von den erwarteten Antworten:

$$E \;=\; \frac{1}{2} \int d\vec{x} \left[\alpha_1 p_1(\vec{x})(y(\vec{x}) - 1)^2 \;+\; \alpha_2 p_2(\vec{x})(y(\vec{x}))^2\right] \tag{7.17}$$

Das Integral geht über den gesamten Musterraum; die α_i sind die Häufigkeiten, mit denen die Klassen C_i auftreten; die $p_i(\vec{x})$ sind die Wahrscheinlichkeitsverteilungen der Muster $\vec{x}$, wenn sie jeweils einer der beiden Klassen angehören. Mit den Definitionen aus dem vorigen Abschnitt gilt dann also:

$$\begin{aligned} \alpha_i \;&=\; P(C_i) \\ p_i(\vec{x}) \;&=\; p(\vec{x}|C_i) \end{aligned} \tag{7.18}$$

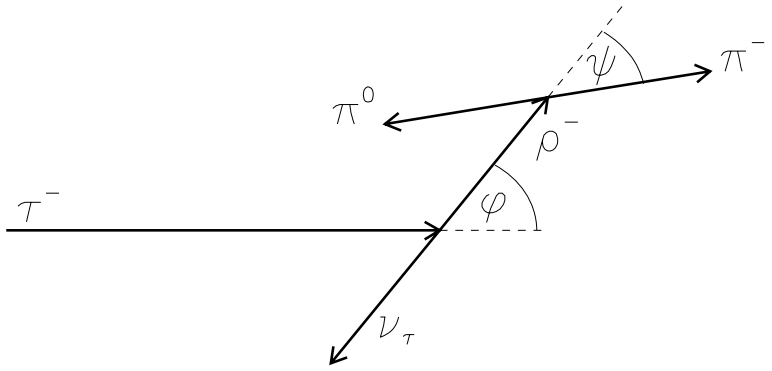


Abbildung 7.24: Darstellung der Zerfallswinkel in Reaktion (7.22).

Bei überlappenden Verteilungen können in der Fehlerfunktion (7.17) die Fehleranteile beider Klassen ungleich Null sein. Dann wird das Minimum nicht mehr unbedingt für $y = 0$ oder 1 erreicht, sondern es gibt eine optimale Wahl des Netzes für y, die sich an jeder Stelle des Musterraumes aus folgender Bedingung herleiten läßt:

$$\frac{\partial E}{\partial y} \;=\; \alpha_1 p_1(\vec{x})(y(\vec{x}) - 1) \;+\; \alpha_2 p_2(\vec{x}) y(\vec{x}) \;=\; 0 \tag{7.19}$$

Die Auflösung nach y ergibt:

$$y(\vec{x}) \;=\; \frac{\alpha_1 p_1(\vec{x})}{\alpha_1 p_1(\vec{x}) + \alpha_2 p_2(\vec{x})} \tag{7.20}$$

Die Verallgemeinerung auf m Klassen lautet:

$$y_i(\vec{x}) \;=\; \frac{\alpha_i p_i(\vec{x})}{\sum_{j=1}^{m} \alpha_j p_j(\vec{x})} \tag{7.21}$$

Das maximale y_i bestimmt, in welche Klasse das Muster einzuordnen ist. Bei zwei Klassen ist der Übergang offensichtlich gerade da, wo die beiden Wahrscheinlichkeiten gleich sind:

$$\alpha_1 p_1 \;=\; \alpha_2 p_2 \;\Longrightarrow\; y \;=\; 0.5$$

Im anschließenden Beispiel werden wir sehen, daß ein Netzwerk die optimale Lösung (7.21) approximieren kann.

Beispiel für die Approximation des Bayes-Diskriminators durch ein Netz:

Als Beispiel für die Trennung von Klassen mit unterschiedlichen, aber überlappenden Verteilungen nehmen wir die Zerfallswinkelverteilungen von τ-Leptonen in den beiden möglichen Helizitätszuständen $h = +1$ und $h = -1$ (das τ-Lepton hat Spin 1/2; die Helizität ist der auf ± 1 normierte Erwartungswert der Projektion des Spins eines Teilchens auf seine Flugrichtung). Wir nehmen an, die τ's seien in einem reinen Helizitätszustand ($h = \pm 1$) produziert worden.

Ein Zerfall, in dem sich die Spininformation im Endzustand gut messen läßt, ist der Zerfall des τ's in ein ρ-Meson mit Spin 1 und ein Neutrino mit Spin 1/2. Während das Neutrino nicht nachzuweisen ist, läßt sich die ρ-Spineinstellung über den ρ-Zerfall in zwei Pionen analysieren:

$$\tau \;\rightarrow\; \rho^- \nu_\tau \;\rightarrow\; \pi^- \pi^0 \nu_\tau \tag{7.22}$$

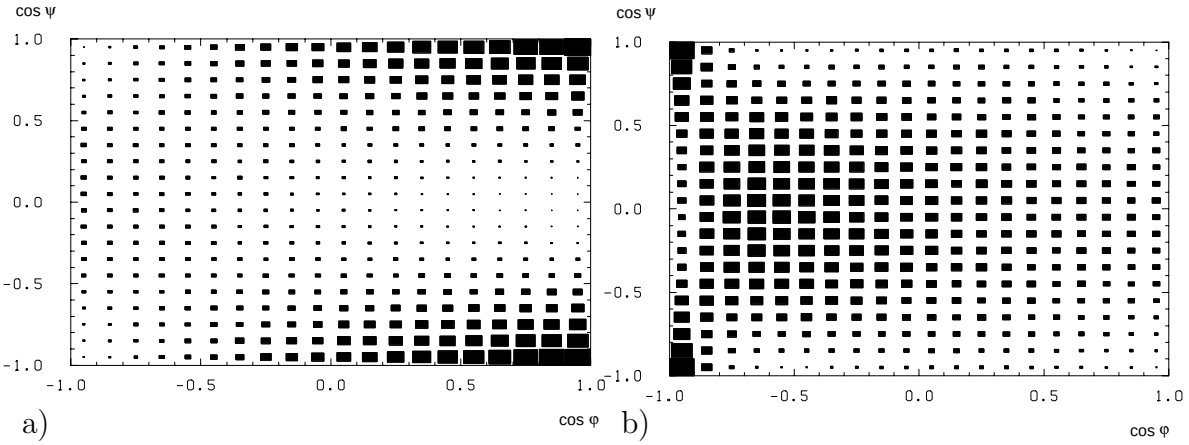


Abbildung 7.25: Winkelverteilung nach (7.23) für τ- Zerfälle im Helizitätszustand $+1$ (a) oder -1 (b).

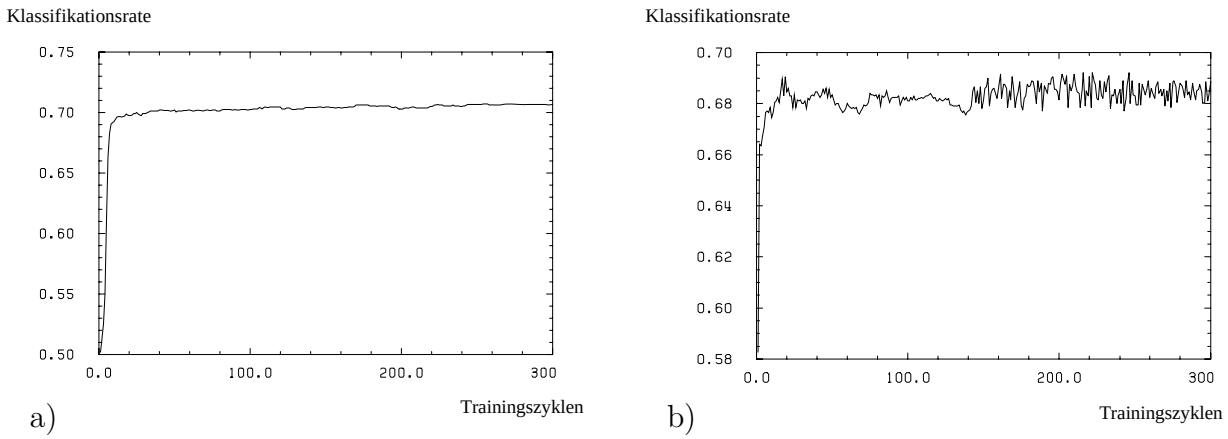


Abbildung 7.26: Effizienzen für die Zuordnung des richtigen Helizitätszustandes. Das Netz wurde mit den Lernparametern a) $\eta = 0.001$, $\alpha = 0.9$ und b) $\eta = 0.1$, $\alpha = 0.9$ trainiert.

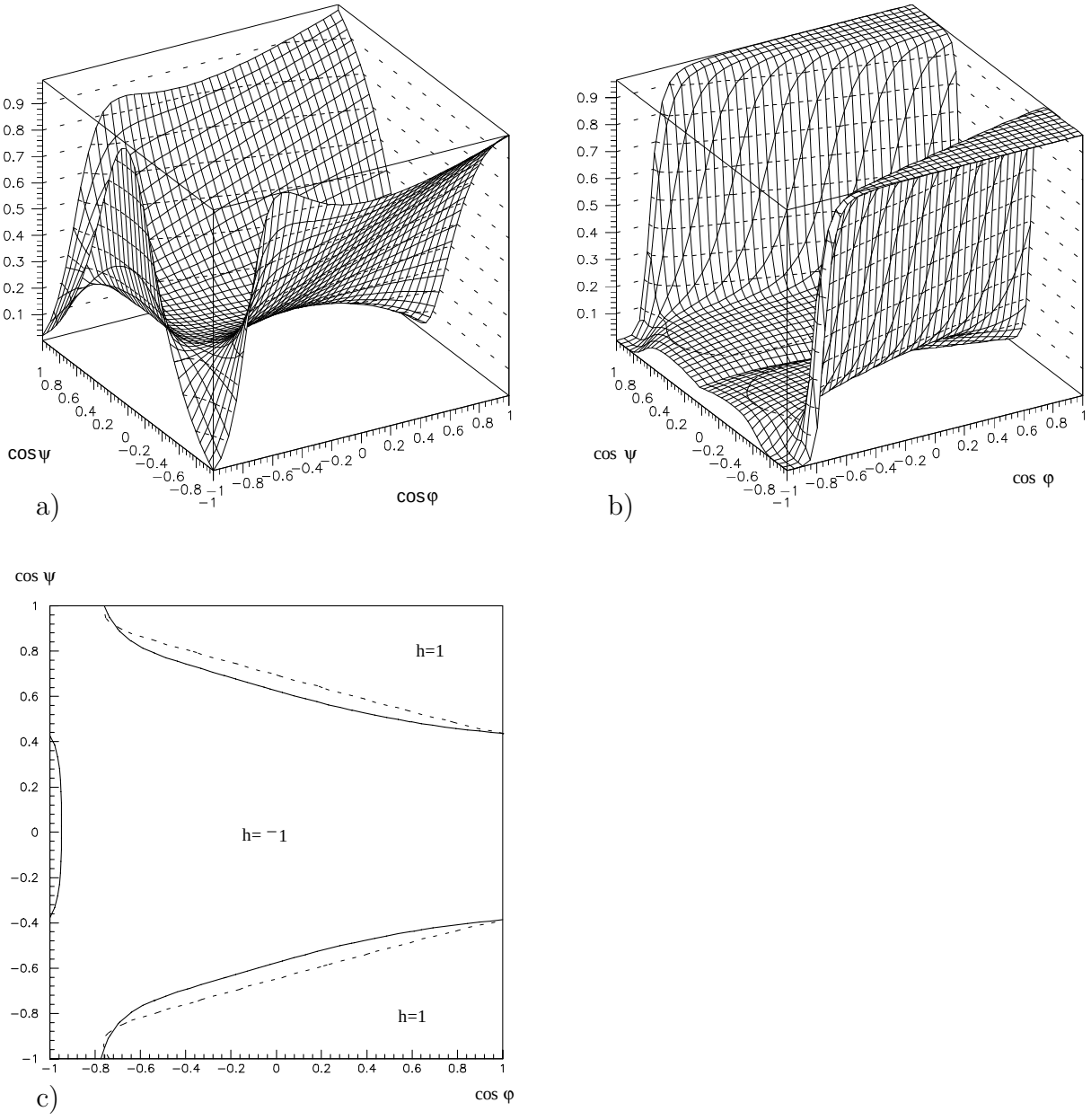


Abbildung 7.27: a) Bayes-Diskriminanten-Funktion aufgetragen über der $(\cos\phi, \cos\psi)$-Ebene; b) dasselbe für den Ausgang y des Netzes. c) Klassifikationsgrenzen für die beiden Helizitäten (volle Linie: Bayes, gepunktete Linie: Netz).

Die meßbaren Winkel sind der Winkel ϕ zwischen dem ρ und der Laborrichtung des τ (im Ruhesystem des τ) und der Winkel ψ zwischen dem π^- und dem ρ (im ρ Ruhesystem), siehe Abb. 7.24. Die beiden Winkelverteilungen sind Funktionen von $\cos\phi$ und $\cos\psi$:

$$\begin{aligned} P_{+1} &= \cos^2\psi\left[\cos\eta\cos\frac{\phi}{2}+\frac{m_\rho}{m_\tau}\sin\eta\sin\frac{\phi}{2}\right]^2 \\ &\quad +\frac{\sin^2\psi}{2}\left[\left(\sin\eta\cos\frac{\phi}{2}-\frac{m_\rho}{m_\tau}\cos\eta\sin\frac{\phi}{2}\right)^2+\left(\frac{m_\rho}{m_\tau}\right)^2\sin^2\frac{\phi}{2}\right] \\ P_{-1} &= \cos^2\psi\left[\cos\eta\sin\frac{\phi}{2}-\frac{m_\rho}{m_\tau}\sin\eta\cos\frac{\phi}{2}\right]^2 \\ &\quad +\frac{\sin^2\psi}{2}\left[\left(\sin\eta\sin\frac{\phi}{2}-\frac{m_\rho}{m_\tau}\cos\eta\cos\frac{\phi}{2}\right)^2+\left(\frac{m_\rho}{m_\tau}\right)^2\cos^2\frac{\phi}{2}\right] \end{aligned} \tag{7.23}$$

Dabei ist

$$\cos\eta = \frac{m_\tau^2-m_\rho^2+(m_\tau^2+m_\rho^2)\cos\phi}{m_\tau^2+m_\rho^2+(m_\tau^2-m_\rho^2)\cos\phi}$$

Abbildung 7.25 zeigt die sich ergebenden zwei-dimensionalen Verteilungen für die beiden Helizitäten.

Mit diesen Verteilungen wurde ein 3-lagiges FF-Netz darauf trainiert, die beiden Helizitäten zu unterscheiden. Die Netzkonfiguration war 2-8-1; der Trainingsdatensatz bestand aus 1000 Ereignissen, gleichviel von jeder Helizität. Abbildung 7.26 zeigt die Effizienz (Anzahl der richtig erkannten Ereignisse zur Gesamtzahl) in Abhängigkeit vom Lernzyklus für einen Testdatensatz. Mit dem Lernparameter $\eta = 0.001$ und dem Trägheitsparameter$\alpha = 0.9$ wird nach 300 Trainingszyklen eine Effizienz von nahezu 71% erreicht. Das kann verglichen werden mit der theoretisch berechenbaren Effizienz bei Benutzung des Bayes-Diskriminators, die sich zu 71.7% ergibt.

In Abb. 7.27 wird gezeigt, daß die Bayes-Diskriminanten-Funktion (Abb. 7.27a) von dem Ausgang y des Netzes (Abb. 7.27b) approximiert wird. Nach einem Schnitt bei $y = 0.5$ ergeben sich die Klassentrennungen, wie in Abb. 7.27c gezeigt. Ob noch eine bessere Approximation der Bayes-Trennung möglich ist, hängt neben einer ausreichenden Netzgröße auch von der Statistik des Trainingsdatensatzes ab. Es ist verständlich, daß z.B. der kleine Zipfel bei $(-1, 0)$ von dem Netz nur dann richtig eingeordnet werden kann, wenn in diesem kleinen Bereich Ereignisse liegen.