BAKKALAUREATSARBEIT

Signalverarbeitung mittels eines Neuronalen Netzwerkes für einen Smart Sensor

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Univ. Ass. Wilfried Elmenreich
Institut für Technische Informatik 182

durchgeführt von

Hubert Kraut
Matr.-Nr. 0025471, Kennzahl: e535
Buchengasse 17-19/1/1/10, A-1100 Wien

Wien, im Februar 2003 …….....

Danksagung

An dieser Stelle möchte ich mich bei Wilfried Elmenreich bedanken, der den Anstoß für dieses Projekt lieferte und maßgeblich daran beteiligt war, diese Arbeit in eine wissenschaftliche Form zu bringen.

Signalverarbeitung mittels eines Neuronalen Netzwerkes für einen Smart Sensor

Zusammenfassung

Diese Arbeit untersucht, ob man mittels eines Neuronalen Netzwerkes die Messergebnisse eines billigen Infrarotsensors so verarbeiten kann, dass die Messwerte in ein Standardlängenmass umwandelt und falsche Messwerte zu einem bestimmten Grad herausgefiltert werden können.

Für meine Experimente wählte ich, aufgrund von Geschwindigkeits- und Genauigkeitserwägungen, die Implementierung mittels eines Backpropagation-Netzwerkes.

Zu Vergleichszwecken implementierte ich noch einen konventionellen Algorithmus, der mittels linearer Approximation die Umwandlung von Sensormesswerten in cm vornahm. Da der Sensor bei Entfernungen, die über seine Reichweite hinausgehen, nur zufällige Werte anzeigt, musste ich noch einen Algorithmus implementieren, der erkennt, wenn kein Hindernis in Sensorreichweite steht.

Die Ergebnisse dieser Arbeit zeigen, dass der Einsatz von Neuronalen Netzwerken für die gewählte Aufgabenstellung nicht vorteilhaft ist. Dies ergibt sich zum Einen aus der Tatsache, dass aufgrund des Auftretens von lokalen Minima oft kein optimales Netz für das Umrechnen der Sensordaten erstellt wird, zum Anderen ist das Ergebnis des Neuronalen Netzwerkes im besten Fall nur wenig besser als die konventionelle Vergleichsimplementierung.

Inhaltsverzeichnis

1 Einleitung

Egal ob bei Mensch oder Maschine, jede Interaktion mit der Umwelt benötigt Sensoren. Klarerweise gilt, je robuster eine Applikation sein soll, desto zuverlässiger müssen die Sensoren funktionieren. Da es viele verschiedene Möglichkeiten gibt, physikalische Eigenschaften zu messen, gibt es oft große Genauigkeits-, Zuverlässigkeits- bzw. Preisunterschiede zwischen den verschiedenen Sensoren.

Was aber wichtig zu erwähnen ist, und was man vielleicht ausnützen könnte, ist die Tatsache, dass manche Sensoren nicht einfach nur falsche Ergebnisse liefern, sondern es scheint, dass hinter manchen falschen Werten ein System steht bzw., dass falsche Ausgaben nach einem bestimmten Muster erzeugt werden.

Im einfachsten Fall könnte solch ein Fehlermuster zum Beispiel daraus bestehen, dass ein Sensor in regelmäßigen Abständen ein fehlerhaftes Ergebnis liefert. Erkennt man dieses Muster, kann man einen Algorithmus implementieren, der diesen Fehler ausgleicht und dadurch einen Sensor zuverlässiger macht.

Was macht man aber, wenn dieser Fehler stark von seiner Umgebung abhängt? Zum Beispiel von der Umgebungstemperatur oder von der Spannungsversorgung. Wenn der Sensor trotz starker Abhängigkeit zu Umgebungsattributen sein grundlegendes Fehlermuster nicht ändert, kann man einen Algorithmus vorsehen, der in einer bestimmten Lernphase eine kleine Anzahl von Parametern kalibriert. Wenn aber der Sensor ein schwer modellierbares Fehlermuster an den Tag legt, dann benötigt man einen Algorithmus, der sich das Fehlermuster selber sucht, wie zum Beispiel ein selbstlernendes, mustererkennendes, Neuronales Netzwerk.

Warum also nicht einen Sensor wirklich „smart“ machen, indem man die ungefilterten Sensordaten von einem Neuronalen Netzwerk interpretieren und in benutzerfreundliche Daten umwandeln lässt?

1.1 Motivation und Aufgabenstellung

Das Ziel dieser Arbeit liegt darin, zu untersuchen, welche Möglichkeiten es gibt, mittels eines Neuronalen Netzes die fehlerhafte Ausgabe eines Infrarot-Entfernungssensors zu verbessern. Motiviert wurde diese Arbeit durch die Dissertation von Wilfried Elmenreich [El02], der in einem mobilen Roboter Sharp GP2D02-Infrarotsensoren verwendet hat. In dieser Arbeit wurden unter anderem diese Infrarotsensoren auf ihre Genauigkeit und Zuverlässigkeit hin untersucht.

Der Infrarotsensor, der auch für die Untersuchungen der vorliegenden Arbeit ausgewählt wurde, lässt sich durch die folgenden Angaben charakterisieren:

- misst Entfernungen zwischen 10cm und 120cm
- Messergebnisse werden als Integer im Bereich von 30 – 260 geliefert
- je kleiner eine Entfernung desto größer ist der Messwert
- die Ausgangskennlinie verläuft stark nicht-linear

- die Ausgangskennlinie ist stark abhängig von der Stromversorgung und von Temperaturschwankungen
- in unregelmäßigen, zeitlichen Abständen kommt es zu einem kurzzeitlichen, deutlichen Abfall des Sensormesswertes
- bei Entfernungen größer als 120cm zu einem Gegenstand werden zufällige Messwerte vom Sensor geliefert.

(genauere Angaben kann man dem *Anhang A*: *Datenblatt zu GP2D02* entnehmen)

Aufgabenstellung ist es, die selbstlernenden, mustererkennenden Fähigkeiten eines Neuronalen Netzes zu nutzen, um die Ausgangspegel des Sensors in cm umzuwandeln.

2 Grundlagen und Konzepte

In diesem Kapitel werden oft verwendetet Ausdrücke erklärt und Grundlagen für die späteren Kapitel geschaffen.

2.1 Smart Sensor

Ein *Sensor* ist ein Gerät, das physikalische Reize in elektrische Impulse umwandelt, die von einem Computersystem ausgewertet werden können - oder etwas anders ausgedrückt, ein Sensor wandelt analoge, physikalische Attribute in elektronische Signale um. Im Gegensatz dazu ist ein *Aktuator* ein Gerät, das auf physikalische Attribute einwirkt. Meist werden Sensoren und Aktuatoren unter dem Begriff *Transducer* zusammengefasst.

Ein *Smart Sensor*, erstmals beschrieben unter der Bezeichnung *Intelligent Transducers* in [Ko82], ist die Integration eines analogen oder digitalen Sensors mit einer Verarbeitungseinheit und einer Kommunikationsschnittstelle. Das ungefilterte Sensorsignal wird von der Verarbeitungseinheit überprüft, kalibriert, in eine standardisierte Form gebracht und über ein standardisiertes Kommunikationsprotokoll verschickt. Abbildung 2.1 zeigt eine schematische Darstellung eines Smart Sensors.

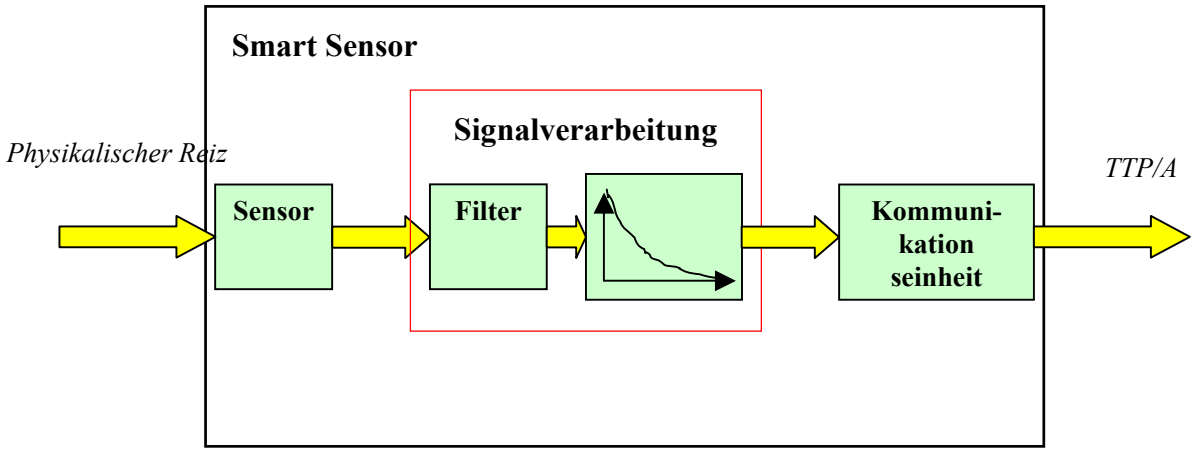


Abbildung 2.1: Konzept eines Smart Sensors

Smart Sensors werden meist sehr kompakt, stromsparend und aus billigen Standardkomponenten aufgebaut. Aus diesem Grund muss die darauf eingebettete Software sehr ressourceneffizient implementiert werden. Typischerweise bietet ein Smart Sensor nur einige kB ROM und weniger als 1kB RAM für Anwendungsprogramme. Abbildung 2.2 zeigt ein Beispiel für die Implementierung der Signalverabeitungs- und der Kommunikationseinheit eines Smart Sensors mittels dem „AT90S8515 Mikrocontroller“. Diese Variante wurde für die Aufnahme der Testreihen verwendet, die in dieser Arbeit ausgewertet wurden.

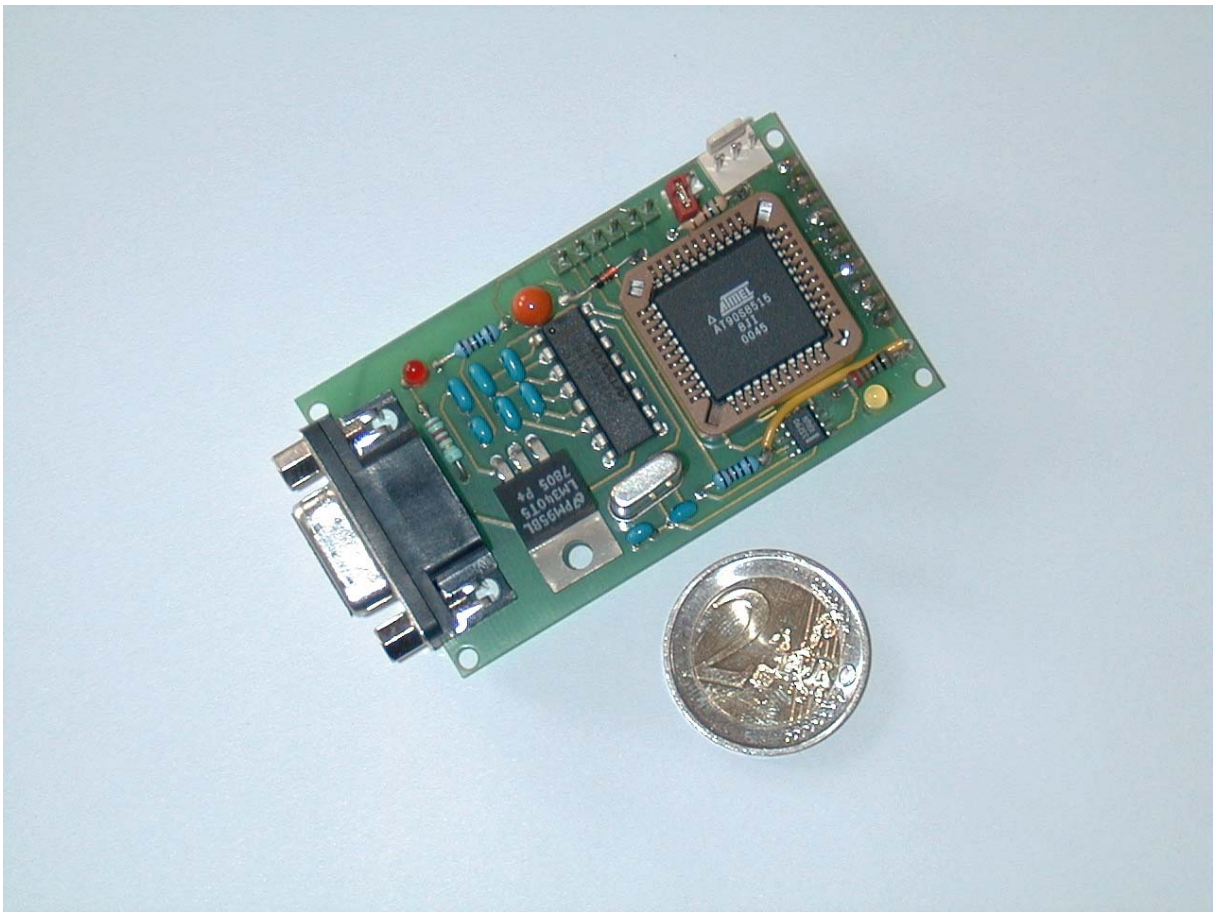


Abbildung 2.2: Kommunikations- und Signalverarbeitungseinheit eines Smart Sensors (basierend auf einem AT90S8515 Mikrocontroller)

2.2 Künstliche Neuronale Netze

Mit künstlichen Neuronalen Netzen werden biologische Neuronale Netze als informationsverarbeitende Systeme nachgeahmt. Künstliche als auch biologische Neuronale Netze bestehen aus vielen, mit einander verbundenen *Neuronen*, im weiteren Text werden diese Knoten *Units* genannt. Je nach Netzarchitektur können diese Units Signale verschiedenartig verarbeiten

> *Der wesentliche Unterschied zwischen Neuronalen Netzen und herkömmlichen algorithmischen Modellen ist, dass erstere eine Struktur und ein Lernverfahren vorgegeben bekommen. Es wird nicht für jedes Problem ein spezielles Programm geschrieben, sondern das Netz selber muss in einen Lernprozess die richtige Konfiguration finden.*[Ro93]

Grundsätzlich werden die Units eines Neuronales Netzwerkes in eine Eingabeschicht, eine Ausgabeschicht und, je nach Netztopologie, beliebig viele Zwischenschichten strukturiert. An die Eingabeschicht werden die Inputsignale angelegt und an der Ausgabeschicht werden die Berechnungsergebnisse ausgegeben. Die Units sind mit gerichteten Kanten verbunden. Das heißt, wenn beispielsweise eine Kante von Unit A zu Unit B geht, fließt in die Berechnung des internen Zustandes der Unit B der Ausgabewert der Unit A ein.

In der nachfolgenden Übersicht soll betrachtet werden, in welchen Anwendungsgebieten künstliche Neuronale Netze in der Praxis verwendet werden. Dabei handelt es sich in den meisten Fällen um Anwendungsgebiete, die mit den Methoden der „konventionellen Informatik" nur mit großem Aufwand in den Griff bekommen werden. Neuronale Netze werden zur Bewältigung dieser Aufgaben nicht im klassischen Sinn programmiert, sondern werden vielmehr auf ihre Aufgabe trainiert.

- **Klassifizierung:** Bei der Klassifizierung wird für jedes Eingangsmuster durch das Neuronale Netz entschieden, zu welcher Klasse von Eingangssignalen es gehört. Ein klassisches Beispiel für die Klassifizierung ist die Mustererkennung (Pattern Recognition). Dabei wird versucht, in verrauschen Eingangssignalen bestimmte Strukturen zu identifizieren. Bei der Schrifterkennung wird z.B. versucht, eine Pixel-Matrix (Eingangssignal) einem Buchstaben des Alphabets (Klasse) zuzuordnen. [Lu97]
- **Assoziativspeicher:** Bei dieser Art des Speicherzugriffs wird nicht eine bestimmte Speicheradresse (Inputvektor) auf den Inhalt einer Speicherzelle abgebildet, vielmehr wird die „Umgebung" der Speicheradresse auf die Speicherzelle abgebildet. Für den Fall, dass der Eingangsvektor x mit der Speicherzelle y assoziiert wird, soll auch die Eingabe x' zu der Speicherzelle y führen, wenn x' in der Umgebung von x liegt. Auf diese Weise können z.B. verrauschte Eingaben dem richtigen Ausgangswert zugeordnet werden. [Ro93]

Zusätzlich zu den Eigenschaften der individuellen Neuronen, werden die Eigenschaften eines Neuronalen Netzes auch durch folgende Charakteristika bestimmt: [Lu97]

- **Netzwerktopologie:** Die Topologie des Netzwerkes bezeichnet das Muster der Verbindungen zwischen den einzelnen künstlichen Neuronen. Meistens werden die Neuronen in Schichten (Layer) angeordnet.
- **Lernalgorithmus:** Darunter versteht man jenen Algorithmus, mit dem das Netzwerk auf seine zukünftige Aufgabe trainiert wird. Im weiteren Text wird der *Backpropagation*-Algorithmus (= Fehlerrückführungs-Algorithmus) als grundlegender Algorithmus für das Lernen in Neuronalen Netzen vorgestellt.

2.2.1 Künstliche Neuronen

Die Neuronen stellen die Berechnungselemente des Neuronalen Netzes dar und werden auch *Units* oder *Knoten* genannt. Jede Unit kann beliebig viele unabhängige Eingänge haben und hat genau einen Ausgang. Die Kanten des Netzwerkes sind gerichtete Informationskanäle, die die Information von einem Neuron zum anderen transportieren. Wenn an einem Knoten n ankommende Verbindungen angeschlossen sind, transportiert jede davon ein Argument für die Funktionsauswertung an diesem Knoten.

Ein Neuron besitzt folgende Eigenschaften:
- Erregungszustand bzw. Aktivierungszustand: kann diskrete oder kontinuierliche Werte annehmen und repräsentiert den internen Zustand des Neurons
- Aktivierungsfunktion: berechnet aus der Summe der gewichteten Eingänge den Erregungszustand
- Outputfunktion: berechnet aus dem Erregungszustand einen Outputwert
- Input: alle Kanten, die zur Unit führen
- Output: die Kante, die von der Unit weg führt

Im weiteren Text werden folgende Vereinfachungen für das Neuronenmodell angenommen:
- die Outputfunktion wird weggelassen

- Ausgabewert e_i = Aktivierungszustand u_i

Das im weiteren Text verwendetet Neuronenmodell wird in *Abbildung 2.3* dargestellt. (übernommen aus [Lu97])
Die Unit besitzt *J* Eingänge, Kanten die mit anderen Units verbunden sind oder Kanten über die ein Netzwerkinput geschickt wird. Jeder dieser Kanten besitzt ein Gewicht w_{ij} im Intervall [0,1]. Der Netto-Input u_i berechnet sich durch $u_i = \Sigma(w_{ij}*s_j)$, daher die Summe aller Output-Werte der Units die mit der Unit i verbunden sind, wobei diese Output-Werte mit dem jeweiligen Kantengewicht multipliziert werden.
Der Aktivierungszustand der Unit wird über die Aktivierungsfunktion durch $e_i = \alpha$(Nettoinput) berechnet.
Bei diesem vereinfachten Model wird der Ausgabewert nicht durch eine Ausgabefunktion bestimmt, sondern es wird der Aktivierungszustand als Ausgabe verwendet.

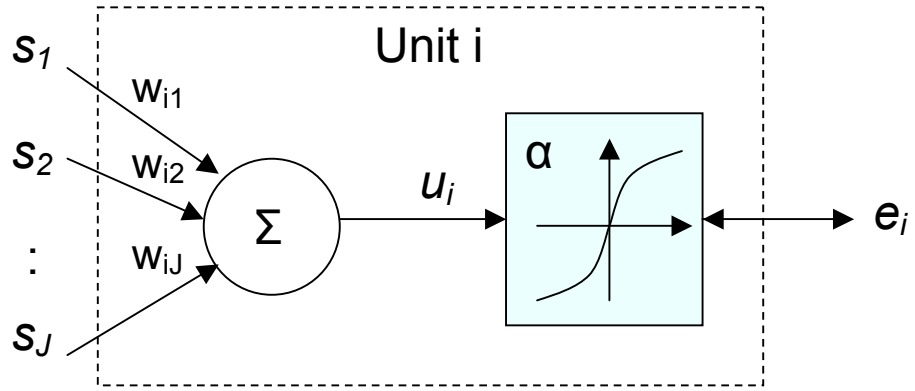


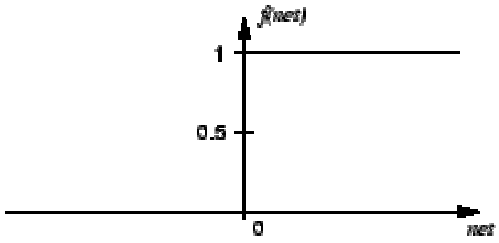
Abbildung 2.3: Einfaches Neuronenmodell für Neuronale Netze, s_j: Eingänge, w_{iJ}: Übertragungsgewichte, Σ Summation, u_i: Nettoinput = $\Sigma(w_{ij}*s_j)$, α: Aktivierungsfunktion, e_i: Aktivierungszustand = $\alpha(u_i)$

2.2.2 Aktivierungsfunktionen

Die Aktivierungsfunktion bestimmt, abhängig vom Netto-Input, den Output des Neurons. Dieser Output kann das Eingangssignal für ein anderes Neuron sein oder das Ausgangssignal des Neuronalen Netzes. Als Aktivierungsfunktion wird meist eine der nachfolgenden Grundtypen verwendet:

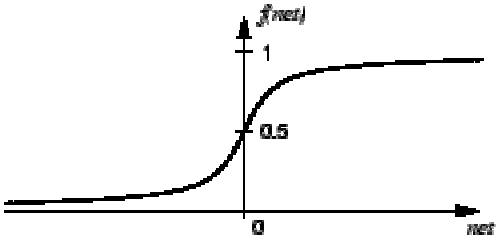
- **Schwellwertfunktion:** Der Funktionsverlauf einer Schwellwertfunktion ist in *Abbildung 2.4* graphisch dargestellt. Neuronen (Gatter), die diese Aktivierungsfunktion verwenden, nennt man Schwellwertgatter. Das im weiteren Text vorgestellte Perzeptron, stellt ein solches Schwellwertgatter dar.
- **Sigmoide Funktion:** Der Funktionsverlauf einer sigmoiden Funktion ist in *Abbildung 2.5* graphisch dargestellt. Der Unterschied der sigmoiden Funktion zur Schwellwertfunktion ist, dass diese ein kontinuierliches Ausgangssignal liefert und differenzierbar ist. Erst dadurch werden Lernverfahren wie das Backpropagation-Verfahren, das im weiteren Text vorgestellt wird, möglich. Gatter, die diese Aktivierungsfunktion verwenden, nennt man sigmoide Gatter.
- **Lineare Funktion:** Der Funktionsverlauf einer linearen Funktion ist in *Abbildung 2.6* graphisch dargestellt. Die lineare Funktion stellt die einfachste und am wenigsten mächtige Aktivierungsfunktion dar. Gatter die diese Aktivierungsfunktion verwenden, nennt man

Lineargatter. Bei der linearen Aktivierungsfunktion ist zu beachten, dass eine Hintereinanderschaltung mehrerer linearer Gatter nur dieselbe Funktion berechnen kann, wie ein einzelnes Gatter. Der Grund dafür ist, dass jede Hintereinanderschaltung von linearen Funktionen wieder linear ist.



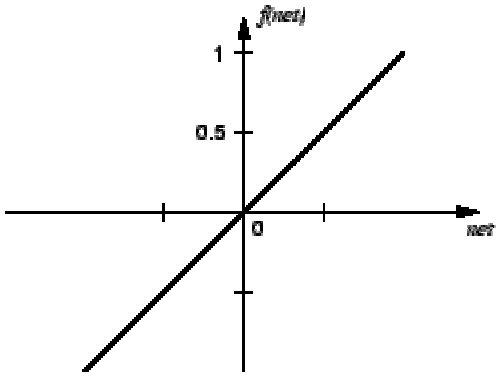
$$f(net) = \begin{cases} 0 : net < 0 \\ 1 : net \geq 0 \end{cases}$$

Abbildung 2.4: Schwellwertfunktion als Aktivierungsfunktion



$$f(net) = \frac{1}{1 + e^{-net}}$$

Abbildung 2.5: Sigmoide Funktion als Aktivierungsfunktion



$$f(net) = net$$

Abbildung 2.6: Lineare Funktion als Aktivierungsfunktion

2.2.3 Netzwerktopologien

Laut Mazzetti [Ma92] lassen sich folgende Eigenschaften von Modellen Neuronaler Netze unterscheiden (*Abb. 2.7*):

- **Vorwärtsgekoppelte Netze** = Netze, in denen Verbindungen nur in eine Richtung gehen, vom Input zum Output (*Abb. 2.7a*). Dies ist das Gegenteil der zyklischen Netze.

- **Vollkommen verbunden Netze** = Netze, in denen jedes Element mit allen anderen verbunden ist (meistens bis auf sich selbst). (*Abb. 2.7b*) In vollkommen verbundenen Netzen werden nicht verbundene Elemente durch eine Kante mit Gewicht 0 dargestellt.
- **Geschichtete Netze** = Netze in denen nicht verbundene Units in Schichten organisiert werden (*Abb. 2.7c*). Man unterscheidet zwischen der Input-Schicht, an die die Eingabe von außerhalb des Netzes angelegt wird, der Output-Schicht, die für die Ausgabe nach außerhalb des Netzes verwendet wird, und den „Versteckten Schichten", die zwischen Input- und Output-Schicht liegen.
- **symmetrische Netze** = Netze, in denen die Verbindung zwischen zwei beliebigen Elementen in beiden Richtungen gleich ist. Es gilt daher $W_{ij}=W_{ji}$ (W_{ij} = Gewicht der Kante von Unit j zu Unit i) (*Abb. 2.7d*).
- **Selbstassoziative Netze** = *Netze, in denen Input- und Output-Elemente übereinstimmen. Die Aufgabe solcher Netze ist es, einen Reiz aus der Umgebung zu bekommen, ihn zu entwickeln und als Ergebnis eine modifizierte Version des Inputs zu geben*.[Ma92] *(Abb. 2.7e)*
- **Asynchrone Netze** = Netze, in denen die Aktivierungszustände der Units in zufälliger Reihenfolge abgearbeitet werden.

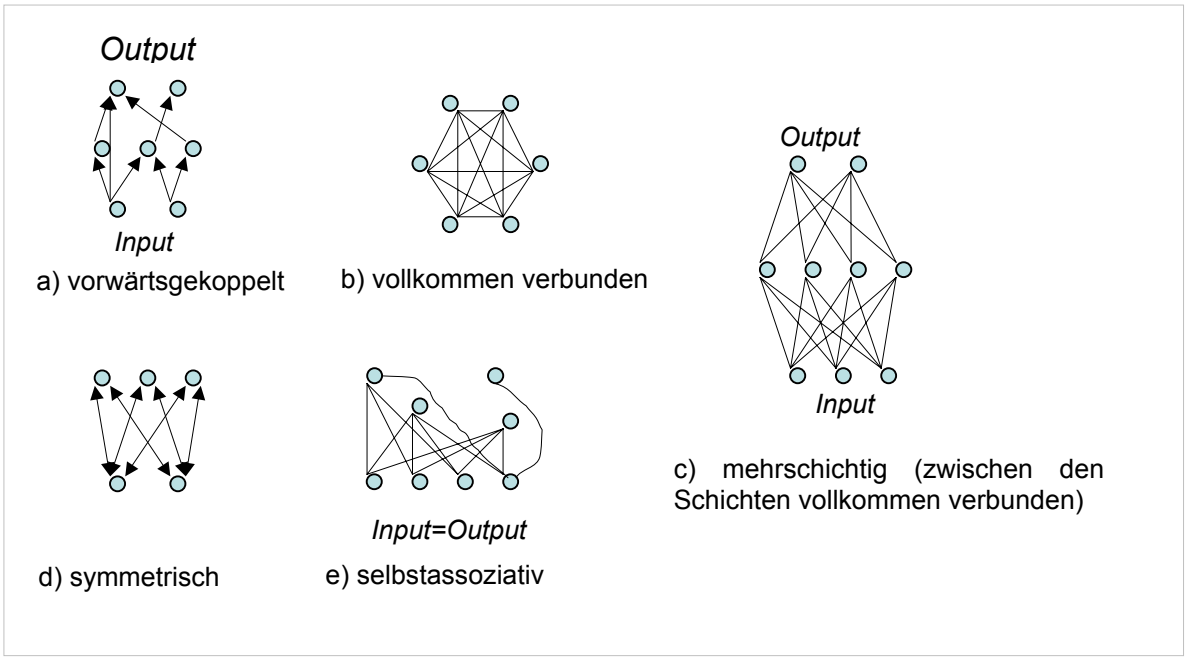


Abbildung 2.7: Modelle Neuronaler Netze

2.2.4 Lernmethoden

Es existieren zwei Lerngrundmethoden ([Ma92]):

- **mit Unterweisung = überwachtes Lernen:** dem Netz werk werden in der Lernphase Input/Output-Paare gezeigt. Es geht darum eine Funktion Y=F(X) zu lernen.
- **ohne Unterweisung = nicht überwachtes Lernen**: das Netzwerk bekommt in der Lernphase nur Input-Vektoren vorgesetzt und hat die Aufgabe, die beobachteten Beispiele zu organisieren und einzuordnen und Regelmäßigkeiten zu entdecken.

2.2.5 Trainieren mittels überwachtem Lernen

Um ein Neuronales Netz dazu zu bringen, die gewünschte Funktion auszuführen, muss es zuerst „programmiert werden. Im Gegensatz zu einem konventionellen Programm wird aber

nicht ein bestimmter Lösungsalgorithmus vorgegeben, sondern es wird dem Neuronalen Netz nur „gezeigt“, welche Ausgaben man bei bestimmten Eingaben erwartet. Man trainiert das Netz sozusagen mit Input/Output-Paaren und überlässt ihm selbst den Aufbau der internen Struktur.

Die Grundidee des Neuronalen Lernens ist folgende: für jedes gelieferte Beispiel ist die Verbindung der Elemente zu verstärken, die gleichzeitig aktiv sind. Das bedeutet, wenn dem Neuronalen Netz ein Input/Output-Beispiel vorgestellt wird, wird das Gewicht W_{ij} der Verbindung zwischen der Unit i und der Unit j proportional zum Aktivierungspegel der Elemente i und j gesteigert. Mathematisch wird diese Regel durch die so genannte *Hebbsche Formel* ausgedrückt:

(Hebbsche Lernregel) $$\Delta W_{ij} = \varepsilon * O_i * O_j \tag{2.1}$$

wobei ε *Proportionalitäts- oder Lernkonstante* genannt wird und die Lerngeschwindigkeit steuert. O_i und O_j stellen die Aktivierungspegel der zwei miteinander verbundenen Units dar. Diese Regel hat 2 Mängel: sie ändert immer die Gewichte, auch wenn dies nicht nötig wäre und, umgekehrt, ändert sie in den härtesten Fällen die Gewichte nicht genügend. Eine bessere Möglichkeit, gezeigt in [Ma92], besteht darin, wiederholt eine Formel, die schrittweise jedes Gewicht in Richtung des optimalen Wertes korrigiert, anzuwenden. Nach jedem Schritt wird das Beispiel erneut getestet und die Abweichung festgestellt. Ist diese über einem bestimmten Schwellwert, so werden die Gewichte der Elemente, die einen Fehler gemacht haben, um den Betrag, der den Fehler kompensiert korrigieren. Dieser Vorgang wird solange wiederholt, bis der Fehler genügend klein geworden ist. Jeder Zyklus, in dem immer wieder die gleichen Beispiele vorgetragen werden, heißt Lern-Schritt. Für einen Lernschritt lautet dann die Formel:

(Delta-Lernregel) $$\Delta W_{ij} = \varepsilon * (D_i - O_i) * O_j \tag{2.2}$$

wobei D_i den gewünschten Wert von O_i angibt, D_i-O_i der Fehler des i-Elements ist, O_j der Aktivierungspegel von der verbundenen Unit j ist und ε die, bereits in Gleichung 2.1 verwendete, Lernkonstante darstellt. Wichtig: Ein Neuronales Netz kann nie gleichzeitig trainiert und für Berechnungen angewendet werden!

2.2.6 Zusammenfassung

Die Hauptmerkmale der Neuronalen Netze:

- Laut dem Theorem von Hecht-Nilsson [Ni90] ist es möglich, jede beliebige Funktion Y=F(X) von einem vorwärtsgekoppelten, dreischichtigen Neuronalen Netzwerk mit einer angemessenen Anzahl von Elementen in der Zwischenschicht ohne irgendwelche Verbindung im Inneren der Schicht, aber mit vollständigen Verbindungen zwischen den Schichten sehr präzise nachzubilden.
- Ein Neuronales Netzwerk wird trainiert. Anstatt eines vorgegebenen Algorithmus fordert das Training eine Menge spezifischer Beispiele
- Ein Neuronales Netz kann nie trainiert und gleichzeitig für Berechnungen angewendet werden.
- Schwächen der Neuronalen Netze: mangelhafte Rechenleistung, mangelhafte logische Fähigkeiten, niedrige Genauigkeit, Unfähigkeit, die gelieferten Ergebnisse zu erklären

- Stärken der Neuronalen Netze: assoziative Fähigkeiten (Analogien bilden, Ähnlichkeiten finden), Verallgemeinerungsfähigkeit, Unempfindlichkeit gegenüber Störungen und Defekten, Vertrautheit mit häufigen Ereignissen.

3 Bestehende Ansätze

Die Ansätze die in diesem Kapitel besprochen werden haben alle die Gemeinsamkeit, dass sie mit *überwachtem* Lernen arbeiten. Damit ist gemeint, dass zu erlernende und damit zu assoziierende Muster dem Netz bekannt sind. Man spricht in diesem Fall von einer Klassifizierung oder von überwachtem Lernen.
Im Gegensatz dazu existiert nicht überwachtes Lernen, bei dem nur Muster (ohne zugehörige Klasse) vorliegen, und das Netz soll aufgrund der, in den Muster vorhandenen Gemeinsamkeiten, diese selbsttätig entsprechenden Klassen zuordnen. Netze mit diesem Lerntyp sind für die Aufgabe der Interpretation von Sensordaten nicht geeignet und werden deshalb im weiteren Text nicht weiter betrachtet.

3.1 Muster-Assoziator

3.1.1 Aufbau des Netzes

Ein Muster-Assoziator ist ein einschichtiges, vorwärtsgekoppeltes Netz mit N_E Eingängen und N_A Ausgängen. Jedes Neuron ist mit allen Eingängen verbunden und gibt seinen Ausgang lediglich an den Netzausgang weiter. Daher ist die Anzahl der Neuronen gleich der Ausgangsanzahl. Die *Abbildung 3.1* zeigt einen Muster-Assoziator mit 3 Eingängen $E_1...E_3$ und 4 Ausgängen $A_1...A_4$.

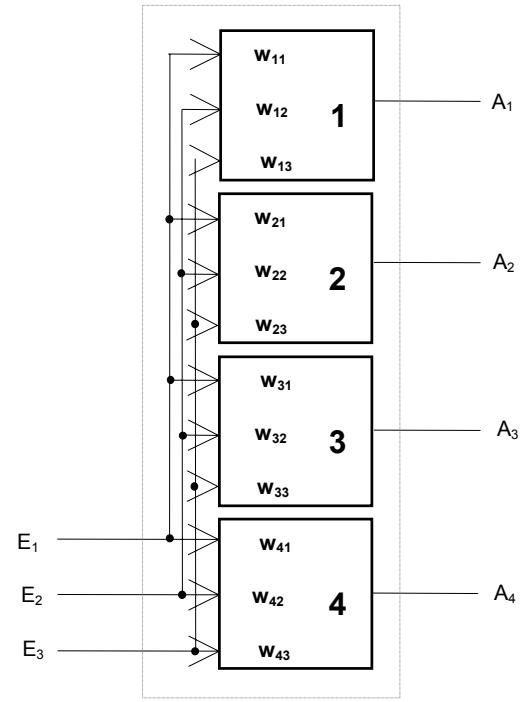


Abbildung 3.1: Muster-Assoziator: Ein einschichtiges Neuronales Netzwerk mit 3 Eingängen $E_1...E_3$ und 4 Ausgängen $A_1...A_4$. w_{ij} bezeichnen die Gewichte der, zu den Neuronen, zuführenden Kanten.

Der Netto-Input der Unit i lässt sich wie folgt berechnen:

(Netto-Input der Unit i) $$\varepsilon_i = \sum_{j=1}^{n} w_{ij} E_j \qquad \text{i=1…N} \tag{3.1}$$

Die Aktivität der Neuronen spielt beim Muster-Assoziator keine direkte Rolle, so dass man die Identität als Aktivierungsfunktion verwenden kann. Die Ausgangsfunktion ist beliebig.
Da die Neuronenausgänge mit den Netzausgängen übereinstimmen, gilt für den Muster-Assoziator folgende einfache Beziehung:

$$A_i = a_i\left(\sum_{j=1}^{n} w_{ij} E_j\right) \qquad \text{i=1…N} \tag{3.2}$$

wobei a_i die Ausgangsfunktion ist und der Index von a der Tatsache Rechnung trägt, dass jedes Neuron seine eigene Ausgangsfunktion haben kann.

3.1.2 Lernaufgabe

Der Muster-Assoziator gibt für jeden angelegten Eingangsvektor einen dazugehörigen Ausgangsvektor aus; er stellt also eine Assoziation zwischen Musterpaaren her. Gibt man nun einen Eingangsvektor und einen dazugehörigen Ausgangsvektor, als Beispiel, an, so besteht die Lernaufgabe nun darin, die Gewichte der Kanten (in der *Abbildung 3.1* bezeichnet als w_{ij}) so zu bestimmen, dass jedes Eingangsmuster zu einem gewünschten Ausgangsmuster führt. Diesen Satz von Musterpaaren kann man durch folgenden Ausdruck angeben:

$$(E_j^{\mu}, S_i^{\mu}), \quad \mu=1\ldots p \tag{3.3}$$

Wobei μ den Index des Paares angibt, i bzw. j ist der Index eines Elements eines Vektors, E ist der Eingangsvektor und S der Ausgangsvektor.

3.1.3 Linearer Muster-Assoziator mit Hebbscher Lernregel

Die wesentlichen Eigenschaften eines Muster-Assoziators lassen sich bereits an Hand linearer Ausgangsfunktionen zeigen, deswegen werden nachfolgende Betrachtungen an Hand von linearen Neuronen gemacht. Die Charakteristik dieser Neuronen kann man *Tabelle 3.1* entnehmen.

Typ	Ausgangswerte-Bereich	Formel	Eingangsfunktion	Ausgangs-Funktion
Linear	(-∞,+∞)	a=σ(Net-δ) Net … Netto-Input δ … Schwellenwert σ … Steigung	Skalarprodukt	Linear

Tabelle 3.1: Charakteristik von linearen Neuronen

Für weitere Betrachtungen wird der Schwellenwert δ auf 0 gesetzt. Verbindet man nun die Formel für die Aktivität der Neuronen aus *Gleichung 3.2* mit der Ausgangsfunktion in *Tabelle*

3.1, wobei die Steigung zu den Gewichten dazu geschlagen wird, ergibt sich die Aktivierungsfunktion in *Gleichung 3.4*.

$$A_i = \left(\sum_{j=1}^{n} w_{ij} E_j \right) \qquad \text{i=1…N} \tag{3.4}$$

Die Anwendung der Hebbschen Lernregel (Gleichung 2.1) ergibt nun für die Lernaufgabe die Gleichung 3.5.

$$\Delta W_{ij} = \varepsilon * \sum_{\mu=1}^{p} S_i^{\mu} E_j^{\mu} \tag{3.5}$$

Die Hebbsche Lernregel führt jedoch nur in Sonderfällen zu brauchbaren Ergebnissen.

Als wesentlichen Punkt ist anzumerken, dass die Hebbsche Lernregel nur dann brauchbar ist, wenn die Beispiel-Eingangsvektoren orthogonal zueinander stehen. Ansonsten wird ein gelerntes Muster durch das nächste Muster wieder zerstört. Der genaue Beweis ist in [Ho93] nachzulesen. Es ist zwar möglich, voneinander unabhängige Eingangsmuster durch Standartmethoden in ein Orthogonalsystem umzuwandeln, günstiger ist es jedoch stattdessen die Delta-Lernregel zu verwenden.

3.1.4 Linearer Muster-Assoziator mit Delta-Lernregel

Die Lernaufgabe ist wiederum durch Gleichung 3.3 gegeben, die Delta-Lernregel wird in Gleichung 2.2 eingeführt. Verwendet man die Gleichung 2.2 im Kontext zur Lernaufgabe erhält man für die Berechnung der Gewichtsänderung die Gleichung 3.6.

$$\Delta W_{ij} = \varepsilon * (S_i^{\mu} - \sum_{j=1}^{n} W_{ij} E_j^{\mu}) E_j^{\mu} \tag{3.6}$$

Wenn die Eingangsmuster linear unabhängig sind, konvergiert die Anwendung der Delta-Lernregel zu einer exakten Lösung der Lernaufgabe. Eine ausführliche Analyse der Delta-Lernregel findet man in [St88].

3.1.5 Grenzen des Muster-Assoziators

Ein Muster-Assoziator ist auf Musterpaare begrenzt, deren Eingangsmuster voneinander linear unabhängig sind. Die Beschränkung auf linear unabhängige Eingangsvektoren ist eine Folge des linearen Ausdrucks $W_{ij}E_j$ (*siehe Gleichung 3.6*), mit dem der effektive Eingang und damit der Ausgang der Neuronen berechnet wird. Da jede Ausgangsfunktion monoton wachsen muss, ist nicht zu erwarten, dass nichtlineare Neuronen diese Einschränkung aufheben werden. In [Ho93] wird dieses Problem genauer behandelt.

3.2 Perzeptron

Das Perzeptron wurde erstmal 1958 von Rosenblatt [Ro58] vorgestellt und war eines der ersten näher untersuchten Neuronalen Netze. Es ist eine Form des in Punkt 3.1 vorgestellten Muster-Assoziators und unterscheidet sich in erster Linie dadurch, dass dem musterassozierenden Netz ein Vorverarbeitungsnetz vorgeschalten wird.

3.2.1 Aufbau des Netzes

In seiner ursprünglichen Form ist das Perzeptron ein dreischichtiges Netz. Die *Abbildung 3.2* zeigt ein Netz mit 10 Eingängen E_1 bis E_{10} und 2 Ausgängen A_1 bis A_2.

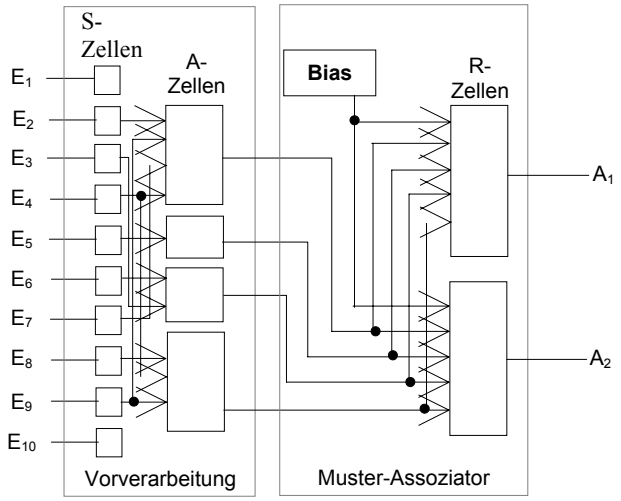


Abbildung 3.2: Aufbau eines klassischen Perzeptrons. Das Perzeptron besteht aus einem Muster-Assoziator (rechts im Bild), dem ein Vorverarbeitungsnetz vorgeschaltet ist.

Die Neuronen der ersten Schicht (der Retina) heißen *S-Zellen* (auch S-Units oder Stimulus-Zellen). Ihre einzige Aufgabe besteht darin, das Eingangsmuster E an die so genannten A-Zellen (auch A-Units oder Assoziations-Zellen) der Assoziationsschicht zu verteilen. Die Verbindungen zwischen den S- und den A-Zellen sowie die Gewichte der A-Zellen werden nach Zufallskriterien ausgewählt und sind fest vorgegeben. Diese beiden Schichten bilden daher jedes Eingangsmuster auf ein „Zwischenmuster" ab, führen also eine „Vorverarbeitung" durch.

Das eigentliche Perzeptron wird von der dritten Schicht (Perzeptron-Schicht, auch Verarbeitungsschicht oder Response-Schicht genannt) gebildet. Diese Schicht ist ein Muster-Assoziator mit Bias-Neuron und verarbeitet den Eingangsvektor E; ihre Neuronen heißen R-Zellen (R-Units, Response-Zellen) und sind (wie die A-Zellen) McCulloch-Pitts-Neuronen, werden also gemäß Gleichung 3.7 berechnet:

(McCulloch-Pitts Netto-Input)

$$\varepsilon_i = \sum_{j=0}^{n} W_{ij} e_j \tag{3.7}$$

(McCulloch-Pitts Aktivierungsfunktion)

$$a_i = 0, \quad \text{wenn } \varepsilon_i < 0$$
$$a_i = 1, \quad \text{wenn } \varepsilon_i \geq 0 \tag{3.8}$$

3.2.2 Lernregel

Der Lernvorgang erstreckt sich beim Perzeptron nur auf die R-Zellen und erfolgt gemäß der folgenden Regeln, vorgestellt von Rosenblatt 1958 (siehe [Ro58]):

(Perzeptron-Lernregel)	$\Delta W_{ij} = 0$	*für* $S_i = A_i$	ε … Lernkonstante	(3.9)
	$\Delta W_{ij} = \varepsilon * E_j$	*für* $S_i = 1, A_i = 0$	S_i … Eingangsmuster	
	$\Delta W_{ij} = -\varepsilon * E_j$	*für* $S_i = A_i\ A_i = 1$	A_i … Ausgangsmuster	

Dieses Verfahren nennt man *fehlerkorrigierendes* Lernverfahren: Wenn die Antwort einer R-Unit der gewünschten entspricht, so werden die Gewichte nicht verändert. Entspricht die Antwort nicht der gewünschten, so werden die Geweichte der zu dieser R-Unit führenden Verbindungen erhöht, wenn der Fehler (die Differenz zwischen der gewünschten und der tatsächlichen Antwort) positiv ist, sonst erniedrigt. Wendet man diese Lernprozedur für alle zu erlernenden Musterpaare wiederholt an, so stellen sich die Gewichte in endlicher Zeit entsprechend ein.

Wegen S_i, A_i in {0,1} sind diese beiden Formen zur Delta-Lernregel äquivalent. Durch das Bias-Neuron sind auch die Schwellen in den Lernvorgang einbezogen.

Den Beweis dafür, dass das Perzeptron nach endlich vielen Lernschritten mit einer geeigneten Lernrate ein beliebiges Eingangsmuster auf ein beliebiges Ausgangsmuster abbilden kann, findet man in der Literatur [Ri91].

3.2.3 Grenzen des Perzeptrons

Die Grenzen des Perzeptrons sind grundsätzlich dieselben wie beim Muster-Assoziator. Der Versuch, die Gewichte der Assoziationsschicht sowie deren Verbindungen mit der Retina geeignet zu wählen, um diese Grenzen zu überwinden, führt nicht zum Ziel [Ho93].

3.3 Auto-Assoziator

3.3.1 Aufbau des Netzes

Der Auto-Assoziator ist ein einschichtiges, vollständig verbundenes Netz. Im Gegensatz zum Muster-Assoziator speichert er einzelne Muster. Daher ist die Anzahl der Netz-Eingänge gleich der Anzahl der Netzausgänge. Seine Struktur kann durch Angabe einer einzigen Zahl, etwa der Neuronenanzahl charakterisiert werden. Im Gegensatz zum Muster-Assoziator, von dem man nur externe Eingänge kennt, existieren beim Auto-Assoziator auch interne Eingänge, also Verbindungen vom Ausgang einer Unit zum Eingang derselben Unit. Das Netz ist also rückgekoppelt [Ho93].

3.3.2 Lernaufgabe

Zum Lernen gibt man dem Auto-Assoziator einen Satz von Mustern vor. Wird dem Netz anschließend eines dieser Muster angeboten, so soll am Ausgang dasselbe Muster erscheinen. Der Auto-Assoziator assoziiert also die Muster mit sich selbst.

Legt man ein beliebiges Muster an den Eingang des trainierten Netzes an, so ergibt die Reproduktion im Idealfall eines der gelernten Muster. Das kann man für folgende Aufgaben ausnützen:

- Ergänzung: Ein Teilmuster eines gelernten Musters (bei dem man einen Teil der Komponenten etwa gleich „0“ gesetzt hat) wird an das Netz angelegt. Am Ausgang erscheint das vollständige Muster.
- Rekonstruktion: Eine „verstümmelte“ Variante eines gelernten Musters (bei dem die einzelnen Komponenten geringfügig verändert wurden) wird angelegt; am Ausgang erscheint das korrekte Muster.
- Generalisierung: Ein Muster, das nicht gelernt wurde, aber einem gelernten Muster ähnlich ist, wird angelegt. Am Ausgang erscheint dasjenige gelernte Muster, welches dem angebotenen am ähnlichsten ist.

3.4 Hopfield-Netz

Hopfield-Netze haben denselben Aufbau und denselben Zweck wie Auto-Assoziatoren mit den 2 großen Unterschieden, dass Rückkopplungen nicht erlaubt sind und ein anderer Lernalgorithmus verwendet wird.

3.5 Boltzmann-Maschine

3.5.1 Aufbau des Netzes

Die Boltzmann-Maschine besteht aus einer Eingangsschicht, einer verborgenen Schicht und einer Ausgangsschicht (siehe *Abbildung 3.3*). Die verwendeten Neuronen sind Boltzmann-Neuronen (siehe *Tabelle 3.2*) Das Netz ist vollständig verbunden, wobei jedoch folgende Einschränkungen gelten:

- Eingangs- und Ausgangsschicht haben keine direkte Verbindung; die Gewichte zwischen diesen Schichten sind also gleich „0“.
- Selbstrückkopplung ist ausgeschlossen, d.h. $w_{ii}=0$
- die Gewichte sind symmetrisch, d.h. $w_{ij}=w_{ji}$

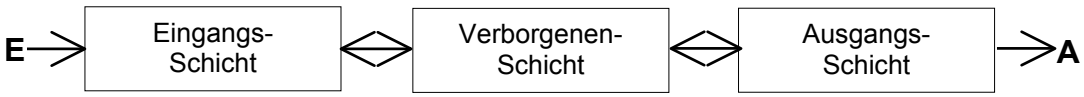


Abbildung 3.3: Schematischer Aufbau der Boltzmann-Maschine, Die Eingangsschicht ist mit der verborgenen Schicht in beiden Richtungen vollständig verbunden, was der Doppelpfeil andeutet; dasselbe gilt für die Beziehung zwischen der verborgenen und der Ausgangsschicht

Typ	Ausgangs-wertebereich	Formel	Eingangsfunktion	Ausgangs-Funktion
Boltzmann	(0 , 1)	P(a=1)= $\frac{1}{1+e^{-(Net-\vartheta)/T}}$	Nettoinput= $Net=\sum_{j=1}^{N} W_{ij} a_j$ N … Gesamtzahl der Neuronen	Boltzmann

Tabelle 3.2: Charakteristik von Boltzmann-Neuronen

3.5.2 Simuliertes Kühlen

Am Ende der Lernphase sollte das Netz im globalen Minimum zur Ruhe kommen. Daher beginnt man mit einem hohen Wert der „Temperatur“ T, wenn man die Aktivierungsfunktion der Boltzmann-Neuronen nimmt, und „kühlt“ das Netz allmählich ab, verringert also T schrittweise bis auf 0. Diese Vorgangsweise bezeichnet man als simuliertes Kühlen. Genaueres zu diesem Thema findet man in [Br91]. Selbstverständlich ist der Begriff Temperatur nur eine formale Analogie zur Physik; mit einer physikalischen Temperatur eines Neuronalen Netzes hat das nichts zu tun.

3.5.3 Lernregel

Die Boltzmann-Maschine ist heteroassoziativ, lernt also Musterpaare und hat dieselbe Funktion wie ein Muster-Assoziator. Zunächst setzt man die Gewichte auf Zufallswerte, wobei natürlich die Beschränkung der Gewichts-Symmetrie $w_{ij}=w_{ji}$ zu berücksichtigen sind. Dann folgen die Lernschritte, welche aus jeweils drei Teilen bestehen. *Abbildung 3.4* zeigt das Lernschema.

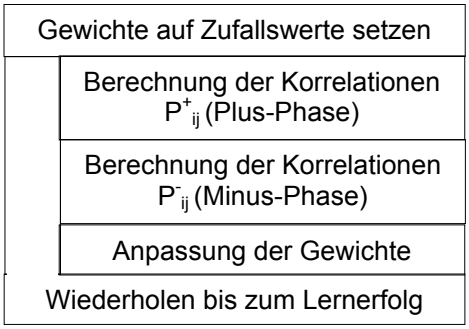


Abbildung 3.4: Lernschema der Boltzmann-Maschine

Der erste Teil eines Lernschrittes ist die *Plus-Phase*. Man legt ein Musterpaar an das Netz an; dass bedeutet, dass die Ausgänge der Eingangsneuronen gleich dem Eingangsvektor E und die Ausgänge der Ausgangsneuronen gleich dem Ausgangsvektor A gesetzt und festgehalten werden. Nun führt man simuliertes Kühlen (siehe Erklärung im *Punkt 3..5.3*) durch, bis sich bei einer niedrigen „Temperatur“ T_0 ein „thermisches Gleichgewicht“ einstellt; dabei verändern sich nur die Ausgänge der verborgenen Neuronen. Für jedes Neuronenpaar (i,j) kann man nun die Korrelation $P^{\mu+}_{ij}$ definieren. Dabei handelt es sich um die Wahrscheinlichkeit (d.h. um die relative Häufigkeit), dass die beiden Neuronen i und j im thermischen Gleichgewicht gleichzeitig aktiv sind. Diese Größe wird für die spätere Auswertung benötigt.

Dieser Vorgang (Musterpaar anlegen, simuliertes Kühlen, Bestimmung der Korrelationen) wird r-mal durchgeführt; im Allgemeinen wird r ein vielfaches der Musteranzahl p sein. Zum Schluss berechnet man mit der *Formel 3.10* die Korrelationsmittelwerte.

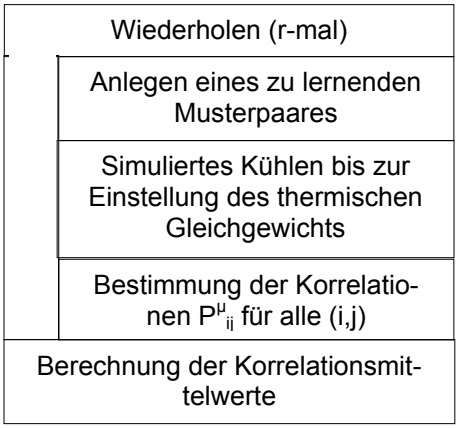


Abbildung 3.5: Struktogramm zur Berechnung eines Korrelationsmittelwertes

$$P_{ij}^{+} = \frac{1}{r} \sum P_{ij}^{\mu+} \tag{3.10}$$

Die Summe erstreckt sich dabei über alle r Korrelationen. *Abbildung 3.5* fasst die Schritte, die zu den Korrelationsmittelwerten führen, zusammen.

Der zweite Teil eines Lernschrittes ist die *Minus-Phase*. Die Vorgangsweise ist dieselbe wie in der Plus-Phase, jedoch bringt man das Netz in einen Startzustand, indem man die Ausgänge der Eingangsneuronen gleich einem Eingangsmuster E^{μ} setzt; anschließend unterwirft man alle Neuronen des Netzes der simulierten Kühlung. Dabei erhält man Korrelationen $P^{\mu-}_{ij}$, die man anschließend zu den Mittelwerten

$$P_{ij}^{-} = \frac{1}{r} \sum P_{ij}^{\mu-} \tag{3.11}$$

zusammenfasst. Nun können die Gewichte geändert werden; die Lernregel lautet:

$$\Delta W_{ij} = \eta * (P_{ij}^{\mu+} - P_{ij}^{\mu-}) \tag{3.12}$$

Die Vorgänge in der Minus-Phase folgen ebenfalls dem Schema von *Abbildung 3.5*.
Dieser dreiteilige Lernschritt wird so oft wiederholt, bis das Lernziel erreicht ist.

3.5.4 Probleme

Boltzmann-Maschinen sind nicht problemlos einzusetzen. Bereits der große Rechenaufwand lässt sie für viele Anwendungen als ungeeignet erscheinen. Darüber hinaus arbeitet das Netz nur dann zufrieden stellend, wenn die Parameter *r* (=die Anzahl der Schleifendurchgänge von *Abbildung 3.5* = ein Vielfaches der Musterpaar-Anzahl) und der Lernfaktor η günstig gewählt werden. Dafür gibt es keine allgemeingültigen Regeln.

4 Verwendete Ansätze

In diesem Kapitel werden die von mir verwendeten Ansätze vorgestellt, mit denen ich die Zuverlässigkeit der Sensor-Ausgangssignale erhöhen will.
Im Punkt 4.1 werden Themen besprochen, die allgemein von Bedeutung sind. Zum Beispiel wird beschrieben, wann die Aufnahme von Beispieldaten zu erfolgen hat. In *4.2* stelle ich einen konventionellen Algorithmus vor, der einerseits auf *linearer Approximation* und andererseits auf die Implementierung einfacher Regeln setzt. In *4.3* stelle ich eine Implementierung eines Neuronalen Netzwerkes, basierend auf *Back Propagation* vor und zeige mögliche Konfigurationen und Anwendungsmöglichkeiten auf.

4.1 Allgemeines

4.1.1 Ablauf der Lernphase

Egal welcher Mechanismus zur Datenauswertung der Sensorausgabe verwendet wird, muss es vor dem Einsatz des *Smart Sensors* eine Phase geben, in der Referenzwerte bzw. Beispieldaten für die Algorithmen eingegeben werden müssen. Da diese Daten im RAM gespeichert werden, ist vorgesehen, dass die Lernphase automatisch dann beginnt, wenn die Stromversorgung der Verarbeitungseinheit aktiviert wird.

Die Lernphase verläuft, für alle im weiteren Kapitel vorgestellten Ansätze, folgendermaßen:
- Ein Gegenstand wird, in 10cm Schritten, in einer Entfernung von 10cm bis 120cm zum Sensor platziert. Für jede dieser Entfernungen werden in etwa 10 Sensorausgaben gespeichert.
- Diese Daten werden dem jeweiligen Lernalgorithmus übergeben.

4.2 Implementierung mittels Konventionellen Algorithmus

Um einen Anhaltspunkt für den zu tolerierenden Fehler zu bekommen, vergleiche ich die Ergebnisse der Neuronalen Netze mit einem Vergleichssystem, das mittels 12 Referenzwerten von 10cm bis 120cm durch lineare Approximation Ausgangspegel in cm umrechnet.

4.2.1 Filtern von fehlerhaften Messwerten

Da die Sensordaten sporadisch fehlerhafte Pegel aufweisen, die erkennbar kleiner als die „richtigen“ Ausgangspegel sind, müssen diese Fehlstellen gefiltert werden.
Der Output des Sensors wird in den Variablen diagnose[0] bis diagnose[3], nach dem *„First in First out“* Prinzip, wie in eine Queue, geschrieben. Das bedeutet, dass in der Variable diagnose[3] der letzte gemessene Sensorwert steht, in der Variable diagnose[2] der vorletzte Sensorwert usw. Fehlerhaften Messwerte werden durch folgende Abfrage ausgefiltert:

```
if     ((diagnose[0]>diagnose[1])&&
       (diagnose[0]>diagnose[2])&&
       (diagnose[3]>diagnose[2]))
       {
              diagnose[1]=diagnose[0];
              diagnose[2]=diagnose[3];
       }
else
       if ((diagnose[0]>diagnose[1])&&
           (diagnose[2]>diagnose[1]))
       {
              diagnose[1]=diagnose[2];
       }
       else
              if ((diagnose[1]>diagnose[2])&&
                 (diagnose[3]>diagnose[2]))
              {      diagnose[2]=diagnose[3];   }
```

4.2.2 Referenzwerte

In der Lernphase werden 10 gefilterte (siehe Punkt 4.2.1) Sensormesswerte pro Entfernung (von 10cm bis 120cm in 10cm Schritten) gespeichert. Der Durchschnittswert der 10 Sensormesswerte wird als Referenzwert gespeichert. Dieser Wert ist wichtig für die spätere Umrechnung der Sensorausgabe zu Entfernungsangaben.

4.2.3 Berechnung durch lineare Approximation

Gegeben sind Referenzsensormesswerte, einer pro Entfernung von 10cm bis 120cm in 10cm Schritten. Diese Referenzwerte sind in einem Array a gespeichert, sodass sich in a[0] der Referenzwert für die 10cm Entfernung befindet, in a[1] ist der Referenzwert für 20cm Entfernung usw. bis zu a[11].
Für das Array gilt für jedes i im Bereich von 0 bis 11: $a[i] \geq a[i+1]$.
Man kann die Ausgabe des Sensors mittels folgender Formel in cm umrechnen:

$$x = \left(\frac{y - a[i]}{a[i] - a[i-1]} + i\right).10cm \tag{4.1}$$

wobei gelten muss: $a[i-1] < y \leq a[i]$

a[0] bis a[11]	...	Ausgangspegel-Referenzwerte für 10cm bis 120cm
y	...	aktuelle Sensorausgabe
x	...	aktuelle Entfernung in cm

4.2.4 Erkennen von Entfernungen > 120cm

Messungen mit dem Sharp GP2D02 haben ein Problem im Erkennen von Entfernungen die größer als 120cm sind. Wird vor dem Sensor innerhalb von 120cm kein Hindernis platziert, kommt es zu starken Schwankungen der Sensorausgabewerte. In [Di02] wurde dokumentiert, wie der GP2D02 in einem mobilen Roboter eingesetzt wurde und man kann unter anderem das oben beschriebene Sensorverhalten bei großen Entfernungen nachschlagen.
Die einfachste Methode zu erkennen, wenn der Sensor fehlerhafte Werte statt konkreten Entfernugsmessungen ausgibt, besteht darin, die absoluten Differenzen zwischen 5 aufeinander folgenden Ausgangspegel zu addieren und mit einem gewissen Schwellenwert zu vergleichen. Bei diesen Berechnungen dürfen die Pegel noch nicht, wie in diesem Kapitel bereits vorgestellt, gefiltert sein.

Da aber die Wahrscheinlichkeit über 90% beträgt, in 5 aufeinander folgenden Pegel einen abweichenden Messwert zu finden, auch wenn dem Sensor innerhalb von 120cm ein Hindernis vorgestellt wurde, muss auch hier ein Filter eingesetzt werden.
Geht man so vor, dass man den Messwert mit dem größten Abstand zu den anderen Messwerten herausfiltert, liegt laut meinen Testfällen die Wahrscheinlichkeit bei ca. 95 %, richtig zwischen endlichen und unendlichen Entfernungen zu unterschieden.

4.3 Implementierung mittels Neuronalen Backpropagation-Netzwerk

4.3.1 Auswahl des Netzwerkes

Für das gestellte Problem ist ein Neuronales Netzwerk notwendig, das nicht nur Assoziationen vornehmen, sondern auch Funktionen darstellen kann. Daher ist es notwendig, dass das Netz

- anhand von linear-unabhängigen Daten lernen kann,
- jede beliebige Funktion darstellen kann

In Kapitel 3 wurden die gebräuchlichsten Ansätze von Neuonalen Netzen, mit überwachtem Lern-Algorithmus, gezeigt. Diese sind aber alle nicht, oder zumindest weniger geeignet für die Arbeit mit dem Smart Sensor, als die Backpropagation-Netzwerke.

Der vorgestellte, klassischen Muster-Assoziator und das Perzeptron fallen deshalb aus der Entscheidung heraus, da sie linear unabhängige Eingangsmuster benötigen.
Der klassische Auto-Assoziator und das Hopfield-Netz haben die Aufgabe, unvollständige Eingabemuster zu ergänzen oder ähnliche Eingabemuster zu finden und sind deshalb ebenfalls nicht geeignet.

Die Boltzmann-Maschine wäre für die Implementierung geeignet und hat, im Gegensatz zu Backpropagation-Netzwerke, den großen Vorteil, dass lokale Minima (genauere Erklärung dazu findet man im *Abschnitt 4.3.5*) vermieden werden. Andererseits ist das Finden von globalen Minima dermaßen rechenintensiv, dass die Einbettung des Algorithmus in einen Smart Sensor nicht wirtschaftlich wäre.

Laut dem Theorem von Hecht-Nielsen (siehe [Ni90]) kann eine beliebige Funktion von einem dreischichtigen Neuronalen Netz mit vollständigen Verbindungen zwischen den Schichten ohne Zyklen (vorwärtsgekoppelt, „feed forward network“) berechnet werden. Mehr als 3 Schichten können zur Verkürzung der Lernphase verwendet werden, jedoch ist die Wahrscheinlichkeit größer, dass der Lernalgorithmus (wie weiter unten vorgestellt) eher in einem ungünstigen Minimum der Fehlerfunktion abbricht.

4.3.2 Netzarchitektur

Ein Backpropagation-Netzwerk (Fehlerrückführungsnetzwerk) ist ein mehrschichtiges, vorwärtsgekoppeltes Netz. *Abbildung 4.1* zeigt ein typisches Beispiel. Von anderen Netzen unterscheidet es sich weniger durch seine Struktur als vielmehr durch die verwendete Fehler-

rückführungs-Lernregel, die im Gegensatz zur Hebbschen und zur Delta-Lernregel auch auf verborgene Neuronen anwendbar ist.

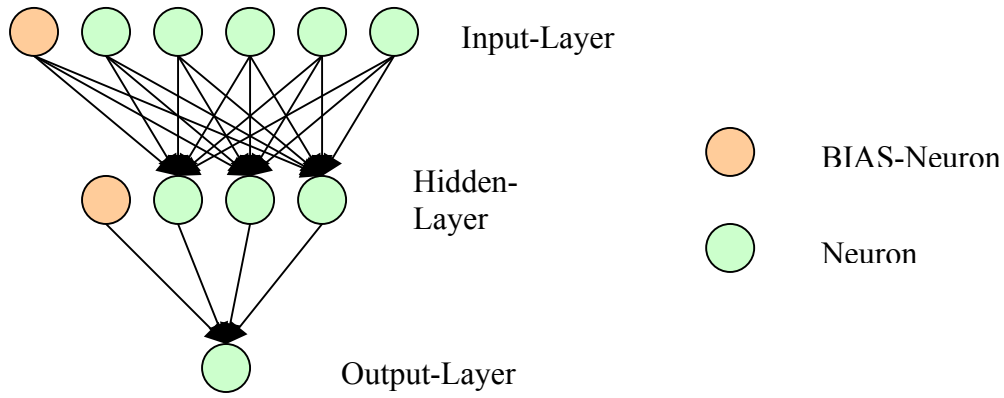


Abbildung 4.1: Dreischichtiges Neuronales Feed-Forward Netwerk

4.3.3 Netto-Input einer Unit

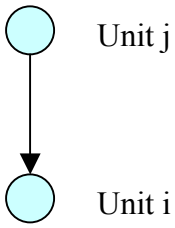


Abbildung 4.2: Verbindung zwischen 2 Units

$$net_i = \sum_{j}^{LowNum} w_{ij} * Output[j] \tag{4.2}$$

LowNum	....	Anzahl der Neuronen (=Units) im vorigen (lower) Layer
w_{ij}	....	Gewicht der Verbindung von Unit j zu Unit i, ε [0,1]
net_i	....	Netto-Input der Unit i

4.3.4 Output- und Aktivierungsfunktion

Eine Outputfunktion berechnet aus dem Netto-Input einer Unit deren Ausgabe dieser Unit aus. Für für negative oder (zu) kleine positive Werte sollte die Funktion ein definiertes Minimum und ab einem gewissen Schwellwert allmählich oder auch abrupt einen Maximalwert annehmen. *Abbildung 4.3* zeigt die gewählte Outputfunktion:

$$sigmoid(net) = \frac{1}{1 + e^{-net}} \tag{4.3}$$

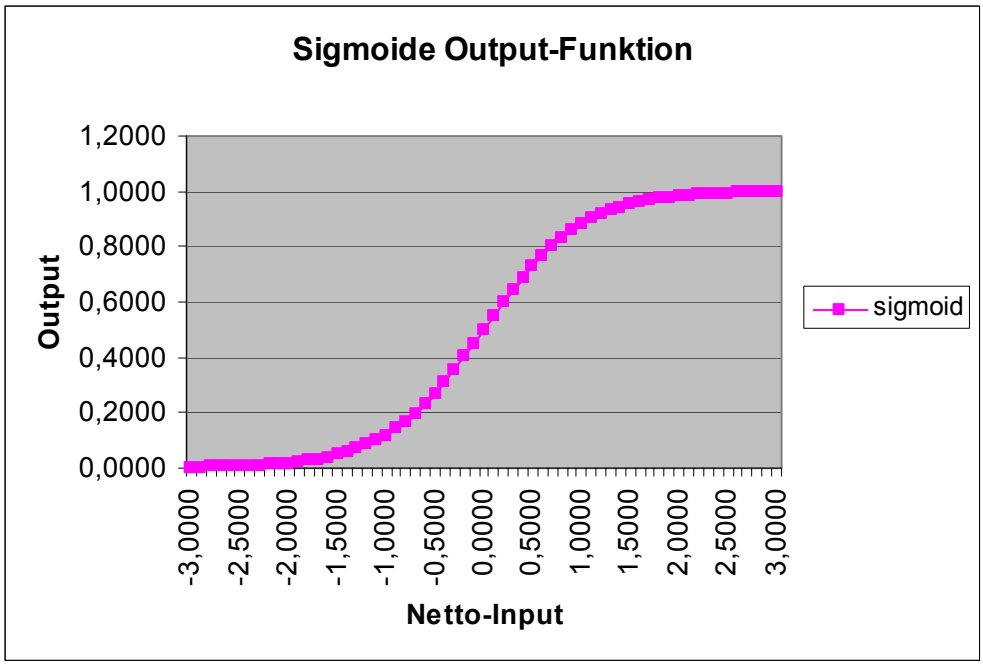


Abbildung 4.3: gewählte Aktivierungs- und Outputfunktion der Neuronen

Die Funktion ist in ihrem ganzen Definitionsbereich differenzierbar. Da ein mehrschichtiges Netz ausschließlich eine Reihe von Funktionskompositionen berechnet, wird bei der Verwendung einer stetig differenzierbaren Aktivierungsfunktion auch die Fehlerfunktion selbst stetig differenzierbar, wodurch ein Lernalgorithmus mittels Gradientenabstiegsverfahren ermöglicht wird. Die Ableitung, die wir für den Backpropagation-Algorithmus brauchen, lautet:

$$\frac{d \quad sigmoid}{d \quad net} = sigmoid\,(net) * (1 - sigmoid\,(net)) \tag{4.4}$$

Die sigmoide Funktion ist auf Null zentriert. Daraus folgt, dass jedes Netz-Element mehr oder weniger „eingeschaltet“ ist wenn seine Aktivierung positiv ist und „ausgeschaltet“, wenn negativ. Bei vielen Anwendungen ist es aber notwendig, den Punkt der Aktivierung zu steuern, daher gibt es im Netz sogenannte BIAS-Knoten (nach dem englischen Wort für „Tendenz“) die immer Output=1 haben und mit jedem Knoten der nächsten Schicht verbunden sind. Durch das Gewicht dieser Leitungen kann man demnach den Schwellwert für die Aktivierung des Knotens steuern.

4.3.5 Lernalgorithmus: Back Propagation

Das Lernproblem besteht für solche Netze darin, Paare von Eingangsvektoren so genau wie möglich auf Ausgangsvektoren abzubilden. Diese Paare bilden die Trainingsmenge.
Der Backpropagation-Algorithmus sucht das Minimum der Fehlerfunktion eines bestimmten Lernproblems durch Abstieg in der Gradientenrichtung. Die Kombination derjenigen Gewichte eines Netzes, die den Berechnungsfehler minimiert, wird als Lösung des Lernproblems betrachtet. Der Gradient der Fehlerfunktion muss also für alle Punkte des Gewichteraums existieren, d.h. die partiellen Ableitungen der Fehlerfunktion nach den einzelnen Gewichten müssen überall definiert sein.

Das Lernverfahren erfolgt in 2 Schritten: Zuerst wird das angelegte Muster in Richtung Output-Layer propagiert um dort die Reaktion des Netzes auf das präsentierte Muster zu generieren. In der zweiten Phase erfolgt die Gewichtsänderung, abhängig vom Grad der Falschheit der Netzantwort.
Die Ausbreitung der Aktivierung durch das Netz erfolgt schichtweise, d.h. die Aktivierungen der Units werden zuerst in jenem Hidden-Layer berechnet, der dem Input-Layer am nächsten liegt. Dann erfolgt die Berechnung der Aktivierungen des nächsten, weiter bei Output-Layer gelegenen Layer, bis schließlich der Output-Layer selbst erreicht ist.

In der zweiten Phase erfolgen die Fehlerbestimmung und die entsprechende Gewichtsänderung, wiederum schichtweise. Dabei wird beim Output-Layer begonnen, da hier das gewünschte Muster zur Verfügung steht und mit dem tatsächlich vom Netz produzierten Muster verglichen werden kann. Aus der Differenz dieser beiden Muster wird ein so genanntes Fehlersignal gebildet, von dem einerseits die Änderung der Gewichte zwischen dem Output-Layer und dessen benachbartem Hidden-Layer und andererseits auch die Berechnung des neuen Fehlersignals für den nächsten Hidden-Layer abhängen.

$Error_i$	...	(der schon bekannte) Fehler der Unit i im Upper-Layer
$Error_j$	...	(der noch zu berechnende) Fehler der Unit j im Lower-Layer
$Output_i$	...	die vom Neuronalen Netz berechnete Ausgabe der Unit i
Gain	...	Steigung der Aktivierungsfunktion

Berechnung des Backpropagate Error:

```
Für jeden Layer k vom letzten Layer bis zum zweiten Layer
     Für jede Unit i im (k-1)ten Layer (=Lower Layer)
          Err=0
          Für jede Unit j im (k)ten Layer (=Upper Layer)
               Err+=wji*Errorj
          Errori:=Gain*Outputi*(1-Outputi)*Err
```

Ist der Fehler bis zum letzten Hidden-Layer rückgesendet und wurden dabei alle Gewichtsänderungen vorgenommen, kann wieder ein (neues) Muster angelegt und vorwärts propagiert werden. Wendet man diese Lernprozedur wiederholt an, so wird der Fehler schrittweise verringert. Es kann aber auch sein, dass erlernte Muster mit diesem Vorgehen wieder zerstört werden.

Um das Auftreten von Oszillationen zu vermeiden, wo wiederholt einige Muster andere größtenteils zerstören, wird der Gewichtsänderung einen Term beifügen. Dieser macht die aktuelle Gewichtsänderung von der vergangenen abhängig und wirkt dadurch einer allzu abrupten Gewichtsänderung entgegen.

Das Gewicht der Verbindung von Neuron i zu Neuron j:

$w_{ij}(t)$	...	Gewicht der Verbindung von Unit j zu Unit i
$dw_{ij}(t-1)$	...	vorhergehende Gewichtsveränderung
$Error_i$	...	Fehler der Unit i im Upper-Layer
$Output_j$	...	Ausgabe der Unit j im Upper-Layer
Eta	...	Lernrate
Alpha*dWeight(t-1)	...	Momentumterm=Einfluss der vorhergehenden Gewichtänderung

$$w_{ij}(t) = w_{ij}(t) + Eta * Error_i * Output_j + Alpha * dw_{ij}(t-1)$$
$$dw_{ij}(t) = Eta * Error_i * Output_j \quad (4.5)$$

Abbildung 4.4: Verbindung zwischen 2 Units

4.3.6 Wertebereich für die Aktivierung der Neuronen

Da die sigmoide Aktivierungsfunktionen (->(0,1)) sich den Werten 0 und 1 nur asymptotisch nähert, wird für ein schnelleres Lernverhalten auf niedrigere Ausgabewerte zurückgegriffen.
Gewählt: $Output_{min}$=LO=0,1
$Output_{max}$=HI=0,9

Die Eingabewerte und gewünschten Ausgabewerte müssen wie folgt normalisiert werden:

$$x_{normalisiert} = ((x-min) / (max-min)) * (HI-LO) + LO \quad (4.6)$$

4.3.7 Abbruchbedingung für den Lernalgorithmus

Die Genauigkeit der Abbildung wird durch den quadratischen Fehler des Netzes definiert:

m	...	Anzahl der Trainingspaare
y_1 bis y_m	...	Ausgabevektoren
t_1 bis t_m	...	gewünschte Ausgabevektoren

(quadratischer Fehler) $$E = \sum_{i=1}^{m} \|t_t - y_i\|^2 \quad (4.7)$$

Bei diesem speziellen Problem wird der Backpropagation-Algorithmus solange Durchgeführt, bis der quadratische Fehler 20x hintereinander der selbe war, oder
der Fehler größer ist als das 1,5 fache vom kleinsten berechneten Fehler, oder
bis das Netzwerk 6000x trainiert wurde.

4.3.8 Grenzen des „Back Propagation“-Algorithmus

Der „Back Propagation“ Lernalgorithmus hat die Nachteile, dass sie nicht immer die Lernfähigkeit garantiert und darüber hinaus sehr langsam ist. Diese beiden Eigenschaften sind das Ergebnis der „Gradientenabstieg“-Methode. Die Bewegung in Richtung Gefälle garantiert nicht, das absolute Minimum zu erreichen, sondern die nächste Senkung, die durchaus nicht die niedrigste sein muss. Außerdem lässt die Ableitung proportionale Bewegung die weniger steilen Abstiege in „Zeitlupentempo“ befahren.
Der häufigste Grund für die Entstehung lokaler Minima in einem Netz besteht darin, dass die Fehlerfunktion für andere Ausgabewerte als 0 und 1 berechnet wird.

5 Ergebnisse

In diesem Kapitel werden verschiedene Konfigurationen des in Punkt 4 vorgestellten Neuronalen Netzwerkes und die konventionelle Umrechnung getestet. Für einen besseren Vergleich wurden alle Statistiken mit denselben Sensorausgaben erstellt.
Weiters werden die Sensorausgaben einer genaueren Betrachtung bezüglich Filterung, Schwankungshöhe und Schwankungsstärke unterzogen.

5.1 Testaufbau

Um die Konfigurationen des Neuronalen Netzes zu testen, wurde wie folgt vorgegangen:
Jeweils 30 Sensormesswerte des Sensor GP2D02 wurden für die Entfernungen zwischen 10cm und 120cm in 5cm Schritten aufgenommen und über die RS232 Schnittstelle in eine Datei auf einem PC gespeichert. Die Konfigurationen des Neuronalen Netzes und die Referenzlösung wurden am PC implementiert und mit den am PC gespeicherten Sensormesswerten getestet. Die Kalibrierung der Referenzlösung, bzw. das Trainieren der Neuronalen Netze erfolgte nur mit, durch 10 ohne Rest, teilbaren Sensormesswerten, das Testen erfolgte mit allen aufgenommenen Sensormesswerten

5.2 Erkennen von „unendlich“ Entfernungen

Geht man dabei so vor wie in Punkt 4.2.4 beschrieben, erhält man bei einem automatisch in der Lernphase generierten Schwellenwert, in rund 95 % der Fälle eine korrekte Unterscheidung zwischen endlichen und unendlichen Entfernungen.

5.3 Typische Sensorausgaben

In der folgenden *Tabelle 5.1*, werden typische, ungefilterte Sensorausgabewerte gezeigt:

Entfernung	Sensorausgabe																	
10 cm:	223	223	223	223	223	223	223	223	223	223	223	192	223	223	223	223	223	222
20 cm:	163	163	163	163	164	163	163	163	163	163	160	163	164	164	164	163	163	163
30 cm:	136	136	136	136	136	136	136	136	136	128	136	136	136	136	136	136	136	128
40 cm:	123	96	123	123	123	96	123	123	123	123	123	123	123	123	123	123	123	123
50 cm:	119	96	118	118	119	119	118	118	118	118	118	118	96	118	119	118	118	118
60 cm:	112	112	112	112	112	96	112	112	112	96	112	112	112	112	112	96	96	112
70 cm:	109	109	109	109	109	109	109	96	109	110	109	109	109	109	109	109	109	109
80 cm:	106	106	96	106	106	106	96	106	106	107	106	106	96	106	107	96	106	96
90 cm:	103	103	103	103	103	103	103	103	103	103	103	96	103	103	103	103	103	96
100 cm:	102	102	96	101	102	102	102	102	102	102	96	101	102	102	102	102	102	102
110 cm:	101	101	101	101	101	96	100	100	101	101	100	100	101	100	100	101	101	101
120 cm:	100	100	96	100	100	100	96	101	101	101	100	96	100	100	100	101	100	100
>120cm:	108	101	96	100	96	96	103	111	100	109	104	111	111	100	111	108	98	106

Tabelle 5.1: ungefilterte Sensorausgabe

In *Abbildung 5.1* werden 2 typischen Ausgangskennlinien gezeigt. Beide Kennlinien stammen vom selben Sensor der jeweils an derselben Spannungsquelle angeschlossen war. Die Kennlinien wurden nach unterschiedlichen Betriebsdauer aufgenommen.

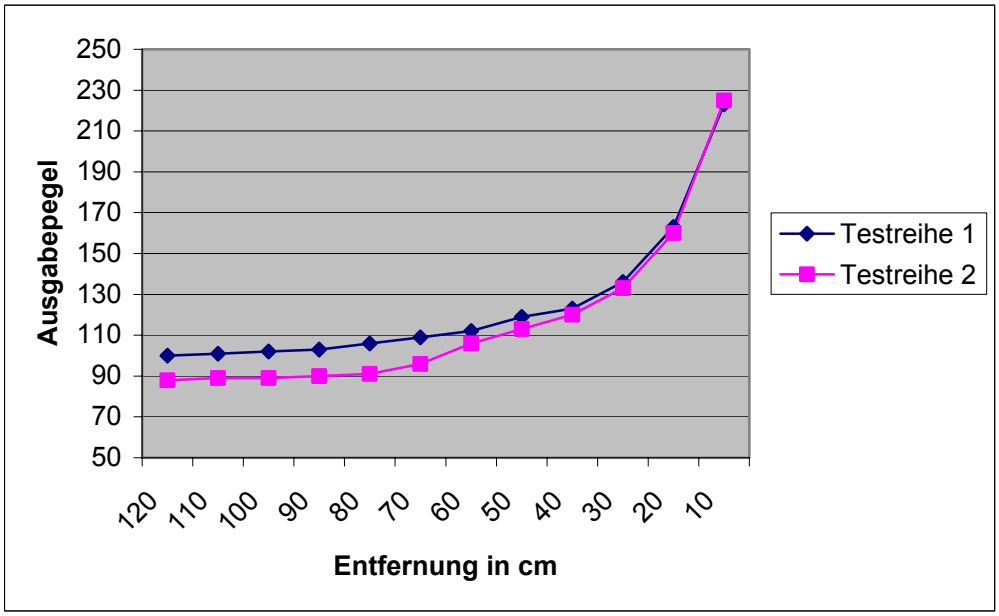


Abbildung 5.1: Ausgangskennlinien des Sensors: zeigt die Veränderung der Ausgabe des Sensors nach unterschiedlicher Betriebsdauer.

Die zwei Kennlinien in *Abbildung 5.1* wurden mit demselben Sensor zu unterschiedlichen Zeitpunkten mit verschiedenen Netzgeräten als Stromversorgung aufgenommen. Zusätzlich ist, laut Kennblatt, die Sensorausgabe des GP2D02 abhängig von der Temperatur. Eine Temperatur- oder eine Spannungsversorgungsschwankung kann also die Genauigkeit der cm-Umrechnung schwerwiegend beeinträchtigen.

5.4 Ergebnisse der konventionellen Umrechnung

In diesem Kapitel werden die Ergebnisse der konventionellen Umrechnung von Sensordaten, wie in Punkt 4.2 vorgestellt, präsentiert.

5.4.1 Filtern der Sensormesswerte

Da in unregelmäßigen Abständen die Sensorausgabe kurzfristig stark absinkt und der Algorithmus für die konventionelle Umrechnung davon stark beeinträchtigt wird, ist es auf jeden Fall von Vorteil, die ungefilterten Sensorwerte durch einen Filter zu schicken.
Filtert man kurzfristige Schwankungen heraus (mit der in Punkt *„Filtern von fehlerhaften Messwerten“* gezeigten Methode) und betrachtet nur die Abstände zwischen den Sensorausgaben, bekommt die Werte in *Tabelle 5.2*:

Entfernung	Absolute Differenz zwischen zwei aufeinander folgenden gefilterten Sensorwerten																	
10 cm:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2
20 cm:	0	0	0	0	1	0	0	0	1	1	1	2	0	1	1	0	1	0
30 cm:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40 cm:	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50 cm:	2	1	2	2	1	1	0	0	0	0	1	2	3	2	1	0	1	0
60 cm:	0	0	0	0	0	16	0	0	0	0	0	0	0	0	16	32	0	16
70 cm:	0	0	0	0	0	1	2	2	0	1	0	0	0	0	0	0	0	0
80 cm:	0	0	10	0	0	1	2	0	2	1	1	21	12	3	3	12	2	1
90 cm:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
100 cm:	2	2	1	1	0	0	0	1	2	2	1	1	0	0	0	0	6	1
110 cm:	0	0	1	1	2	1	2	2	2	3	0	1	2	1	1	0	1	2
120 cm:	0	0	4	1	1	1	1	1	1	1	0	1	0	1	2	2	2	2
>120cm:	13	8	8	11	10	19	15	14	12	12	7	3	3	14	13	18	18	5

Tabelle 5.2: absolute Abstände zur nächsten Sensorausgabe nach Filterung

Man erkennt, dass im Vergleich zu den ungefilterten Sensordaten die „Ausgangskennlinie“ des Sensors in *Tabelle 5.1* wesentlich glatter ist. Die einzige Ausnahme bildet die Sensorausgabe bei einer Entfernung von über 120cm. Hier gibt der Sensor nur mehr zufällige Werte aus. Anhand dieser starken Schwankungen kann man während der Sensor in Betrieb ist erkennen, ob ein Hindernis innerhalb von 120 cm vor dem Sensor vorhanden ist oder nicht.

5.4.2 Ergebnisse der lineare Approximation

Verwendet man nun Filterung und die konventionelle Umrechnung durch lineare Approximation, erhält man die Fehlerkennlinie von *Abbildung 5.1*. Die x-Achse bezeichnet die Entfernung eines Gegenstandes zum Sensor und die y-Achse den durchschnittlichen, absoluten Fehler von 10 Testfällen, der bei der Umrechnung von Sensorwerten in cm entstanden ist.

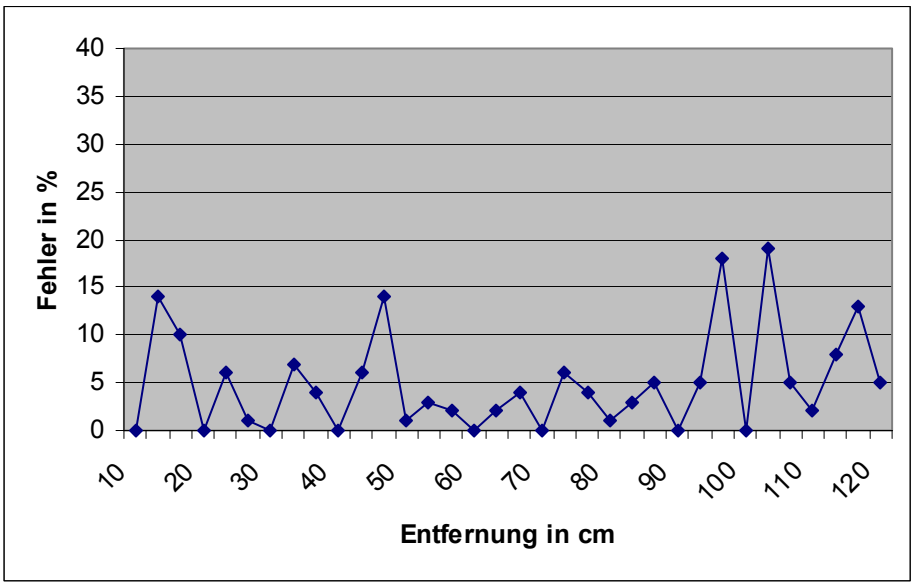


Abbildung 5.2: Fehlerkurve bei Messwertumrechnung durch linearer Approximation

Die Fehler sind folgendermaßen zu erklären:

- Der Algorithmus arbeitet nur mit 12 Referenzpunkte (ein Referenzpunkt pro Entfernungen von 10cm bis 120cm in 10cm Schritten) wobei die Ausgangskennlinie (siehe *Abb. 5.1*) stark nicht-linear ist.
- Mit steigender Entfernung sinkt die Anzahl der signifikanten Stellen welche Messwerte unterschiedliche Entfernungen unterscheiden. Dieser Effekt ist vor allem bei Entfernungen ab 80cm zu beobachten.

5.5 Neuronales Netzwerk mit Input: Max, Min, Mean

Bei den folgenden Experimenten werden 8 verschiedene Konfigurationen eines Neuronales Netzwerkes miteinander verglichen. Jede dieser Konfigurationen hat die gleiche Anzahl von Input-Neuronen und unterscheidet sich nur durch ihre Zwischenschichten. Die Eingabe ist bei allen Konfigurationen gleich: Es wird das Maximum, das Minimum und der Durchschnitt von 5 aufeinander folgenden, ungefilterten Sensormesswerten gebildet und an die 3 Input-Units angelegt. In *Abbildung 5.3* werden die Konfigurationen einander gegenüber gestellt.

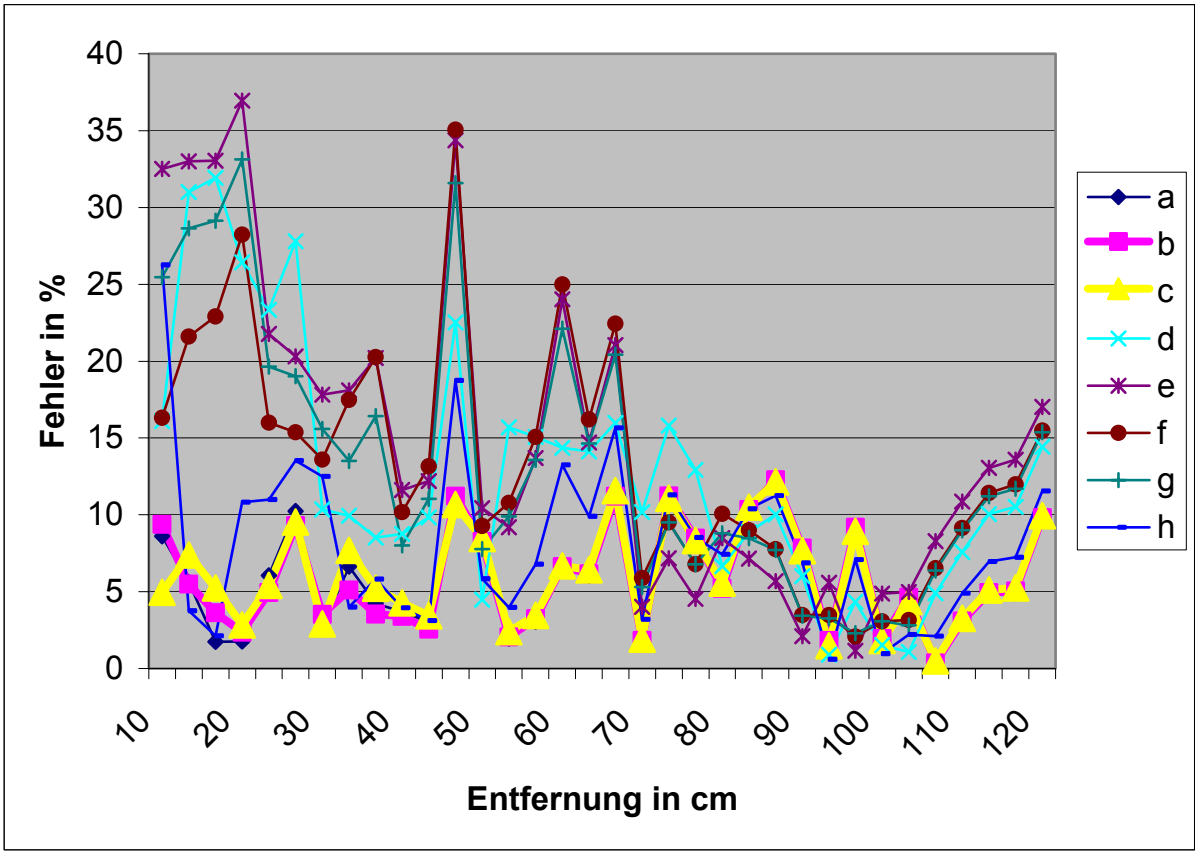


Abbildung 5.3: Fehlerkurve verschiedener Netzwerkkonfigurationen;
a) 3 schichtiges Netz: Zwischenschicht hat 4 Units
b) 3 schichtiges Netz: Zwischenschicht hat 8 Units
c) 3 schichtiges Netz: Zwischenschicht hat 12 Units
d) 3 schichtiges Netz: Zwischenschicht hat 16 Units
e) 4 schichtiges Netz: Zwischenschicht 1 hat 5 Units, Zwischenschicht 2 hat 5 Units
f) 4 schichtiges Netz: Zwischenschicht 1 hat 8 Units, Zwischenschicht 2 hat 5 Units
g) 4 schichtiges Netz: Zwischenschicht 1 hat 5 Units, Zwischenschicht 2 hat 8 Units
h) 4 schichtiges Netz: Zwischenschicht 1 hat 9 Units, Zwischenschicht 2 hat 9 Units

Am Besten bei der Gegenüberstellung in *Abbildung 5.3* schneiden die Konfigurationen *b* und *c ab*. Dieses Ergebnis kann man wie folgendermaßen erklären:

- Konfiguration a hat zu wenig Units um die gewünschte Funktion aufnehmen zu können
- Konfiguration b und c haben vor allen anderen Konfigurationen die am besten passende Anzahl von Units.
- Konfiguration d hat zu viele Units: die Wahrscheinlichkeit dass der Lernalgorithmus abbricht, bevor eine passende Gewichtseinteilung für das Netz gefunden wurde, ist hoch.
- Konfiguration e, f, g und h können dadurch, dass sie aus mehreren Zwischenschichten aufgebaut sind, schneller lernen. Die Wahrscheinlichkeit ist aber höher, dass der Lernalgorithmus, wegen der verwendeten Gradientenabstiegsmethode (siehe *4.3.7*), in einem lokalen Minimum abbricht.

5.6 Neuronales Netzwerk mit Input: Max von 5 Sensorwerten

Um vergleichen zu können, wie sich das Netzwerk verhält, wenn nur eine Input-Unit existiert, wird im folgenden das selbe Netzwerk wie in Punkt *5.4 (Neuronales Netzwerk mit Input: Max, Min, Mean)* verwendet, mit dem Unterschied, dass statt Maximum, Minimum und Durchschnitt nur das Maximum von 5 aufeinander folgenden, ungefilterten Sensormesswerten als Input dient. Die *Abbildung 5.4* zeigt die Fehlerkurve zu dieser Netzwerkkonfiguration. Da die Ergebnisse der Experimente mit mehrschichtigen Netzwerkkonfigurationen dermaßen schlecht ausfielen, wurden sie nicht in die *Abbildung 5.4* aufgenommen.

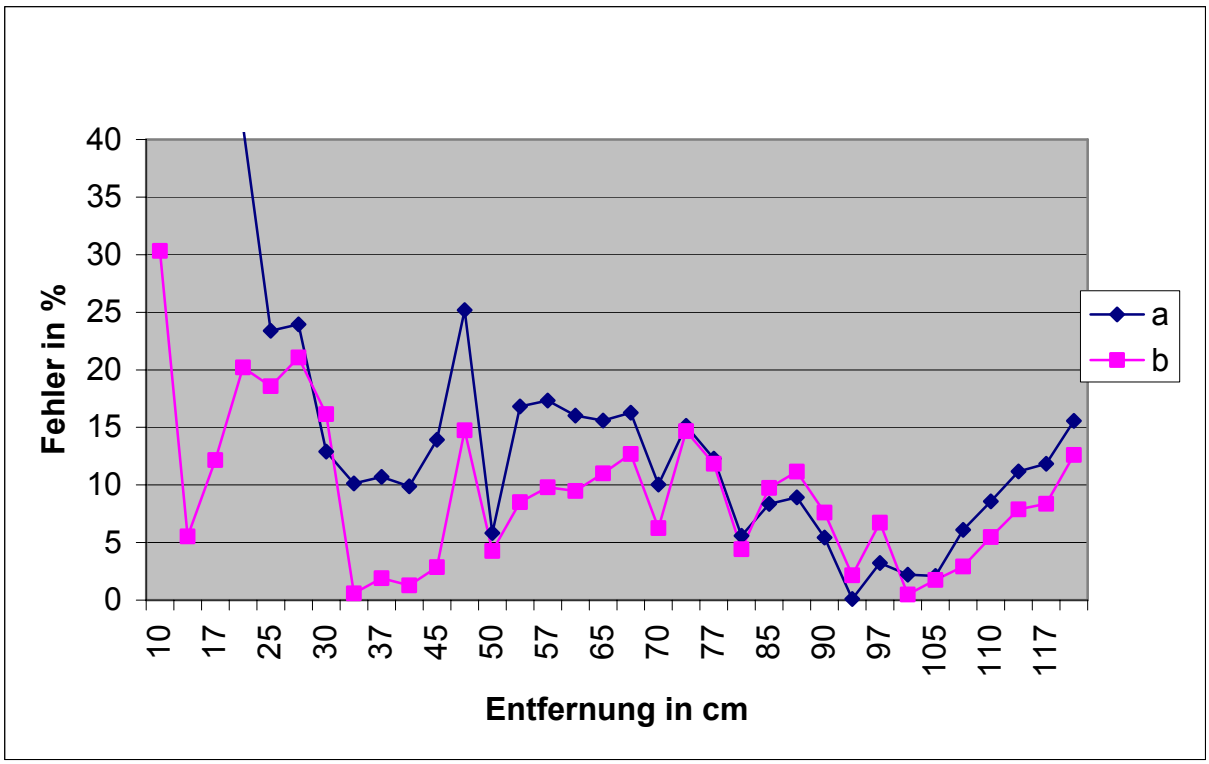


Abbildung 5.4: Fehlerkurve verschiedener Netzwerkkonfigurationen;
a) 3 schichtiges Netz: Zwischenschicht hat 8 Units
b) 3 schichtiges Netz: Zwischenschicht hat 16 Units

Der wesentlichen Unterschiede zu der Konfiguration mit Min, Max und Mean als Input ist die höhere Anzahl der erforderlichen Units in der Zwischenschicht, um den maximalen Fehler und mittleren Fehler klein zu halten.

5.7 Neuronales Netzwerk mit Input: 5 ungefilterte, aufeinander folgende Sensormesswerte

In dieser Konfiguration wurde ein Netzwerk mit 5 Input-Units getestet an die ein Vektor mit 5 aufeinander folgenden, unbearbeiteten Sensormesswerten angelegt wird. *Abbildung 5.5* zeigt die Fehlerkurven.

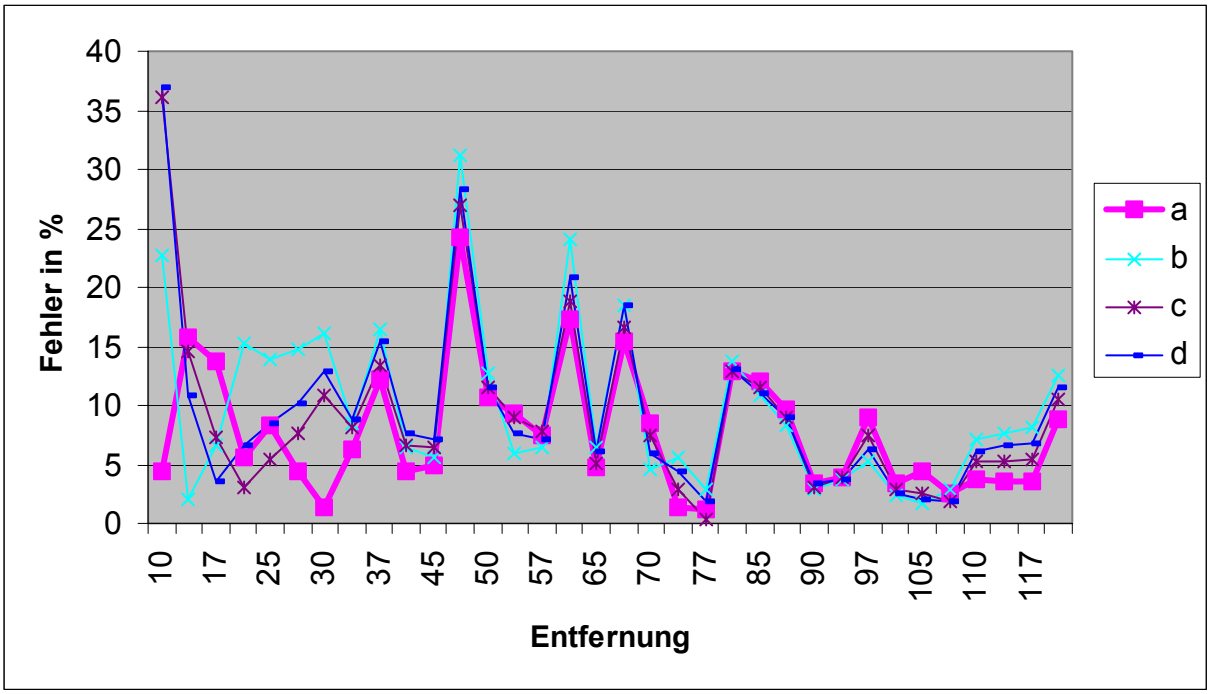


Abbildung 5.5: Fehlerkurve verschiedener Netzwerkkonfigurationen;
a) 3 schichtiges Netz: Zwischenschicht hat 8 Units
b) 3 schichtiges Netz: Zwischenschicht hat 16 Units
c) 4 schichtiges Netz: Zwischenschicht 1 hat 5 Units, Zwischenschicht 2 hat 5 Units
d) 4 schichtiges Netz: Zwischenschicht 1 hat 9 Units, Zwischenschicht 2 hat 9 Units

Der wesentliche Unterschied zwischen dem Netz mit Max, Min und Durchschnitt als Eingabe und diesem Netz hier, besteht darin, dass bei dieser Konfiguration der Lernalgorithmus wesentlich stabiler läuft. Das „Max, Min, Mean“-Netzwerk hängt sehr stark von der gewählten Anzahl der Units ab.

5.8 Neuronales Netzwerk mit Input: 3 ungefilterte, aufeinander folgende Sensormesswerte

In dieser Konfiguration wurde ein Netzwerk mit 3 Input-Units getestet an die ein Vektor mit 3aufeinander folgenden, unbearbeiteten Sensormesswerten angelegt wird. *Abbildung 5.6* zeigt die Fehlerkurven.

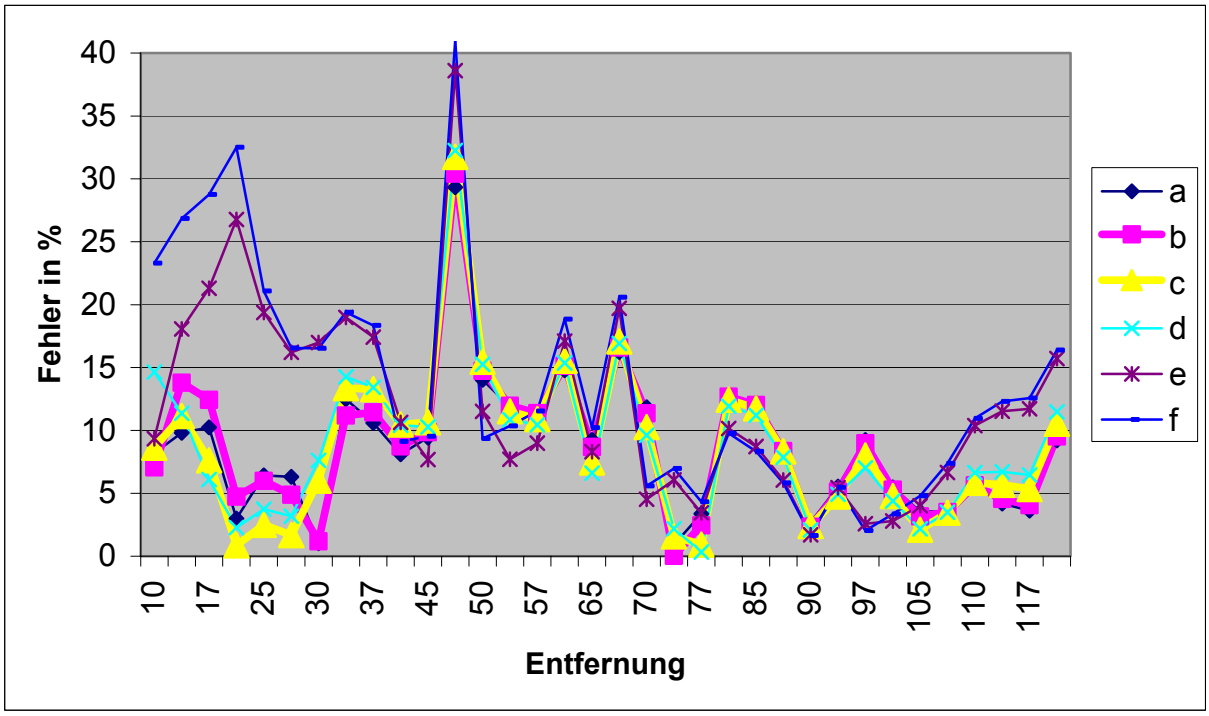


Abbildung 5.6: Fehlerkurve verschiedener Netzwerkkonfigurationen;
a) 3 schichtiges Netz: Zwischenschicht hat 4 Units
b) 3 schichtiges Netz: Zwischenschicht hat 8 Units
c) 3 schichtiges Netz: Zwischenschicht hat 12 Units
d) 3 schichtiges Netz: Zwischenschicht hat 16 Units
e) 4 schichtiges Netz: Zwischenschicht 1 hat 5 Units, Zwischenschicht 2 hat 5 Units
f) 4 schichtiges Netz: Zwischenschicht 1 hat 9 Units, Zwischenschicht 2 hat 9 Units

Die Konfigurationen b und c verhalten sich fast genauso wie die Konfigurationen a und b in Punkt 5.6. Genauer gesagt: Bei einem 3 schichtigen Neuronalen Netzwerk mit 8 bzw. 12 Units in der Zwischenschicht ist das Ergebnis bei einer Eingabe von 3 ungefilterten Sensordaten beinahe das selbe wie bei der Eingabe von 5 ungefilterten Sensordaten mit dem einzigen Unterschied, dass bei der Eingabe mit 3 Sensordaten höhere Fehlerspitzen entstehen können.

5.9 Zusammenfassung & Diskussion

Die Untersuchung der Ausgangspegel des Entfernungsmesssensors hat ergeben, dass man nur aufgrund eines konkreten Ausgangspegels eine Aussage über die Entfernung treffen kann. Die relative Größe von Schwankungen des Ausgangspegels oder ein zeitliches Auftreten von Schwankungen (mit Ausnahme bei Entfernungen über 120cm) hängen jedoch nicht mit der

Entfernung zusammen. Wenn sich nun die Ausgangskennlinie des Sensors aufgrund von Stromschwankungen verschiebt, ist der Fehler weder vom Neuronalen Netzwerk noch bei der linearen Approximation detektierbar.

Ändert sich nicht die komplette Ausgangskennlinie des Sensors sondern nur die Größe der Fehlerstellen, hat das nur Auswirkungen auf das Neuronale Netzwerk weil dieses die Filterung dieser fehlerhaften Pegel aufgrund von konkreten Werten ausführt. Dagegen hat die Implementierung der Umrechnung mittels linearer Approximation damit keine Probleme, da bei dieser Variante solche Fehlerstellen aufgrund ihrer relativen Differenz zu den nachfolgenden und vorgehenden Ausgangspegeln erkannt und herausgefiltert werden.

Ein großer Nachteil der Implementierung eines Neuronalen Netzwerkes liegt darin, dass der Musterassoziators auf das Finden von lokalen Minima begrenzt ist. Es kann somit nicht garantiert werden, dass ein optimales Netz für die Umrechnung der Sensordaten erstellt wird. Es ist dem Zufall überlassen, ob die Anfangsinitialisierung der Leitungsgewichte zu einem gut angepassten Netz führt oder nicht.

Beim Experimentieren mit verschiedenen Parametern und Arten von Eingabevektoren ergaben sich die Vergleichswerte in *Tabelle 6.1*:

	MaxMinMean von 5 Ausgangspegel	Maximum von 5 Ausgangspegel	5 aufeinanderfolgende Ausgangspegel	3 aufeinanderfolgende Ausgangspegel
durchschnittlicher Fehler	6%	9%	8%	9%
maximaler Fehler	12%	30%	24%	32%

Tabelle 6.1: Vergleich der Fehlerkurven der in Punkt 5 implementierten Neuronalen Netzwerke

Diese Werte kann man noch verbessern indem man die Ausgangspegel je nach Größe von verschiedenen, parallel arbeitenden Neuronalen Netzwerken verarbeiten lässt.
Im Vergleich zur der oben gezeigten Fehlerstatistik ergeben sich bei linearer Approximation, welche mit dem einfachen Algorithmus zur Erkennung fehlerhafter Pegel arbeitet, die Werte in *Tabelle 6.2*:

	lineare Approximation
durchschnittlicher Fehler	5%
maximaler Fehler	19%

Tabelle 6.2: Fehlerkurven der in Punkt 5 implementierten linearen Approximation

Für alle von mir in *Punkt 5* implementierten Mechanismen gilt, dass sie zur Erkennung von „unendlichen“ Entfernungen mit dem unter Punkt 4.2.4 vorgestellten Algorithmus arbeiten. Dieser weist eine Treffersicherheit von circa 95% für das Erkennen von „unendlichen“ Entfernungen auf.

6 Fazit

Die Ergebnisse dieser Arbeit zeigen, dass es für die Umwandlung der Sensordaten in cm-Angaben wesentlich effektiver, statt einem Neuronalen Netzwerk eine einfache Approximation zu verwenden. Der Fehler vom Neuronalen Netzwerk ist im besten Fall nur um Weniges kleiner als der Fehler von der konventionellen Approximation, aber weil die Funktionsangleichung des Muster-Assoziators auf das Finden von lokalen Minima begrenzt ist, wird nicht garantiert, dass ein optimales Netz erstellt wird.

Die Implementierung der Sensordatenumrechnung mittels linearer Approximation muss vor dem Einsatz mit dem Smart Sensor noch wesentlich verbessert werden. Unter anderem muss der Algorithmus dahingehend verändert werden, dass dieser nicht mit einer fixen Zuordnung, Referenzpunkte zu Entfernung, arbeitet, sondern dass je nach Krümmung der Ausgangskennlinie des Sensors die Referenzpunkte dynamisch gesetzt werden.

Literaturverzeichnis

[Br91] R. Brause, Neuronale Netze: Eine Einführung in die Neuroinformatik. Stuttgart: Teubner, 1991

[Di02] A. Dias. Documentation of the Smart Car Demonstrator. Research Report 45/2002, Technische Universität Wien, Institut für Technische Informatik, Wien, Österreich, 2002

[El02] W. Elmenreich. Sensor Fusion in Time Triggered Systems. Dissertation. Technische Universität Wien, Institut für Technische Informatik, Wien, Österreich, 2002

[Ho93] N. Hoffmann. Kleines Handbuch Neuronale Netze: Anwendungsorientiertes Wissen zum Lernen und Nachschlagen. Braunschweig, Wiesbaden. Vieweg, 1993

[Ko82] W. H. Ko, C. D. Fung. VLSI and Intelligent Transducers. Sensors and Actuators, (2):239-250, 1982

[Lu97] G. Luger, W. Stubblefield: Artificial Intelligence, Structures and Strategies for Complex Problem Solving. Addison Wesley Verlag Reading, Mass., 1997

[Ma92] A. Mazzetti: Praktische Einführung in Neuronale Netze. Verlag Heinz Hesse, 1992

[Ni90] N. Nilsson. Learning Machines (1965). Morgan Kaufmann, San Mateo, Neuausgabe 1990

[Ri91] H. Ritter, T. Martinez, K. Schulten. Neuronale Netze: Eine Einführung in die Neuroninformatik selbstorganisierender Netzwerke. Bonn etc: Addsion-Wesley, 1991

[Ro58] F. Rosenblatt. The Perceptron: A Probalistic Model for Information Storage and Organization in the Brain. Psychological Review 65:386-408

[Ro93] R. Rojas. Theorie der Neuronalen Netze, Eine systematische Einführung. Springer-Verlag, 1993

[St88] G. Stone. An Analysis of the Delta Rule and the Learning of Statistical Associations. Rumelhart, 1988

Anhang A: Datenblatt zum GP2D02 Sensor

GP2D02

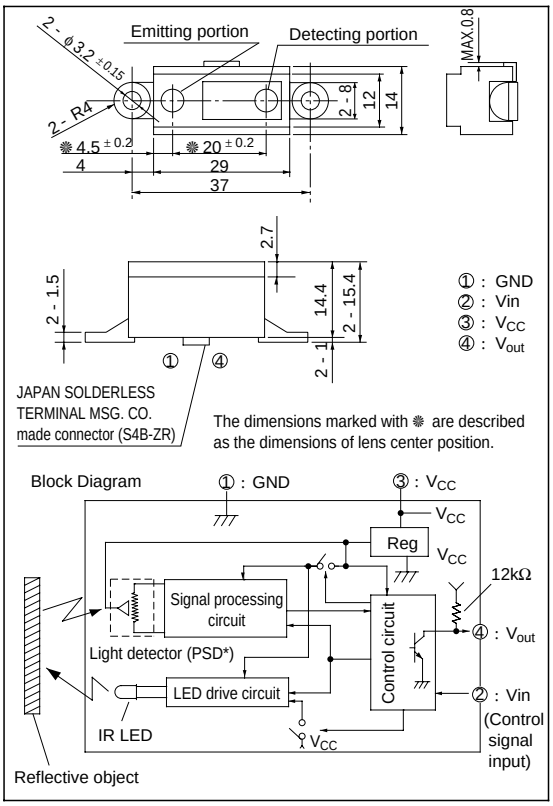
Compact, High Sensitive Distance Measuring Sensor

■ Features

1. Impervious to color and reflectivity of reflective object
2. High precision distance measurement output for direct connection to microcomputer
3. Low dissipation current at OFF-state
   (dissipation current at OFF-state : TYP. 3 μA)
4. Capable of changing of distance measuring range through change the optical portion (lens)

■ Outline Dimensions

(Unit : mm)



■ Applications

1. Sanitary sensors
2. Human body sensors for consumer products such as electric fans and air conditioners
3. Garage sensors
   * PSD : Position Sensitive Detector

■ Absolute Maximum Ratings

(Ta=25˚C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	- 0.3 to + 10	V
*1Input terminal voltage	V_{in}	- 0.3 to + 3	V
Output terminal voltage	BV_O	- 0.3 to + 10	V
Operating temperature	T_{opr}	- 10 to + 60	˚C
Storage temperature	T_{stg}	- 40 to + 70	˚C

*1 Open drain operation input

■ Operating Supply Voltage

Symbol	Rating	Unit
V_{CC}	4.4 to 7	V

■ Electro-optical Characteristics

(Ta=25˚C,Vcc=5V)

Parameter		Symbol	Conditions		MIN.	TYP.	MAX.	Unit
Distance measuring range		Δ L	*1		10	-	80	cm
Output terminal voltage		V_{OH}	Output voltage at High	L = 20cm *1	V_{CC} - 0.3	-	-	V
		V_{OL}	Output voltage at Low		-	-	0.3	V
Distance characteristics of output		D	L = 80cm, *1		-	75	-	DEC
		Δ D	Output change at L=80 cm to 20 cm, *1		48	58	68	DEC
Dissipation current	at operating	I_{CC}	L = 20cm, *1, *2		-	22	35	mA
	at OFF-state	I_{off}	L = 20cm, *1		-	3	8	μA
Vin terminal current		I_{vin}	Vin = 0V		-	- 170	- 280	μA

Note) L : Distance to reflective object

DEC : Decimalized value of sensor output (8-bit serial)

*1 Reflective object : White paper (reflectivity : 90%)

*2 Average dissipation current value during distance measuring operation when detecting of input signal, Vin as shown in the timing chart

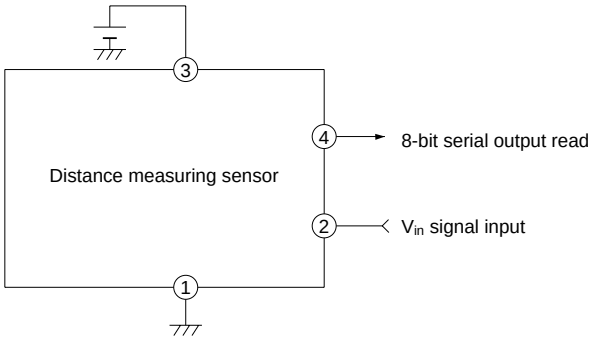
*3 Vin terminal : Open drain drive input.

Conditions : Vin terminal current at Vin OFF-state : -1 μA

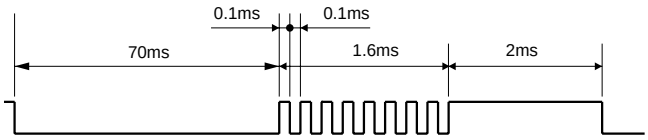
Vin terminal current at Vin ON-state : 0.3V

■ Test Circuit

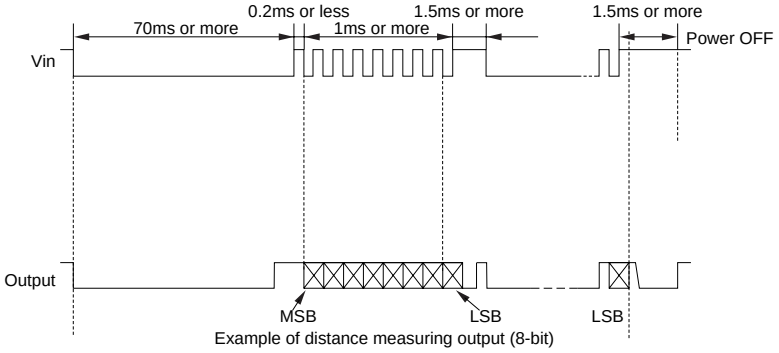
1. Test circuit



2. Vin input signal for measurement



■ Timing Chart



Example of distance measuring output (8-bit)

Fig. 1 Distance Measuring Output vs. Distance to Reflective Object

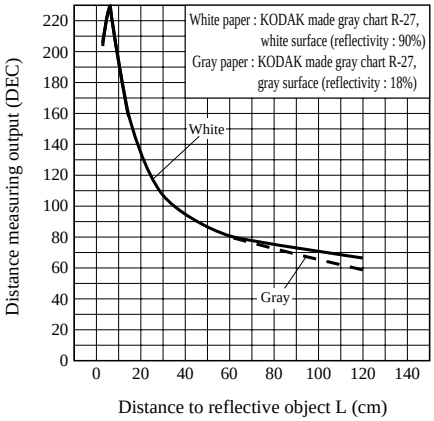
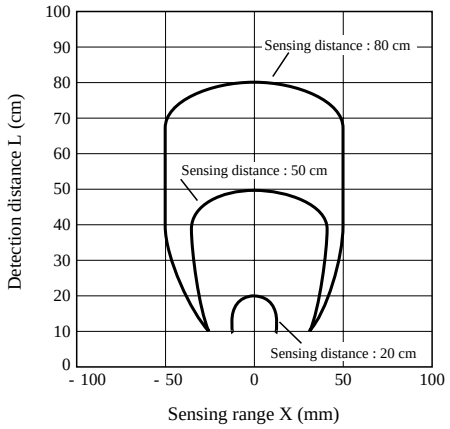


Fig. 2 Detection Distance vs. Sensing Range



Test Method for Sensing Range Characteristics

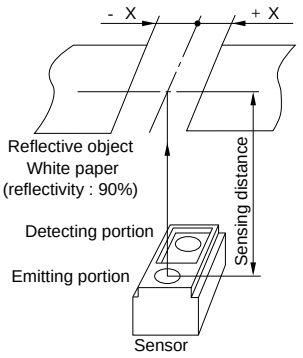
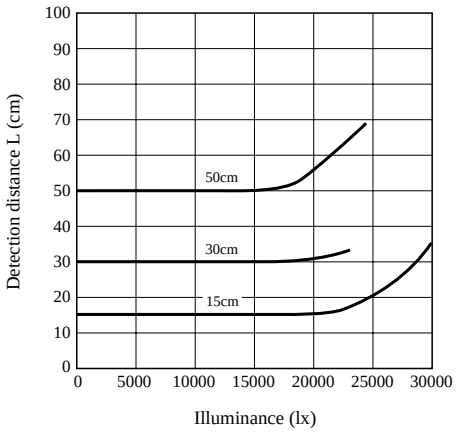


Fig. 3 Detection Distance vs. Illuminance



Test Method for Anti External Disturbing Light Characteristics