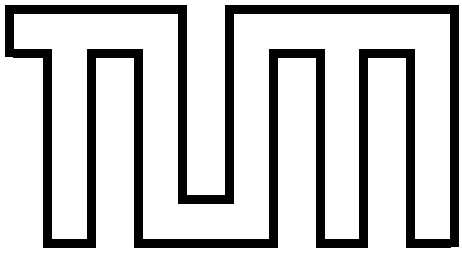


# Technische Universität München

## Fakultät für Informatik

**Forschungs- und Lehreinheit Informatik VI**


Neuronale Netze - Eine Einführung in Lernverfahren

Seminar Kognitive Robotik (SS12)


**Martin Schrimpf**


**Betreuer:** Florian Röhrbein

**Leitung:** Prof. Alois Knoll

**Abgabetermin:** 07. Juli 2012

# Inhaltsverzeichnis

# 1 Einführung und Grundbegriffe

## 1.1 Allgemeine Beschreibung

Die elementare Frage in dieser Einführung ist die nach der Definition eines neuronalen Netzwerkes. Generell lässt sich dies wie folgt beantworten: Ein neuronales Netzwerk ist ein System, das aus miteinander verbundenen Elementen besteht, die Informationen verarbeiten können und als Neuron bezeichnet werden [3] [2]. Dabei unterscheidet man zwischen biologischen und künstlichen neuronalen Netzwerken. Im ersten Fall sind die Neuronen Nervenzellen und das Netzwerk ist Teil des Nervensystems eines biologischen Organismus. Verarbeitet werden biologische Informationen, im Wesentlichen Nervenimpulse. Beim Fokus dieser Arbeit, den künstlichen neuronalen Netzwerken, sind die Neuronen hingegen als mathematische Modelle mit mehreren Ein- und Ausgängen realisiert, deren mathematisches Verhalten sich mitunter an den biologischen Neuronen orientiert. Die verarbeiteten Informationen werden hierbei im Allgemeinen als Eingangsmuster bezeichnet und als Ausgangsmuster wieder ausgegeben (vgl. Abb. 1.1), was das Netz durch seine Neuronen und deren Aktivierung erreicht. Ein Neuron erzeugt im Wesentlichen erst dann eine Ausgabe, wenn der kumulative Effekt der Eingabereize einen bestimmten Schwellwert übersteigt. Die Gewichtungen, welche die synaptischen neuronalen Verknüpfungen biologischer Netze nachbilden, verstärken entweder die Eingabesignale (exzitatorische) für das Neuron oder schwächen sie ab (inhibitorische). Für die Gewichtungen werden in der Regel reelle Zahlen verwendet, manchmal finden aber auch binäre Zahlen Verwendung.

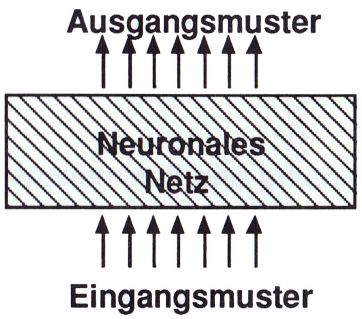


Abbildung 1.1: Schematische Darstellung eines neuronalen Netzes [3]

Die Neuronen lassen sich in drei verschiedene Klassen (Layer) gruppieren: Die Eingangsschicht (Input-Layer), an welcher das externe Eingangsmuster anliegt, die verdeckte Schicht, worin die Neuronen nur interne Verbindungen zu anderen Neuronen und keine externen Verknüpfungen haben, sowie die Ausgangsschicht (Output-Layer) mit Neuronen, die das Ausgangsmuster ausgeben. Dabei ist anzumerken, dass die Anzahl der Neuronen in der verdeckten Schicht oft sehr groß sein kann und man diese Schicht in weitere Schichten unterteilen kann, die "Hidden-Layers" (Siehe hierzu auch Abbildung 1.2). Pflanzt sich das Signal innerhalb des Netzes von den Eingabe- zum Ausgabeknoten nur vorwärts fort, so spricht man von einem Feedforward-Netzwerk. Wandern die Signale auch seitlich oder rückwärts durch die Neuronen, so bezeichnet man dies als rekurrentes Netz. Weiterhin kann ein Netz einschichtig mit einer einzigen Neuronenschicht oder mehrschichtig mit mehreren Schichten sein. Außerdem kann man solche Netzwerke noch hinsichtlicher Synchronität (zeitge-

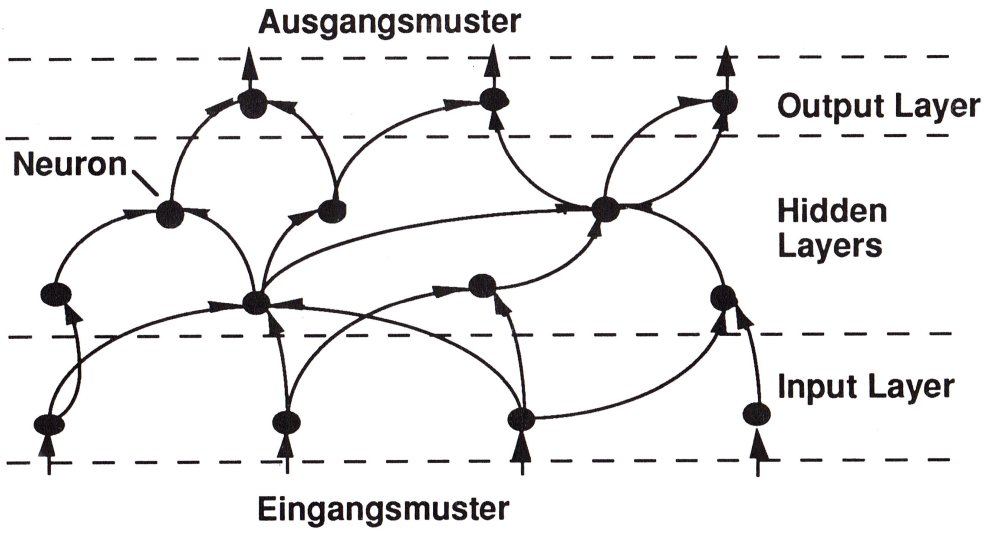


Abbildung 1.2: Detailliertere Darstellung eines künstlichen neuronalen Netzes [3]

steuert) bzw. Asynchronität (keine zeitliche Regulierung) unterscheiden. Im synchronen Operationsmodus werden die Neuronenzustände nach einer bestimmten Reihenfolge und zu bestimmten Zeiten aktualisiert, wohingegen im asynchronen Modus die Aktualisierung in belieber Reihenfolge stattfinden kann, wobei keine Zeitpunkte festgelegt sind.

Die Herangehensweise zur Lösung eines Problems bei einem neuronalen Netz ist etwas anders als von mathematischen bzw. informationstechnischen Ansätzen gewöhnt: Nicht die Untersuchung des Problems steht im Vordergrund, sondern die Frage, wie man die Parameter wählen muss, um zwischen Eingangs- und Ausgangsmuster ein bestimmtes Übertragungsverhalten zu realisieren. Diese Fragestellung lässt sich durch mehrere Beispielmuster am Ein- bzw. Ausgang lösen sowie einem Optimierungsverfahren, welches versucht, die Parameter des Netzes so zu bestimmen, dass das gewünschte Übertragungsverhalten erreicht wird. Diese Phase wird bei einem neuronalen Netzwerk als Lernphase oder auch Trainingsphase bezeichnet, in der das Netz aus Beispielen lernt, seine Parameter so anzupassen, dass es das gewünschte Übertragungsverhalten annimmt. Lernen bedeutet in diesem Sinne, dass ein neuronales Netz versucht, seine Parameter, also die Gewichtungen, so einzustellen, dass es aus den präsentierten Referenzeingangsmustern Ausgangsmuster erzeugt, welche den Referenzausgangsmustern möglichst ähnlich sind.

Zusammenfassend ist ein künstliches neuronales Netz also als System zur Informationsverarbeitung mit Hilfe von einfachen vernetzten Elementen mit gerichteten Ein- und Ausgängen und gewichteten Verbindungen zu verstehen, das Eingangsmuster verarbeitet und daraus Ausgangsmuster erzeugt. Die Anwendung besteht dabei aus zwei Phasen: In der Lerphase werden die Gewichtungen anhand von Beispielen so ermittelt, dass das Netz den Zusammenhang zwischen den Beispielmustern erfassen kann. In der zweiten, eigentlichen Anwendungsphase löst das neuronale Netz Musterverarbeitungsaufgaben, indem es mit

seinen gelernten Gewichtungen aus Eingangsmustern Ausgaben erzeugt.

## 1.2 Besondere Eigenschaften

Die im vorangehenden Kapitel beschriebene Lernfähigkeit von neuronalen Netzen verleiht den neuronalen Algorithmen eine besondere Eigenschaft, über die die meisten anderen Algorithmen nicht verfügen, welche aber beim Lösen von Problemen - insbesondere aus dem Bereich der künstlichen Intelligenz - sehr vorteilhaft und teilweise notwendig sind. Dadurch haben neuronale Netze in den letzten Jahren sehr schnell an Popularität gewonnen und entwickeln sich weiter in einem raschen Tempo. Doch es gibt noch weitere Eigenschaften, die neuronale Netze von klassischen Methoden unterscheiden.

### 1.2.1 Lernfähigkeit

Aus der bereits erwähnten Lernfähigkeit lassen sich interessante Konsequenzen ableiten. Nach aktuellem Stand werden Computerprogramme meistens imperativ geschrieben, der Mensch muss also alle Einzelschritte zur Vorgehensweise bei der Lösung einer Aufgabe eigenhändig programmieren. Dafür muss die Aufgabe im Vorfeld genau analysiert werden und bei jeder Systemänderung muss auch der Code angepasst werden. Mit neuronalen Netzen wird diese Aufgabe massiv erleichtert: Das Programmieren wird durch das Training des Computers ersetzt und auch nachträglich lässt sich das System durch Lernen leicht anpassen.

Neuronale Netze können weiterhin unter bestimmten Bedingungen aus einer sehr großen Anzahl von unterschiedlichen Beispielmusterpaaren systematisch die essentiellen Zusammenhänge zwischen diesen Musterpaaren erkennen. Damit ist das Netzwerk zu weit mehr als nur reinem "Input-Output-Mapping" in der Lage: Es kann die essentielle Systematik der Musterverarbeitungsaufgabe erlernen und aus einer Vielzahl von Beispielen zu einer Verallgemeinerung der präsentierten Einzelexemplare gelangen. So hat das neuronale Netz die Fähigkeit, in der Anwendungsphase Eingangsmuster korrekt zu verarbeiten, die es in der Trainingsphase noch nie gesehen hat. Insofern diese Muster der selben Systematik der Lernbeispiele entsprechen, können sie zum Teil völlig andersartig sein als die in der Lernphase präsentierten Beispiele. Einige Typen neuronaler Netze besitzen die Fähigkeit, nicht nur in der Trainingsphase zu lernen, sondern dies auch in der Anwendungsphase fortzuführen. Dabei werden die erlernten Gewichtungen nicht konstant gehalten, sondern werden laufend an die aktuellen Verhältnisse angepasst. Diese Nachadaptierung mit aktuellen Mustern ist beispielsweise bei der Zusammenarbeit mit zeitvarianten Systemen sehr wünschenswert, da hier eine Anpassung an sich langsam verändernde Bedingungen erforderlich ist.

### 1.2.2 Fähigkeit zur Verarbeitung fehlerhafter Information

Nahezu alle neuronalen Netze sind in der Lage, aus fehlerhafter oder gar unvollständiger Information ein korrektes Ausgangsmuster zu generieren. Klassische (wie etwa regelbasierte) Methoden sind den neuronalen Netzen hier klar unterlegen, da diese bei solchen Eingaben meistens versagen. Neuronale Netze können sogenannte "unscharfe Informationen" verarbeiten, wohingegen die konventionellen Systeme auf die Verarbeitung von korrekter und vollständiger Information ausgelegt sind. Die "alten" Systeme sind in gewisser Weise nicht an das alltägliche Leben angepasst, da hier viele Informationen unscharf sind wie beispielsweise ein schnell gesprochenes Wort. Das menschliche Gehirn kann so ein Wort meistens aus dem Kontext heraus trotzdem erfolgreich verarbeiten. Somit ist es auch nicht verwunderlich, dass neuronale Netze gerade bei der maschinellen Verarbeitung von Sprach- und Bildsignalen erfolgreich eingesetzt werden können.

### 1.2.3 Parallelität

Wie auch in Abbildung 2 ersichtlich, sind neuronale Netze massiv parallel. Die Neuronen können dabei als autonome Systeme betrachtet werden, deren interne Operationen voneinander unabhängig sind und die nur durch ihre Verbindungen miteinander kommunizieren. Somit lässt sich ein neuronales Netz als ein Netzwerk von unabhängigen, asynchronen Einzelsystemen betrachten. Würde man dieses Verhalten mit konventionellen Rechnern simulieren, so würde man das Simulationsprogramm für ein Neuron auf einem Prozessor realisieren, wobei für diesen Fall ein sehr simpler Prozessor bereits ausreicht. Verbindet man nun sämtliche Einzelprozessoren, lässt sich damit das Gesamtnetz darstellen. Die Simulationsprogramme der einzelnen Neuronen müssen dann nicht mehr sequentiell auf einem Rechner ablaufen, sondern können parallel auf allen Prozessoren berechnet werden. Will man das Netz nun vergrößeren, so muss man nur in gleichem Maße die Prozessorenanzahl vergrößern, was sich nicht negativ auf die Rechenzeit für das Netz auswirkt. Der Grund hierfür ist, dass jeder Prozessor separat mit normaler Geschwindigkeit seiner Rechenleistung zur Gesamtleistung des Netzes beitragen kann. Ein weiterer Aspekt ist der günstige Preis: Aufgrund der einfachen Ausführung der Prozessoren sind die Einzelelemente eines solchen Hardware-Netzes sehr billig und große Netze können kostengünstig realisiert werden und trotzdem über eine hohe Rechenleistung verfügen.

Im Kontrast zu diesem neuronalen Ansatz der parallel arbeitenden Rechner stehen die Bemühungen, sequentielle Algorithmen durch spezielle Computer automatisch zu parallelisieren, um sie dann auf Parallelrechnern ablaufen zu lassen, die mit einer größeren Anzahl von relativ komplexen Prozessoren in paralleler Verarbeitungsweise operieren. Nach aktuellen Erwartungen werden sich beide Lösungsansätze in Zukunft ergänzen. Zum einen werden die neuronalen Netze bei der parallelen Verarbeitung von unscharfer Information (z.B. Muster- und Signalverarbeitung) zum Einsatz kommen. Auf der anderen Seite werden Parallelrechner voraussichtlich bei Problemen eingesetzt, bei denen exakte Informationen in großer

Menge und mit großer Geschwindigkeit verarbeitet werden müssen, beispielsweise bei der Berechnung komplexer numerischer Probleme. Der entscheidende Unterschied der beiden Ansätze ist, dass die neuronalen Netze eine natürliche Parallelität aufweisen, wohingegen Parallelrechner diese Parallelität erst bei jedem neu zu bearbeitenden Problem erzeugen müssen. Dadurch ist die Parallelisierung eines sequentiellen Algorithmus sehr schwierig und aufwändig. Im Gegensatz dazu führt ein neuronales Netz diese Parallelisierung durch seine Struktur bereits automatisch durch, die Überführung - der schwierigste Schritt für einen Parallelrechner mit Compiler - wird völlig automatisch in eine parallele Form überführt.

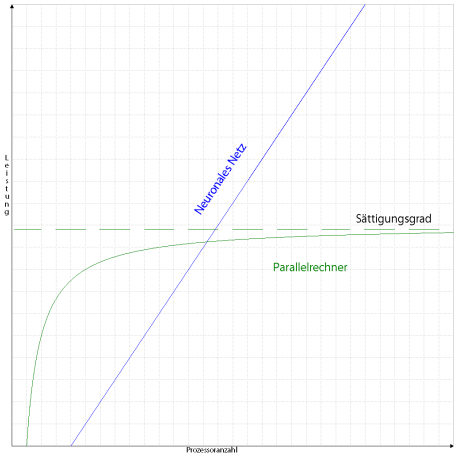


Abbildung 1.3: Beispielhafte Leistungskurve für neuronale Netze und Parallelrechner

Die neuronalen Netze besitzen noch einen weiteren Vorteil: Bei ihnen lässt sich die Anzahl der Elemente beliebig variieren - bei einem Parallelrechner bringt dies neue Optimierungsprobleme mit sich. Das Problem vergrößert sich durch die erhöhte Komplexität der Kommunikation zwischen den Parallelprozessoren; die Kommunikation der neuronalen Netze besteht nur aus einer gewichteten Verbindung und ist damit extrem einfach. Dies zeigt sich auch im Vergleich der Leistungskurve beider Systeme bei einer Vervielfachung der parallelen Elemente (Abbildung 1.3). Bei den neuronalen Netzen ist das Verhältnis Rechenleistung zu Verarbeitungselementen konstant. Bei den Parallelrechnern hingegen nähert sich die Leistungskurve einem Grenzwert an, bei dem eine Erhöhung der Anzahl der Elemente zu keiner Erhöhung des Durchsatzes mehr führt, da der Aufwand für die Prozessor-Kommunikation zu groß wird.

### 1.2.4 Ausfallsicherheit

Bei einem System, das aus so vielen parallel arbeitenden Elementen zusammengesetzt ist, besteht natürlich auch die Möglichkeit des Ausfalls einiger Bestandteile. Bei einem Parallelrechner würde dies zu einem Totalausfall führen, da jeder Prozessor essentiell für das Gesamtsystem ist und insbesondere der Ausfall der Kommunikation zwischen Prozessen zu erheblichen Problemen führen kann. Wollte man den Parallelrechner weiterbetreiben, so müsste man entweder die Komponenten austauschen oder alle Anwendungen der geringeren Prozessoranzahl anpassen, beispielsweise durch Neukompilierung. Bei einem neuronalen Netz hingegen ist bei dem Ausfall einzelner Neuronen das Gesamtnetz noch intakt und weist keine nennenswerte Veränderung auf. Allerdings gilt das nur unter der Annahme, dass die Anzahl der ausgefallenen Elemente relativ klein im Vergleich zur Gesamtanzahl der Neuronen ist. Das neuronale Netz hat also auch seine Grenzen, welche dabei eher fließend ist: Das Netzverhalten wird kontinuierlich schlechter mit der Anzahl der ausgefallenen

Komponenten. Ein plötzlicher Totalausfall tritt in der Regel nicht auf.

Diese Eigenschaft der Ausfallsicherheit lässt sich anhand einiger bereits erwähnter Eigenschaften erklären. Durch die massive Parallelität ist die Gesamtfunktionalität des Netzes stark verteilt: Jedes einzelne Element ist sehr einfach aufgebaut und ist für sich allein nicht sonderlich leistungsfähig. Da die Effektivität neuronaler Netze durch eine starke, gewichtete Konnektivität und eine Aufteilung der Aufgaben erreicht wird, kompensieren intakte Elemente den Ausfall eines einfachen Elements. Des Weiteren ist das Netzwerk im Allgemeinen in der Lage, unscharfe Informationen zu verarbeiten. So kann auch fehlerhafter Input von den eigenen Neuronen bewältigt werden und das Netzwerk weiterhin funktionsfähig bleiben.

# 2 Mathematische Konzepte

Im Folgenden werden einige Themen aus der Mathematik nebst ihren Einsatzgebieten erklärt. Dabei wird sicherlich nicht auf alles eingegangen, jedoch soll ein grober Überblick verschafft werden. Zugrundeliegend für die folgenden Erläuterungen sind mathematische Grundkenntnisse auf Abiturniveau.

## 2.1 Aufbau eines künstlichen Neurons

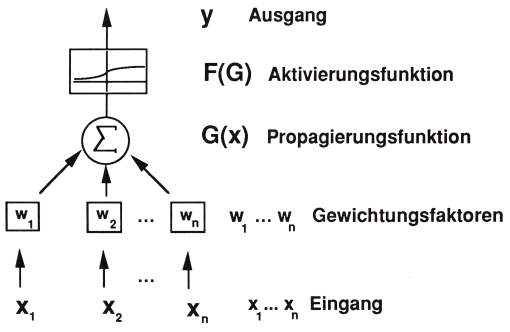


Abbildung 2.1: Übersicht Propagierungs- und Aktivierungsfunktion

Wie bereits in der Einführung erwähnt, besitzt ein Verarbeitungselement Eingänge $x_1 ... x_n$, die mithilfe der Gewichtungsfaktoren $w_1 ... w_n$ gewichtet werden und das Ergebnis im Ausgang $y$ ausgibt. Das eigentliche Neuron wird durch die Propagierungsfunktion $G(x)$ und die Aktivierungsfunktion $F(x)$ repräsentiert. Die Propagierungsfunktion ist in den meisten Fällen eine Summierungsfunktion, welche die gewichteten Eingangsgrößen summiert und ausgibt. Gelegentlich findet auch eine Multiplikationsfunktion Verwendung oder manchmal sogar eine Funktion, welche als Resultat den euklidischen Abstand zwischen den Gewichtungen und den Eingangswerten liefert. Die Ausgabe der Propagierungsfunktion wird von der Aktivierungsfunktion auf vielfältigere Art und Weise weiterverarbeitet, wodurch ein nichtlineares Verhalten erreicht wird. Zusammenfassend sind die Propagierungs- und Aktivierungsfunktion in Abb. 2.1 dargestellt. Die Beziehung zwischen den Eingangsgrößen und der Ausgabe eines einzelnen Verarbeitungselements ist damit durch $y = F(\sum_{i=1}^{n} w_i \times x_i)$ definiert. In Abbildung 2.2 sind die am

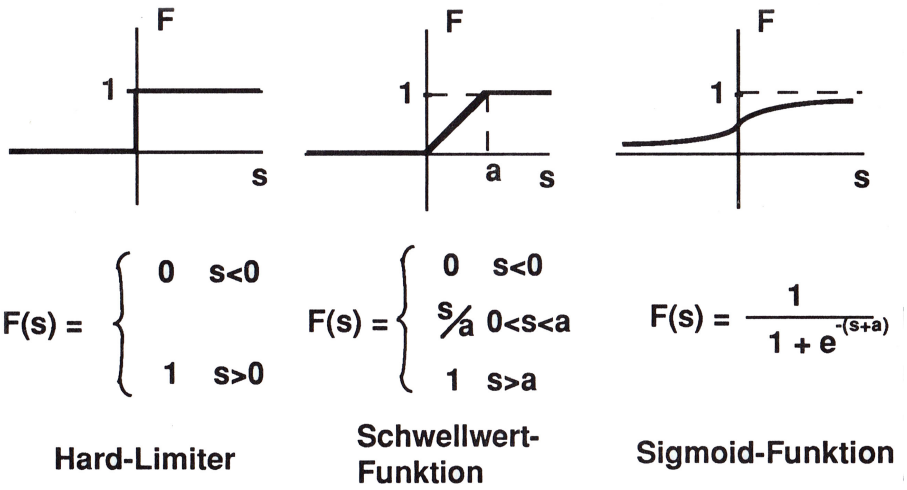


Abbildung 2.2: Die häufigsten Aktivierungsfunktionen in neuronalen Netzen [3]

häufigsten verwendeten Aktivierungsfunktionen dargestellt, wobei $s$ den Ausgang der Propagierungsfunktion und zugleich den Eingang der Aktivierungsfunktion bezeichnet und $a$ ein frei wählbarer Parameter zur Feineinstellung des Netzes ist. Der "Hard-Limiter" findet vor allem bei binären Netzwerken, also bei Netzen mit Ausgabewerten "0" und "1", Verwendung. Durch die häufige Nutzung solcher Netze kommt der Hard-Limiter oft zum Einsatz. Weniger populär ist die "Schwellwert-Funktion", die sich mehr für spezielle Fälle eignet. Will man kontinuierliche Ausgangswerte erreichen, so verwendet man in aller Regel die "Sigmoid-Funktion". Es lässt sich leicht erkennen, dass diese Funktion für $s \to \infty$ gegen 1 und für $s \to -\infty$ gegen 0 konvergiert. Für $s = 0$ erreicht die Funktion den Mittelwert 0.5 (für $a = 0$), womit der Wertebereich der Sigmoid-Funktion wie bei den anderen bisher angesprochenen Aktivierungsfunktionen zwischen 0 und 1 liegt. Genau das ist der typische Wertebereich der Ausgangsgrößen aller neuronalen Netze.

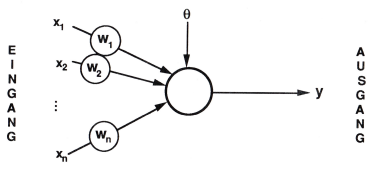


Abbildung 2.3: Einzelnes Verarbeitungselement

In Bild 2.3 ist nochmal ein Verarbeitungselement zu sehen, welches als neu eingeführtes Element am Eingang die Größe $\delta$, welche wie der soeben besprochene Faktor $a$ die Funktion eines Parameters zur Feineinstellung übernimmt. So sorgt $\delta$ dafür, dass der Ausgang des künstlichen Neurons im Mittel eher positiv oder negativ ausfällt. In vielen Fällen jedoch kann man $\delta$ auf 0 setzen. Es gibt aber auch viele Beispiele, in denen es sinnvoll ist, diese Variable auf einen Wert $\neq 0$ zu setzen, sodass der Parameter bei der mathematischen Beschreibung eines Verarbeitungselementes zu berücksichtigen ist. Damit ergibt sich folgende

Gleichung für die Ausgangsgröße $y$ des Verarbeitungselements:

$$y = F(\sum_{i=1}^{n} w_i \times x_i + \delta) \tag{1}$$

Hier empfiehlt es sich, zur Vektor-Schreibweise überzugehen. Gleichung 2 lässt sich mit den erweiterten Vektoren $\underline{x}$ und $\underline{w}$ darstellen:

$$\underline{x} = [x_1, x_2, ..., x_n, 1]^T \tag{2}$$

$$\underline{w} = [w_1, w_2, ..., w_n, \delta]^T \tag{3}$$

Somit kann man Gleichung 2 auch folgendermaßen schreiben:

$$y = F(\underline{x}^T \times \underline{w}) \tag{4}$$

Aus den Gleichungen 3 und 4 ist zu erkennen, dass der Faktor $\delta$ eine permanente Eingangsgröße von $x_{n+1} = 1$ erhält. Betrachtet man die häufigste Aktivierungsfunktion, den Hard-Limiter, so ergibt sich:

$$y = \begin{cases} 1, \ falls \, \underline{x}^T \times \underline{w} \geq 0 \\ 0, \ falls \, \underline{x}^T \times \underline{w} < 0 \end{cases} \tag{5}$$

und für die Sigmoid-Funktion:

$$y = \frac{1}{1+e^{-(\underline{x}^T \times \underline{w} + a)}} \tag{6}$$

## 2.2 Mathematische Beschreibung des Gesamtnetzes

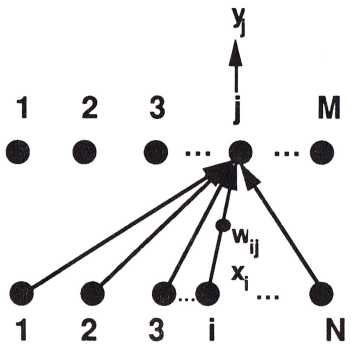


Abbildung 2.4: Schematische Darstellung der Übergänge eines zweischichtigen Netzes [3]

Die Ausgangswerte der einzelnen Neuronen $j$ lassen sich mit einer indizierten Form der Gleichung 4 darstellen:

$$y_j = F(\underline{x}^T \times \underline{w}_j) \tag{7}$$

Diese Verarbeitungselemente lassen sich nun zu Netzwerkschichten zusammenfassen, wobei der Ausgangswert eines Neurons der 1. Schicht einem Eingangswert eines Neurons der 2. Schicht entsprechen kann. Der Vektor $\underline{w}_j$ enthält dabei die Gewichtungen der Verbindungen von den N Neuronen der unteren Schicht zum $j$-ten Neuron der oberen Schicht und ist somit ein Vektor mit der Dimension N. Für die M Neuronen der oberen Schicht existieren weiterhin M Gewichtungsvektoren $w_j$, $j = 1, ..., M$ mit jeweils N Komponenten. Summeran gibt es also $N \times M$ Gewichtungsfaktoren. $w_j$ lässt sich damit auch in der folgenden Form schreiben:

$$\underline{w}_j = [w_{1j}, w_{2j}, ..., w_{Nj}]^T \tag{8}$$

Fasst man die Gleichungen 7 und 8 zusammen, so kann man auch schreiben:

$$[y_1, y_2, ..., y_M] = F(x^T \times [\underline{w}_1, \underline{w}_2, ..., \underline{w}_M]) \tag{9}$$

bzw. mit Einführung der Gewichtungsmatrix $W$:

$$\underline{y}^T = F(\underline{x}^T \times W) \tag{10}$$

Dabei gilt für die Matrix $W$:

$$W = [\underline{w}_1, \underline{w}_2, ..., \underline{w}_M] = \begin{pmatrix} w_{11} & ... & w_{1M} \\ ... & ... & ... \\ w_{N1} & ... & w_{NM} \end{pmatrix} \tag{11}$$

Die $N$ Zeilen entsprechen dabei der Anzahl der Neuronen der unteren Schicht und die $M$ Spalten der Zahl der Neuronen in der oberen Schicht. Ein einzelner Wert $w_{ij}$ der Matrix steht für die Gewichtung zwischen dem $i$-ten Element der unteren Schicht und dem $j$-ten Element der oberen Schicht.

Die Ausweitung der mathematischen Beschreibung von zwei Schichten auf das Gesamtnetz wird in Abbildung 2.5 dargestellt.

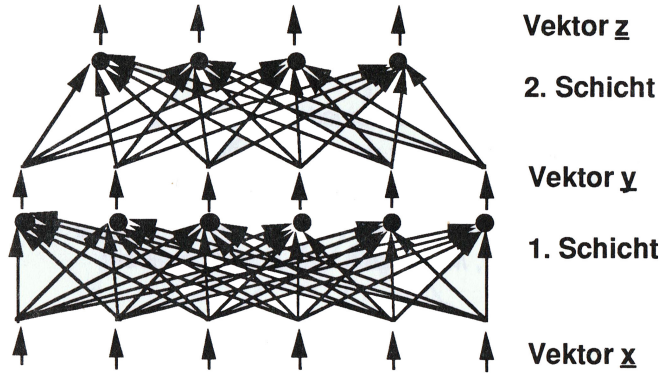


Abbildung 2.5: Zweischichtiges Netz [3]

Betrachtet man die zwei einschichtigen Netze des zweischichtigen Netzes, so lässt sich festhalten, dass das erste Netz aus dem Eingangsvektor $\underline{x}$ unter Einbeziehung der Gewichtungen in der Matrix $W_1$ den Ausgangsvektor $\underline{y}$ erzeugt. Mathematisch wird dies durch die Funktion F beschrieben:

$$\underline{y}^T = F(\underline{x}^T W_1) \tag{12}$$

Dieser Ausgangsvektor $\underline{y}$ der ersten Schicht wird durch die zweite Schicht wiederum als Eingangsvektor betrachtet und mithilfe der in $W_2$ enthaltenen Gewichtungen wird der Vektor $_z$ berechnet und ausgegeben. Aus mathematischer Sicht ist $\underline{z}$ als zweifache Matrixtransformation mit anschließender nichtlinearer Verzerrung des Eingangsvektors $\underline{x}$ beschrieben:

$$\begin{aligned} \underline{z}^T = F(\underline{y}^T \times W_2) &= \underbrace{F(F(\underline{x}^T \times W_1) \times W_2)}_{} \\ \underline{z}^T = & \qquad G(\underline{x}^T) \end{aligned} \tag{13}$$

Es lässt sich erkennen, dass die Transformation $G(\underline{x})$ nur von der für alle Verarbeitungselemente festgelegten Aktivierungsfunktion $F$ sowie den Werten der Matrizen $W_1$ und $W_2$

abhängt. Dadurch wird erneut das Grundprinzip des Lernens in neuronalen Netzen deutlich gemacht: Die Parameter (die Gewichtungen) der komplexen Transformation $G$ werden so bestimmt, dass bei einer Präsentation von Beispielsvektoren $\underline{x}$ die dazugehörigen Beispielvektoren $\underline{z}$ durch Transformation mit $G$ möglichst gut nachgebildet werden. Dieser Lernvorgang besteht aus der Bestimmung der Parameter in den Gewichtungsmatrizen. Dazu werden im folgenden Abschnitt einige grundlegende Prinzipien erläutert.

## 2.3 Lernverfahren

Zur Optimierung der Bestimmung der Gewichtungen eines neuronalen Netzwerks anhand von Beispielen für Ein- und Ausgangsmuster gibt es verschiedene Strategien. Die grundlegende Vorgehensweise ist jedoch fast immer gleich: Man hat eine bestimmte Anzahl von Beispielen vorliegen sowie ein Netzwerk, dessen Gewichtungen mit Startwerten besetzt ist. Diese Startwerte können zufällige Werte, Null oder nach anderen Kriterien ausgesucht sein. Daraufhin präsentiert man dem Netzwerk das erste Beispiel-Eingangsmuster und berechnet unter Einbeziehung der Anfangswerte für die Gewichtungen das entsprechende Ausgangsmuster. Im Folgenden vergleicht man dieses berechnete Ausgangsmuster mit dem Soll-Ausgangsmuster, woraus sich verschiedene Regeln herleiten lassen, mit denen man die Gewichtungen des Netzes entsprechend verändern kann. Diese Schritte werden für jedes vorliegende Beispiel angewendet, wobei man das Verfahren auch mehrfach iterativ wiederholen kann, bis eine Konvergenz eintritt. Die Regeln zur Veränderung der Gewichtungen sind der Grund für verschiedene Lernverfahren. Die allgemeine Systematik des Lernvorgangs in neuronalen Netzen ist in Abbildung 2.6 dargestellt. Im Bild lässt sich auch ein

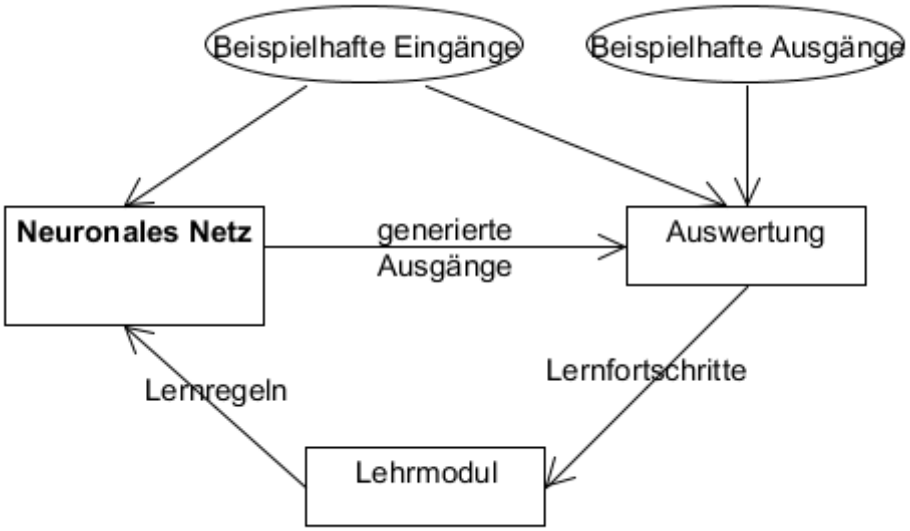


Abbildung 2.6: Systematische Darstellung des Lernvorgangs [3]

"Lehrmodul" erkennen, welches dafür sorgt, dass die Gewichtungen des Netzes so verändert werden, dass die generierten Ausgänge und die beispielhaften Ausgänge möglichst nahe zusammenrücken und der Lernfortschritt vergrößert wird. Dazu benötigt das Modul Informationen über Ein- und Ausgänge sowie über den momentanen Lernfortschritt, welche ausgewertet werden, um die Gewichtungen zu verändern. Die Variation der Lernverfahren ergibt sich dabei aus der unterschiedlichen Gewichtung des Auswerte- und des Lehrmoduls. Die gröbste Unterscheidung findet dabei in der Existenz statt: Zum einen wird im "Unsupervised Learning" der Lernvorgang überhaupt nicht überwacht, wohingegen im "Supervised Learning" die Lernmodule durch exakte mathematische Algorithmen repräsentiert sind. Diese beiden Typen von Lernverfahren werden im Folgenden näher erläutert.

### 2.3.1 Unsupervised Learning

Beim Unsupervised Learning kommt das Auswertemodul nicht zum Einsatz und das Lehrmodul existiert nur noch in einer sehr einfachen Form, in der die Lernregeln nicht mehr dem aktuellen Fortschritt angepasst werden. Durch diese Simplizität können die Gewichtungen des Netzes oft in einem einzigen Rechenschritt berechnet werden. Der Lernfortschritt wird dabei nicht überwacht, es gibt nur eine festgelegte Regel zur Berechnung der Gewichtungen. Allerdings kann es in speziellen Netzwerktypen auch vorkommen, dass kompliziertere Versionen des Unsupervised Learning zum Einsatz kommen, die von den erwähnten Merkmalen abweichen und sogar rekursiv implementiert sein können. So lässt sich manchmal auch nicht eindeutig zwischen supervised oder unsupervised unterscheiden, da Lernalgorithmen angewendet werden, die charakteristische Merkmale beider Lernarten aufweisen. Der entscheidende Unterschied ist dabei die Überprüfung der Annäherung an vorgegebene Zielwerte: Beim Unsupervised Learning wird diese während des Lernens nicht überprüft.

### 2.3.2 Hebb'sche Lernregel

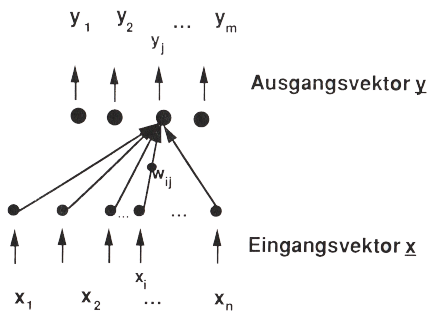


Abbildung 2.7: Einschichtiges Netzwerk

Eines der einfachsten Verfahren des Unsupervised Learning ist die Hebb'sche Lernregel. Der Algorithmus ist relativ simpel und kann in einem einzigen Rechenschritt durchgeführt werden [2]. Zur Herleitung der Formeln kann man sich ein in Abbildung 2.7 dargestelltes einschichtiges Netzwerk vorstellen. Das Eingangsmuster ist hierbei als Vektor $\underline{x}$ definiert und wird der Eingangsschicht zugeführt, deren Komponenten mit allen anderen Komponenten der Ausgangsschicht verbunden sind. Der Ausgangsvektor berechnet sich nach Gleichung 10 mit der Formel $\underline{y}^T = \underline{x}^T \times W$. Um die Gewichtungen zu bestimmen, liegt eine

Anzahl von Trainingspaarmustern $(\underline{x}_k, \underline{y}_k), k = 1, ..., K$ für die Ein- und Augangsmuster vor. Das Ziel des Lernvorgangs ist, für ein Beispielmuster $\underline{x}_k$ ein Ausgangsmuster zu erzeugen, das möglichst identisch mit $\underline{y}_k$ ist. Zunächst betrachtet man wie in Bild 2.7 gezeigt, nur ein Gewicht $w_{ij}$ aus $W$ sowie ein Trainingspaar $(\underline{x}_i, \underline{y}_j)$. Bei der Hebb'schen Lernregel kommt nun das Prinzip zur Geltung, dass sich eine Gewichtung zwischen zwei Neuronen umso stärker ausprägt, je größer die Aktivierung dieser Neuronen ist. Enthält die Komponente $i$ der Eingangsschicht beispielsweise große Werte und soll Komponente $j$ der Ausgangsschicht auch große Werte erzeugen, so soll bei jeder Präsentation dieses Musterpaars die Verbindung proportional zur Größe der Komponenten verstärkt werden. Mathematisch gilt somit für die Veränderung der Gewichtung $w_{ij}$ nach Präsentation des Musterbeispiels (x, y):

$$\Delta w_{ij} = x_i \times y_j \tag{14}$$

Übertragt man diese Regel auf die gesamte Gewichtungsmatrix $W$, so gilt:

$$\begin{pmatrix} \Delta w_{11} & ... & \Delta w_{1M} \\ ... & ... & ... \\ \Delta w_{N1} & ... & \Delta w_{NM} \end{pmatrix} = \begin{pmatrix} x_1 \\ ... \\ x_N \end{pmatrix} * \begin{pmatrix} y_1 & ... & y_M \end{pmatrix} \tag{15}$$

bzw. kurz:

$$\Delta W = \underline{x} \times \underline{y}^T \tag{16}$$

Diese Gleichungen gelten für die Präsentation des Beispielpaares $(\underline{x}, \underline{y})$. Für K Trainingsbeispiele $(\underline{x}_k, \underline{y}_k), k = 1, ..., K$ kommt man zu der Überlegung, wie sich die Gewichtungsmatrix durch die Präsentation aller $K$ Beispiele ändert. Die für ein $k$-tes Beispiel gültige Gleichung 16 lässt sich in eine Rekursionsgleichung umschreiben, die die Veränderung von $W$ zum diskreten Lernzeitpunkt $k$ im Vergleich zum vorangehenden Lernschritt $k-1$ beschreibt. Dadurch erhält man bei Präsentation des $k$-ten Beispielpaares $(\underline{x}_k, \underline{y}_k)$:

$$W(k) = W(k-1) + \underline{x}_k \times \underline{y}_k^T \; mit k = 1, ..., K \tag{17}$$

Setzt man alle Gewichtungsfaktoren vor dem Start des Lernvorgangs auf den Wert 0, so ist $W(0) = 0$. Somit nimmt die Matrix nach Präsentation des ersten Lernbeispiels den Wert $W = \underline{x}_1 \times \underline{y}_1^T$ an und man kann die obige Gleichung als Summenformel schreiben:

$$W = W(k) = \sum_{k=1}^{K} \underline{x}_k \times \underline{y}_k^T \tag{18}$$

Nun lässt sich erkennen, was bereits zum Anfang des Abschnitts erwähnt wurde: Die Gewichtungen des neuronalen Netzes lassen sich in einem einzigen Rechenschritt ermitteln. Es werden zwar Zielwerte für die Ausgangsneuronen vorgegeben, jedoch fällt durch die Berechnung in einem Schritt die Überwachung weg. Hierbei wird einfach angenommen, dass nach Beendigung des Lernvorgangs das Netzwerk den richtigen Ausgang erzeugen wird. Jedoch ist hier anzumerken, dass es nicht sicher ist, dass das Netz nach dem Lernvorgang auch wirklich den richtigen Ausgang erzeugen wird. Deswegen gibt es auch noch eine Vielzahl anderer Ansätze zum Lernen von Gewichtungen.

### 2.3.3 Supervised Learning

Beim Supervised Learning sind das Lehrmodul sowie das Auswertemodul weitaus ausgeprägter als beim unüberwachten Lernen. So ist das Auswertemodul im Wesentlichen ein Modul zur Berechnung des quadratischen Fehlers, welcher sich aus dem Quadrat der Differenz des Sollwerts $y_{soll}$ und dem aktuell berechneten Wert $y_{ist}$ berechnet. Dieser Fehler dient als Kriterium zur Bewertung des Lernfortschritts - hat das Netzwerk seine Gewichtungen aus den Beispielmustern optimal gelernt, so ist er gleich Null. Der Lernfortschritt wird damit in jeder Iteration ermittelt und an das Lehrmodul weitergegeben. Dieses leitet daraus wiederum eine aktuelle Lernregel ab, um die Gewichtungen des Netzes entsprechend dem Lernfortschritt zu verändern. Eine wichtige Eigenschaft des Lehrmoduls ist die variable und an den aktuellen Lernfortschritt angepasste Veränderung der Lernregel. So wird die nächste Veränderung des Netzes bei einem großen Fehler größer sein als bei einem kleinen Fehler, bei welchem der Lernvorgang fast beendet ist. Die "Intelligenz" des neuronalen Netzwerks wird beim Supervised Learning also dadurch erreicht, dass der Lernfortschritt iterativ überwacht wird und aus Beispielen eine bestimmte Systematik erlernt wird. Durch die Minimierung des Fehlerkriteriums werden die Gewichtungen so gesetzt, dass auch ein neues Beispiel verarbeitet werden kann - vorausgesetzt, die richtige Lernregel wird gewählt.

### 2.3.4 Delta-Regel

Die Delta-Regel ist die am weitesten verbreitete Lernregel. Für ihre Herleitung betrachtet man das einfachste, aus nur einem Verarbeitungselement bestehendes, neuronale Netz.

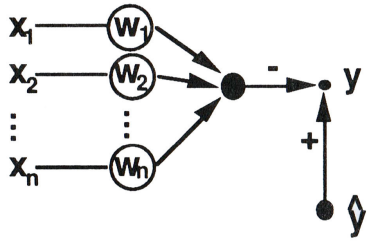


Abbildung 2.8: Neuronales Netzwerk mit einem einzigen Verarbeitungselement

Der Ausgang $y$ des Netzes ergibt sich aus der Formel

$$y = \sum_{i=1}^{n} w_i \times x_i \tag{19}$$

Anfangs wird das Netzwerk wiederum mit zufälligen Werten für die Gewichtungsfaktoren $w_1 - w_n$ besetzt. Daraus wird nach Glg. 19 ein Ausgangswert $y$ berechnet, der natürlich nicht identisch mit dem gewünschten Ausgangssollwert ist. Dieser Sollwert ist in Abb. 2.8 als $\hat{y}$ bezeichnet und wird mit dem Ausgangswert vergleichen. Aus der Differenz $(\hat{y} - \underline{y})$ ergibt sich der Fehler, der für die Erzeugung einer passenden Lernregel verwendet wird. Der Istwert $y$ wird durch Manipulation der Gewichtungen beeinflusst, und somit auch der Fehler, sodass man diesen verkleinern kann. Der quadratische Fehler wird dann durch folgende Formel ausgedrückt:

$$\varepsilon = \tfrac{1}{2}(\hat{y} - y)^2 = \tfrac{1}{2}(\hat{y} - \sum_{i=1}^{n} w_i \times x_i)^2 \tag{20}$$

Bei Betrachtung nur eines Gewichtungsfaktors $w_i$ lässt sich erkennen, dass der Fehler $\varepsilon$ eine quadratische Funktion $\varepsilon(w_i)$ und damit eine Parabel (vgl. Abb. 2.9) ist. Die Parabel nimmt für einen bestimmten Wert von $w_i$ natürlich ein Minimum an, welches den optimalen Wert für die Wahl von $w_i$ darstellt, da der Fehler hier so gering wie möglich ist. Bei einer iterativen Vorgehensweise bestimmt man bei der Iteration $k$ den Wert $w_i(k+1)$ aus dem momentanen Wert $w_i(k)$ für die Gewichtung. Um zum Minimum zu gelangen, ermittelt man die Tangente im Schritt $k$. Ist diese fallend, so befindet man sich auf der linken Hälfte der Parabel und man muss einen positiven Wert zum momentanen Wert der Gewichtung addieren. Für eine steigende Tangente gilt dementsprechend, dass man einen negativen Wert zu $w_i(k)$ addieren muss. Als iterative Regel zur Veränderung der Gewichtungen unter Auswertung des Fehlers gilt damit:

$$w_i(k) = w_i(k-1) - \beta \times \tfrac{\delta\varepsilon}{\delta w_i} \tag{21}$$

Die partielle Ableitung $\frac{\delta\varepsilon}{\delta w_i}$ ergibt sich aus Gleichung 20 zu

$$\tfrac{\delta\varepsilon}{\delta w_i} = -(\hat{y} - \underbrace{\sum_{i=1}^{n} w_i \times x_i}_{=y}) \times x_i \tag{22}$$

Insgesamt ergibt sich somit folgende iterative Formel für die Delta-Regel zur Berechnung der Gewichtung:

$$w_i(k) = w_i(k-1) + \beta \times [\hat{y} - y(k-1)] \times x_i \tag{23}$$

Der Faktor $\beta$ dient dabei zur Feineinstellung des Netzes und wird für gewöhnlich im Intervall $(0; 1]$ gewählt.

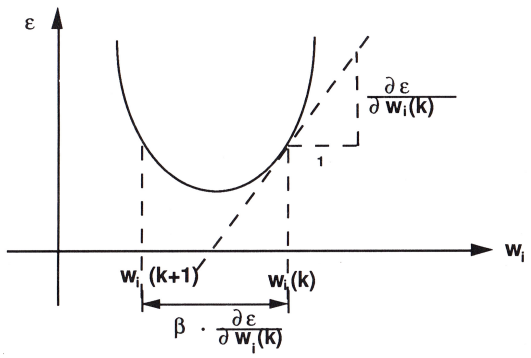


Abbildung 2.9: Quadratischer Fehler $\varepsilon$ als Funktion des Gewichungsfaktors $w_i$ [3]

Bei der üblichen Verwendung von mehreren Lernbeispielen sollte die Delta-Regel entsprechend umformuliert werden. Der quadratische Fehler muss dabei alle Lernbeispiele berücksichtigen: Für $M$ Beispielsmusterpaare $(\underline{x}_m, \hat{y}_m), m = 1, ..., M$ etwa lässt sich der Fehler als Summe der Fehler aller Lernbeispiele ausdrücken:

$$\varepsilon = \tfrac{1}{2}\sum_{m=1}^{M}(\hat{y}_m - y_m)^2 = \tfrac{1}{2}\sum_{m=1}^{M}(\hat{y}_m - \sum_{i=1}^{n} w_i \times x_{i_m})^2 \tag{24}$$

Mit der Ableitung $\frac{\delta\varepsilon}{\delta w_i} = -\sum_{m=1}^{M}(\hat{y}_m - y_m) \times x_{i_m}$ für den Gewichtungsfaktor $w_i$ lautet die Delta-Regel:

$$w_i(k) = w_i(k-1) + \beta \times \sum_{m=1}^{M}[\hat{y}_m - y_m(k-1)] \times x_{i_m} \tag{25}$$

Hier sei nochmal darauf hingewiesen, dass der Unterschied zur einfachen Delta-Regel (23) lediglich in der Aufsummierung der Einflüsse liegt - ss findet somit eine Mittelung der Faktoren statt.

# 3 Anwendungsmöglichkeiten

Neuronale Netze finden in einer Vielzahl von Anwendungen Verwendung. Im Folgenden kann leider nicht auf alle Teilgebiete näher eingegangen werden, da dies den Rahmen der Arbeit sprengen würde. Trotzdem soll ein Überblick sowie Schlüsselbegriffe vermittelt werden.

Eine der häufigsten Bereiche ist die Verarbeitung von fehlerhaften oder unvollständigen Informationen, wie sie etwa bei der Sprach- und Bildverarbeitung als auch der Signalverarbeitung auftritt. Dabei sei zu erwähnen, dass bereits Technologien existieren, die auf neuronalen Netzen aufbauen: Ein Beispiel hierfür ist die Optical Character Recognition "Tesseract" von Google - eine Schrifterkennungssoftware, welche die Fähigkeit zum Training und zum Lernen besitzt. Weiterhin eignen sich neuronale Netze zur Prädikation, der Vorhersage. Dies findet beispielsweise bei Aktienprognosen bzw. Expertensystemen in der Wirtschaft Verwendung, aber auch bei der Vorhersage von Wetterdaten. Auch in der Produktionstechnik lassen sich neuronale Netze zur Überwachung und Diagnose von Maschinen und Anlagen einsetzen: Dabei muss etwa anhand von Meßsignalen entschieden werden, ob der Zustand "normal" ist. Zu guter Letzt sei hier die Anwendung in der Robotik erwähnt. Beispiele für diesen Bereich sind die Erlernung von Bewegungsabläufen oder eine Arm-Kamera-Koordinierung zur Berechnung von Trajektorien zum Ergreifen von durch die Kamera erfasste Objekte. Viele Verwendungen Anwendungsgebiet Robotik teilen sich in kleinere Anwendungsgebiete wie Mustererkennung bzw. Bildverarbeitung auf.

Es lässt sich erwarten, dass neuronale Netze in Zukunft vor allem bei künstlicher Intelligenz eine tragende Rolle spielen und weiterhin Verwendung finden. Somit kann man in den nächsten Jahren mit weiteren interessanten Entwicklungen zu rechnen.

# Literatur


[1] Noriyasu Homma Madan M. Gupta, Liang Jin. *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*. John Wiley & Sons Inc., 2003.

[2] Dan Patterson. *Künstliche neuronale Netze: Das Lehrbuch*. Prentice Hall, 2nd edition, 1996.

[3] Gerhard Rigoll. *Neuronale Netze: Eine Einführung für Ingenieure, Informatiker und Naturwissenschaftlicher*. expert verlag, 1994.

[4] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag Berlin, 1996.

[5] Sibylle Schwarz. Künstliche neuronale Netze, 2012.