2. Assoziativspeicher

Im Gegensatz zu einem herkömmlichen Speicher, bei dem Daten $x \in \{0,1\}^n$ ur Speicheradressen $a \in \mathbb{N}$ abgelegt und ausgelesen werden, hat ein Assoziativspeicher die Eigenschaft, daß der Inhalt selbst dazu dient, die gespeicherten Daten zu identifizieren, die man auslesen möchte. Man spricht da auch von einem *Inhaltsadressierbaren Speicher* (*CAM, content adressable mem* In einem solchen Speicher werden Tupel der Form (x,y) abgelegt, wobei *x* oder Schlüssel dient, mit dem das fehlende Tupelelement *y* oder *x* ausgelesen werde kann. Im Gegensatz zu einem Adressspeicher dürfen sowohl *x* als auch *y* dabei beliebige Daten sein (etwa Bilder, Texte, oder ähnliches...).

In diesem Aufgabenblatt werden wir uns formale Neuronen ansehen, die wir zu einfachen Assoziativspeicher zusammensetzen. Dieser Speicher wird die Eigens haben, daß er recht tolerant gegenüber von Fehlern ist und sich Daten auch nac einer Beschädigung wieder korrekt auslesen lassen.

Die Art der Speicherung, die wir uns gleich ansehen werden, basiert auf einer Entdeckung des kanadischen Psychologen Donald Olding Hebb aus dem Jahre 1949. Er verfaßte eine berühmte Arbeit über das Lernen in biologischen Neuron Netzen (*The Organization of Behavior: A Neuropsychological Theory*). Seine entscheidende Beobachtung war, daß sich die synaptischen Verbindungen zwis zwei Neuronen verstärken, wenn diese über längere Zeit oder wiederholt gleichz zusammen aktiv sind.

2.1 Der Hebb'sche Assoziativspeicher

Betrachten wir vorerst den Gewichtsvektor $\underline{w} = [w_1, ..., w_n]^T \in \mathbb{R}^n$ eines einzelnen Neurons (ohne Bias). Sei $\underline{x} \in \{+1, -1\}^n$ und die Ausgabefunktion unseres Neurons

$$S(z) = \begin{cases} +1, & z \geq 0 \\ -1, & z < 0 \end{cases}.$$

Dann berechnet sich die Neuronenausgabe zu $y = S(\underline{w}^T \underline{x})$ mit $y \in \{+1, -1\}$. Die von Hebb gefundene Lernregel besagt nun, daß sich das synaptische Gewicht w_i (für ein $i \in \{1, ..., n\}$) verstärkt, wenn die beiden Neuronen, die über w_i miteinander verbunden sind, gleichzeitig aktiv sind. Wir legen fest, daß dies bei $x_i = y$ der Fall ist: der Eingangsreiz über diese Verbindung stimmt mit der Ausgabe unseres Neurons überein. Ist dies nicht der Fall, wollen wir stattdessen eine Verminderung des Verbindungsgewichts erreichen.

Eine Möglichkeit, dies in eine konkrete Formel umzusetzen, ist folgende Lernregel für das Gewicht *des i-ten* Eingangs

$$\forall i : w_i \leftarrow w_i + \gamma \cdot x_i \cdot y \qquad \text{Hebb'sche Lernregel}$$

oder in vektorieller Darstellung

$$\underline{w} \leftarrow \underline{w} + \gamma \cdot \underline{x} \cdot y$$

Der Parameter γ ($\gamma > 0$) wird dabei als *Lernrate* bezeichnet und gewichtet den Einfluß von $\underline{x} \cdot y$ auf $\underline{w}$. Man beachte, daß bei $x_i = y$ wenn $x_i y = 1$ gilt es somit zu einer Erhöhung von w_i kommt, anderenfalls gilt $x_i y = -1$ und es kommt zu einer Verminderung, genau wie wir gefordert hatten.
Diese Lernregel wird dabei gemeinhin als *Hebb'sche Lernregel* bezeichnet, auch wenn mehrere mathematische Interpretationen für die Beobachtungen Hebbs möglich

sind.

Schauen wir uns nun eine mögliche Architektur eines Assoziativspeichers auf Basis mehrerer formaler Neuronen an, der auf Grundlage der Hebb'schen Lernregel arbeitet. Siehe dazu Abbildung 2.1.

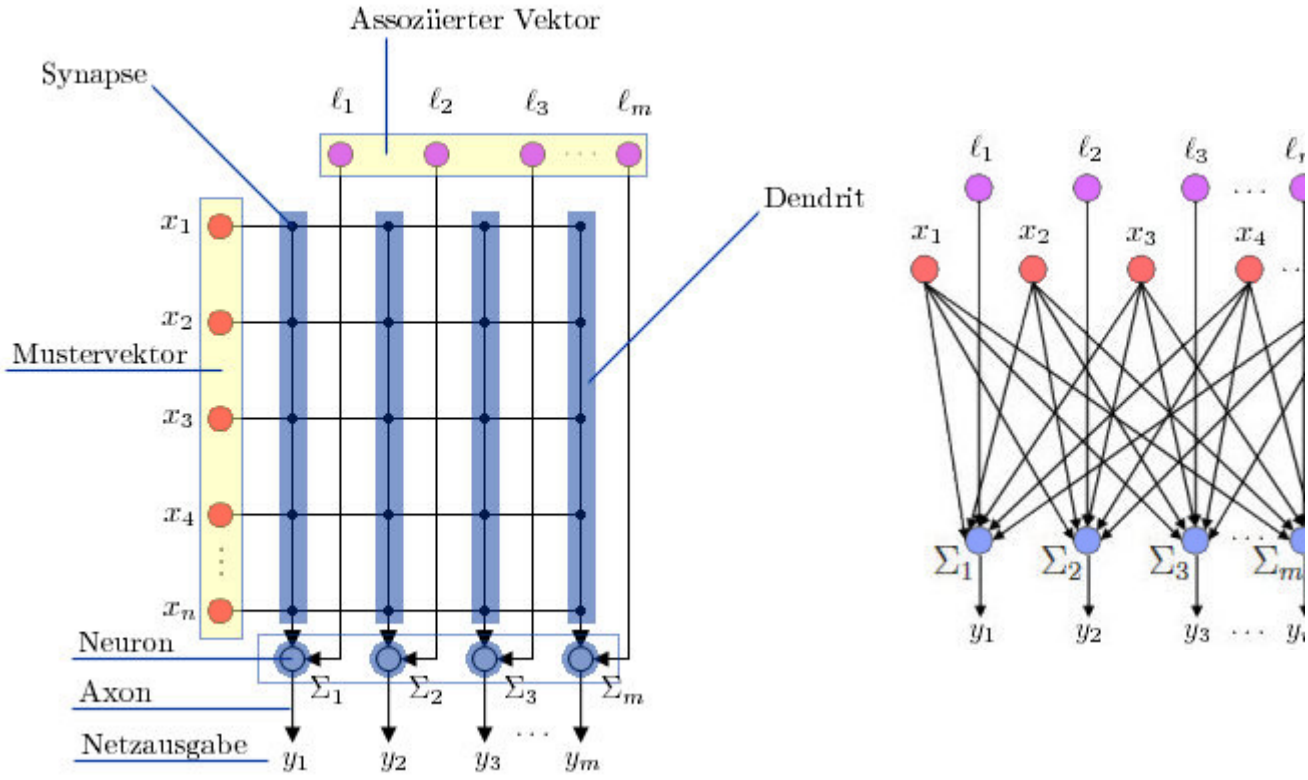

Abbildung 2.1 *Links:* Neuronenmodell eines Assoziativspeichers. *Rechts:* Äquivalente Darstellung als Netz.

Die gelb unterlegten Kreise stellen die Eingabe des Netzes dar, wobei $x_1, ..., x_n \in \{+1, -1\}$ und L die Lehrervorgabe $\ell_1, ..., \ell_m \in \{+1, 0, -1\}$ ist. Die blau unterlegten und mit $\Sigma_1, ..., \Sigma_m$ beschrifteten Kreise sind formale Neuronen, welche die Eingaben verarbeiten. Gitternetzpunkte im selben blauen Bereich symbolisieren synaptische Verbindungsstellen, die jeweils mit einem Gewicht versehen sind und an denen die Eingaben eines bestimmten Neurons abgegriffen werden. Wie ersichtlich, erhält das i-te Neuron, Σ_i, als Eingabe genau den Vektor $\underline{x} := [x_1, ..., x_n]^T$ und ein

Sei nun $\underline{w}^{(i)} := [w_{1,i}, ..., w_{n,i}]^T$ der zu den dargestellten synaptischen

Verbindungsstellen des i-ten Neurons gehörige Gewichtsvektor und ein zu ℓ_i gehöriges festes Verbindungsgewicht, das wir im Folgenden als "nicht modifizierbar" ansehen (daher rührt die gesonderte Darstellung in Abbildung 2.1).
Dann berechnet sich die **Aktivierung** z_i des i-ten Neurons zu

$$\begin{aligned} z_i &= w_{n+1,i} \cdot \ell_i + \sum_{j=1}^{n} w_{j,i} \cdot x_j \\ &= \underline{w}^{(i)^T} x + w_{n+1,i} \cdot \ell_i \end{aligned}$$

und die **Ausgabe** y_i zu

$$y_i = S(z_i) = S\left(\underline{w}^{(i)^T} x + w_{n+1,i} \cdot \ell_i\right) = S(\ell_i) = \ell_i$$

Dies gilt, wenn die Gewichte der Lehrervorgabe ausreichend groß sind (etwa), so dass das Vorzeichen von z_i lediglich durch das von ℓ_i bestimmt wird bei $\ell_i \neq 0$.

Wie werden die Gewichte verändert, um Muster abzuspeichern? Wenden wir hier die Hebbsche Lernregel auf das *i*-te Neuron an, so erhalten wir die Lernregel

$$\begin{aligned} &\underline{\ell_i \neq 0} \\ &\qquad \underline{w}^{(i)} \leftarrow \underline{w}^{(i)} + \underline{x} \cdot S(z_i) \\ \Longleftrightarrow \quad &\qquad \underline{w}^{(i)} \leftarrow \underline{w}^{(i)} + \underline{x} \cdot \ell_i \end{aligned}$$

Fassen wir dies für alle Gewichtsvektoren zusammen,

$$[\,\underline{w}^{(1)}, \ldots, \underline{w}^{(m)}] \leftarrow [\,\underline{w}^{(1)}, \ldots, \underline{w}^{(m)}] + [\,\underline{x} \cdot \ell_1, \ldots, \underline{x} \cdot \ell_m]$$

so läßt sich das Produkt ($\underline{x} \cdot \ell_i$) als äußeres Vektorprodukt formulieren. Auch die Gewichtsvektoren selbst können als Spaltenvektoren in einer gemeinsamen Matrix zusammengefasst werden.

Damit haben wir zwei grundlegende Operationen definiert, nämlich **Einspeichern** eines Musterpaares $(\underline{x}^{(k)}, \underline{\ell}^{(k)})$ mittels der Hebb-Regel

$$\boxed{W \leftarrow W + \underline{x}^{(k)} \cdot \underline{\ell}^{(k)^T}}$$

und **Auslesen** eines zu $\underline{x}^{(k)}$ assoziierten Musters durch Berechnen der Ausgabe des Netzes

$$\boxed{\underline{y} \;=\; S(W^T \underline{x}^{(k)})}$$

Schauen wir uns nun an was passiert, wenn wir nun die Hebbsche Lernregel sukzessiv auf *s* Musterpaare $(\underline{x}^{(1)}, \underline{\ell}^{(1)}), ..., (\underline{x}^{(s)}, \underline{\ell}^{(s)})$ mit $\underline{x}^{(k)} \in \{+1, -1\}^n$ und $\underline{\ell}^{(k)} \in \{+1, -1\}^m$, $1 \leq k \leq s$ anwenden.

Starten wir mit auf Null gesetzten Gewichten und fassen wieder alle Gewichtsvektoren in einer Matrix $W = [\underline{w}^{(1)}, ..., \underline{w}^{(m)}]$ zusammen, dann wird nach Anwenden der Hebb-Regel für ***W*** nach Abspeichern von *s* Mustern die Gewichtsmatrix des Assoziativspeichers zu

$$W = \sum_{i=1}^{s} \underline{x}^{(i)} \cdot \underline{\ell}^{(i)T}$$

Präsentieren wir dem Netz nun den Vektor $\underline{x}^{(k)}$ und keine Lehrervorgabe, also anstelle von $\underline{\ell}^{(k)}$ den Nullvektor als Eingabe, so berechnet sich die Netzausgabe

$$\underline{y} \;=\; S(W^T \cdot \underline{x}^{(k)})$$

Der Ausdruck soll dabei andeuten, daß die Funktion $S(\cdot)$ komponentenweise auf den Vektor $W^T \cdot \underline{x}^{(k)}$ angewendet wird. Dabei gilt:

$$\begin{aligned}
W^T \cdot \underline{x}^{(k)} &= \left(\textstyle\sum_{i=1}^{s} \underline{x}^{(i)} \cdot \underline{\ell}^{(i)T}\right)^T \cdot \underline{x}^{(k)} \\
&= \left(\textstyle\sum_{i=1}^{s} \underline{\ell}^{(i)} \cdot \underline{x}^{(i)T}\right) \cdot \underline{x}^{(k)} \\
&= \underline{\ell}^{(k)} \cdot \underline{x}^{(k)T} \cdot \underline{x}^{(k)} + \textstyle\sum_{i=1, i\neq k}^{s} \underline{\ell}^{(i)} \cdot \underline{x}^{(i)T} \underline{x}^{(k)} \\
&= \underline{\ell}^{(k)} \cdot n + \underbrace{\textstyle\sum_{i=1, i\neq k}^{s} \underline{\ell}^{(i)} \cdot \underline{x}^{(i)T} \underline{x}^{(k)}}_{Störterm\ (Übersprechen)}
\end{aligned}$$

Die letzte Zeile folgt aus $\underline{x}^{(k)} \in \{+1, -1\}^n$. Ist der Störterm klein genug, so daß $\underline{\ell}^{(k)} \cdot n$ das Vorzeichen des Ausdrucks $\underline{\ell}^{(k)} \cdot n + \sum_{i=1, i\neq k}^{s} \underline{\ell}^{(i)} \cdot \underline{x}^{(i)T} \underline{x}^{(k)}$ bestimmt,

dann stellen wir fest, daß in der Tat $\underline{y} = S(W^T \cdot \underline{x}^{(k)}) = \underline{\ell}^{(k)}$ gilt, wir also bei Eingabe des *k*-ten Musters die gewünschte Ausgabe erhalten! Die Muster $\underline{x}^{(k)}$ und $\underline{\ell}^{(k)}$ sind also zueinander assoziiert und wir können $\underline{\ell}^{(k)}$ fehlerfrei durch Angabe von $\underline{x}^{(k)}$ auslesen.

Es ist allerdings nicht selbstverständlich, daß beim Auslesen das Übersprechen klein genug ist, so daß $\underline{y} = \underline{\ell}^{(k)}$ gilt. Wie bekommen wir nun aber einen Störterm, der klein genug ist? Die naheliegenste Möglichkeit besteht darin, die Mustervektoren $\underline{x}^{(1)}, ..., \underline{x}^{(s)} \in \{+1, -1\}^n$ paarweise orthogonal zu wählen, also so, daß $\left(\underline{x}^{(i)T} \underline{x}^{(j)} = 0\right) \Leftrightarrow i \neq j$ gilt.

Wir erhalten dann beim Bilden der Neuronenausgabe

$$\begin{aligned} W^T \cdot \underline{x}^{(k)} &= \underline{\ell}^{(k)} \cdot n + \sum_{i=1, i \neq k}^{s} \underline{\ell}^{(i)} \cdot \underbrace{\underline{x}^{(i)T} \underline{x}^{(k)}}_{=0} \\ &= \underline{\ell}^{(k)} \cdot n \\ \Longrightarrow \quad S(W^T \cdot \underline{x}^{(k)}) &= \underline{\ell}^{(k)} \end{aligned}$$

Der Störterm wird bei orthogonalen Mustern also immer Null, und alle gespeicherten Musterpaare können fehlerfrei wieder ausgelesen werden.

nach oben

Ein Assoziativspeicher für beliebige Muster

Die Einschränkung, daß wir nur orthogonale Muster in unserem Netz verwenden können, ist recht unbefriedigend. Allerdings läßt sich diese Bedingung mit ein wenig Randomisierung leicht umgehen. Entscheidend dafür ist die Beobachtung, daß zufällig gewählte Vektoren aus $\{+1, -1\}^n$ die Tendenz dazu haben, orthogonal zu sein. Das Interessante dabei ist, daß diese Vektoren dabei umso eher dazu neigen senkrecht aufeinander zu stehen, je größer ihre Dimension *n* ist.

Man kann diese Tatsache ausnutzen, um *beliebige* Musterpaare $(\underline{x}, \underline{\ell}), \underline{x} \in \{+1, -1\}^n, \underline{\ell} \in \{+1, -1\}^m$ in den Hebbschen Assoziativspeicher einzuspeisen, indem man $\underline{x}$ einfach durch eine Pseudozufallsfunktion schickt, in einen pseudozufälligen Vektor $\underline{\hat{x}}$ umwandelt und $(\underline{\hat{x}}, \underline{\ell})$ statt $(\underline{x}, \underline{\ell})$ in dem Netz speichert. Beim Einspeichern mehrerer Paare werden die resultierenden $\underline{\hat{x}}$'e dann dazu neigen, orthogonal zueinander zu sein und den Störterm klein zu halten. Man kann zeigen, daß sich auf diese Art und Weise $O(n/\log n)$ beliebige Paare speichern lassen, so daß die Wahrscheinlichkeit eines fehlerhaft ausgelesenen Musters mit $n \to \infty$ gegen 0 geht, sofern $\underline{\ell}$ nicht mehr als *n* Komponenten besitzt).

nach oben

2.2 Adaline

In diesem Aufgabenblatt werden wir neben dem Hebb-Assoziativspeicher auch noch ein weiteres Netz, das Adaline-Modell, betrachten, daß wir in späteren Aufgaben noch öfter verwenden werden. Daher hier eine kleine Einführung über seine Funktionsweise und dessen Eigenschaften.

Das Adaline-Modell (**Ada**ptive **Lin**ear **E**lement) wurde 1960 von Bernard Widrow und Marcian Edward Hoff vorgeschlagen. Man kann es sich als eine kleine Erweiterung des Perzeptron-Modells vorstellen und es verfügt über eine Reihe interessanter Eigenschaften, die es recht nützlich macht. Der Aufbau dieses Netzes ist in Abbildung 2.2 dargestellt und entspricht einem einfachen Netzwerk aus nur einer Schicht.
Der entsprechende Lernalgorithmus ist darüberhinaus sehr simpel und wird uns in leicht modifizierter Form auch noch bei anderen Netzen begegnen. Die Unterschiede zum Perzeptron sind dabei recht einfach. Wie wir zuvor gesehen haben, lautet die Perzeptron-Lernregel für ein Neuron mit Gewichtsvektor $\underline{w} \in \mathbb{R}^n$ (inklusive Bias), einer Eingabe $\underline{x} \in \mathbb{R}^n$, der Lehrervorgabe $\ell(\underline{x}) \in \{0, 1\}$ und der Heaviside-

Ausgabefunktion $S(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$

$$\underline{w} \quad \leftarrow \quad \underline{w} - \gamma \cdot (S(\underline{w}^T \underline{x}) - \ell(\underline{x})) \cdot \underline{x}$$

Die Lernregel eines Adaline-Netzes sieht dieser sehr ähnlich, jedoch wird nicht die Ausgabe $S(\underline{w}^T \underline{x})$ des Netzes, sondern stattdessen die Aktivierung $\underline{w}^T \underline{x}$ zum Lernen verwendet. Außerdem ist man im Adaline-Modell nicht auf die Heaviside-Funktion des Perzeptrons als Ausgabefunktion angewiesen und kann $S(z)$ entsprechend an die Problemstellung anpassen. Die Änderung der Lernregel ist dabei mathematisch motiviert und da es ein wichtiger Punkt zum Verständnis der Funktionsweise (auch anderer Netze) darstellt, wollen wir uns jetzt einmal etwas näher damit beschäftigen.

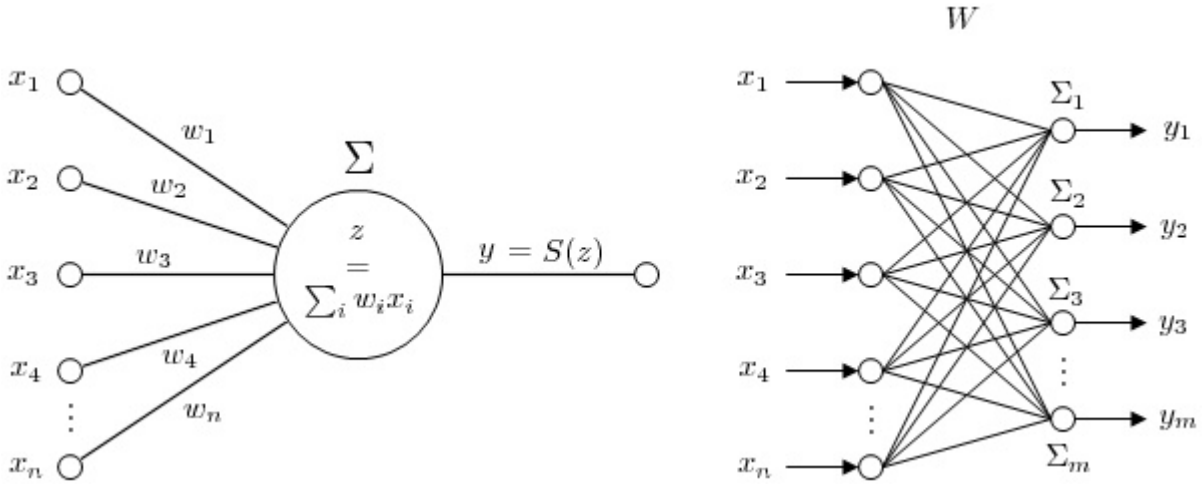

Abbildung 2.2 Aufbau eines Adaline-Netzes.

Links: Neuronenmodell. *Rechts:* Netzstruktur.

Mit dem Trainieren von Neuronalen Netze wollen wir erreichen, daß das Netz eine bestimmte Aufgabe erfüllt. Bei vielen Netzen, wie Adaline oder dem Perzeptron, besteht diese Aufgabe darin, möglichst gut die Beziehungen zwischen den Eingabe- und den Ausgabevektoren zu lernen, die wir dem Netz präsentieren. Ein vernünftiger Ansatz zur Erreichung dieses Ziels wäre daher, gezielt eine Lernregel zu konstruieren, die den Fehler der Ausgabe minimiert, der bei Eingabe der einzelnen Trainingsvektoren auftritt. Was wir genau unter "Fehler" verstehen, bleibt dabei uns überlassen.
Ein bewährtes Maß dafür ist der sogenannte *erwartete quadratische Fehler* ("**M**ean

Squared **E**rror", MSE). Betrachten wir ein einzelnes Neuron mit linearer Ausgabefunktion, dann ist der erwartete quadratische Fehler $R(\underline{w})$, den wir machen, formuliert mit den Erwartungswertklammern $\langle\cdot\rangle_x$ für den Erwartungswert über alle

$$R(\underline{w}) \quad = \quad \left\langle \left(\underline{w}^T \underline{x} - \ell(\underline{x})\right)^2 \right\rangle_x \qquad (2.1)$$

Unser Ziel ist es nun, diesen Fehler durch Anpassen der Gewichte $\underline{w}$ zu minimieren, damit die Ausgabe unseres Neurons möglichst gut mit der gewünschten Ausgabe übereinstimmt. Man bezeichnet die zu minimierende Funktion in diesem Zusammenhang auch als *Fehler-* oder *Zielfunktion*.

Ein allgemeines und recht einfaches Prinzip, um dieses Ziel zu erreichen, ist ein Verfahren, das man als *Gradientenabstieg* bezeichnet. Der Gradient einer skalaren Funktion ist die allgemeine Ableitung in alle Raumrichtungen a_i, mit *grad(.)* oder symbolisch mit dem Zeichen ∇ ("Nabla") bezeichnet. Er ist definiert als der Vektor der partiellen Ableitungen:

$$\nabla f = \left[\frac{\partial f}{\partial a_1}, ..., \frac{\partial f}{\partial a_n}\right]$$

oder als Spaltenvektor

$$\nabla f = \left[\frac{\partial f}{\partial a_1}, ..., \frac{\partial f}{\partial a_n}\right]^T$$

Er kann je nach Zusammenhang als transponiert oder nicht transponiert interpretiert werden.

Der Gradient einer Funktion hat die Eigenschaft, daß er (wie dies bei der gewöhnlichen Ableitung einer Funktion $f : \mathbb{R} \to \mathbb{R}$ auch der Fall ist) in die Richtung des größten Anstiegs des Funktionswertes zeigt. D.h. eine Veränderung des Argumentes einer Funktion in Richtung ihres Gradienten führt zu einer bestmöglichen Vergrößerung des Funktionswertes unter allen Richtungen, die wir wählen können. Er kann daher verwendet werden, um lokale Maxima von Funktionen aufzuspüren, indem man schrittweise, startend bei einem Vektor $\underline{a} = [a_1, ..., a_n]^T$, zu $\underline{a}$ ein Vielfaches γ des Gradienten addiert, dann den neuen Gradienten berechnet und wiederum abändert usw., bis man dem gesuchten Maximum ausreichend nahe

gekommen ist.

Daß man sich in der Nähe eines lokalen Optimums befindet, kann man daran feststellen, daß der Gradient dort den Nullvektor darstellt und sich diesem entsprechend annähert, sofern die Funktion in der Nähe des Optimums keinen starken Oszillationen unterworfen ist.

Diesen Vorgang bezeichnet man als *Gradientenaufstieg*, oder auch als "Hill-Climbing". Ist man an lokalen Minima statt Maxima interessiert, so kann man in jedem Schritt ein Vielfaches γ des Gradienten abziehen, da der negative Gradient in die Richtung des steilsten Absinkens des Zielfunktionswertes zeigt, was man entsprechend als *Gradientenabstieg* bezeichnet. γ bezeichet man dabei auch als die Schrittweite des Gradientenverfahrens.
Ein Haken an der Sache ist allerdings, daß man nie genau weiß, wie groß diese Schrittweite genau gewählt werden muß, da man leicht aus Versehen ein lokales Optimum einer Funktion überspringen kann, anstatt genau darauf oder möglichst nahe davor zu landen, siehe Abbildung 2.3. Dies kann evtl. sogar eine Veränderung des Funktionswertes in entgegengesetzter Richtung zur Folge haben, jedoch gibt es Methoden dafür, dies in den Griff zu bekommen (eine mit $\Theta(1/t)$ absinkende Lernrate zum Beispiel, wobei *t* die aktuelle Iteration bezeichnet, würde dieses Problem lösen).

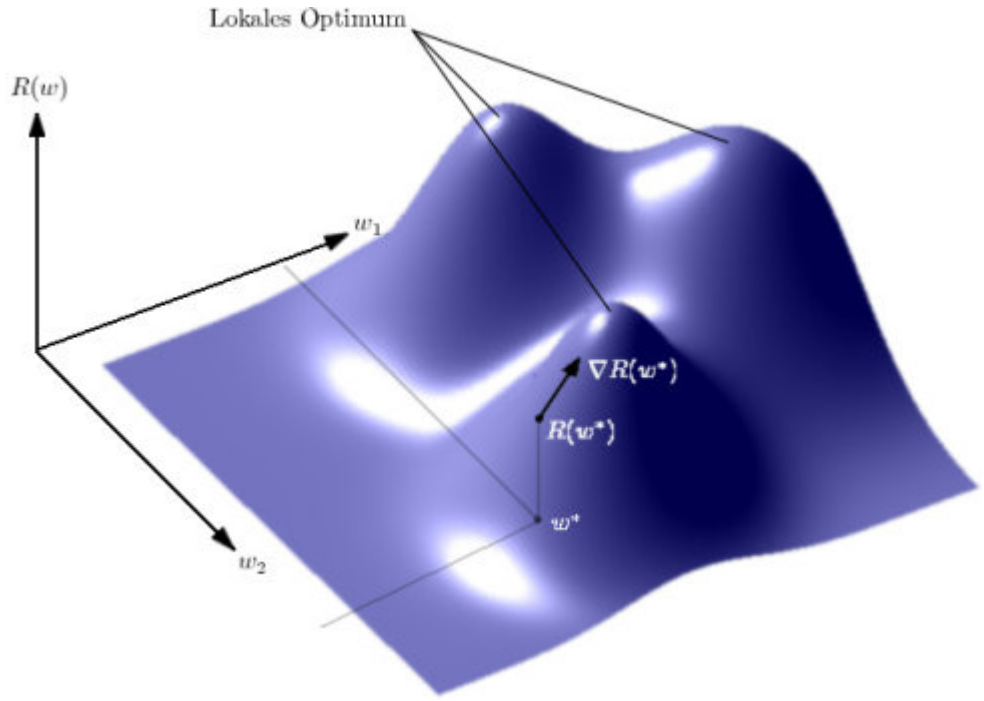

Abbildung 2.3 Darstellung des Gradientenverfahrens.

Dargestellt sind die Funktionswerte einer bestimmten Zielfunktion $R(\underline{w})$ für einen Gewichtsvektor $\underline{w} = [w_1, w_2]^T \in \mathbb{R}^2$ mit 2 Komponenten. Es ergibt sich eine mehrdimensionale Fehlerlandschaft, die je nach Art der Zielfunktion mehrere lokale Optima aufweisen kann.

Um nun Gleichung 2.1 bezüglich des Gewichtsvektors $\underline{w}$ zu minimieren, können wir einen Gradientenabstieg verwenden. Allerdings werden wir statt Gleichung 2.1 aus Schönheitsgründen den mit 1/2 multiplizierten Ausdruck als Zielfunktion verwenden, der ja an der selben Stelle x* das Optimum hat:

$$R(\underline{w}) = \frac{1}{2} \cdot \left\langle \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right)^2 \right\rangle_x \qquad (2.2)$$

Wir erhalten damit

$$
\begin{aligned}
\nabla R(\underline{w}) &= \nabla \left(\tfrac{1}{2} \cdot \left\langle \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right)^2 \right\rangle_x \right) \\
&= \tfrac{1}{2} \cdot \left\langle \nabla \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right)^2 \right\rangle_x \\
&= \left\langle \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right) \nabla \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right) \right\rangle_x \\
&= \left\langle \underline{x} \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right) \right\rangle_x
\end{aligned}
$$

Mittels Gradientenabstieg erhalten wir dann folgende Lernregel für ein einzelnes Neuron:

$$\boxed{\underline{w} \leftarrow \underline{w} - \gamma \cdot \left\langle \underline{x} \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right) \right\rangle_x}$$

Ein Nachteil dieser Regel ist, daß wir erst einen Erwartungswert bestimmen und dazu sämtliche Muster $\underline{x}$ vorab kennen müssen, damit wir dann den Gradienten berechnen können. Aus diesem Grund verwendet man meist eine sogenannte *stochastische Approximation* dieser Lernregel (die Lernregel wird dann auch als ein sogenanntes *"online Verfahren"* bezeichnet - im Gegensatz dazu wird die vorige Lernregel auch *"offline-"* oder *"batch-Verfahren"* genannt):

$$\boxed{\underline{w} \leftarrow \underline{w} - \gamma \cdot \underline{x} \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right)}$$

Da der Ausdruck $\delta := \underline{w}^T \underline{x} - \ell(\underline{x})$ den Fehler der Neuronenausgabe bzw. die Differenz zwischen der Lehrervorgabe und der Ausgabe darstellt und die Lernregel ohne die zusätzliche Normierung die Gestalt $\underline{w} \leftarrow \underline{w} - \gamma \cdot \delta \cdot \underline{x}$ annimmt, wird sie häufig auch als *Fehlerlernregel* oder auch als ***Delta-Regel*** bezeichnet.

An dieser Stelle läßt sich nun sehr gut die Ähnlichkeit zur Perzeptron-Lernregel erkennen. Wir können diese Lernregel noch ein klein wenig verbessern, wenn wir fordern, daß in einem Schritt der Ausgabefehler $\underline{w}^T \underline{x} - \ell(\underline{x})$ des Netzes möglichst gut kompensiert werden soll. Die Ausgabe $\left(\underline{w} - \gamma \cdot \underline{x} \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right) \right)^T \underline{x}$ des Netzes nach Anwenden eines Gradientenschrittes soll dabei nahe der geforderten $\ell(\underline{x})$ sein. Es

gilt:

$$\begin{aligned} & \left(\underline{w} - \gamma \cdot \underline{x} \left(\underline{w}^T \underline{x} - \ell(\underline{x})\right)\right)^T \underline{x} \\ = \quad & \underline{w}^T \underline{x} - \gamma \cdot \left(\underline{w}^T \underline{x} - \ell(\underline{x})\right) \underline{x}^T \underline{x} \end{aligned}$$

Für $\gamma = \frac{1}{\underline{x}^T \underline{x}}$ erhalten wir gerade $\ell(\underline{x})$ und der Fehler würde vollständig kompensiert werden, jedoch nur für das aktuelle Trainingsmuster $\underline{x}$. Besser wäre ein etwas weniger starker Einfluß eines einzelnen Musters auf die Änderung von $\underline{w}$, so daß das "globale" Verhalten der $\underline{x}$ besser eingefangen werden kann. Daher macht es Sinn, nicht γ durch $\frac{1}{\underline{x}^T \underline{x}}$ zu ersetzen, sondern es auf $\gamma < 1$ einzuschränken und einen zusätzlichen Normierungsfaktor $\frac{1}{\underline{x}^T \underline{x}}$ einzuführen. Dies führt auf die sogenannte ***Widrow-Hoff-Lernregel***:

$$\underline{w} \quad \leftarrow \quad \underline{w} - \gamma \cdot \underline{x} \cdot \frac{1}{\underline{x}^T \underline{x}} \cdot \left(\underline{w}^T \underline{x} - \ell(\underline{x})\right)$$

Erweitert man die Widrow-Hoff-Lernregel auf mehrere Neuronen mit den Gewichtsvektoren $\underline{w}_1, ..., \underline{w}_m \in \mathbb{R}^n$, so erhält man für $W = [\underline{w}_1, ..., \underline{w}_m] \in \mathbb{R}^{n \times m}$ die Lernregel:

$$W \quad \leftarrow \quad W - \gamma \cdot \frac{1}{\underline{x}^T \underline{x}} \cdot \underline{x} \cdot \left(W^T \underline{x} - \ell(\underline{x})\right)^T$$

Dabei ist $\ell(\underline{x}) \in \mathbb{R}^m$ hier nun ein anzutrainierender Ausgabe*vektor* (Lehrervorgabe) zur Eingabe $\underline{x}$.

Bei Eingabe eines Vektors $\underline{x}$ berechnet das Netz den Ausgabevektor $\underline{y} = W^T \underline{x}$ wir stellen insbesondere fest, daß Adaline eine lineare Transformation der Eingabevektoren durchführt, die durch die Gewichtsmatrix W gegeben ist. Andererseits haben wir oben gesehen, daß wir mit der Widrow-Hoff-Lernregel eine Gewichtsmatrix W ganz einfach durch Training mit Beispielen lernen lassen können. In der Tat ist es so, daß wir Adaline auf diese Weise *beliebige* lineare Transformationen beibringen können, sofern wir nur über genügend Trainingsbeispiele verfügen!

nach oben

Konvergenzbetrachtungen

Eine entscheidende Frage dabei ist allerdings, wie "gut" Adaline diese Transformationen lernen kann, bzw. um es präziser zu formulieren, ob die Widrow-Hoff-Lernregel von Adaline für die Gewichtsvektoren der Neuronen tatsächlich ein *globales* Minimum der Zielfunktion 2.2 findet, was eine optimale Lösung darstellen würde. Diese Frage stellt sich, da das Gradientenverfahren lediglich dazu ausgelegt ist, *lokale* Optima zu bestimmen, siehe Abbildung 2.3.

Optimiert man die Zielfunktion durch einen Gradientenaufstieg (bzw. -Abstieg), so folgt man, startend bei einem Gewichtsvektor $\underline{w}^*$, durch Addition (bzw. Subtraktion) eines kleinen Vielfachen des Gradienten zu $\underline{w}^*$, der Richtung des steilsten Anstiegs (bzw. Abstiegs) des Zielfunktionswertes. Im Einzugsbereich eines lokalen Optimums kann dieses jedoch in der Regel nicht mehr verlassen werden, da häufig alle Gradientenschritte der näheren Umgebung auf dieses zulaufen. Dies führt dazu, daß in den meisten Fällen keine optimalen Lösungen gefunden werden. Im Übrigen ist dazu anzumerken, daß das Auffinden eines globalen Optimums einer beliebigen Zielfunktion generell ein äußerst schwieriges Problem ist, für das bislang keine zufriedenstellende Lösung existiert. In der Praxis gibt man sich daher meist mit suboptimalen Lösungen zufrieden, die häufig dennoch sehr gute Ergebnisse liefern können. Dies wird uns später im Praktikum noch in Zusammenhang mit dem Multilayer-Perzeptron und dem Backpropagation-Algorithmus bei etwas komplexeren Optimierungsaufgaben begegnen.

Die Konvergenz des Gradientenverfahrens zu einem globalen Optimum ist also nicht selbstverständlich und hängt sehr stark von der "Gutartigkeit" der Zielfunktion ab. Da dies aber eine wichtige Eigenschaft ist, untersuchen wir nun, ob wir sie vielleicht bei Adaline nachweisen können. Wie schon erwähnt, zeigt der Gradient einer Funktion in die Richtung des steilsten Anstiegs des Funktionswertes und der negative Gradient entsprechend in Richtung des steilsten Absinkens. Das aber bedeutet, daß der Gradient in einem lokalen Optimum den Nullvektor darstellen muß; denn, wäre er nicht Null, so würden wir uns nicht in einem lokalen Optimum befinden, da wir den Funktionswert durch eine kleine Änderung entlang des Gradienten noch "verbessern" könnten. Jedes lokale Optimum $\underline{\hat{w}}$ (Minimum oder Maximum) von Gleichung 2.2 erfüllt deshalb:

$$
\begin{aligned}
& \nabla R(\underline{\hat{w}}) && = && 0 \\
\iff & \left\langle \underline{x} \cdot \left(\underline{\hat{w}}^T \underline{x} - \ell(\underline{x}) \right) \right\rangle && = && 0 \\
\iff & \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle && = && \langle \underline{x}\ell(\underline{x}) \rangle
\end{aligned}
\tag{2.3}
$$

Wir würden nun gerne zeigen, daß unter dieser Bedingung $R(\underline{\hat{w}})$ bereits schon minimal ist. Zum Zwecke des Widerspruchs nehmen wir daher an, daß ein $\underline{w}$ mit einem kleineren Zielfunktionswert existiert, als der Zielfunktionswert für $\underline{\hat{w}}$. Es folgt dann:

$$
\begin{aligned}
& R(\underline{w}) && < && R(\underline{\hat{w}}) \\
\iff & \tfrac{1}{2} \cdot \left\langle \left(\underline{w}^T \underline{x} - \ell(\underline{x}) \right)^2 \right\rangle && < && \tfrac{1}{2} \cdot \left\langle \left(\underline{\hat{w}}^T \underline{x} - \ell(\underline{x}) \right)^2 \right\rangle \\
\iff & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} - 2\underline{w}^T \underline{x}\ell(\underline{x}) + \ell(\underline{x})^2 \right\rangle && < && \left\langle \underline{\hat{w}}^T \underline{x} \underline{\hat{w}}^T \underline{x} - 2\underline{\hat{w}}^T \underline{x}\ell(\underline{x}) + \ell(\underline{x})^2 \right. \\
\iff & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} - 2\underline{w}^T \underline{x}\ell(\underline{x}) \right\rangle && < && \left\langle \underline{\hat{w}}^T \underline{x} \underline{\hat{w}}^T \underline{x} - 2\underline{\hat{w}}^T \underline{x}\ell(\underline{x}) \right\rangle
\end{aligned}
$$

Mt Gleichung 2.3 vereinfacht sich dies zu:

$$
\begin{aligned}
& \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} - 2\underline{w}^T \underline{x}\ell(\underline{x}) \right\rangle && < && \left\langle \underline{\hat{w}}^T \underline{x} \underline{\hat{w}}^T \underline{x} - 2\underline{\hat{w}}^T \underline{x}\ell(\underline{x}) \right\rangle \\
\iff & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} \right\rangle - 2\underline{w}^T \cdot \langle \underline{x}\ell(\underline{x}) \rangle && < && \underline{\hat{w}}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle - 2\underline{\hat{w}}^T \cdot \langle \underline{x}\ell(\underline{x} \\
\overset{(1.8)}{\iff} & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} \right\rangle - 2\underline{w}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle && < && \underline{\hat{w}}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle - 2\underline{\hat{w}}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \right. \\
\iff & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} \right\rangle - 2\underline{w}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle && < && -\underline{\hat{w}}^T \cdot \left\langle \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle \\
\iff & \left\langle \underline{w}^T \underline{x} \underline{w}^T \underline{x} - 2\underline{w}^T \underline{x}\underline{\hat{w}}^T \underline{x} + \underline{\hat{w}}^T \underline{x}\underline{\hat{w}}^T \underline{x} \right\rangle && < && 0 \\
\iff & \left\langle \left(\underline{w}^T \underline{x} - \underline{\hat{w}}^T \underline{x} \right)^2 \right\rangle && < && 0
\end{aligned}
$$

Da der Erwartungswert auf der linken Seite von einem nicht-negativen Ausdruck gebildet wird, ist auch der Erwartungswert selbst nicht negativ. Es gilt daher $\left\langle \left(\underline{w}^T \underline{x} - \underline{\hat{w}}^T \underline{x} \right)^2 \right\rangle \geq 0$ und es folgt:

$$\begin{array}{rccccc} & 0 & \leq & \left\langle \left(\underline{w}^T \underline{x} - \underline{\hat{w}}^T \underline{x}\right)^2 \right\rangle & < & 0 \\ \Longrightarrow & 0 & & < & & 0 \end{array}$$

Wir erhalten einen Widerspruch, so dass wir wissen: die ursprüngliche Annahme, daß ein $\underline{w}$ mit einem kleineren Zielfunktionswert als $R(\underline{\hat{w}})$ existiert, ist falsch. Daher folgt:

$$\forall \underline{w} \in \mathbb{R}^n : R(\underline{\hat{w}}) \leq R(\underline{w})$$

Das $\underline{\hat{w}}$ ist damit ein *globales Minimum* der Zielfunktion und wir haben gezeigt, daß dazu bereits die Bedingung $\nabla R(\underline{\hat{w}}) = 0$ für ein lokales Optimum ausreicht. Damit besitzt unsere Zielfunktion 2.2 jedoch nur ein einziges lokales Optimum, das auch gleichzeitig ein globales Optimum darstellt! Da der Gradient nirgends sonst gleich Null wird, haben wir außerdem gezeigt, daß ein Gradientenabstieg *von einem beliebigen Startpunkt aus* als Konvergenzziel einen Gewichtsvektor $\underline{\hat{w}}$ mit kleinstem möglichen Wert der Fehlerfunktion 2.2 besitzt.

In wie weit sich das Gradientenverfahren diesem Minimum nähert, hängt allerdings von der genauen Wahl der Schrittweite γ ab. Ist diese zu groß, kann das Optimum übersprungen werden, so daß nachfolgende Gradientenschritte um das Minimum herum "oszillieren", anstatt sich diesem zu nähern. Kleinere Werte für γ wären zwar besser, jedoch sind dann auch mehr Gradientenschritte notwendig. In der Praxis verwendet man daher meist eine variable Lernrate, die sich erhöht, wenn der Wert der Zielfunktion sinkt und sich verringert, wenn der Zielfunktionswert ansteigt. Letzteres deutet darauf hin, daß man das Optimum aufgrund eines zu großen Gradientenschrittes "übersprungen" hat, daher ist eine Verringerung der Lernrate sinnvoll.

Mit Adaline halten wir ein recht starkes Werkzeug in der Hand. Konvergenz zu einem globalen Minimum in Verbindung mit der Fähigkeit, lineare Abbildungen lernen zu können, bedeutet nämlich, daß Adaline *alle* linearen Abbildungen lernen kann, die wir uns nur ausdenken können. Eine sehr starke und recht nützliche Eigenschaft, denn mit linearen Abbildungen lassen sich bereits eine Menge interessanter und

recht schwieriger Probleme lösen, wie wir später noch genauer sehen werden. Außerdem ist globale Konvergenz eine besondere Eigenschaft, die viele komplexere und ausdrucksstärkere Netze in der Regel nicht mehr besitzen.

nach oben