Vorlesung Neuroinformatik

Kapitel 2

Das Neuron als lernender Elementarprozessor

R. Der
Universität Leipzig
Institut für Informatik

28. April 1998

Kapitel 2

Das Neuron als lernender Elementarprozessor

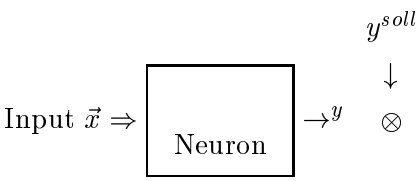
Wir untersuchen nun das oben eingeführte formale Neuron in seiner Eigenschaft als universeller Elementarprozessor neuronaler Netze.

2.1 Das Neuron als lernender Funktionsapproximator

Mathematisch ist das Neuron eine Funktion $y = f(\vec{x})$, die von einer Reihe von Parametern, zusammengefaßt im Parametervektor $\vec{w}$, abhängt, $f(\vec{x}) = f(\vec{x} \mid \vec{w})$. Das Neuron kann deshalb als Funktionsapproximator dienen, indem $\vec{w}$ so gewählt wird, daß die Funktion $f(\vec{x})$ möglichst wenig von der Zielfunktion $F(\vec{x})$ abweicht. Gesucht ist ein Lernalgorithmus, der den optimalen Parametervektor $\vec{w}$ findet. Das Lernen geschieht durch Präsentation von Instanzen der Zielfunktion $F(\vec{x})$, jede Instanz besteht aus dem Wertepaar

$$Z_{\vec{x}} = (\vec{x}, F(\vec{x})) = \left(\vec{x}, y^{soll}\right) \tag{2.1}$$

d. h. aus dem Solloutput y^{soll} des Neurons zu gegebenem Input $\vec{x}$, wobei sich im allgemeinen eine Abweichung zwischen dem Istwert $y = f(\vec{x})$ und dem Solloutput ergibt.



(2.2)

Eine geeignete Fehler– oder Kostenfunktion wie z. B.

$$E = E(\vec{x}|\vec{w}) = \frac{1}{2}\left(y^{soll} - y\right)^2 = \frac{1}{2}\left(f\left(\vec{x}\right) - F\left(\vec{x}\right)\right)^2 \tag{2.3}$$

mißt die Güte der Approximation bei gegebenem Parametersatz $\vec{w}$. Ziel ist die Minimierung dieses Fehlers durch Anpassung von $\vec{w}$ und zwar $\forall \vec{x} \in \mathbf{D}$ wobei $\mathbf{D}$ der Definitionsbereich von $F\left(\vec{x}\right)$ ist.

Im allgemeinen steht allerdings die Funktion $F\left(\vec{x}\right)$ nicht explizit zur Verfügung sondern etwa nur in Form gemessener Werte für eine definierte Auswahl von Argumenten $\vec{x}$. Sei Ω die Menge der Argumente für die die Funktionswerte $F\left(\vec{x}\right)$ bekannt sind (z. B. durch Messung). Diese Menge der bekannten Instanzen bildet die sog. Trainingsmenge oder Beispielmenge

$$\mathcal{T} = \{Z_{\vec{x}}\} = \{(\vec{x}, F\left(\vec{x}\right)) \mid \vec{x} \in \Omega\} \tag{2.4}$$

für die Bestimmung des optimalen Satzes von Parametern $\vec{w}$ durch Minimierung des Fehlers 2.3.

Die Suche nach dem besten $\vec{w}$ geht in einer Abfolge von **Lernschritten** vonstatten, wobei für jede Trainingsinstanz $\left(\vec{x}, y^{soll}\right)$, $\vec{x} \in \Omega$ der Parametervektor $\vec{w}$ um ein $\Delta\vec{w}$ (Update) gerade so verschoben wird, daß sich die Kosten verringern. Dazu muß längs jeder Achse des Koordinatensystems die Verschiebung umgekehrt proportional zur Ableitung der Kostenfunktion gewählt werden, der

Lernschritt

Lernschritt ist also

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} \quad \forall i \tag{2.5}$$

Der Vektor $\vec{G} = grad_{\vec{w}} E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right)$ ist der Gradient der Kostenfunktion, der in die Richtung des steilsten **Anstieges** von E zeigt. Das Verfahren 2.5 ist demzufolge (wegen des negativen Vorzeichens) ein Gradien-

Gradientenabstieg

tenabstieg**abstieg** auf der Landschaft der Kosten. Explizit gilt

$$\boxed{\Delta w_i = \epsilon dx_i} \tag{2.6}$$

oder vektoriell

$$\boxed{\Delta\vec{w} = \epsilon d\vec{x}}$$

wobei

$$d = \left(y^{soll} - y\right) S'\left(\vec{x} \cdot \vec{w}\right) \tag{2.7}$$

mit $S'(z) = dS(z)/dz$. Die sog. Fehleraktivität d ist proportional zur Abweichung zwischen dem aktuellen Output $y = f(\vec{x})$ des Neurons und der Sollvorgabe $y^{soll} = F(\vec{x})$ für den betrachteten Input $\vec{x}$.

Der Algorithmus konvergiert gegen ein mittleres (lokales) Minimum der Kostenfunktion wenn alle Instanzen hinreichend oft präsentiert werden und der Lernparameter ϵ geeignet "abgekühlt" wird. In der Praxis bedeutet das, daß ϵ im Laufe des Lernens von einem Startwert $\epsilon \approx 0.1$ beginnend auf Werte $\epsilon = 0.01$ bis zu $\epsilon = 0.001$ reduziert wird: Sehr kleine ϵ bedeuten sehr kleine Lernschritte, so daß nur im Mittel über viele Trainingsinstanzen eine merkliche Verschiebung von $\vec{w}$ zustande kommen kann. Die Bewegung von $\vec{w}$ ist demzufolge eine systematische Bewegung (hin zum Minimum) überlagert von kleinen Fluktuationen. Geht ϵ hinreichend langsam gegen Null, so mitteln sich die Fluktuationen immer besser heraus, ohne daß die systematische Bewegung abgebrochen wird. In der Grenze konvergiert der Algorithmus damit gegen das lokal optimale $\vec{w}$. Für die richtige Strategie zur Abkühlung gibt es eine genaue definierte Strategie, die streng aus dem Robins–Monroe–Theorem folgt, s. [1].

Dabei können die Instanzen als Ergebnisse von Messungen auch "verrauscht" sein, der Lernalgorithmus 2.6 konvergiert dann gegen ein $\vec{w}$ so, daß $f(\vec{x} \mid \vec{w})$ den Funktionszusammenhang im Mittel repräsentiert.

Ist $S(z) = z$ (lineares Neuron), so ist nach dem Lernen $f(\vec{x} \mid \vec{w})$ mit dem Ergebnis der Methode der linearen Regression identisch.

2.1.1 On–line Lernen vs. batch Lernen

Im Lernalgorithmus 2.7 wird für jeden Input $\vec{x}$ sofort der Lernschritt ausgeführt, deshalb ist das ein on–line Algorithmus. Der besondere Vorteil dieses Lernverfahrens besteht in seiner Echtzeitfähigkeit, er nähert sich inkrementell dem Optimum und kann deshalb nach jedem Einzelschritt abgebrochen werden, eventuell um den Preis eines suboptimalen Ergebnisses. Außerdem werden neu hinzukommende Trainingsbeispiele zwanglos integriert.

Das eigentliche Ziel ist aber natürlich, den Fehler im Mittel über alle Inputs, d. h. für alle $\vec{x} \in \Omega$ zu minimieren, die zugehörige Fehlerfunktion ist dann durch den Mittelwert aller Einzelfehler gegeben

$$E(\vec{w}) = \sum_{\vec{x} \in \Omega} E(\vec{x} \mid \vec{w}) P(\vec{x}) \tag{2.8}$$

wobei $P(\vec{x})$ die Wahrscheinlichkeit (relative Häufigkeit) für den Input $\vec{x}$ ist. Gradientenabstieg auf dieser Kostenfunktion entspricht einem sog. batch Algorithmus, bei dem also alle Inputs erst gesammelt werden müssen, um E berechnen und dann den Gradientenabstieg durchführen zu können. Der on–line Algorithmus ist die stochastische Approximation des batch Algorithmus. Bei geeigneter Abkühlungsstrategie für die Lernrate ϵ konvergiert der Algorithmus mit Wahrscheinlichkeit 1 gegen das Minimum von $E(\vec{w})$.

2.1.2 Beipiele

Konvergenzverhalten bei quadratischer Fehlerfunktion

Wir betrachten zunächst den einfachen Fall, daß die Kostenfunktion nicht von x abhängt und daß diese quadratisch sei

$$E(\vec{w}) = \sum_{ij} w_i A_{ij} w_j \tag{2.9}$$

wobei das w–Koordinatensystem willkürlich so gewählt wurde, daß am Minimum $\vec{w} = 0$. Die Form 2.9 der Fehlerfunktion ist allgemein in der Umgebung eines lokalen Minimums gültig. Sie ergibt sich aus einer Taylorentwicklung um das Minimum unter Berücksichtigung von $\partial E/\partial w_i = 0 \quad \forall i$ am Minimum. Dann ist $A_{ij} = A_{ji}$ so daß die quadratische Form diagonalisiert und damit das Problem auf die Betrachtung des eindimensionalen Falles reduziert werden kann, d. h. wir betrachten (mit $a > 0$)

$$E = \frac{1}{2} a w^2$$

und folglich

$$\Delta w = -\epsilon a w \tag{2.10}$$

wobei a die Öffnungsweite der Parabel ist. Dieser Parameter macht also eine Aussage über die Steilheit der Abhänge des Tales. Sei t mit $t = 0, 1, ...$ der Schrittzähler der Lernschritte, so entspricht Gleichung 2.10

$$w(t+1) = (1 - \epsilon a)\, w(t) \tag{2.11}$$

mit Lösung

$$w(t) = e^{-t/\tau} w(0) \quad \text{falls} \quad \epsilon a < 1 \tag{2.12}$$

wobei

$$\tau = -\frac{1}{\ln(1 - \epsilon a)} \tag{2.13}$$

Zeitkonstante des Lernens

Von einem beliebigen Wert $w(0)$ startend konvergiert für $\epsilon a < 1$ der Algorithmus offensichtlich exponentiell mit Zeitkonstante τ gegen das Minimum. Für hinreichend kleine ϵ so daß $\epsilon a << 1$ folgt durch Taylorentwicklung

$$\tau = \frac{1}{\epsilon a} \quad \text{für} \quad \epsilon a << 1 \tag{2.14}$$

Die Lernrate ϵ bestimmt somit direkt die Konvergenzgeschwindigkeit gegen das Minimum der Fehlerfunktion.

Formel 2.12 ist allerdings nur für $\epsilon a < 1$ gültig. Für $\epsilon a = 1$ wird das Minimum in einem Schritt erreicht. Das ist formal in 2.13 enthalten, da für $\epsilon a \to 1$ die Zeitkonstante $\tau \to 0$ konvergiert, was einer unendlich großen Konvergenzgeschwindigkeit des Algorithmus entspricht. Für $1 < a\epsilon < 2$ ergibt sich eine alternierende aber noch konvergente Folge und für $\epsilon a > 2$ divergiert der Lernalgorithmus schließlich.

Divergenz des Algorithmus bei zu großen Lernraten

Das Neuron als Lehrer und Schüler

Wir betrachten – der Einfachkeit halber für einen eindimensionalen Input – den Fall, daß ein Neuron mit fester synaptischer Kopplungsstärke $w = 1$ als Lehrer die Solloutputs für ein Schülerneuron vorgibt, d h.

$$F(x) = \frac{1}{1 + e^{-x}}$$

Wie aus der Abbildung 2.1 klar ersichtlich hat die Kostenfunktion

$$E = \frac{1}{2}\left(\frac{1}{1 + e^{-wx}} - \frac{1}{1 + e^{-x}}\right)^2$$

das erwartete globale Minimum bei $w = 1$, so daß das Schülerneuron durch Gradientenabstieg tatsächlich exakt das Verhalten des Lehrers erlernen kann. Dieses Minimum ist unabhängig von x, so daß der Lernalgorithmus auch für feste Werte von ϵ konvergiert.

Synaptische Divergenz und synaptischer Kollaps

Eine häufige "Fehlleistung" beim Lernen sind Divergenz oder Kollaps der synaptischen Kopplungen. Das tritt immer dann auf, wenn die Sollvorgaben mit der Transferfunktion des Neurons inkompatibel sind, wir betrachten z. B. den Fall $F(x) = 1$ mit Fehlerfunktion

$$E = \frac{1}{2}\left(\frac{1}{1 + e^{-xw}} - 1\right)^2 \tag{2.15}$$

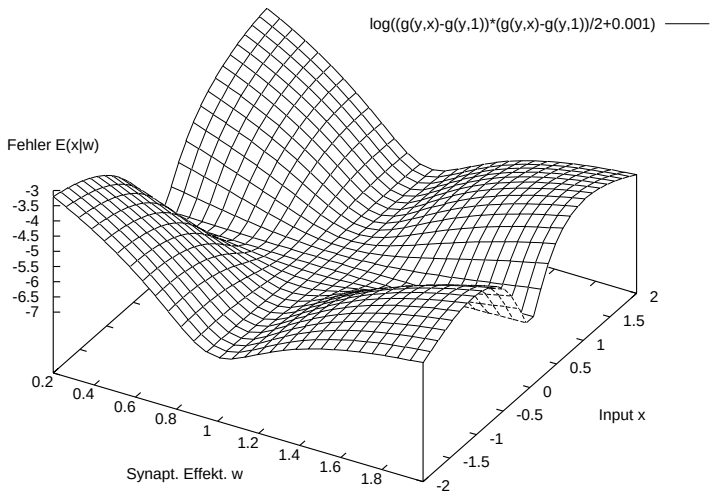


Abbildung 2.1: Die Fehlerfunktion für das Training eines Schülerneurons durch ein Lehrerneuron mit $w = 1$. Da die eigentliche Fehlerfunktion sehr flach ist, wurde zur besseren Sichtbarmachung des Minimums bei $w = 1$ die Darstellung $\log\left(E\left(x \mid w\right) + 0.001\right)$ für den Fehler gewählt.

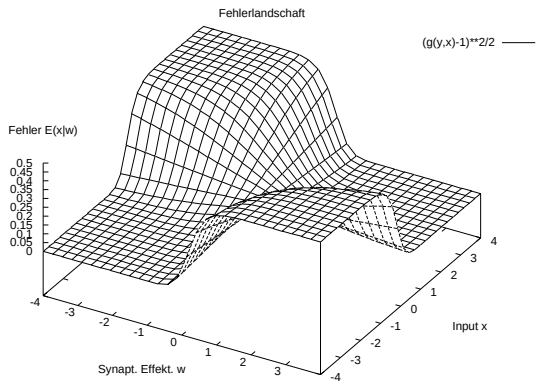


Abbildung 2.2: Die Fehlerlandschaft für das Erlernen der Zielfunktion $F = 1$.

Die Fehlerfunktion ist urpsrungssymmetrisch. Wie schon aus der Fehlerfunktion explizit ersichtlich treibt für $x > 0$ die Gradientenabstiegsdynamik $w \to -\infty$, für $x < 0$ folgt analog $w \to \infty$.Werden also ausschließlich Werte aus dem einen oder anderen Bereich von x angeboten, so divergiert die synaptische Kopplungsstärke. Werden andererseits Beispiele mit sowohl $x > 0$ als auch $x < 0$ angeboten, so konvergiert w gegen einen (kleinen) Wert. Sind die x Werte im Vorzeichen gleichverteilt, so konvergiert w gegen Null. Man kann hier vom Kollaps der Synapsen sprechen oder auch von der Selbstabschaltung des Neurons als Reaktion auf Überforderung durch die aus der Sicht des Neurons widersprüchlichen Sollvorgaben.

Synaptische Divergenz

Synaptischer Kollaps durch inkonsistente Lehrer

Beide Phänomene – synaptischer Kollaps wie synaptische Divergenz – treten leicht in den feed–forward Netzen (s.u.) auf, die nach der Methode der Fehlerrückpropagierung trainiert werden.. Danach erhält ein jedes Neuron die Sollvorgaben im Lernschritt von ihm nachgeordneten Neuronen zugewiesen. Ist die Architektur unglücklich gewählt, so können die Forderungen (Sollwerte y_j^{soll}), die an ein bestimmtes Neuron j im Netz auf diese Weise herangetragen werden, mit der Transferfunktion des Neurons inkompatibel sein. Im Umkehrschluß können aus dem Verhalten der Synapsen im Lernprozeß Aussagen über eventuell erforderliche Veränderungen der Architektur abgeleitet werden.

Synapsendynamik und Netzwerkarchitektur

Erlernen einer komplizierteren Funktion

Als Beispiel für eine kompliziertere Fehlerlandschaft betrachten wir die Approximation der sin–Funktion durch ein Neuron, d. h. wir setzen

$$F(x) = \frac{1}{2}(\sin x + 1) \tag{2.16}$$

so daß $y^{soll} \in [0,1]$ und

$$f(x) = \frac{1}{1+e^{-wx}} \tag{2.17}$$

wie gehabt. Für $w \approx 2$ ergibt sich für $x \in [-2,2]$ eine recht gute Übereinstimmung zwischen f und F, s. Abbildung 2.3, für Werte außerhalb dieses Intervalls wird die Übereinstimmung aber beliebig schlecht.

Die Fehlerfunktion $E(x \mid w)$ ist durch eine recht komplizierte Struktur gekennzeichnet. Wir beobachten einerseits für $x \in [-2,2]$ um $w = 2$ einen relativ konzentrierten Minimumsbereich, was die recht gute Übereinstimmung zwischen f und F in diesem Bereich widerspiegelt, s. Abbildungen 2.3 und 2.5. Anderer-

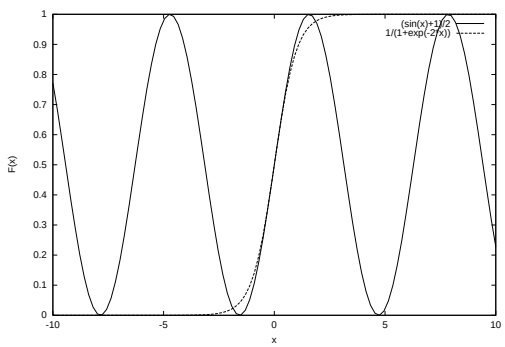


Abbildung 2.3: Die Funktion $F(x) = \frac{1}{2}(\sin x + 1)$ und die Transferfunktion $f(x) = 1/(1 + e^{-xw})$ für $w = 2$.

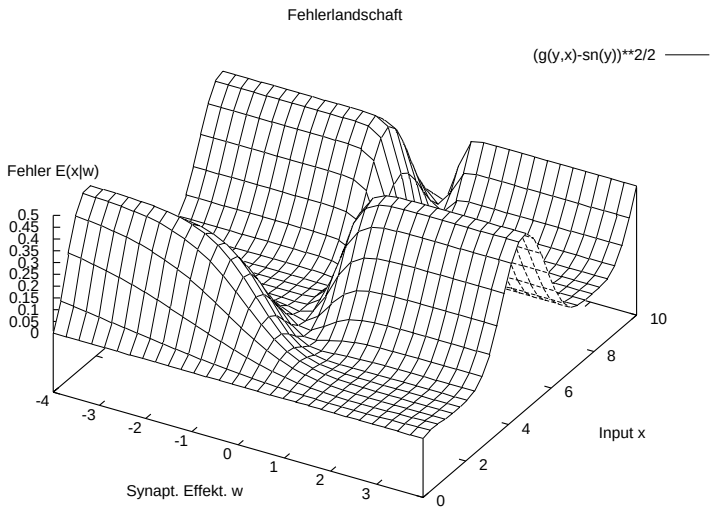


Abbildung 2.4: Die Fehlerfunktion für die Approximation der Funktion $F(x) = (\sin x + 1)/2$ durch ein Neuron mit Transferfunktion $f(x) = \frac{1}{1+e^{-wx}}$.

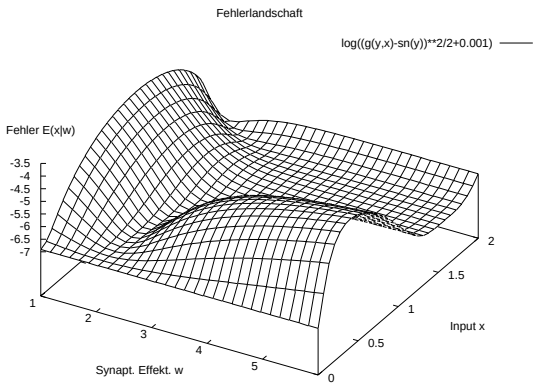


Abbildung 2.5: Die Fehlerfunktion für die Approximation der Funktion $F(x) = (\sin x+1)/2$ durch ein Neuron im Bereich $x \in [0, 2]$. Obwohl das Neuron mit $w \approx 2$ die Zielfunktion $F(x)$ in dem betrachteten Bereich gut approximiert hängt das Minimum von $E(x \mid w)$ deutlich von x ab. Da die eigentliche Fehlerfunktion sehr flach ist, wurde $\log(E(x \mid w) + 0.001)$ dargestellt.

seits hängt, wie aus Abbildung 2.6 ersichtlich, außerhalb dieses Bereiches das Minimum von E sehr stark von x ab.

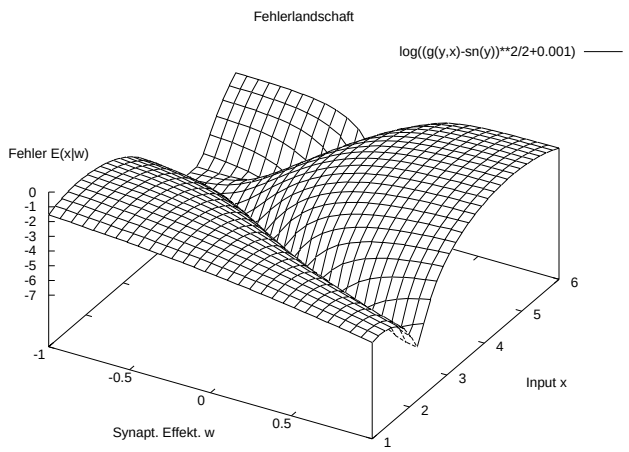


Abbildung 2.6: Die Fehlerfunktion für die Approximation der Funktion $F(x) = (\sin x + 1)/2$ durch ein Neuron im Bereich $x \in [2, 6]$. Das Neuron approximiert $F(x)$ in diesem Bereich sehr schlecht, was durch die komplizierte Fehlerlandschaft reflektiert wird. Dargestelt wurde wieder $\log(E(x \mid w) + 0.001)$.

2.2 Das Neuron als lernender Klassifikator

Sei $\mathcal{K}$ eine Menge von Datenpunkten, die in zwei Teilmengen (Klassen) $\mathcal{K}_1, \mathcal{K}_2$ zerfällt, mit $\mathcal{K}_1 \cap \mathcal{K}_2 = \emptyset$. Ein Klassifikator ist eine Funktion $x \longmapsto K(\vec{x})$ mit der Eigenschaft

$$K(\vec{x}) = \begin{cases} 1 & \text{für} \quad \vec{x} \in \mathcal{K}_1 \\ \\ 0 & \text{für} \quad \vec{x} \in \mathcal{K}_2 \end{cases} \tag{2.18}$$

Ein Neuron mit Transferfunktion $y = f(\vec{x})$ und einer sigmoiden Ausgabefunktion kann diese Funktion K nur repräsentieren, wenn wir zusätzlich eine Akzeptanzschwelle einführen, d. h. wir definieren (für den Fall der sigmoiden Ausgabefunktion) die Klassifikatorfunktion $K(\vec{x})$ in Termen der Transferfunktion $f(\vec{x})$ des Neurons als

$$K(\vec{x}) = \begin{cases} 1 & \text{falls} \quad f(\vec{x}) > 1 - k \qquad \text{(Klasse 1)} \\ \\ 0 & \text{falls} \quad f(\vec{x}) < k \qquad \text{(Klasse 2)} \end{cases} \tag{2.19}$$

wobei $0 < k < 1/2$. Die Menge der $\vec{x}$, für die $k \leq K(\vec{x}) \leq 1 - k$ gilt, bilden die sog. Rückweisungsklasse, da sie durch den Klassifikator nicht klassifiziert

werden. Ist speziell $k = 1/2$, dann ist

$$K = 1 \quad falls \quad \vec{w} \cdot \vec{x} > U \qquad \text{(Klasse 1)} \tag{2.20}$$

$$K = 0 \quad falls \quad \vec{w} \cdot \vec{x} < U \qquad \text{(Klasse 2)}$$

Das Klassifikationsproblem ist durch das Neuron lösbar, wenn die beiden Teilmengen linear separabel sind. Das ist allgemein dann der Fall, wenn eine Trennebene existiert, so daß jede Teilmenge auf jeweils einer Seite der Ebene liegt. Die Gleichung $f(\vec{x}) = S(\vec{w} \cdot \vec{x} - U) = 1/2$ bzw.

$$\vec{w} \cdot \vec{x} = U$$

oder

$$\sum_{i=1}^{n} w_i x_i = U \tag{2.21}$$

definiert eine zum Vektor $(w_1, ..., w_n)$ orthogonale (Hyper–)Ebene im Abstand $U/|\vec{w}|$ zum Ursprung des Koordinatensystems. Das Neuron kann damit ein beliebiges linear separables Klassifikationsproblem lösen, wenn nur $\vec{w}$ (und U) so gewählt ist, daß die Menge der Datenpunkte $\mathcal{K}_1$ oberhalb (in Richtung des Vektors $(w_1, ..., w_n)$ gesehen) und die der Menge $\mathcal{K}_2$ unterhalb dieser Trennebene liegt.[1]

Diese Eigenschaft kann das Neuron nach obiger Regel erlernen[2]. Als Trainigsinstanzen dienen dabei die Datenpunkte $\vec{x} \in \mathcal{K}$ mit der dazugehörigen Klassenzuweisung $y^{soll} = K(\vec{x}) \in \{0, 1\}$, d. h.

$$\mathcal{T} = \{(\vec{x}, K(\vec{x}))\}$$

Damit ist das Neuron ein lernender Klassifikator für jedes beliebige lineare Klassifikationsproblem. Umgekehrt kann das Neuron nichtlineare Klassifikationsprobleme bestenfalls nur näherungsweise lösen.

[1] Falls $k < 1/2$ gewählt wurde, die Rückweisungsklasse also nicht leer ist, wird die Grenze zwischen den klassifizierbaren Objekten durch einen Streifen markiert, dessen Breite eine Funktion von k ist.

[2] Dazu ist es günstig, die Schwelle durch die Einführung eines zusätzlichen Inputs einzubeziehen, wie in Kapitel ?? diskutiert. Dadurch wird die Schwelle in Form der synaptischen Verbindungsstärke zu diesem Input mitgelernt.

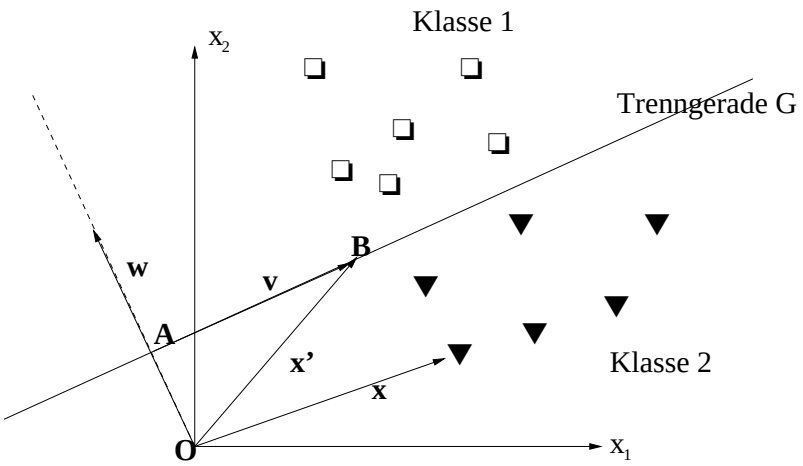


Abbildung 2.7: Die durch Gleichung 2.21 definierte Trenngerade. Elemente der Klassen 1 bzw. 2 sind durch Quadrate bzw. Dreiecke dargestellt. Jedes Element einer Klasse ist durch seinen Ortsvektor $\vec{x}$ repräsentiert. Der Abstand OA ist gleich $U/|\vec{w}|$, wobei $|\vec{w}|$ die Länge des Vektors $\vec{w}$ ist. Der Vektor $\vec{v}$ ist die gerichtete Strecke von A nach B. Für jeden Punkt B, (dargestellt durch seinen Ortsvektor $\vec{x}'$) auf der Trenngeraden gilt $\vec{v}\vec{w} = 0$ was gerade Gleichung 2.21 entspricht.

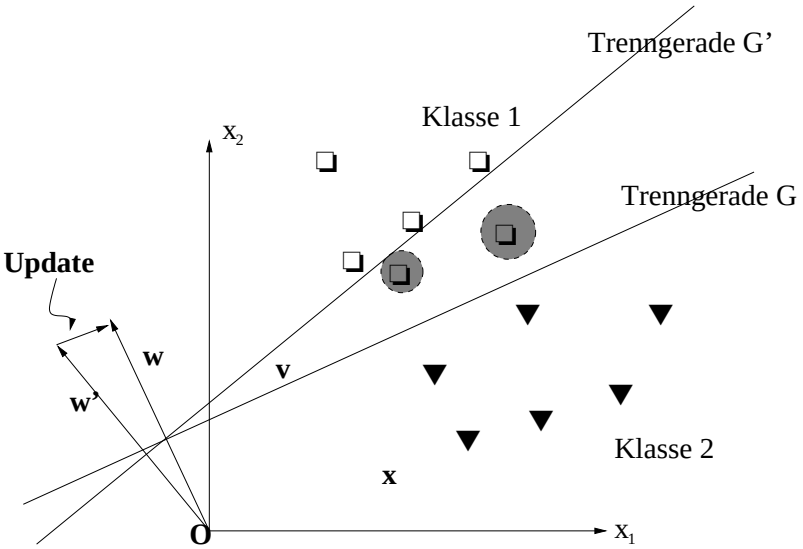


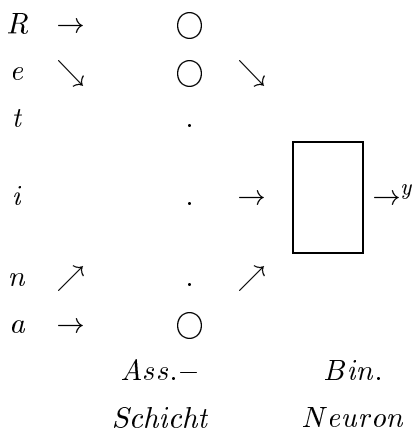
Abbildung 2.8: Fehlklassifikationen können durch Variation $\vec{w}' \rightarrow \vec{w}$ des Vektors $\vec{w}$ der synaptischen Effektivitäten korrigiert werden.

2.3 Das Perzeptron

Das Perzeptron spielte in der Frühphase des Studiums neuronaler Systeme als Medien der KI eine zentrale Rolle. Dieses besteht aus einer (künstlichen) Retina, die über synaptische Verbindungen die binären Neuronen der sog. Assoziationsschicht speist. Diese koppeln die binären Outputs $x_1, ..., x_n$, $x_i \in \{0, 1\}$ aus, die als Inputs eines binären Neurons mit Transferfunktion

$$y = \theta\left(\vec{w} \cdot \vec{x} - U\right) \tag{2.22}$$

dienen, das die Klassifikation der auf die Retina projizierten Muster vornimmt.



(2.23)

Die Vorverarbeitungsschicht sollte im wesentlichen die Umsetzung der retinalen Information in eine linear seperable Klassifikationsaufgabe bewerkstelligen, die dann durch ein formales Neuron lösbar wäre.

Damit ist das Perzeptron ebenfalls ein lernender Klassifikator, allerdings ist hier der oben abgeleitete Lernalgorithmus nicht einsetzbar, da die Ableitung $S'(z)$ für die unstetige Funtkion $S(z) = \theta(z)$ an $z = 0$ nicht existiert. Das richtige Inkrement von $\vec{w}$ im Lernschritt kann man sich durch die folgenden Überlegungen herleiten:

1. Falls das präsentierte $\vec{x}$ durch das Perzeptron richtig klassifiziert wurde, muß $\vec{w}$ nicht geändert werden, d. h. $\Delta\vec{w} = 0$.

2. Bei Fehlklassifikation gilt für $\vec{x} \in \mathcal{K}_1$, daß $z = \vec{w} \cdot \vec{x} < U$. [3] Die Wahl $\Delta\vec{w} = \epsilon\vec{x}$ (wobei $0 < \epsilon < 1$) bewirkt die Vergrößerung des PSP um die

[3] Wir nehmen an, daß die Schwelle richtig gewählt wurde, sonst kann diese wieder durch die Mitnahme eines fiktiven Zusatzinputs mitgelernt werden.

positive Größe $\Delta z = \epsilon \vec{x}^2$ und ist damit ein Schritt in die richtige Richtung, da ja bei richtiger Klassifikation $z > U$ sein muß vgl. Abb. 2.9. Analog gilt für $\vec{x} \in \mathcal{K}_2$, daß $\Delta \vec{w} = -\epsilon \vec{x}$ gewählt werden muß.

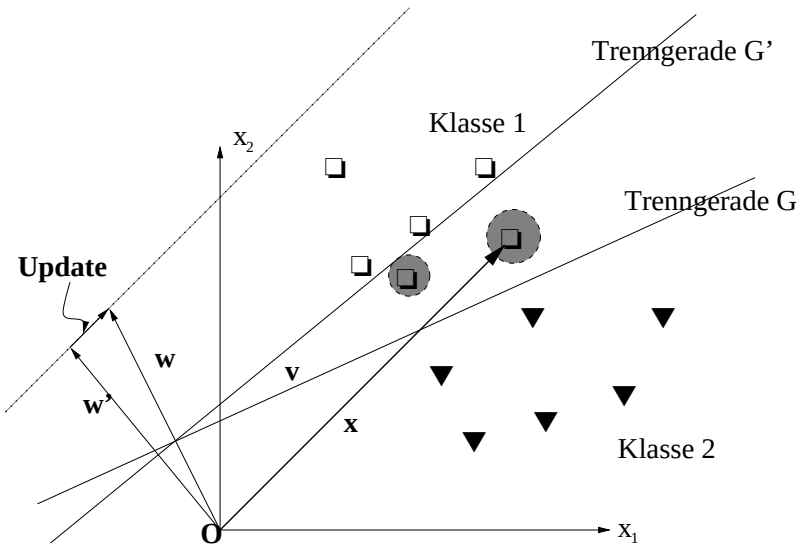


Abbildung 2.9: Zur Perzeptron–Lernregel: Der Input mit Vektor x ist fehlklassifiziert. Der Update $\Delta \vec{w} = \epsilon \vec{x}$ verschiebt $\vec{w}$ in die richtige Richtung. Die gepunktete Linie verläuft parallel zum Vektor $\vec{x}$. Durch den Update verändert sich nicht nur die Richtung sondern auch die Länge des Vektors $\vec{w}$, was einer Änderung der Schwelle entspricht. Das kann vermieden werden, wenn die Schwelle durch Einführung eines fiktiven Inputs x_0 wie eine zusätzliche Synapse behandelt werden kann.

Das kann in die Perzeptron–Lernregel

$$\boxed{\Delta \vec{w} = \epsilon d \vec{x}} \tag{2.24}$$

mit

$$\boxed{d = y^{soll} - y}$$

zusammengefaßt werden wobei $K(\vec{x}) = y^{soll} = 1$ falls $x \in \mathcal{K}_1$ und $y^{soll} = 0$ sonst. Gleichung 2.24 kann wieder im Sinne einer Hebbschen Regel interpretiert werden, wenn wir die Fehleraktivität $d = \left(y^{soll} - y\right)$ als Outputaktivität

des lernenden Neurons verstehen. Für diese Lernregel existiert ein Konvergenzbeweis, der zeigt, daß der Algorithmus nach einer **endlichen** Zahl von Schritten konvergiert, falls die Klassen linear trennbar sind und ϵ geeignet gewählt wurde.

2.4 Grenzen des Neurons – das XOR Problem

Binäre Neuronen mit binären Inputs $x_i \in \{0,1\}$, $i = 1, ..., n$ stellen eine Boolsche Funktion $\{0,1\}^n \mapsto \{0,1\}$ dar. Allerdings ist die Realisierung Boolscher Funktionen durch ein einzelnes Neuron in vielen Fällen unmöglich. Schon die wichtige zweistellige Boolsche Funktion XOR

$$\begin{array}{ccc} x_1 & x_2 & y \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \tag{2.25}$$

ist nicht mehr realisierbar. In der Tat, die Inputs $\vec{x}$ liegen auf den Eckpunkten des Einheitsquadrates $[0,1]\times[0,1]$ in der $x_1 - x_2$–Ebene wobei sich die Repräsentanten für jede der beiden Klassen diagonal gegenüberliegen. Sie sind deshalb durch eine Gerade nicht trennbar, so daß das Problem nicht linear separabel ist. Eine Lösung ergibt sich erst in der Zusammenschaltung mehrerer Neuronen zu einem neuronalen Netz, s. Kapitel **??**.

Weiterführende Literatur: Geeignete Lehrbücher sind [4, 5, 3, 2], wobei in [3] eine besonders anschauliche Darstellungsweise gefunden wurde.

Literaturverzeichnis

[1] R. Der. Vorlesung Maschinelles Lernen. UniversitätLeipzig Institut für Informatik. kap. 2: Parameteradaptive Lernverfahren. http://www.informatik.uni-leipzig.de/ der/Vorlesungen/ml-parad.ps.Z.

[2] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, New York, 1991.

[3] R. Raul. *Theorie der neuronalen Netze.* Springer-Verlag, 1993.

[4] H. J. Ritter, T. M. Martinetz, and K. J. Schulten. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Abbildungen.* Addison-Wesley, Munich, Germany, 1990.

[5] A. Zell. *Simulation neuronaler Netze.* Addison–Wesley, 1994.