Einleitung

In diesem ersten Aufgabenblatt wollen wir uns erst einmal einige grundlegende Eigenschaften von Neuronen und Künstlichen Neuronalen Netzen ansehen. Die Aufgaben dieses Aufgabenblattes sind recht einfach gehalten und dienen dazu, Sie sich erst einmal ein wenig mit Python, Eclipse und dem Plotter vertraut mach können.

Ein Dokument mit einer Kurzbeschreibung des Praktikumsinhalts, sowie einiger Hilfestellungen zu Eclipse, ist im Aufgabenverzeichnis vorhanden. Hier steht unt anderem auch, was Sie bei der Bearbeitung der Aufgaben beachten sollten. Ein Codebeispiel zur Verwendung des Plotters kann im Material-Verzeichnis abgeru werden.

Biologische Nervenzellen

In diesem Praktikum werden wir uns überwiegend mit einem bestimmten Neuronenmodell beschäftigen, aus dem wir dann lernfähige Neuronennetze bild werden, um damit verschiedenste Arten von praktischen Problemen in den Griff bekommen. Schauen wir uns daher kurz an, wie überhaupt eine echte Nervenze arbeitet und wie die formalen Neuronen funktionieren, mit denen wir uns im folge beschäftigen werden.

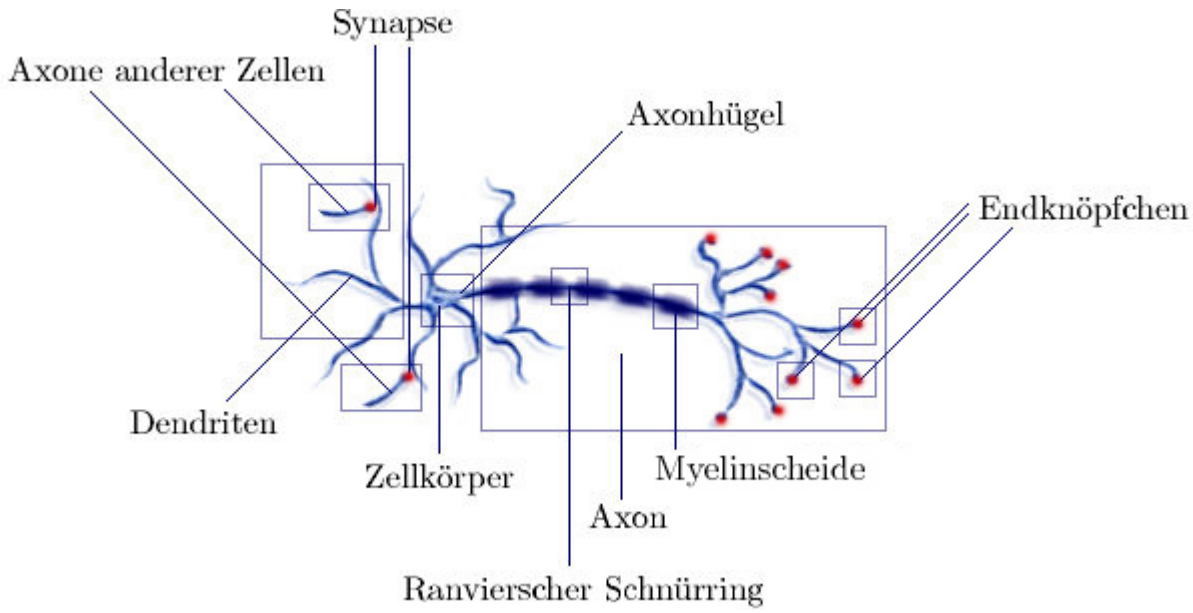

Abbildung 1,1 Darstellung einer typischen biologischen Nervenzelle.

In Abbildung 1 ist das Schema einer typischen biologischen Nervenzelle dargestellt, wie sie im Zentralnervensystem (dazu zählen das Gehirn und das Rückenmark, im Gegensatz zum peripheren Nervensystem) des Menschen (und anderer Tiere) vorkommt.

Vereinfacht gesagt, besteht die Funktionsweise einer solchen Zelle darin, über die Dendriten viele Signale (etwa von anderen Nervenzellen) aufzunehmen und, abhängig davon, ein einzelnes Ausgangssignal zu erzeugen. Die Eingangssignale führen dabei zu einer Veränderung des elektrischen Potentials an der Stelle des Dendriten, an dem dieses Signal "abgegriffen" wird. Diese Potentialveränderung breitet sich dann vom Dendriten über die ganze Zelle hinweg aus, wobei sie mit zunehmender Entfernung vom Entstehungsort an Intensität verliert. Finden mehrere Potentialänderungen innerhalb eines kurzen Zeitabschnittes statt, etwa an der selben oder an verschiedenen Stellen der Dendriten, so können sich diese Potentialänderungen addieren. Dieses Verhalten wird als *räumliche* (verschiedene Reize an unterschiedlichen Stellen) bzw. *zeitliche* (aufeinanderfolgende Reize an der selben

Stelle) Summation bezeichnet.

Wird aufgrund der eingehenden Reize nun ein bestimmter Schwellenwert des Membranpotentials am Axonhügel der Zelle überschritten, so beginnt die Zelle ihrerseits damit, am Axonhügel ein elektrisches Ausgangssignal - das sogenannte Aktionspotential - zu erzeugen, das sich dann selbständig über das Axon bis hin zu den Endknöpfchen fortpflanzt.
An dieser Stelle ist anzumerken, dass die Intensität der Aktionspotentiale hier *nicht* der zurückgelegten Wegstrecke abnimmt, da die Ausbreitung *aktiv* erfolgt, d.h. es kommt genau genommen zu einer Kettenreaktion von Aktionspotentialen, die nacheinander entlang des gesamten Axons gebildet werden und so den Eindruck eines einzelnen sich "fortbewegenden" Aktionspotentials vermitteln.

In der Regel werden dabei viele Aktionspotentiale hintereinander erzeugt, deren Frequenz mit der Stärke aller aufsummierten Eingangsreize zusammenhängt. Oftmals sind die Axone von Nervenzellen dabei noch mit einer sog. Myelinschicht umgeben, die das Axon umhüllen und deren Aufgabe sowohl darin besteht, das Axon von den Aktionspotentialen anderer Zellen elektrisch zu isolieren, als auch die Ausbreitungsgeschwindigkeit der Aktionspotentiale beträchtlich (ca. Faktor 10-100) zu erhöhen.
An den Endknöpfchen setzen die ankommenden Aktionspotentiale dann chemische Botenstoffe (Neurotransmitter) frei, die den Spalt zwischen Axon und Dendrit (die Synapse) überbrückt und an den Dendriten der nachgeschalteten Zellen wiederum eine Änderung des elektrischen Potentials bewirken.

Wichtig ist noch die Tatsache, dass dabei (abhängig von der Art der ausgeschütteten Neurotransmitter) sowohl eine Erhöhung als auch eine Verminderung des elektrischen Potentials am Dendriten der nachgeschalteten Zelle bewirkt werden kann. Damit kann also sowohl eine erregende (die Bildung von Aktionspotentialen begünstigende), als auch eine hemmende (die Bildung von Aktionspotentialen unterdrückende) Wirkung resultieren, je nach Art der Synapse.

Interessante dabei ist auch, dass Synapsen in der Lage sind, ihre Struktur und ihre physikalischen Eigenschaften in Abhängigkeit von der elektrischen Aktivität anzupassen (man spricht dabei von "Synaptischer Plastizität"). Der Einfluß von Aktionspotentialen auf nachgeschaltete Zellen kann also durch die Aktivität selbst sich verändern. Man ist der Überzeugung, dass dies die Grundlage für die Lernenfähigkeit von Nervenzellen darstellt.

nach oben

Formale Neuronen

Man kann nun versuchen, die Art und Weise, wie eine biologische Nervenzelle (oder Netze aus Nervenzellen) aus ihren Eingangssignalen ein Ausgangssignal erzeugt, in ein einfaches Modell (siehe Abbildung 2) umzusetzen, mit dem sich gut "arbeiten" läßt. Sinn davon ist, die Vorteile biologischer Neuronen für die Informationsverarbeitung nutzen zu können: die Lernfähigkeit, der starken Parallelisierung und der Fehlertoleranz derartiger Systeme.

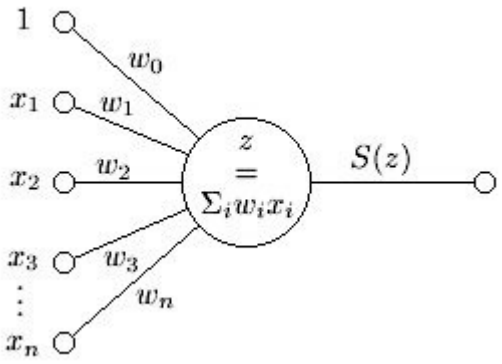

Abbildung 1,2 Neuronenmodell.

Die Funktionsweise des Neuronenmodells, das wir hier überwiegend verwenden werden, ist die folgende: ein *formales* Neuron, wie es in Abbildung 2 dargestellt ist,

besteht aus einer Reihe von reellen Eingaben $x_1, ..., x_n \in \mathbb{R}$ und Verbindungsgewichten $w_1, ..., w_n \in \mathbb{R}$ zu diesen Eingaben. Dies modelliert den Dendritenbaum und die Synapsen eines biologischen Neurons.
Meist wird dabei noch eine konstante Eingabe $x_0 = 1$ (sog. *Bias* oder *offset*) mit einem Verbindungsgewicht w_0 hinzugenommen. Hier wird das Schwellenpotential modelliert, ab dem eine Nervenzelle anfängt, Aktionspotentiale zu erzeugen.
Anschließend werden die Eingaben mit den Gewichten mittels $z = \sum_{i=0}^{n} x_i w_i$ aufsummiert (wir bezeichnen z dabei als die *Aktivierung* des Neurons) und eine Ausgabefunktion $S(z)$ darauf angewandt, welche die endgültige Ausgabe des Neurons berechnet. Diese kann dann wiederum anderen Neuronen als Eingabe dienen, so dass wir Neuronen untereinander verbinden und Netzwerke daraus bilden können.

Die Aktivierung z läßt sich auch in Vektorform (notiert als Variable mit Unterstrich) schreiben mit Hilfe des Skalarprodukts zweier Spaltenvektoren $\underline{x}$ und $\underline{w}$ als

$$\begin{array}{lll} & z = \underline{w}^T \underline{x}, & \underline{w} := [w_0, w_1, ..., w_n]^T, \quad \underline{x} := [1, x_1, ..., x_n]^T \\ \iff & z = \underline{\tilde{w}}^T \underline{\tilde{x}} + w_0, & \underline{\tilde{w}} := [w_1, ..., w_n]^T, \quad \underline{\tilde{x}} := [x_1, ..., x_n]^T \end{array}$$

Ist die Aktivierung des Neurons gleich null, so erhalten wir:

$$\begin{array}{rcl} z & = & \underline{\tilde{w}}^T \underline{\tilde{x}} + w_0 = 0 \\ \iff \underline{\tilde{w}}^T \underline{\tilde{x}} & = & -w_0 \end{array}$$

Ein Ausdruck der Form $\underline{c}^T \underline{x} = \theta$ mit konstanten $\underline{c} \in \mathbb{R}^n, \theta \in \mathbb{R}$ und variablem $\underline{x}$ bezeichnet man auch als die *Hessesche Normalform* einer Hyperebene. Streng genommen gehört dazu noch die Normierung des Vektors *c* - in diesem Fall stellt gerade den Abstand der Hyperebene vom Nullpunkt dar - aber dies wollen wir hier einmal nicht so eng sehen. Sie beschreibt durch

$$\mathcal{H}(\underline{c}, \theta) = \{\underline{x} \in \mathbb{R}^n : \underline{c}^T \underline{x} = \theta\}$$

eine Hyperebene im *n*-dimensionalen euklidischen Raum und ist durch die Angabe eines $\underline{c} \in \mathbb{R}^n$ und eines Skalars $\theta \in \mathbb{R}$ eindeutig bestimmt. $\underline{c}$ ist dabei ein Vektor (der sogenannte Normalenvektor), der senkrecht auf der Hyperebene steht und dadurch die Drehlage der Hyperebene im Raum beschreibt. θ gibt die Verschiebung der

Hyperebene an (bei $\theta = 0$ geht die Hyperebene durch den Nullpunkt).

Mit den vorstehenden Überlegungen können wir folgern, dass, mathematisch gesehen, die Aktivierung unserer Neuronen genau dann null wird, wenn die Eingabe auf der durch die Neuronengewichte definierten Hyperebene liegt. Wir können also jedem Neuron anhand seines Gewichtsvektors eine solche Hyperebene zuordnen. Die einzige Ausnahme ist der pathologische Fall $w_1 = w_2 = ... = w_n = 0$, da dann $[w_1, w_2, ..., w_n]^T$ keinen Normalenvektor beschreibt.

Haben wir mehrere, parallel auf den selben Eingabedaten arbeitende Neuronen, so bezeichnen wir sie als *Schicht*. In der folgenden Abbildung sind zwei solcher Schichten hintereinander gezeichnet; die Ausgabe der ersten Schicht mit *s* Neuronen wird von der zweiten Schicht aus *m* Neuronen als Eingabe verarbeitet. Da wir nur die Ausgabe beobachten können und nicht die internen Aktivitäten, werden die Anfangsschichten als "versteckte Schichten" (*hidden layer*) bezeichnet.

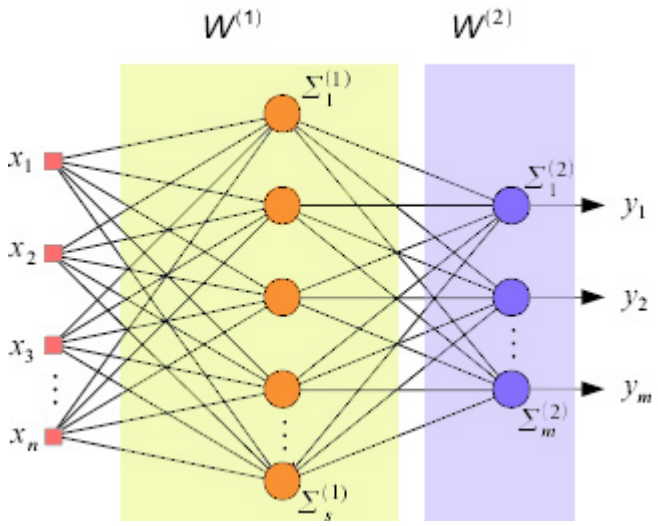

Abbildung 1,3 Neuronenschichten.

Alle Neuronen einer Schicht arbeiten parallel; die einzelnen Schichten aber sequentiell hintereinander. Jeweils alle Gewichtsvektoren einer Schicht lassen sich zu einer Matrix **W** zusammenfassen. Beispielsweise hat die Gewichtsmatrix der obigen ersten, versteckten Schicht *s* Spalten und *n* Zeilen und die der Ausgangsschicht *m* Spalten

und *s* Zeilen.

nach oben

Das Perzeptron

Eine klassische Anwendung Künstlicher Neuronaler Netze ist die Klassifikation von Eingabedaten. Ziel dabei ist es, zu einer Reihe von Messwert-Tupeln, die wir als *Muster* bezeichnen, eine Entscheidung darüber zu treffen, ob das Muster Bestandteil einer ganz bestimmten Menge von Mustern ist, die wir als *Klasse* bezeichnen. Ein Beispiel dazu ist in der folgenden Tabelle dargestellt. Die Tabelle enthält reale Daten über zwei verschiedene Arten von Schwertlilien (auch als "Iris" bekannt).

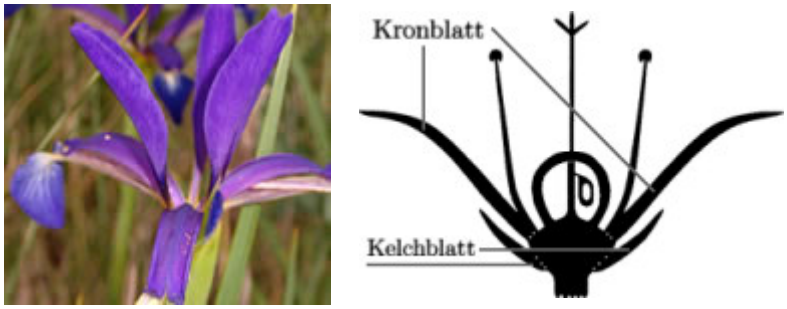

Abbildung 1.4. Schema einer Iris-Blüte

Jede Zeile entspricht einer Pflanze, die untersucht wurde. Die beiden Spalten auf der linken Seite enthalten die Breite und Höhe des Kelchblattes (das Muster) (s. Abb. 3) und die rechte Spalte die Art von Schwertlilie (die zugehörige Klasse), um die es sich dabei konkret handelte.

Kelchblatt Breite [cm]	Kelchblatt Höhe [cm]	Art
4.9	3.0	Iris-setosa
4.7	3.2	Iris-setosa
5.1	3.5	Iris-setosa
4.6	3.1	Iris-setosa
5.0	3.6	Iris-setosa
5.4	3.9	Iris-setosa
4.6	3.4	Iris-setosa
6.3	3.3	Iris-virginica
5.8	2.7	Iris-virginica
7.1	3.0	Iris-virginica
6.3	2.9	Iris-virginica
6.5	3.0	Iris-virginica
7.6	3.0	Iris-virginica
4.9	2.5	Iris-virginica
5.1	3.3	?
5.9	3.0	?
5.7	2.5	?
5.4	3.7	?

Wir werden in einer der Aufgabenstellungen ein Neuron darauf trainieren, eine Klassifikation dieser Daten zu *lernen* und die beiden Iris-Typen voneinander zu unterscheiden. Der Lernvorgang wird dabei so aussehen, dass wir unserem Neuron Beispiele von Eingaben präsentieren (Tupel aus Höhe und Breite des Kelchblattes) und dazu unsere *gewünschten* Ausgaben angeben (zu dieser Eingabe gehöriger Iris-Typ). Man bezeichnet dies als *beispielbasiertes Lernen*, eine spezielle Art eines *überwachten Lernverfahrens*. Durch einen einfachen Algorithmus wird das Neuron dann diese Zuordnung lernen können, indem die Neuronengewichte schrittweise so abgeändert werden, dass sich die Neuronenausgabe immer weiter der gewünschten Ausgabe annähert.

Verwenden werden wir dazu ein sogenanntes *Perzeptron*. Es handelt sich dabei um eines der ersten Künstlichen Neuronalen Netze, das sowohl über die Fähigkeit zu lernen, als auch über die Fähigkeit verfügt, zu *generalisieren*: Der Lernerfolg dieser Struktur geht über ein reines "Auswendiglernen" hinaus, so dass das Netz auch unbekannte Eingaben korrekt zuordnen kann, mit denen es *zuvor nicht* trainiert worden ist.
Die Struktur eines Perzeptrons ist recht simpel, siehe Abbildung 4 links. Es besteht aus einer "künstlichen Retina" E als Eingabe, einer Assoziationsschicht A und einer

Response-Schicht R. Die Neuronen in A besitzen dabei fest vorgegebene Verbindungen zu E, während die Gewichte der Verbindungen zwischen A und R veränderbar sind. Der Sinn der Assoziationsschicht besteht darin, Merkmale von E zu extrahieren, während Neuronen aus R dafür zuständig sind, die gewonnenen Informationen zu verarbeiten. Die Neuronen aus R summieren dabei ihre Eingaben gewichtet auf und bilden bei Überschreiten eines Schwellenwertes $s \in \mathbb{R}$ eine binäre Ausgabe von 1, sonst 0. Damit verhalten sich jedes dieser Neuronen wie dasjenige aus Abbildung 2.

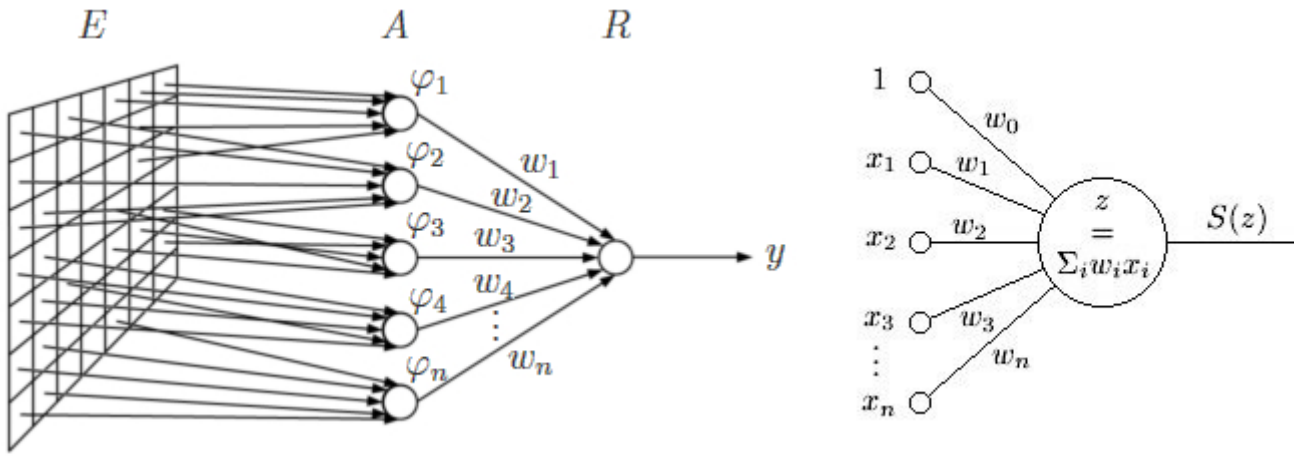

Abbildung 4. *Links:* Allgemeines Schema eines Perzeptrons. *Rechts:* Äquivalente Darstellung mit $x_i := \varphi_i(E)$ und $S(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$.

Es handelt sich um Abbildung 1.1 mit einer binären Ausgabefunktion. Der Einfachheit halber werden wir die Assoziationsschicht A daher vernachlässigen und die Eingabe des Perzeptrons direkt von Neuronen aus R verarbeiten lassen.

Bezeichnen wir mit $x_i := \varphi_i(E)$ die Ausgabe des i-ten Neurons aus A, so können wir - da die Verbindungen zwischen E und A fest und unveränderlich sind - ein Perzeptron auch vereinfacht als schon bekannte Neuronen aus Abbildung 4, rechts, auffassen und werden uns im Folgenden dabei nicht weiter um die Assoziationsschicht A kümmern. Da die Ausgabe eines Perzeptrons binär ist, verwenden wir aber zusätzlich noch $S(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$ als Ausgabefunktion unserer Neuronen.

Wie schon angesprochen lassen sich die Gewichte eines Perzeptrons lernen. Dies

geschieht folgendermaßen: Wir haben eine Reihe von Eingabemuster $\underline{x}_1, ..., \underline{x}_m$ die jeweils genau einer von zwei Mengen (Klassen) Ω_0 oder Ω_1 angehören. Außerdem gibt es eine Reihe von *Lehrervorgaben* $\ell_1, ..., \ell_m \in \{0, 1\}$, die zu einem Muster $\underline{x}$ Zugehörigkeit zur Klasse Ω_0 (bei $\ell_i = 0$) oder zu Ω_1 (bei $\ell_i = 1$) angeben. Bezeichnen wir mit den Gewichtsvektor eines Neurons aus R und mit $\underline{\tilde{x}}_i := [1, x_1^{(i)}, ..., x_n^{(i)}]^T$ den um 1 erweiterten Eingabevektor des Perzeptrons, dann erfolgt das Lernen von $\underline{w}$ mittels der Iterationsvorschrift:

$$\underline{w} \leftarrow \underline{w} + \gamma \cdot (\ell_i - y_i) \cdot \underline{\tilde{x}}_i, \quad y_i := S(\underline{w}^T \underline{\tilde{x}}_i)$$

Sind also errechnete Ausgabe und gewünschte Ausgabe gleich, so wird nichts gelernt, andernfalls schon. Man startet dabei meist mit einem zufällig gewählten Gewichtsvektor , dessen Komponenten betragsmäßig kleine reelle Zahlen sind. Bei Eingabe des *i*-ten Musters $\underline{x}_i$ stellt y_i die Ausgabe des Neurons dar, wobei γ ein Parameter ist, die sogenannte *Lernrate,* der die Größe der Veränderung beeinflußt. Sie ist eine Abwägung zwischen der Geschwindigkeit und der Genauigkeit des Lernvorganges und wird nach Erfahrung gewählt.

Die obige Iterationsvorschrift bezeichnet man als *Perzeptron-Lernregel* und wird solange wiederholt, bis zu allen Eingabemustern $\underline{x}_i$ die Neuronenausgabe y_i der gewünschten Lehrervorgabe ℓ_i möglichst gut entspricht oder es keine wesentliche Verbesserung mehr gibt. Da - wie wir oben gesehen haben - die Gewichte eines Neurons eine Hyperebene definieren, können wir den Lernprozess auch als das Lernen einer Hyperebene auffassen, die zwei Mengen (Klassen) trennt. Man kann zeigen, dass die Iteration mit der Perzeptron-Lernregel immer dann konvergiert, wenn sich die Eingabemuster in Ω_0 von denjenigen in Ω_1 durch eine Hyperebene voneinander trennen lassen. Man bezeichnet diese Eigenschaft als *lineare Separierbarkeit*.

Insbesondere kann ein Perzeptron mit der Perzeptron-Lernregel alle Funktionen lernen, die mit Hyperebenen beschrieben werden können.

nach oben