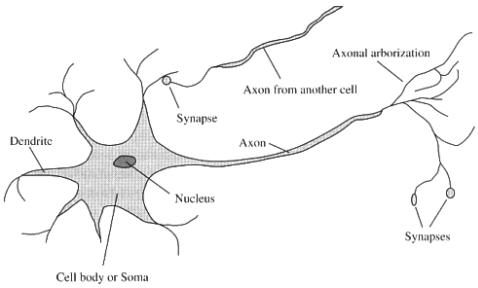
Biologische Motivation

Lernfähige biologische Systeme sind aus komplexen Netzwerken miteinander verbundener Neuronen aufgebaut.

Das Gehirn des Menschen besteht aus ca. 10^{11} Neuronen die jeweils mit ca. 10^4 anderen Neuronen verknüpft sind. Die Schaltzeit eines Neurons beträgt etwa 10^{-3}s (beim Computer $< 10^{-10}$s). Trotzdem sind komplexe Entscheidungen erstaunlich schnell möglich.

Beispiel: 10^{-1}s um ein Bild der eigenen Mutter zu erkennen. Die Abfolge der feuernden Neuronen kann folglich nicht länger als ein paar hundert Schritte sein. Die Verarbeitung und Repräsentation der Informationen muß also stark parallelisiert sein.

Das Neuron



Aus S. Russel, P. Norvig: Artificial Intelligence – A Modern Approach. S. 11, 2nd Ed., Prentice Hall, 2003.

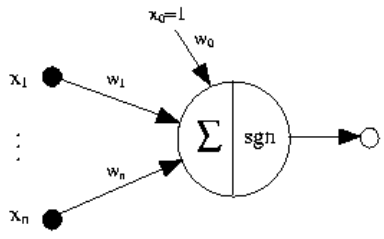
- gerichtete Reizleitung von den Dendriten zu den Synapsen
- Weiterleitung nur bei hinreichend starkem Eingangsreiz
- Lernen durch Strukturänderung (Verknüpfungen)

Charakterisierung passender Probleme

- Eingabe als Menge von Attribut-Wert-Paaren
- Ausgabe als (Vektor) diskreter oder reellwertiger Attribute
- Trainingsbeispiele dürfen Fehler enthalten
- lange Trainingszeiten sind akzeptabel
- schnelle Abarbeitung des gelernten Netzes im Einsatz
- menschliches Verständnis des gelernten Funktion ist nicht notwendig

Das Perceptron

Ein möglicher Knotentyp für den Aufbau Künstlicher Neuronaler Netze ist das **Perceptron**. Es errechnet die gewichtete Summe der reellwertigen Eingabedaten $x_1 \ldots x_n$ und gibt 1 zurück falls diese Summe über einem zuvor festgelegten Schwellwert liegt, sonst -1.



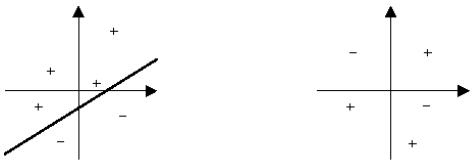
- $w_1 \ldots w_n$ sind die Kantengewichte
- Der Schwellwert ergibt sich aus $-w_0$, der zugehörige Eingabewert x_0 ist konstant 1
- Der Ausgabewert ergibt sich als $sgn(\vec{w} \cdot \vec{x}) = sgn(\sum_{i=0}^{n} w_i x_i)$
- $sgn(y) = \begin{cases} 1 \text{ , falls } y > 0 \\ -1 \text{ , sonst} \end{cases}$

Lineare Trennbarkeit

Ein einzelnes Perceptron kann als Hyperebene ($\vec{w} \cdot \vec{x} = 0$) betrachtet werden, die den n-dimensionalen Raum in zwei Bereiche aufteilt (Bewertung 1 bzw. -1).

Läßt sich eine Menge bewerteter Punkte im Raum durch eine Hyperebene korrekt in einen positiven und einen negativen Bereich aufteilen, so heißt sie **linear trennbar**. Nicht alle Mengen sind linear trennbar.

Beispiel: linear trennbar nicht linear trennbar



Darstellung boolscher Funktionen

Alle Boolschen Funktionen der Form „m von n", für die mindestens m von n Eingabewerten wahr (bzw. falsch) seien müssen, definieren linear trennbare Mengen und sind somit durch ein einzelnes Perceptron darstellbar.

Vorgehen:

- alle Gewichte erhalten den selben Wert
- der Schwellwert wird je nach Funktion angepaßt
- Negation von Eingabewerten durch Negieren des Gewichtes
- Interpretation: -1 falsch, 1 wahr

Beispiel: AND („2 von 2") OR („1 von 2")

Darstellung boolscher Funktionen

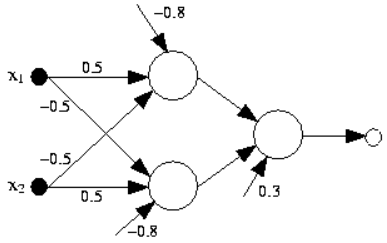
Es gibt einige Boolsche Funktionen, für die die Menge der Paare (Eingabewerte, Ausgabewert) nicht linear trennbar ist. Sie lassen sich somit nicht mit einem einzelnen Perceptron darstellen (z.B. XOR).

Jede beliebige boolsche Funktion kann aber mit einem 2-schichtigen Netz von Perceptronen dargestellt werden, weil:

- alle grundlegenden Funktionen (AND, OR, NOT) mittels Perceptron darstellbar sind
- sich jede boolsche Funktion in Distributiver Normalform darstellen läßt

Beispiel:

$\mathrm{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$



Lernen in Künstlichen Neuronalen Netzen

Problem: Für komplexere Zusammenhänge ist die zu wählende Struktur und Gewichtung des Netzes nicht offensichtlich

Idee: Anhand einer Menge von Trainingsdaten wird das Netz, ausgehend von einem initialen Zustand, so angepaßt, daß es auf den Trainingsdaten hinreichend genau arbeitet. Waren die Trainingsdaten repräsentativ für das gesamte Anwendungsgebiet, wird das Netz auch auf anderen Eingabewerten und sogar auf fehlerhaften Eingaben korrekt arbeiten.

Lernen in Künstlichen Neuronalen Netzen

Überwachtes Lernen: Die Ausgabe $\vec{o}$ des Netzes auf eine Eingabe $\vec{x}$ wird mit dem gewünschten Wert $\vec{t}$ verglichen. Die Gewichte werden relativ zum Fehler geändert

- Varianten: Auto Associator (Eingabe = Ausgabe); Pattern Associator (Paare von Eingabe / Ausgabe); Klassifikator (Muster als Eingabe, Klasse als Ausgabe)
- Lernverfahren: Perceptron-Regel, Delta-Regel, Back-Propagation Verfahren

Unüberwachtes Lernen: Es werden nur Eingaben $\vec{x}$ präsentiert, mit dem Ziel darin selbstständig Ähnlichkeiten/Zusammenhänge zu erkennen (Clusterbildung).

- Varianten: Regularity Detector
- Lernverfahren: Hebbsche Lernregel (Verbindungen zwischen gleichzeitig aktiven Neuronen stärken); Competitive Learning (Eingänge des am meisten aktivierten Neurons erhalten höhere Gewichte)

Lernverfahren für ein einzelnes Perceptron

1. Gewichte mit zufälligen Werten initialisieren
2. für jedes Trainingsbeispiel $(\vec{x}, t)$ die Perceptron-Regel zur Anpassung der Gewichte anwenden:

$$w_i = w_i + \triangle w_i \qquad \triangle w_i = \eta(t - o)x_i$$

(η: Lernrate, x_i: i-ter Wert im Eingabevektor, t: gewünschte Ausgabe, o: errechnete Ausgabe)

3. Schritt 2 wiederholen, bis alle Trainingsbeispiele korrekt klassifiziert werden

Wenn die Menge der Trainingsdaten linear trennbar ist und die Lernrate η nicht zu groß gewählt wurde, dann konvergiert das Verfahren in einer endlichen Zahl von Schritten gegen einen Gewichtsvektor der alle Trainingsbeispiele korrekt klassifiziert.

Verallgemeinerte Netzstruktur

Ein Neuronales Netz wird beschrieben durch:

- eine Menge U von Knoten u_i
- eine Menge gerichteter Verbindungen zwischen den Knoten mit Wichtung w_{ji} für die Verbindung von u_i nach u_j
- eine (im Allgemeinen einheitliche) Struktur der Knoten u_i mit:
  - Berechnung des Eingangswertes $net_i(t)$ zum Zeitpunkt t als gewichtete Summe der von den Vorgängern propagierten Werten
  - einer Funktion f_i zur Berechnung der Aktivierung $a_i(t)$
  - einer Funktion g_i zur Berechnung der Ausgabewerte $o_i(t)$

Ablauf der Berechnung

1. die Neuronen u_i senden zur Zeit t den Wert $o_i(t)$ an ihre Nachfolger
2. die Nachfolger u_j empfangen diese Werte zum Zeitpunkt $t+1$ und berechnen den Eingangswert:
$$net_j(t+1) = \sum_i w_{ji} o_i(t)$$
3. die Aktivierung für u_j ergibt sich als:
$$a_j(t+1) = f_j(a_j(t), net_j(t+1))$$
meist hängt die Aktivierung nur von $net_j(t+1)$
4. die Ausgabe von u_j ergibt sich als:
$$o_j(t+1) = g_j(a_j(t+1))$$

Einige Architekturvarianten

- **Schichtweiser Aufbau:** Die Neuronen sind in Schichten angeordnet, so daß jeweils aus einer Schicht nur Verbindungen zu Neuronen der der folgenden Schicht bestehen. Es wird unterschieden zwischen
  - Eingabeschicht/-knoten – ohne eigene Verrechnung; nur zum Verteilen der Eingabewerte
  - verdeckte Schicht(en)/Knoten – zur internen Repräsentation und Verrechnung
  - Ausgabeschicht/-knoten
- **Vorwärtsverkettete Netze:** (feed-forward networks) beliebige Anordnung der Knoten; alle Kanten sind in ihrer Orientierung von den Eingabeknoten zu den Ausgabeknoten gerichtet (d.h. es gibt keine Rückkopplungen)
- **rekurente Netze:** mit Rückkopplung – Ausgabe als Eingabe für einen „früheren" Knoten

Typische Knotenstrukturen

Bezeichnung	Aktivierung	Ausgabe
Perceptron	$f_i(x) = sgn(x)$	$g_i(x) = x$
Lineare Einheit	$f_i(x) = x$	$g_i(x) = x$
beschränkte Lineare Einheit	$f_i(x) = \begin{cases} -1, \text{ falls } x \leq -1 \\ x, \text{ falls } -1 < x < 1 \\ 1, \text{ falls } x \geq 1 \end{cases}$	$g_i(x) = x$
Sigmoid-Einheit	$f_i(x) = \frac{1}{1+e^{-x}}$	$g_i(x) = x$

Eigenschaften der Aktivierungsfunktionen

- Allgemeine Charakteristik
  - minimale Aktivierung bei geringen Eingangswerten
  - maximale Aktivierung bei groSen Eingangswerten
  - steiler Anstieg am Schwellwert
- **Perceptron:** nur diskrete Werte; nicht differenzierbar
- **Beschränkte Lineare Einheit:** nicht differenzierbar; kann nur lineare Zusammenhänge darstellen
- **Lineare Einheit:** unbeschränkter Wertebereich (z.B. für Ausgabeschicht sinnvoll); differenzierbar; kann nur lineare Zusammenhänge darstellen
- **Sigmoid Einheit:** differenzierbar; kann nicht-lineare Zusammenhänge darstellen.

Idee des Lernverfahrens: Gradientenabstieg

Der quadratische Fehler des Netzes auf den Trainingsdaten (Trainingsfehler) läßt sich wie folgt berechnen:

$$E = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Dabei ist t_d der gewünschte Wert im Beispiel d und o_d der errechnete Wert. Der Fehler hängt von den gewählten Gewichten $\vec{w}$ im Netz ab und läßt sich daher auch als Funktion $E(\vec{w})$ auffassen.

Mittels Gradientenabstieg soll im Raum der möglichen Gewichtsvektoren nun derjenige Gewichtsvektor gesucht werden, der auf den Trainingsdaten den geringsten Fehler verursacht. Es wird also das globale Minimum von $E(\vec{w})$ gesucht.

Die Delta-Regel

Die Gewichte für eine Lineare Einheit sollen gelernt werden.

Der Anstieg der Fehlerfunktion im Punkt $\vec{w}$:

$$\bigtriangledown E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

Die notwendige Anpassung $\triangle\vec{w}$ der Gewichte ergibt sich in Richtung des steilsten Abstiegs $-\bigtriangledown E(\vec{w})$ mit der Schrittweite η (Lernrate).

Für ein einzelnes Gewicht ergibt sich:

$$\triangle w_i = -\eta\frac{\partial E}{\partial w_i} \qquad \text{mit} \qquad \frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$$

Die Delta-Regel

Vorgehen:

1. alle Gewichte mit kleinen zufälligen Werten initialisieren
2. das Netz auf alle Trainingsbeispiele anwenden und dabei die notwendige Änderung der Gewichte bestimmen:
   $\triangle w_i = \eta \sum_{d\in D}(t_d - o_d)x_{id}$
3. alle Gewichte anpassen: $w_i = w_i + \triangle w_i$
4. die Schritte 2. und 3. wiederholen bis ein Zielkriterium erreicht ist

Dieses Verfahren konvergiert auch wenn die Trainingsmenge nicht linear trennbar ist. In so einem Fall wird die am besten passenden Annäherung des Zielkonzepts gefunden.