

6. Andere Paradigmen

Inhalt dieses Kapitels

6.1 Induktive Logik-Programmierung

6.2 Genetische Algorithmen

6.3 Neuronale Netze

6.1 Induktive Logik-Programmierung

Einführung [Muggleton & De Raedt 1994]

- Gegeben: Menge von Fakten in einer prädikatenlogischen Sprache 1. Stufe
- Gesucht: prädikatenlogische Regeln 1. Stufe, die in der Faktenmenge gelten
- Methode: Suche im Raum aller möglichen Regeln
- Abgrenzung zu Assoziationsregeln

dort: Regeln der Form $P(x) \wedge Q(x) \Rightarrow R(x)$

hier: z.B. Regeln der Form $Kurs(x) \wedge Hyperlink(x, y) \Rightarrow Professor(y)$

oder



$$\forall X \exists Y \exists Z P(x,y) \wedge P(x, z) \Rightarrow Q(y, z)$$

komplexe Regeln mit *mehreren Variablen*

6.1 Induktive Logik-Programmierung

Einführung

- Zusammenhang mit relationalen Datenbanken

n -stelliges Prädikat P \leftrightarrow Relation P mit Attributen A_1 bis A_n

Tupel (x_1, \dots, x_n) genau dann in P enthalten, wenn das Fakt $P(x_1, \dots, x_n)$ gilt

- Ausdruckskraft der Regelsprache

 Zusammenhänge zwischen *verschiedenen* Relationen einer Datenbank

- Integration von vorhandenem Hintergrundwissen (*Domain Knowledge*)

Formulierung von Hintergrundwissen ebenfalls in Prädikatenlogik 1. Stufe

6.1 Induktive Logik-Programmierung

Methode

- Initialisierung
 - einer Menge von Hypothesen
- Generierung weiterer Hypothesen
 - aus den vorhandenen
 - durch Anwendung von *induktiven Inferenzregeln*
 - Regeln zur Ableitung neuer aus bekannten Hypothesen
 -  z.B. Spezialisierung und Generalisierung

6.1 Induktive Logik-Programmierung

Methode

Initialisiere eine Menge von Hypothesen QH ;

repeat

Wähle eine Hypothese $H \in QH$ und lösche sie aus QH ;

Wähle Inferenzregeln r_1, \dots, r_k aus der Menge der gegebenen Inferenzregeln;

Wende die Inferenzregeln r_1, \dots, r_k auf H an, um neue Hypothesen H_1, \dots, H_k „abzuleiten“;

Füge H_1, \dots, H_k zu QH hinzu;

Filtere aus der Menge QH ungültige oder uninteressante Hypothesen wieder heraus;

until Stopkriterium(QH) erfüllt

6.1 Induktive Logik-Programmierung

Diskussion

- hohe Ausdruckskraft der Regeln



z.B. für Temporal Data Mining oder Web Mining

- einfache Integration von Hintergrundwissen
- Ineffizienz

Suchraum sehr groß

- Ansätze zur Verbesserung der Effizienz



Vorgabe von *Regelschemata*

ähnliche Monotonie-Bedingung wie für Frequent Itemsets

6.2 Genetische Algorithmen

Grundlagen [Bäck 1996]

- allgemeines Suchverfahren
- basierend auf dem Prinzip der biologischen Evolution
- *Individuum*:
 - potentielle Lösung eines Problems
- *Chromosom*:
 - Codierung eines Individuums durch einen (typischerweise binären) String
- *Gen*:
 - zusammenhängender Teilstring eines Chromosoms

6.2 Genetische Algorithmen

Grundlagen

- Suchmechanismus: Reproduktion von Individuen und Auswahl der besten
- zwei verschiedene Arten der Reproduktion:
 - *Kombination* von ausgewählten Individuen
 - zufällige Veränderung (*Mutation*) eines existierenden Individuums
- Auswahl der besten Individuen (*Selektion*)
 - Zielfunktion: Chromosomen \rightarrow Fitness-Werte
 - Fitness*: Maß für die Qualität einer Lösung
- Anwendung der Fitness
 - Wahrscheinlichkeit, daß ein Individuum sich reproduziert
 - endgültige Auswahl der besten Lösung

6.2 Genetische Algorithmen

Beispiel

- Data-Mining-Problem:

Suche nach besonders häufigen Attribut-Wert-Paaren

- Gen: Attribut-Wert-Paar
- Chromosom: Konjunktion von Attribut-Wert-Paaren
- Kombination: Mischen der Attribut-Wert-Paare zweier Chromosomen
- Mutation: zufällige Veränderung eines Attributwertes
- Fitness-Wert:
Häufigkeit, mit der diese Konjunktion der Attribut-Werte-Paare in der Datenmenge auftritt

6.2 Genetische Algorithmen

Methode

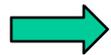
```
Initialisiere eine Population von Individuen;  
while (Stopkriterium ist nicht erfüllt) do  
    Wähle Individuen gemäß ihrer Fitness als Eltern  
    aus;  
    Kombiniere Eltern, um neue Individuen zu  
    erzeugen;  
    Mutiere die neuen Individuen;  
    Füge die neuen Individuen zur Population hinzu;  
return beste Individuen;
```

6.2 Genetische Algorithmen

Diskussion

- besser als zufällige / erschöpfende Suche

maximale Fitness in einer Population wächst monoton



im Laufe der Generationen

- im allgemeinen nicht sehr effizient

langsame Konvergenz

- Einsatz sinnvoll wenn

keine intelligente, problemspezifische Suchstrategie bekannt



Qualität einer Gesamtlösung = „Summe“ der Qualitäten der Teillösungen

6.3 Neuronale Netze

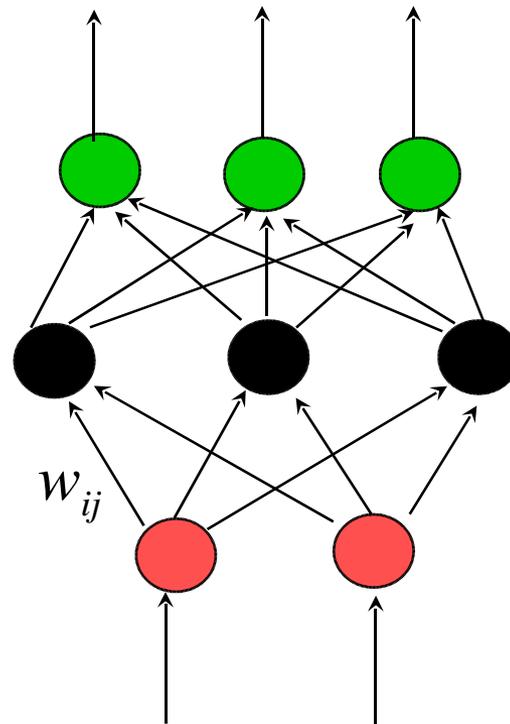
Grundlagen [Bigus 1996], [Bishop 1995]

- Paradigma für ein Maschinen- und Berechnungsmodell
- Funktionsweise ähnlich der von biologischen Gehirnen
- *Neuronales Netz*: Menge von Neuronen, über Kanten miteinander verbunden
- *Neuron*: entspricht biologischem Neuron
 - Aktivierung durch Input-Signale an den Synapsen
 - Erzeugung eines Output-Signals, das zu anderen Neuronen weitergeleitet wird
- Organisation eines neuronalen Netzes
 - Input-Schicht, verborgene Schichten, Output-Schicht*
 - Knoten einer Schicht mit allen Knoten der vorhergehenden Schicht verbunden

6.3 Neuronale Netze

Grundlagen

- Kanten besitzen *Gewichte*
- Funktion eines neuronalen Netzes



Output-Vektor y

Output-Schicht

verborgene Schicht

Input-Schicht

Input-Vektor x

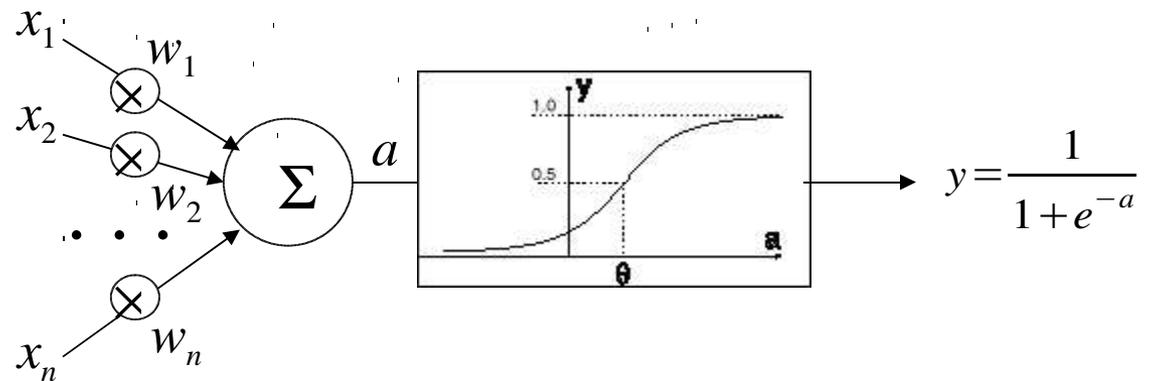
6.3 Neuronale Netze

Neuronen

- allgemeines Neuron

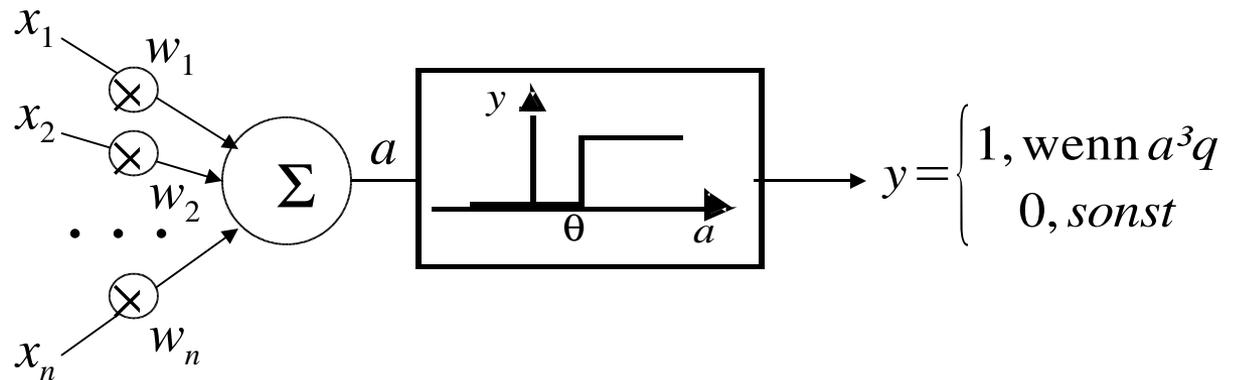
a : Aktivierungswert

$$a = \sum_{i=1}^n w_i \cdot x_i$$



- *Threshold Logic Unit*

(TLU)



6.3 Neuronale Netze

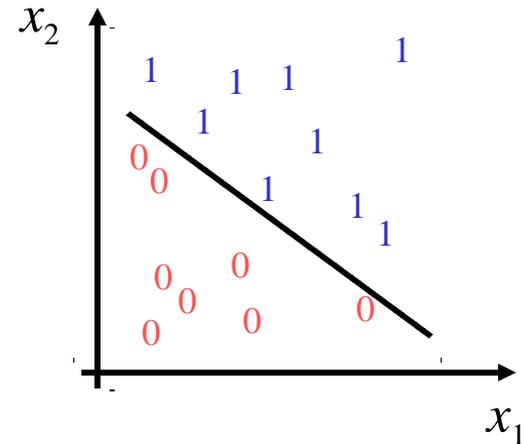
Neuronen

- Klassifikation mit Hilfe einer TLU

repräsentiert eine (Hyper-)Ebene $\sum_{i=1}^n w_i \cdot x_i = q$

links von der Ebene: Klasse 0

rechts von der Ebene: Klasse 1



- Trainieren einer TLU

Lernen der „richtigen“ Gewichte zur Unterscheidung der zwei Klassen
Iterative Anpassung der Gewichte w_{ij}

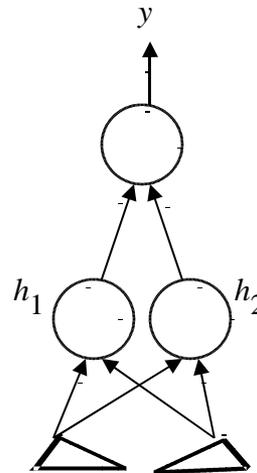
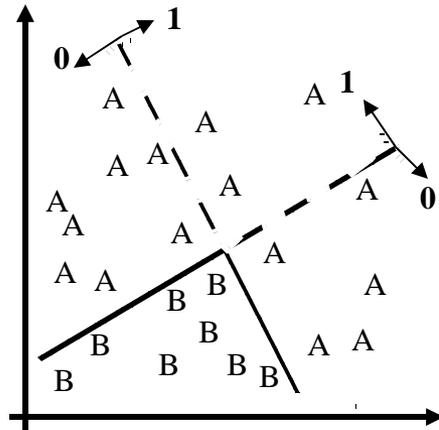
Rotation der durch w und θ gegebene Hyperebene um einen kleinen Betrag
in Richtung v , wenn v noch nicht auf der richtigen Seite der Ebene liegt

6.3 Neuronale Netze

Kombination mehrerer Neuronen

- zwei Klassen, die nicht linear separierbar sind:
➔ zwei innere Knoten und ein Output-Knoten

- Beispiel



$h_1 = 0 \wedge h_2 = 0 : y = 0$ (Klasse B)

andernfalls : $y = 1$ (Klasse A)

6.3 Neuronale Netze

Lernalgorithmus für komplexe Neuronale Netze

- bei Abweichung von vorhergesagter und tatsächlicher Klasse:
Anpassung der Gewichte mehrerer Knoten
- Frage
in welchem Maße sind die verschiedenen Knoten an dem Fehler beteiligt?
- Anpassung der Gewichte
durch Gradientenverfahren, das den Gesamtfehler minimiert
Gesamtfehler: Summe der (quadratischen) Abweichungen des tatsächlichen Outputs y vom gewünschten Output t für die Menge der Inputvektoren
Voraussetzung: Output y *stetige* Funktion der Aktivierung a

6.3 Neuronale Netze

Algorithmus Backpropagation

für jedes Paar(v, t) // v = Input, t = gewünschter Output

„*forward pass*“:

Bestimme den tatsächlichen Output y für Eingabe v ;

„*backpropagation*“:

Bestimme den Fehler $(t - y)$ der Output-Einheiten
und passe die Gewichte der Output-Einheiten in die
Richtung an, die den Fehler minimiert;

Solange der Input-Layer nicht erreicht ist:

Propagiere den Fehler auf die nächste Schicht
und passe auch dort die Gewichte der
Einheiten in fehlerminimierender Weise an;

6.3 Neuronale Netze

Design der Netztopologie

- Bestimmung von
 - Anzahl der Input-Knoten
 - Anzahl der inneren Schichten und jeweilige Anzahl der Knoten
 - Anzahl der Output-Knoten



starker Einfluß auf die Klassifikationsgüte

- zu wenige Knoten
 - ➔ niedrige Klassifikationsgüte
- zu viele Knoten
 - ➔ Overfitting

6.3 Neuronale Netze

Bestimmung der Netztopologie [SPSS Clementine 2000]

- Statische Topologie

Topologie wird apriori festgelegt

eine verborgene Schicht reicht in vielen Anwendungen aus

- Dynamische Topologie

dynamisches Hinzufügen von Neuronen (und verborgenen Schichten)

solange Klassifikationsgüte signifikant verbessert wird

- Multiple Topologien

Trainieren mehrerer dynamischer Netze parallel

z.B. je ein Netz mit 1, 2 und 3 verborgenen Schichten

6.3 Neuronale Netze

Bestimmung der Netztopologie

- Pruning

Trainieren eines Netzes mit statischer Topologie

nachträgliches Entfernen der unwichtigsten Neuronen

solange Klassifikationsgüte verbessert wird

Schlußfolgerung



statische Topologie: niedrige Klassifikationsgüte, aber relativ schnell

Pruning: beste Klassifikationsgüte, aber sehr hoher Laufzeitaufwand zum

Training

6.3 Neuronale Netze

Diskussion

- + im allgemeinen sehr hohe Klassifikationsgüte
 - beliebig komplexe Entscheidungsflächen
- + robust gegen Rauschen in den Trainingsdaten
- + Effizienz der Anwendung

- schlechte Verständlichkeit
 - lernt nur Gewichte, aber keine Klassenbeschreibung
- Ineffizienz des Lernens
 - sehr lange Trainingszeiten
- keine Integration von Hintergrundwissen

