

## 7. Andere Paradigmen

### *Inhalt dieses Kapitels*

7.1 Induktive Logik-Programmierung

7.2 Genetische Algorithmen

7.3 Neuronale Netze

## 7.1 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

## 7.1 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*

## 7.1 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

## 7.1 Induktive Logik-Programmierung

### Methoden

```
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
```

Vorlesung Knowledge Discovery

5

## 7.1 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

Vorlesung Knowledge Discovery

6

## 7.2 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

Vorlesung Knowledge Discovery

7

## 7.2 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

Vorlesung Knowledge Discovery

8

## 7.2 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

## 7.2 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;
```

## 7.2 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

## 7.3 Neuronale Netze

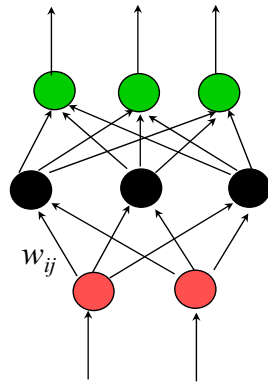
### Grundlagen [Bigus 1996], [Bishop 1995]

- 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

## 7.3 Neuronale Netze

### Grundlagen

- Kanten besitzen *Gewichte*
- Funktion eines neuronalen Netzes



Output-Vektor  $y$

Output-Schicht

verborgene Schicht

Input-Schicht

Input-Vektor  $x$

## 7.3 Neuronale Netze

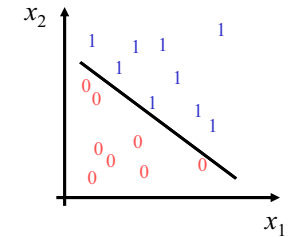
### Neuronen

- Klassifikation mit Hilfe einer TLU

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

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  $\theta$  gegebene Hyperebene um einen kleinen Betrag in Richtung  $v$ , wenn  $v$  noch nicht auf der richtigen Seite der Ebene liegt

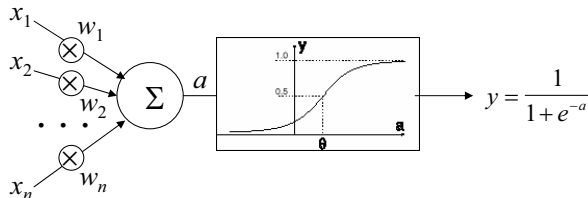
## 7.3 Neuronale Netze

### Neuronen

- allgemeines Neuron

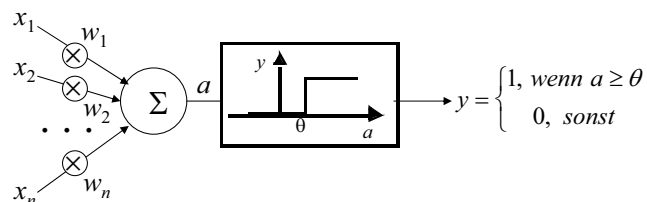
$a$ : Aktivierungswert

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



- *Threshold Logic Unit (TLU)*

(TLU)



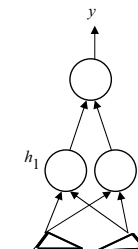
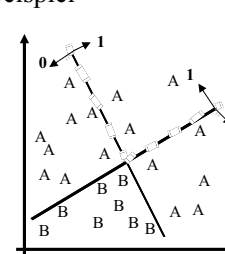
## 7.3 Neuronale Netze

### Kombination mehrerer Neuronen

- zwei Klassen, die nicht linear separierbar sind:

➡ zwei innere Knoten und ein Output-Knoten

- Beispiel



Wenn  $h_1 = h_2 = 0$ , dann  $y = 0$  (Klasse A)  
sonst  $y = 1$  (Klasse B)




## 7.3 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  $a$

## 7.3 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

## 7.3 Neuronale Netze

### *Algorithmus Backpropagation*

**für jedes** Paar  $(v, t)$  //  $v = \text{Input}, t = \text{gewünschter Output}$   
„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;

## 7.3 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

## 7.3 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

## 7.3 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

