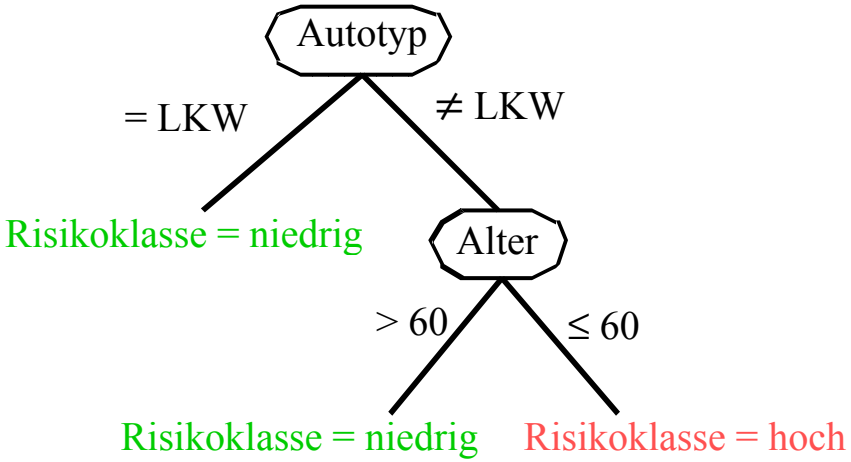
3.5 Entscheidungsbaum-Klassifikatoren

Motivation

ID	Alter	Autotyp	Risiko
1	23	Familie	hoch
2	17	Sport	hoch
3	43	Sport	hoch
4	68	Familie	niedrig
5	32	LKW	niedrig

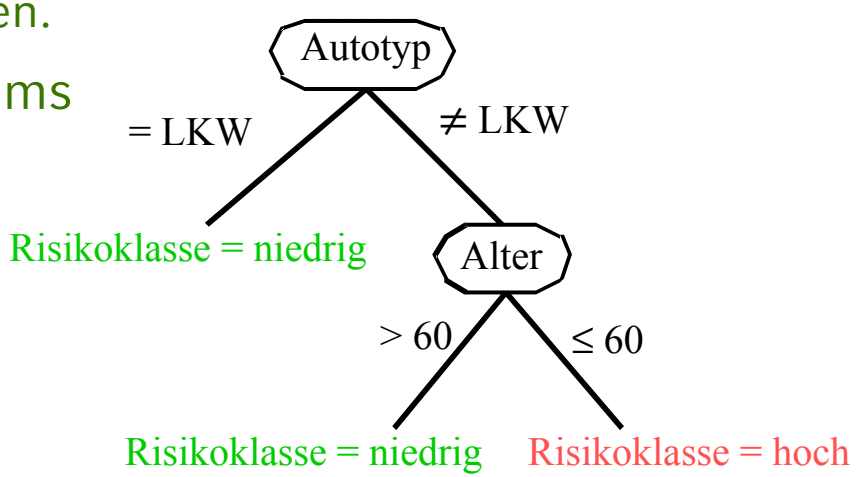


finden explizites Wissen

Entscheidungsbäume sind für die meisten Benutzer verständlich

Grundbegriffe

- Ein *Entscheidungsbaum* ist ein Baum mit folgenden Eigenschaften:
  - ein innerer Knoten repräsentiert ein Attribut,
  - eine Kante repräsentiert einen Test auf dem Attribut des Vaterknotens,
  - ein Blatt repräsentiert eine der Klassen.
- Konstruktion eines Entscheidungsbaums
  - anhand der Trainingsmenge
  - Top-Down



- Anwendung eines Entscheidungsbaums

  Durchlauf des Entscheidungsbaum von der Wurzel zu einem der Blätter

  $\Rightarrow$ eindeutiger Pfad

  Zuordnung des Objekts zur Klasse des erreichten Blatts

Konstruktion eines Entscheidungsbaums

Basis-Algorithmus

- Anfangs gehören alle Trainingsdatensätze zur Wurzel.
- Das nächste Attribut wird ausgewählt (Splitstrategie).
- Die Trainingsdatensätze werden unter Nutzung des Splitattributs partitioniert.
- Das Verfahren wird rekursiv für die Partitionen fortgesetzt.

  $\Rightarrow$ lokal optimierender Algorithmus

Abbruchbedingungen

- keine weiteren Splitattribute
- alle Trainingsdatensätze eines Knotens gehören zur selben Klasse

Entscheidungsbaum-Klassifikatoren

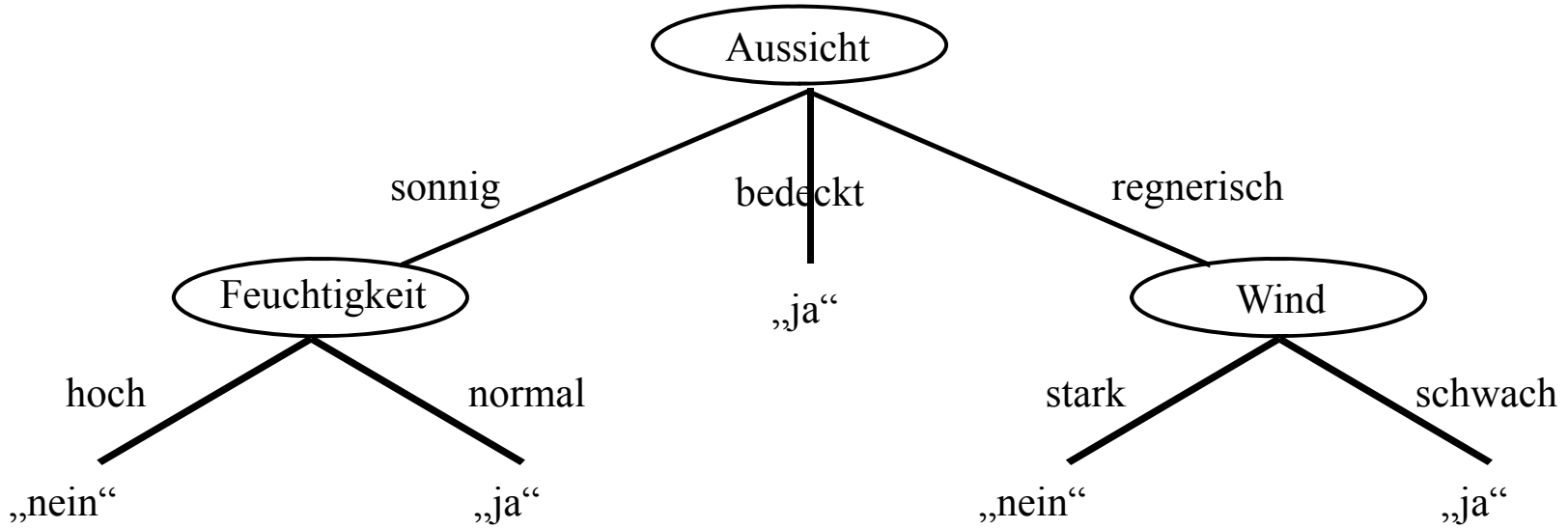
Beispiel

Tag	Aussicht	Temperatur	Feuchtigkeit	Wind	Tennispielen
1	sonnig	heiß	hoch	schwach	nein
2	sonnig	heiß	hoch	stark	nein
3	bedeckt	heiß	hoch	schwach	ja
4	regnerisch	mild	hoch	schwach	ja
5	regnerisch	kühl	normal	schwach	ja
6	regnerisch	kühl	normal	stark	nein
7	bedeckt	kühl	normal	stark	ja
8	sonnig	mild	hoch	schwach	nein
9	sonnig	kühl	normal	schwach	ja
10	regnerisch	mild	normal	schwach	ja
11	sonnig	mild	normal	stark	ja
12	bedeckt	mild	hoch	stark	ja
13	bedeckt	heiß	normal	schwach	ja
14	regnerisch	mild	hoch	stark	nein

Ist heute ein Tag zum Tennisspielen?

Entscheidungsbaum-Klassifikatoren

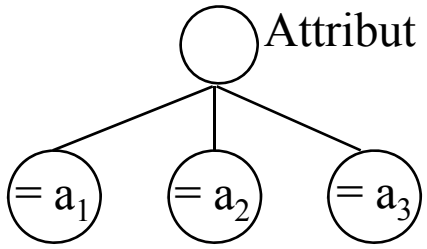
Beispiel

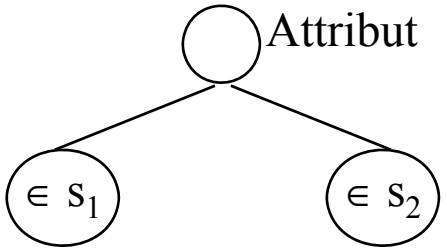


Splitstrategien

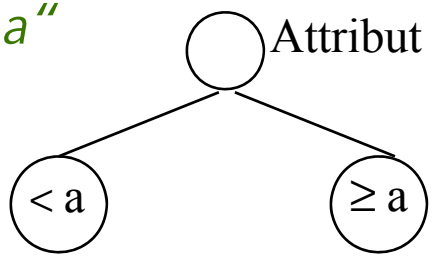
Typen von Splits

- Kategorische Attribute
  Splitbedingungen der Form „*Attribut = a*“ or „*Attribut ∈ set*“
  viele mögliche Teilmengen





- Numerische Attribute
  Splitbedingungen der Form „*Attribut < a*“
  viele mögliche Splitpunkte



Numerische Splitgrenzen

Wo sollen diskrete Attribute gesplittet werden?

=> An den Stellen, die die Qualität maximieren.

Idee: Ordnen der numerischen Attributwerte

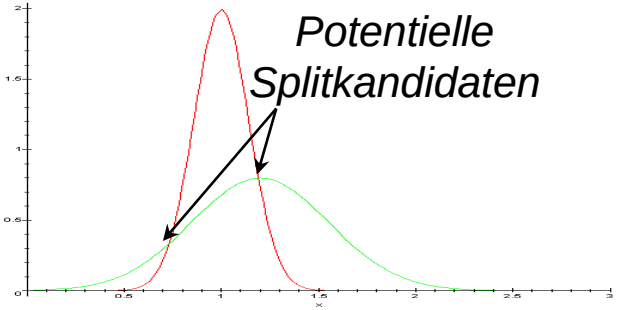
Wert	0.9	0.8	0.65	0.5	0.45	0.3	0.15	0.01
Klasse	A	A	B	B	B	A	A	A

Potentielle Splitkandidaten

Teste die Kombination, die den höchsten Information Gain erzielen.

Schnellere Methode:

- Bilde Gauß-Kurve über alle Klassen
- Wähle Schnittpunkte der Gauß-Kurven als Kandidaten.



Splitstrategien

Qualitätsmaße für Splits

Gegeben

- eine Menge T von Trainingsobjekten
- eine disjunkte, vollständige Partitionierung $T_1, T_2, \ldots, T_m$ von T
- p_i die relative Häufigkeit der Klasse c_i in T

Gesucht

- ein Maß der *Unreinheit* einer Menge S von Trainingsobjekten in Bezug auf die Klassenzugehörigkeit
- ein Split von T in $T_1, T_2, \ldots, T_m$, der dieses Maß der Unreinheit *minimiert*

  $\Rightarrow$ Informationsgewinn, Gini-Index

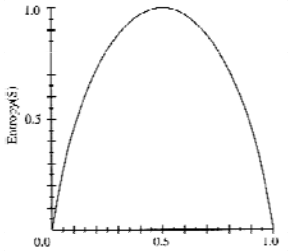
Splitstrategien

Informationsgewinn

Entropie: minimale Anzahl von Bits zum Codieren der Nachricht, mit der man die Klasse eines zufälligen Trainingsobjekts mitteilen möchte.

Die *Entropie* für eine Menge T von Trainingsobjekten ist definiert als

$$entropie(T) = -\sum_{i=1}^{k} p_i \cdot \log p_i$$



$entropie(T) = 0$, falls $p_i = 1$ für ein i

$entropie(T) = 1$ für $k = 2$ Klassen mit $p_i = 1/2$

Das Attribut A habe die Partitionierung $T_1, T_2, \ldots, T_m$ erzeugt.

Der *Informationsgewinn* des Attributs A in Bezug auf T ist definiert als

$$Informationsgewinn(T, A) = entropie(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropie(T_i)$$

Splitstrategien

Gini-Index

Gini-Index für eine Menge *T* von Trainingsobjekten

$$gini(T) = 1 - \sum_{j=1}^{k} p_j^2$$

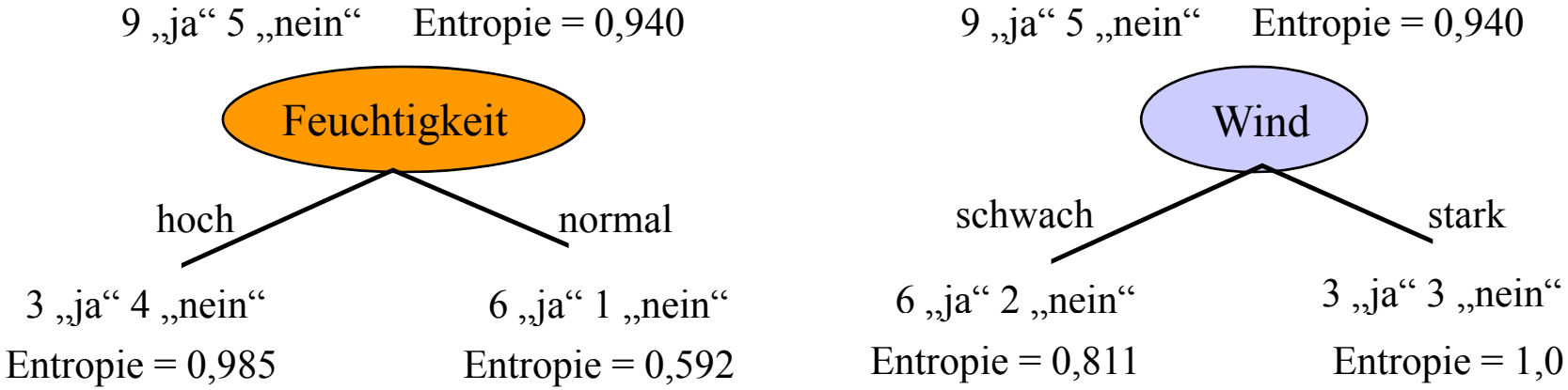
- kleiner Gini-Index ⇔ geringe Unreinheit,
- großer Gini-Index ⇔ hohe Unreinheit

Das Attribut *A* habe die Partitionierung $T_1, T_2, \ldots, T_m$ erzeugt.
Gini-Index des Attributs *A* in Bezug auf *T* ist definiert als

$$gini_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i)$$

Splitstrategien

Beispiel

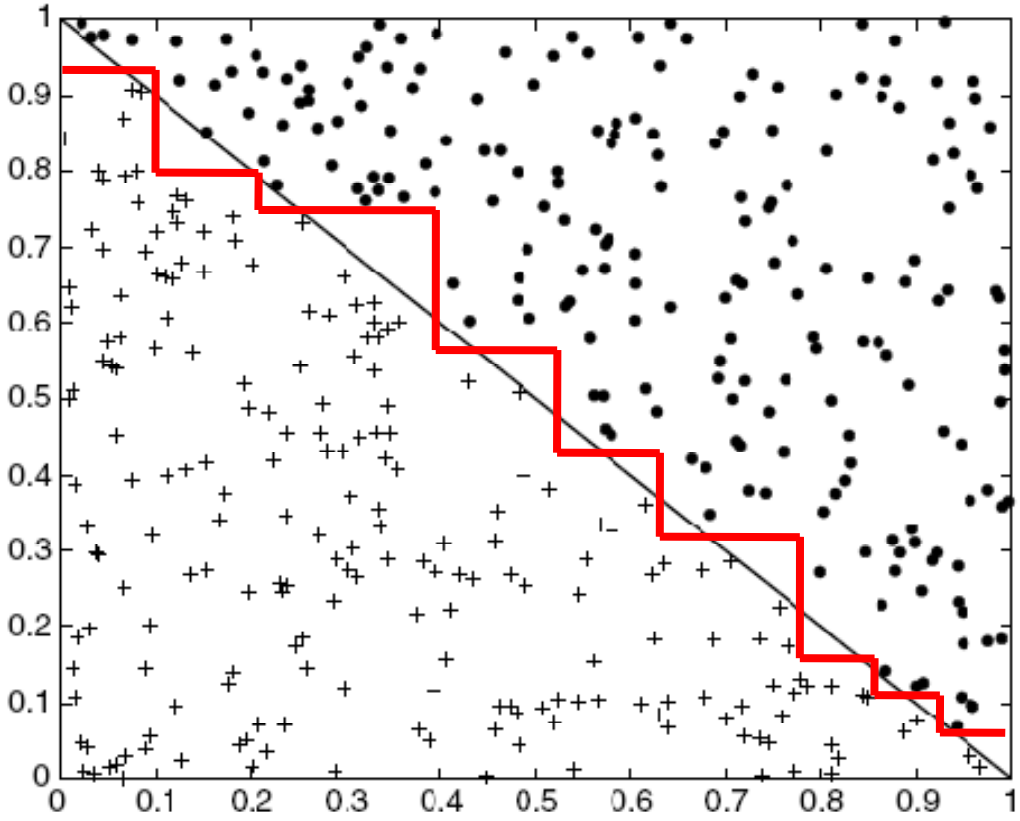


$$Informationsgewinn(T, Feuchtigkeit) = 0{,}94 - \frac{7}{14} \cdot 0{,}985 - \frac{7}{14} \cdot 0{,}592 = 0{,}151$$

$$Informationsgewinn(T, Wind) = 0{,}94 - \frac{8}{14} \cdot 0{,}811 - \frac{6}{14} \cdot 1{,}0 = 0{,}048$$

$\Rightarrow$ Feuchtigkeit liefert den höheren Informationsgewinn

Bias

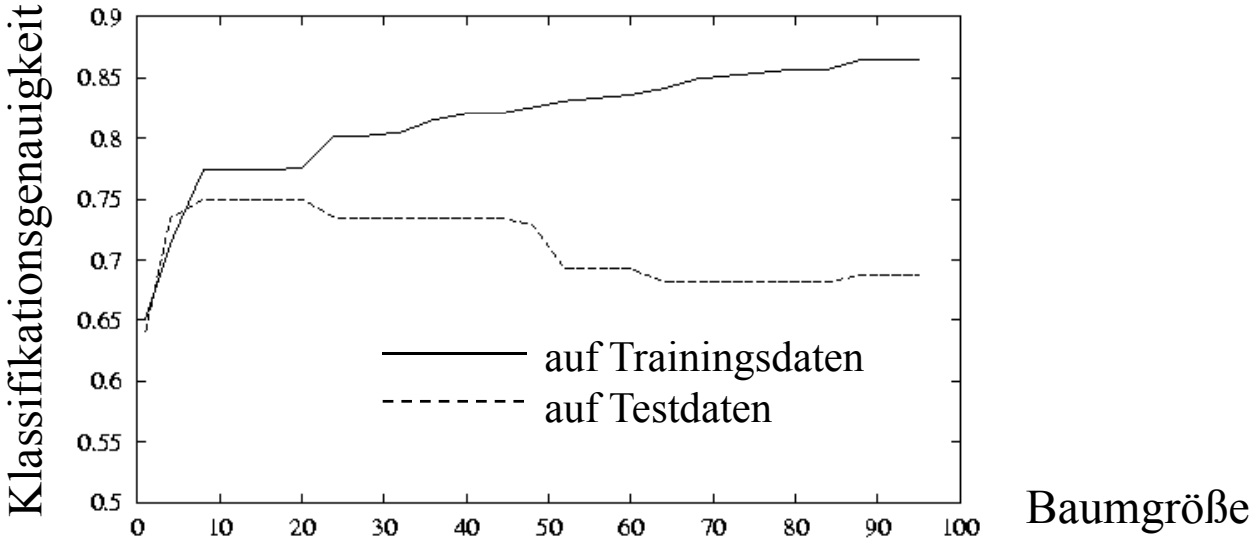


Overfitting

Einführung

Overfitting bei der Konstruktion eines Entscheidungsbaums, wenn es zwei Entscheidungsbäume E und E' gibt mit

- E hat auf der Trainingsmenge eine kleinere Fehlerrate als E',
- E' hat auf der Grundgesamtheit der Daten eine kleinere Fehlerrate als E.



Overfitting

Trainingsdaten

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

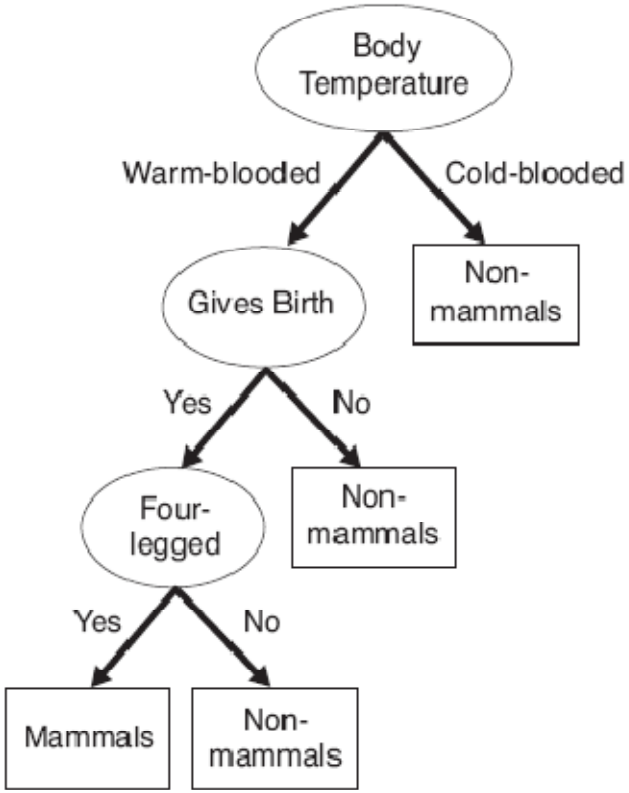
*Fehlerhafte Daten

Overfitting

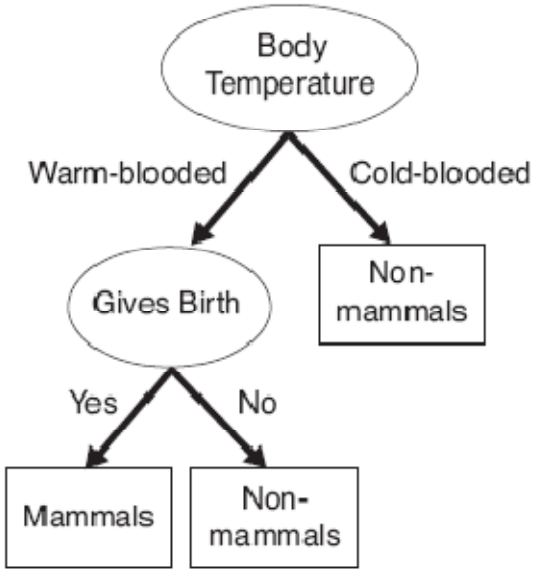
Testdaten

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	warm-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

Overfitting



(a) Model M1



(b) Model M2

Trainingserror: 0%
Testerror: 30%

Trainingserror: 20%
Testerror: 10%

M1:

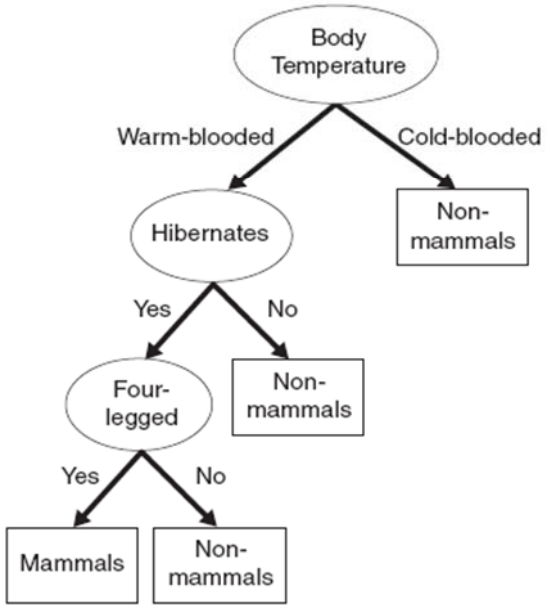
- Fehlklassifikation von human und dolphin aufgrund von fehlerhaften Trainingsdaten
- Spiny anteater: ungewöhnlicher Fall

M2:

ungewöhnlicher Fall kann nicht vermieden werden

Overfitting

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
salamander	cold-blooded	no	yes	yes	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no



Problem:

Mangel an repräsentativen Daten

Overfitting

Ansätze zum Vermeiden von Overfitting

- Entfernen von fehlerhaften Trainingsdaten

  insbesondere widersprüchliche Trainingsdaten

- Wahl einer geeigneten Größe der Trainingsmenge

  nicht zu klein, nicht zu groß

- Wahl einer geeigneten Größe des minimum support

  minimum support:

  Anzahl der Datensätze, die mindestens zu einem Blattknoten des Baums gehören müssen

  minimum support >> 1

Overfitting

Ansätze zum Vermeiden von Overfitting

- Wahl einer geeigneten Größe der minimum confidence

  minimum confidence: Anteil, den die Mehrheitsklasse eines Blattknotens mindestens besitzen muß.

  minimum confidence << 100%

  Blätter können auch fehlerhafte Datensätze oder Rauschen „absorbieren“

- nachträgliches Pruning des Entscheidungsbaums

  Abschneiden der überspezialisierten Äste

  kleinere Bäume sind weniger anfällig für Overfitting (Occam’s Razor)

Pruning von Entscheidungsbäumen

Fehlerreduktions-Pruning [Mitchell 1997]

- Aufteilung der klassifizierten Daten in Trainingsmenge und Testmenge
- Konstruktion eines Entscheidungsbaums *E* für die Trainingsmenge
- Pruning von *E* mit Hilfe der Testmenge *T*
  - bestimme denjenigen Teilbaum von *E*, dessen Abschneiden den Klassifikationsfehler auf *T* am stärksten reduziert
  - entferne diesen Teilbaum
  - fertig, falls kein solcher Teilbaum mehr existiert

nur anwendbar, wenn genügend viele klassifizierte Daten

Entscheidungsbaum-Klassifikatoren

Diskussion

+ Interpretation des gefundenen Baumes relativ einfach
+ Implizite Gewichtung der Attribute
+ Leistungsfähiger Klassifikator, häufig in der Praxis verwendet
+ Effiziente Auswertung des gefundenen Modells

- Finden eines optimalen Entscheidungsbaums ist exponentiell
- Heuristische Methoden können nur lokales Optimum finden
- Anfällig für Overfitting (besondere Methoden zur Vermeidung von Overfitting für Entscheidungsbäume)

3.6 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
- Menschliches Gehirn: 10^{11} Neuronen, jedes verbunden mit ~10^4 anderen, schnellste Aktionszeit 10^{-3} Sek. (Computer: 10^{-10}) – aber komplexe Entscheidungen sehr schnell (visuelle Erkennung der eigenen Mutter: 10^{-1} Sek. ≈ 100 Schritte)
- Erzeugung eines Output-Signals, das zu anderen Neuronen weitergeleitet wird.
- Organisation eines neuronalen Netzes

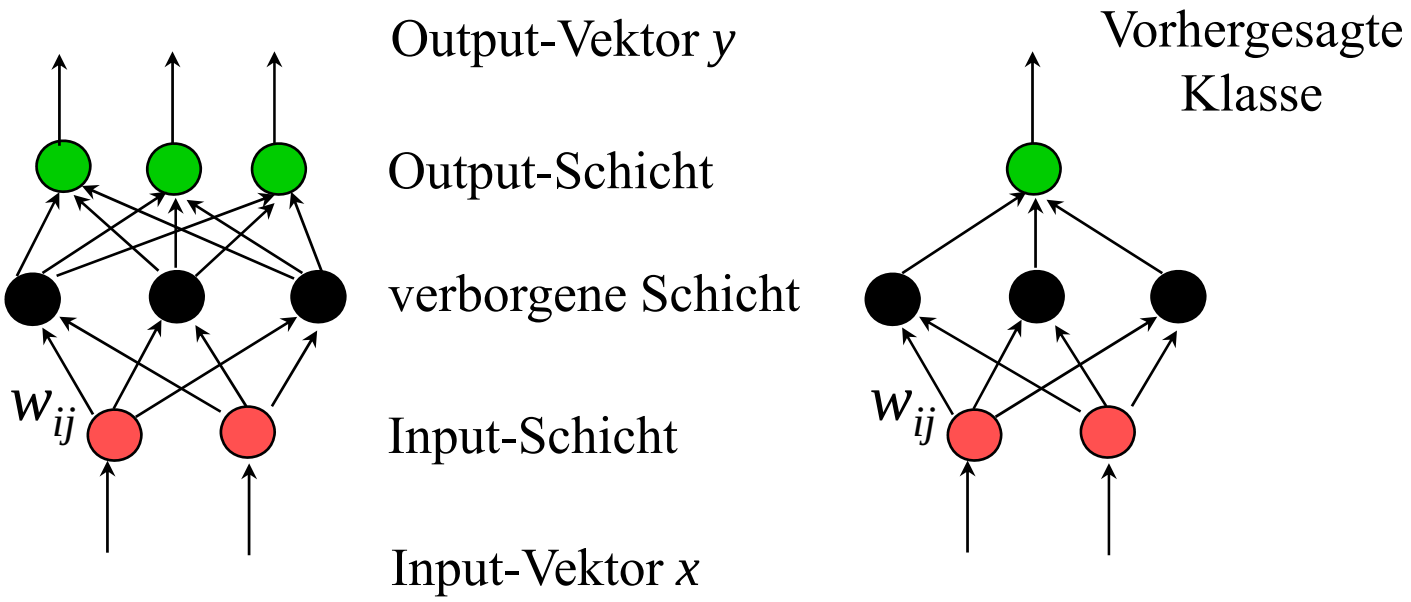
  Input-Schicht, *verborgene Schichten*, *Output-Schicht*

  Knoten einer Schicht mit allen Knoten der vorhergehenden Schicht verbunden

Grundlagen

Kanten besitzen *Gewichte*

Funktion eines neuronalen Netzes

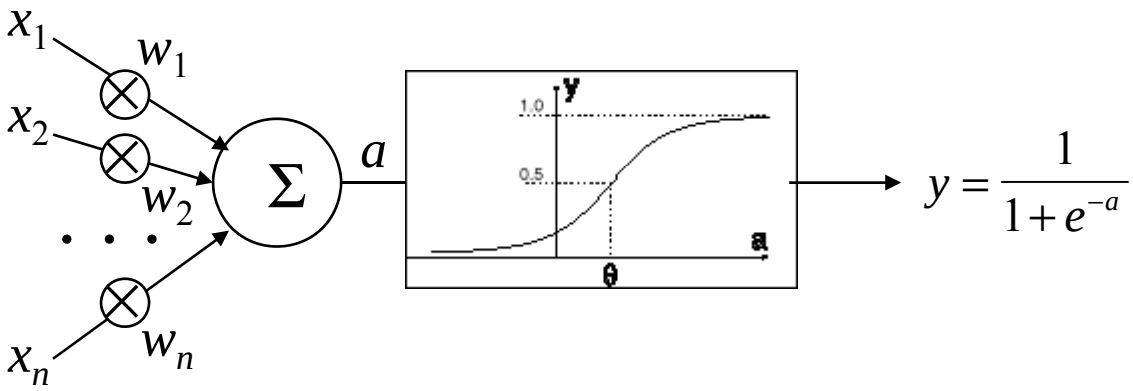


Neuronen

allgemeines Neuron (auch: Perzeptron)

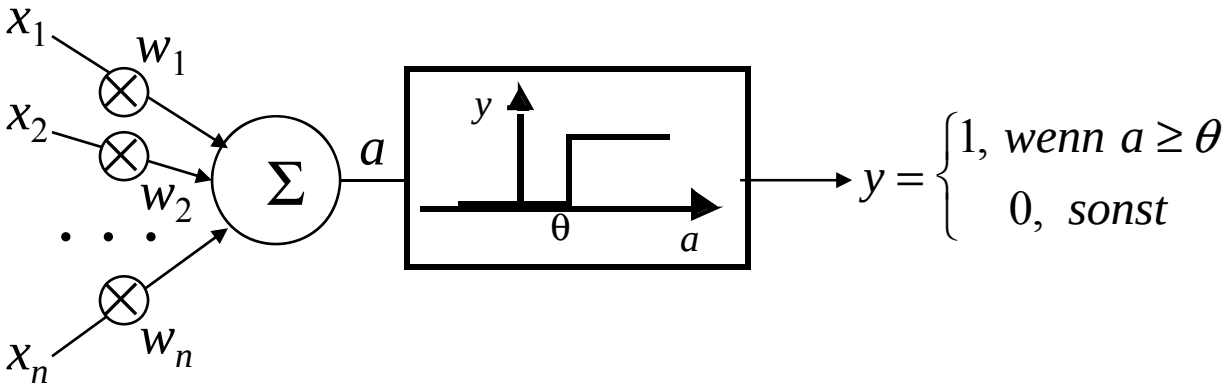
a: *Aktivierungswert*

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



Threshold Logic Unit

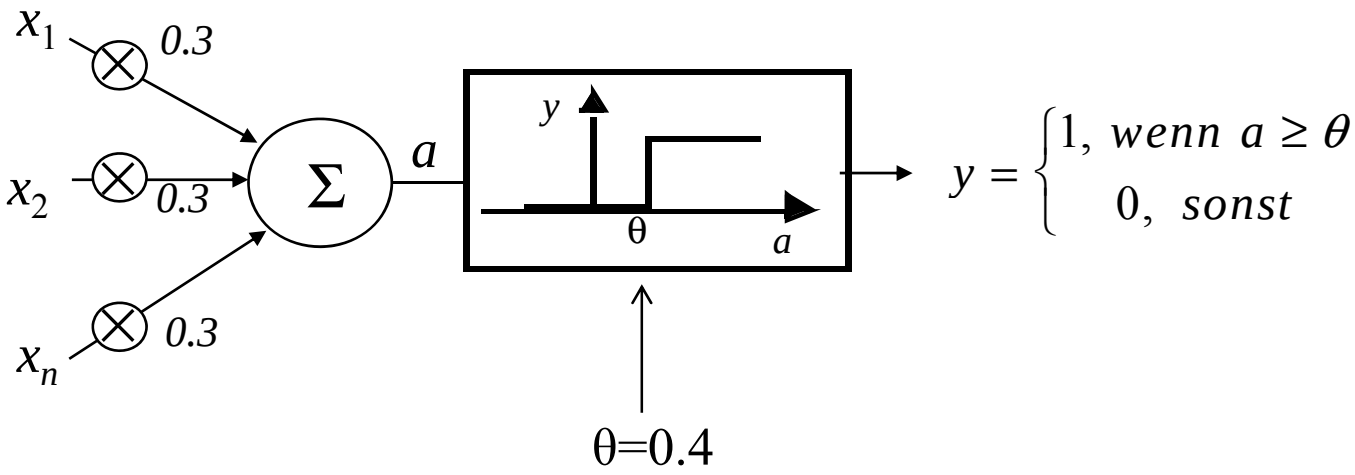
(*TLU*)



Neuronen

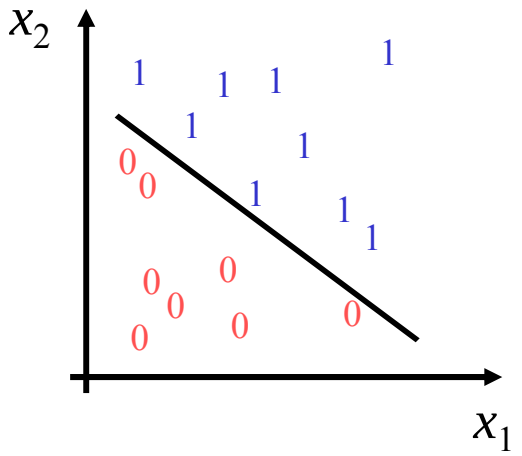
Beispiel: Modellierung einer booleschen Funktion

x_1	x_2	x_3	y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



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

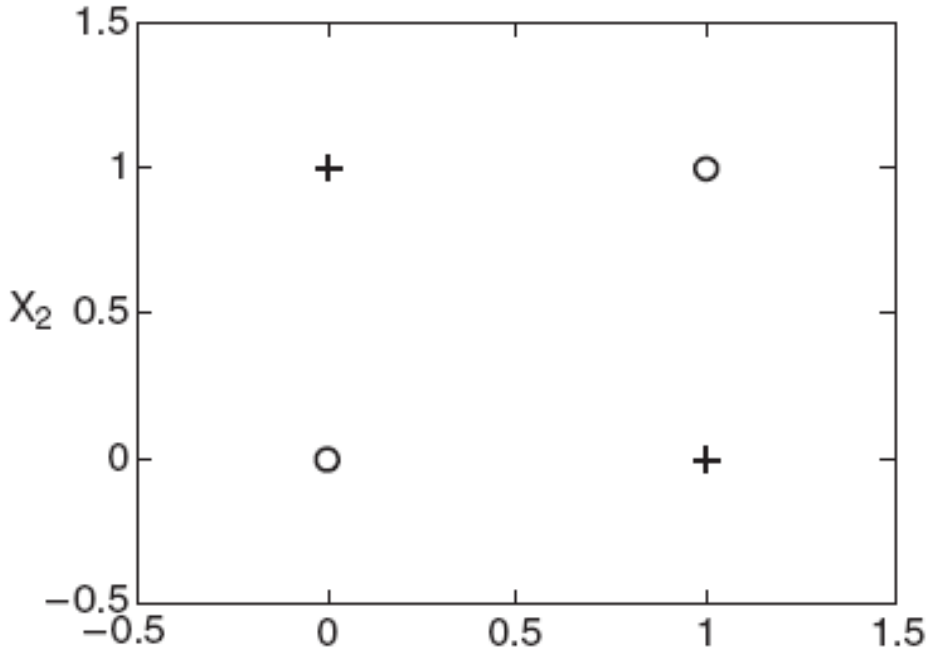


- Trainieren einer TLU
  - Lernen der „richtigen“ Gewichte zur Unterscheidung der zwei Klassen
    Iterative Anpassung der Gewichte w_{ij}
  - Rotation der durch w und θ gegebene Hyperebene um einen kleinen Betrag in Richtung v, wenn v noch nicht auf der richtigen Seite der Ebene liegt

XOR-Problem

nicht linear separierbares Problem

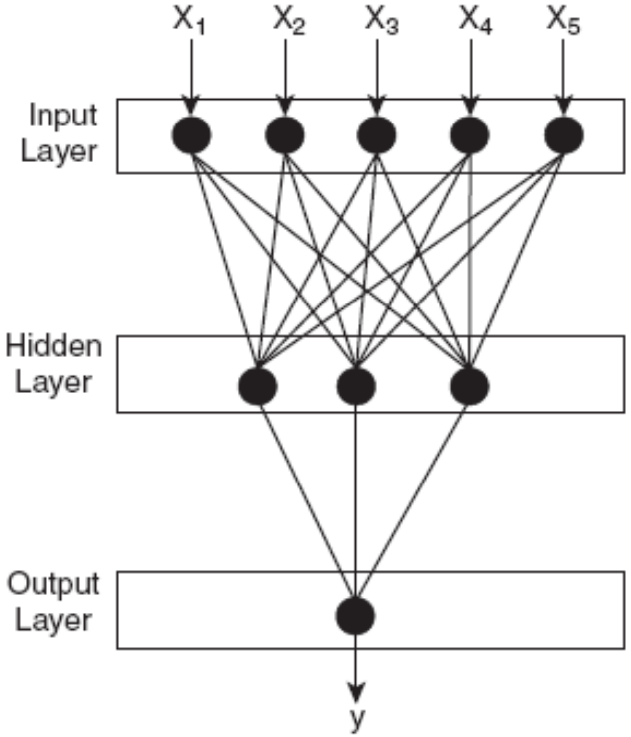
X_1	X_2	y
0	0	–1
1	0	1
0	1	1
1	1	–1



Kombination mehrerer Neuronen

zwei Klassen, die nicht linear separierbar sind:

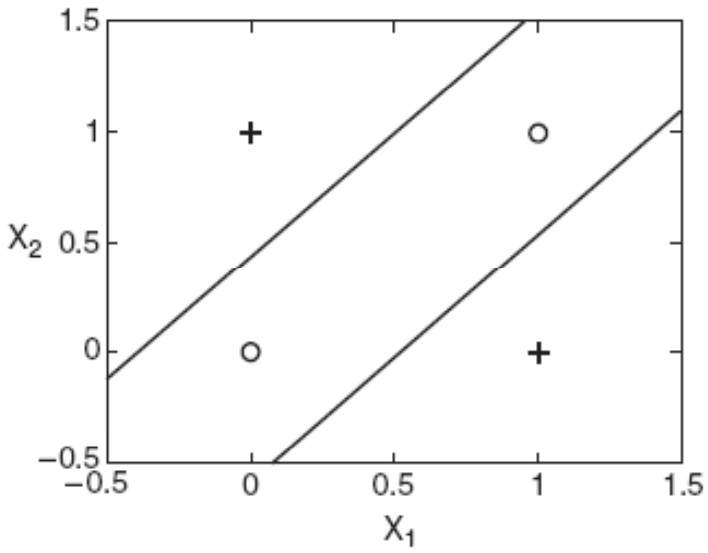
⇒ mehrere innere Knoten (hidden layer) und ein Output-Knoten (zwei-Klassen-Problem, sonst mehr)

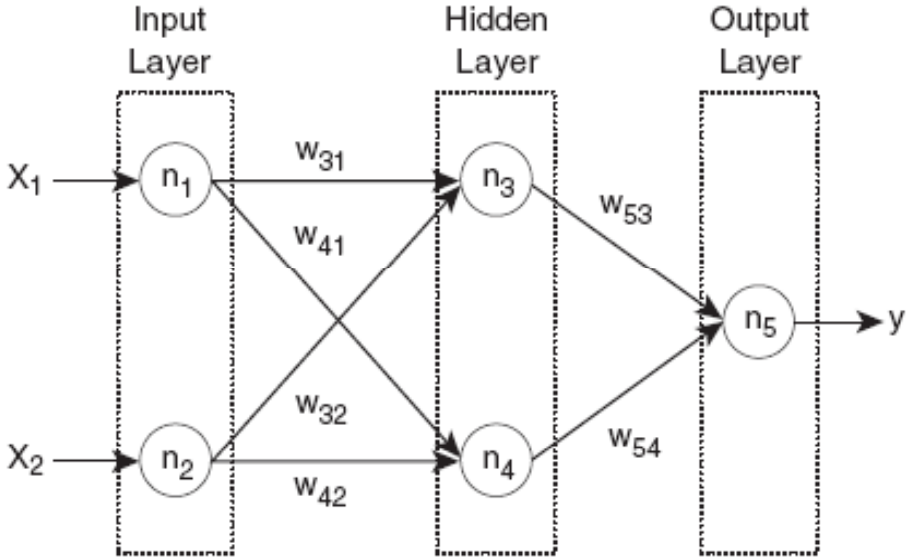


Varianten: feed-forward vs. recurrent (auch Verbindungen auf derselben Ebene oder zu vorherigen Ebenen möglich)

Kombination mehrerer Neuronen

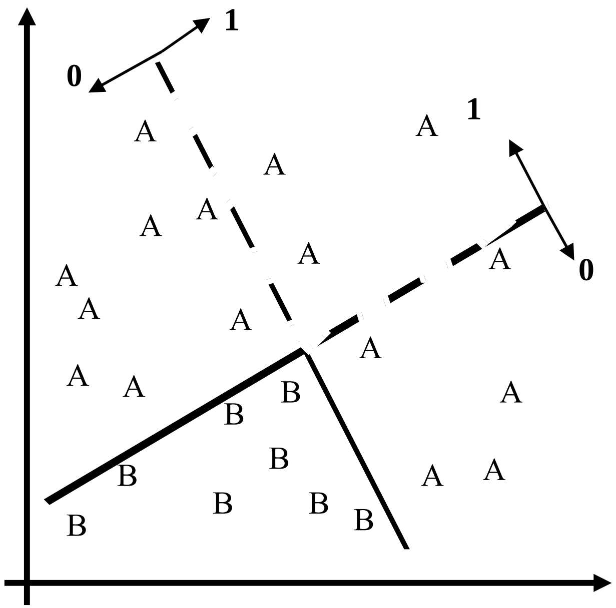
XOR

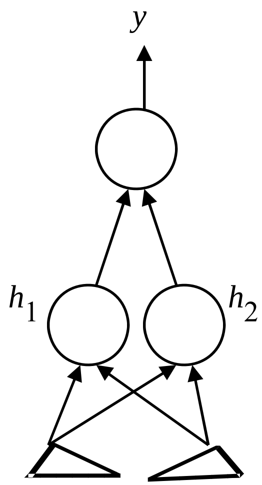




Kombination mehrerer Neuronen

Beispiel





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

$andernfalls: \ y = 1 \ (Klasse \ \ A)$

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. (Least Squares Optimierung)
- *Voraussetzung:* Output *y stetige* Funktion der Aktivierung *a*

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;

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

  $\Rightarrow$ niedrige Klassifikationsgüte

- zu viele Knoten

  $\Rightarrow$ Overfitting

Bestimmung der Netztopologie

nach [SPSS Clementine 2000]

Statische Topologie

- Topologie wird a priori 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

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.

Diskussion

+ im Allgemeinen sehr hohe Klassifikationsgüte
  beliebig komplexe Entscheidungsflächen (möglichst einfache Topologie auswählen, um Overfitting zu vermeiden)
+ redundante Features relativ problemlos, Gewichte werden klein
+ Effizienz der Anwendung

- schlechte Verständlichkeit
  (lernt nur Gewichte, aber keine Klassenbeschreibung)
- Ineffizienz des Lernens (sehr lange Trainingszeiten)
- keine Integration von Hintergrundwissen
- empfindlich gegen Fehler in den Trainingsdaten
- mögliche Konvergenz in lokales Minimum