Extraktion von Regeln aus neuronalen Netzen

Rule extraction from neural networks

Bachelor-Thesis von Barbara Zöller

Juli 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Knowledge Engineering

Extraktion von Regeln aus neuronalen Netzen
Rule extraction from neural networks

Vorgelegte Bachelor-Thesis von Barbara Zöller

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Frederik Janssen, Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 29. Juli 2014

(Barbara Zöller)

Inhaltsverzeichnis

1 Einleitung **6**
- 1.1 Motivation . . . 6
- 1.2 Ziel . . . 6
- 1.3 Aufbau . . . 6

2 Klassifikation im Kontext maschinellen Lernens **7**
- 2.1 Grundlagen der Klassifikation . . . 7
- 2.2 Verfahren der Klassifikation . . . 7

3 Neuronale Netze **10**
- 3.1 Idee . . . 10
- 3.2 Notation . . . 10
- 3.3 Aufbau . . . 10
- 3.4 Lernverfahren . . . 14
- 3.5 Grenzen . . . 16

4 Regel-Lerner **17**
- 4.1 Notwendigkeit der Regelextraktion . . . 17
- 4.2 Typen von Regelmengen . . . 17
  - 4.2.1 Boolesche Funktionen . . . 17
  - 4.2.2 IF-THEN-Regeln . . . 18
  - 4.2.3 M-aus-N–Regeln . . . 19
  - 4.2.4 Entscheidungsdiagramm . . . 19
  - 4.2.5 hierarchische Regeln . . . 20
  - 4.2.6 Entscheidungsbaum . . . 21
  - 4.2.7 Fuzzy-Regeln . . . 21
- 4.3 Qualität der Regeln . . . 22
- 4.4 Unterschiedliche Ansätze zur Regelextraktion . . . 23

5 Kriterien zur Bewertung der Verfahren **24**

6 Untersuchte Verfahren zur Regelextraktion **25**
- 6.1 Extraktion von Regeln aus trainierten neuronalen Netzen [Tsu00] . . . 25
- 6.2 Extraktion von M-aus-N-Regeln [Set00] . . . 27
- 6.3 Regelextraktion aus neuronalen Netzen mit Entscheidungsdiagrammen [CZ11] . . . 31
- 6.4 Rekursive Regelextraktion für Daten mit gemischten Attributen [SBM08] . . . 33
- 6.5 Regelextraktion mit genetischen Algorithmen [SNF00] . . . 35
- 6.6 Regelextraktion via Entscheidungsbauminduktion [ST01] . . . 38
- 6.7 Extraktion von Prioritätsregeln basierend auf Statistik [ZCC00] . . . 40
- 6.8 Extraktion von Fuzzy-Regeln [KzB13] . . . 41

7 Vergleich der Verfahren zur Regelextraktion **44**
- 7.1 Art des verwendeten Ansatzes . . . 44
- 7.2 Methoden zur Vereinfachung der Regelextraktion . . . 44
- 7.3 Form der extrahierten Regeln . . . 45

8 Ausblick **46**

Abbildungsverzeichnis

1 Datenmenge für zwei gegebene Variablen x_6 und x_7 und mit drei Klassen [Bis06, S. 34] . 7
2 Darstellung Overfitting [web] 8
3 Modell eines Neurons [RN03, S. 737] 11
4 Sigmoide Funktion $\frac{1}{(1+e^{-x})}$ [RN03, S. 738] 11
5 Graph der Ausgabe eines Perzeptrons mit zwei Eingängen und einer sigmoiden Aktivierungsfunktion [RN03, S. 740] 12
6 Graph der Ausgabe eines neuronalen Netzes vier Hidden Units und sigmoider Aktivierungsfunktion [RN03, S. 744] 12
7 Schematische Darstellung eines feedforward-Netzes [Dre05] 13
8 Schematische Darstellung eines rekurrenten Netzes 13
9 Realisierung der UND-Funktion (links) und der ODER-Funktion 14
10 Beispiel für ein Entscheidungsdiagramm für Ausdruck $v_1 = v_2 \vee v_5 = 1$ [CZ11] 20
11 Beispiel für einen Entscheidungsbaum zur Frage „Fahre ich mit dem Fahrrad zur Uni ?“ . . 21
12 Unterteilung eines kontinuierlichen Attributes in fünf fuzzy Bereiche [KzB13] 22
13 Qualitative Norm [Tsu00, S. 386] 27
14 hyperbolische Tangensfunktion 29
15 Veranschaulichung der Regelexpansion für 3 Regeln R_1, R_2, R_3 [ST01, S. 1873] 39

Tabellenverzeichnis

1 mathematische Notation 10
2 Wertetabellen mit Berechnungen für logisches Und $\wedge$ (links) und logisches Oder $\vee$ (rechts) 14
3 Wertetabellen für Entscheidungsdiagramm (siehe Abbildung 10) 20
4 Untersuchung der Regelextraktion aus trainierten Netzen nach Tsukimoto [Tsu00] 25
5 Untersuchung der Extraktion von M-aus-N-Regeln [Set00] 27
6 Untersuchung der Extraktion mit Entscheidungsdiagrammen [CZ11] 31
7 Untersuchung der Regelextraktion für Daten mit gemischten Attributen [SBM08] 33
8 Untersuchung der Regelextraktion mit genetischen Algorithmen [SNF00] 35
9 Untersuchung der Regelextraktion via Entscheidungsbauminduktion [ST01] 38
10 Untersuchung der Regelextraktion basierend auf Statistik [ZCC00] 40
11 Untersuchung der Extraktion von Fuzzy-Regeln [KzB13] 41
12 Erzeugung der binären Form einer Fuzzy-Regel [KzB13] 43

Algorithmenverzeichnis

1 Perceptron-Learning($examples, network$) (vgl. [RN03, S. 742]) 15
2 Backpropagation-Learning($examples, network$) (vgl. [RN03, S. 746]) 16
3 polynomieller Algorithmus zur Erezeugung von booleschen Funktionen(vgl. [Tsu00, S. 382]) 26
4 LORE (local rule extraction) (vgl. [CZ11, S. 2]) 32
5 Re-RX(S,D,C) (vgl. [SBM08, S. 301]) 34
6 angepasster RX-Algorithmus(S,D,C) (vgl. [SNF00, S. 134]) 37
7 CRED Algorithmus (vgl. [ST01, S. 1871]) 39
8 STARE-Algorithmus (vgl. [ZCC00, S. 404]) 41
9 DIFACONN-miner Algorithmus (vgl. [KzB13]) 43

1 Einleitung

1.1 Motivation

In der heutigen Zeit ist es möglich immer größere Datensätze zu erzeugen und zu speichern. In vielen Situationen des alltäglichen Lebens werden Daten gespeichert mit dem Ziel, daraus neue Erkenntnisse zu gewinnen. So werden zum Beispiel die betrachteten Produkte bei Online-Versandhändlern gespeichert um dem Nutzer ähnliche Produkte zu zeigen, die möglicherweise ebenfalls für ihn interessant sind. Sie können so gezielt Werbung erstellen, die dem Interessengebiet des jeweiligen Nutzers entspricht. Dazu werden Methoden des maschinellen Lernens verwendet. Diese können Muster in Datenmengen erkennen und neue Erkenntnisse gewinnen, die noch nicht im angewendeten Programm vorgegeben waren.

Künstliche neuronale Netze sind ein solches Verfahren. Sie funktionieren nach dem Prinzip des menschlichen Gehirns, indem Neuronen über Synapsen aktiviert werden um Informationen weiterzuleiten. Künstliche neuronale Netze finden häufig Verwendung, da sie in der Lage sind große Datensätze mit vielen Parametern zu klassifizieren. Jedoch haben diese den großen Nachteil, dass der Klassifikationsvorgang für den menschlichen Nutzer nicht verständlich ist. Somit ist nicht nachvollziehbar, auf welcher Grundlage die Klassifikation getroffen wurde. Dies führte auch dazu, dass neuronale Netze misstrauisch betrachtet wurden und ihnen zeitweise wenig Beachtung gegeben wurde. Daher ist es wünschenswert dafür Abhilfe zu schaffen. Dazu werden Verfahren gesucht, die Regeln aus den Netzen extrahieren und den Klassifikationsvorgang auf eine für den Menschen verständlichere Weise zu beschreiben. Diese Regeln sind im Prinzip Vorschriften, die beschreiben, unter welchen Voraussetzungen ein Datensatz einer bestimmten Klasse zugewiesen wird. Hierfür gibt es bereits unterschiedliche Ansätze, die mitunter die Struktur des Netzes analysieren oder aber einfach nur die Ergebnisse der Klassifikation durch das neuronale Netz verwenden. Hierbei werden unterschiedliche Arten von Beschreibungen (für die Regeln) extrahiert.

1.2 Ziel

Ziel dieser Arbeit ist es einen Überblick über verschiedene Verfahren zur Regelextraktion aus neuronalen Netzen zu geben. Dazu werden Kriterien definiert, anhand derer die vorgestellten Verfahren zunächst anhand einiger Kenngrößen verglichen werden können. Zudem werden die Verfahren bezüglich ihrer unterschiedlichen „Strategien“ verglichen und Ideen zur Weiterentwicklung der Verfahren gegeben.

1.3 Aufbau

Die vorliegende Arbeit beschreibt zunächst allgemein in Kapitel 2 die Grundlagen der Klassifikation im Kontext maschinellen Lernens um anschließend in Kapitel 3 ein spezielles Klassifikationsverfahren, die neuronalen Netze, vorzustellen. In Kapitel 4 werden Regel-Lerner eingeführt, wobei die bei der Regelextraktion verwendeten Regeln beschrieben werden. Um die Verfahren zu vergleichen werden Kriterien in Kapitel 5 definiert, die bei der anschließenden Vorstellung der untersuchten Verfahren in Kapitel 6 angewendet werden. Abschließend werden in Kapitel 7 die Verfahren verglichen.

2 Klassifikation im Kontext maschinellen Lernens

Klassifikation ist ein allgemeiner Vorgang um verschiedene Objekte oder Ereignisse einer oder mehreren Klassen zuzuordnen und darauf aufbauend Entscheidungen zu treffen. Im Alltag gilt es auch immer wieder Dinge einzuordnen, wie zum Beispiel „genießbar“ – „ungenießbar“ oder Äpfel von Birnen zu unterscheiden. Jedoch gibt es auch komplexere Zusammenhänge, die vom Menschen nur schwer beim reinen Ansehen erkannt werden können. Dazu gehören beispielsweise Diagnosen in der Medizin, die aufgrund verschiedener Messwerte getroffen werden sollen. Oder auch Prozesse, die automatisiert ohne den Eingriff eines Benutzers ablaufen sollen. Darunter fällt unter anderem die Erkennung von Spams im Email-Postfach oder auch das maschinelle Lesen von Handschrift (wie zum Beispiel zur Ermittlung der Adresse auf einem Brief). Somit ergeben sich viele Bereiche, für die automatisierte Klassifikationsverfahren von Bedeutung sind. Dazu wird zunächst die Motivation der Klassifikation erläutert (siehe Abschnitt 2.1) und anschließend werden verschiedene Ansätze vorgestellt (siehe Abschnitt 2.2).

2.1 Grundlagen der Klassifikation

Die Klassifikation ordnet einem Vektor $x = (x_1, \ldots, x_n)$ aus einer gegebenen Datenmenge X mit n gemessenen Merkmalen einer der K diskreten Klassen C_k zu. In der Regel sind diese Klassen disjunkt und somit wird jeder Vektor x genau einer Klasse C_k zugeordnet. Es wird ein entsprechendes Modell gesucht, das diese Klassifikation möglichst gut ausführt, also möglichst viele Beispiele richtig klassifiziert. Diese Modelle müssen Funktionen finden, die den Eingaberaum entsprechend unterteilen. Dabei unterscheidet man zwischen linearen Klassifizierern, die nur lineare Funktionen zur Trennung der verschiedenen Klassen finden und nicht-linearen, die auch nicht-lineare Trennungen durchführen. Ein Beispiel für eine solche Klassifikationsaufgabe ist in Abbildung 1 zu sehen. Hier wurden zwei Merkmale gemessen anhand denen die Objekte verschiedenen Klassen zugeteilt werden sollen. Hier sieht man gut, dass die „blaue“ Klasse leichter von den anderen beiden abzugrenzen ist, als die „rote“ von der „grünen“. Die Bezeichnung der unterschiedlichen Klassen werden auch Label genannt. Hat man eine guten Klassifizierer gefunden, kann man weitere Beobachtungen, wie etwa das in Abbildung 1 eingezeichnete **x**, klassifizieren.

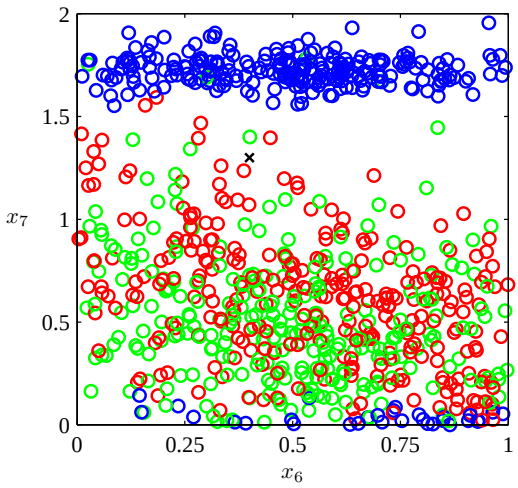

Abbildung 1: Datenmenge für zwei gegebene Variablen x_6 und x_7 und mit drei Klassen [Bis06, S. 34]

2.2 Verfahren der Klassifikation

Um ein System entsprechend zu trainieren ist ein Lernalgorithmus und eine Trainingsdatenmenge notwendig. Damit lässt sich ein entsprechendes Modell zur Klassifikation finden. Durch Feedback wird der

Lernprozess unterstützt. Dies hat auch Einfluss auf die Klassifikationsmethoden. Hier kann man zwischen folgenden Prinzipien unterscheiden [RN03, S. 650]:

überwachtes Lernen (engl. supervised learning) Das lernende System hat Zugriff auf die zugehörige Klasse der Objekte aus den Trainingsbeispielen. Im Allgemeinen werden hier jeweils aktuelles Klassifikationsergebnis und gewünschtes Ergebnis verglichen, wobei das aktuelle Modell bei Fehlern entsprechend angepasst wird. Dazu zählen zum Beispiel Support-Vektor-Maschinen (SVM) und künstlich neuronale Netze.

unüberwachtes Lernen (engl. unsupervised learning) Hier stehen keine Informationen über die Klassenzugehörigkeit der Trainingsbeispiele zur Verfügung. Deshalb werden die Daten so segmentiert, dass stark zusammenhängende Partien, die sich nur kaum anhand ihrer Merkmale unterscheiden (Cluster), in eine Klasse fallen oder es werden Hauptkomponenten gesucht, indem man unwichtige Komponenten der Daten weglässt.

bestärkendes Lernen (engl. reinforcement learning) Dieses Verfahren beurteilt den Nutzen einer Aktion und kann so das System „belohnen“, wenn eine (strategisch) günstige Aktion ausgeführt wurde. Dazu werden meist Wahrscheinlichkeitsverteilungen (wie zum Beispiel Markovketten) verwendet.

Da die Akzeptanz von Klassifizierern von der Genauigkeit abhängt, sollen auch unbekannte Beispiele gut klassifiziert werden können (wie schon in Abschnitt 2.1 erwähnt). Dazu ist eine gute Generalisierungsfähigkeit erstrebenswert. Oder anders formuliert, Overfitting (Überanpassung) soll vermieden werden [RN03, S. 662]. Diese entsteht, wenn das Modell zu speziell auf die Trainingsdaten angepasst wird und somit zum Beispiel Regeln entstehen, die nur einzelne Beobachtungen abdecken, also durchaus uneffizient sind. In Abbildung 2 sieht man die stark geschlängelte grüne Trennungslinie, die zwar jeden einzelnen Punkt exakt seiner zuordnet, jedoch komplex zu beschreiben ist. Im Gegensatz dazu die einfachere schwarze Kurve, die zwar nicht alle Beobachtungen exakt ihrer Klasse zuweist, aber eine deutlich bessere Generalisierung aufweist. Diese Funktion ist deutlich einfacher zu beschreiben und beachtet Ausreißer nicht.

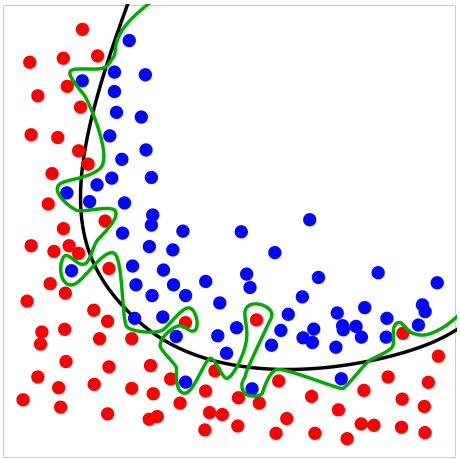

Abbildung 2: Darstellung Overfitting [web]

Die Quote von falschen Klassifikationen wird durch Anwendung des trainierten Modells auf eine Validierungsmenge ermittelt. Die Validierungsmenge dient dazu, die Klassifizierungseigenschaften des Verfahrens zu messen (z.B. ob eine bestimmte Klassifizierungsquote schon erreicht wurde) und auch um einen Vergleich mit anderen Verfahren aufzustellen. Dazu existieren folgende gängigen Möglichkeiten um aus den gegebenen Daten eine geeignete Validierungsmenge zu erhalten:

Begrenzung der Trainingsgröße Die gegebenen Daten werden zunächst in eine Trainings- und eine Validierungsmenge eingeteilt. Wobei in regelmäßigen Abständen, während des Lernens auf der Trainingsmenge, das aktuelle Modell auf die Validierungsmenge angewendet wird um festzustellen,

ob das Training noch zu einer angemessenen Verbesserung beiträgt. Wenn erkannt wird, dass dies nicht mehr effizient geschieht, wird das Training abgebrochen.

Kreuzvalidierung Hier wird die gegebene Datenmenge in k Datenmengen unterteilt, wobei nach und nach jede Datenmenge je einmal als Trainings- sowie Validierungsmenge verwendet wird. Hierbei werden $(1 - \frac{1}{k})$-tel der Daten zum Training verwendet und der restliche kleinere Teil $(\frac{1}{k})$ zur Validierung [Bis06, S. 32f].

3 Neuronale Netze

In diesem Kapitel werden die Grundlagen von neuronalen Netzen erläutert, die für das Verständnis dieser Arbeit erforderlich sind. Dabei wird eine einheitliche mathematische Notation eingeführt sowie grundlegende Informationen zu Aufbau, Training und Einsatzmöglichkeiten/-grenzen von künstlichen neuronalen Netzen gegeben.

3.1 Idee

Aus biologischen Untersuchungen hat man festgestellt, dass das menschliche Gehirn sehr leistungsfähig ist und schnell komplexe Entscheidungen treffen kann, obwohl die Vermittlungszeit mit 10^{-3} um ein 10^7-faches höher ist als die eines Computers. Daraus zog man den Schluss, dass ein hoch paralleler Prozess, verteilt auf mehrere Neuronen, ablaufen muss [Mit97]. Das menschliche Gehirn kann komplexe Aufgaben, wie z. B. Gesichtserkennung in weniger als einer Sekunde durchführen. Zudem ist es weniger fehleranfällig und kann in der Regel ohne Weiteres mit neuen Eingabewerten umgehen – im Gegensatz zu Computerprogrammen, wo dies von der Konfiguration des Programmierers abhängt [RN03]. Schließlich ist das menschliche Gehirn selbst in der Lage Fehler zu erkennen und auszugleichen sowie fähig aus den aufgenommenen Informationen zu Lernen, d. h. diese zu verarbeiten und daraus neue Schlüsse zu ziehen. Diese Fähigkeiten sind auch für die Informatik von Bedeutung und deshalb entwickelte man künstliche neuronale Netze, die eine Art Modellierung des menschlichen Gehirns sind.

3.2 Notation

Zunächst wird die in der vorliegenden Arbeit verwendete Notation für mathematische Formeln vorgestellt. Soweit nicht anders angegeben, wird diese, wie in der Tabelle 1 dargestellt, verwendet.

x	Variable
$\mathbf{v}$	Vektor
$\mathbf{M}$	Matrix
$\mathbf{W}$	Matrix aller Kantengewichte eines Netzes
$W_{i,j}$	Gewicht der Kante von Knoten i zu Knoten j
in_i	Eingabefunktion des i-ten Knoten
a_j	Ausgabewert des j-ten Knotens
$g(x)$	Aktivierungsfunktion
$h_{\mathbf{W}}(\mathbf{x})$	Ausgabewert eines Neurons mit den Kantengewichten W
Err	Fehler
e	ein Beispieldatensatz, bestehend aus $\mathbf{x}[e]$ und $\mathbf{y}[e]$
α	Lernrate
$\mathbf{x}[e]$	Eingabevektor des Beispiels e
$\mathbf{x}_j[e]$	Eingabewert des Beispiels e am j-ten Knoten in der Eingabeschicht
$\mathbf{y}[e]$	Ausgabevektor des Beispiels e
$\mathbf{y}_j[e]$	Ausgabewert des Beispiels e am i-ten Knoten in der Ausgabeschicht

Tabelle 1: mathematische Notation

3.3 Aufbau

Ein neuronales Netz besteht aus mehreren Neuronen (auch Perzeptronen genannt, auf engl. unit), die miteinander kombiniert werden können um verschiedene Funktionen zu realisieren [RN03, S. 736ff].

Über die Verbindungen werden die Aktivierungswerte a_{ij} von einem Knoten i zu einem anderen Knoten j durch das Netz geleitet. Zudem ist jede Verbindung mit einem Gewicht $W_{j,i}$ versehen, das die Stärke der Verbindung angibt. Wie in Abbildung 3 zu sehen ist, beinhaltet jedes Neuron zwei Rechenschritte:

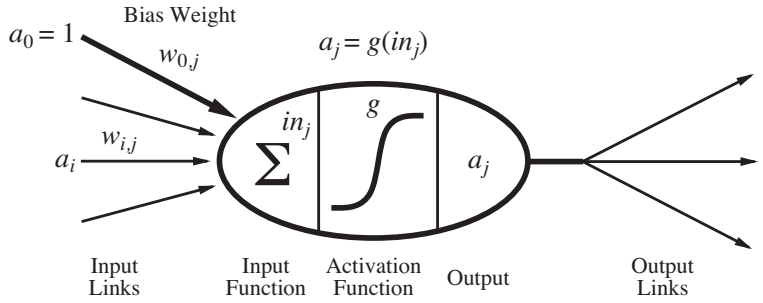

Abbildung 3: Modell eines Neurons [RN03, S. 737]

- *Eingabefunktion* (engl. input function): Berechnet die gewichtete Summe aus den Eingabewerten (Ausgangswerte anderer Neutronen oder Eingabewerte des neuronalen Netzes), dabei stellen Kantengewichte $W_{j,i}$ mit positiven Werten eine anregende Neuronenverbindung dar und negative Werte eine hemmende. Zudem kann gegebenenfalls ein Bias $W_{0,i}$ gesetzt werden, dessen Wert zur Eingabefunktion addiert wird.

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j \tag{1}$$

- *Aktivierungsfunktion* (engl. activation function): nichtlineare Funktion die den Ausgabewert a_i aus der Eingabe berechnet und somit festlegt, wann das Neuron „feuert“– also einen positiven Wert weitergibt

$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$

  Als Aktivierungsfunktion wird häufig die sigmoide Funktion $1/(1+e^{-x})$ an Stelle einer einfachen Schwellwertfunktion (Signumsfunktion), wie auf Abbildung 4, verwendet, da diese stetig differenzierbar ist, was für den Trainingsalgorithmus von Bedeutung ist (siehe Abschnitt 3.4).

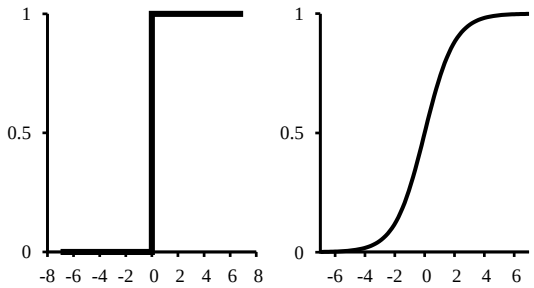

Abbildung 4: Sigmoide Funktion $\frac{1}{(1+e^{-x})}$ [RN03, S. 738]

Mit einem einzelnen Neuron können alle drei logischen Grundoperatoren AND, OR und NOT dargestellt werden. Es handelt sich um einen linearen Separator, da die Eingabefunktion (siehe Gleichung1) eine Hyperebene (vergleiche Abbildung 5)

$$\sum_{j=0}^{n} W_{j,i} x_j > n \Leftrightarrow \mathbf{W} \cdot \mathbf{x} > n, n \in \mathbb{R}$$

aufspannt (n ergibt sich aus der verwendeten Aktivierungsfunktion). Somit können mit Netzen aus diesen Grundoperatoren (mindestens zwei Schichten) jede beliebige boolesche Funktion dargestellt werden [Mit97, S. 87]. Eine graphische Darstellung für ein Perzeptron mit vier Hidden Units, also eine Kombination aus vier einfachen sigmoiden Funktionen, ist in Abbildung 6 zu sehen. Die Schichten werden wie

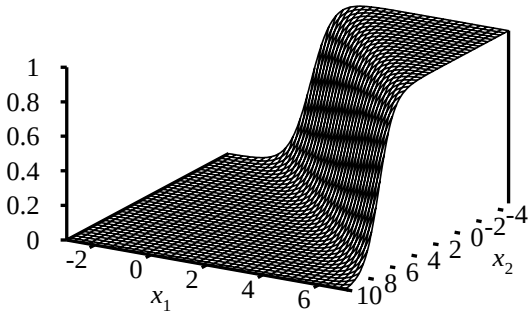

Abbildung 5: Graph der Ausgabe eines Perzeptrons mit zwei Eingängen und einer sigmoiden Aktivierungsfunktion [RN03, S. 740]

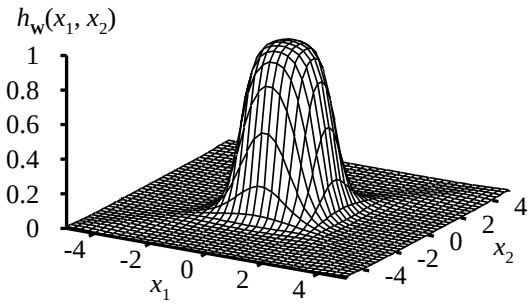

Abbildung 6: Graph der Ausgabe eines neuronalen Netzes vier Hidden Units und sigmoider Aktivierungsfunktion [RN03, S. 744]

folgt bezeichnet:

- *Eingabeschicht* (engl. Input Layer): vorderste Schicht des Netzes, die die Eingabewerte des Netzes erhält
- *verdeckte Schicht* (engl. Hidden Layer): dazwischenliegende Schicht, die weder mit Netzeingängen noch -ausgängen verbunden ist
- *Ausgabeschicht* (engl. Output Layer): hinterste Schicht des Netzes, die die Ausgabewerte des Netzes berechnet

Man unterscheidet dabei zwischen zwei grundlegenden Arten von Netzstrukturen der künstlich neuronalen Netze [RN03, S. 738]:

- *Netze ohne Rückkopplung (engl. feedforward)* haben keine Zyklen, so dass die Funktion nur von den aktuellen Eingabewerten abhängig ist. Einschichtige Netze stellen eine Hyperebene dar und können nur linear separierbare Funktionen darstellen. Um dagegen komplexere Funktionen (wie z. B. XOR) darzustellen werden mehrere Schichten benötigt. Ein mehrschichtiges Netz mit n Eingabewerten $x_1, \ldots, x_n$, einer verdeckten Schicht N_c und einer Ausgabeschicht N_o ist in Abbildung 7 zu sehen.

- *Netze mit Rückkopplung (engl. recurrent)* haben Zyklen und Ausgabewerte können wieder im selben Netz zu Neuronen derselben Schicht oder vorangegangenen Schichten zurückgeleitet werden. Somit können Werte mehrfach verwendet werden, auch über mehrere Schichten hinweg und es entsteht ein „Zwischenspeicher“. Wie in der Abbildung 8 zu sehen ist, können auch Ausgaben eines Knotens direkt an diesen wieder als Eingabewert geleitet werden.

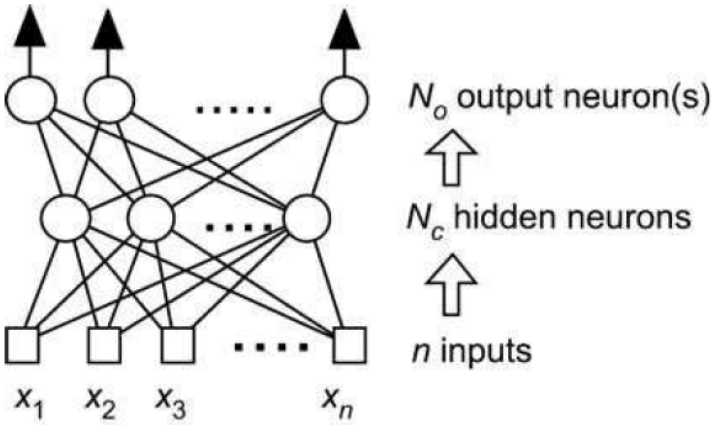

Abbildung 7: Schematische Darstellung eines feedforward-Netzes [Dre05]

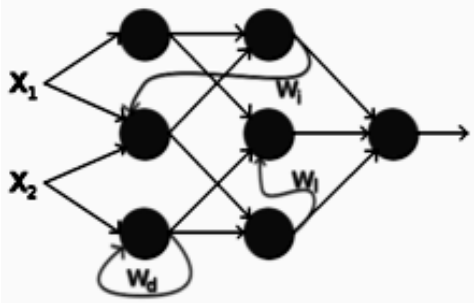

Abbildung 8: Schematische Darstellung eines rekurrenten Netzes

Durch Anpassen bzw. Verändern der Kantengewichte kann die Ausgabe des Netzes verändert werden. Dies ist für das Training von künstlichen neuronalen Netzen von Bedeutung und wird im Abschnitt 3.4 näher erläutert.
Zur Veranschaulichung der Funktionsweise zur Berechnung des Ausgabewertes sind in Abbildung 9 Beispiele zu sehen, die die logische UND-Funktion sowie die ODER-Funktion mit einem neuronalen Netz darstellen.

Die Berechnung des Ausgabewertes a erfolgt unter Anwendung der Aktivierungsfunktion g auf die Eingabefunktionswert in (siehe Formel 1). Dabei kann man gut sehen wie die Kantengewichte sowie ein gegebenenfalls gesetzter Bias $W_{0,i}$ das Endergebenis der Eingabefunktion beeinflussen (siehe dazu Tabellen 2).

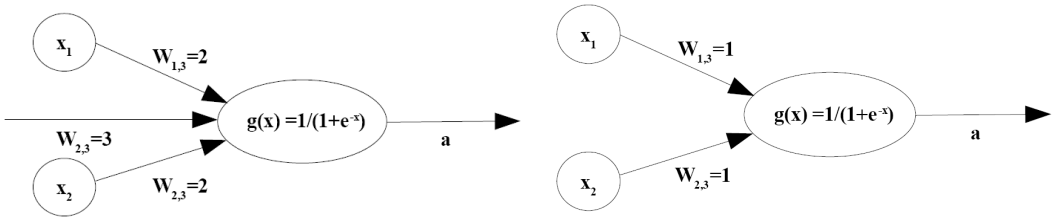

Abbildung 9: Realisierung der UND-Funktion (links) und der ODER-Funktion

x_1	x_2	in	$a = g(in)$
0	0	$-3+2\cdot 0+2\cdot 0=-3$	0
0	1	$-3+2\cdot 0+2\cdot 1=-1$	0
1	0	$-3+2\cdot 1+2\cdot 0=-1$	0
1	1	$-3+2\cdot 1+2\cdot 1=+1$	1

x_1	x_2	in	$a = g(in)$
0	0	$1\cdot 0+1\cdot 0=0$	0
0	1	$1\cdot 0+1\cdot 1=1$	1
1	0	$1\cdot 1+1\cdot 0=1$	1
1	1	$1\cdot 1+1\cdot 1=2$	1

Tabelle 2: Wertetabellen mit Berechnungen für logisches Und $\wedge$ (links) und logisches Oder $\vee$ (rechts)

3.4 Lernverfahren

Um ein künstliches neuronales Netz zum Klassifizieren von Daten verwenden zu können, ist zuvor ein Training notwendig (siehe Abschnitt 2.2). Das bedeutet, dass das Modell ermittelt wird, welches anhand der zur Verfügung stehenden Trainings- und Validierungsmenge die Klassifikationsaufgabe am besten erfüllt. Dabei werden die Kantengewichte so angepasst, dass die Fehlerrate der Klassifikation minimiert wird. Somit handelt es sich hierbei um ein Minimierungsproblem.

Eine gängige Methode hierfür ist die Methode der kleinsten Quadrate. Dazu werden die Fehler quadriert und aufsummiert. Die Quadratur hat zur Folge, dass kleine Fehler (< 1) noch kleiner werden und größere Fehler (> 1) noch größer werden, also ihr Einfluss auf die Fehlerrate zu nimmt. Somit ist es effizienter zur Minimierung der Fehlerrate diese „großen“ Fehler zu reduzieren. Der Fehler lässt sich für ein einzelnes Neuron wie folgt aus der Differenz des erwarteten y und des tatsächlichen Ausgabewertes $h_{\mathbf{W}}(\mathbf{x}) = g\left(\sum_{j=0}^{n} W_j x_j\right)$ berechnen:

$$Err_{ges} = \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 = \frac{1}{2}Err^2 \quad mit \quad Err = y - h_{\mathbf{W}}(\mathbf{x})$$

Eine verbreitete Methode zur Minimierung einer Funktion ist das Gradientenverfahren. Das versucht anschaulich ausgedrückt, den kürzesten Weg vom „Gipfel“ nach unten ins „Tal“ zu finden, was einer Suche nach dem Minimum (=Tal) entspricht. Dazu folgt man stets dem steilsten Weg (entspricht der größten Steigung), da dieser die kürzeste Länge aufweist. Jedoch besteht dabei die Gefahr in einer örtlichen Senke (=lokales Minimum) stecken zu bleiben. Mathematisch betrachtet, berechnet man die partielle Ableitung des Gesamtfehlers Err_{ges} nach W_j (also für jedes Kantengewicht) :

$$\frac{\partial Err_{ges}}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right) = -Err \times g'(in) \times x_j$$

Mit der partiellen Ableitung sowie einer Lernrate α, die angibt in welchem Maße W_j verändert wird, enthält man den Perzeptron-Learning Algorithmus, der die Kantengewichte mit

$$in_i = \sum_{j=0}^{n} W_j a_j \tag{2}$$

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e] \tag{3}$$

anpasst (siehe Algorithmus 1). Dabei werden die Gewichte W_j vergrößert, wenn der Fehler $Err = y[e] - g(in)$ positiv ist und somit der Ausgabewert des Netzes zu klein ist und W_j wird verringert, wenn Err negativ ist.

Algorithmus 1 Perceptron-Learning($examples, network$) (vgl. [RN03, S. 742])

return a perceptron hypothesis
input $examples$, a set of examples, each with input $\mathbf{x} = x_1, \ldots, x_n$ and output y
input $network$, a perceptron with weights W_j, $j = 0, \ldots, n$ and activation function g
repeat
  for each e **in** $examples$ **do**
    $in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$
    $Erry \leftarrow y[e] - g(in)$
    $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
until some stopping criterion is satisfied
return Neural-Hypothesis($network$)

Der Perzeptron-Lernalgorithmus kann nun auf künstliche neuronale Netze erweitert werden, indem man statt einzelnen Werten h_W, y und x jetzt Vektoren $\mathbf{h_W}$, $\mathbf{y}$ und $\mathbf{x}$ hat. Um zu erfahren welche Kantengewichte für den Fehler verantwortlich sind und somit angepasst werden müssen, wird der Fehler der Ausgabeschicht $\Delta y_i = \mathbf{y} - \mathbf{h_W}$ wieder in das Netz zurückgeleitet. Dies wird als Backpropagation-Lernen bezeichnet (siehe Algorithmus 2). Dabei ist auch die Aktivierungsfunktion zu berücksichtigen, da diese maßgeblich den Ausgabewert a_i beeinflusst. Somit lässt sich das Update für die Kanten zwischen Neuronen der verdeckten und der Ausgabeschicht wie folgt berechnen (analog zur Gleichung 3):

$$W_{j,i} = W_{j,i} + \alpha \times a_j \times \Delta_i \quad mit \quad \Delta_i = Err_i \times g'(in_i)$$

Für die Berechnung der neuen Kantengewichte zwischen Eingabe- und verdeckter Schicht ist die Berechnung aufwändiger, da die bereits beim zurückgehen verwendeten Kantengewichte mit einberechnet werden müssen. Für den Fehler Δ_j gilt:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

Somit ergibt sich als Aktualisierung für die restlichen Kanten (die nicht zur Ausgabeschicht führen)

$$W_{k,j} = W_{k,j} + \alpha \cdot a_k \cdot \Delta_j$$

Es ist zu sehen, dass das Update stets nach dem gleichen Prinzip berechnet wird, wobei sich der Fehler jeweils ändert, da alle vorhergegangen immer mit weiter übernommen werden.

Neben der Optimierung der Kantengewichte, hat auch der Aufbau eines neuronalen Netzes Einfluss auf das Klassifizierungsergebnis. Hierzu gibt es verschiedene Herangehensweisen: Man beginnt mit einem einzelnen Neuron und fügt nach und nach neue Knoten und Schichten hinzu bis das Klassifizierungsergebnis der Trainingsdaten zufriedenstellend ist. Eine Alternative dazu ist, ein vollständig verbundenes Netz zu haben und entfernt dort iterativ alle Kanten (evtl. auch Knoten), deren Entfernen das Trainingsergebnis nicht negativ beeinflusst. Bei allen Trainingsverfahren ist darauf zu achten, dass das Netz nicht zu gut auf die Trainingsdaten angepasst ist, denn dann werden spätere Datensätze mit höherer Wahrscheinlichkeit falsch klassifiziert, da die Generalisierungsfähigkeiten des Netzes zu schlecht sind. Um diesem Phänomen des Overfittings (siehe hierzu Abschnitt 2.2) entgegenzuwirken, werden unter anderem Kreuzvalidierungsmethoden (wie ebenfalls in Abschnitt 2.2 beschrieben) eingesetzt.

Algorithmus 2 Backpropagation-Learning($examples, network$) (vgl. [RN03, S. 746])

return a neural network
input $examples$, a set of examples, each with input vector **x** and output **y**
input $network$, a multilayer network with L layers, weights $W_{j,i}$, activation function g
repeat
  for each e **in** $examples$
    for each node j in the input layer **do** $a_j \leftarrow \mathrm{x}_j[e]$
    for $\ell = 2$ **to** L
      $in_i \leftarrow \sum_j W_{j,i} a_j$
      $a_i \leftarrow \mathrm{g}(\mathrm{in}_i)$
    for each node i in the output layer **do**
      $\Delta_j \leftarrow \mathrm{g'}(\mathrm{in}_i) \times (y_i[e] - a_i)$
    for $\ell = L - 1$ **to** 1
      for each node j in the layer ℓ **do**
        $\Delta_j \leftarrow \mathrm{g'}(\mathrm{in}_j) \sum_i W_{j,i} \Delta_i$
        for each node i in the layer $\ell + 1$ **do**
          $W_{j,i} \leftarrow \mathrm{W}_{j,i} + \alpha \times a_j \times \Delta_i$
until some stopping criterion is satisfied
return Neural-Net-Hypothesis($network$)

3.5 Grenzen

Wie schon in der Beschreibung des Trainingsvorgangs (vgl. Abschnitt 3.4) der neuronalen Netzen ersichtlich ist, ist es schwierig den Klassifikationsvorgang nachzuvollziehen. Die Ergebnisse sind für den Nutzer nicht transparent, das Training ist im Prinzip für den Menschen wie eine Black Box, denn es ist sehr schwer und aufwändig zu verstehen, was dort passiert (vgl. [Tsu00]).

Das liegt zum einen an der oft komplexen Datenmenge, die viele verschiedene Klassifikationsmerkmale aufweist, aber auch an der abstrakten Darstellung mit den Kantengewichten, die darüber entscheiden, welcher Klasse ein zu identifizierendes Objekt zugeschrieben wird. Oft ist die Generalisierung von künstlichen neuronalen Netze auch zu schlecht. Damit ist gemeint, dass das neuronale Netz sehr viele (oft nur leicht verschiedene) Spezialfälle abdeckt und somit zu wenig allgemeine Aussagen trifft. In diesen Fällen ist es für den menschlichen Nutzer schwer zu erkennen, wie die Klassifikation angepasst werden muss. Schwerwiegende Fehler sind zudem ebenso schwer aufzudecken, da es mit dem bloßen Auge sehr aufwändig ist, die komplexen Berechnungen selbst durch zu führen. Da künstliche neuronale Netze in sensiblen Bereichen angewendet werden, z. B. von Banken zur Abschätzung der Kreditwürdigkeit eines Kunden, zur Optimierung von komplexen Funktionen, zur automatisches Steuerung und zur Mustererkennung (Bilder, Sprache, Schrift), ist es wichtig, dass das Ergebnis für den Anwender (also den Menschen) nachvollziehbar ist. Dies ist gerade in sicherheitskritischen Bereichen von Bedeutung (vgl. [Lud98]).

Daher ist man auf der Suche nach Methoden, die die Klassifikation der künstlichen neuronalen Netze in einer für den Menschen intuitiv verständlicheren Form darstellen um auch dem „ungeübten“ Benutzer erklären zu können, nach welchen Kriterien klassifiziert wird. Deshalb liegt die Idee nahe, künstliche neuronale Netze auf eine für den menschlichen Nutzer intuitivere Weise dar zu stellen, zum Beispiel in Form von Regeln oder Entscheidungsbäumen. Diese sind sowohl maschinell lesbar, als auch für den Menschen leichter verständlich, da sie näher an einer natürlichen Sprache sind.

4 Regel-Lerner

Die Anwendung von künstlichen neuronalen Netzen bringt, insbesondere im Hinblick auf die „Interaktion“ mit dem Menschen, Nachteile mit sich (siehe Abschnitt 3.5). Um trotzdem die Vorteile der künstlichen neuronalen Netze ausnutzen zu können, ist es das Ziel (eine) Methode(n) zu finden, die das Klassifikationsergebnis von neuronalen Netzen nachbilden können. So wurden unterschiedliche Verfahren zur Extraktion von Entscheidungsregeln aus künstlich neuronalen Netzen entwickelt. Im Folgenden wird auf die Vorteile, den Nutzen und die Ziele der Regelextraktion eingegangen und die unterschiedlichen Typen von Extraktionsansätzen werden vorgestellt.

4.1 Notwendigkeit der Regelextraktion

Wie bereits in Abschnitt 3.5 erläutert, ist es wichtig neuronale Netze in verständlicher Form darzustellen bzw. auszudrücken um deren „Inneres“ zu verstehen. Die wichtigsten Gründe hierfür sind (vergleiche [Lud98], [ADT95]):

- Vorteile der neuronalen Netze ausnutzen: Sie können komplexe nicht-lineare Funktionen darstellen, durch die parallele Verarbeitung können auch große Datensätze in angemessener Zeit verarbeitet werden und aufgrund der hohen Fehlertoleranz können sie mit Rauschen und Messfehlern/-ungenauigkeiten umgehen.
- Erklärung für das ermittelte Ergebnis: Es ist immer wichtig und sinnvoll zu wissen, wie das Ergebnis zu Stande kam um die Entscheidung nach zu vollziehen. Wie z. B. bei der Anwendung in Banken: dort möchte der Kunde gerne darüber informiert werden, warum er als nicht kreditwürdig eingestuft wurde.
- automatische Verifikation: Dies ist bei einer Integration in größere Softwaresysteme in der Regel unabdingbar.
- Erkennen von neuen Querverbindungen: Der Mensch hat so die Möglichkeit weitere Beziehungen und Abhängigkeiten, die durch das maschinelle Verfahren nicht aufgedeckt wurden, zu erkennen
- Verbesserung der Generalisierung: Anhand der Regeln kann der Systemnutzer leichter Bereiche erkennen, für die die Klassifikation fehlschlägt und so dementsprechend die Trainingsmenge anpassen
- Erweiterung für sicherheitskritische Bereiche: Durch die bessere Darstellung, was im Inneren des Netzes passiert, können falsche Ergebnisse einfacher erkannt werden. Dies ist gerade für die Anwendung in sicherheitskritischen Bereichen wichtig.

4.2 Typen von Regelmengen

Die in Kapitel 6 vorgestellten Verfahren extrahieren Regeln unterschiedlichen Typs. Allgemein formuliert, beschreiben Regeln Aussagen über Zusammenhänge zwischen Informationen. Dies ermöglicht neues Wissen aus einer Datenmenge mit vorhandenen Regeln abzuleiten. Im Folgenden werden die verwendeten Regelmengen kurz erläutert:

4.2.1 Boolesche Funktionen

Boolesche Funktionen werden gebildet, indem atomare Aussagen mit den booleschen Grundoperatoren verknüpft werden [RN03, S. 204ff]. Atomare Aussagen sind elementare Aussagen, die nicht aus mehreren Aussagen zusammengesetzt sind und somit nicht mehr weiter aufgeteilt werden können. Sie können die Wahrheitswerte wahr oder falsch annehmen. Komplexere Aussagen können durch Verwendung der booleschen Grundoperatoren gebildet werden:

- *Negation* $\neg$: Atomare Aussagen können verneint werden, ihr Wert wird also umgekehrt (wahr wird zu falsch und falsch wird zu wahr). Somit erhält man positive Literale - positive Atomaraussagen (z. B. B) - und negative Literale - negierte Atomaraussagen (z. B. $\neg B$)
- *Konjunktion* $\wedge$: Mehrere Aussagen können durch ein „Und“ miteinander verknüpft werden. Um eine solche verknüpfte Aussage $A \wedge B$ zu erfüllen, müssen beide atomaren Aussagen A und B erfüllt sein, um die Gesamtaussage zu erfüllen.
- *Disjunktion* $\vee$: Dies ist eine Verknüpfung mit „Oder“. Jedoch nicht im umgangssprachlich verwendetem ausschließendem Sinne („entweder ... oder ...“). Eine durch ein $\vee$ verknüpfte Aussage $A \vee B$ ist erfüllt, wenn mindestens eine der Aussagen A, B erfüllt ist.

Jede boolesche Regel lässt sich in einer Normalform darstellen, die eine bestimmte festgelegte Form haben [Sch05, S. 26ff]. Man unterscheidet hier zwischen:

- *Konjunktive Normalform (KNF)*: Dies ist eine Konjunktion von Disjunktionen. Es werden also $\vee$-Verknüpfungen durch $\wedge$ miteinander verbunden.

  $$\bigwedge_i \bigvee_j (\neg) x_{ij}$$

  Beispiel: $(A \vee B \vee C) \ \wedge \ (A \vee \neg B \vee C) \ \wedge \ (\neg A \vee B \vee C)$
- *Disjunktive Normalform (DNF)*: Dies ist eine Disjunktion von Konjunktionen - also das Gegenstück zur KNF. Es werden also $\wedge$-Verknüpfungen durch $\vee$ miteinander verbunden.

  $$\bigvee_i \bigwedge_j (\neg) x_{ij}$$

  Beispiel: $(A \wedge B \wedge C) \ \vee \ (A \wedge \neg B \wedge C) \ \vee \ (\neg A \wedge B \wedge C)$

4.2.2 IF-THEN-Regeln

Regeln in der $IF\text{-}THEN$-Form sind sehr verbreitet und intuitiv verständlich. Diese gehören zur Aussagenlogik, die folgende Basissyntax hat (vergleiche Abschnitt 4.2.1):

- *atomare Aussagen*
- *Verneinung* $\neg$
- *Konjunktion* $\wedge$
- *Disjunktion* $\vee$
- *Implikation* $\Rightarrow$: Dies ist eine konditionale Aussage aus Verknüpfung von Aussagen mit $\wedge$ und $\vee$. Sie besteht aus zwei Teilen: Prämisse, auch Bedingung genannt (IF-Aussage) und Konklusion, auch Konsequenz/Folge (THEN-Folge). Wenn die Prämisse, bestehend aus Konjunktionen und Disjunktionen, zu wahr ausgewertet wird, ist die Aussagen der Konklusion ebenfalls wahr. Die Implikation wird im Allgemeinen durch einen Pfeil $\Rightarrow$ dargestellt beziehungsweise als IF-THEN Aussagen (z. B $IF\ [(A \wedge B) \vee (C \wedge \neg D)]\ THEN\ E$), dabei werden zusammenhängende Verknüpfungen durch Klammern ( ) ausgedrückt.

So kann man beispielsweise die Klassifizierung zwischen einer Birne und einem Apfel vereinfacht (runde Form und gelb-rote Farbe genügen zur Entscheidung, dass es sich um einen Apfel handelt) folgendermaßen formulieren:

$$IF\ (Form\ ist\ rund)\ AND\ (Farbe\ ist\ gelb-rot)\ THEN\ (Obst\ ist\ Apfel)$$

Die IF-THEN-Regeln haben den Vorteil, dass sie prinzipiell leicht und intuitiv verständlich sind. Jedoch nimmt dies mit zunehmender Größe der Bedingung ab.

4.2.3 M-aus-N–Regeln

M-aus-N-Regeln (engl. M-of-N) sind ähnlich zu den IF-THEN-Regeln (siehe Abschnitt 4.2.2). Allerdings ist die Bedingung hier anders formuliert. Diese ist erfüllt, wenn eine bestimmte Anzahl M aus der Menge aller Bedingungen N erfüllt ist [Set00]:

```
IF M der N Bedingungen sind erfüllt
THEN
   Konklusion trifft zu,
ELSE
   Konklusion trifft nicht zu
END
```

So ließe sich beispielsweise folgende Regel bilden:

$$IF\ [\ 2\ aus\ 4:\ (\ \neg A\ )\ (\ C \vee \neg B\ )\ (D)\ (E \wedge F)]\ THEN\ S$$

M-aus-N-Regeln haben den Vorteil, dass manche Regeln, wie z.B. solche mit einer XOR-Verknüpfung

$$IF\ [a \wedge \neg b] \vee [\neg a \wedge b]\ THEN\ T$$

kürzer formuliert werden können:

$$IF\ (genau\ 1\ der\ 2\ Aussagen\ wahr:\ A\ oder\ B\}\ THEN\ T$$

4.2.4 Entscheidungsdiagramm

Entscheidungsdiagramme sind (ähnlich zu Entscheidungsbäumen in Abschnitt 4.2.6) ein gerichteter azyklischer Graph mit einer Wurzel und mehreren Zielknoten, die die verschiedenen Klassen repräsentieren (siehe Abbildung 10). In den Knoten werden Tests zu den verschiedenen Merkmalen gemacht und dann entsprechend des Ergebnisses zu einem anderen Knoten geleitet. Man unterscheidet zwischen geordneten Entscheidungsdiagrammen – die Merkmale F haben eine lineare Ordnung $F_1 < F_2 < \ldots < F_k$ – und reduziert geordneten – hier sind alle isomorphen Teilgraphen vermengt und es gibt keine Knoten mit gemeinsamen Kindern (es handelt sich hierbei um eine kompaktere Darstellung des geordneten Entscheidungsdiagrammes) [CZ11].

Die dargestellte Funktion des in Abbildung 10 Entscheidungsdiagramms lässt sich ermitteln, indem man die Variablenwerte auf den unterschiedlichen Pfaden abliest und ihnen entsprechend ihres Zielfeldes den Wert wahr T oder falsch F zuordnet. Somit erhält man:

$$T = (v_1 = 2 \wedge v_2 = 2) \vee (v_1 = 1 \wedge v_2 = 1) \vee (v_1 = 3 \wedge v_2 = 3) \vee (v_1 = \neg 2 \wedge v_2 = 2 \wedge v_5 = 1) \vee (v_1 = \neg 1 \wedge v_2 = 1 \wedge v_5 = 1) \vee (v_1 = \neg 3 \wedge v_2 = 3 \wedge v_5 = 1)$$

$$F = (v_1 = \neg 2 \wedge v_2 = 2 \wedge v_5 = \neg 1) \vee (v_1 = \neg 1 \wedge v_2 = 1 \wedge v_5 = \neg 1) \vee (v_1 = \neg 3 \wedge v_2 = 3 \wedge v_5 = \neg 1)$$

(vergleiche Tabelle 3). Dieser Ausdruck lässt sich zu $T = v_1 = v_2 \vee v_5 = 1$ zusammenfassen.
vorteilhafte Eigenschaften der reduzierten Entscheidungsdiagrammes:

- Für symmetrische Funktionen ist ihre Größe durch das Quadrat der Anzahl der Merkmale beschränkt

v_1	v_2	v_5	Ergebnis
2	2	-	T
¬2	2	1	T
¬2	2	¬1	F
1	1	-	T
¬1	1	1	T
¬1	1	¬1	F
¬3	3	1	T
¬3	3	¬1	F
3	3	-	T

Tabelle 3: Wertetabellen für Entscheidungsdiagramm (siehe Abbildung 10)

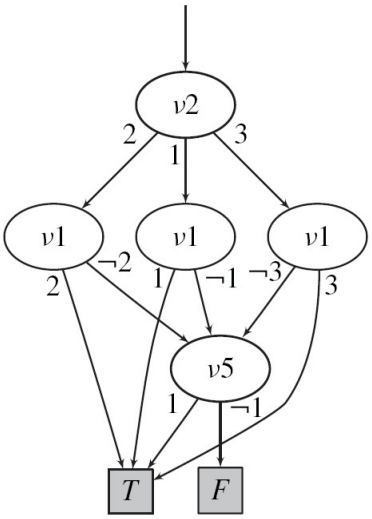

Abbildung 10: Beispiel für ein Entscheidungsdiagramm für Ausdruck $v_1 = v_2 \vee v_5 = 1$ [CZ11]

- Die Laufzeit für binäre Operationen wird durch das Produkt der Größe der Operanden begrenzt

Jedoch ist die Wahl der Merkmalsordnung maßgeblich, denn diese hat Einfluss auf die notwendige Anzahl an Knoten (Varianz zwischen linear und exponentiell möglich).

4.2.5 hierarchische Regeln

Regeln können einer bestimmten hierarchischen Ordnung unterliegen. Dazu wird für jede Regel eine bestimmte Priorität angegeben anhand derer dann eine hierarchische Ordnung aufgestellt werden kann, indem die Regeln anhand ihrer Priorität sortiert werden. Dabei liegen Regeln mit einer höheren Priorität auf einer höheren hierarchischen Ebene als solche mit einem niedrigeren Wert. Die hierarchische Reihenfolge der Regeln hat Auswirkungen auf die Ausführungsreihenfolge. So werden Regeln mit höherer Priotität zuerst zum Klassifizieren angewendet. Die Verteilung der Priorität kann auf verschiedene Weisen erfolgen. So kann die Priorität abhängig von der „Trefferwahrscheinlichkeit“ sein, also wie hoch der Anteil der mit dieser Regel abgedeckten Beispiele ist oder aber von der Anzahl der Bedingungen der jeweiligen Regel.

Durch die hierarchische Ordnung, werden gerade bei einer großen Anzahl von Regeln, die Regeln kenntlich gemacht, die einen Großteil des Klassifikationsergebnisses beeinflussen. So kann der Anwender durch Betrachten dieser Regeln, bereits grobe Informationen über die relevanten Klassifikationsparame-

ter erhalten.

4.2.6 Entscheidungsbaum

Entscheidungsbäume sind eine spezielle Form der Entscheidungsdiagramme. Sie besitzen einen Baumstruktur mit Wurzel, Knoten und Blättern (siehe Abbildung 11) die über Verbindungen, sogenannte Kanten, miteinander verknüpft sind. In jedem Knoten wird der Wert eines anderen Merkmals getestet, womit dann entschieden wird, welcher Pfad zu wählen ist. Der Ursprung des Entscheidungsvorganges ist dabei stets die Wurzel, also der oberste Knoten, von dem alle Kanten ausgehen (vergleiche Knoten „Wolken“ in Abbildung 11). Abhängig von dem Ergebnis, dieser ersten Entscheidung wird entweder der linken oder rechten Kante gefolgt und man gelangt zum nächsten Knoten. Dieses Vorgehen wiederholt sich so oft, bis man an einen Knoten gelangt, von dem keine Kanten mehr wegführen - ein Blatt. Aus diesem kann man dann das Klassifikationsergebnis ablesen [Mit97, S. 52f]. Aus diesem Entscheidungsbaum lassen sich einfach IF-THEN-Regeln bilden, indem man einfach die auf dem zurückgelegten Pfad getroffenen Entscheidungen als Konjunktion aufschreibt. Zudem können Prioritäten anhand der Reihenfolge im Entscheidungsbaum hinzugefügt werden – je näher bei der Wurzel desto größer die Priorität des jeweiligen Merkmals.

In dem in Abbildung 11 dargestellten Entscheidungsbaum beginnt die Klassifikation anhand der

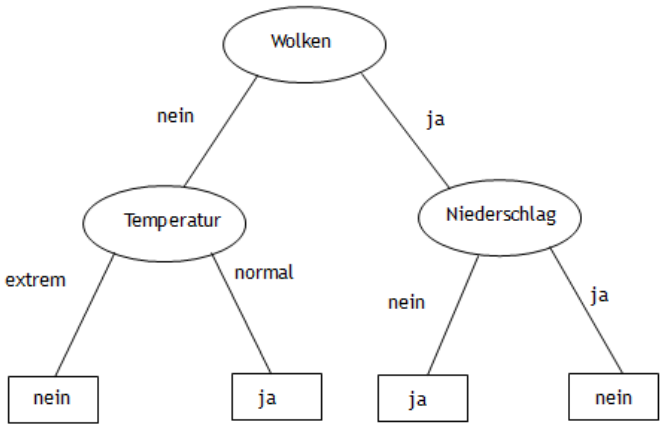

Abbildung 11: Beispiel für einen Entscheidungsbaum zur Frage „Fahre ich mit dem Fahrrad zur Uni ?“

Feststellung, ob es Wolken am Himmel gibt. Wird dies bejaht, hängt das Ergebnis vom Niederschlag ab. Wird es verneint, beeinflusst die Temperatur die Entscheidung. Verfolgt man die verschiedenen Wege zu den Blättern ergeben sich folgende Regeln:

$$IF\ \neg Wolken\ AND\ (Temperatur = normal)\ THEN\ Fahrrad\}$$
$$IF\ Wolken\ AND\ \neg Niederschlag\ THEN\ Fahrrad\}$$

$$IF\ \neg Wolken\ AND\ (Temperatur = extrem)\ THEN\ \neg Fahrrad\}$$
$$IF\ Wolken\ AND\ Niederschlag\ THEN\ \neg Fahrrad\}$$

4.2.7 Fuzzy-Regeln

Fuzzy-Regeln sind im Gegensatz zu den klassichen IF-THEN-Regeln (siehe Abschnitt 4.2.2) eher ungefähre Angaben entsprechend der Bedeutung des englischen Wortes fuzzy (auf deutsch unscharf,verschwommen). Sie besitzen ebenfalls eine IF-THEN-Struktur, jedoch sind die einzelnen Werte,

die ein Attributen haben kann, nicht klar voneinander abgetrennt und werden durch linguistische Ausdrücke dargestellt. Für die Umwandlung von kontinuierlichen Attributen in Fuzzy-Regeln wird beispielsweise eine dreieckige Zugehörigkeitsfunktion verwendet um das Attribut in fünf gleich große Bereiche einzuteilen [KzB13]. Auf Abbildung 12 ist dargestellt wie ein kontinuierliches Attribut, z.B. Länge in cm, das an der x-Achse angetragen ist in fünf Dreiecke, also in die fünf Kategorien „very low“, „low“, „medium“, „high“ und „very high“ unterteilt wird, die sich jeweils überschneiden. Zieht man eine senkrechte Linie an der entsprechenden Stelle des x-Wertes senkrecht nach oben, sieht man anhand der Schnittpunkte mit der Zugehörigkeitsfunktion in welchem Grad (Werte zwischen 0 und 1) der x-Wert tu den entsprechenden Klassen gehört. Das verdeutlicht, das ein bestimmter Wert nicht nur zu einer bestimmten Kategorie gehört, sondern auch in einem bestimmten Grad einer anderen Klasse angehört [Sch93]. Dies ermöglicht die Realität besser abzubilden, da auch dort Attribute häufig nicht eindeutig kategorisierbar sind.

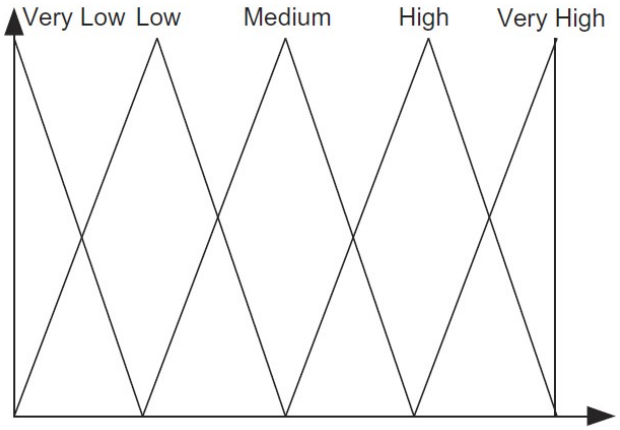

Abbildung 12: Unterteilung eines kontinuierlichen Attributes in fünf fuzzy Bereiche [KzB13]

4.3 Qualität der Regeln

Die Qualität der extrahierten Regeln ist ebenso von Bedeutung, denn das Ziel wäre verfehlt, wenn diese die Qualität der Klassifikation des neuronalen Netzes nicht zufrieden stellen oder zu kompliziert für das menschliche Verständnis sind. Diese lässt sich anhand der folgenden Punkte einordnen [ADT95]:

Genauigkeit Dies beschreibt inwieweit unbekannte Beispiele korrekt klassifiziert werden.

Zuverlässigkeit Regeln können das Verhalten des neuronalen Netzes imitieren, man erhält also das gleiche Ergebnis bei beiden Klassifikationsvorgängen.

Konsistenz Klassifikationsergebnisse aus Regeln und neuronalem Netz sind für unbekannte Beispiele identisch

Komplexität Diese wird anhand der Größe der Regelmenge (Anzahl der Regeln) und an der Größe der Bedingungen (Anzahl der Aussagen) gemessen. Oft wird dieses Kriterium auch als Verständlichkeit bezeichnet. Dieser Begriff ist jedoch aufgrund der unterschiedlichen psycho-visuellen Wahrnehmung des Menschen irreführend, da jeder unterschiedliche Präferenzen hat und somit manche eine größere Regelmenge mit kurzen Bedingungen, andere aber eine kleinere Regelmenge mit größeren Bedingungen bevorzugen.

Jedoch ist es problematisch die Regeln nach allen genannten Kriterien zu optimieren. So ist es unter Umständen nicht möglich die Genauigkeit und die Zuverlässigkeit zu erhöhen, da sich diese beiden Merkmale gegenseitig beeinflussen [Zho04].

4.4 Unterschiedliche Ansätze zur Regelextraktion

Bei den im folgenden untersuchten Verfahren (siehe Abschnitt 6) kann man zwischen drei verschiedenen Ansatztypen unterscheiden:

- *dekompositionell* (engl. decompositional): Hier wird das künstlich neuronale Netz zerlegt und für jedes einzelne Neuron werden Regeln erstellt. Dadurch hat man die Möglichkeit die innere Struktur eines Netzes zu verstehen. Dabei werden am Ende häufig nochmal die Regeln in der Gesamtheit betrachtet und zur besseren Generalisierung werden unter Umständen wieder Regeln entfernt.
- *pädagogisch* (engl. pedagogical): Bei diesem Ansatz wird das künstlich neuronale Netz als Lehrer (oder auch als Orakel bezeichnet) verwendet. Das heißt, es wird verwendet um das Klassifikationsergebnis zu haben und die Regelmenge entsprechend erstellen zu können. Hier wird der Aufbau des Netzes nicht beachtet.
- *eklektisch* (engl. eclectic): Hier werden die beiden vorher genannten Ansätze kombiniert. Es entsteht so eine Mischung aus Zerlegen und Orakel, bei der z.B. Gruppen von Neuronen als Orakel eingesetzt werden.

5 Kriterien zur Bewertung der Verfahren

Die Verfahren zur Regelextraktion werden stets zur besseren Einschätzung anhand einer kurzen Tabelle mit den untenstehenden Kriterien eingeordnet um eine bessere Übersicht der verschiedenen Methoden zu erhalten. Deshalb werden diese Kriterien zuerst kurz beschrieben.

notwendige Voraussetzungen Unter dem Punkt Voraussetzungen werden Bedingungen aufgeführt, die für die Anwendung des Verfahrens notwendig sind. Dies ist auch ein Indiz für die Portabilität des Verfahrens: je weniger Voraussetzungen existieren, desto portabler ist das Verfahren.

Art des Ansatzes Wie schon in Abschnitt 4.4 vorgestellt, gibt es verschiedene Ansätze zur Regelextraktion. Hier wird das jeweilige Verfahren anhand in einen dieser dreien eingeordnet.

verwendeter Trainingsalgorithmus

Form der extrahierten Regeln Hier wird die Form der erzeugten Regeln beschrieben (vgl. dazu Abschnitt {subsection:aufbau-regeln).

Netzstruktur Da manche Verfahren bestimmte Netzstrukturen voraussetzen, beziehungsweise für diese optimiert sind, werden unter diesem Punkt die spezifischen Anforderungen an die Netzstruktur genannt.

Bereich der Eingabewerte Hier wird der mögliche Wertebereich der Eingabewerte angegeben. Bei den Eingabewerten wird zwischen dem kontinuierlichen und diskreten Eingabebereich unterschieden, da bei kontinuierlichen Werten oft eine vorherige Diskretisierung der Werte notwendig ist um dann Regeln zu erstellen.

Einschränkungen Weitere Einschränkungen, die keiner der vorher genannten Kategorien zugeordnet werden konnten, werden unter diesem Stichwort beschrieben.

Komplexität Zur Vergleichbarkeit des Rechenaufwandes der unterschiedlichen Algorithmen, wird ebenfalls die Komplexität des Verfahrens angegeben.

Zusammenfassend kann man sagen, dass je weniger Einschränkungen bei den verschiedenen Kriterien existieren, desto größer ist die Portabilität des jeweiligen Algorithmus, das bedeutet, dass er einfacher auf vielfältige künstliche neuronale Netze angewendet werden kann.

6 Untersuchte Verfahren zur Regelextraktion

Nachdem in den vorherigen Abschnitten die Grundlagen zum Verständnis der Problemdomäne gegeben wurden, werden im Folgenden verschiedene Verfahren beschrieben und anhand der vorher vorgestellten Kriterien (siehe Abschnitt 5) untersucht. Dabei wurde eine Auswahl an vorhandenen Verfahren getroffen, die unterschiedliche Ansätze verfolgen und das neuronale Netz in verschiedene Regeltypen abbilden. Es gibt darüber hinaus eine Vielzahl an Ansätzen, die für bestimmte Anwendungsgebiete

6.1 Extraktion von Regeln aus trainierten neuronalen Netzen [Tsu00]

notwendige Voraussetzungen:	monoton wachsende Ausgabefunktion (wie z.B. sigmoid)
Art des Ansatzes:	dekompositionell
verwendeter Trainingsalgorithmus:	unabhängig, verändert nichts
Form der extrahierten Regeln:	boolesche Funktionen (siehe Abschnitt 4.2.1)
Netzstruktur:	alle möglich
Bereich der Eingabewerte:	kontinuierlich
Einschränkungen:	nur für Units mit monoton steigenden Ausgabefunktionen
Komplexität:	polynomiell

Tabelle 4: Untersuchung der Regelextraktion aus trainierten Netzen nach Tsukimoto [Tsu00]

In diesem Ansatz werden die Units eines neuronalen Netzes durch boolesche Funktionen approximiert. Die Komplexität wäre dann eigentlich exponentiell, aber um diese zu verringern werden Terme niedrigerer Ordnung (das bedeutet boolesche Regeln mit möglichst wenigen Bedingungen) erzeugt, so dass die Komplexität auf polynomiell reduziert werden kann. Um die Genauigkeit bei der Approximation (der Hidden Units) zu erhalten, werden die Abstände zwischen den Units und den Funktionen nicht im ganzen Bereich berücksichtigt, sondern nur bei den Trainingsdaten. Zudem werden nur solche mit hohen Gewichten berücksichtigt.
Der Basisalgorithmus zur Approximation der booleschen Funktionen:
Sei $f(x_1, \ldots, x_n)$ ein Unit eines neuronalen Netzes und $(f_i)(i = 1, \ldots, 2^n)$ die Werte eines Units. Durch die sigmoide Ausgabefunktion liegen die Werte im Intervall $[0, 1]$. Des weiteren sei $g(x_1, \ldots, x_n)$ eine boolesche Funktion mit den Werten $(g_i)(g_i = 0\ oder\ 1, i = 1, \ldots, 2^n)$. Somit kann die boolesche Funktion wie folgt (als Disjunktion) dargestellt werden:

$$g(x_1, \ldots, x_n) = \sum_{i=1}^{2^n} g_i a_i$$

mit dem Atom $a_i = \prod_{j=1}^{n} e(x_j)(i = 1, \ldots, 2^n)$ und $e(x_j) = x_j$ mit $x_j = 1$ bzw. $e(x_j) = \overline{x_j}$ für $x_j = 0$, sowie dem Wert, der dem Bereich des Atoms entspricht:

$$g_i = \begin{cases} 1 & (f_i \geq 0.5) \\ 0 & (f_i < 0.5) \end{cases} \tag{4}$$

Dies (siehe Gleichung 4) ist der Basisalgorithmus, der die euklidische Distanz minimiert. Seine Komplexität ist exponentiell.

Es lässt sich zeigen, dass neuronale Netze für die Werte $\{0, 1\}$ multilineare Funktionen sind (vgl. [Tsu00, S. 380 ff]), deren Raum für eben diese Werte euklidisch ist. Somit kann der polynomielle Algorithmus zur Erzeugung der DNF Formeln aus Termen niedriger Ordnung angewendet werden

(siehe Algorithmus 3).

Algorithmus 3 polynomieller Algorithmus zur Erezeugung von booleschen Funktionen(vgl. [Tsu00, S. 382])

1. Überprüfe nach der Approximation, ob die Terme existieren und beginne dabei mit der niedrigsten Ordnung. Denn ein Term existiert, wenn die Werte des neuronalen Netzes in dem dem Term entsprechenden Bereich größer als 0,5 sind.
2. Verbinde diese Terme um eine DNF Formel zu erhalten.
3. Wiederhole die beiden oberen Schritte bis zu einer bestimmten Ordnung.

Erzeugung der DNF Formeln (siehe Abschnitt 4.2.1):
Dazu werden Terme erzeugt, die zunächst Terme niedrigerer Ordnung und dann bis zu einer begrenzten höherer Ordnung verwenden. Die DNF Form wird durch Disjunktion der generierten Terme erstellt. Zur Verbesserung der Genauigkeit der Regeln werden das trainierte Netz und die Trainingsdaten benötigt. Dabei wird die Distanz zwischen Units und Funktionen im Bereich der Trainingsdaten gemessen und minimieren diese Distant, da dadurch die Funktionen an die Units angepasst werden.

Um komplizierte Regeln zu vereinfachen, müssen unwichtige Attribute entfernt werden. Die Wichtigkeit eines Attributs berechnet sich für den diskreten Wertebereich aus dem Gewichtsparameter des binarisierten Attributs p_i und der Anzahl der Attributswerte des Attributs n:

$$\left(\sum_{1}^{n} p_i^2\right)/n$$

Für wichtige Attribute ist dieser Wert groß und daher werden Attribute bis zu einer bestimmten Anzahl behalten. Diese wird aus dem Fehler zwischen neuronalem Netz und der booleschen Funktion bestimmt, der unter einem festgelegten Schwellwert liegen soll.

Die Komplexität um einen Term m-ter Ordnung zu generieren ist polynomiell $\binom{n}{k}$, also polynomiell von n. Dies gilt bis zur k-ten Ordnung, danach $\sum_{m=1}^{m=k}\binom{n}{m}$. Die Generierung von DNF Formeln ist auch polynomiell zu n. Bis zu den Termen höchster Ordnung ist diese exponentiell: $\sum_{m=1}^{m=n}\binom{n}{m=2^n}$. Somit generiert man zur Reduzierung der Komplexität die Terme nur bis zu einer bestimmten Ordnung. Denn es lässt sich zeigen, dass Terme höherer Ordnung für die Verständlichkeit unnötig sind (siehe [Tsu00, S. 384]), da diese nur vergleichsweise wenige Daten abdecken und somit das Klassifikationsergebnis nur unwesentlich beeinflussen (siehe [LMN93]).

Es besteht zudem eine Erweiterung des Algorithmus für kontinuierliche Bereiche. Dazu werden kontinuierliche boolesche Funktionen gebildet, so dass dann der gleiche Algorithmus angewendet werden kann. Zuerst wird dazu eine Normalisierungsmethode verwendet, die die kontinuierlichen Werte auf den Bereich $[0, 1]$ skaliert. Dazu werden Ausdrücke über direkte und indirekte Proportion, Konjunktion sowie Disjunktion gebildet. Diese erfüllen die Axiome der booleschen Algebra (siehe [Tsu00, S. 386 f]). Vorrangig werden qualitative Aussagen gesucht, also keine genauen Angaben zu den einzelnen Werten, sondern es genügt zu wissen, in welcher Relation sie zueinander stehen. Quantitative Werte müssen daher nicht beachtet werden. Dazu wird die qualitative Norm eingeführt. Somit wird der Abstand zwischen den beiden Funktionen x und A in Abbildung 13 entfernt und die beiden Funktionen x und A sind gleich. Da das neuronale Netz für den Bereich $[0, 1]$ eine multilineare Funktion ist und der Raum der multilinearen Funktionen euklidisch ist, kann das Unit über eine boolesche Funktion mit Verwendung der euklidischen Norm approximiert werden. Zudem ist das Unit eines neuronalen Netzes eine multilineare Funktion in einem qualitativen Ausdruck und der Raum der multilinearen Funktionen ist der euklidische

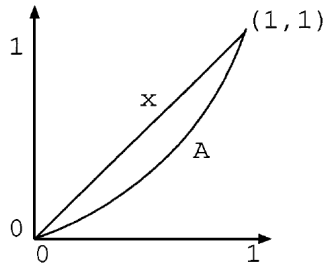

Abbildung 13: Qualitative Norm [Tsu00, S. 386]

in der qualitativen Norm. Somit kann das Unit mit einer kontinuierlichen booleschen Funktion unter Verwendung der euklidischen Norm approximiert werden

Der polynomielle Algorithmus (siehe Algorithmus 3) für den kontinuierlichen Bereich ist derselbe wie für den diskreten. Jedoch unterscheidet sich die Methode zur Auswahl der Attribute. Da hier die Gewichtsparameter p_i nicht binarisiert sind, lassen sich diese sortieren: $|p_1| \geq |p_2| \geq \ldots \geq |p_n|$. Attribute mit kleineren Beträgen als $|p_i|$ werden gelöscht. Dieses Verfahren funktioniert nur für kontinuierliche Werte (nicht für diskrete, da die Auswahl der Attribute bei binären Attributen zu keinem sinnvollen Ergebnis führt). Die Anzahl der Attribute wird über den Fehler zwischen neuronalem Netz und der booleschen Funktion bestimmt.
Tsukimoto konnte durch Tests zeigen, dass der Algorithmus gut für kontinuierliche Attribute funktionert, jedoch können noch keine Regeln extrahiert werden, die Intervalle von Attributen beschreiben, wie z.B. in Gleichung 5 mit den Attributen x,y und der Klasse z.

$$(3 \leq x \leq 5) \wedge (y \leq 0.7) \rightarrow 0.4 \leq z \leq 0.5 \tag{5}$$

Anhand der durchgeführten Test kann eine Genauigkeit von mindestens 95% nachgewiesen werden.

Zusammenfassend können wir sagen, dass dieses Verfahren nicht nur für diskrete, sondern auch für kontinuierliche Attribute geeignet ist und bereits gute Resultate bezüglich der Genauigkeit (für kontinuierliche Attribute) liefert. Jedoch können noch keine Regeln realisiert werden, die Intervalle beschreiben (siehe Gleichung 5).

6.2 Extraktion von M-aus-N-Regeln [Set00]

notwendige Voraussetzungen:	keine
Art des Ansatzes:	eklektisch
verwendeter Trainingsalgorithmus:	Feedforward
Form der extrahierten Regeln:	M-aus-N-Regeln (siehe Abschnitt 4.2.3)
Netzstruktur:	3-schichtiges Feedforward-Netz
Bereich der Eingabewerte:	diskret (-1 oder 1)
Einschränkungen:	keine weiteren
Komplexität:	keine Angaben

Tabelle 5: Untersuchung der Extraktion von M-aus-N-Regeln [Set00]

Die Grundidee dieses Ansatzes besteht darin, den hyperbolischen Tangens auf alle Verbindungen zwischen Eingabe- und verdeckter Schicht anzuwenden um daraus die Aktivierungswerte zu berechnen.

Dabei werden M-aus-N-Regeln (siehe Abschnitt 4.2.3) verwendet. Zur Vereinfachung der Anwendung sind die Eingabewerte auf -1 und 1 beschränkt, sowie die Größe des Netzes auf drei Schichten (vgl. Tabelle 5).

Zunächst wird das Netz trainiert um Kanten mit kleinen Gewichten, die keinen (oder einen vernachlässigbar geringen) Einfluss auf die Klassifizierung haben, beim Training zu entfernen. Dieses abschneiden von Kanten und Blättern wird auch als Prunen bezeichnet. Da dies jedoch bei Netzen mit vielen Neuronen in der verdeckten Schicht oder vielen Eingangsattributen sehr zeitaufwändig sein kann, wird hier die BFGS-Methode (siehe [GK99]), eine Variante des Quasi-Newton-Algorithmus eingesetzt, da sich diese als effizienter erwiesen hat. Der Newton-Algorithmus wird zur numerischen Lösung von nichtlinearen Gleichungssystemen eingesetzt. Dabei wird die Nullstelle der nichtlinearen, stetig differenzierbaren Funktion $f : \mathbb{R} \to \mathbb{R}$ durch Linearisierung ermittelt. Beginnend bei einem Startpunkt $f(x_0)$ wird die Tangente $t(x) = f(x_n) + f'(x_n) \cdot h$ mit $h = x - x_n$ für diesen Punkt berechnet und die Nullstelle der Tangente wird ermittelt, indem $t(x_{n+1}) = 0$ nach x_{n+1} aufgelöst wird:

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$$

Diese Schritte werden so oft wiederholt, bis das Abbruchkriterium greift und die Berechnung beendet wird. Für den mehrdimensionalen Raum $f : \mathbb{R}^n \to \mathbb{R}^n$ wird die Jacobi-Matrix $J(x)$ (enthät alle partiellen Ableitungen von f) verwendet:

$$x_{n+1} = x_n - (J(x_n))^{-1} f(x_n)$$

Um das Newton-Verfahren für Optimisierungsprobleme (finden des Maximums oder Minimums einer Funktion) einzusetzen, wird das Verfahren auf die Ableitung der Funktion angewendet. Somit verändert sich das Update für x_{n+1}, da anstelle der Jacobi-Matrix die Hesse-Matrix $H_f(x)$ (enthält die partiellen Ableitungen 2. Ordnung) und für $f(x)$ der Gradient $\nabla f(\mathbf{x}_n)$ verwendet werden. Es ergibt sich somit:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [H_f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n),\ n \geq 0$$

Jedoch ist hier die Berechnung der Inversen der Hesse-Matrix $[H_f(\mathbf{x}_n)]^{-1}$ aufwändig, weshalb Quasi-Newton-Verfahren entwickelt wurden, die dessen Werte nur annähern und nicht genau berechnen. Dazu wird hier das BFGS-Verfahren (BFGS: Broyden-Fletcher-Goldfarb-Shanno) angewendet. Dieses berechnet das Inverse der Hesse-Matrix $[H_f(\mathbf{x}_n)]^{-1}$ approximativ und spart dadurch Rechenkapazitäten ein. Zudem wird eine Schrittweitensteuerung γ eingeführt, die vermeiden soll, dass sich der Algorithmus zu lange mit kleinen Iterationsschritten an flachen Stellen aufhält:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \cdot [H_f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n),\ n \geq 0$$

Der komplette Algorithmus besteht aus folgenden Schritten:

1. **Trainiere und Prune das Netz**
   Die hyperbolische Tangensfunktion (siehe Abbildung 14)

   $$\tanh(K, x) = \frac{e^{Kx} - e^{-Kx}}{e^{Kx} + e^{-Kx}},\ K > 0$$

   wird auf alle Verbindungen zwischen Eingabe und Hidden Units angewandt. Somit ergibt sich für die Aktivierungswerte der Hidden Units:

   $$\alpha_{ph} = \tanh\left(K_0, \left(\sum_{n=1}^{N} x_{pn} \times \tanh(K_1, w_{nh})\right) + \tau_h\right) \tag{6}$$

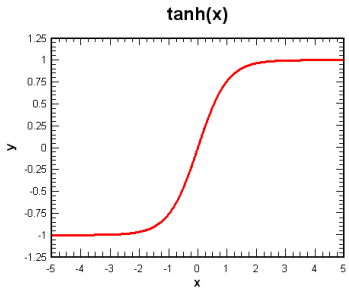

Abbildung 14: hyperbolische Tangensfunktion

N: Anzahl der Eingabe Units,
α_{ph}: Aktivierungswert der Kante zwischen p-tem Input Unit und h-tem Hidden Unit,
x_p: n-dimensionales Eingabepattern (n-tes Element ist x_{pn}),
w_{nh}: Kantengewicht zwischem n-tem Input Unit und h-tem Hidden Unit,
τ_h: Bias des h-ten Hidden Units

Dabei wird die hyperbolische Tangensfunktion

$$\tanh(K,x) = \left(e^{Kx} - e^{-Kx}\right) / \left(e^{Kx} + e^{-Kx}\right), K > 0$$

auf jede Verbindung zwischen Eingabe- und verdeckter Schicht in Gleichung 6 angewendet. Die Vereinfachung der Regelextraktion erfolgt durch Beschränkung der Eingabewerte und Kantengewichte auf -1 und 1.
Die zu minimierende Fehlerfunktion (Minimierung erfolgt mit bereits genannter BFGS-Methode)

$$F(w,v) = -\sum_{p=1}^{P}\sum_{c=1}^{C}(t_{pc}\log\beta_{pc} + (1-t_{pc})\log(1-\beta_{pc})) + P(w,v)$$

P: Anzahl der Muster in der Traininsmenge,
C: Anzahl der Units in der Ausgabeschicht,
H: Anzahl der Units in der verdeckten Schicht,
t_{pc}: erwartete Ausgabe am c-ten Output Unit für Eingabe x_p,
v_{hc}: Kantengewicht zwischem h-tem Hidden Unit und c-tem Output Unit
β_{pc}: Ausgabe des Netzes am c-ten Output Unit für Eingabe x_p
wird mit dieser Straffunktion (mit positivem Strafparameter ϵ)

$$P(w,v) = \epsilon\sum_{h=1}^{H}\left(\sum_{n=1}^{N}\frac{\tanh(K_1,w_{nh})^2}{1+\tanh(K_1,w_{nh})^2} + \sum_{c=1}^{c}v_{hc}^2\right)$$

minimiert. Diese Straffunktion weist eine schnellere Konvergenz als die Standard-Methode der kleinsten Quadrate. Durch die Quadratur verfallen kleinere Kantengewichte schneller als größere. Dies wird solange fortgeführt bis keine Kanten mehr im neutrainierten Netz entfernt werden können, ohne dabei das Klassifikationsergebnis zu beeinflussen.

2. **Gruppieren der Aktivierungswerte zur Vereinfachung der Datenmenge**
   Die Werte der Hidden Units des geprunten Netzes werden mit dem Chi2-Algorithmus zu Clustern geformt (vgl. [LS95]). Das Intervall $[-1,1]$ wird in disjunkte Teilintervalle unterteilt. Dazu werden die Aktivierungswerte in aufsteigender Reihenfolge sortiert und solche mit einem kleinen χ^2-Wert, werden zusammengefügt, solange wie die Aktivierungswerte von Mustern, die zu verschiedenen

Klassen gehören noch unterschieden werden können. Dies wird sowohl für die Units in der Eingabeschicht als auch in der verdeckten Schicht durchgeführt. Da die Anzahl der Regeln abhängig von der Anzahl der Cluster, das Ziel ist es also möglichst wenige Cluster zu haben um eine kompakte Regelmenge zu erhalten.
So kann die Anzahl der Datendimensionen von H auf die Anzahl der nach dem Prunen verbliebenen Units in der verdeckten Schicht $\overline{N}$ reduziert werden.

3. **Generiere Klassifizierungsregeln**
   Aus den gruppierten Aktivierungswerten werden mit dem X2R Algorithmus[1] (vgl. [LT95]) Klassifizierungsregeln erstellt:

   a) Regeln mit den wenigsten Bedingungen werden generiert und die dadurch abgedeckten Beispiele werden markiert, solange bis alle Beispiele durch mindestens eine Regel abgedeckt sind.

   b) Die Regeln werden entsprechend ihrer Klassenlabel gruppiert.

   c) Redundante Regeln in einem Cluster werden geprunt.

   Die Regeln bestehen aus DNF Termen, wobei in jeder Bedingung der Aktivierungswert α enthalten ist. Dabei gilt: $L \leq \alpha < U$ mit L und U als linker bzw. rechte Intervallgrenze, es gilt für die Teilintervalle: $S_i = [L_i, U_i)$, $L_j < U_j$, $L_{i+1} = U_i$, $-1 < L < U < 1$ für alle j

4. **Erzeugung der M-aus-N-Regeln**
   Ersetze die Bedingungen der in Schritt 3 generierten Regeln durch M-aus-N-Bedingungen. Sei C die Disjunktion

$$\left(\bigvee_{I_n \in I_+} (I_n = 1)\right) \bigvee \left(\bigvee_{I_n \in I_-} (I_n = 1)\right),$$

   wenn M der $\overline{N}$ Bedingungen aus C erfüllt sind, dann gilt $L \leq \alpha < U$. Für $\overline{M}$, die Anzahl der nicht erfüllten Bedingungen aus C, gilt:

$$\overline{L} \leq M - \overline{M} \leq \overline{U} \tag{7}$$

   Da $M + \overline{M} = \overline{N}$ erfüllt das Wertepaar $(M, \overline{M})$ mit $M = \lfloor 0.5(\overline{N} + \overline{U}) \rfloor$ und $\overline{M} = \overline{N} - M$ die Bedingung 7. Alle weiteren Wertepaare $(M, \overline{M})$ können durch verringern von M um eins (somit erhöhen von $\overline{M}$), solange die Bedingungen $M - \overline{M} \geq \overline{L}$, $M \geq 0$ eingehalten sind, gefunden werden, .

Das Verfahren erreicht in den vorgestellten Tests mit künstlichen Daten fast 100% Genauigkeit, für reale Daten über 92%. Die Regeln sind zudem aufgrund der relativ geringen Anzahl an Bedingungen und der insgesamt kleinen Anzahl an Regeln einfach gehalten und höchst zuverlässig mit Werten von 99%.

Zusammenfassend können wir feststellen, dass dieses Verfahren sehr gute Ergebnisse in den Test gezeigt hat und das Ziel, einfache Regeln aus dem neuronalen Netz zu extrahieren erreicht wurde. Es wurden durchschnittlich zwischen zwei und fünf Regeln extrahiert, mit Werten zwischen eins und zwei für M. Aufgrund der guten Ergebnisse gilt es zu überprüfen, inwieweit das Verfahren auf andere Netzstrukturen, außer dem hier verwendeten 3-schichtigen Feedforward-Netz verwendet werden kann.

notwendige Voraussetzungen:	keine fehlenden Werte in Trainingsdaten
Art des Ansatzes:	eklektisch
verwendeter Trainingsalgorithmus:	feedforward
Form der extrahierten Regeln:	DNF
Netzstruktur:	mehrschichtiges Netz
Bereich der Eingabewerte:	diskret
Einschränkungen:	keine
Komplexität:	exponentiell

Tabelle 6: Untersuchung der Extraktion mit Entscheidungsdiagrammen [CZ11]

6.3 Regelextraktion aus neuronalen Netzen mit Entscheidungsdiagrammen [CZ11]

Für dieses Verfahren wird zuerst ein mehrschichtiges Perzeptron unter Standardbedingungen trainiert und anschließend in eine gleichwertige Form gebracht, die die gleichen numerischen Ergebnisse liefert wie das ursprüngliche Netz. Jetzt ist es bereits in der Lage Regeln zu erstellen, die für Fälle, die ähnlich zu den gegebenen Eingabewerten sind, die Ausgabe verallgemeinern können. Die Teilregeln, die für jede Trainingsmenge extrahiert wurden, werden gemergt um ein Entscheidungsdiagramm zu bilden, aus dem logische Regeln extrahiert werden können. Dies ermöglicht ein effizientes Zusammenfügen der Teilregeln. Ein weiterer Algorithmus setzt das Entscheidungsdiagramm in boolesche Ausdrücke um.
Dabei wird zur Erleichterung die Analyse des Merkmalsraum auf die Teile nahe der Trainingsbeispiele begrenzt. Chorowski und Zurada nennen dafür folgende Gründe:

- Primäres Ziel ist es von den Daten zu lernen, nicht vom Netz
- Das Netz findet einige Beziehungen in der Trainingsmenge (je mehr wir davon abweichen, desto komplizierter könnte die Entscheidungsoberfläche des Netzes werden und desto unnötiger wird es, diese wirklichkeitsgetreu zu beschreiben)
- NP-Schwere der Regelextraktion hat seinen Ursprung in den Daten, denn wenn das Netz so komplizierte Funktionen lernt, muss die Relation schon in den Daten gewesen sein.

Zur lokalen Regelextraktion wird der Algorithmus LORE (vgl. Algorithmus 4) verwendet. Dieser erfasst die Aktionen des Netzes für gegebene Beispiele in Form von Teilregeln mit einer effizienten Datenstruktur um die Regeln gut manipulieren zu können
Die wichtigsten Aspekte des Algorithmus:

- Klassifizierung des Daten in zwei Klassen $\mathscr{T}$ (true) und $\mathscr{F}$ (false) (der Einfachheit wegen nur zwei). $\mathscr{U}$ gibt an, dass die Regel den Datensatz nicht in unterschiedlichen Klassen spaltet.
- Abschätzen der minimalen und maximalen Netzanregung: Um die wichtigen Merkmale, die zur Klassifizierung ausreichen, herauszufinden, werden dazu alle Werte nach und nach entfernt, die die Klassifizierung nicht beeinflussen.
  - mehrschichtiges Netz: minimale Anregung von Neuronen, die mit Ausgabe-Neuron über positive Gewichte verbunden sind – maximale Anregung von Neuronen, die mit Ausgabe-Neuron über negative Gewichte verbunden sind. Somit erhält man eine obere Grenze für die minimale und eine untere Grenze für die maximale Anregung.

[1] Ein einfacher und schneller Algorithmus, der Regeln basierend auf Datenmengen (diskret und kontinuierlich) generiert. Zur Diskretisierung wird der Chi2-Algorithmus verwendet, der basierend auf der statistischen χ^2-Verteilung die kontinuierlichen Daten diskretisiert.

Algorithmus 4 LORE (local rule extraction) (vgl. [CZ11, S. 2])

fun LORE()
G ← U [start with empty rule]
for all X ∈ Training Set **do**
    [Create new partial rule describing just this sample]
    PR ← derivePR(net,X)
    G ← merge(G,PR)
endfor
[now G correctly classifies all training samples and needs to be extended over whole feature space I]
G ← generalize(G)
Pruned ← prune(G) [optionally further simplify]

  - Ableiten von Teilregeln aus Eingabedaten mit Methode `derivePR(net,x)`: Es ist das Ziel eine möglichst kleine Menge wichtiger Merkmale zu erhalten. Dazu wird eine hierarchische Ordnung der Merkmale erstellt (z.B. entsprechend des Entscheidungsdiagramms) und es wird nach und nach versucht Merkmale aus den Bedingungen einer Regel zu entfernen, ohne dabei das Klassifizierungsergebnis zu verschlechtern.

- Verwendung von (reduzierten) Entscheidungsdiagrammen um das Erkennen von Untergraphen zu vereinfachen (siehe Abschnitt 4.2.4): Aufgrund der geordneten Struktur von Entscheidungsdiagrammen sind Untergraphen gut erkennbar.

Die Hauptschritte für die Merkmalsextraktion mit Entscheidungsdiagrammen sind:

1. Wahl der Merkmalsordnung:
    - Diese hat Einfluss auf die Größe des Entscheidungsdiagramms und die Generalisierbarkeit, denn je besser die Ordnung, desto kleiner das resultierende Entscheidungsdiagramm. Die Wahl der optimalen Ordnung ist NP-hart und Heuristiken sind abhängig vom Anwendungsbereich.
    - Die entwickelte Heuristiken verwendet folgende Annahmen: Merkmale, die oft gemeinsam in Regeln vorkommen, sollten in der Ordnung nahe beisammen liegen und wichtige Merkmale sind weiter oben im Entscheidungsdiagramm.
    - Zur Berechnung der Bedeutung der Merkmale werden die maximale, minimale und durchschnittliche Differenz der Neuronenausgabewerte für alle möglichen Werte eines Merkmals berechnet, wobei letztere sich als beste Wahl für alle Merkmalswerte erwies und deshalb als Defaultwert gesetzt wird. Mit der bereits genannten Methode `derivePR(net,X)` kann daraus eine Teilregel erzeugt werden.
    - Mittels der Heuristiken wird anschließend eine hierarchische Ordnung erstellt, indem zunächst herausragende Merkmale bestimmt werden und die, die oft zusammen auftreten, in ein Cluster kommen. Von jedem wird das herausragendste Merkmal bestimmt und somit ergibt sich die Ordnung.

2. Merge-Operation: Auf zusammenpassende Diagramme wird die Methode `merge(G,PR)` (siehe Algorithmus 4) angewendet. Es tritt ein Fehler auf, wenn die Diagramme nicht zusammenpassen.

3. Generalisierung: Wenn genug Trainingsbeispiele verfügbar sind und alle Teilregeln zusammengefügt wurden, sollte das Diagramm den ganzen Eingaberaum abdecken. Zur Beseitigung von Kanten, die auf $\mathscr{U}$ zeigen, werden in der Methode `generalize(G)` (siehe Algorithmus 4) die Kanten an das häufigste Kind des Knotens gesetzt.

4. Prunen: Das Diagramm wird, unter Beibehaltung (des Großteils) der Ergebnisse, verkleinert, da es ansonsten häufig zu komplex ist. Dazu wird in der Methode `prune(G)` (siehe Algorithmus 4) die Nutzungshäufigkeit von jedem Pfad gezählt, um dann kaum benutzte Pfade auf $\mathscr{U}$ zu richten. Der Generalisierungsvorgang mit `generalize(G)` wird anschließend wiederholt, bis die Größe nicht mehr verändert wird.

5. Disjunktive Normalform (DNF): Um die Regeln in DNF umzuschreiben müssen Artikulationspunkte (Knoten durch die jeder Pfad zu einem bestimmten Endknoten verläuft) im Diagramm gefunden werden. Hiermit können dann die Regeln aufgestellt werden.

Der vorgestellte Algorithmus kann verbessert werden, indem Werte eines Merkmals, die die Klassifizierung nicht beeinflussen, im Entscheidungsdiagramm zusammengefasst werden. Außerdem können für (bezüglich der Hamming-Distanz) benachbarte Datensätze, die nicht zum Bereich der aktuellen Regel passen, neue Teilregeln abgeleitet werden.

Der LORE-Algorithmus behält eine hohe Wiedergabetreue des Netzes auf der Trainingsmenge bei, wobei die Regeln im verbleibenden Merkmalsraum vom Netz abweichen können. Die Struktur des reduzierten Entscheidungsdiagramms unterstützt die Verfahren zum Mergen und zum Generaliseren. Zudem ist es möglich, die Regeln zu vereinfachen ohne Trainingsgenauigkeit zu verlieren.

Dieses Verfahren beruht sehr stark auf den Trainingsdaten, weshalb unserer Meinung nach eine sorgfältig ausgewählter Datensatz für das Training essentiell für das Ergebnis, also die erzeugten Regeln ist. Die Darstellung als Entscheidungsdiagrammes scheint zunächst eine gute Idee zu sein, jedoch sind große Diagramma schwer zu lesen und nachzuvollziehen, da darin dann eine Vielzahl an Regeln stehen. Hier wäre es erstrebenswert eine für den Menschen besser lesbare Form, wie z. B. schon in Form von Entscheidungsbäumen (siehe Abschnitt 4.2.6) zu haben, da hier die einzelnen Pfade im Baum übersichtlicher lesbar sind und es keine Kantenüberschneidungen gibt.

6.4 Rekursive Regelextraktion für Daten mit gemischten Attributen [SBM08]

notwendige Voraussetzungen:	gepruntes trainiertes Netz
Art des Ansatzes:	pädagogisch
verwendeter Trainingsalgorithmus:	backpropagation
Form der extrahierten Regeln:	hierarchische Regeln (siehe Abschnitt 4.2.5)
Netzstruktur:	keine Vorgaben
Bereich der Eingabewerte:	diskret und kontinuierlich
Einschränkungen:	keine
Komplexität:	keine Angaben

Tabelle 7: Untersuchung der Regelextraktion für Daten mit gemischten Attributen [SBM08]

Ziel des Verfahrens ist es Regeln mit diskreten und kontinuierlichen Attributen zu erhalten, wobei die Bedingungen der beiden Regeltypen disjunkt sind (also „kontinuierliche“ und „diskrete Regeln“ voneinander getrennte Bereiche abdecken). Bei unzureichender Genauigkeit werden weitere Regeln (mit diskreten Merkmalen) rekursiv generiert oder es wird eine Hyperebene erzeugt, die nur kontinuierliche Merkmale enthält. Zur Erzeugung der diskreten Regeln wird die Entscheidungsbaummethode C4.5 [Qui93] verwendet. Dazu wird zunächst ein Entscheidungsbaum erstellt, beginnend mit dem Merkmal, dass die Datenmenge am besten nach ihren Klassen trennt. Dafür wird der Informationsgehalt eines jeden Merkmals berechnet und das mit dem höchsten normalisierten Wert ausgewählt (die Normalisierung wird notwendig, da sich die Anzahl der Beispiele nach der Trennung unterscheiden und daher die

Trennungsrate immer relativ zu der neuen Anzahl betrachtet werden muss). Aus diesem Entscheidungsdiagramm lässt sich schließlich eine Regelmenge bilden. Die Regeln sind hierarchisch aufgebaut, dabei sind auf der unteren Ebene nur Kombinationen aus kontinuierlichen Attributen und alle anderen darüber liegenden bestehen nur aus diskreten (siehe Abschnitt 4.2.6). Der Algorithmus lässt sich in vier Schritte aufteilen (vgl. Algorithmus 5):

1. Beliebige Trainings- und Prune-Methode kann angewendet werden. Dadurch werden die Regeln prägnanter und Rauschen wird entfernt.

2. Alle korrekt klassifizierten Daten werden weitergeleitet und Kanten mit kleinen Gewichten werden über eine Straffunktion entfernt.

3. Falls alle diskreten Attribute aus dem Netz entfernt wurden, wird eine Hyperebene

$$\sum_{C_i \in C'} w_i C_i = w_0$$

   erzeugt, die die zwei Beispielgruppen trennt. w_0 und w_i können mit Hilfe von statistischen und maschinellen Lernverfahren (z. B. Regression, Support-Vektor-Maschinen) bestimmt werden (siehe [Bis06]). Wenn mindestens ein diskretes Attribut erhalten bleibt, werden Klassifizierungsregeln nur aus den diskreten Werten erzeugt. Dadurch wird der Eingaberaum in Teilräume, entsprechend den Regeln, unterteilt.

4. Die „Unterstützer“ $support(R_i)$, werden für jede Regel bestimmt - also der prozentuale Anteil an Beispielen, die von der Regel abgedeckt werden. Zudem wird die Fehlerrate $error(R_i)$ jeder Regel überprüft. Danach wird in Abhängigkeit von δ_1 und δ_2 der Unterraum weiter verfeinert.

Somit erzeugt das Verfahren getrennte Regeln für diskrete und kontinuierliche Werte, die leichter zu verstehen sind als gemischte Regeln.

Algorithmus 5 Re-RX(S,D,C) (vgl. [SBM08, S. 301])

Input: Menge von Beispieldaten S mit den diskreten Attributen D und kontinuierlichen Attributen C
Output: Menge von Klassifizierungsregeln

1) Trainiere und Prune das nN mit den Daten S und allen Attributen D und C
2) D' und C' seien die Attribute, die noch im Netz sind. S' seien die vom geprunten Netz korrekt klassifizierten Daten.
3) Falls $D' = \emptyset$, dann erzeuge eine Hyperebene um die Beispiele in S' entsprechend der Werte ihrer kontinuierlichen Attribute C' zu teilen. Ansonsten verwende nur die diskreten Attribute D', erzeuge eine Menge von Klassifizierungsregeln R für die Datenmenge S'.
4) Generiere für jede Regel R_i:
Wenn $support(R_i) > \delta_1$ und $error(R_i) > \delta_2$, dann
   - Sei S_i die Menge der Beispieldaten, die die Bedingung der Regel R_i erfüllen und D_i sei die Menge der diskreten Attribute, die nicht in der Bedingung der Regel R_i enthalten sind
   - Falls $D_i = \emptyset$, dann generiere eine Hyperebene um die Beispiele in S_i entsprechend der Werte ihrer kontinuierlichen Werte C_i zu teilen
   - Ansonsten: Rufe $Re{-}RX(S_i, D_i, C_i)$ auf

Der Algorithmus ist nicht empfindlich bezüglich Overfitting, obwohl er mehr Regeln generiert als andere Algorithmen. Das liegt vermutlich daran, dass zu Begisn das Netz geprunt wird. Insgesamt ist er ausgewogen in Bezug auf Performanz und Interpretierbarkeit beim Klassifizieren von neuen Kreditbewerbern [SBM08, S. 304]. δ_1, δ_2 sind kritische Parameter hinsichtlich Genauigkeit und Komplexität. Durch größere Werte werden weniger und einfachere Regeln generiert. Um die Genauigkeit geringfügig zu verbessern, sind viel mehr Regeln nötig. Jedoch können die erzeugten Regeln für kontinuierliche Werte,

deren Klassen am Besten durch eine nichtlineare Funktion beschrieben werden, nicht die ursprüngliche Genauigkeit des neuronalen Netzes erhalten. Dies liegt an dem verwendeten Re-RX Algorithmus, der nur lineare Klassifizierer produziert um die Komplexität der Regeln zu vermindern und die Verständlichkeit zu erhöhen.

Dieses Verfahren bietet durch die Regeln mit kontinuierlichen und diskreten Attributen einen guten Ansatz um für sehr große Mengen gute Regeln zu erstellen. Aber es wäre nützlich eine Erweiterung zu haben, die für kontinuierliche Attribute auch eine nicht-lineare Trennung ermöglicht, denn bisher ist das nur mit Genauigkeitsverlust möglich.

6.5 Regelextraktion mit genetischen Algorithmen [SNF00]

notwendige Voraussetzungen:	keine
Art des Ansatzes:	eklektisch
verwendeter Trainingsalgorithmus:	
Form der extrahierten Regeln:	IF-THEN Regeln (siehe Abschnitt 4.2.2)
Netzstruktur:	keine Vorgaben
Bereich der Eingabewerte:	diskret
Einschränkungen:	keine
Komplexität:	keine Angaben

Tabelle 8: Untersuchung der Regelextraktion mit genetischen Algorithmen [SNF00]

Dieser Ansatz verwendet einen genetischen Algorithmus um ein gute Netztopologie zu finden, die dann von einem Regelextraktionsalgorithmus verwendet wird. Die Qualität der extrahierten Regeln wird anschließend zurückgeleitet zum genetischen Algorithmus, da dieser Qualität berücksichtigt, wenn er neue mögliche Varianten für Netztopologien generiert. Die Qualität der Topologie wird über die Vorhersagegenauigkeit und die Verständlichkeit bewertet. Genetische Algorithmen eignen sich für schwierige Probleme, wie zum Beispiel bei großen Suchräumen, da ihr Suchverfahren robust ist. Dabei werden mögliche Kandidaten gesucht, die dann wieder neu kombiniert werden um den Stärksten (Besten) zu erhalten. Dazu wird eine Fitnessfunktion gewählt um die Qualität eines Individuums (hier eine Regelmenge) zu ermitteln. Diese gilt es zu optimieren um das Zielproblem zu lösen. Auf die auserwählten Kandidaten werden genetische Funktionen, wie zum Beispiel Mutation oder Crossover, angewendet um eine neue Generation mit möglichst besseren Individuen zu erhalten.
Der Evolutionszyklus besteht aus folgenden Schritten:

- Selektion
- Crossover
- Mutation
- Training
- Regelextraktion
- Evaluation der Regeln
- wieder weiter bei Selektion

Zur Regelextraktion wird der RX-Algorithmus [LSL95] verwendet, da dieser relativ einfach ist und gute Ergebnisse liefert. Der angepasste RX-Algorithmus (siehe Algorithmus 6):

- adaptierte Regelextraktion:

- Diskretisierung der Aktivierungswerte im Hidden Layer durch Clustern
- Sendung aller Kombinationen von Aktivierungswerten durch das Netz, um die Aktivierungswerte der Ausgabeneuronen zu erhalten
- Regelextraktion (IF-THEN) aus der Beobachtung des Zusammenhangs von durchgeleiteten und am Ende erhaltenen Werten
- analog dazu werden Zusammenhänge zwischen Eingabewerten und Werten im Hidden Layer als Regeln verfasst
- Zusammenfügen dieser beiden Regelmengen, sodass die Bedingung nur aus Eingabewerten und die Konklusion nur aus Ausgabewerten des Gesamtnetzes besteht.

- Einhalten der Qualität der Regeln:
  - Um den Algorithmus öfters anzuwenden um die aktuellen Qualitätswerte zurück zu leiten, werden mehrere Mengen dazu benötigt, denn die Evaluation ist nicht nur auf der Testmenge möglich. Dazu wird die Datenmenge in Trainingsmenge, Validierungsmenge und Testmenge unterteilt. Somit wird die Traningsmenge zum trainieren verwendet, die Validierungsmenge zur Evaluation der Qualität und am Schluss (nach Beendigung der Evolution) wird die Qualität der Regeln mit der Testmenge ermittelt. Dies vermeidet zudem Overfitting. Dabei wird ein Konfidenzfaktor eingeführt, der bei Konflikten – wenn Bedingungen von mehreren Regeln erfüllt werden – entscheidet, welche Regel zum Klassifizieren angewendet wird:

    $$CF = \frac{|Ant\&Cons|}{|Ant|}$$

    mit $|Ant\&Cons|$: der Anzahl an Beispielen, die Bedingung und Konklusion einer Regel erfüllen und $|Ant|$: der Anzahl an Beispielen, die die Bedingung erfüllen.
  - Verständlichkeit wurde zur Bewertung der Qualität hinzugefügt.

Zur Berechnung der Fitnessfunktion, die die Eignung der extrahierten Regeln beschreibt, werden die Verständlichkeit (engl. comprehensibility) und die Genauigkeit (engl. accuracy) bestimmt. Letztere wird aus dem Verhältnis der Anzahl der korrekt klassifizierten Beispielen $Correct$ zu allen vorhandenen Daten $Total$ der Validierungsmenge: $Acc = \frac{Correct}{Total}$. Zur Bestimmung der Verständlichkeit wird zunächst die Komplexität einer Regelmenge mit $Complexity = 2R + C$ berechnet(mit R: Anzahl aller Regeln, C: Anzahl der Bedingungen in allen Regeln). Dabei wird die Anzahl der Regeln doppelt gewichtet, da angenommen wird, dass eine hohe Anzahl an Regeln schlechter ist, als die Anzahl der Bedingungen in einer Regel. Daraus kann man die Formel zur Berechnung der Verständlichkeit erstellen, indem die R und C normalisiert werden:

$$Comprehensibility = 1 - \frac{2 * \frac{R}{Max_R} + \frac{C}{Max_C}}{3}$$

mit Max_R: maximale Anzahl an Regeln, die aus einem Individuum generiert wurde, Max_C: maximale Anzahl an Bedingungen, die aus einem Individuum bis jetzt generiert wurde. Für die Fitnessfunktion erhält man schließlich:

$$Fitness = 1 - W_{acc} \cdot Acc + W_{comp} \cdot Comprehensibility$$

(dabei werden die Gewichte W_{acc}, W_{comp} vom Benutzer definiert, der damit die Bedeutung von Vorhersagegenauigkeit und Verständlichkeit beeinflussen kann).
Zur Erzeugung einer Anfangspopulation wird versucht die minimale Anzahl an Neuronen in der versteckten Schicht abzuschätzen, die benötigt werden um eine gute Generalisierung zu erhalten. Das Auswahlverfahren ordnet die einzelnen Neuronen gemäß ihrer „Fitness“ und höher eingestufte Neuronen werden mit höherer Wahrscheinlichkeit für die Reproduktion ausgewählt. Die Reproduktionsverfahren sind:

Algorithmus 6 angepasster RX-Algorithmus(S,D,C) (vgl. [SNF00, S. 134])

Activation value discretization via clustering:
(a) Let $\epsilon(0,1)$. Let D be the number of discrete activation value in the hidden node. Let δ_1 be the activation value in the hidden node for the first pattern in the training set. Let $H(1) = \delta_1$, $count(1) = 1$, $sum(1) = \delta_1$ and set $D = 1$

(b) For all patterns $i = 2, 3, 4, \ldots, k$ in the training set:

- *Let δ be its activation value*

- *If there exista an index J such that*

$$|\delta - H(J)| = \min_{j \in (1,2,3,\ldots,D)} |\delta - H(J)| and |\delta - H(J)| \leq \epsilon$$

then set $count(J) := count(J) + 1$
$sum(J) := sum(J) + \delta$
else $D = D + 1, H(D) = \delta$

(c) Replace H by the average of all activation values that have been clustered into this cluster:

$$H(j) := sum(j)/count(j), j = 1, 2, \ldots, D$$

1. Enumerate the discretized activation values and compute the network output. Generate perfect rules that have a perfect cover of all the tuples from the hidden node activation values to the output values.
2. For the discretized hidden node activation values appeared in the rules found in the above step, enumerate the input values that lead to them, and generate perfect rules.
3. Generate rules that relate the input values and output values by rule substitution based on the results of the above two steps.

- Möglichkeiten für Crossover:
  - Das Kind wird mit allen Verbindungen der Eltern verknüpft.
  - Es wird ein Hidden Unit des besten Elternteils zufällig ausgewählt und mit dem Kind verbunden.
- Möglichkeiten für Mutation:
  - Entfernen oder Hinzufügen eines Units
  - Hinzufügen eines neuen Eingabewerts zusammen mit allen dazugehörigen Verbindungen entsprechend der Netztopologie

Aus den durchgeführten Tests sieht man, dass die Regeln nicht zu komplex sind und daher gut verständlich. Trotz Genauigkeitsverlust (im Vergleich zum Ausgangsnetz) erreicht dieses Verfahren höhere Werte als der C4.5-Algorithmus [Qui93].

Dieses Verfahren ist gerade deswegen interessant, da es die Qualität der bereits extrahierten Regeln berücksichtigt und die Verständlichkeit der Regeln bewertet wird. Denn in den bisher vorgestellten Verfahren werden meist nur die Genauigkeit und Zuverlässigkeit verglichen und zur Verbesserung und zum Vergleich der Verfahren benutzt. Da aber das Ziel der Regelextraktion ist, den Klassifikationsvorgang des neuronalen Netzes für den Benutzer verständlicher zu machen und zu erklären was passiert, ist die Verständlichkeit der Regeln ein wichtiger Aspekt. Wobei es sicherlich auch andere Möglichkeiten gäbe diese zu berechnen, indem zum Beispiel die durchschnittliche Anzahl an Bedingungen pro Regel interessant wären.

6.6 Regelextraktion via Entscheidungsbauminduktion [ST01]

notwendige Voraussetzungen:	keine
Art des Ansatzes:	dekompositionell
verwendeter Trainingsalgorithmus:	egal
Form der extrahierten Regeln:	IF-THEN-Regeln (siehe Abschnitt 4.2.2)
Netzstruktur:	egal
Bereich der Eingabewerte:	diskret und kontinuierlich
Einschränkungen:	keine
Komplexität:	keine Angaben

Tabelle 9: Untersuchung der Regelextraktion via Entscheidungsbauminduktion [ST01]

Dieses Verfahren ist unabhängig von Netzstruktur und Trainingsverfahren. Auf ein trainiertes Netz wird der CRED-Algorithmus (continous/discrete Rule Extractor via Decision tree tnduction, siehe Algorithmus 7) angewendet. Dazu werden im ersten Schritt die Aktivierungswerte des Ausgabeschicht für eine bestimmte Anfrage, wie z.B. welchen Wert haben die Aktivierungswerte um die Daten einer bestimmten Klasse zu zuweisen (bei diskreten Daten) oder in welchem Intervall liegen die Ausgabewerte (für kontinuierliche Variablen). Dabei werden kontinuierliche Aktivierunsgwerte für die anschließende Entscheidungsbauminduktion in diskrete Klassen übertragen. In Schritt zwei werden aus den erzeugten Entscheidungsbäume Regeln ausgelesen (siehe Abschnitt 4.2.6) und nutzlose Literale und bereits durch andere Regeln abgedeckte Regeln werden entfernt um diese Regeln zu vereinfachen. Dies wird für die Eingabe- und verdeckte Schicht wiederholt Demnach wird das neuronale Netz zerlegt und basierend auf den Aktivierungsmodellen zwischen Eingabe- und verdeckter Schicht, sowie zwischen verdeckter und Ausgabeschicht werden Entscheidungsbäume erzeugt. Die Gesamtregeln entstehen durch Verbinden der beiden Regelmengen.

Algorithmus 7 CRED Algorithmus (vgl. [ST01, S. 1871])

1. Lege eine Zielklasse fest und bestimme das Zielmuster des Ausgabeunits im trainierten neuronalen Netz
2. Baue einen Baum für die verdeckte und Ausgabeschicht unter Verwendung des Ziel- und Aktivierungsmusters der Hidden Units, extrahiere Zwischenregeln aus dem Baum und zerlege das Netz in grundlegende Funktionen
3. Bilde für jede Funktion einen Baum für die Verbindungen von Eingabe- zu verdeckter Schicht und extrahiere Eingaberegeln
4. Durch das Ersetzen der Zwischenregeln durch die in Schritt 3 generierten Eingaberegeln erhält man die Gesamtregeln
5. Führe die Gesamtregeln zur Vereinfachung mit Hilfe eines Mergekriteriums zusammen

Da das Verfahren die Muster aus der verdecktenn Schicht mit Entscheidungsbauminduktion diskretisiert, werden nur notwendige Grenzen erzeugt und es sind keine besonderen Lernalgorithmen notwendig. Zudem können durch die Verwendung der Entscheidungsbäume diskrete und kontinuierliche Variablen verwendet werden.

Um Regeln um kontinuierliche Werte zu erweitern, wird ein angepasster Hill-Climbing-Algorithmus eingesetzt, der mittels Heuristiken und schrittweiser Annäherung eine optimale Lösung findet. Dazu wird für die Kombination von zwei Regeln R_i und R_j (wobei $R_i \neq R_j$) aus der Regelmnege $\mathscr{R}$ der Integrationsverlust mit einem Integrationsmaß berechnet. So wird die unbedenklichste Integration durchgeführt unter Einhaltung eines gegebenen Schwellwerts für den Genauigkeitsverlust. Zudem werden redundante Regeln entfernt. Dies wird auf alle Regelmengen angewendet. Zusätzlich werden Regeln verallgemeinert, wenn bei Regeln mit unterschiedlichen Bedingungen, die gleiche Konklusion folgt (siehe Abbildung 15).

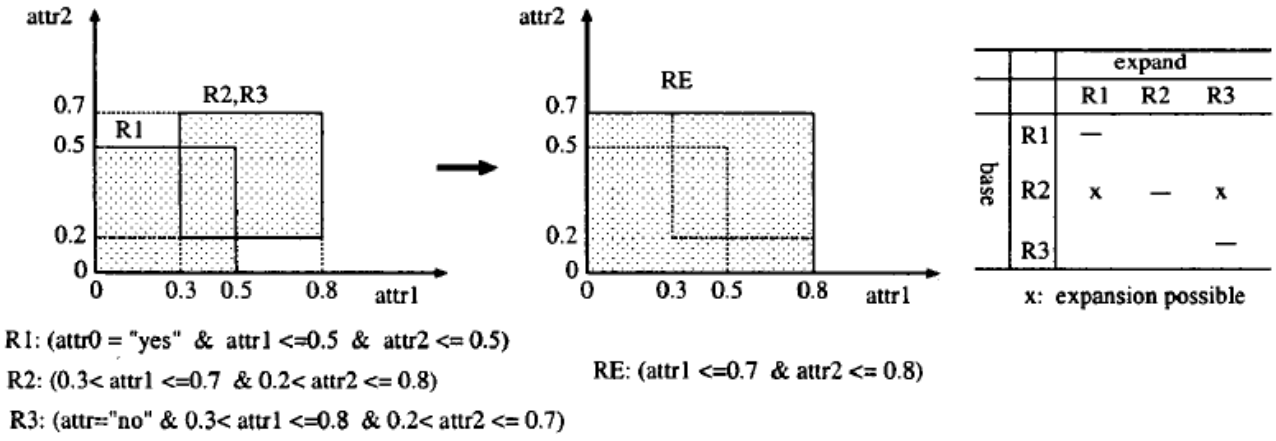

Abbildung 15: Veranschaulichung der Regelexpansion für 3 Regeln R_1, R_2, R_3 [ST01, S. 1873]

Das Verfahren hat den Vorteil, dass es für diskrete und kontinuierliche Variablen gleichzeitig funktioniert. Zudem ist es flexibel, in dem Sinne, dass es nicht von der Netzstruktur oder Trainingsverfahren abhängt. Jedoch zeigt das Ergebnis aus den Tests, dass für kontinuierliche und diskrete Attribute bei kontinuierlichen Klassen die Genauigkeit unter 90% liegt. Dahingegen ist die Genauigkeit ansonsten für

(nur) kontinuierliche Attribute bei mindestens 98%.

6.7 Extraktion von Prioritätsregeln basierend auf Statistik [ZCC00]

notwendige Voraussetzungen:	keine
Art des Ansatzes:	pädagogisch
verwendeter Trainingsalgorithmus:	egal
Form der extrahierten Regeln:	hierarchische Regeln (siehe Abschnitt 4.2.5
Netzstruktur:	egal
Bereich der Eingabewerte:	diskret und kontinuierlich
Einschränkungen:	nein
Komplexität:	keine Angaben

Tabelle 10: Untersuchung der Regelextraktion basierend auf Statistik [ZCC00]

Dieses Verfahren erstellt die Regeln basierend auf statistischen Verfahren. Zusätzlich wird die Anzahl der Bedingungen in den Regeln beschränkt um die Verständlichkeit zu erhöhen. Dazu wird zunächst das neuronale Netz trainiert und verwendet um Klassifizierungsergebnisse für alle Attributskombinationen zu erhalten. Bei kontinuierlichen Daten muss hier eine entsprechend hohe Frequenz zum Abtasten gewählt werden. Anschließend werden die Werte mit einer Variante des ChiMerge-Algorithmus diskretisiert (benachbarte Gruppen mit dem kleinsten χ^2-Wert werden zusammengefügt bis der χ^2-Wert für alle Cluster einen bestimmten Schwellwert überschreitet). Durch diesen Ansatz werden unnötige Diskretisierungen vermieden und die kontinuierlichen Werte können unabhängig voneinander diskretisiert werden. Damit ist gemeint, dass eine unterschiedliche Anzahl an diskretisierten Werten und eine unterschiedliche Größe der diskretisierten Werte möglich sind. Jedoch werden dadurch Abhängigkeiten zwischen den einzelnen Attributen schlechter erkannt.

Zur Regelerzeugung werden zunächst versucht möglichst einfache Regeln zu finden. So wird zuerst nur für einzelne diskrete Attribute überprüft, ob Beispiele damit klassifizierbar sind, indem überprüft wird, ob alle Beispiele mit einem bestimmten Wert für ein Attribut zur gleichen Klasse gehören. Wenn ja, wird die entsprechende Regel zur Regelmenge $\mathscr{R}$ hinzugefügt. Wurden hier alle überprüft, werden anschließend Kombinationen von zwei Attributen verwendet. Dies wird bis zu einer bestimmten Stufe fortgeführt und dabei werden immer komplexere Regeln erstellt. Damit die Regeln noch gut verständlich für den Menschen sind, wird dies nur bis maximal drei Bedingungen pro Regel fortgeführt. Dadurch wird zwar die Anzahl der Regeln steigen, aber nach Meinung von Zhou et al. [ZCC00] führt dies zu einer besseren Verständlichkeit. Werden hier keine Regeln mehr gefunden, werden die kontinuierlichen Werte diskretisiert und es wird damit versucht Regeln zu finden. Alle bereits abgedeckten Beispiele werden jeweils aus der Testmenge $\mathscr{S}$ entfernt.

Zudem erhalten früher gefundene Regeln, also solche mit weniger Bedingungen, eine höhere Priorität als später gefundene. Dadurch ist es einfacher allgemeinere Regeln zu finden und eine Überprüfung der gefundenen Regeln auf Redundanz am Ende ist nicht notwendig. Zum Schluss werden Regeln mit der gleichen Konklusion zusammengesetzt. Da durch die Diskretisierung der kontinuierlichen Attribute die Dimension des noch unbekannten Attributsraums erhöht wird, können dadurch die Regeln nur noch für Teile gültig sein, da der Raum ungünstig verzerrt wurde. Deshalb wird jede neue Regel r zunächst auf ihre Zuverlässigketi überprüft. Dazu wird die Anzahl der Fälle, für die die Regel r und das neuronale Netz die gleiche Klassifizierung treffen, verglichen und wenn diese ausreichend erscheint (über einem festgelegten Schwellwert δ liegt), wird die Regel r zu $\mathscr{R}$ hinzugefügt. Die Anzahl der extrahierten Regeln kann somit über den Wert für δ reguliert werden, denn je kleiner δ, desto mehr Regeln werden gefunden.

Algorithmus 8 STARE-Algorithmus (vgl. [ZCC00, S. 404])

1. Erzeuge eine Beispielsmenge $\mathscr{S}$ mit dem neuronalen Netz $\mathscr{N}$.
2. Erstelle eine neue Regel r basierend auf der entsprechenden Teilmenge $\mathscr{H}$ eines kategorischen Attrbitues, gehe zu Schritt 3. Wenn keine Teilmenge existiert, gehe zu Schritt 5.
3. Erstelle m Beispiele mit $\mathscr{N}$. Wenn r's Zuverlässigkeit über dem Schwellwert δ liegt, dann erweitere die Regelmenge $\mathscr{R}$ und gehe zu Schritt 4. Ansonsten wird r verworfen und gehe zu Schritt 2.
4. Entferne alle Beispiele, die durch r abgedeckt werden, aus $\mathscr{S}$. Wenn $\mathscr{S}$ leer ist, dann gehe zu Schritt 6. Ansonsten gehe zu Schritt 2.
5. Wenn alle kontinuierlichen Attribute diskretisiert wurden, gehe zu Schritt 6. Ansonsten wird das kontinuierliche Attribut das am besten Cluster unterteilen kann diskretisiert, gehe zu Schritt 2.
6. Verbinde alle Regeln, die die gleiche Konklusion und eine entsprechende Priorität haben.

In Tests konnten Zhou et al.[ZCC00] zeigen, dass der STARE-Algorithmus immer eine bessere Zuverlässigkeit als der Vergleichsalgorithmus von Craven und Shavlik [WCJWS] und in den meisten Testfällen auch eine höhere Genauigkeit hat, als der zum Vergleich verwendete C4.5 Algorithmus [Qui93]. Bei beiden Tests hat der STARE-Algorithmus immer eine höhere Anzahl an Regeln extrahiert als die beiden Vergleichsalgorithmen.

Zusammenfassend stellen wir fest, dass das Verfahren das Ziel für den Menschen verständliche Regeln herzustellen klar als Ziel verfolgt und flexibel einsetzbar ist, da keine Beschränkung bezüglich des Traininsalgorithmus und der Netzform gibt. Wobei es fraglich ist, ob drei Bedingungen pro Regel, was mitunter zu einer großen Regelmenge führen kann wirklich sinnvoll und besser verständlich ist. Hier wäre es sinnvoll zu überprüfen, ob durch eine weniger strikte Regel und somit doch mehreren Bedingungen eine kompaktere Regelmenge erzeugbar wäre. Außerdem könnte die Abhängigkeit zwischen den verschiedenen Attributen noch ausgenutzt werden um die Regelmenge zu reduzieren.

6.8 Extraktion von Fuzzy-Regeln [KzB13]

notwendige Voraussetzungen:	keine
Art des Ansatzes:	eklektisch
verwendeter Trainingsalgorithmus:	Differential-Evolution-Algorithmus [SP97]
Form der extrahierten Regeln:	Fuzzy-Regeln in KNF (siehe Abschnitt 4.2.7
Netzstruktur:	Feedforward
Bereich der Eingabewerte:	diskret und kontinuierlich
Einschränkungen:	keine
Komplexität:	keine Angaben

Tabelle 11: Untersuchung der Extraktion von Fuzzy-Regeln [KzB13]

Die Fuzzy-Regeln werden aus dem neuronalen Netz mit dem DIFACONN-miner Algorithmus (siehe Algorithmus 9) extrahiert. Dieser besteht im Wesentlichen aus vier Schritten:

1. **Wissensbasis**
   Diese besteht zum einen aus einer Datenbank, wobei kontinuierliche Variablen in fünf Label mittels

der dreieckigen Zugehörigkeitsfunktion fuzzifiziert werden (siehe Abschnitt 4.2.7). Kategorische Attribute werden direkt in binäre Form gebracht und so in der Datenbank abgespeichert. Zum anderen gibt es eine Regelbasis mit Fuzzy-Regeln in konjunktiver Normalform (siehe Abschnitt 4.2.1, die folgendes Schema haben:

$$R_i : \textbf{IF } \left(Attr_1 \textbf{ is } \widehat{A^i_{1,1}} \textbf{ or } Attr_1 \textbf{ is } \widehat{A^i_{1,2}}\right) \textbf{ and } (Attr_2 \textbf{ is } q \textbf{ or } p) \textbf{ and } \ldots \textbf{ and } \left(Attr_n \textbf{ is } \widehat{A^i_{n,5}}\right) \textbf{ THEN } C_j \tag{8}$$

mit R_i die Regel i bezeichnet und $Attr_i$ die Attribute der Regel i sowie n die Anzahl der Attribute einer Regel. p und q sind die Werte der kategorischen Attribute, $\widehat{A^i_{k,l}}$ ist das Label für kontinuierliche Attribute für die i-te Regel und das k-te Attribut mit Labelwert l. C_j ist das Klassenlabel. Die Regelmenge wird aus den geeignetsten Regeln gebildet und jede Regel wird binär kodiert (siehe Tabelle 12). Dazu wird für jeden Wert eines kategorischen Attributes entweder eine 0 gesetzt, wenn es nicht zutrifft oder eine 1, wenn das Attribut den entsprechenden Wert hat. Dies ergibt für die immer gleiche Abfragereihenfolge eine Folge von 0 und 1 für alle Datensätze.

2. **Training der neuronalen feedforward Netze**
   Wie bereits in Abschnitt 3.4 beschrieben, erfolgt die Anpassung der Kantengewichte über die Minimierung des Fehlers. In diesem Ansatz wird ein Differential-Evolution-Algorithmus verwendet [SP97], der eine stochastische populations-basierte Optimierungsmethode anwendet und damit ein feedforward Netz mit einer verdeckten Schicht trainiert. Neben dem normalerweise verwendetem Kriterium, dem Fehler (F), werden zusätzlich die Trainingsgenauigkeit (TG) und die Anzahl der Regeln (AR) mitbetrachtet. Das Abbruchkriterium des Algorithmus ist gegeben über die maximale Anzahl an Generationen. Sofern diese noch nicht erreicht wurde, werden neue Regellisten durch Mutationen, Kreuzungen und Selektion erstellt, wobei die beste Lösung, die mit der geringsten Eignung in der letzten Iterationen ist.

3. **Erzeugung der Regellisten**
   Der DIFACONN-miner Algorithmus erzeugt für jede erhaltene Gewichtungsmatrix (der Kantengewichte) eine Regelliste mit linguistischen Ausdrücken mittels des TACO-Regelextraktions-Algorithmus [zBKY09]. Dieser berücksichtigt sowohl die Sensitivität S_e, also den Anteil von korrekt klassifizierten positiven Beispielen als auch die Genauigkeit S_p, also den Anteil korrekt klassifizierter negativer Beispiele. Die Qualität lässt sich damit wie folgt bestimmen:

$$Quality = S_e \times S_p = \frac{true\ \ positive}{true\ \ positive + false\ \ negative} \times \frac{true\ \ negative}{true\ \ negative + false\ \ positive}$$

   Diese wird verwendet um Regeln mit einer hohen Qualität als Kandidaten abzuspeichern. Nach Beendigung der Regelerzeugung wird die Regelliste, ausgehend von der geeignetsten Regel, generiert. Als Abbruchkriterium gilt, wenn 85% der Trainingsmenge klassifiziert wurden oder es keine Regeln mehr gibt. Daneben wird die Trainingsgenauigkeit und die Anzahl der Regeln bestimmt.

4. **Evaluation der Eignung**
   Über eine Zielfunktion, die die Trainingsgenauigkeit maximiert und die Anzahl der Regeln sowie den Fehler im neuronalen Netz minimiert, wird die Gesamteignung berechnet. Dazu wird die Methode der kleinsten Abweichungen verwendet, die als Lösung einen Kompromiss aus den einzelnen Optimierungsparametern findet. Die Gesamteignung der Regeln wird mit Hilfe des jeweils angestrebten Wertes $f^{max}(TA) = 100\%$, $f^{min}(E) = 0$ und $f^{min}(AR) =$ Anzahl der Klassen, sowie des jeweils schlechtesten Wertes für jeden Parameter $f^0(TA) = 0\%$, $f^0(E) = 0.999$ und $f^{min}(AR) =$ Vielfaches der Klassenanzahl, berechnet. Man erhält folgende Eignungsfunktion:

$$Eignung = \frac{f_1^{max}(TG) - f_1(TG)}{f_1^{max}(TG) - f_1^0(TG)} + \frac{f_2^{min}(F) - f_2(F)}{f_2^{min}(F) - f_2^0(F)} + \frac{f_3^{min}(AR) - f_3(AR)}{f_3^{min}(AR) - f_3^0(AR)}$$

Algorithmus 9 DIFACONN-miner Algorithmus (vgl. [KzB13])

begin
  Datenmenge fuzzifizieren
  Parameter und Parametergrenzen initialisieren
  repeat
    für alle Chromosome einer Population
    begin
      Wähle drei untereinander unterschiedliche Chromosome und das aktuelle aus
      Mache eine Mutation
      Mache eine Kreuzung
      Berechne den Fehler des Netzes
      Wende den TACO-basierten Regelerzeugungsalgorithmus auf die neuen Chromosomen an um eine Regelliste zu erhalten
      Berechne die Eignung der Chromosome
      Treffe eine Auswahl
    end
  until maximale Anzahl an Generationen ist erreicht
  die extrahierten Regeln auf der fuzzifizierten Testdatenmenge
end

Niederschlag (kategorisch)			Sonne (kontinuierlich)			Wind (kontinuierlich)				Fahre Fahrrad
Nieselregen	Schnee	Hagel	keine	gering	stark	kein	wenig	stark	sehr stark	Ja/Nein
1	0	0	0	1	0	1	1	0	0	1
Regel: **WENN** Niederschlag ist Nieselregen **UND** Sonne ist gering **UND** Wind ist kein **ODER** wenig **DANN** fahre Fahrrad.										

Tabelle 12: Erzeugung der binären Form einer Fuzzy-Regel [KzB13]

Das Verfahren wurde auf sechs verschiedenen Testdatensätzen mit unterschiedlichen Eigenschaften getestet und mit drei anderen Algorithmen verglichen [KzB13, S. 943f]. Dabei wurde das Verfahren der zehnfachen-Kreuzvalidierung angewendet. Hierbei konnte gezeigt werden, dass die Genauigkeit, sowohl für die Trainings- als auch die Testdaten stets besser als bei den anderen Verfahren war. Die Anzahl der Regeln erreicht nie die minimale Anzahl, liegt aber immer im mittleren Bereich zwischen bestem und schlechtestem Wert aus den anderen Verfahren.

Der DIFACONN-Algortihmus erreicht zwar gute Werte für die Genauigkeit, jedoch sehen wir, dass die extrahierte Regelmenge noch verbesserungsfähig ist, denn da die Regeln oft mehr als fünf Bedingungen haben, wird die Lesbarkeit der Regeln deutlich erschwert und die Regelmenge mit gleichzeitig mehr als zehn Regeln unübersichtlich.

7 Vergleich der Verfahren zur Regelextraktion

Im Folgenden werden die in Kapitel 6 vorgestellten Verfahren anhand verschiedener Aspekte verglichen.

7.1 Art des verwendeten Ansatzes

Die vorgestellten Verfahren verwenden drei verschiedene Arten von Ansätzen (siehe Abschnitt 4.4). Einen dekompositionellen Ansatz, der die einzelnen Verbindungen des neuronalen Netzes betrachtet, verfolgen die Verfahren zur Extraktion von booleschen Regeln (siehe Abschnitt 6.1) und zur Extraktion via Entscheidungsbauminduktion (siehe Abschnitt 6.6). Diese beiden Verfahren sind sowohl für diskrete und kontinuierliche Attribute anwendbar und zeigen den Tests Genauigkeiten von mindestens 95 %.

Die Verfahren für gemischte Attribute mit dem Re-RX-Algorithmus(siehe Abschnitt 6.4) und der STARE-Algorithmus basierend auf Statistik (siehe Abschnitt 6.7) verwenden den pädagogischen Ansatz und beachten deshalb die Struktur des Netzes nicht. Diese beiden Verfahren extrahieren beide hierarchische Regeln, die in der Reihenfolge entsprechend ihrer Priorität angewendet werden. Beide Verfahren befassen sich mit der Verständlichkeit der extrahierten Regeln und nehmen dabei auch einen Genauigkeitsverlust in Kauf, wobei der Re-RX-Algorithmus nur eine Genauigkeit von maximal 75,07% im durchgeführten Test erreicht. Dabei hat eine höhere Fehlerrate von 0,1 nur eine Verschlechterung auf eine Genauigkeit von 72,43% zur Folge. Der STARE-Algorithmus hat in einem von sechs Testfällen eine Genauigkeit von 78,7%, für die restlichen aber immer über 88.9%. Da die schlechteste Genauigkeit für einen Datensatz mit den meisten Klassen, 10, erreicht wird, ist dies möglicherweise der Grund dafür. Es ist also kein Zusammenhang zwischen dem pädagogischen Ansatz und der Genauigkeit der Regeln erkennbar.

Schließlich verwenden das Verfahren zur Extraktion von M-aus-N-Regeln (siehe Abschnitt 6.2), das Verfahren mit Entscheidungsdiagrammen (siehe Abschnitt 6.3), die genetischen Algorithmen (siehe Abschnitt 6.5) und der DIFACONN-Algorithmus (siehe Abschnitt 6.8) einen eklektischen Ansatz, eine Mischform der beiden genannten Ansätze. Die Verfahren zeigen alle eine Genauigkeit von mindestens 85% bis hin zu 100%. Diese Ergebnisse sind abhängig von den Eigenschaften der Testdatensätze.

Anhand des Vergleichs der Genauigkeit der Regeln kann kein Rückschluss darauf geschlossen werden, welche Art des Ansatzes zum Erreichen einer hohen Genauigkeit besser geeignet ist. Viel mehr scheint es auf die Beschränkungen bezüglich der extrahierten Regeln, also z.B. eine begrenzte Anzahl an Bedingungen pro Regel, anzukommen.

7.2 Methoden zur Vereinfachung der Regelextraktion

Die in Abschnitt 6 vorgestellten Verfahren zur Regelextraktion lassen sich nicht nur anhand der extrahierten Regeln unterscheiden, sondern auch in den Verfahrensweisen, also wie wird vorgegangen um die Regeln zu extrahieren. Interessant hierbei ist der Aspekt, wie Methoden zur Vereinfachung angewendet werden. Es lassen sich folgende Ideen unterscheiden:

- Es werden nur Regeln bis zu einer bestimmten Ordnung erzeugt, also die Anzahl der Bedingungen pro Regel ist beschränkt (siehe Abschnitte 6.1, 6.7).
- Das neuronale Netz wird geprunt, wobei Kanten mit geringen Gewichten, die kaum Einflsus auf die Klassifikation haben, entfernt werden und so die Netzstruktur vereinfacht wird. Dies wird sowohl für die Extraktion von M-aus-N-Regeln verwendet, was ein eklektischer Ansatz ist (siehe Abschnitt 6.2) als auch für das Verfahren mit den gemischten Attributen, was ein pädagogischer Ansatz ist (siehe Abschnitt 6.4).
- Die Netztopologie wird so angepasst, dass daraus bessere Regeln, im Sinne von genau und verständlich, gefunden werden können. Dieser Ansatz wird vom Verfahren mit den genetischen Algorithmen (siehe Abschnitt 6.5) verfolgt.

- Die erzeugten Entscheidungsdiagramme werden generalisiert, indem selten benutzte Pfade entfernt werden (siehe Abschnitt 6.3).
- Das dreischichtige Netz wird aufgeteilt in Eingabe- und verdeckte Schicht sowie verdeckte und Ausgabeschicht, wobei aus diesen beiden Teilen Regeln extrahiert werden, die dann zusammengeführt werden (siehe Abschnitt 6.6).
- Der Trainingsalgorithmus des neuronalen Netzes wird bereits auf eine bestimmte Anzahl an Iterationen begrenzt, so dass das Netz nicht „übertrainiert“ wird (siehe Abschnitt 6.8).

Die verwendeten Methoden haben als Ziel, die Menge der extrahierten Regeln zu begrenzen, den Aufwand der Extraktion zu verringern oder die Verständlichkeit der Regeln zu verbessern. Jedoch ist dies teilweise je nach Methode mit zusätzlichem Aufwand verbunden, abhängig davon, ob die Methode in einen bereits vorhandenen Schritt integriert wird oder als neuer eigener Schritt dem Regelextraktionsverfahren hinzugefügt wird. So ist es bezogen auf den Aufwand sinnvoll er bereits den Lernalgorithmus zu beschränken als später das fertig trainierte Netz in einem Extraschritt nochmal zu prunen oder erst viele Regeln zu extrahieren, die dann wiederum gekürzt werden.

7.3 Form der extrahierten Regeln

Die vorgestellten Verfahren verwenden unterschiedliche Typen von Regelmengen (siehe Abschnitt 4.2) zur Regelextraktion, die letztlich alle auf der booleschen Logik beruhen. Dabei sind die Formen, mit Ausnahme der Fuzzy-Regeln, eigentlich gleich bezüglich ihrer Aussage, die lautet: Wenn diese deterministischen Bedingungen erfüllt sind, wird der Datensatz jener Klasse zugewiesen. Jedoch bleibt hier unberücksichtigt, dass die Klassen nicht immer deutlich voneinander trennbar sind, weshalb Fuzzy-Regeln eine realistischere Darstellung bilden.

Hier wäre es zudem interessant die verschiedenen Formen an Regelmengen auf ihre Verständlichkeit aus Sicht des Menschen zu untersuchen bzw. die Ergebnisse hieraus verstärkt zu berücksichtigen, denn die meistgenannte Motivation und das Ziel der Regelextraktion aus neuronalen Netzen ist es, ebendiese komplexen neuronalen Netze für den menschlichen Benutzer verständlicher zu machen.

8 Ausblick

In dieser Arbeit wurden Verfahren zur Extraktion von Regeln aus neuronalen Netzen vorgestellt. Dabei ist erkennbar, dass es sowohl unterschiedliche Ansätze zur Extraktion der Regeln gibt, als auch unterschiedliche Regelformen verwendet werden. Dabei wurde auch verdeutlicht, dass die Regelextraktion den Einsatz von neuronalen Netzen fördert, da die Netze dann zwar zur Klassifikation verwendet werden können, aber zusätzlich zur Erklärung noch durch die Regeln für den Menschen verständlicher dargestellt werden können.

Dabei ist nicht ganz klar, in wie weit die extrahierten Regeln mit dem neuronalen Netz übereinstimmen sollen. Oft werden Genauigkeit bezüglich der Klassifikationsergebnisse und Verständlichkeit der Regeln als Kriterien genannt, jedoch bleibt hier die Frage unbeantwortet, inwieweit die Regeln nur das gleiche Klassifikationsergebnis haben sollen oder vielleicht auch den internen Vorgang des neuronalen Netzes beschreiben sollen, also auch eine ähnliche Vorgehensweise zur Klassifikation haben. Es gilt also die Anforderungen an die Regelextraktion besser zu definieren.

Manche der vorgestellten Verfahren erwarten eine bestimmte Struktur des neuronalen Netzes, andere sind auf beliebige Netze anwendbar. Da die Extraktion der Regeln rechenaufwändig ist, wäre hier zu überprüfen, ob es Sinn macht und sich lohnt, auf Kosten der Portabilität effizientere Verfahren zu entwickeln, indem die Verfahren nur für bestimmte Netzstrukturen optimiert werden.

Die bessere Verständlichkeit der Regeln wird häufig als Motivation für die Regelextraktion genannt. In den vorgestellten Verfahren wurden unterschiedliche Arten von Regelmengen extrahiert und es wurden auch Beschränkungen an die Regeln angewendet um eine bessere Verständlichkeit zu gewährleisten. Hierbei wäre es interessant, welche Arten von Regeln wirklich am besten für den Menschen verständlich sind oder vielmehr, welche Kriterien für das menschliche Verständnis von Bedeutung sind. Dabei gilt es zu überlegen, ob nicht weitere Darstellungen in Form von graphischen Visualisierungen oder Animationen sinnvoll einsetzbar sind. Dazu ist es auch überlegenswert, ob Interaktion ein zweckmäßiges Mittel wäre, sodass der Endnutzer neuronale Netze selbst erkunden kann und dabei einen Einblick in die Struktur erhält und das neuronale Netz nicht nur als Black Box wahrnimmt.

Literatur

[ADT95] Andrews, Robert ; Diederich, Joachim ; Tickle, Alan B.: A survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge Based Systems, Vol.8, No.6, 1995, S. 373–389

[Bis06] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 0387310738

[CZ11] Chorowski, Jan ; Zurada, Jacek M.: Extracting Rules from Neural Networks as Decision Diagrams, Transactions on neural networks, 2011, S. 1–12

[Dre05] Dreyfus, Gerard: *Neural Networks - Methodology and Apllications*. Springer, 2005. – ISBN 10–3–540–22980–9

[GK99] Geiger, Carl ; Kanzow, Christian: *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Gabler Wissenschaftsverlage, 1999. – ISBN 3–54–066220–0

[KzB13] Kulluk, Sinem ; Özbakir, Lale ; Baykasoğlu, Adil: Fuzzy DIFACONN-miner: A Novel Approach for Fuzzy Rule Extraction from Neural Networks. In: *Expert Syst. Appl.* 40 (2013), Februar, Nr. 3, S. 938–946

[LMN93] Linial, Nathan ; Mansour, Yishay ; Nisan, Noam: Constant Depth Circuits, Fourier Transform, and Learnability. In: *J. ACM* 40 (1993), Nr. 3, S. 607–620

[LS95] Liu, Huan ; Setiono, Rudy: Chi2: Feature selection and discretization of numeric attributes, 7th IEEE International Conference Tools Artificial Intelligence, 1995, S. 388–391

[LSL95] Lu, Honhjun ; Setiono, Rudy ; Liu, Huan: NeuroRule: a connectionist approach to data mining, Procedings 21st Conference on Very Large Databases, 1995, S. 478–489

[LT95] Liu, Huan ; Tan, Sun T.: X2R: A fast rule generator, International Conference on Systems, Man and Cybernetics New York, 1995, S. 1631–1635

[Lud98] Ludermir, Teresa B.: Extracting Rules from Feedforward Boolean Neural Networks, Neural Networks 1998 Proceedings Vth Brazilian Symposium, 1998, S. 61–66

[Mit97] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill International Editions, 1997. – ISBN 0–07–115467–1

[Qui93] Quinlan, Ross: *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. – ISBN 1–55860–238–0

[RN03] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice-Hall International, 2003. – ISBN 0–13–080302–2

[SBM08] Setiono, Rudy ; Baesens, Bart ; Mues, Christopher: Recursive Neural Network rule extraction for Data with Mixed Attributes, Transactions on neural networks, Vol.19, No.2, 2008, S. 299–307

[Sch93] Schulte, Ulrich: *Einführung in Fuzzy-Logik*. München : Franzis-Verlag GmbH, 1993. – ISBN 3–7723–4771–1

[Sch05] Schoening, Uwe: *Logik für Informatiker*. Spektrum Akademischer Verlag GmbH, 2005. – ISBN 3–8274–1005–3. – 5. Auflage

[Set00] Setiono, Rudy: Extracting M-of-N Rules from Trained Neural Networks, Transactions on neural networks, Vol.11, No.2, 2000, S. 512–519

[SNF00] Santos, Raul T. ; Nievola, Julio C. ; Freitas, Alex A.: Extracting comprehensible Rules from Neural Networks via Genetic Algorithms, Proceddings 2000 IEEE Symposium on combination of evolutionary algorithm and neural network, 2000, S. 130–139

[SP97] Storn, Rainer ; Price, Kenneth: Differential Evolution &Ndash; A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. In: *J. of Global Optimization* 11 (1997), Dezember, Nr. 4, S. 341–359

[ST01] Sato, Makoto ; Tsukimoto, Hiroshi: Rule Extraction from Neural Networks via Decision Tree Induction, 2001, S. 1870–1875

[Tsu00] Tsukimoto, Hiroshi: Extracting Rules from Trained Neural Networks, Transactions on neural networks, Vol. 11, No. 2, 2000, S. 377–389

[WCJWS] W. Craven ärk ; Jude W. Shavlik", booktitle = In Proceedings of the Eleventh International Conference on Machine Learning year = 1994 pages = 37–45 publisher = . title = Using Sampling and Queries to Extract Rules from Trained Neural Networks . title = Using Sampling and Queries to Extract Rules from Trained Neural Networks

[web] *Diagram showing overfitting of a classifier*. `https://upload.wikimedia.org/wikipedia/commons/1/19/Overfitting.svg`, . – Aufgerufen am 15/07/2014

[zBKY09] Özbakir, Lale ; Baykasoğlu, Adil ; Kulluk, Sinem ; Yapici, Hüseyin: TACO-miner: An ant colony based algorithm for rule extraction from trained neural networks. In: *Expert Systems with Applications* 36 (2009), Nr. 10, S. 12295 – 12305

[ZCC00] Zhou, Zhi-Hua ; Chen, Shi-Fu ; Chen, Zhao-Qian: A Statistics based Approach for Extracting Priority Rules from Trained Neural Networks, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks Vol. 3, 2000, S. 401–406

[Zho04] Zhou, Zhi-Hua: Rule Extraction: Using Neural Networks or for Neural Networks?, Journal of Computer Science and Technology, Vol. 19, 2004, S. 249–253