Thema

Ein Hybrides System zur Erkennung von handgeschriebenen Buchstaben

Enrico Altmann

Inhalt

1. Motivation

- Handschrift - Erkennung wichtig für die automatische Verarbeitung von Vorlagen (Bsp. Überweisungsträger von Kreditinstituten)

- Seit langem Verwendung von Neuronalen Netzwerken zur Erkennung (= schnelle Verarbeitung, ungenau)

- Und Verwendung von strukturellen Analysen mittels Inferenz von Array-Grammatiken (= detailgenaue Ergebnisse, langsame Verarbeitung)

Benutzung der Stärken beider Module und dadurch Kompensierung ihrer Schwächen!

Neuronales Netzwerk	Strukturelle Analyse
- Aufgabe dieses statistischen Moduls ist die Bereitstellung einer Vorauswahl = hohe Ähnlichkeit zum zu-suchenden Muster - Neuronales Netzwerk arbeitet schnell, liefert aber keine korrekte Lösung (d.h. nur Eliminierung von nicht in Frage kommenden Zeichen)	- detailgenaue Untersuchung jedes einzelnen Zeichens, da sich einige Zeichen nur durch wenige Pixel unterscheiden - strukturelle Analyse basiert auf Array-Grammatiken = Verwendung von speziellen Array-Grammatiken für jedes einzelne Zeichen - Problem: Nichtdeterminismus = Lösung: Erweiterung der Array-Grammatiken um Kontrollmechanismen

2. Preprocessing

Datensammlung

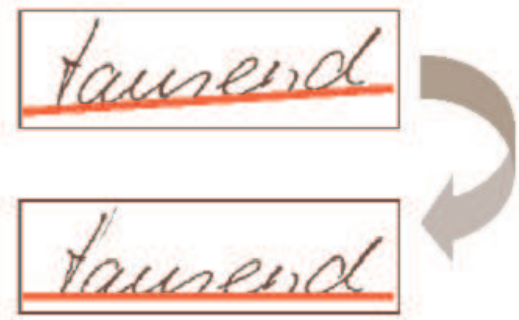
- Datenbasis: handgeschriebene Zeichen hunderter Personen → spezielle Vorlagen!
  (! keine Überlappung innerhalb der Vorlagen, kleine Buchstaben müssen ausgezeichnet sein)

Datenaufbereitung

- Eliminierung von Noise-Pixeln
- Normalisierung der handgeschriebenen Buchstaben:

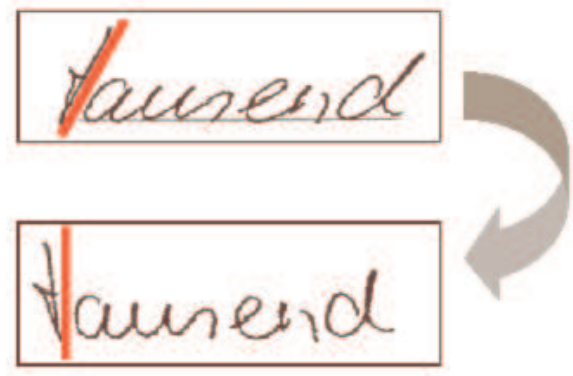
(1) Skewness-Korrektur (Neigung der Basislinie)

- Basislinie wird so verändert, dass sie nach der Korrektur waagerecht liegt



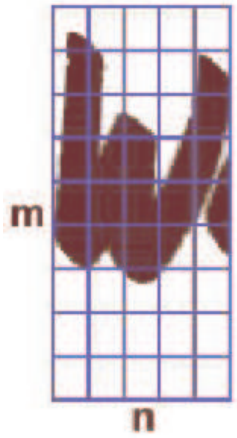
(2) Slant-Korrektur (Schriftneigung)

- Schriftneigung wird so verändert, dass sie nach der Korrektur senkrecht liegt

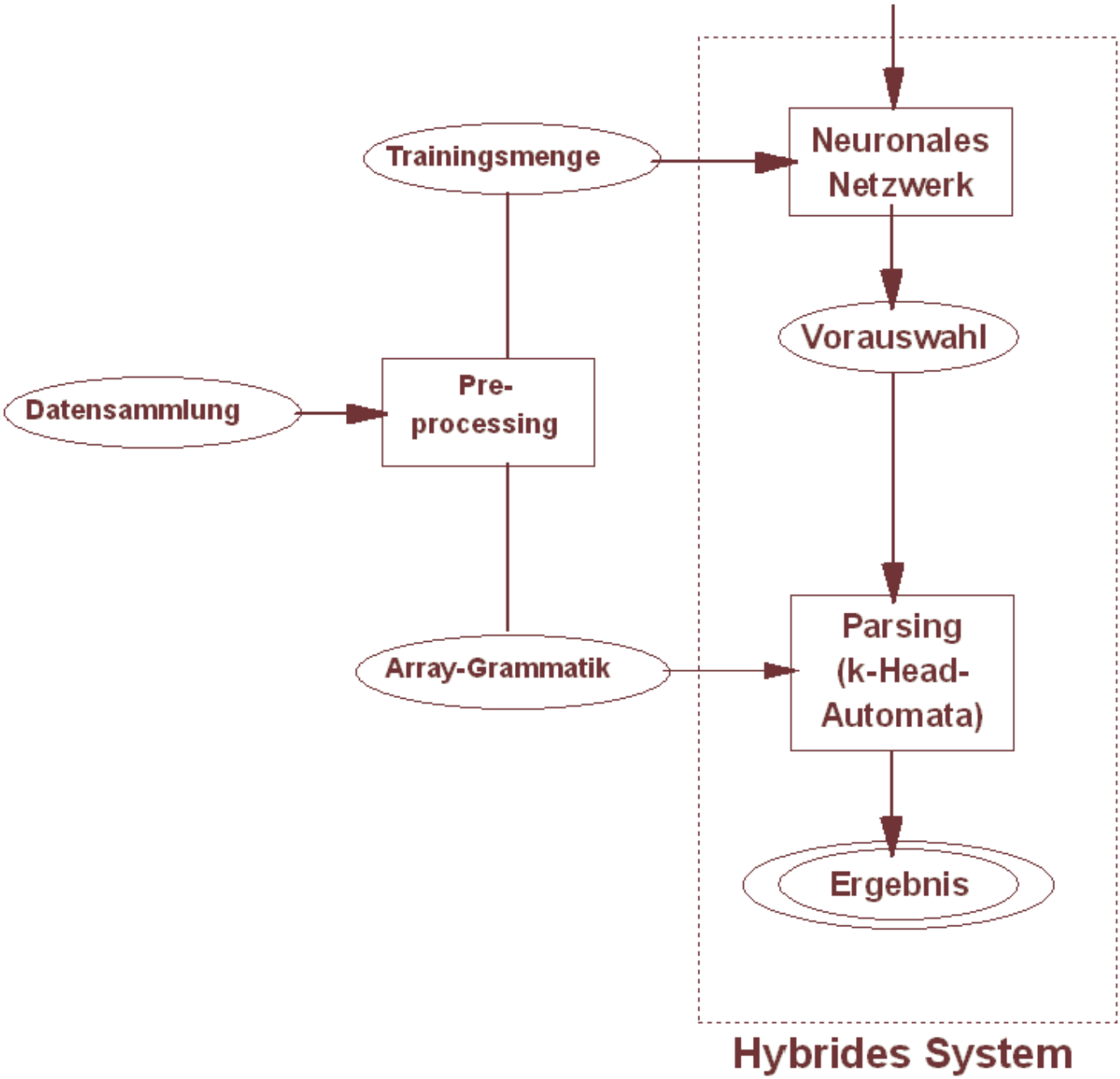


(3) Skalierung/ Rasterung/ Ausdünnung

- Normalisierung(Größe) der Buchstaben auf ein 20×25 Raster

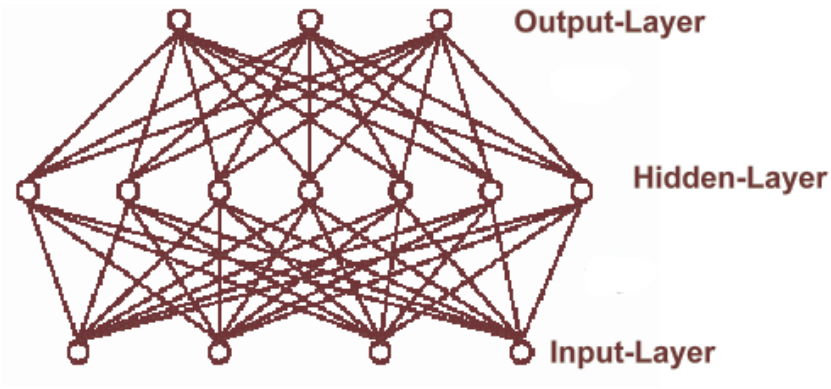


3. Überblick: Hybrides System



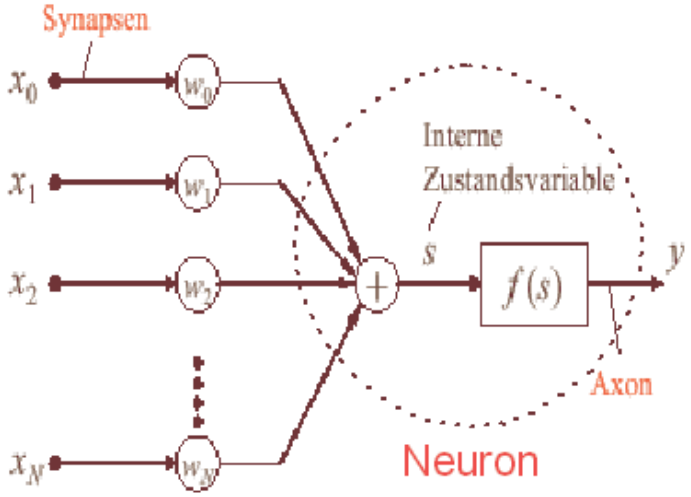
3.1. (Künstliches) Neuronales Netzwerk

Was ist ein Neuronales Netzwerk?



- bestehend aus:
  (1) 1 : Input-Layer
  (2) n : Hidden-Layer
  (3) 1 : Output-Layer

- Jede Schicht besitzt unterschiedliche Anzahl von sog. Neuronen:



$$y = f(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

- Jedes Neuron(Knoten) symbolisiert eine mathematische Funktion

- Eingänge(Synapsen bzw. Axon) des Neurons besitzen bestimmten Wert

- Neuronales Netz kann trainiert werden, dazu werden Testdaten verwendet, Teacher überwacht dabei Ergebnis
  = bei Fehler:
  Strukturelle Backpropagation!
  (Kantengewichte w_i werden verändert!)

(1) **Ziel des Moduls ist nicht herauszufinden welches Zeichen gesucht wird, sondern welches Zeichen nicht gesucht wird**
(2a) Für erfolgreiche Erkennung des hybriden Verfahrens muss das richtige Zeichen in der Präselektion enthalten sein
(2b) ...und sollten die Wahrscheinlichkeitswerte der einzelnen Zeichen möglichst nahe am Wert 1 liegen

Spezialisierung des Netzwerks für den Erkennungsprozess

- ein Input-Vektor wird dem (trainierten) Neuronalen Netzwerk übergeben:

  (1) Pixelsumme pro Spalte und Zeile
  (2) Anzahl benachbarter Pixel pro Linie für jede Zeile und Spalte

- Ausgabe ist berechneter Wahrscheinlichkeitswert

- Dadurch schnelle Eliminierung von Zeichen, die offensichtlich nicht durch das gesuchte Muster(Zeichen) repräsentiert werden

3.2. Schnittstelle: Neuronales Netzwerk - Strukturelle Analyse

- Benötigen geeignetes interface zwischen den Methodiken
- Ergebnisse des neuronalen Netzwerkes(Zeichen + Wahrscheinlichkeit = UNIT) müssen übergeben werden
  (dabei symbolisieren die Werte die Ähnlichkeit zum zu suchenden Muster!)
- programmiertechnisch: verkette Liste mit den Units
- Strategien um Kandidaten zu extrahieren:

Absteigende Auswahl zu einem definierten Grenzwert	**Auswahl mittels Aufsummierung**	**Auswahl einer vordefinierten Gruppe von Zeichen**
- festgelegter Grenzwert g - wir nehmen alle Units, deren Wahrscheinlichkeit größer g - Festlegung: falls keiner größer g ist, werden alle genommen	- sortieren der Werte der einzelnen Units - aufsummieren der Werte bis ein bestimmtes Summen-Limit erreicht ist	- trainierte neuronale Netzwerk „merkt“ sich ähnliche Buchstaben = Gruppenbildung

3.3. Strukturelle Analyse mittels Array-Grammatiken und zugehörigem Endlichen Array Automaten

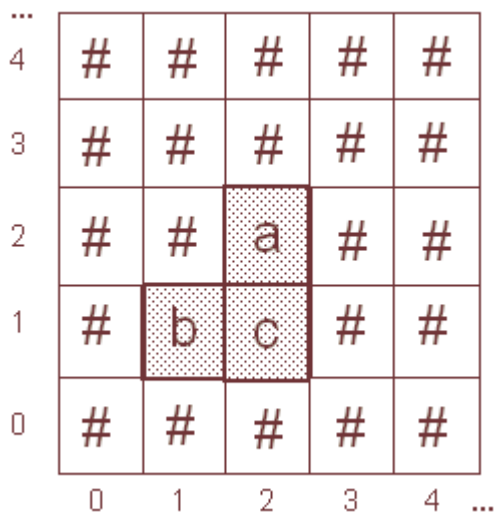
Definitionen: Array

Array: $A : Z \times Z \rightarrow \Sigma \cup \{\#\}$

Σ.. sei beliebiges Alphabet

Bsp. für Arrays: sei $a, b, c \in \Sigma$
(rechts: fiktiver Satz von Arrays)

A(1,1) = b
A(2,1) = c
A(2,2) = a
A(1,2) = #
...

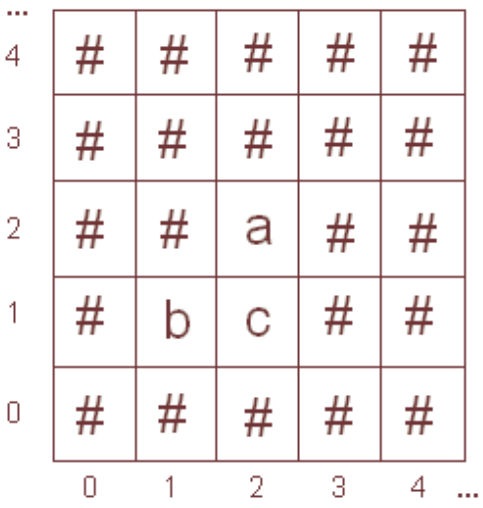


Satz aller Arrays: Σ^{*2}

Shape(A): $shape(A) := \{v \in Z^2 \mid A(v) \neq \#\}$ sei dabei endliche Menge

Bsp:(siehe Beispiel vorherige Seite)

$shape(A) = \{(1,1), (2,1), (2,2)\}$



Notation für A: $A = \{(v, A(v)) \mid v \in shape(A)\}$

Definition: Array-Grammatik:

- $G = (N, T, \#, P, \{(v_0, S)\})$

  N.. Nichtterminale/ Metasymbole
  T.. Terminale
  P.. Produktionsmenge mit: $(\alpha \to \beta) \in P, \quad \alpha, \beta \in (N \cup T)^{*2}$:
  (1) $shape(\alpha) = shape(\beta) \Rightarrow \alpha, \beta$ sind isometrisch

  v_0.. Startvektor
  S.. Startsymbol

Klassifizierung der Array-Grammatiken

(1) monoton (MON)
- falls für jedes $(\alpha \to \beta) \in P$ gilt: (die Nicht-#-Symbole in α werden nicht durch ein # in β ersetzt)

(2) #-kontextfrei (#-CF)
- falls für jedes $(\alpha \to \beta) \in P$ gilt: (α besteht nur aus einem Nichtterminal und mindestens einem #)

(3) kontextfrei (CF)
- falls für G gilt: (MON und #-CF)

Definition: Ableitungsrelation $(\Rightarrow)$

- Sei $A, B \in \Sigma^{*2}$ $\quad A \overset{(\alpha \to \beta)}{\Rightarrow} B$: $\exists (\alpha \to \beta) \in P$ für die gilt:

  (1) α ist Submuster in A und wird durch β in A ersetzt

Definition: Reflexiver, transitiver Abschluss

- Notation: $A \Rightarrow ... \Rightarrow B$, dann: $A \overset{*}{\Rightarrow} B$, $\quad (A \in \Sigma^{*2}$ und $B \in (\Sigma \setminus N)^{*2})$

Definition: erkannte Sprache einer Array-Grammatik

- $L(G) = \{A \in (\Sigma \setminus N)^{*2} \mid \{(v_0, S)\} \overset{*}{\Rightarrow} A\}$

Beispiel für einfache Array-Grammatik

(welche alle rechtwinkligen Dreiecke, mit rechtem Winkel in der linken unteren Ecke, erzeugt)

$$G = (\{S, T\}, \{1\}, \#, P, S)$$

P : **p1**: $\begin{matrix} \# \\ S\ \# \end{matrix} \rightarrow \begin{matrix} S \\ 1\ T \end{matrix}$ **p2**: $S \rightarrow 1$ **p3**: $\begin{matrix} T \\ \# \end{matrix} \rightarrow \begin{matrix} 1 \\ \# \end{matrix}$ **p4**: $\begin{matrix} T \\ 1\ \# \end{matrix} \rightarrow \begin{matrix} 1 \\ 1\ T \end{matrix}$

Ableitung:

$$S \overset{p_1}{\rightarrow} \begin{matrix} S \\ 1\ T \end{matrix} \overset{p_1}{\rightarrow} \begin{matrix} S \\ 1\ T \\ 1\ T \end{matrix} \overset{p_3}{\rightarrow} \begin{matrix} S \\ 1\ T \\ 1\ 1 \end{matrix} \overset{p_4}{\rightarrow} \begin{matrix} S \\ 1\ 1 \\ 1\ 1\ T \end{matrix} \overset{p_1}{\rightarrow} \begin{matrix} S \\ 1\ T \\ 1\ 1 \\ 1\ 1\ T \end{matrix} \overset{p_4}{\rightarrow} \begin{matrix} S \\ 1\ 1 \\ 1\ 1\ T \\ 1\ 1\ T \end{matrix} \overset{p_2}{\rightarrow} \begin{matrix} 1 \\ 1\ 1 \\ 1\ 1\ T \\ 1\ 1\ T \end{matrix} \overset{p_3}{\rightarrow}$$

$$\begin{matrix} 1 \\ 1\ 1 \\ 1\ 1\ T \\ 1\ 1\ 1 \end{matrix} \overset{p_4}{\rightarrow} \begin{matrix} 1 \\ 1\ 1 \\ 1\ 1\ 1 \\ 1\ 1\ 1\ T \end{matrix} \overset{p_3}{\rightarrow} \begin{matrix} 1 \\ 1\ 1 \\ 1\ 1\ 1 \\ 1\ 1\ 1\ 1 \end{matrix}$$

Modifikation der Array-Grammatiken

Die nachfolgenden Mechanismen zur Modifikation von Array-Grammatiken tragen dazu bei:

- Den Nichtdeterminismus, bei Inferenz der Regeln p_i, zu eliminieren
- Und somit Parsinggeschwindigkeit zu minimieren

<u>Graph-kontrollierte Array-Grammatiken</u>

$$G = (N, T, \#, (P, L_{IN}, L_{FIN}), \{(v_0, S)\})$$

- Wobei gilt: (1) $L_{IN} \subseteq Lab(G)$ ...Satz von Start-Labels
  $L_{FIN} \subseteq Lab(G)$ ...Satz von Ende-Labels

  (2) $\forall p \in P: \quad (l(p) : \pi(l(p)),\ \sigma(l(p)),\ \varphi(l(p)))$

  $l(p) \in Lab(G)$...assozierte Satz von Labels zu jeder Regel p
  $\pi(l(p))$ ...ist Array Produktion über $N \cup T$
  $\sigma(l(p)) \in Lab(G)$ ...das Erfolgs-Feld

$\varphi(l(p) \in Lab(G)$...das Fehler-Feld

(3) Ableitungsschritte: $(v,\ l(p)) \overset{G_p}{\Rightarrow} (w,\ t)$ mit $v, w \in (N \cup T)^{+2}$

entweder

(3.1) eine Array-Produktion in $\pi(l(p))$ ist anwendbar auf v, ist:
Ergebnis w ist und $t \in \sigma(l(p))$

oder

(3.2) keine Array-Produktion $\pi(l(p))$ ist anwendbar auf v, ist:
Ergebnis $v = w$ und $t \in \varphi(l(p))$

Allgemeiner Ableitungsverlauf

$$(w_0, l_0) \overset{G}{\Rightarrow} (w_1, l_1) \overset{G}{\Rightarrow} \ldots \overset{G}{\Rightarrow} (\underbrace{w_k}, l_k),\ \text{wobei } l_0 \in L_{IN}, l_k \in L_{FIN},$$
$$A \in (\Sigma \setminus N)^{*2} \qquad 0 \le j \le k : l_j \in Lab(G)$$

Vorgeschriebene Gruppen

- Einteilung der Array-Produktionen P in endliche Multisets R über P:

$$R = \langle p_1, \ldots, p_m \rangle; \quad p_i \in P;$$

- Produktionen eines Multisets werden parallel in einem Ableitungsschritt ersetzt
- Dadurch parallele Analyse von Linien mit gleichen Eigenschaften, dadurch Reduzierung der Analyse-Zeit(Komplexität)

Endlicher Array Automat (mit k-Köpfen)

Definition: Endlicher Array Automat (mit k-Köpfen) vom Typ X

$$M = (N,\ T,\ \#,\ (P, \Re, F),\ (v_0, S)\ );\ \text{ wobei } X \in \{\# - CF,\ CF\}$$

- weiter gilt: (1)$(N,\ T,\ \#,\ P,\ (v_0, S))$ ist Array-Grammatik vom Typ X

  (2) $R\ \in\ \Re$
  (beinhaltet die vorgeschriebenen Gruppen R)

  (3) $F\ \subseteq\ P;$ falls p_i nicht anwendbar, dann können p_i in F vernachlässigt werden
  (für den appearance-checking-mode)

Konfigurationen von M

$$C = \{(a, a),\ (a, X),\ (\#, Y) |\ a \in T,\ X \in N \cup \{\#\},\ Y \in N\}^{+2}$$

- kann eingeteilt werden in:

  (1) schon gelesenen Arrays $(a,\ a)$

  (2) Arrays der Form $(a,\ \#)$ welche noch abgeleitet werden müssen

  (3) Arrays $(b,\ X)$ mit $(b \in T \cup \{\#\};\ \ X \in N)$
  (Zustand des Automaten, wird durch seine Nonterminale festgelegt)

- Initiale Konfiguration:

$$C_0(A) = \{(v, (A(v),\ \#) |\ v \in Z^2 \backslash \{v_0\} \cup \{(v_0, (A(v_0), S))\}$$

Ableitungsrelation $(\vdash_R)$ von M

- meint: Übergang von $C \vdash_R C'$, gdw. folgendes erfüllt ist:

  (1) jede Variable in$C(2)$ tritt nur einmal auf

  (2) jede Variable in$C'(2)$ tritt nur einmal auf

  (3) jedes $p_i \in R$ ist entweder anwendbar auf $C(2)$ oder gehört zu F

  (4) Sei $R(C)$ der Satz von anwendbaren Regeln aus R, dann korrespondiert der Satz der Variablen der linken Seite der Regeln, mit den Satz der Variablen, die in C auftreten

  (5) Subarrays, die ersetzt werden, dürfen keine # erzeugen

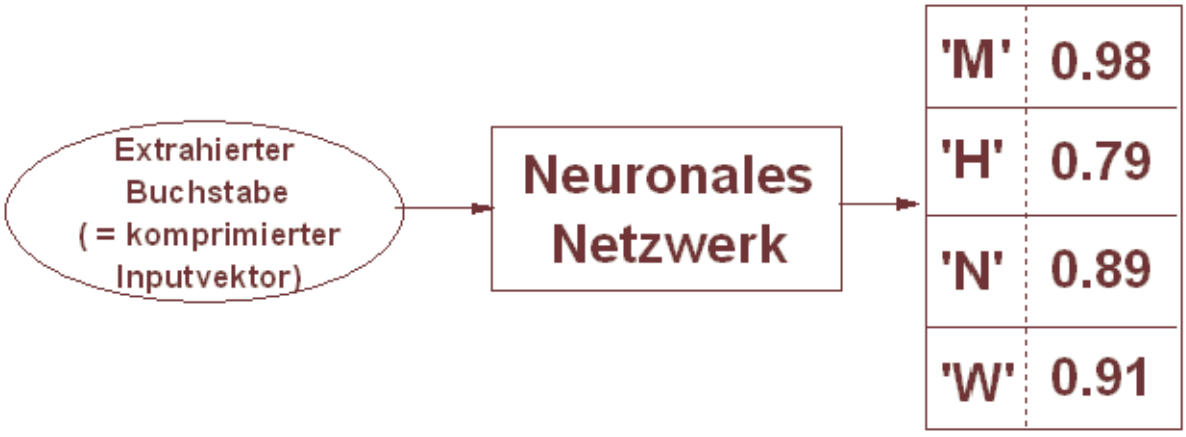
Erkannte Sprache von M:

$$L(M) = \{A |\, A \in (\Sigma \setminus N)^{*2}, C_0(A) \Rightarrow_M^* \{(v, (A(v), A(v)))|\, v \in shape(A)\}\}$$

4. Abschließendes Beispiel (Automatisierung: Bearbeitung von Überweisungsträgern)

(1)

(2)



- Da das Neuronale Netzwerk den Buchstaben ‘M’ favorisiert, wird der Endliche Array Automat, der den Buchstaben ‘M’ erkennen kann, zu einer weiteren Analyse herangezogen:
- (BEACHTE! Jeder Endliche Array Automat kann genau nur einen Buchstaben „erkennen“)

(3)

$$M = (\{S, L, R, D_L, U_L, D_R, U_R\}, \{x\}, \#, (P, \Re, \emptyset), \{v_0, S\})$$

$$P = \{\ S\ \#\ \rightarrow\ L\ R\ ,\ \ \#\ L\ \rightarrow\ L\ x\ ,\ \ R\ \#\ \rightarrow\ x\ R\ ,$$

$$\begin{matrix}\# \\ L \\ \#\end{matrix} \rightarrow \begin{matrix}U_L \\ x \\ D_L\end{matrix}\ ,\ \ \begin{matrix}R \\ \# \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ U_R \\ D_R\end{matrix}\ ,\ \ \begin{matrix}\# \\ U_L\end{matrix} \rightarrow \begin{matrix}U_L \\ x\end{matrix}\ ,\ \ \begin{matrix}\# \\ U_R\end{matrix} \rightarrow \begin{matrix}U_R \\ x\end{matrix}\ ,$$

$$\begin{matrix}D_L \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ D_L\end{matrix}\ ,\ \ \begin{matrix}D_R \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ D_R\end{matrix}\ ,\ U_L \rightarrow x,\ U_R \rightarrow x,\ D_L \rightarrow x,\ D_R \rightarrow x\}$$

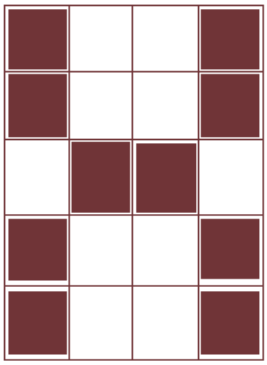
$$\Re = \{\{\ S\ \#\ \rightarrow\ L\ R\ \},\ \{\ \#\ L\ \rightarrow\ L\ x\ ,\ \ R\ \#\ \rightarrow\ x\ R\ \},$$

$$\{\ \begin{matrix}\# \\ L \\ \#\end{matrix} \rightarrow \begin{matrix}U_L \\ x \\ D_L\end{matrix}\ ,\ \ \begin{matrix}R \\ \# \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ U_R \\ D_R\end{matrix}\ \},$$

$$\{\ \begin{matrix}\# \\ U_L\end{matrix} \rightarrow \begin{matrix}U_L \\ x\end{matrix}\ ,\ \ \begin{matrix}\# \\ U_R\end{matrix} \rightarrow \begin{matrix}U_R \\ x\end{matrix}\ ,\ \ \begin{matrix}D_L \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ D_L\end{matrix}\ ,\ \ \begin{matrix}D_R \\ \#\end{matrix} \rightarrow \begin{matrix}x \\ D_R\end{matrix}\ \}$$

$$\{U_L \rightarrow x,\ U_R \rightarrow x,\ D_L \rightarrow x,\ D_R \rightarrow x\}\}$$

- Der extrahierte Buchstabe (siehe Abschnitt 2.) wird für eine strukturelle Analyse „aufbereitet“



$\Rightarrow\!\!\!\Rightarrow$

$$\begin{array}{ccc} (x,\#) & & (x,\#) \\ (x,\#) & & (x,\#) \\ & (x,S)\ (x,\#) & \\ (x,\#) & & (x,\#) \\ (x,\#) & & (x,\#) \end{array}$$

(extrahierter Buchstabe)

(Automat-gerechte Darstellung für Start des Automaten)

Ableitungsschritte für ‘M’:

$$\begin{array}{ccccccccccccccc} (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) \\ (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,U_L) & & (x,U_R) \\ & (x,S)\ (x,\#) & & \Rightarrow^M & & (x,L)\ (x,R) & & \Rightarrow^M & & (x,x)\ (x,x) & \\ (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,D_L) & & (x,D_R) \\ (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) & & (x,\#) \end{array}$$

$$\begin{array}{ccccccccc} & (x,U_L) & & & (x,U_R) & & (x,x) & & (x,x) \\ & (x,x) & & & (x,x) & & (x,x) & & (x,x) \\ \Rightarrow^M & & (x,x)\ (x,x) & \Rightarrow^M & & (x,x)\ (x,x) & & & \\ & (x,x) & & & (x,x) & & (x,x) & & (x,x) \\ & (x,D_L) & & & (x,D_R) & & (x,x) & & (x,x) \end{array}$$

- Da das Array nur noch aus Terminalen besteht, ist der Parsingvorgang beendet, somit wurde der Buchstabe ‘M’ erfolgreich erkannt
(siehe dazu: Abschnitt 3.3: erkannte Sprache von M)

5. Quellenangaben

1.) Rudolf Freund/ Markus Neubauer/ Martin Summerer/ Stefan Gruber/ Jürgen Schaffner/ Roland Swoboda: «A Hybrid System for the Recognition of Hand-Written Characters», LNCS 1889 (2000), S.67-76

2.) Henning Fernau/ Rudolf Freund/ Markus Holzer: «Character Recognition with k-Head Finite Array Automata»,
http://www-fs.informatik.uni-tuebingen.de/~fernau/pub-fernau/papers/ps/SSPR98.ps.gz
(letzter Zugriff: 16.10.2006)

3.) Prof. Dr. Horst Buhnke/ Andreas Humm: «Erkennung handgeschriebener Buchstaben»
http://homeweb2.unifr.ch/humma/pub/publications/Informatikprojekt.pdf
(letzter Zugriff: 04.04.2007)

4.) Andrew Mercer/ Azriel Rosenfeld: «An Array Grammar Programming System»
ACM Press Volume 16, (Issue 5), S. 299-305
http://delivery.acm.org/10.1145/370000/362198/p299-mercer.pdf
(letzter Zugriff: 04.04.2007)

5.) Htttp://download.informatik.uni-freiburg.de/lectures//Mustererkennung/2003-2004WS/Misc/Slides/ME-16-03.pdf
(letzter Zugriff: 04.04.2007)