Robert Salvador **salvador@htlinn.ac.at**

Neuronales Netz mit Backpropagation-Algorithmus

- Mathematische / Fachliche Inhalte in Stichworten:

  Matrizenrechnung, Extremwertberechnung einer Funktion mit vielen Variablen, Iteration

- Kurzzusammenfassung

  Das vorliegende Mathcad-Dokument simuliert ein Neuronales Netz zur Zeichenerkennung auf der Basis des Backpropagation-Algorithmus. Es können Zeichen vorgegeben werden, die das Netz - auch wenn sie in etwas veränderter Form präsentiert werden - richtig erkennen sollte.

- Lehrplanbezug (bzw. Gegenstand / Abteilung / Jahrgang):

  Im Rahmen der Regelungstechnik (nach dem bisherigen Lehrplan der Abteilung Elektronik) habe ich versucht, in wenigen Unterrichtsstunden den Schülern u. a. die Grundkonzepte Neuronaler Netze näherzubringen. Zur einfachen Veranschaulichung (man könnte auch sagen zum Spielen) habe ich diese Mathcad-Simulation geschrieben.

- Mathcad-Version:

  Mathcad 2000

- Literaturangaben:

  Jörg Kahlert, Fuzzy-Control für Ingenieure, Vieweg (enthält einen kurzen Abschnitt über Neuronale Netze)
  Uwe Lämmel, Jürgen Cleve, Künstliche Intelligenz, Fachbuchverlag Leipzig
  James A. Freeman, David M. Skapura, Neural Networks, Addison-Wesley Publishing Company
  Spektrum der Wissenschaft, November 1992

- Anmerkungen bzw. Sonstiges:

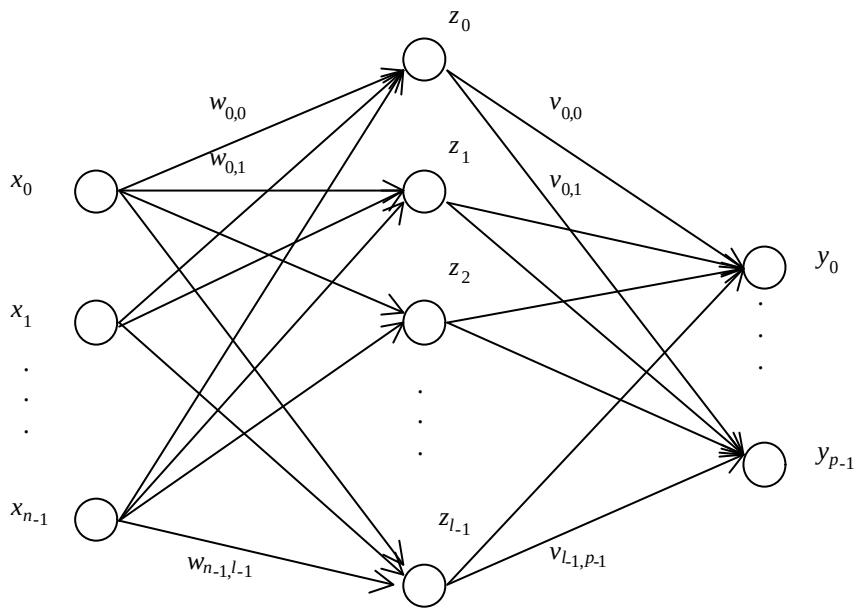
  Dieses Mathcad-Dokument eignet sich wohl nur zur Demonstration eines Neuronalen Netzes und wird an unserer Schule (HTL1, Innsbruck) von manchen Kollegen der Abteilung Elektronik bzw. Wirtschaftsingenieurwesen auch für diesen Zweck verwendet. Einige Schüler zeigen sich dabei immer wieder recht interessiert.

Hier befindet sich zunächst der Verweis auf ein zweites Mathcad-Dokument, in dem Definitionen, Hilfsfunktionen, usw. sowie auch der Backpropagation-Algorithmus untergebracht sind, um an dieser Stelle nicht zu stören:

Übersicht:D:\Eigene Dateien\math-tech\Fremde\Neuro_Include.mcd(R)

Einleitung

Beim verwendeten Neuronalen Netz handelt es sich um ein 3-Schicht-Perzeptron-Netz, wie es das folgende Bild zeigt. Es wird über den Backpropagation-Algorithmus darauf trainiert, 26 Muster als Zeichen zu erkennen; im speziellen Fall handelt es sich dabei um die Erkennung der Großbuchstaben A ... Z. Es können aber beliebige eigene Zeichen definiert werden.



Die Grundidee dabei ist folgende:

Die Zeichen sind als 7x5-Matrizen mit den Matrixelementen 0 oder 1 definiert. Diese Zeichen-Muster mit ihren jeweils 35 0/1-Informationen entsprechen ebensovielen Neuronen in der Eingangsschicht (input layer). Die Eingangsneuronen (also jene, die sozusagen einen äußeren Reiz aufnehmen) beeinflussen über die Gewichtungen $w_{j,k}$ und die Aktivierungsfunktion die Neuronen der versteckten Zwischenschicht (hidden layer), diese wiederum über die Gewichte $v_{m,n}$ die Neuronen der Ausgangsschicht (output layer).

Die Ausgangsneuronen sind letzten Endes jene, die nach außen hin auf die äußeren Reize reagieren. In unserem Fall gibt es 26 Stück davon, für jedes zu erkennende Zeichen eines. Konkret wird das Zeichen mit der Nummer *n* "erkannt", wenn das dementsprechende Ausgangsneuron mit der Nummer *n* einen großen Funktionswert aufweist (möglichst 0.9) und alle anderen einen kleinen (möglichst 0.1).

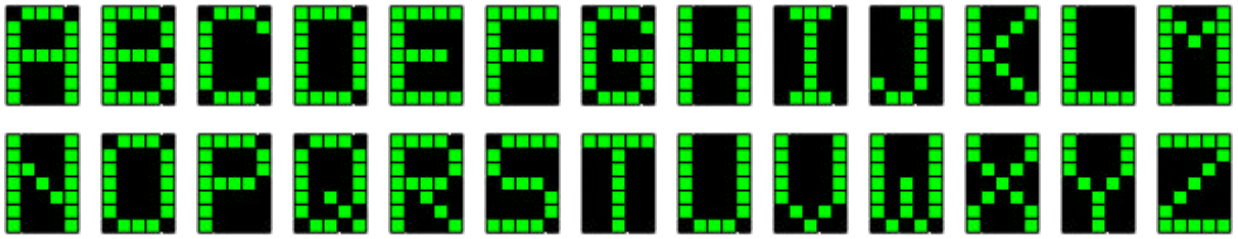
Um nun die Gewichtsmatrizen *w* und *v* im Sinne einer maximalen Erkennungsquote richtig einzustellen, wird das Neuronale Netz trainiert. Dazu startet man mit Zufallsgewichten, präsentiert dem Netz der Reihe nach bekannte Muster, schaut sich an, wie das Netz darauf reagiert und vergleicht damit, wie es reagieren soll. Entsprechend dieser Abweichungen werden die Gewichte korrigiert. Anders ausgedrückt: Eine Fehlerfunktion (die Summe aller quadrierten Differenzen), die von allen Gewichten abhängt, soll während der Trainingsphase minimiert werden. Mathematisch: Man sucht das möglichst globale Minimum einer Funktion von sehr vielen Variablen.

Der Backpropagation-Algorithmus dient nun dazu - geometrisch gesprochen - von irgendeinem Punkt auf der Fläche dieser Fehlerfunktion ausgehend in vielen kleinen Schritten auf dem jeweils steilsten Weg (Methode des steilsten Abstiegs) nach unten dieses globale(?) Minimum zu erreichen. Im allgemeinen wird man allerdings wohl mit einem tiefen relativen Minimum oder einer anderen tief liegenden Stelle zufrieden sein müssen.

Definition der Grundzeichen

Die Zeichen werden über entsprechende Matrizen definiert (siehe *Neuro_Include.mcd*!).

Vordefinierte Zeichen



Vordefinierte Zeichen

Die Trainingsphase

Die Funktion *make_train* generiert die Trainingsdaten. *y_soll* enthält die zugehörigen Soll-Ausgangswerte.

$train := make\_train(Zeichenzahl)$

$y\_soll := 0.1 + matrix(Zeichenzahl, Zeichenzahl, f\_null) + 0.8 \cdot einheit(Zeichenzahl)$

Zusammengefassung der bisherigen Daten:

$in := zeilen(train)$	Zahl der Eingangsneuronen:	$in = 35$
$hid := 30$	Zahl der inneren Neuronen:	$hid = 30$
$out := Zeichenzahl$	Zahl der Ausgangsneuronen:	$out = 26$

Wir starten das Training mit Zufallsgewichten zwischen -1 und 1:

$w\_start := matrix(in, hid, f\_rand\_wv)$ $\quad$ $v\_start := matrix(hid, out, f\_rand\_wv)$

$gewichte := zeilen(w\_start) \cdot spalten(w\_start) + zeilen(v\_start) \cdot spalten(v\_start)$

Gesamtanzahl der Gewichte: $gewichte = 1830$

Jetzt beginnt die eigentliche Lernphase: das Netz reagiert der Reihe nach auf alle vorgelegten Muster, die Gewichte werden korrigiert, das Spiel beginnt von vorn.
Wie oft der Algorithmus die erwähnten Gewichtskorrekturen vornimmt, hängt von verschiedenen Faktoren ab:
Man kann das Training

- über die Zahl N_{max} der Lernzyklen (Iterationen) und
- über die obere Schranke für den Fehler err_{max} steuern (beides grün unterlegt als Zeichen für eine Eingabemöglichkeit).
- Außerdem spielt der "Zufall" eine Rolle (Start-Gewichtsmatrizen)

Das Training stoppt, sobald die erste Abbruchbedingung erfüllt ist.
Hinweis: err_{max} sollte möglichst nicht über 0.05 liegen

Ein zeitlicher Richtwert: Ein Pentium III (1000 MHz/ 256 MB) braucht für 10000 Iterationen etwa 3:30 Minuten.

$\alpha := 0.15$ bestimmt die Länge der Schritte beim Abstieg zum Minimum (ca. 0.15 hat sich experimentell als günstig erwiesen)

$N_{max} := 10000$ Maximale Zahl der Lernzyklen

$err_{max} := 1$ Obere Schranke für den Fehler

$wv := back\_prop(w\_start, v\_start, train, y\_soll, \alpha, N_{max}, akt, err_{max})$

Die beiden Teilmatrizen *w_stop* und *v_stop* müssen aus *wv* extrahiert werden:

$w\_stop := submatrix(wv, 0, in - 1, 0, hid - 1)$ $v\_stop := submatrix(wv, in, zeilen(wv) - 1, 0, hid - 1)^T$

$Iter := wv_{zeilen(wv)-1,0} + 1$

Was kann das Neuronale Netz?

Die Fähigkeiten des Netzes werden nun zweifach überprüft:
Zuerst werden die Anzahl der Lernzyklen, sowie die oben erwähnten Fehler vor und nach der Trainigsphase angezeigt (*Fehler_start*, *Fehler_stop*) und daraus der Quotient gebildet (*Fehler_quot*); um diesen Faktor hat sich der Fehler durch das Training gegenüber dem Anfang verkleinert.

Anzahl der Lernzyklen: $Iter = 363$

$Fehler\_start := wv_{zeilen(wv)-1,1}$ $Fehler\_start = 54.195$

$Fehler\_stop := wv_{zeilen(wv)-1,2}$ $Fehler\_stop = 0.99944$

$Fehler\_quot := \frac{Fehler\_start}{Fehler\_stop}$ $Fehler\_quot = 54$

Die zweite Überprüfung ist wesentlich anschaulicher. Der Anwender kann die Matrix *test_zeichen* (siehe unten - wieder grün unterlegt) definieren und sich ansehen, wie das Neuronale Netz dieses Muster interpretiert.
Im Vektor *ist_ergebnis* sind alle Ausgangszustände des Netzes zusammengefasst, die den einzelnen Zeichen entsprechen (siehe darunter liegende Buchstaben!). Das dem größten Teilergebis zugeordnete Zeichen ist das vom Netz "erkannte" Zeichen. Es wird zum schnelleren Auffinden von einem Schieberegler angezeigt. Außerdem gibt es darunter noch eine Klartextausgabe.
Bei Eingabe eines der 26 Idealmuster sollte sich beim betreffenden Buchstaben 0.9 ergeben, bei allen anderen 0.1.
Aber auch ein verrauschtes Muster sollte richtig erkannt werden, wenn man beim Training entsprechend großzügig ist.

$$test\_zeichen := \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Ziffernerkennung - Test

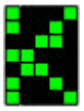
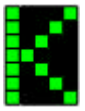
$test := mat\_to\_vek(test\_zeichen)$

$l := 0 .. hid - 1 \qquad p := 0 .. out - 1 \qquad z\_test_l := akt\left(w\_stop^{\langle l \rangle} \cdot test\right) \qquad ist\_ergebnis_p := akt\left(v\_stop^{\langle p \rangle} \cdot z\_test\right)$

$nr := zeichen\_auswertung(ist\_ergebnis)$

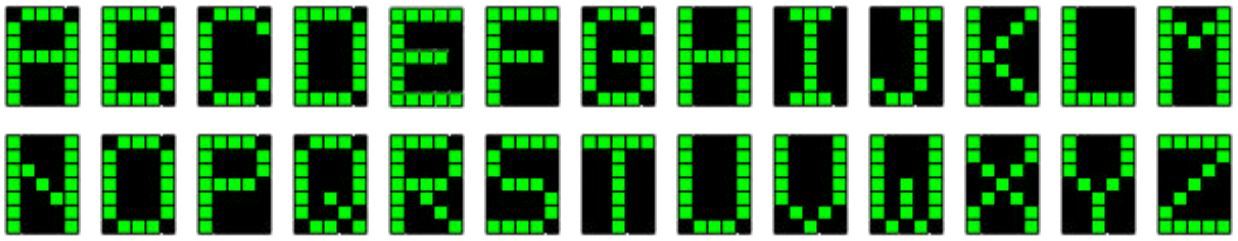
$ist\_ergebnis^T = (0.05\ 0.04\ 0.04\ 0.03\ 0.03\ 0.12\ 0.01\ 0.14\ 0.03\ 0.18\ 0.74\ 0.05\ 0.10\ 0.04\ 0.02\ 0.02\ 0.07\ 0.14)$

A B C D E F G H I J K L M N O P Q R

Das oben definierte Muster  **wird als**  **erkannt.**

Und wie denken Sie darüber?

Noch einmal zur Erinnerung:



Ziffernerkennung - Test