Zeitschrift: Berichte der St. Gallischen Naturwissenschaftlichen Gesellschaft
Herausgeber: St. Gallische Naturwissenschaftliche Gesellschaft
Band: 91 (2008)

Artikel: Industrielle Anwendungen künstlicher neuronaler Netze
Autor: Fässler, Angela / Loher, Marcel
DOI: https://doi.org/10.5169/seals-832617

Nutzungsbedingungen
Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation
L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use
The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

Download PDF: 07.02.2026

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

91. Band | Seiten 341–353 | 22 Abbildungen | 0 Tabellen | St.Gallen 2008

Industrielle Anwendungen künstlicher neuronaler Netze

Angela Fässler und Marcel Loher

Inhaltsverzeichnis

Zusammenfassung 341
1. Geschichte 341
2. Natürliche Neuronale Netze: biologische Vorbilder 342
3. Mathematischer Hintergrund 342
4. Industrielle Anwendungen 349
Literaturverzeichnis 353

Zusammenfassung

Erste künstliche neuronale Netze wurden um 1960 ursprünglich entwickelt, um die Funktionsweise des menschlichen Gehirns besser zu verstehen. Dieses Vorhaben erwies sich als enorm schwierig und ist auch heute noch längst nicht abgeschlossen. Parallel dazu wurden künstliche neuronale Netze benutzt für Aufgaben aus Industrie und Technik, die kaum algorithmisch lösbar sind, für die aber Lösungsbeispiele vorliegen. Klassifikation, Mustererkennung und Prognose von Zeitreihen sind typische Probleme, die sich mit neuronalen Netzen gut lösen lassen.

Nach einer historischen Einleitung wird eine kurze Theorie des am häufigsten eingesetzten Netzwerk-Typs, des Multilayer-Perceptrons präsentiert. Daran schliesst sich ein Einblick in industrielle Projekte an, die an der Hochschule für Angewandte Wissenschaften St.Gallen durchgeführt wurden.

1. Geschichte

Die Entwicklung der elektronischen Datenverarbeitung liest sich wie eine einzige grosse Erfolgsgeschichte. Was im zweiten Weltkrieg mit dem Versuch begann, deutsche Funksprüche zu entschlüsseln, ist zu einer Techno-

Dr. Angela Fässler, NTB Interstaatliche Hochschule für Technik,
Campus Waldau, Schönauweg 4, CH-9013 St.Gallen
Dr. Marcel Loher, FHS Hochschule für Angewandte Wissenschaften, Tellstrasse 2, CH-9001 St.Gallen

logie geworden, die Beruf und Alltag in einem Ausmass prägt, das damals unvorstellbar war. Anfänglich wurden die Rechner mit elektromechanischen Relais aufgebaut, kurze Zeit später mit Elektronenröhren und ab etwa 1960 mit integrierten elektronischen Schaltungen. Entscheidend war dabei die Erfindung des Transistors um 1947, die Shockley, Bardeen und Brattain den Physik-Nobelpreis einbrachte. Rechengeschwindigkeit und Zuverlässigkeit nahmen sprunghaft zu, die Kosten ab. MOORE (1965) formulierte die nach ihm benannte Regel, dass die Transistordichte sich alle 18 Monate verdoppelt. Weshalb diese Regel volle weitere 40 Jahre gültig bleiben sollte, ist selbst für ihren Urheber unerklärlich.

In dieser Zeit hat sich die Rechnerleistung (gemessen in Fliesskomma-Operationen pro Sekunde) um den unglaublichen Faktor 10^7 vergrössert. Und der Nutzen? Bereits 1955 wurde von MCCARTHY, MINSKY, ROCHESTER UND SHANNON (1955) gefordert, dass 10 Männer (!) während 2 Monaten (!) alle Aspekte des Lernprozesses und menschlicher Intelligenz beschreiben sollen, so dass anschliessend intelligentes Verhalten auf einem Rechner simuliert werden kann. Rechner sollten nicht nur schneller, sondern auch intelligenter werden. Die gleiche Gruppe hat den Begriff ‹künstliche Intelligenz› geprägt.

2. Natürliche Neuronale Netze: biologische Vorbilder

Die Idee, gehirn-ähnliche Funktionen vereinfacht auf dem Rechner nachzubilden, wurde schon 1943 von McCullochs und Pitts geäussert (MCCULLOCH-PITTS 1943), wurde aber mangels technischer Möglichkeiten nicht realisiert. Erst 1957 wurde ein erster Rechner, dessen Architektur sich an biologischen Vorbildern orientierte, zur Mustererkennung eingesetzt. Die Motivation ist einleuchtend, denn das Gehirn selbst von niederen Tieren erbringt erstaunliche Leistungen: Eine Stubenfliege fliegt, weicht einem Hindernis aus, pflanzt sich fort. Ein konventioneller Rechner hingegen war gerade mal in der Lage, zwei Speicherzellen zu addieren und den Inhalt von einer Speicherzelle in die andere zu verschieben – das allerdings sehr schnell und zuverlässig. Offensichtlich nutzt die Natur die beschränkte Hardware sehr effizient.

Schnell wurde klar, dass selbst das Stubenfliegengehirn viel zu komplex ist, um als Ganzes auf einem Rechner abgebildet zu werden. Es konnte also nur darum gehen, Konzepte und Strukturen des biologischen Vorbilds zu übernehmen und daraus radikal vereinfachte ‹künstliche neuronale Netze› zu entwickeln. Nichtsdestotrotz – die Parallelen zwischen künstlichen und natürlichen neuronalen Netzen sind frappant:

Wie das biologisches Gehirn ist auch das künstliche neuronale Netz aus einfachen Elementen – Neuronen – aufgebaut. Ein Neuron bezieht Signale von Nachbarneuronen, die mit ihm verschaltet sind. Die Eingangssignale werden zu einem Ausgangssignal verarbeitet, und das Ausgangssignal wird an weitere Nachbarneuronen weitergegeben.

Das neuronale Netz enthält freie Parameter, die festlegen, wie das Eingangssignal zu einem Ausgangssignal verarbeitet wird. Der Vorgang, die Parameter zu bestimmen, wird als ‹Lernen› bezeichnet. Das neuronale Netz erhält seine Funktionalität nicht durch explizite Programmierung, sondern durch Lernen an Beispielen.

3. Mathematischer Hintergrund

Weil künstliche Neuronen auf sehr unterschiedliche Weise miteinander verbunden werden können, ist eine unübersehbare Menge von Netzwerktopologien entstanden. Meist werden diese eingeteilt in:

- überwacht lernende und nicht überwacht lernende sowie
- rückgekoppelte und nicht rückgekoppelte neuronale Netze.

Aus Platzgründen beschränkt sich diese Arbeit auf Multilayer Perceptrons (MLP), einen Vertreter der überwacht lernenden, nicht rückgekoppelten Netze. MLPs sind die am besten untersuchten und am häufigsten eingesetzten neuronalen Netze.

In der oben erwähnten Arbeit von MCCULLOCH, PITTS (1943) arbeiteten zwei Fachleute aus Neurophysiologie und Mathematik zusammen, um nach dem Vorbild biologischer Neuronen ein einfaches mathematisches Neuron zu modellieren, welches mit anderen solchen Neuronen via synaptische Verbindungen kommunizieren kann.

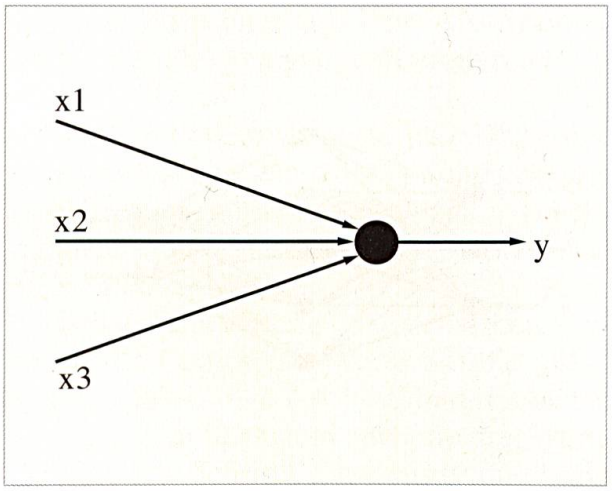

Abbildung 1:
Mathematisches Modell-Neuron.

3.1 Netzarchitektur

In heutiger Darstellung (vergleiche zum Beispiel HAYKIN (1995)) wird ein mathematisches Modell-Neuron wie in Abbildung 1 veranschaulicht.

Inputs $x_1, x_2, ..., x_n$ von anderen Neuronen werden verarbeitet zu einem Output y. Allerdings werden nicht alle Inputs gleich stark gewichtet. Beschreibt man die Verarbeitung mit Hilfe einer so genannten *Aktivierungsfunktion f* (oft auch *Transferfunktion* genannt) und die Gewichtung mit Zahlen $w_1, w_2, ..., w_n$ so lautet die mathematische Notation für dieses Neuronen-Verhalten:

$$y=f(w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n)$$

In Anlehnung an das biologische Vorbild soll das Modellneuron nur dann einen Ausgabewert liefern, wenn alle Inputs zusammen eine gewisse Grenze (Schwellwert oder *Bias* genannt) überschritten haben. Wie in Abbildung 2 gezeigt, wird der Output des Neurons erst dann zu Eins, wenn der Gesamtwert u aller Inputs den Wert b übersteigt.

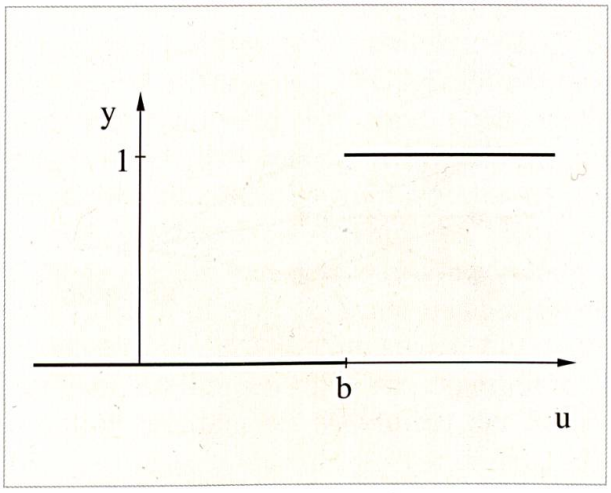

Abbildung 2:
Neuron-Aktivität ab einer gewissen Grenze b.

In mathematischer Notation ausgedrückt: Für $u=w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n$ ist

$$y = \begin{cases} 0, \textit{wenn } u<b \\ 1, \textit{wenn } u \geq b \end{cases}$$

MCCULLOCH und PITTS nennen dies ein ‹Alles-oder-nichts›-Ereignis: Das Neuron gibt einen Impuls weiter – oder nicht.

Allerdings verursachen solche unstetigen

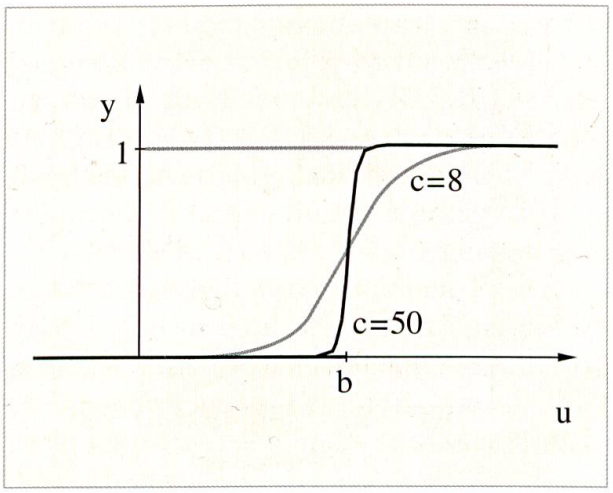

Abbildung 3:
Approximation der Neuron- Aktivität.

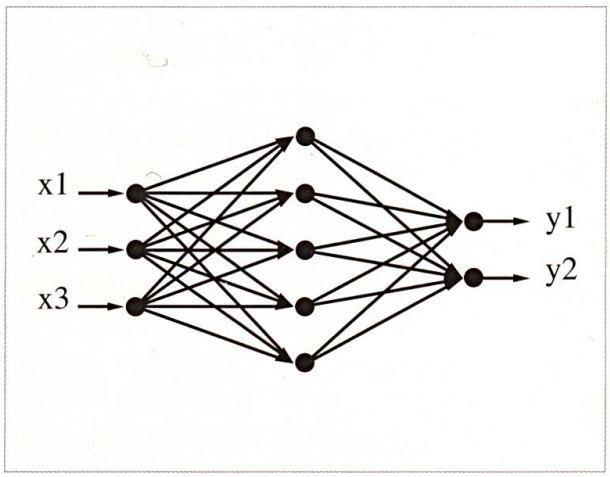

Abbildung 4:
Zusammengeschaltete mathematische Modell-Neuronen.

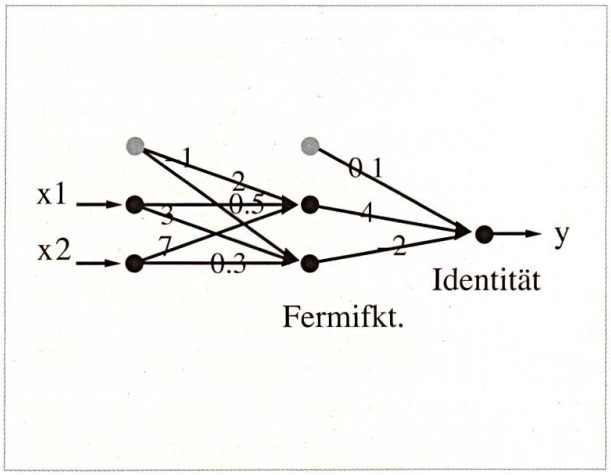

Abbildung 5:
Ein erstes vollständiges Multilayer Perceptron (MLP).

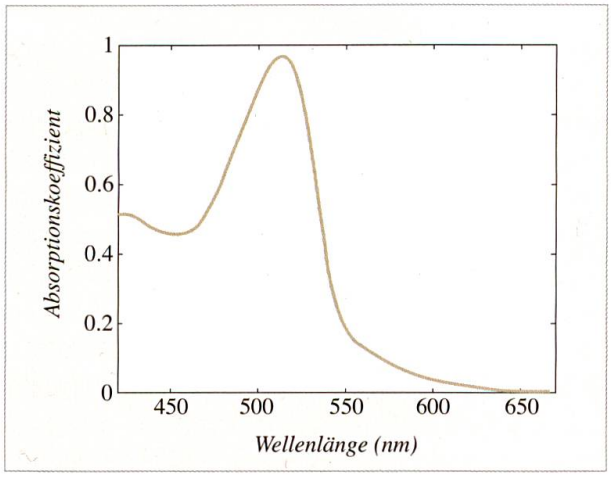

Abbildung 6:
Messwerte aus einer geeignet vorbereiteten Blutprobe (nach PURDIE et al. [1992]).

Funktionen Probleme, weil sie nicht ableitbar sind. Deshalb wird die oben eingeführte Funktion angenähert durch:

$$y = \frac{1}{1 + e^{-c\cdot(u-b)}}$$

für eine geeignete Zahl c. Wie Abbildung 3 zeigt, wird diese Approximation mit $c=50$ schon recht gut – und umso besser, je grösser c gewählt wird. Oft ist ein gradueller Übergang vom Funktionswert 0 auf 1 geradezu erwünscht. In solchen Fällen wird c als freier Parameter betrachtet, der an die konkreten Anforderungen angepasst wird.

Die eben verwendete Funktion heisst *logistische Funktion* oder *Fermi-Funktion*:

$$f(u) = \frac{1}{1 + e^{-u}}$$

In technischen Anwendungen wird die Fermi-Funktion oft ersetzt durch $y=\tanh u$, da deren Punktsymmetrie in Bezug auf den Koordinatenursprung mitunter als Vorteil betrachtet wird.

Nun werden die einzelnen Neuronen zu einem Netz zusammengeschaltet. Dabei kann der Output eines Neurons an verschiedene andere Neuronen weitergegeben werden. Je nach Art und Weise wie das geschieht, entsteht eine andere so genannte Netzwerkarchitektur. Im Folgenden wird hier nur eine Bauart betrachtet, die eines *Multilayer-Perceptrons* (kurz: *MLP*): Neuronen werden schichtweise angeordnet und jede Schicht liefert ihre Outputs an die nächste Schicht vorwärts wie in Abbildung 4 gezeigt. Die Input-Schicht enthält hier 3 Neuronen, die nächste – so genannte *verdeckte Schicht* – 5 Neuronen und die Output-Schicht 2 Neuronen.

Die Anzahl der Neuronen pro Schicht ist beliebig und es können auch mehrere solcher verdeckter Schichten auftreten. In technischen Anwendungen wird die Aktivierungsfunktion in der ersten (und oft auch in der letzten) Schicht abgeändert zur *Identität* $f(u)=u$.

Für jede der gezeichneten synaptischen Verbindungen existiert gemäss obigen Ausführungen eine Zahl, die für die Gewichtung

dieser Verbindung zuständig ist, und für jedes Neuron der Input-Schicht sowie jedes der verdeckten Schicht(en) eine Zahl, die den zugehörigen Bias beschreibt. Zusammen mit der Festlegung der Transferfunktionen ist damit das MLP vollständig festgelegt. Abbildung 5 zeigt ein einfaches, rein theoretisches Beispiel - sozusagen als Fingerübung. Synapsen, welche von grau gezeichneten Neuronen ausgehen, enthalten dabei die Bias-Werte.

Bezeichnet man die Fermifunktion mit f, so lautet die rechnerische Vorschrift für dieses einfache Netz schon recht kompliziert:

$$y = 0.1 + 4 \cdot f(2 + 0.5x_1 + 7x_2) - 2 \cdot f(-1 + 3x_1 + 0.3x_2)$$
$$= 0.1 + \frac{4}{1+e^{-2-0.5x_1-7x_2}} - \frac{2}{1+e^{1-3x_1-0.3x_2}}$$

3.2 Eine typische Aufgabenstellung für Multilayer-Perceptrons

Obwohl das bisher dargestellte mathematische Modell die Vernetzung biologischer Neuronen nur sehr rudimentär erfasst, ist es doch schon in der Lage, gewisse schwierige Probleme der Praxis immerhin näherungsweise zu lösen. Mit dem Begriff ‹näherungsweise› ist auch gleich angedeutet, dass Neuronale Netze nur dann zur Anwendung kommen sollen, wenn keine andere, exakte Theorie zu einer Lösung verhilft. Die folgende Problemstellung zeigt eine typische solche Situation:

In vielen Zusammenhängen sind benötigte Messwerte nur sehr schwierig oder teuer zu bekommen und man würde sich wünschen, diese Werte rechnerisch voraussagen zu können aus Messgrössen, welche auf einfacherem Weg, schnell und günstig zu gewinnen sind. Matlab®, eine weit verbreitete Software für die Simulation neuronaler Netze, greift in ihrer Dokumentation (MATLAB 2005) ein Beispiel von PURDIE et al. (1992) auf: Es wird beschrieben, wie teuer und zeitintensiv es damals war, Cholesterol-Mengen im Blut eines Patienten zu messen. Sie schlugen vor, diese Werte näherungsweise zu berechnen aus dem Absorptionsspektrum einer geeignet vorbereiteten Blutprobe. Im Detail:

Von 264 Patienten wurde eine Blutprobe entnommen und ein Absorptionsspektrum (Abbildung 6) gemessen. Pro Person wurden daraus bei 21 Wellenlängen Absorptionskoeffizienten entnommen.

Gleichzeitig wurden von denselben Personen die Cholesterol-Werte auf die altbekannte, aufwändige Art bestimmt. Es liegen also zu jedem Spektrum auch die drei Werte für very-low-densitiy, low-density und high-density Cholesterol-Anteile vor. Gesucht ist jetzt ein Neuronales Netz, welches aus 21 Eingängen des Spektrums drei Ausgänge mit den Cholesterol-Werten berechnet.

Man erkennt aus der Problemstellung sofort einen Teil der Netzarchitektur: 21 Eingänge und 3 Ausgänge. Die Anzahl der verdeckten Neuronen ist aber noch völlig ungewiss, ebenso wie die richtige Wahl der Gewichte und Bias-Werte. Letzteres ist das Thema des nächsten Abschnittes. Über die richtige Anzahl verdeckter Neuronen hingegen kann in diesem Rahmen nicht wirklich eingegangen werden. Nur soviel: Für reale Aufgabenstellungen muss sie experimentell ermittelt werden mit Methoden der Statistik.

3.3 Backpropagation

Backpropagation als Technik zur Bestimmung der Gewichte eines MLP beruht auf dem in der Mathematik bekannten so genannten Gradientenverfahren. Im Umfeld Neuronaler Netze wurde es vor allem geläufig durch ein Paper von RUMELHART, HINTON UND WILLIAMS (1986). Dieses Gradientenverfahren soll hier anhand eines möglichst einfachen Beispiels erklärt werden – so nämlich, dass die Überlegungen graphisch dargestellt werden können. Rein rechnerisch sind sie dann sehr einfach auf die allgemeine Situation übertragbar: Anstelle von zweidimensionalen Vektoren, wie sie hier vorkommen werden, rechnet man mit hochdimensionalen.

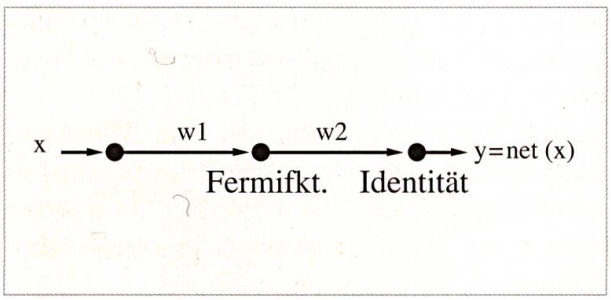

Abbildung 7:
Trivialbeispiel eines Multilayer Perceptrons (nur zu Demonstrationszwecken).

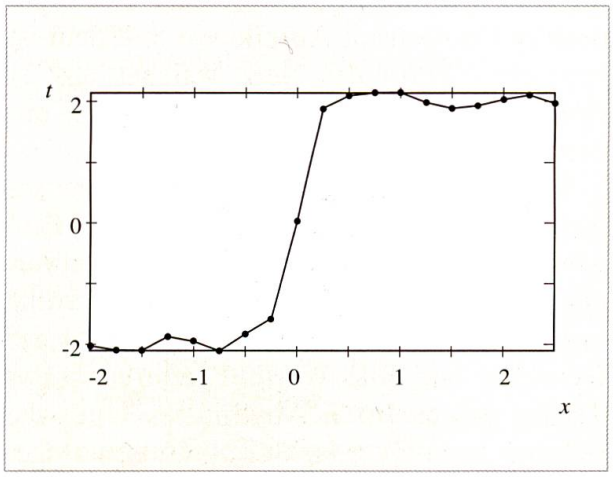

Abbildung 8:
Messdaten für Eingang x und Zielgrösse t im obigen Trivialbeispiel.

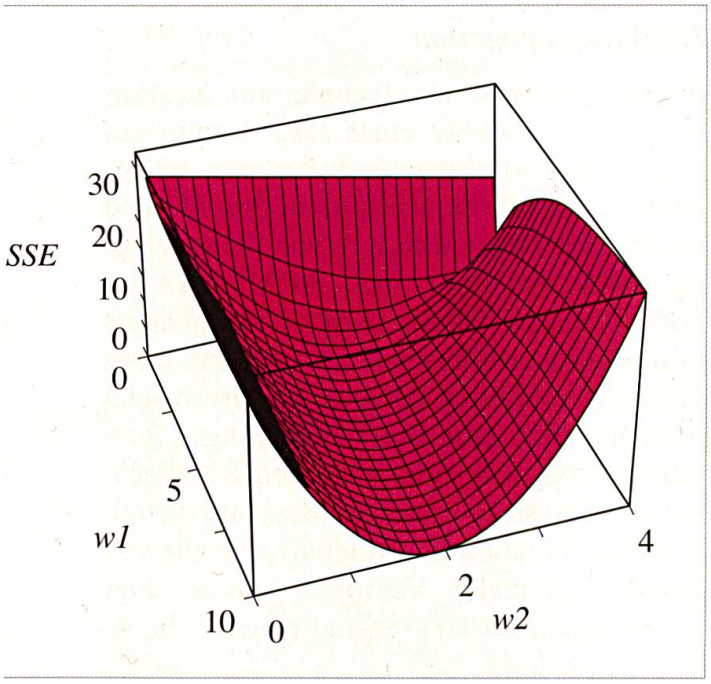

Abbildung 9:
Summe aller Fehlerquadrate SSE als Funktion der Gewichte

Angenommen, ein neuronales Netz habe (wie in Abbildung 7 dargestellt) je einen Ein- und Ausgang, ein verdecktes Neuron und keine Bias-Werte.

Weiter seien Messdaten $x_1, x_2, ..., x_{10}$ gegeben für eine einfach zu messende Grösse x und zugehörige schwierig zu messende Werte $t_1, t_2, ..., t_{10}$ einer Zielgrösse t (Target), von der man weiss, dass sie von x abhängig ist. Eine graphische Darstellung davon könnte zum Beispiel für Messungen, die leicht fehlerbehaftet sind, so aussehen wie in Abbildung 8.

Die Gewichte w_1 und w_2 im Trivialbeispiel sollen nun so eingestellt werden, dass dieses Netz die gegebenen Daten einerseits gut wiedergibt, aber auch die Messfehler einigermassen ausgleicht. Das Netz soll also die Rolle eines Funktionsapproximators übernehmen. Das heisst, es soll $y_i \approx t_i$ sein für $i=1, ..., n$. Zu diesem Zweck wird für jeden Messpunkt der Fehler $y_i - t_i$ betrachtet, beziehungsweise sein Quadrat und alle diese Fehlerquadrate werden aufsummiert zur Fehlerquadratsumme $(y_1 - t_1)^2 + ... + (y_{10} - t_{10})^2$.

Dieser Ausdruck ist sicher dann klein, wenn alle Fehler klein sind. Umgekehrt: Wenn die Summe klein ist, so müssen die einzelnen Summanden klein sein, da jeder Summand nur positiv oder Null sein kann. Deshalb setzt man sich jetzt das Ziel, diesen Term zu minimieren. Aus mathematischer Sicht ist die Fehlerquadratsumme (SSE genannt, für sum squared error) eine Funktion der beiden Variablen w_1 und w_2 deren Minimum gesucht ist. Im Fall des obigen Trivialbeispiels kann man die Fehlerquadratsumme graphisch darstellen. Zu jedem Wertepaar (w_1, w_2) wird der SSE in Richtung einer dritten Koordinatenachse aufgetragen. Die Abbildungen 9 und 10 zeigen die entstehende Fläche von zwei verschiedenen Standpunkten aus. Weiter hat Abbildung 10 in der w_1 w_2 -Ebene die zugehörigen Höhenlinien eingetragen (wie wenn die darüber liegende Fläche als Ausschnitt aus einem Gebirge interpretiert würde mit darunter liegender Karte im Massstab 1:1).

Wo liegt nun der Punkt (w_1, w_2) mit minimalem SSE-Wert? Die Frage ist gar nicht so

einfach zu beantworten, da es kein exaktes Verfahren gibt, das für alle je auftretenden solchen Flächen die Aufgabe löst. Man ist auf numerische Näherungslösungen angewiesen. Zudem sieht man in obigen Graphiken, dass auch von Auge das Minimum nicht klar zu finden ist: Wo in diesem ‹langen Tal› ist denn der tiefste Punkt?

Um das erwähnte numerische Verfahren zu verstehen, betrachten wir ein einfacheres Beispiel, das nichts zu tun hat mit neuronalen Netzen: In Abbildung 11 ist eine Fläche gegeben, von der man aus der Graphik die Minimalstelle sofort sieht – oder mindestens vermutet.

Die Minimalstelle scheint bei $w_1=0$, $w_2=0$ zu liegen. Das wird auch rechnerisch sofort bestätigt, da für diese spezielle Fläche die z-Koordinate gerechnet wurde als $SSE=0.3 \cdot w_1^2+w_2^2$. Gesucht ist nun ein numerisches Verfahren, mit dem der Computer diese Stelle ebenfalls findet. Man verwendet hier Ideen, die jedermann vom Lesen von Landkarten her kennt: Wenn man sich auf der Karte senkrecht zu den Höhenlinien abwärts bewegt, so hat man den Weg des steilsten Abstiegs gewählt. Mathematisch wird diese Idee umgesetzt durch die Berechnung des Gradienten der Funktion $SSE(w_1, w_2)$: Man wählt eine beliebige Startstelle für den Punkt (w_1, w_2), geht ein Stück weit in die Gegenrichtung des Gradienten (also senkrecht zur ersten Höhenlinie) und berechnet dann an der neuen Stelle wieder den Gradienten. Dieser zeigt jetzt wahrscheinlich in eine leicht andere Richtung, und man bewegt sich wieder ein (kleineres) Stück weit in die Gegenrichtung des neuen Gradienten. Durch Wiederholung dieser Idee hofft man, sich sukzessive der Minimalstelle zu nähern. Die Abbildungen 12 und 13 zeigen dieses Vorgehen für den Startwert $w_1=-1.8$, $w_2=1.5$, in Abbildung 12 die zugehörige 3D-Darstellung, in Abbildung 13 nur noch das Höhenlinienbild.

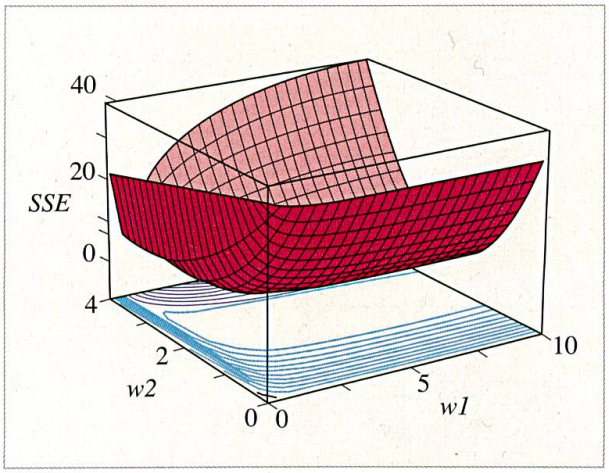

Abbildung 10:
Summe aller Fehlerquadrate SSE (aus neuem Blickwinkel).

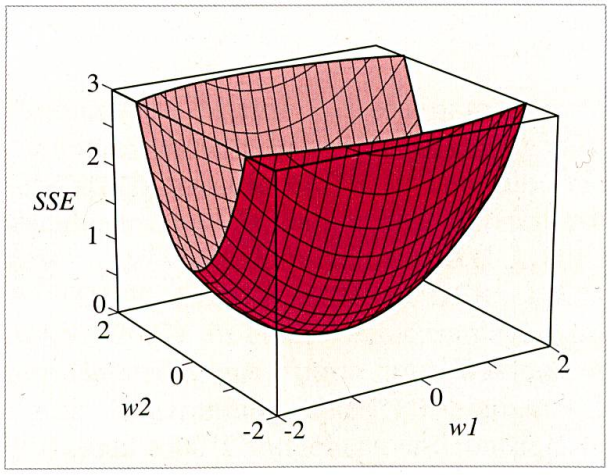

Abbildung 11:
Funktion mit leicht zu erratender Minimalstelle.

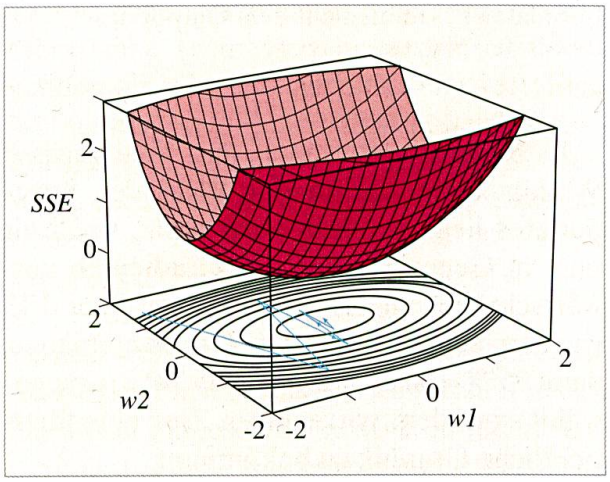

Abbildung 12:
Gradientenverfahren zur Auffindung der Minimalstelle (3D-Darstellung).

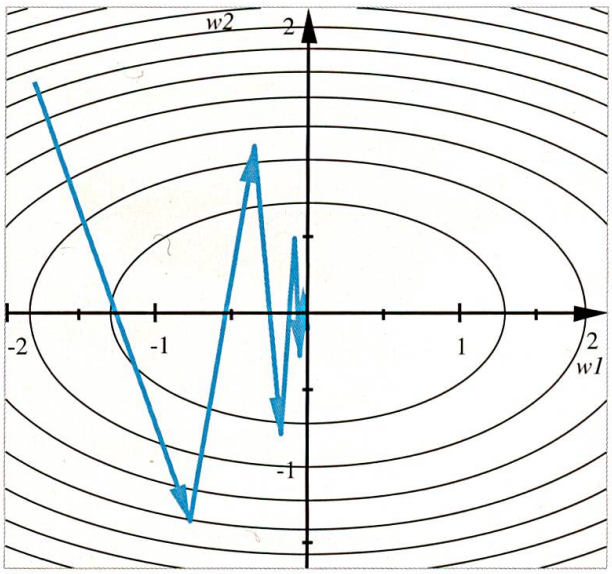

Abbildung 13:
Gradientenverfahren zur Auffindung der Minimalstelle (2D-Darstellung).

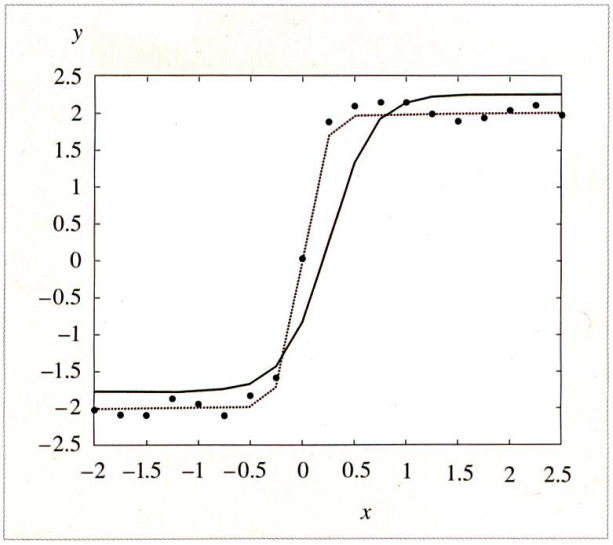

Abbildung 14:
Lerndaten zusammen mit dem Output des trainierten Netzes.

Es bleibt nachzutragen, dass eine gewisse Willkür nicht nur in der Wahl des Startpunktes liegt, sondern auch darin, wie weit man in Gegenrichtung des Gradienten vorwärtschreitet. In den Abbildungen 12 und 13 wurden zuerst 95% der Gradientenlänge, dann $95^2\% = 90.25\%$, $95^3\% = 85.74\%$, usw. gewählt - mit dem vorrangigen Ziel, eine übersichtliche Graphik zu bekommen.

In mathematischer Notation lautet das Gradienten-Verfahren:
Wähle einen Startwert für den Näherungsvektor $\vec{w}^{(0)} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$

Wenn ein Näherungsvektor $\vec{w}^{(n)}$ vorliegt, so berechnet sich der nächste Näherungsvektor durch $\vec{w}^{(n+1)} = \vec{w}^{(n)} - \lambda_n \cdot \text{grad SSE}\,(\vec{w}^{(n)})$

Die Zahl λ_n heisst im Kontext Neuronaler Netze *Lernrate*. (In obigen Beispiel war also $\lambda_1 = 0.95$, $\lambda_2 = 0.95^2$, $\lambda_3 = 0.95^3$,...) Das Anpassen der Gewichte wird *Lernen* genannt und die obige Formel, die besagt, wie man aus $\vec{w}^{(n)}$ den nächsten Satz $\vec{w}^{(n+1)}$ von Gewichten berechnet, *Lernregel*. Die Messdaten schliesslich, mit welchen der *SSE* als Funktion der Gewichte berechnet wurde, heissen *Lernmuster* oder *Lerndaten*. Ist das Gradienten-Verfahren abgeschlossen, heisst das Netz *trainiert*. Hinter diesen Namensgebungen steckt die Idee, dass auch der Mensch in vielen Fällen nur aufgrund von einzelnen Beispielen lernt. Je mehr Beispiele vorliegen, desto besser ist (hoffentlich oder vielleicht) der Lerneffekt.

Professionelle Neuro-Software hat raffinierte Varianten des Gradientenverfahrens implementiert und wählt auch den Startvektor für die Gewichte nicht völlig nach Zufallsprinzip. Setzt man zum Beispiel die Neural Network Toolbox von MATLAB® auf das oben eingeführte Trivialbeispiel eines Netzes an, so erhält man als Ergebnis die in Abbildung 14 dargestellte Funktion $y(x)$. Die ‹Messdaten› (als Punkte gezeichnet) wurden erstellt durch Verrauschen von Funktionswerten der gestrichelten Kurve. MATLAB findet daraus gemäss dem oben besprochenen Vorgehen für die Gewichte die Werte $w_1 = 3.3634$, $w_2 = 1.0217$ und die durchgezogene Kurve stellt den Output des damit gebauten Netzes dar, also die Funktion $y = net(x)$.

An diesem Beispiel sind schon die ersten Schwierigkeiten erkennbar, welche bei der Bestimmung der Gewichte auftreten:
Bei schlecht gewählter Lernrate wird die Lösung nicht gefunden.

Wenn die Funktion *SSE* (in Abhängigkeit von den Gewichten und Biaswerten) mehrere lokale Minimas hat, kann es sein, dass man – je nach Startwert – nur eine solche Stelle findet und nicht das gesuchte absolute Minimum der Funktion.

Zum Schluss dieses Abschnittes noch eine Bemerkung zur Namensgebung: Im Kontext Neuronaler Netze heisst das Gradienten-Verfahren *Backpropagation*. Der Grund dafür liegt beim mathematischen Handwerk. Um den Gradienten zu bilden, braucht man die partiellen Ableitungen nach allen Variablen, also auch nach den Gewichten der ersten Schicht. Dabei muss die Kettenregel verwendet werden: Man arbeitet sich beim Ableiten von verschachtelten Funktionen von aussen nach innen durch. Im Netz heisst das, dass man von der Ausgangsschicht her rückwärts rechnet bis zum entsprechenden Gewicht. Man ‹back-propagiert›.

4. Industrielle Anwendungen

4.1 Herstellung von Nanopartikeln

Nanopartikel sind feste Teilchen mit Abmessungen zwischen 1 nm und 100 nm. Im Vergleich zum Volumen haben sie eine grosse Oberfläche: Silizium-Pulver erreicht zum Beispiel eine spezifische Oberfläche von weit über 100 m^2/g. Nanopartikel sind deshalb enorm reaktionsfähig und haben völlig andere chemische und physikalische Eigenschaften als der Basiswerkstoff. Sie sind Ausgangsmaterial für eine unüberblickbare Anzahl Anwendungen in Medizin, Werkstofftechnologie, Oberflächentechnologie und als Katalysatoren in der Chemie. Seit etwa 2004 werden allerdings vermehrt Risiken thematisiert – unter anderem, weil Nanopartikel wegen ihrer Kleinheit die Blut-Hirnschranke durchbrechen können. (BUNDESAMT FÜR GESUNDHEIT 2006)

Zur Herstellung von Nanopartikeln sind eine Reihe von Verfahren entwickelt worden (siehe PASCHEN, COENEN, FLEISCHER (2004)). Hohe Produktionsraten – im Bereich

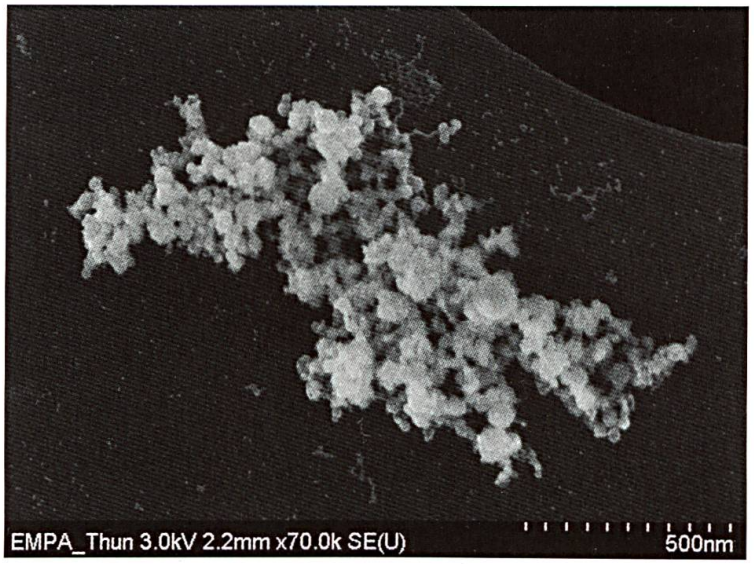

Abbildung 15:
Elektronenmikroskopische Aufnahme von Silizium-Nanopulver (Aufnahme: EMPA Thun).

Gramm pro Minute – werden in heissen Flammen erreicht. In einem Forschungsprojekt optimieren die EMPA in Thun und die Fachhochschule St.Gallen den Herstellprozess für Silizium-Nanopartikel (LEPAROUX, LOHER, SCHREUDERS, SIEGMANN 2007). In einer Versuchsanlage wird ein Gasstrom aus Argon und Wasserstoff durch Einstrahlung eines HF-Signals (13 MHz) auf 3000° C aufgeheizt und ionisiert. In den Plasmastrom eingebrachtes Silizium-Pulver (ursprünglicher Durchmesser um 45 Mikrometer) schmilzt und verdampft. Nach einer Flugstrecke von wenigen Zentimetern kondensiert Silizium zu kleinen Partikeln. Ziel ist, Nanopartikel mit hoher spezifischer Oberfläche (gemessen in in g/m^2, im Folgenden als y bezeichnet), mit eng verteilten Abmessungen und möglichst hoher Produktionsrate zu erzeugen. Es wurden 27 Versuche durchgeführt, bei denen die Prozessparameter

x1: Ar-H_2 Gasstrom-Stärke
x2: Feedrate des Si-Pulvers

variiert wurden. Die Feedrate bezeichnet die Masse des Pulvers, das pro Zeiteinheit in den Gasstrom eingeschossen wird. In jedem Experiment wurde die spezifische Oberfläche

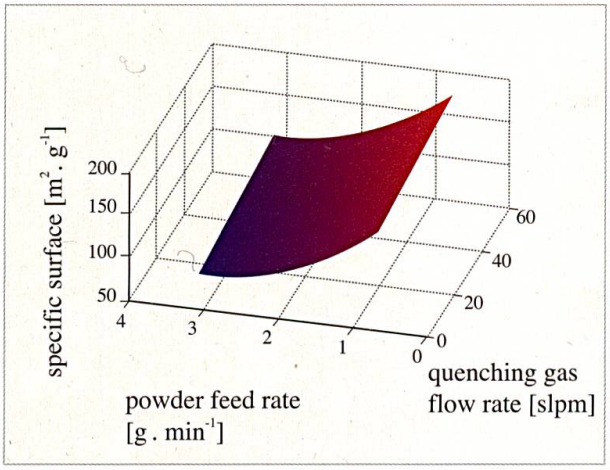

Abbildung 16:
Prozessfläche erzeugt durch lineare Regression mit signifikanten quadratischen Regressoren.

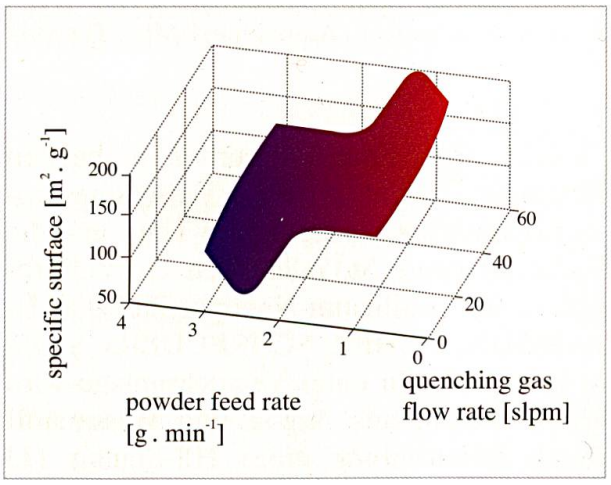

Abbildung 17:
Prozessfläche erzeugt durch ein MLP mit 2 verdeckten Neuronen.

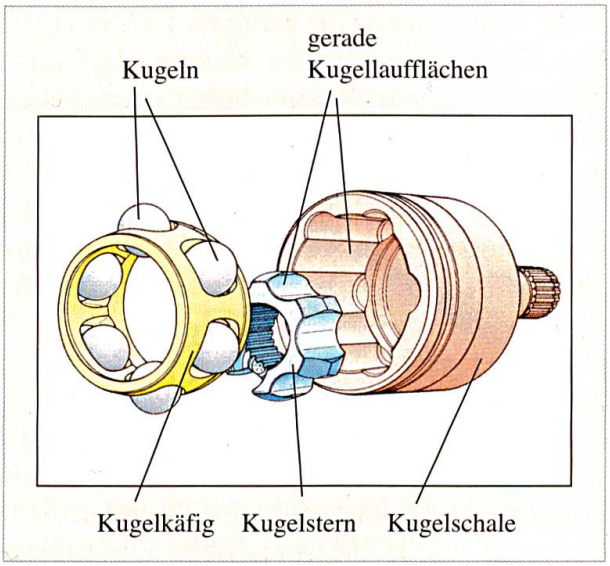

Abbildung 18:
Einsatz einer Kugelnabe in einem Kugelgelenk.

in m^2/g gemessen. Mit diesen Daten wurden einerseits lineare Regressionsmodelle berechnet und anderseits ein Multilayer-Perceptron trainiert.

Lineare Regression

Die lineare Regressionsrechnung benutzt den Ansatz

$$y = a_1 \cdot x_1^2 + a_2 \cdot x_2^2 + a_{1,2} \cdot x_1 \cdot x_2 + b_1 \cdot x_1 + b_2 \cdot x_2 + c$$

y spezifische Oberfläche [m^2/g]
x_1 Gasstromstärke [Standardliter/min]
x_2 Pulver-Förderrate [g/min]

Die insgesamt 6 Koeffizienten $a_1, a_2, a_{1,2}, b_1, b_2$, und c werden nach der Methode der kleinsten Fehlerquadrate bestimmt: Die quadrierte Abweichung zwischen Rechnung und Experiment, summiert über alle 27 Datenpunkte, soll minimal sein. Anschliessend werden alle Summanden gestrichen, deren Koeffizient auf einem 95 %-Signifikanzniveau nicht von null verschieden sind. Das führt zu folgender Prozessfläche:

$$y = 13.6 \cdot x_2^2 + 0.473\, x_1 - 82.1 \cdot x_2 + 208.9 \cdot x_1$$

Es ist zu beachten, dass die lineare Regressionsrechnung eine nichtlineare und somit gekrümmte Regressionsfläche erzeugt – dies wegen des quadratischen Terms in der Prozessfläche. Der Begriff ‹linear› bezieht sich auf die 6 unbekannten Koeffizienten a_i, b_j und c.

Abbildung 16 zeigt, dass grosse spezifische Oberflächen (und damit kleine Partikelgrössen) bei niedriger Pulver-Förderrate und hoher Gasstrom-Stärke auftreten. Das Ergebnis deckt sich mit der Erwartung: Je tiefer ist die Produktivität in g/min, desto kleiner fallen die Partikel aus und desto höher wird die spezifische Oberfläche in g/m^2.

Multilayer-Perzeptron

Wie oben diskutiert ist die erste Aufgabe bei der Berechnung eines MLP die Festlegung der Architektur. Die schwierigste Aufgabe besteht darin, die Schicht mit den verdeckten Neuronen zu dimensionieren. Je grösser die Zahl der Neuronen in der einzigen verdeckten Schicht eines MLP gewählt wird, desto kleiner wird die Abweichung zwischen

Rechnung und Experiment. Allerdings sollte vermieden werden, dass Eigenheiten der Lerndaten zwar sehr gut, neue Daten aber schlecht dargestellt werden. Dieses Phänomen wird als ‹Overfitting› bezeichnet und kann auftreten, wenn zu viele freie Parameter angepasst werden. Allerdings besteht keine theoretisch unterlegte, allgemein gültige Regel, wie viele Datensätze für einen freien Parameter erforderlich sind. Als heuristische Regel nützlich ist die Forderung, dass bei normalverteilten Abweichungen zwischen Messung und Rechnung deren Standardabweichung grösser sein muss als die Standardabweichung des Messfehlers. Salopp: Es macht keinen Sinn, präziser zu modellieren als gemessen worden ist. Overfitting kann mit einem Resampling-Verfahren vermieden werden. Dabei hat sich gezeigt, dass für die vorliegenden Daten ein MLP mit n=2 Neuronen in der verdeckten Schicht optimal ist.

Im Vergleich zwischen den beiden durch lineare Regression und durch ein MLP erzeugten Prozessflächen fällt auf, dass die Grobstruktur gleich ist: eine grosse spezifische Oberfläche wird in erster Linie mit kleinen Förderraten und in zweiter Linie mit grossen Gasströmen erreicht. Die mit einem MLP erzeugte Prozessfläche (Abbildung 17) zeigt aber entscheidende zusätzliche Details:

1. Tiefe Förderrate führen zu grossen spezifischen Oberflächen, aber unterhalb von 1.5 g/min hängt das Ergebnis nicht mehr entscheidend von der Förderrate ab.
2. Je grösser die Gasstromstärke, desto grösser die spezifische Oberfläche. Zwischen 10 slpm und 60 slpm vergrössert sich die spezifische Oberfläche um 10%.
3. Bei hoher Gasstromstärke und ganz tiefer Förderrate (rechte hintere Ecke in Abbildung 17) zeigt die Prozessfläche ein vom Gesamtbild abweichendes Verhalten. Bei der jetzigen Datenlage ist zu vermuten, dass es sich dabei um einen statistischen Artefakt handelt. Nur ein zusätzliches Experiment könnte Klarheit schaffen.

4.2 Kaltumformen von Kugelnaben

Eine Kugelnabe ist Teil eines Kugelgelenks (Abbildung 18), das bei einem frontangetriebenen Fahrzeug das Drehmoment von der Vorderachse auf die gelenkten und gefederten Räder überträgt. Der weltweite Markt für Kugelnaben hat eine Volumen von etwa 110 Mio/Jahr. Solche hoch beanspruchten, komplex geformten Teile (Abbildung 19) in grosser Stückzahl prozesssicher und präzis zu fertigen ist eine grosse Herausforderung – insbesondere am Standort Mitteleuropa und gegen weltweite Konkurrenz in einem attraktiven und entsprechend hart umkämpften Markt.

Bei der Firma ThyssenKrupp Presta in Eschen/LI werden Kugelnaben mit einem Kaltmassiv-Umformverfahren einbaufertig hergestellt. Dabei wird ein Rohling, ein Stahlzylinder mit 30 mm Durchmesser und etwa 50 mm Länge in ein Werkzeug eingelegt, das eine Hohlform definiert. Mit einer Kraft von 40 MN (400 t) wird das Metall innerhalb von etwa 0.2 Sekunden in die endgültige Form gebracht.

Die Abmessungen der fertigen Kugelnabe sind qualitätsrelevant. So ist der Abstand gegenüberliegender gekrümmter Flächen auf +/– 0.01 mm toleriert. Primär ist für die Formgebung das Werkzeug verantwortlich. Innerhalb gewisser Grenzen haben Prozessparameter Auswirkung auf die Teilegeometrie.

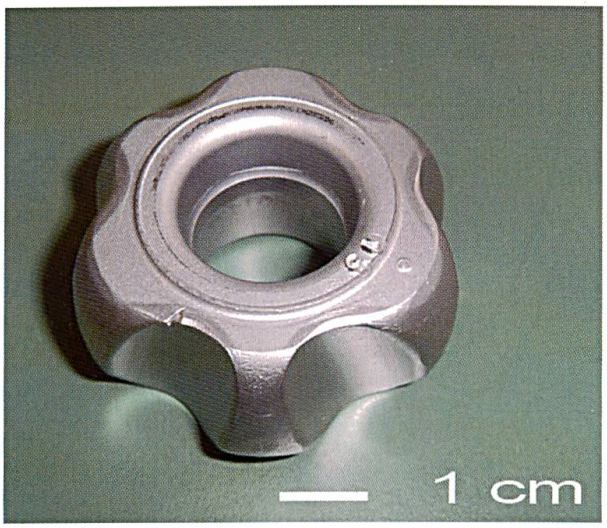

Abbildung 19:
Einbaufertige Kugelnabe.
Foto: Christoph Battaglia, NTB.

Abbildung 20:
Links der Rohling, rechts unterschiedlich ausgepresste Kugelnaben.
Foto: Christoph Battaglia, NTB.

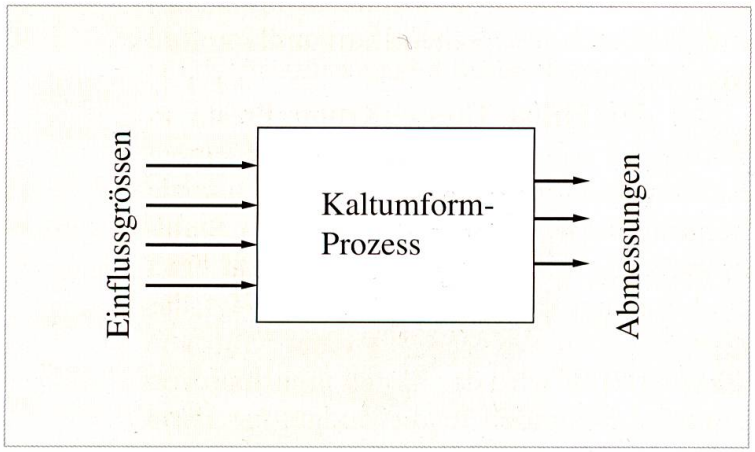

Abbildung 21:
Black-Box-Sicht des Kaltumform-Prozesses.

Projektidee

Intuitiv einleuchtend und experimentell bestätigt ist, dass eine höhere Einpresskraft zu grösserem Teiledurchmesser führt. Allgemeiner besteht die Projektidee darin, gezielt gewisse Teileabmessungen durch Variation der Einflussgrössen zu steuern. Um das zu erreichen, muss die Beziehung zwischen Einflussgrössen und Zielgrössen (hier: Abmessungen) bekannt sein. Eine solche Beziehung wird als Prozessmodell bezeichnet und hier in Form eines neuronalen MLP realisiert (Abbildung 21).

Mit einem solchen Modell im Hintergrund können Störungen im Prozess (z.B. durch veränderte Werkstoffeigenschaften einer neuen Materialcharge) auskorrigiert werden. Dabei wird in Kauf genommen, dass diese Störungskorrektur schrittweise erfolgt.

Projektidee war, mit 13 Versuchen ein grobes Prozessmodell für 4 verschiedene Einflussgrössen zu erstellen und die Störungskompensation schrittweise durchzuführen.

Ergebnis

Abbildung 22 zeigt, dass die Projektidee realisierbar ist: Die blaue Kurve ist die Idealkontur. Nach einer Störung ergeben sich Teile mit der grünen Kontur. Eine erste Kompensation liefert Teile mit der roten Kontur. Nach der zweiten Kompensation sind die Teile wieder masshaltig.

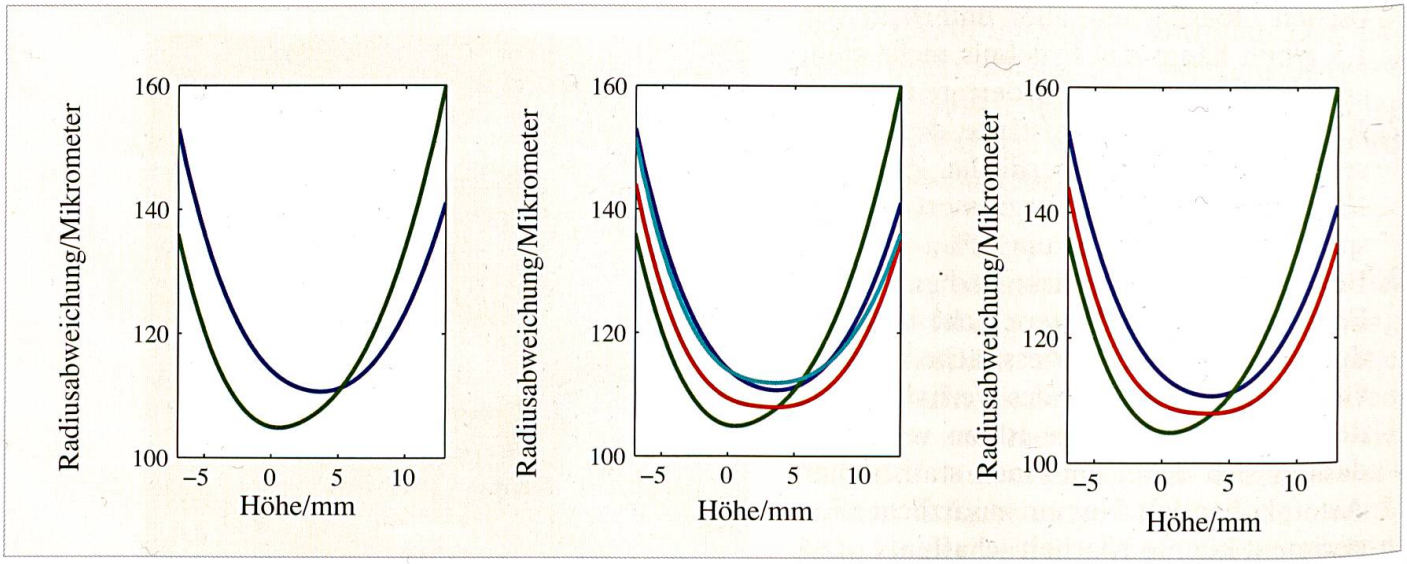

Abbildung 22:
Die drei Diagramme zeigen, dass die schrittweise Korrektur mit Hilfe eine Prozessmodells funktioniert. Bereits nach zwei Schritten ist die Störung kompensiert. Die Idealkontur ist dunkelblau.

Ausgangpunkt ist eine Sollkontur, die als Abweichung von einer Referenzfläche definiert ist. Auf Grund einer Störung liefert die gleiche Maschineneinstellung ein verändertes Produkt (Abbildung 22a). Das Prozessmodell liefert einen korrigierten Satz von Einflussgrössen, der gemäss aktuellem Prozessmodell voraussichtlich die Störung kompensiert. Die Durchführung des Versuchs zeigt (Abbildung 22, rote Kurve), dass die Störungskompensation noch nicht vollständig erfolgt ist. Nach einem weiteren Schritt ist das Ergebnis zufrieden stellend (Abbildung 22c, hellblaue Kurve). Verglichen mit einer manuellen Prozessoptimierung ist das modellgestützte Verfahren sehr effizient. Bei 4 freien Parameter ist ein intuitives Vorgehen fast aussichtslos. Die Methode, jeweils nur einen einzigen Parameter zu verändern, erfordert gegenüber dem hier vorgestellten Verfahren ein Mehrfaches an Versuchen.

Literaturverzeichnis

BUNDESAMT FÜR GESUNDHEIT (2006): http://www.bag.admin.ch/themen/chemikalien/00228/00510/index.html?lang=de

HAYKIN, S. (1998): Neural Networks: A Comprehensive Foundation. – Prentice Hall.

LEPAROUX, M., LOHER, M., SCHREUDERS, C., SIEGMANN, S. (2007): Thermal Plasma Production of Si-Nanoparticles: Process Optimization with Neural Networks. Zur Publikation eingereicht beim Journal of Powder Technology.

MATLAB® (2005): Neural Network Toolbox. – The MathWorks, Inc.

MCCARTHY, J., MINSKY, M. L., ROCHESTER, N., SHANNON, C.E. (1955): A proposal for the Dartmouth summer research project on artificial intelligence. http://www-formal.stanford.edu/jmc/history/ dartmouth/dartmouth.html

MCCULLOCH, W.S., PITTS, W. (1943): A logical calculus of the ideas immanent in nervous activity. – Bulletin of Mathematical Biophysics, vol.5, pp.115–133.

MOORE, G. E. (1965): Cramming More Components on Integrated Circuits. – Electronics (38)8, pp. 114–117.

PASCHEN, H., COENEN, C., FLEISCHER, T. (2004): Nanotechnologie – Forschung, Entwicklung, Anwendung. – Springer-Verlag; ISBN: 3-540-21068-7.

PURDIE, N., LUCAS, E.A., TALLEY, M.B. (1992): Direct measure of total cholesterol and its distribution among major serum lipoproteins. - Clinical Chemistry, vol. 38, no. 9, pp. 1645–1646.

RUMELHART, D.E., HINTON, G.E., WILLIAMS, R.J. (1986): Learning internal representations by error propagation. – Parallel Distributed Processing: Exploratations in the Microstructure of Cognition, Vol.1, Cambridge, MA: MIT Press.