In Georg Dorffner, Knut Möller, Gerhard Paaß, Raúl Rojas und Stephan Vogel, Hrsg., *Konnektionismus und Neuronale Netze: Beiträge zur Herbstschule HeKoNN96, Münster/Westf.*, S. 41–52, St. Augustin, 1996, GMD-Studien Nr. 300, GMD-Forschungszentrum Informationstechnik GmbH.

Einführung in die Komplexität des Rechnens und Lernens mit neuronalen Netzen

Michael Schmitt

Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz
Klosterwiesgasse 32/2
A-8010 Graz, Austria
mschmitt@igi.tu-graz.ac.at

> Suppose you want to teach the "cat" concept to a very young child. Do you explain that a cat is a relatively small, primarily carnivorous mammal with retractile claws, a distinctive sonic output, etc.? I'll bet not. You probably show the kid a lot of different cats, saying "kitty" each time, until it gets the idea.
> —R. P. Boas, Can we make Mathematics intelligible?, *American Mathematical Monthly*, 88, 1981, 727–731.

1 Einführung

Der rasante Aufschwung, den neuronale Netze im vergangenen Jahrzehnt erfahren haben, hat auch an der Grundlagenforschung nicht haltgemacht. Theoretische Untersuchungen haben vielmehr eine unüberschaubare Zahl von Resultaten hervorgebracht, die es jemandem, der sich nur am Rande damit beschäftigt, schwer machen, sich einen Überblick zu verschaffen, geschweige denn, etwas Substantielles daraus zu gewinnen. Zunächst unverbindlich Interessierte sehen sich sehr bald mit einem weiteren Problem konfrontiert. Genügt in der Praxis meist eine erfolgreiche Simulation oder Demonstration eines Produktes, um potentielle Anwender von dessen Vorteilen zu überzeugen, so setzt die Theorie Maßstäbe, denen man nur mit Papier-und-Bleistift Methoden gerecht werden kann. Und darin liegt die Schwierigkeit: Theoretische Aussagen über Algorithmen sind meist so vereinfachend und allgemeingültig, daß die oder der einzelne nicht erkennt, was sie mit dem konkreten Problem zu tun haben, das man mit ihnen lösen möchte. Diese Distanz zu überwinden erfordert oft einen zu großen Schatz an Hintergrundwissen, weshalb viele davor zurückschrecken, sich näher mit theoretischen Betrachtungen zu befassen.

Diese kleine Einführung soll einige wichtige Ergebnisse aus der Theorie des Rechnens und Lernens mit neuronalen Netzen beleuchten. Daß die Auswahl subjektiv ist, versteht sich von selbst. Es kann sich dabei auch nur um die Spitzen eines oder mehrerer Eisberge handeln, deren Ausmaße kaum zu erahnen sind. Dies hat zur Folge, daß viele Aussagen nur in ihrer einfachsten Form dargestellt und oftmals nur anhand eines Beispiels veranschaulicht werden können. Aufgrund des eingangs Gesagten dürfte dem Leser klar sein, daß eine tiefergehende Betrachtung nicht möglich ist. Dem Autor zumindest ist klar, daß jener sie vielleicht gar nicht wünscht. Trotzdem hofft er, dem Leser Appetit zu machen, sich mit so manchem Gegenstand näher zu befassen oder sogar anhand der angegebenen Literatur an dem einen oder anderen Beweis festzubeißen. Der Zweck dieser bewußt einfach gehaltenen Darstellung nicht ganz so einfacher Komplexitätsbetrachtungen über neuronale Netzen wäre für den Autor erfüllt, wenn sie dazu anregen könnte, ähnliche Überlegungen anzustellen oder sich zumindest mit der Denkweise anfreunden zu wollen.

Dieser Beitrag ist wie folgt gegliedert: Zunächst betrachten wir Neuronen als Rechenbausteine in Schaltkreisen und untersuchen, welche Arten von Funktionen man mit ihnen berechnen kann. Mit dem Problem, solche Funktionen zu lernen, befassen wir uns im dritten und vierten Teil. Ausgehend von der Perceptron-Lernregel betrachten wir Verfahren zur Berechnung von Gewichten für Neuronen und versuchen, Aussagen über ihre Komplexität zu machen. Der vierte Teil schließlich steht ganz im Zeichen des PAC-Lernens, dem wohl populärsten Modell der algorithmischen Lerntheorie. Schwerpunkte legen wir dabei auf den Berechnungsaufwand beim Lernen, was uns zur algorithmischen Komplexität des sogenannten Konsistenzproblems führt, und auf die Frage nach der zum Lernen erforderlichen Anzahl der Beispiele, die sich in der VC-Dimension ausdrückt.

Am Ende befindet sich eine kleine Literatursammlung, auf die wir die interessierten Leserinnen und Leser an den entsprechenden Stellen verweisen. Wir vermeiden es dabei möglichst, Originalarbeiten zu zitieren und geben statt dessen leichter zugängliche Literatur eher einführenden Charakters an, wo das jeweilge Thema auch kommentiert und in einen größeren Zusammenhang gestellt wird. Wer trotzdem bis zu den Anfängen zurückgehen möchte sei ermuntert, dies zu tun. Er findet die entsprechenden Quellen dann in den angegebenen Werken.

2 Rechnen mit künstlichen Neuronen

2.1 Ein formales Modell für biologische Neuronen

Zur Beschreibung des logischen Verhaltens eines biologischen Neurons stellten McCulloch und Pitts im Jahre 1943 ihr formales Modell vor, das als McCulloch-Pitts Neuron bekannt wurde, heute jedoch häufig Schwellenneuron, *threshold gate* oder auch Perceptron genannt wird. Ein Neuron mit n Eingängen ist charakterisiert durch $n+1$ reelle Zahlen $w_1, \ldots, w_n, t$, wobei $w_1, \ldots, w_n$ die Gewichte an den Eingängen und t die Schwelle am Ausgang darstellen. Ein Neuron „feuert“ d.h. gibt eine 1 aus genau dann, wenn die gewichtete Summe der Eingänge die Schwelle erreicht, ansonsten gibt es eine 0 aus. Für die durch das Neuron berechnete Funktion f gilt also

$$\begin{aligned} f(x_1, \ldots, x_n) &= 1, \quad \text{falls} \quad w_1x_1 + \cdots + w_nx_n \geq t \\ f(x_1, \ldots, x_n) &= 0, \quad \text{falls} \quad w_1x_1 + \cdots + w_nx_n < t. \end{aligned}$$

Die Definition macht Sinn wenn die Eingänge beliebige reelle Zahlen annehmen, ursprünglich hat man sich jedoch auf die Werte 0 und 1 beschränkt, da nur das Feuern oder Nicht-Feuern eines Eingangs repäsentiert werden sollte. In diesem Fall spricht man auch von Booleschen Schwellenneuronen. Ein n-stelliges Boolesches Schwellenneuron berechnet also eine Funktion $f : \{0,1\}^n \to \{0,1\}$. Eine Funktion, die durch ein Schwellenneuron berechnet werden kann, heißt auch *linear trennbar*. Ebenfalls bezeichnet man zwei Mengen A und B als linear trennbar, wenn eine linear trennbare Funktion f existiert, so daß $f(A) \subseteq \{1\}$ und $f(B) \subseteq \{0\}$ gilt. Mehr zur Definition und Motivation von McCulloch und Pitts findet man in dem Buch von Arbib [2].

2.2 Über die Größe der Gewichte

Da es unendlich viele Werte für die Gewichte gibt, ist die Menge der durch Schwellenneuronen repräsentierbaren Funktionen ebenfalls unendlich. Dies gilt natürlich nur, wenn auch die Eingänge unendlich viele Werte annehmen können. Wenn diese sich nun auf eine endliche Menge beschränken, wieviele verschiedene Werte braucht man dann für die Gewichte? Oder anders gefragt: Wenn die Gewichte nur mit einer beschränkten Anzahl von Bits dargestellt werden können, geht dann etwas von der Rechenleistung der Neuronen verloren? Wir gehen dieser Frage für den Fall der binären Eingänge nach.

Satz. Für jedes Gewicht einer linear trennbaren Booleschen Funktion genügen $n \log n$ Bits.

Diese Eigenschaft wurde im Jahre 1961 von Muroga, Toda und Takasu bewiesen. Lange Zeit war unbekannt, ob man auch wirklich so viele Bits braucht. Es ist nicht schwierig, eine Funktion zu konstruieren, die ein ganzzahliges Gewicht vom Wert mindestens $2^{n/2}$ benötigt. Muroga entdeckte im Jahre 1965 sogar eine Funktion mit mindestens n Bits für ein Gewicht. Im Jahr 1992 endlich konstruierte Håstad eine Funktion, die asymptotisch annähernd $n \log n$ Bits benötigt. Ausführliche Beweise dieser Aussagen enthält Kapitel 4 von Parberry [10] sowie Kapitel 6 von Siu *et al.* [13]. In Kapitel 2 von [12] findet man detailliertere Untersuchungen zur Größe der Gewichte.

2.3 Eine arithmetische und eine geometrische Interpretation

Notwendige und hinreichende Bedingungen dafür, daß eine Funktion linear trennbar ist, wurden in den sechziger Jahren ausgiebig in einem Forschungszweig untersucht, der sich *Threshold Logic* nennt. Alle wichigen Ergebnisse zu diesem Thema enthält das Buch von Muroga aus dem Jahre 1971 [8]. Eine arithmetische Beschreibung der linearen Trennbarkeit ist die sogenannte Asummierbarkeit. Eine Funktion f, definiert auf dem n-dimensionalen Raum, mit Funktionswerten 0 und 1 heißt *summierbar*, wenn es eine Zahl $k \geq 2$ gibt, so daß die Menge $\{x : f(x) = 1\}$ Punkte $x^{(1)}, \ldots, x^{(k)}$ und die Menge $\{y : f(y) = 0\}$ Punkte $y^{(1)}, \ldots, y^{(k)}$ enthält mit

$$\sum_{i=1}^{k} x^{(i)} = \sum_{i=1}^{k} y^{(i)}.$$

Die Punkte müssen dabei nicht notwendigerweise verschieden sein. Eine nicht summierbare Funktion heißt *asummierbar*.

Satz. Eine Funktion ist linear trennbar genau dann, wenn sie asummierbar ist.

Ein Gewichtsvektor $(w_1, \ldots, w_n, t)$ repräsentiert eine $(n-1)$-dimensionale Hyperebene im n-dimensionalen Raum, nämlich die Menge

$$\{(x_1, \ldots, x_n) : w_1 x_1 + \cdots + w_n x_n = t\}.$$

Die Vektoren mit Ausgabe 1 bzw. 0 sind somit Punkte, die auf verschiedenen Seiten der Hyperebene liegen, wobei die Ausgabe 1 auch Punkte auf der Hyperebene einschließt. Eine duale Betrachtungsweise faßt nicht den Gewichtsvektor, sondern die Eingabevektoren als Hyperebenen auf. Die lineare Trennbarkeit ist dabei äquivalent zur Lösbarkeit eines linearen Unleichungssystems. (Hierzu muß man die Eingabevektoren um eine Komponente mit dem Wert -1 erweitern.) Durch Multiplikation der Vektoren mit Ausgabe 0 mit -1 und Addition eines hinreichend kleinen Abstandes ε erhält man ein Ungleichungssystem in der folgenden Form:

$$\begin{array}{ll} w_1x_1 + \cdots + w_nx_n - t \geq 0 & \text{falls } f(x_1,\ldots,x_n) = 1 \\ w_1x_1 + \cdots + w_nx_n + t \geq \varepsilon & \text{falls } f(x_1,\ldots,x_n) = 0 \end{array} \tag{1}$$

Dieses Ungleichungssystem definiert die geometrische Figur eines Polyeders.

Satz. Eine Funktion f ist linear trennbar genau dann, wenn der durch (1) definierte Polyeder nicht leer ist.

2.4 Über die Anzahl linearer Zerlegungen

Betrachten wir eine Menge C von n-stelligen Vektoren, wieviele lineare Zerlegungen dieser Menge gibt es, d.h., wieviele verschiedene Paare $A, B \subseteq C$ gibt es, so daß $A \cup B = C$ und A und B linear trennbar sind? Allgemeiner fragen wir, wie groß diese Zahl werden kann, wenn C variiert und nur die Anzahl der Elemente von C durch, sagen wir, m beschränkt ist. Diese Zahl $L(m,n)$ sagt etwas aus über die Rechenleistung von McCulloch-Pitts Neuronen. Nach dem im vorangehenden Abschnitt Gesagten ist sie identisch mit der maximalen Anzahl der Zerlegungen von m Punkten im n-dimensionalen Raum durch eine $(n-1)$-dimensionale Hyperebene, und sie gibt die Anzahl von Teilen an, die bei der Zerlegung des n-dimensionalen Raumes durch m Hyperebenen höchstens entstehen können. Wir sagen, eine Menge von m Punkten im n-dimensionalen Raum befindet sich in allgemeiner Lage, falls $m > n$ und keine $(n+1)$-elementige Teilmenge in einer $(n-1)$-dimensionalen Hyperebene enthalten ist, oder falls $m \leq n$ ist und keine $(m-2)$-dimensionale Hyperebene alle Punkte enthält.

Satz. Eine Menge von m Punkten im n-dimensionalen Raum, die sich in allgemeiner Lage befindet, kann auf $L(m,n)$ verschiedene Arten linear zerlegt werden, wobei

$$L(m,n) = \begin{cases} 2\sum_{i=0}^{n} \binom{m-1}{i} & \text{falls } m > n \\ 2^m & \text{falls } m \leq n. \end{cases}$$

Setzt man $m = 2^n$ so erhält man mit 2^{n^2} eine gute Abschätzung für die maximale Anzahl linear trennbarer Boolescher Funktionen. Weiter zeigt die Formel, daß eine Menge in allgemeiner Lage mit höchstens $n+1$ Elementen auf alle möglichen Arten zerlegt werden kann. Man sagt auch, die Menge wird zersplittert. Diese Eigenschaft wird uns im Zusammenhang mit der erforderlichen Anzahl von Beispielen beim PAC-Lernen wieder beschäftigen. Dazu mehr im vierten Teil. Mehr über lineare Zerlegungen findet man außer in dem schon erwähnten Buch von Muroga [8] bei Nilsson [9].

2.5 Eine nicht berechenbare Funktion

Im vorangehenden Abschnitt haben wir gesehen, daß nicht alle Booleschen Funktionen linear trennbar sind, sogar nur ein relativ geringer Teil, vergleicht man 2^{n^2} mit 2^{2^n}. Wie sieht nun eine nicht berechenbare Funktion aus? Das berühmteste Beispiel ist die Paritätsfunktion. Sie ist definiert durch

$$f(x_1,\ldots,x_n) = 1 \iff x_1 + \cdots + x_n \text{ ungerade.}$$

Auch die Negation von f, die genau dann 1 ist, wenn $x_1 + \cdots + x_n$ gerade ist, wird als Paritätsfunktion bezeichnet. In der Tat sind die Paritätsfuntionen die „kleinsten“ Funktionen, die nicht linear trennbar sind. Denn für $n = 2$ gibt es 16 Boolesche Funktionen, wovon genau 14 linear trennbar sind. Mit wachsendem n wird die Lücke dann jedoch sehr schnell größer.

Satz. Für $n \geq 2$ ist die Paritätsfunktion nicht durch ein McCulloch-Pitts Neuron berechenbar.

Die Paritätsfunktion ist eine symmetrische Funktion. Das bedeutet, daß ihr Funktionswert nur von der Summe der Eingänge abhängt. Es folgt aus dem von Minsky und Papert bewiesenen sogenannten Gruppeninvarianztheorem, daß die einzigen symmetrischen Funktionen, die linear trennbar sind, die Form

$$f(x_1, \ldots, x_n) = 1 \quad \Longleftrightarrow \quad x_1 + \cdots + x_n \geq k$$

haben, oder die Negation davon, wobei k fest ist. Minsky und Papert haben noch viele weitere Probleme dieser Art studiert. In der Tat bezieht sich das Gruppeninvarianztheorem auf Perceptrons, worunter die Autoren McCulloch-Pitts Neuronen verstehen, denen eine Schicht von Booleschen Schaltelementen vorgelagert ist. Das Buch von Minsky und Papert, zuerst 1969 erschienen, ist immer noch eine reichhaltige Fundgrube für Fragestellungen dieser Art [7].

2.6 Schaltkreise von Schwellenneuronen

Das wiedererwachte Interesse an neuronalen Netzen in den achtziger Jahren nach dem Abflauen gegen Ende der sechziger ist nicht zuletzt der Entwicklung von Lernverfahren für mehrschichtige Netzarchitekturen zuzuschreiben. Schaltkreise von Neuronen können wesentlich mehr Funktionen berechnen als einzelne Schwellenneuronen, und so eröffnet sich die Möglichkeit, bessere Lösungen mit Netzen finden zu können als mit einzelnen Neuronen. In der Schaltkreistheorie untersucht man unter anderem, was hinter dieser Behauptung steckt. Der Fortschritt zeigt sich insbesondere bei den symmetrischen Funktionen.

Satz. Jede symmetrische Boolesche Funktion kann mit Schaltkreisen von Schwellenneuronen der Größe $2n + 3$ und Tiefe 2 berechnet werden, wobei die Gewichte nur die Werte -1 und 1 annehmen.

Schaltkreise der Tiefe 2 können also besonders effizient zählen, auch wenn sie nur $\{-1, 1\}$-Gewichte verwenden dürfen. Von besonderem Interesse sind Schaltkreise, deren Gewichte polynomiell beschränkt sind, d.h., die dem Betrag nach nicht größer sind als ein vorgegebenes Polynom. Die Boolesche Funktion IP („Inneres Produkt“) spielt eine wichtige Rolle bei Komplexitätsuntersuchungen. Sie ist definiert durch

$$\mathrm{IP}(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1 \quad \Longleftrightarrow \quad x_1 y_1 + \cdots + x_n y_n \text{ ungerade.}$$

Satz. Die Funktion IP kann durch Schaltkreise von Schwellenneuronen mit Tiefe 3 berechnet werden, nicht jedoch mit Tiefe 2, wenn die Zahl der Neuronen und der Betrag der Gewichte jeweils durch ein Polynom beschränkt sind.

Die Frage, ob sich auch größere Tiefen in ihrer Rechenleistung unterscheiden, ist bislang noch nicht geklärt. Es ist bekannt, daß die Multiplikation zweier n-Bit Zahlen in Tiefe 3 mit polynomieller Größe und polynomiellen Gewichten durchführbar ist. Weitere Resultate zur Komplexität von Schwellenneuronen-Schaltkreisen und deren Beziehungen zu anderen Schaltkreis-Klassen enthalten Kapitel 7 von Parberrys Buch [10], die Beiträge von Hofmeister und Maass im Buch von Roychowdhury *et al.* [11, Kapitel 2 bzw. 5] und das Buch von Siu *at al.* [13]. Heute übliche Lernverfahren für mehrschichtige Architekturen verwenden sigmoide Neuronen, das sind Neuronen, deren Ausgabewert durch Anwendung der sigmoiden Funktion

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

auf die Differenz zwischen gewichteter Eingabesumme und Schwelle berechnet wird. Solche Approximationen von Schwellenfunktionen sind differenzierbar, eine notwendige Voraussetzung für das Back-Propagation Lernverfahren. Eine Untersuchung der Leistungsfähigkeit von Schaltkreisen mit sigmoiden Neuronen findet man in dem Beitrag von Maass *et al.* zum Buch von Roychowdhury *et al.* [11, Kapitel 4].

3 Die Zeitkomplexität des Lernens von Gewichten

3.1 Lernen aus Fehlern

Die Perceptron-Lernregel, entwickelt von Rosenblatt gegen Ende der fünfziger Jahre, hat bis heute nichts von ihrer Attraktivität verloren. War es ursprünglich das Perceptron-Konvergenztheorem, also der Nachweis, daß eine beliebige lineare Zerlegung in endlich vielen Schritten gelernt werden konnte[1], so fasziniert jetzt immer noch die Leichtigkeit und Einfachheit, mit der sich diese Regel implementieren läßt. Rosenblatts Perceptrons sind zweischichtige Netze aus Schwellenneuronen, die Lernregel modifiziert jedoch nur das Neuron in der zweiten Schicht, weshalb man die erste außer acht lassen kann. Der Perceptron-Algorithmus ist ein Beispiel für ein *on-line* Lernverfahren, d.h., es wird in jedem Schritt nur ein Beispiel der zu lernenden Funktion präsentiert. Macht das Neuron einen Fehler, dann wird dieses Beispiel zum Gewichtsvektor addiert, wenn dieser es als negativ klassifiziert, ansonsten wird es subtrahiert. Korrekt klassifizierte Beispiele bewirken keine Änderung. Die Güte einer solchen Fehlerkorrekturregel beurteilt man nach der maximalen Anzahl möglicher Fehler.

Satz. Die Perceptron-Lernregel macht beim Lernen einer linearen Zerlegung Boolescher Vektoren höchstens $(n+1)^2 \cdot 2^{(n+1)\log(n+1)}$ Fehler.

Dies ist die Formulierung des Perceptron-Konvergenztheorems im Beitrag von Maass zu Roychowdhury *et al.* [11, Kapitel 10]. Sie setzt voraus, daß die Beispielvektoren binär sind und als Initialisierung für den Gewichtsvektor der Nullvektor gewählt wird. Man kann jedoch auch reelle Beispielvektoren zulassen, mit einem beliebigen Gewichtsvektor starten oder die Regel so abändern, daß man immer ein konstantes Vielfaches des Beispielvektors addiert bzw. subtrahiert. Alle diese Modifikationen ändern nichts an der Tatsache der Konvergenz, lediglich die obere Schranke für die Anzahl der Fehler muß korrigiert werden. Bemerkenswert an dieser Schranke ist, daß sie exponentiell in n ist. Ist dies wirklich notwendig? Aus der Tatsache, daß sich pro Anwendung der Lernregel jedes Gewicht nur um

[1] Im Jahre 1962 bezeichnete Novikoff das Perceptron-Konvergenztheorem als "one of the most often re-proved results in the field".

einen konstanten Betrag ändert, und aufgrund der Existenz von linear trennbaren Funktionen mit exponentiellen Gewichten kann man leicht eine exponentielle untere Schranke für die Anzahl der Lernschritte herleiten. Mehr über den Perceptron-Algorithmus enthalten Kapitel 11 von Minsky und Papert [7], Kapitel 10 von Parberry [10] und Kapitel 2 von Siu *et al.* [13]. Weitere Kriterien für die Identifikation von linear trennbaren Funktionen mit einfachen Lernregeln werden in Kapitel 3 von [12] untersucht.

3.2 Geometrische Verfahren zur Berechnung von Gewichtsvektoren

Ein *on-line* Algorithmus hat es schwer, da ihm zu jedem Zeitpunkt nur ein Beispiel gezeigt wird, das er dann so verarbeiten muß, daß das bisher Gelernte beibehalten und die Zahl zukünftiger Fehler verringert wird. Daher verwundert es einerseits, daß die vom Perceptron-Algorithmus verwendeten primitiven Operationen (Addition und Subtraktion von Konstanten) das Lernen überhaupt in endlicher Zeit durchführbar machen, andererseits überrascht es nicht, daß man dabei eine exponentielle Anzahl von Fehlern in Kauf nehmen muß. Es gibt jedoch *on-line* Verfahren, die nur eine polynomiell beschränkte Anzahl von Fehlern machen, allerdings verwenden sie, wie zu erwarten, wesentlich aufwendigere Grundoperationen. Sie basieren auf Methoden zur Lösung von linearen Ungleichungssystemen.

Satz. Boolesche linear trennbare Funktionen können mit höchstens $O(n^2 \log n)$ Fehlern gelernt werden.

In der Tat kann jeder Algorithmus zum Lösen von linearen Ungleichungssystemen zur Berechnung von Gewichten aus Beispielen verwendet werden; wie in Abschnitt 2.3 erwähnt ist die Existenz eines Gewichtsvektors äquivalent zur Lösbarkeit des Systems (1). Jedoch lassen sich nicht alle Lösungsmethoden als *on-line* Verfahren im Stil des Perceptron-Algorithmus implementieren. Lernalgorithmen, die verlangen, daß alle Beispiele auf einmal präsentiert werden, bezeichnet man als *batch* Verfahren. Mit der allgemeinen Aufgabe, Lösungen von linearen Ungleichungssystemen zu finden, befaßt sich das Gebiet der Linearen Programmierung. Einen Überblick über Lösungsmethoden gibt das Buch von Karloff [5]. Mehr zu *on-line* Implementierungen solcher Verfahren enthält der Beitrag von Maass und Turán im Buch von Hanson *et al.* [3, Kapitel 13].

3.3 Gewichtsberechnung bei Netzen

Bis jetzt haben wir Algorithmen zur Berechnung von Gewichten bei einzelnen Neuronen betrachtet und konnten die Frage, ob es polynomiell zeitbeschränkte Lernalgorithmen gibt, mit ja beantworten. Wie sieht dies aber nun bei „echten“ Netzen aus? Um die Komplexität eines Berechnungsproblems festzustellen, betrachtet man in der Komplexitätstheorie zunächst einmal das zugehörige Entscheidungsproblem, also die Variante, bei der man nur nach der Existenz von Gewichten fragt, und nicht deren Berechnung verlangt. Ist das Entscheidungsproblem schwierig, dann ist auch das Berechnungsproblem nicht einfacher zu lösen; umgekehrt ergibt sich aus einem Beweis für die effiziente Lösbarkeit eines Problems in vielen Fällen auch schon ein Algorithmus zur Berechnung der Lösungen. Die einem Lernproblem entsprechende Entscheidungs-Variante ist das sogenannte *Konsistenzproblem*, in der englischsprachigen Literatur auch *loading problem* genannt. Es lautet für eine fest vorgegebene Architektur $\mathcal{A}$:

Gegeben eine Beispielmenge, gibt es Gewichte für $\mathcal{A}$, so daß die durch $\mathcal{A}$ mit diesen Gewichten berechnete Funktion alle Beispiele korrekt klassifiziert?

Ein berühmtes Resultat von Blum und Rivest besagt, daß dieses Problem für ein zweischichtiges Netz mit drei Neuronen nicht effizient lösbar sein kann, wenn die Komplexitätsklassen P und NP die (noch unbewiesene) Eigenschaft P$\neq$NP erfüllen.

Satz. Das Konsistenzproblem für die Architektur mit 3 Booleschen Schwellenneuronen (2 verborgene Neuronen, 1 Ausgabeneuron) ist NP-vollständig.

Einen Beweis für ein dem Theorem von Blum und Rivest verwandtes Resultat findet man bei Anthony und Biggs [1, Kapitel 10]. Eine andere Form des Konsistenzproblems schreibt die Architektur nicht fest vor sondern läßt sie als Eingabevariable für den Algorithmus zu. Diese Variante wurde für verschiedene Architekturklassen von Judd untersucht [4]. Er formuliert auch hinreichende Bedingungen an Architekturen, die das Konsistenzproblem polynomiell lösbar werden lassen.

Von größerem Interesse im Zusammenhang mit dem Back-Propagation Lernverfahren sind natürlich Netze aus Neuronen mit reellwertiger Aktivierungsfunktion. Für die gebräuchlichen sigmoiden Aktivierungsfunktionen gibt es allerdings noch keine Resultate. Einen NP-Vollständigkeitsbeweis für Netze mit stückweise linearer Aktivierungsfunktion enthält der Beitrag von DasGupta *et al.* im Buch von Roychowdhury *et al.* [11, Kapitel 10].

Das Konistenzproblem ist ein Entscheidungsproblem, das zugeschnitten ist auf den Fall, daß man die Klasse der Funktionen genau kennt, die man lernen möchte. Dies trifft aber in der Praxis in den wenigsten Fällen zu. Man nimmt vielmehr in Kauf, daß das Netz auch nach dem Lernen noch Fehler macht, versucht aber, diese Fehlerzahl möglichst gering werden zu lassen. Das Entscheidungsproblem, das diese Aufgabe widerspiegelt, ist das sogenannte *Fehlerminimierungsproblem*, englisch *minimizing disagreement problem*: Gegeben eine Beispielmenge und eine natürliche Zahl k, gibt es Gewichte für die Architektur $\mathcal{A}$, so daß die durch $\mathcal{A}$ mit diesen Gewichten berechnete Funktion auf der Beispielmenge höchstens k Fehler macht? Hoeffgen *et al.* haben gezeigt, daß dieses Problem für einzelne Schwellenneuronen NP-vollständig ist. Eine Erwähnung dieses Resultats findet man bei Maass im Beitrag zu Roychowdhury *et al.* [11, Kapitel 8].

4 Lernen im PAC-Modell

4.1 Eine Definition der Generalisierung

Eine Möglichkeit, den Begriff Lernen zu definieren, ist, einen Lehrer vorauszusetzen, der dem Algorithmus Gegenbeispiele liefert, damit dieser lernt, in Zukunft weniger Fehler zu machen. Dies setzt natürlich die Existenz eines solchen Lehrers voraus, eine etwas unrealistische Annahme, da vom Lehrer erwartet wird, daß er immer den augenblicklichen Zustand des Lernalgorithmus kennt und ein Gegenbeispiel konstruieren kann. Ein weiteres pessimistisches Merkmal dieses Modells besteht darin, daß es einen böswilligen Lehrer zuläßt, der dem Algorithmus nur solche Gegenbeispiele präsentiert, die sonst nie mehr vorkommen oder aus denen er nichts oder nur sehr wenig lernt. Ebenfalls unnötig streng ist die Forderung an den Algorithmus, die Fehlerzahl ständig zu verringern, anstatt nach einem bestimmten Zeitpunkt den Lernvorgang als abgeschlossen zu betrachten und eine zwar beschränkte, aber von Null verschiedene Fehlerzahl zuzulassen. Alle diese Forderungen sind gelockert im sogenannten PAC-Lernmodell von Valiant, das dieser im Jahre 1984 vorstellte. PAC steht dabei für *probably approximately correct*:

Gegeben eine Menge C, der sogenannten *Begriffsklasse*, von $\{0,1\}$-wertigen Funktionen über einem Grundbereich X, dann ist die Aufgabe des Lernalgorithmus A eine beliebige Zielfunktion $f \in C$ zu lernen. Dazu werden eine gewisse Zahl, sagen wir m, Elemente $x^{(1)}, \ldots, x^{(m)} \in X$ gemäß einer unbekannten, aber als fest angenommenen Wahrscheinlichkeitsverteilung P gezogen und dem Algorithmus zusammen mit der durch f bestimmten Klassifikation als Beispiele übergeben. Dieser ermittelt dann eine *Hypothese* $h \in C$, die dann gegen die Zielfunktion f getestet wird, indem man ein weiteres Beispiel x gemäß P zieht. Mit Wahrscheinlichkeit mindestens $1-\delta$, bezogen auf die Wahl der m Trainingsbeispiele, soll dann die Hypothese eine Fehlerwahrscheinlichkeit beim Testbeispiel x von höchstens ε haben. Eine wichtige Forderung an A ist die Verteilungsunabhängigkeit: Der Algorithmus darf die Wahrscheinlichkeitsverteilung P nicht kennen, d.h., er muß für jede Verteilung korrekt arbeiten.

Eine Begriffsklasse C hat die Eigenschaft, PAC-lernbar zu sein, wenn ein Algorithmus A existiert, der alle diese Bedingungen erfüllt, die wir formal so darstellen können:

$$\forall \varepsilon, \delta \;\; \exists m \;\; \forall P \;\; \forall f \in C \quad P^m[\;\; P[h(x) \neq f(x)] \leq \varepsilon \;\;] \geq 1 - \delta$$

Dabei ist h eine Abkürzung für die vom Algorithmus berechnete Hypothese

$$A(\langle x^{(1)}, f(x^{(1)})\rangle, \ldots, \langle x^{(m)}, f(x^{(m)})\rangle).$$

Die Verteilung P^m steht für das m-malige, unabhängige Ziehen von Elementen aus X mit Verteilung P. Die Definition enthält noch keine Bedingung an die Komplexität. Wir haben nur verlangt, daß die Anzahl der Beispiele m und die Anzahl der Rechenschritte von A endlich ist. Wie m von C, ε und δ abhängen müssen, damit eine Klasse C lernbar ist, untersucht man mittels der Beispielkomplexität des PAC-Lernens, auf die wir im folgenden Abschnitt näher eingehen. Danach betrachten wir Bedingungen an die Existenz effizienter PAC-Lernalgorithmen. Eine Einführung in die Theorie des PAC-Lernens gibt das Buch von Anthony und Biggs [1]. Fortgeschrittene Konzepte behandeln Kearns und Vazirani [6] sowie Turán im Buch von Roychowdhury *et al.* [11, Kapitel 7].

4.2 Die Anzahl der Beispiele beim PAC-Lernen

Wieviele Beispiele muß man wählen, damit ein Algorithmus ein PAC-Lernalgorithmus für eine Begriffsklasse C ist? Es zeigt sich, daß für endliche Klassen eine Anzahl Beispiele genügt, die proportional zum Logarithmus der Zahl der Elemente von C ist.

Satz. Jede endliche Begriffsklasse C ist mit $m = \left\lceil \frac{1}{\varepsilon} \ln \frac{|C|}{\delta} \right\rceil$ Beispielen PAC-lernbar.

Wir haben dieses Resultat nur in seiner einfachsten Form wiedergegeben. Ein konsistenter Lernalgorithmus für C hat die Eigenschaft, bei Eingabe einer Beispielmenge eine Hypothese aus C zu liefern, die mit dieser Beispielmenge konsistent ist, d.h. sie korrekt klassifiziert, vorausgesetzt, es existiert überhaupt eine solche Hypothese. In der Tat ist jeder konsistente Lernalgorithmus für C mit $(1/\varepsilon)\ln(|C|/\delta)$ Beispielen ein PAC-Lernalgorithmus. Für endliche Klassen genügt also insbesondere eine Zahl von Beispielen, die durch ein Polynom in $|C|$, $1/\varepsilon$ und $1/\delta$ beschränkt ist. Ein zentrales Resultat der algorithmischen

Lerntheorie besagt nun, daß unter bestimmten Voraussetzungen auch unendliche Klassen PAC-lernbar sind. Ist C eine Begriffsklasse über dem Grundbereich X, dann ist die *Vapnik-Chervonenkis-Dimension* von C (kurz VC-dim(C), benannt nach den russischen Mathematikern V. Vapnik und A. Chervonenkis) die Kardinalität der größten Menge $S \subseteq X$, so daß C alle Funktionen $f : S \to \{0, 1\}$ enthält. Man sagt auch, S wird zersplittert durch C. Damit läßt sich nun eine hinreichende Anzahl von Beispielen angeben, so daß auch unendliche Klassen, z.B. über den reellen Zahlen, gelernt werden können.

Satz. Sei C eine Begriffsklasse mit endlicher VC-Dimension d. Dann ist C mit

$$m \;=\; \left\lceil \frac{4}{\varepsilon}\left(d \log\left(\frac{12}{\varepsilon}\right) + \log\left(\frac{2}{\delta}\right)\right)\right\rceil$$

Beispielen PAC-lernbar.

Es genügt also eine Zahl von Beispielen, die linear in VC-dim(C) ist. Wie gut die VC-Dimension die Beispielkomplexität des PAC-Lernens charakterisiert, zeigt sich schließlich in einer bekannten unteren Schranke, wo sie sich ebenfalls als linearer Term wiederfindet.

Satz. Für jede Begriffsklasse C mit VC-Dimension d benötigt ein PAC-Lernalgorithmus mindestens $(d - 1)/(32\varepsilon)$ Beispiele.

Beweise für die obere und untere Schranke findet man bei Anthony und Biggs [1, Kapitel 8]. Was läßt sich nun über die VC-Dimension neuronaler Netze sagen? Aus dem Satz von Abschnitt 2.4 können wir folgern, daß die größte Menge von Punkten im n-dimensionalen Raum, die durch ein Schwellenneuron zersplittert werden kann, die Kardinalität $n + 1$ hat. Für Netze von Schwellenneuronen konnte man zeigen, daß sich die VC-Dimension durch $O(w \log w)$ beschränken läßt, wobei w die Zahl der Gewichte des Netzes ist. Netze der Tiefe mindestens 3 haben eine VC-Dimension von mindestens $\Omega(w \log w)$. Dieses Resultat gilt sowohl für Netze mit Schwellenneuronen als auch für Netze mit sigmoiden Neuronen. Für letztere schließlich konnte eine obere Schranke von $O(w^2)$ nachgewiesen werden. Einen Überblick über Ergebnisse zur VC-Dimension neuronaler Netze enthält der Beitrag von Maass im Buch von Roychowdhury *et al.* [11, Kapitel 8].

4.3 Zeitkomplexität des PAC-Lernens

Wir betrachten eine Folge C_n von Begriffsklassen über den Grundbereichen X_n (z.B. die Menge der n-stelligen Booleschen Vektoren oder der n-dimensionale Raum) und sagen, $C = \bigcup C_n$ ist polynomiell PAC-lernbar, wenn ein polynomiell zeitbeschränkter Algorithmus und ein Polynom $m(n, 1/\varepsilon, 1/\delta)$ existieren, so daß für alle n die Klasse C_n mit $m(n, 1/\varepsilon, 1/\delta)$ Beispielen durch A PAC-lernbar ist. Aufgrund der Resultate aus dem vorangehenden Abschnitt ist m polynomiell in $n, 1/\varepsilon, 1/\delta$ genau dann, wenn VC-dim(C_n) polynomiell in n ist. Was sind nun hinreichende und notwendige Bedingungen für die Existenz eines polynomiell zeitbeschränkten Algorithmus? Wir erinnern uns an die Definition des konsistenten Lernalgorithmus in Abschnitt 4.2.

Satz. Sei VC-dim(C_n) polynomiell in n und sei A ein polynomiell zeitbeschränkter konsistenter Lernalgorithmus für $C = \bigcup C_n$. Dann ist C polynomiell PAC-lernbar.

Die Menge RP ist die Menge aller Entscheidungsprobleme, die mit einem probabilistischen, polynomiell zeitbeschränkten Algorithmus der folgenden Art gelöst werden können:

Hat die Eingabe eine Lösung, dann akzeptiert sie der Algorithmus mit Wahrscheinlichkeit mindestens 1/2, Eingaben ohne Lösung akzeptiert er jedoch nie. Aus RP $\neq$ NP folgt P $\neq$ NP, ob auch die Umkehrung gilt, ist nicht bekannt.

Satz. Falls RP $\neq$ NP gilt und das Konsistenzproblem für C NP-vollständig ist, dann ist C nicht polynomiell PAC-lernbar.

Als Folgerung der beiden Sätze erhalten wir, daß einzelne Schwellenneuronen PAC-lernbar sind, das in Abschnitt 3.3 betrachtete 3-Neuronen-Netz jedoch nicht, falls RP $\neq$ NP ist. Weitere Anwendungen von Resultaten aus der Theorie des PAC-Lernens auf neuronale Netze findet man bei Anthony und Biggs [1, Kapitel 10] sowie im Beitrag von Turán zum Buch von Roychowdhury *et al.* [11, Kapitel 7]. Verfeinerte Betrachtungen des Konsistenzproblems für neuronale Architekturen enthält Kapitel 4 von [12].

4.4 Agnostisches PAC-Lernen

Zum Schluß verweisen wir noch auf eine Variante des PAC-Lernens, die für Fälle in der Praxis relevanter ist als das klassische Modell. Dieses ist ja nur anwendbar, wenn die Beispiele von einer Funktion aus einer vorgegebenen Begriffsklasse stammen. Wie schon in Abschnitt 3.3 diskutiert, ist dies eine unrealistische Annahme, da man in der Praxis in den meisten Fällen die Begriffsklasse nicht kennt, der die zu lernenden Funktionen angehören.

Dem *agnostischen* PAC-Lernen liegt ein Szenario zugrunde, bei dem man in einer vorgegebenen Hypothesenklasse H mit hoher Wahrscheinlichkeit eine Hypothese finden möchte, die von der bestmöglichen Hypothese in H nur um ε abweicht.

Man macht also keine Annahme darüber, welcher Begriffsklasse die Zielfunktion angehört, insbesondere erlaubt man auch „verrauschte" Beispiele, bei denen der Fall eintreten kann, daß sie nicht einmal durch eine Funktion repräsentiert werden können. Als zusätzliche Erweiterung gestattet man reellwertige Paare als Beispiele, d.h., die Beschränkung auf 0 und 1 als Ausgabewerte ist aufgehoben. Dies macht die häufig beim Back-Propagation Lernverfahren verwendeten Netze mit sigmoiden Aktivierungsfunktionen einer Analyse in diesem Modell zugänglich.

Die VC-Dimension und ihre Erweiterung auf reellwertige Funktionen, die sogenannte *Pseudo-Dimension*, spielen eine wichtige Rolle bei der Abschätzung der Anzahl der benötigten Beispiele beim agnostischen PAC-Lernen. Untersucht man die Berechnungskomplexität des agnostischen PAC-Lernens, dann kommt dem Fehlerminimierungsproblem die Bedeutung zu, die das Konsistenzproblem für das PAC-Lernen hat. Eine Definition des agnostischen PAC-Lernens findet man bei Maass im Buch von Roychowdhury *et al.* [11, Kapitel 8].

Literatur

[1] M. Anthony und N. Biggs. *Computational Learning Theory.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1992.

[2] M. A. Arbib. *Brains, Machines, and Mathematics.* Springer-Verlag, New York, second edition, 1987.

[3] S. J. Hanson, G. A. Drastal und R. L. Rivest (Hrsg.). *Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospects.* The MIT Press, Cambridge, 1994.

[4] J. S. Judd. *Neural Network Design and the Complexity of Learning.* The MIT Press, Cambridge, MA, 1990.

[5] H. Karloff. *Linear Programming.* Birkhäuser, Boston, 1991.

[6] M. J. Kearns und U. V. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, Cambridge, MA, 1994.

[7] M. L. Minsky und S. A. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, Cambridge, MA, expanded edition, 1988.

[8] S. Muroga. *Threshold Logic and Its Applications.* John Wiley & Sons, New York, 1971.

[9] N. J. Nilsson. *The Mathematical Foundations of Learning Machines.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[10] I. Parberry. *Circuit Complexity and Neural Networks.* Foundations of Computing Series. The MIT Press, Cambridge, MA, 1994.

[11] V. Roychowdhury, K.-Y. Siu und A. Orlitsky (Hrsg.). *Theoretical Advances in Neural Computation and Learning.* Kluwer Academic Publishers, Boston, 1994.

[12] M. Schmitt. *Komplexität neuronaler Lernprobleme.* Peter Lang Verlag, Frankfurt am Main, 1996.

[13] K.-Y. Siu, V. Roychowdhury und T. Kailath. *Discrete Neural Computation: A Theoretical Foundation.* Prentice Hall, Englewood Cliffs, NJ, 1995.