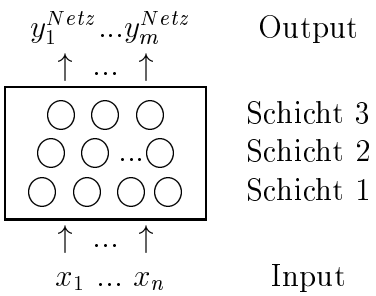
Kapitel 3

Feed–Forward Netze

Künstliche Neuronale Netze sind für viele Anwendungen in doppelter Hinsicht interessant. Einerseits werden sie als Werkzeuge zur Funktionsapproximation und zum Aufbau einer inneren Repräsentation der Außenwelt ausgenutzt. Andererseits dienen sie z. B. in der biologisch orientierten Roboterforschung als willkommene Medien zur Realisierung besonders biologienaher Roboter. Dabei sind die neuronalen Netze besonders wegen ihrer Lernfähigkeit attraktiv – sie können eine Aufgabe anhand von Beispielen erlernen statt aufwendig programmiert werden zu müssen. Zudem können sie direkt in Hardware realisiert werden (Neurocomputer), was zu extrem kurzen Verarbeitungszeiten führt.

3.1 Informationsverarbeitung mit Feed–Forward Netzen

Das einzelne Neuron kann nur eine pseudolineare Approximation einer beliebigen Funktion liefern. Will man komplizierte Funktionen gut approximieren, muß man mehrere Neuronen geeignet zusammenschalten. Feed-Forward Netze bestehen aus mehreren (sog. verborgenen) Schichten, die zwischen die Input– und Output–Einheiten geschaltet sind.



(3.1)

Die Neuronen einer Schicht erhalten dabei ihre Inputs von den Neuronen der jeweils davorgeschalteten Schicht, daher der Name feed–forward Netzwerk. I. a. Propagation der Information

besteht ein Netz aus aus M Schichten $s = 1, ..., M$ zu jeweils $n^{(s)}$ Neuronen. Das Netz in 3.1 besteht aus $M = 3$ Schichten, wobei die $n^{(3)} = 3$ **Output**neuronen die Schicht 3 bilden und die Schichten $1, 2$ verborgen sind. Die Terminologie ist hier nicht ganz eindeutig geregelt, manche Autoren bezeichnen auch die Schicht 1 als Inputschicht, so daß im Beispiel 3.1 nur die Schicht 2 verborgen wäre. Wir schließen uns aber der ersteren Sprechweise an, d. h. alle Neuronen, die keine Outputneuronen sind, bilden die verborgenen Schichten. Wir schreiben symbo-

Architektur Notation

lisch die Architektur des Netzes in der Form $n - n^{(1)} - n^{(2)} - ... - n^{(M)}$ (wobei $n^{(M)} = m$) für ein Netz aus M Schichten mit n Inputs und m Outputs. Das Netz in 3.1 wird (mit $m = 3$) folglich als $n - 4 - n^{(2)} - 3$ geschrieben.

In jeder Schicht s ergibt sich das postsynaptische Potential (PSP) $z_i^{(s)}$des Neurons i aus der Linearkombination der Inputs $\vec{y}^{(s-1)}$ in diese Schicht mit den Komponenten des Gewichtsvektors $\vec{w}_i^{(s)}$ des Neurons

$$z_i^{(s)} = \vec{w}_i^{(s)} \cdot \vec{y}^{(s-1)} = \sum_{k=1}^{n^{(s)}} w_{i,k}^{(s)} \, y_k^{(s-1)} \tag{3.2}$$

Der **Output** desNeurons i in Schicht s ist

$$y_i^{(s)} = S\left(z_i^{(s)}\right) = S\left(\vec{w}_i^{(s)} \cdot \vec{y}^{(s-1)}\right) \tag{3.3}$$

wobei mit der Vereinbarung $\vec{y}^{(0)} = \vec{x}$ und $\vec{y}^{(M)} = \vec{y}$ die In– und Outputs mit eingeschlossen sind.

Transferfunktion

Die Transferfunktion des gesamten Netzes ist folglich eine verschachtelte Funktion, die rekursiv aus 3.4 bei den Output–Einheiten beginnend aufgebaut werden kann. Ist $\vec{\mathbf{W}} = \left\{\vec{w}_i^{(s)}\right\}$ die Menge aller Gewichtsvektoren dann ist die Transferfunktion des Netzes

$$\boxed{\vec{y} = \vec{f}^{Netz}\left(\vec{x} \mid \vec{\mathbf{W}}\right)} \tag{3.4}$$

Die FFN sind universelle Funktionsapproximatoren wie in Abschnitt 3.5.1 näher dargestellt.

3.2 Der Backpropagation–Algorithmus

3.2.1 Die Update–Regel

Zur Bestimmung des optimalen $\vec{\mathbf{W}}$ konstruieren wir analog zu **??** eine Kostenfunktion

$$\begin{aligned} E\left(\vec{x} \mid \vec{\mathbf{W}}\right) &= \tfrac{1}{2}\left(\vec{y}^{Netz} - \vec{y}^{soll}\right)^2 = \tfrac{1}{2}\textstyle\sum_{i=1}^{M} \tfrac{1}{2}\left(y_i^{Netz} - y_i^{soll}\right)^2 \\ &= \tfrac{1}{2}\left(\vec{f}^{Netz}\left(\vec{x} \mid \vec{\mathbf{W}}\right) - \vec{F}\left(\vec{x}\right)\right)^2 \end{aligned} \tag{3.5}$$

Für den Lernschritt (*Update*) ergibt sich (Gradientenabstieg) für ein $\vec{x} \in \Omega$ für jedes Neuron k in jeder Schicht s das Inkrement zu

$$\Delta w_{ik}^{(s)} = -\epsilon \frac{\partial E}{\partial w_{ik}^{(s)}} \tag{3.6}$$

Die Ausführung der Differentiation nach der Kettenregel zeigt, daß dieses immer in der Form

Update Regel

$$\boxed{\Delta w_{ki}^{(s)} = -\epsilon \, d_k^{(s)} y_i^{(s-1)}} \tag{3.7}$$

analog zu **??** geschrieben werden kann, wobei die Fehleraktivität am Output wieder

$$d_k^{(M)} = \left(y_k^{Netz} - y_k^{soll}\right) S'\left(z_k^{(M)}\right) \tag{3.8}$$

ist, vgl. **??**. Für die Neuronen in den verborgenen Schichten stehen explizit keine Sollvorgaben zur Verfügung. Diese können aber wie die Ableitung zeigt (s.u.) aus den Fehleraktivitäten der darüberliegenden Schichten rekursiv gemäß

Propagation der Fehleraktivitäten

$$\boxed{d_k^{(s)} = \sum_{i=1}^{n^{(s+1)}} w_{ik}^{(s+1)} d_i^{(s+1)}} \tag{3.9}$$

ermittelt werden. Die Fehleraktivität zur Korrektur der Gewichte des Neurons k ergibt sich als die gewichtete Summe der Fehleraktivitäten aller der Neuronen, die vom betrachteten Neuron gespeist werden. Formeln 3.7 und 3.9 bilden den Backpropagation–Algorithmus. Der Fehler wird durch die Rekursion von 3.9 durch das Netz hindurch rückpropagiert. Daher der Name des Algorithmus.

3.2.2 Anhang: Herleitung des Backprop-Algorithmus für Feed-Forward Netzwerke

Für die Ableitung der Kostenfunktion nach den synaptischen Vektoren einer beliebigen Schicht s gilt

$$\begin{aligned} &\frac{\partial}{\partial w_{k,l}^{(s)}} E\left(\vec{x}, \vec{W}\right) = \sum_{i=1}^{n^{(\mathcal{M})}} \left(y_i^{(\mathcal{M})} - y_i^{soll}\right) \frac{\partial}{\partial w_{k,l}^{(s)}} y_i^{(\mathcal{M})} = \\ &\sum_{i=1}^{n^{(\mathcal{M})}} \left(y_i^{(\mathcal{M})} - y_i^{soll}\right) S'\left(z_i^{(\mathcal{M})}\right) \frac{\partial}{\partial w_{k,l}^{(s)}} z_i^{(\mathcal{M})} = \sum_{i=1}^{n^{(\mathcal{M})}} d_i^{(\mathcal{M})} \frac{\partial}{\partial w_{k,l}^{(s)}} z_i^{(\mathcal{M})} \end{aligned} \tag{3.10}$$

wobei dfer modifizierte Output-Fehler

$$d_i^{(\mathcal{M})} = \left(y_i^{(\mathcal{M})} - y_i^{soll}\right) S'\left(z_i^{(\mathcal{M})}\right) \tag{3.11}$$

wie früher definiert wurde. Für die output-Schicht wird (3.10)

$$\frac{\partial}{\partial w_{k,l}^{(\mathcal{M})}} E\left(\vec{x}, \{\vec{w}\}\right) = d_k^{(\mathcal{M})} y_l^{(\mathcal{M}-1)} \tag{3.12}$$

Für die darunterliegende Schicht mit $s = \mathcal{M} - 1$ folgt

$$\frac{\partial}{\partial w_{k,l}^{(s)}} E\left(\vec{x}, \{\vec{w}\}\right) = \sum_{i=1}^{n^{(\mathcal{M})}} d_i^{(\mathcal{M})} \frac{\partial}{\partial w_{k,l}^{(s)}} z_i^{(\mathcal{M})}$$

$$= \sum_{i=1}^{n^{(\mathcal{M})}} \sum_{j=1}^{n^{(\mathcal{M}-\infty)}} d_i^{(\mathcal{M})} \frac{\partial}{\partial w_{k,l}^{(s)}} w_{i,j}^{(\mathcal{M})} y_j^{(\mathcal{M}-1)}$$

$$= \sum_{i=1}^{n^{(\mathcal{M})}} \sum_{j=1}^{n^{(\mathcal{M}-\infty)}} d_i^{(\mathcal{M})} w_{i,j}^{(\mathcal{M})} \frac{\partial}{\partial w_{k,l}^{(s)}} S\left(\vec{w}_j^{(\mathcal{M}-1)} \cdot \vec{y}^{(\mathcal{M}-2)}\right)$$

$$= \sum_{i=1}^{n^{(\mathcal{M})}} d_i^{(\mathcal{M})} w_{i,k}^{(\mathcal{M})} S'\left(z_k^{(\mathcal{M}-1)}\right) y_l^{(\mathcal{M}-2)}$$

und damit

$$\frac{\partial}{\partial w_{k,l}^{(s)}} E\left(\vec{x}, \{\vec{w}\}\right) = d_k^{(\mathcal{M}-1)} y_l^{(\mathcal{M}-2)}, \qquad \text{für} \quad s = \mathcal{M} - 1$$

wobei die Fehleraktivität $d_k^{(\mathcal{M}-1)}$ am Ausgang von Neuron k der Schicht $\mathcal{M} - 1$ sich als Linearkombination der Fehler an der Schicht $\mathcal{M}$, d. h. der Folgeschicht (in feed-forward Richtung) der betrachteten Schicht, ergibt

$$d_k^{(\mathcal{M}-1)} = S'\left(z_k^{(\mathcal{M}-1)}\right) \sum_{i=1}^{n^{(\mathcal{M})}} d_i^{(\mathcal{M})} w_{i,k}^{(\mathcal{M})} \tag{3.13}$$

Der Fehler der Ausgabeschicht wird also auf diese Weise zurückpropagiert. Dieses Schema wiederholt sich in jeder Schicht.

Ganz allgemein gilt also für $0 < s < \mathcal{M}$

$$\boxed{\frac{\partial}{\partial w_{k,l}^{(s)}} E\left(\vec{x}, \{\vec{w}\}\right) = d_k^{(s)} y_l^{(s-1)}} \tag{3.14}$$

wobei

$$\boxed{d_k^{(s)} = S'\left(z_k^{(s)}\right) \sum_{i=1}^{n^{(s+1)}} d_i^{(s+1)} w_{i,k}^{(s+1)}} \tag{3.15}$$

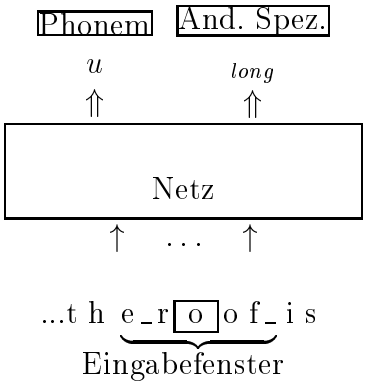
Für den Lernschritt ergibt sich damit die Regel 3.7 mit 3.9.

3.3 Einige Anwendungen

Im folgenden sollen einige Standardanwendungen neuronaler Netze wie NETTALK (das sprechende Netz) oder Zeichenerkennung vorgestellt werden, danach diskutieren wir einige Anwendungen mit Bottleneck–Netzen und untersuchen in 3.3.4 anhand eines Beispieles die Eignung von Feed–Forward–Netzen zur Modellierung kognitiver Prozesse. Weitere wichtige Anwendungen, vor allem für die Zeitreihenvorhersage, finden sich in späteren Kapiteln.

3.3.1 NETTALK

Ein für die Entwicklung der neuronalen Netze wichtige Anwendung war NETTALK, das unmittelbar nach der Erfindung des backpropagation Algorithmus von Sejnowski und Rosenberg (1987) entwickelt wurde. Hier handelt es sich um "das sprechende Netz", also um ein neuronales Netz, das lernt, geschriebenen Text in Lautumschrift zu übersetzen, die einem Sprachsynthesizer zugeführt werden kann. Der Zeichensatz bestand aus den 26 Buchstaben des englischen Alphabets und 3 Satzzeichen. Über den Text wurde ein Fenster geschoben, das sieben Zeichen umfaßt. Aufgabe des Netzes war es, unter Ausnutzung des Kontextes der Umgebungsbuchstaben im Fenster, dem mittleren Zeichen im Fenster das richtige Phonem und weitere Aussprachespezifikationen zuzuordnen. Die Zeichen wurden in einer 1 aus n Kodierung (unäre Kodierung) dargestellt, das Netz "sah" folglich $7 \times 29 = 203$ Inputunits. Für die Darstellung der Outputinformation dienten im gleichen Stil 26 Neuronen. Die Zwischenschicht bestand aus 120 Neuronen.



Es müssen in dieser Architektur (zusätzlich zu den Schwellwerten der Neuronen) also 27 480 Gewichte adaptiert werden.

Nachdem das Netz mit zufälligen Wortkombinationen aus einer Bibliothek von 2000 Wörtern hinreichend lange trainiert wurde konnte es mit einer Trefferquote von 91% die richtige Aussprache der vorgelegten Sätze produzieren. Diese Leistung vergleicht sich mit dem kommerziellen System DECTALK, für dessen Entwicklung immerhin ca. 20 Mannjahre aufgewendet wurden und das eine Trefferrate von 95% vorweisen kann. Zur richtigen Einschätzung des neuronalen Ansatzes ist aber zu sagen, daß eben die fehlenden 4% sich gerade aus der Beachtung im allgemeinen komplizierter Ausnahmeregeln ergeben, die dem neuronalen Netz einzulernen sich als außerordentlich schwierig erweist. Das Netz hat ja keinerlei Verständnis der Regeln der englischen Sprache, es muß diese allein aus den vorgelegten Beispielen inferieren. Das ist aber für die im allgemeinen seltenen Ausnahmen besonders schwierig. Eventuell ist dann auch die durch die umgebenden 6 Buchstaben gelieferte Kontextinformation nicht ausreichend, das Netz müßte eigentlich zusätzlich ein Wörterbuch erlernen, in dem die Ausnahmen abgelegt sind. Das ist aber in der gewählten Architektur nicht möglich. Es haben sich

Hybride Architekturen

deshalb in den letzten Jahren vor allem **hybride Architekturen** durchgesetzt, die eine Kombination von Regelwissen mit neuronalen Netzen realisieren.

3.3.2 Zeichenerkennung

In Vorbereitung.

. . .

3.3.3 Bottleneck–Netze zur Dimensionsreduktion und adaptiven Datenkompression

Viele Problemstellungen in der Informatik laufen auf das Auffinden geeigneter dimensionsreduzierter Darstellungen gegebener Datenverteilungen hinaus. Häufigstes Beispiel ist sicher die Datenkompression aber auch das Auffinden effektiver-Kodierungen gehört zu diesem Problemkreis. Oft geht die Dimensionsreduktion zusätzlich mit einer Informationsreduktion einher, wie z. B. bei der Hauptkomponentenanalyse zum Auffinden der relevanten Information in einer Datenmannigfaltigkeit. In der Signalverarbeitung werden diese Verfahren auch zur Datenentrauschung verwendet.

Der Zugang mit neuronalen Netzen hat den Vorteil, daß (i) diese Eigenschaften von den Netzen erlernt werden und daß (ii) die neuronlaen Netze oft recht zwanglos nichtlineare Verallgemeinerungen der bekannten klassischen Verfahren realisieren. Ein recht eleganter Zugang ergibt sich mit den sog. "bottleneck" Netzen. Dabei wird das Netz gezwungen, die Information möglichst unverfälscht durchzulassen, d. h. der Solloutput ist gleich dem Input

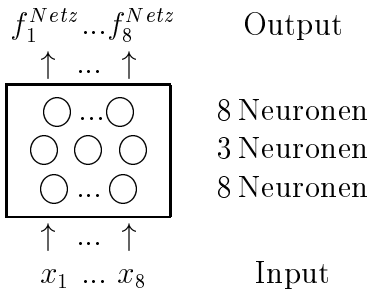
Bottle–neck Architektur

$$\vec{f}^{Netz}(\vec{x}) = \vec{y}^{soll}(\vec{x}) = \vec{x} \tag{3.16}$$

wobei aber das i. a. mehrschichtige Netz mindestens eine Zwischenschicht hat, die aus einer im Vergleich zur Zahl n der Inputunits kleinen Anzahl von Neuronen besteht. Durch diesen "Flaschenhals" muß dann die gesamte Information hindurch, was nur gut geht, wenn diese Schicht eine möglichst effektive Darstellungen für die Inputs $\vec{x} \in \Re^n$ findet.

Auffinden optimaler Kodierungen der Inputinformation

Beispiel ist die "Entdeckung" des Binärkodes durch ein Netz



(3.17)

das als Inputs und Solloutputs gemäß 3.16 die Zahlen eins bis acht in einer 1 aus n Kodierung, also z. B. $(0,0,1,0,0,0,0,0)$ für die Zahl drei, bekommt. Als Ergebnis eines erfolgreichen Trainings entsprechen die Outputs der Neuronen in der Zwischenschicht der Darstellung der Zahlen im Binärcode, also z. B. $(1,1,0)$ für den o. g. Input (bis auf Permutationen zwischen den Neuronen).

Umkodierung der Inputinformation

Dieses Prinzip kann vielfältig angewendet werden. Bei verrauschten Datenmannigfaltigkeiten repräsentieren die Zwischenneuronen die Hauptkomponenten der Daten, genauer gesagt kann man die Outputs der Neuronen in der Zwischenschicht als Darstellung der Inputs in einem i. a. nichtlinearen, krummlinigen Koordinatensystem betrachten, das aber so gewählt ist, daß es die für die gewünschte originalgetreue Rekonstruktion der Inputs in den Folgeschichten erforderliche Information möglichst vollständig enthält. Greift man also diese Zwischenoutputs ab, so hat man einen (i. a. sogar nichtlinearen) Hauptkomponentenanalysator "erlernt".

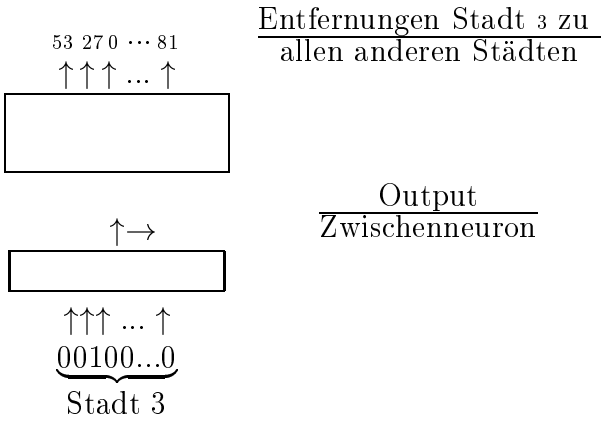
Nichtlineare Hauptkomponentenanalyse

Lösung von Optimierungsproblemen

Eine interessante Anwendung zur Lösung von Optimierungsproblemen wurde kürzlich von Wiles et. al. vorgeschlagen[8]. Die Lösung des NP–vollständigen Problems des reisenden Handelsmannes,der N Städte nacheinander in einer wegoptimierten Reihenfolge aufsuchen soll, wird von den Autoren auf die Abbildung der Landkarte mit den N Städten auf einen eindimensionalen Weg zurückgeführt. Die Autoren verwenden ein $N-1-k-N$ Bottleneck–Netzwerk. Die Inputs sind die Städteindizes, wobei Stadt i durch ein N–Bit Wort repräsentiert ist bei dem nur das i–te Bit gesetzt ist (1 aus n Kodierung), d. h. Stadt $i \quad \Leftrightarrow (0,0,...,1,...,0)$. Diese Kodierung garantiert die informationelle Gleichstellung aller Inputs. Informationen über die relative Lage der Städte ist nur in den Outputs enthalten. Dessen Komponenten sind gerade die Entfernungen der betrachteten Stadt i zu

Travelling Salesman Problem

allen anderen Städten.



Da der Flaschenhals aus einem einzigen Neuron besteht, kann die Output–Information in den nachfolgenden zwei Schichten erfolgreich nur dann aus der Aktivität $y^{(1)}$ des Zwischenneurons konstruiert werden, wenn in der internen, durch den Output des Zwischenneurons gegebenen eindimensionalen Darstellung sich die Ähnlichkeitsrelationen zwischen den Outputvektoren widerspiegeln. Das ist genau dann der Fall, wenn benachbarte Städte auch in der eindimensionalen Darstellung benachbart sind, denn die Soll—Outputs (Entfernungen zu allen anderen Städten) unterscheiden sich gerade für benachbarte Städte am wenigsten voneinander. Greift man den Output des Zwischenneurons ab, so sollte dieser der Lage der Stadt i auf einem Pfad minimaler Entfernung zwischen aufeinanderfolgenden Städte entsprechen.

Das Verfahren wurde mit $N = 10$ und $N = 30$ Städten in zufällig gewählten Lagen getestet. Dabei wurde $k = 4$ bzw. $k = 10$ als günstig gefunden. Die Ergebnisse vergleichen sich mit den Resultaten bekannter Optimierungsverfahren, die speziell für die Lösung des "travelling salesman" Problemes entwickelt wurden.

Interessant ist, daß hier das allgemeine Stetigkeitsprinzip "ähnliche Inputs produzieren ähnliche Outputs" zum Verständnis neuronaler Netze in gewisser Weise auf den Kopf gestellt wurde. Hier führen ähnliche Outputs zu ähnlichen internen Repräsentation.

3.3.4 Emergenz getrennter Verarbeitungskanäle für Objekt– und Positionsinformation im visuellen System

Eine der grundlegenden Leistungen des visulellen System besteht in seiner Fähigkeit, Objekte unabhängig von Position und Größe ihres Abbildes auf der Retina sicher identifizieren zu können. Eine genauere Untersuchung dieser durch seine Alltäglichkeit fast als selbstverständlich angenommenen Fähigkeit offenbart die

hohe Komplexität der unterliegenden Verarbeitungsleistungen. Eine vom neuronalen Substrat abstrahierende, rein komputationale Theorie betrachtet Position und Identität des Objektes als formale Attribute, die in getrennten Informationskanälen oder Funktionseinheiten verarbeitet werden können[1]. Diese Vermutung scheint auch durch neuroanatomische Erkenntnisse gestützt zu sein, die allerdings nicht unumstritten sind.

Dem kann eine rein repräsentationale Theorie gegenübergestellt werden, die das gleiche Objekt in allen möglichen Positionen abgespeichert sieht. Das spart Komplexität in der Verarbeitung, allerdings, im Vergleich zu der o. g. funktionellen Segregation, um den Preis eines viel höheren Speicherbedarfs.

Um zwischen beiden Varianten besser abwägen zu können, entwarfen Kosslyn et al. in [5], [?] ein neuronales Netzwerkmodell für diesenAspekt der visuellen Informationsverarbeitung. Der Input in das als Feed–Forward–Netz ausgelegte Modell bestand aus einer "Retina" mit 5×5 Pixeln, auf der jeweils eines von 9 möglichen Objekten präsentiert wurde. Die Schicht der Output–Neuronen unterteilte sich in eine Gruppe von 9 Neuronen zur Repräsention der 9 verschiedenen Objekte in einer 1 aus n Kodierung plus eine Gruppe von weiteren 9 Neuronen zur Repräsentation der $3 \times 3 = 9$ möglichen Positionen des Objektes (Die Ausdehnung der Objekte schloß die Randpixel der Retina als Objektpositionen aus). Außerdem verfügte das Netz über eine verborgene Schicht, die mit 18 Neuronen so ausgestattet wurde, daß sie potentiell die Information über 'what' and 'where' im robusten 1 aus n Code repräsentieren konnte.

Das Netz wurde mit dem Backpropagation Algorithmus anhand der Muster und den Sollvorgaben für die 'what' und 'where' Outputneuronen trainiert. Im Ergebnis zeigte es sich, daß das Netz sehr wohl in der Lage war, sowohl (i) die Position als auch (ii) die Identität eines Objektes unabhängig von dessen Position zu erkennen.

Die an dem sehr einfachen 5×5 Beispiel gewonnen Aussagen mögen natürlich wenig verallgemeinerungsfähig erscheinen. Wir untersuchten deshalb ein analoges Modell [1] mit einer deutlich größeren Retina aus 11×11 Pixeln. Trotz der um einen Faktor 5 größeren Anzahl von Pixeln beobachteten wir in etwa das gleiche Verhalten. Das betraf insbesondere die Herausbildung der strukturierten rezeptiven Felder (s. u.) bei den für die Verarbeitung der 'where' Information zuständigen Zwischenneuronen, vgl. Abbildungen 3.5 und 3.6.

Interessant erscheint zunächst die Art wie das Netz das Problem gelöst hat. Wie aus den Abbildungen 3.5 und 3.6 ersichtlich, entwickelten sich die Zwischenneuronen zu Feature–Detektoren, die jeweils auf ganz bestimmte Teilfeatures der Objekte sensitiv reagieren. So bildeten sich insbesondere globale (d. h. positi-

[1] Andererseits sollten beide Verarbeitungskanäle auch kooperieren können. In der Tat mag unter einem semantischen Aspekt die Position eines Objektes in einer komplexen Szene Kontextinformation für die Identifikation des Objektes liefern. Umgekehrt kann die Position eines Objektes nur dann zweifelsfrei und eindeutig bestimmt werden, wenn das Objekt als Ganzes identifiziert worden ist.

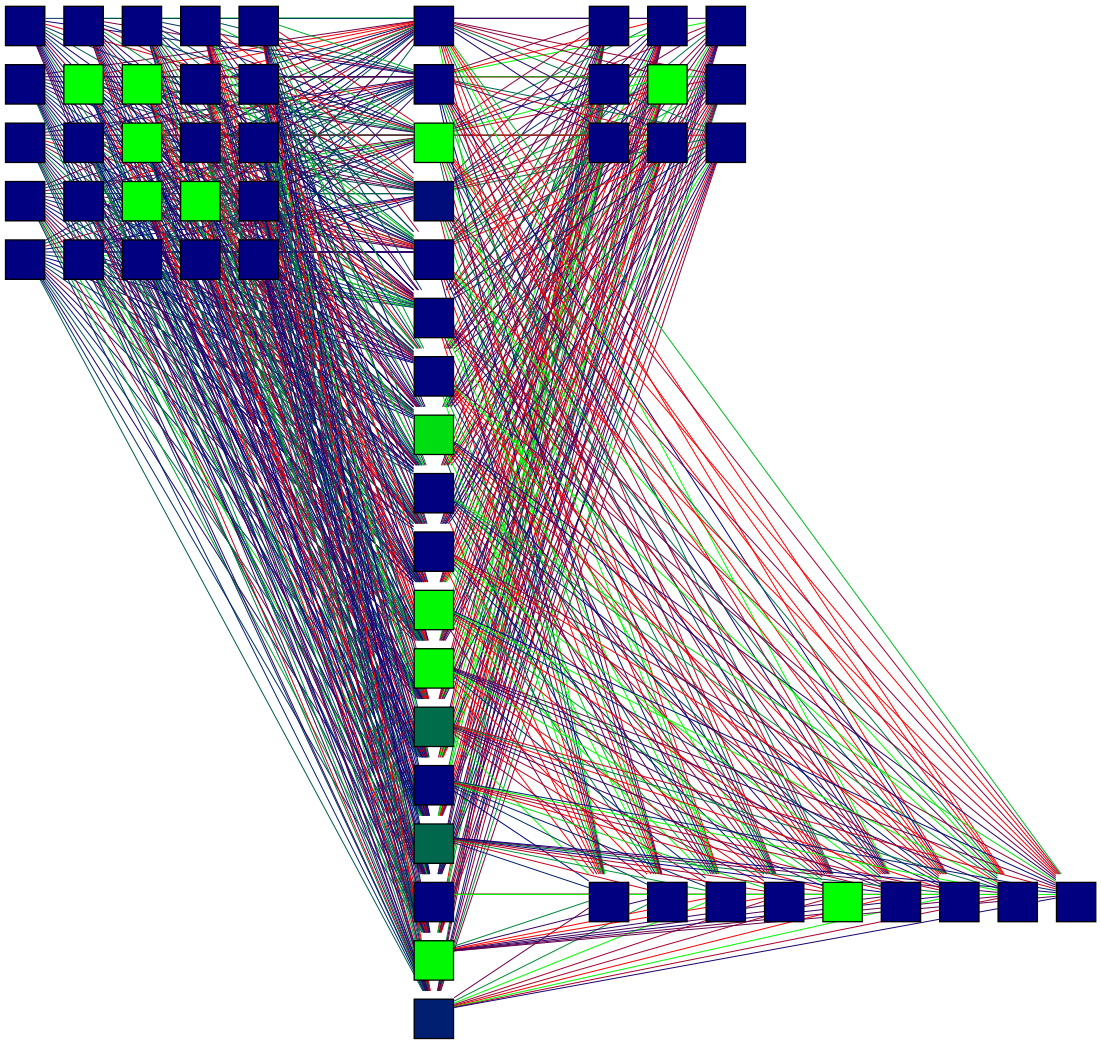


Abbildung 3.1: Das ursprünglich von Rueckl, Cave und Kosslyn verwendete Netz. Wir sehen im linken Teil die Retinaäus 5×5 Pixeln mit einem der Objekte in der Bildmitte. Die Retina speist eine Lage (hidden layer) von 18 Zwischenneuronen und diese wiederum zwei Gruppen von Output–Neuronen, oben die "whereünd darunter in horizontaler Anordnung die "whatNeuronen. Die Farbe (Helligkeit in der s/w Darstellung) der Neuronen kodiert ihre Erregung.

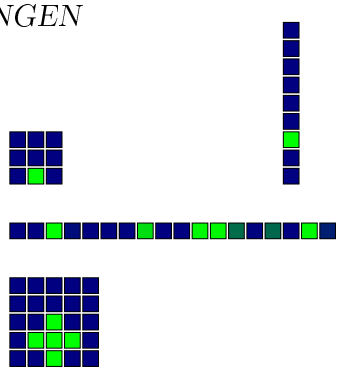


Abbildung 3.2: Das Netz von Abb. 3.1 mit einem weiteren Objekt auf der Retina. Die Darstellung wurde um 90^o gedreht.

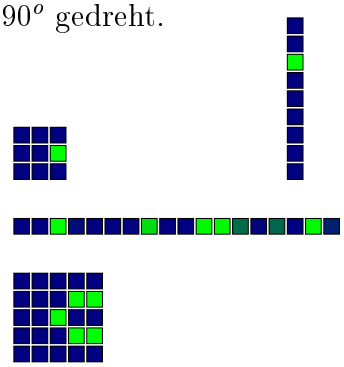


Abbildung 3.3: Wie Abbildung 3.2

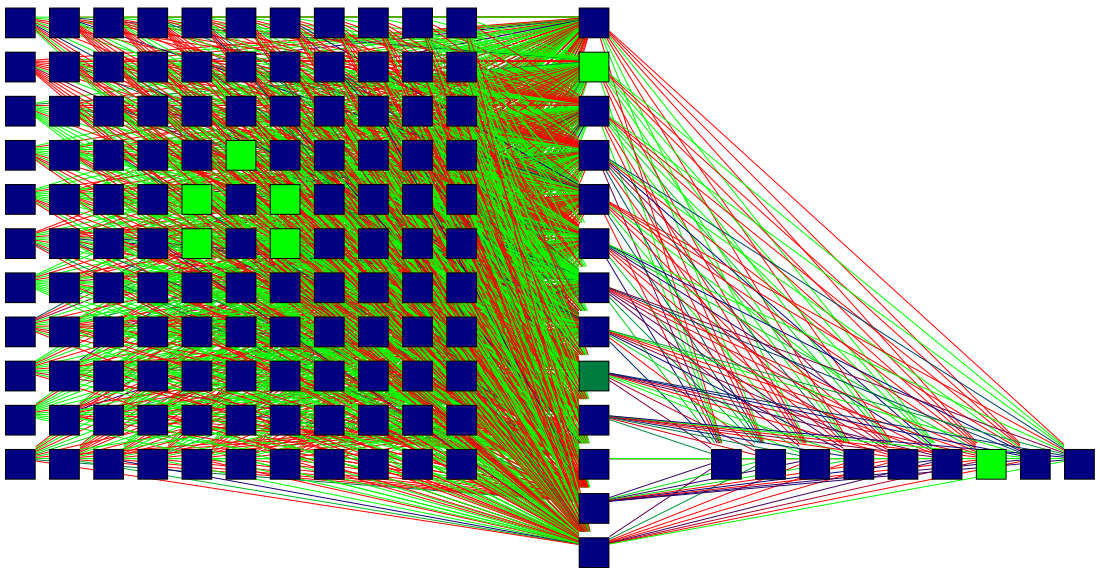


Abbildung 3.4: Wie Bild 3.1 für ein Netz mit einer Retina aus $11x11$ Pixeln. Im Gegensatz zu Bild 3.1 sind die 'where' Outputneuronen aus Platzgründen nicht mit dargestellt.

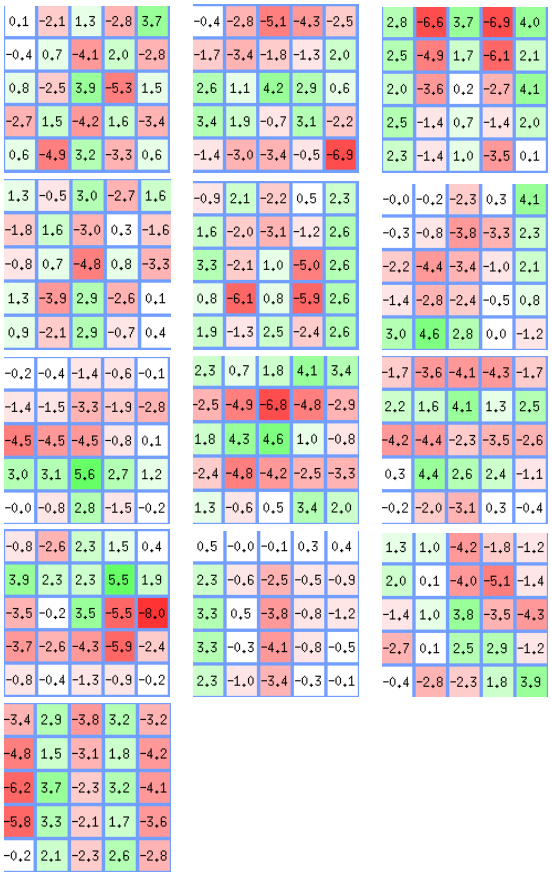


Abbildung 3.5: Die rezeptiven Felder der Zwischeneuronen. Die Abbildung zeigt die Retina, so wie sie von jedem der Zwischenneuronen "gesehen" wird, d. h. wie stark ein Pixel der Retina zur Erregung des Zwischenneurons beiträgt: Jedes Quadrat zeigt die synaptischen Kopplungsstärken von allen Pixeln der Retina zu jeweils einem der 10 Zwischenneuronen, die sich für die Verarbeitung der 'what' Information herausgebildet haben.

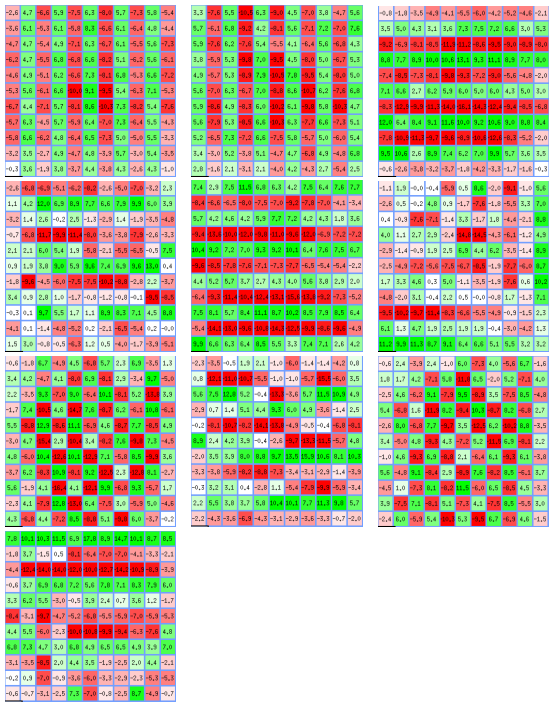


Abbildung 3.6: Wie Abbildung 3.5 aber für den Fall der 11×11 Retina. Die strukturierten rezeptiven Felder für Kanten und Schachbrettmuster sind deutlich zu erkennen.

onsinvariante) Detektoren für horizontale und vertikale Kanten heraus und sehr deutlich auch zwei Detektoren für das Vorhandensein schachbrettartiger Erregungsmuster auf der Retina. Diese Features werden durch die Zwischenneuronen einzeln detektiert, die Aufgabe der Outputneuronen ist dann nur noch, auf die zu einem bestimmten Objekt gehörige Kombination dieser Features selektiv anzusprechen. Diese Funktionalität entsprach speziell der Detektion der 'what' Information. Für das 'where' bildeten andere Zwischenneuronen rezeptive Felder aus, die sich einfach auf den Schwerpunkt der retinalen Erregung konzentrierten. Diese rezeptiven Felder können als simple, die der 'where' Zwischenneuronen als komplexe oder strukturierte charakterisiert werden.

Das Modell ist natürlich recht biologiefern. Insbesondere hat der Backpropagation–Algorithmus keine biologische Entsprechung. Die Ergebnisse können dennoch für die Entscheidung zwischen verschiedenen Paradigmen der visuellen Informationsvearbeitung herangezogen werden. Die Untersuchungen scheinen nämlich auf eine allgemeine Tendenz im Verhalten komplexer adaptiver Systeme, wie das betrachtete neuronale Netz, hinzudeuten. Der äußere Zwang, die retinalen Inputs auf getrennte Outputmodalitäten (what und where) abzubilden, manifestiert sich auch intern – in Form der ausgebildeten Feature–Detektoren – in einer Aufteilung der Verarbeitungsressourcen auf die verschiedenen Kanäle. Spekulativ könnte man weiter schlußfolgern, daß sich in der Evolution diese Trennung der Verarbeitung dann auch so herausgebildet haben sollte, wenn die getrennte Verarbeitung für die **nachgeschalteten** Verarbeitungseinheiten einen selektiven Vorteil darstellt. Das ist für Einheiten, die auf einer höheren abstrakten Ebenen arbeiten, aber durchaus anzunehmen.

.

.

3.4 Architekturen neuronaler Netze

Die Universalarchitektur $n-k-m$ eines neuronalen Netzes mit einer Zwischenschicht aus k Neuronen ist zwar bei hinreichend großem k zur Approximation einer jeden Funktion geeignet, in der Praxis ist es aber zur Beschleunigung des Lernens und zur Verbesserung der Generalsierungsleistungen des Netzes besser, auch andere Architekturen auszuprobieren. Für die praktische Arbeit hat es sich als erfoglreich erwiesen, immer mehrere Netze verschiedener Architektur gleichzeitig zu trainieren, bis sich eine Architektur als überlegen erweist.

3.4.1 XOR Netzwerke

Wie oben dargestellt, ist die Boolsche Funktion XOR nicht durch ein einzelnes Neuron realisierbar. Es gibt nach [4] insgesamt 16 Netze minimaler Komplexität, die eine Lösung des Problems darstellen. Das einfachste ist ein $2-1-2$ Netzwerk,

wobei aber im Unterschied zu den bisher behandelten BP–Netzwerken direkte Verbindungen von den Inputs zum Outputneuron bestehen. Mit dem Output des einzelnen Neurons in der Zwischenschicht sieht das Outputneuron folglich drei Inputs, wir haben somit intern eine dreidimensionale Darstellung der Inputsignale. Diese kann geeignet so gewählt (gelernt) werden, daß die Klassentrennung sich auf ein lineares Problem reduziert. Das ist ein Beispiel für die "Strategie Dimensionserweiterung" zur Überführung des Problems in ein einfacher lösbares.

Strategie Dimensionserweiterung

Kürzlich konnte gezeigt werden, daß die Energielandschaft des XOR Poblems nur globale Minima hat, so daß das Gradientenabstiegsverfahren mit Sicherheit gegen die richtige Lösung konvergiert[6].

3.4.2 Erlernen Boolscher Funktionen

Bei verrauschten Daten oder zu kleinen Datensätzen droht immer das Problem der Überanpassung, das weiter unten behandelt wird. Hier soll zunächst der Fall "präziser" Daten untersucht werden, wie er insbesondere beim Erlernen einer Boolschen Funktion auftritt. Hier besteht das Problem des Overfitting nicht. Hat man ein Netz mit zu hoher Komplexität gewählt, so kann man damit rechnen, daß redundante Neuronen sich selbst "abschalten", d. h. ihre Verbindungen mit anderen Neuronen gehen im Laufe des Lernens gegen Null. Beispiel: Wird ein $2-3-1-2-...-1$ Netzwerk auf die XOR Funktion trainiert, dann wird i. a. diese schon in den ersten zwei Schichten des Netzes realisiert, die folgenden Neuronen leiten den Output des einzelnen Zwischenneurons in der zweiten verborgenen Schicht nur noch weiter, dort können also Neuronen freigesetzt werden, ohne die Funktion des Netzes zu beeinträchtigen.

Dieses Schema scheint ein wichtiges architektonisches Entwicklungsprinzip in der Frühphase der Hirnentwicklung zu sein. Die für eine bestimmte Funktion sich im Verlaufe des Lernens herausbildenden neuronalen Netze sind in gewisser Weise minimal, d. h. sie beziehen nur die unbedingt erforderliche Anzahl von Neuronen ein. Voraussetzung ist aber die "Reinheit" des Datensatzes, wie eben beim Erlernen einer Boolschen Funktion. Die freiwerdenden Neuronen stehen für andere Aufgaben zur Verfügung und werden für diese aufgrund der allgemein vorherrschenden Wettbewerbssituation auch rekrutiert.

Ökonomie der Ressourcen

3.4.3 Selbstorganisierende Architekturen

Der Idealfall ist natürlich das Netz, das sich die optimale Architektur – ausgehend von einer minimalistischen Startarchitektur – selbst sucht. Es existieren eine Reihe von Verfahren zur Lösung dieses Problems, wir wollen hier vor allem das Kaskadenkorrelationsnetz besprechen.

Kaskadenkorrelation

Die Methode der Kaskadenkorrelation dient der automatisierten Konstruktion neuronaler Netze mit minimaler Komplexität zur Lösung einer gegebenen

Aufgabenstellung. Ausgehend von einer Startarchitektur (nur Input- und Outputschicht) besteht das Grundprinzip im Hinzufügen immer weiterer Neuronen (hidden units). Diese fungieren als Repräsentanten für in der aktuellen Architektur noch nicht (oder nicht ausreichend) repräsentierte Merkmale der Inputs. Im Gegensatz zu den "klassischen" Feed-Forward-Architekturen verbinden die Synapsen auch nichtaufeinanderfolgende Schichten miteinander.

Der Algorithmus zur Konstruktion des Netzes und zum Anlernen der jeweils neu hinzukommenden hidden Units besteht aus folgenden Schritten:

1. Initialisierung als Minimalarchitektur bestehend aus der Input- und Outputschicht.

2. Training aller in die Outputunits einlaufenden Verbindungen.

3. Generierung der sog. Kandidatenunits. Das sind hidden Units, die von allen Inputunits und von allen schon existierenden hidden Units gespeist werden.

4. Training aller in die Kandidatenunits **einlaufenden** Verbindungen mit dem Ziel der Maximierung der Korrelationen zwischen dem Output der Kandidatenunit und dem **Fehler** des Netzwerkes.

5. Die beste Kandidatenunit (maximale Korrelation) wird als neue hidden Unit dem Netz hinzugefügt, indem sie als neue Inputeinheit für die Outputschicht interpretiert wird. Ihre Synapsen werden eingefroren.

6. Abbruch? Gehe zurück zu 2.

Eine ausführlichere Beschreibung findet sich in [9].

Interessant ist, daß sich XOR **nicht** mit diesem Verfahren realisieren läßt. Im Gegenteil, der Algorithmus konvergiert nicht, sondern generiert Netze mit bieliebig vielen Neuronen, ohne daß eine Lösung entsteht. Grund ist, daß schon im Schritt 2 die Synapsen des Outputneurons auf Null gelernt werden (synaptischer Kollpas), da die XOR–Funktion für das Einzelneuron nicht lernbar ist. Im Schritt 4 wiederholt sich dann das Dilemma, da das Outputneuron wegen seiner praktisch auf Null gelernten Synapsen keine sicheren Informationen über den Input zur Verfügung stellen kann und demzufolge auch die Kandidatenunits nicht wissen können, auf welche Features der Inputs sie sich konzentrieren sollen. So werden deren Synapsen ebenfalls jeweils auf Null gelernt, da sie nur einander widersprechende Fehlerinformationen erhalten.

Andererseits wurden mit den Kaskadenkorrelationsnetzen auch eine Reihe beeindruckender Ergebnisse erzielt.

3.5 Nichtlineare Regression

Wie im folgenden Abschnitt 3.5.1 dargestellt, können Feed–Forward–Netze Funktionen beliebig genau approximieren. Das besondere an neuronalen Netzen ist, daß sie funktionelle Zusammenhänge aus Beipielen (Instanzen des Funktionszusammenhanges) erlernen können. Das gilt insbesondere auch, wenn diese Daten z. B. durch Meßfehler verrauscht sind. Die Inferenz funktioneller Zusammenhänge aus verrauschten Daten wird allgemein als Regression bezeichnet. Für den Fall linearer Funktionen stehen dafür eine Reihe von Standardverfahren zur Verfügung, die die Parameter der Regressionsgeraden bestimmen. Der backprop Algorithmus zur Bestimmung der Parameter des Funktionsmodells Feed–Forward–Netz, kann als nichtlineare Verallgemeinerungen der linearen Regressionsverfahren betrachtet werden.

Regression

3.5.1 Feed–Forward Netze als universelle Funktionsapproximatoren

Die FFN können als parametrisierte Funktionsdarstellungen angesehen werden. Für jede Wahl einer bestimmten Architektur und des Parametersatzes (gegeben durch die Menge aller synaptischen Verbindungen, der Schwellen und des Steilheitsparameters β der Ausgabefunktion der Neuronen) ergibt sich eine bestimmte Funktion. Zu fragen ist nun nach der Mächtigkeit dieser Darstellung, d.h. ob damit der gesamte Funktionsraum überdeckt oder anders gefragt, ob auf diese Weise jede beliebige Funktion dargestellt werden kann. Die Antwort findet sich in dem Satz von Hornik, Stinchcombe und White, [3]):

- Schon ein Netz, gegeben durch seine Transferfunktion

  $$\vec{y} = F\left(\vec{x}, \{\vec{w}\}\right) \tag{3.18}$$

  mit einer einzigen verborgenen Schicht kann eine beliebige (hinreichend glatte) Funktion $\vec{y} = f\left(\vec{x}\right)$ beliebig genau approximieren, d. h.

  Satz über Funktionsapproximation

  $$\exists\, \vec{w}^{opt} : F\left(\vec{x}, \left\{\vec{w}^{opt}\right\}\right) \simeq f\left(\vec{x}\right) \quad \forall \vec{x}$$

  bzw.

  $$\text{Fehler} \quad E\left(\left\{\vec{w}^{opt}\right\}, \vec{x}\right) = \left(f\left(\vec{x}\right) - F\left(\vec{x}, \left\{\vec{w}^{opt}\right\}\right)\right)^2 \simeq 0, \quad \forall \vec{x} \tag{3.19}$$

Diese Fähigkeit aber eher von rein akademischem Interesse. Typischer Fall in der Praxis: Die Menge Ω der Trainingsdaten $\vec{\phi} \in \Omega = (\Omega_{in}, \Omega_{out})$

$$\vec{\phi} = \left(\vec{x}, \vec{y}^{soll}\right), \quad \vec{y}^{soll} = f\left(\vec{x}\right), \quad \vec{x} \in \Omega_{in} \tag{3.20}$$

ist nur klein, die Daten sind zudem "verrauscht". Die Bestimmung

$$\vec{w}: \quad E\left(\{\vec{w}\},\vec{x}\right)=\left(\vec{y}^{soll}-F\left(\vec{x},\{\vec{w}\}\right)\right)^2\simeq 0, \quad \forall\vec{x}\in\Omega_{in}$$

entspricht einer Überanpassung (overfitting) des Netzes an die Daten ("das Rauschen wird mit angefittet"). Ein solches Netz hat es damit nicht oder nur unvollständig geschafft, die unter dem Rauschen liegende Gesetzmäßigkeit richtig zu erfassen. Das äußert sich z. B. darin, daß ein solches Netz neu hinzukommende Daten in der Regel nur sehr schlecht repräsentieren kann.

3.5.2 Prognosegenauigkeit, Generalisierungsfähigkeit, Overfitting

Im folgenden sollen einige Verfahren zum Umgang mit "verrauschten" Daten vorgestellt werden.

Aufteilung in Trainings– und Testmenge

Trainings– und Testdaten

Liegen ausreichend viele Daten vor, so kann Ω aufgeteilt werden in einen (eigentlichen) Trainigs- und einen Testsatz, $\Omega=\{\Omega^{train},\Omega^{test}\}$. Training mehrerer Netze $n=1,2,...$ auf Ω^{train} liefert $\left\{\vec{w}^{(n)}\right\}$ durch Minimierung des Trainingsfehlers

$$\left\{\vec{w}^{(n)}\right\} \quad : \quad E\left(\left\{\vec{w}^{(n)}\right\}\right)=\sum_{\phi\in\Omega^{train}} E\left(\left\{\vec{w}^{(n)}\right\},\vec{x}\right) \tag{3.21}$$

$$\tag{3.22}$$

$$= \sum_{\phi\in\Omega^{train}}\left(\vec{y}^{soll}-F\left(\vec{x},\left\{\vec{w}^{(n)}\right\}\right)\right)^2\Rightarrow \text{Min} \tag{3.23}$$

Dann Bestimmung des mittleren quadratischen Prognosefehlers auf Ω^{test}

$$V\left(\left\{\vec{w}^{(n)}\right\}\right)=\sum_{\phi\in\Omega^{test}} E\left(\left\{\vec{w}^{(n)}\right\},\vec{x}\right)=\sum_{\phi\in\Omega^{test}}\left(\vec{y}^{soll}-F\left(\vec{x},\left\{\vec{w}^{(n)}\right\}\right)\right)^2 \tag{3.24}$$

Auswahl des besten Netzes das V (oder eine geeignete Kombination von E und V) minimiert. Der mittlere quadratische Prognosefehler ist gleichzeitig ein Maß für die **Verallgemeinerungsfähigkeit des Netzes**.

Sind nur wenige Daten vorhanden, so kommen folgende Verfahren zur Anwendung:

Kreuzvalidierung (cross validation)

Wähle $\forall\phi_i, \quad i=1,2,...,N^{\Omega}, \quad \phi_i\in\Omega$ den reduzierten Datensatz $\Omega^{(i)}=\Omega\backslash\phi_i$ (d. h. $\Omega^{(i)}$ ist die Menge aller Trainingsbeispiele ohne ϕ_i), bestimme $\vec{w}^{(i)}$

$$\vec{w}^{(i)} \quad : \quad E\left(\left\{\vec{w}^{(i)}\right\}\right)=\sum_{\phi\in\Omega^{(i)}} E\left(\left\{\vec{w}^{(i)}\right\},\vec{x}\right)\Rightarrow \text{Min}$$

Kreuzvalidierung mit der mittlere quadratische Prognosefehler zu

$$V\left(\{\vec{w}\}\right) = \frac{1}{N^{\Omega}} \sum_{i=1}^{N^{\Omega}} \sum_{\phi} E\left(\left\{\vec{w}^{(i)}\right\}, \vec{x}\right)$$

abgeschätzt werden kann. Dieser dient dann der Auswahl des besten Netzes wie oben. Standardverfahren zur Auswahl des besten Netzes.

Der Bootstrap Algorithmus

Relativ neues Verfahren in der Neuroinformatik, s. [[2]], [[?, Efron94] Statt den Datensatz um jeweils ein Trainigsbeipiel zu reduzieren werden hier aus dem Trainigssatz Ω neue Datensätze $\Omega^{(i)}$ durch Stichprobenziehung (mit Zurücklegen) gebildet. Der Algorithmus besteht aus den Schritten:

1. Aus Ω werden N^{Ω} Datenvektoren ϕ mit Zurücklegen gezogen. Die resultierende Stichprobe $\Omega^{(b)}$ enthält einige Paare mehrfach und andere (im Mittel 36%) überhaupt nicht.

2. Das Netzwerk wird $\forall b = 1, 2, ..., N^{(b)}$ mit dem Trainingssatz $\Omega^{(b)}$ trainiert. Die so erhaltenen $\left\{\vec{w}^{(b)}\right\}$ bilden die Bootstrap-Verteilung, die als eine Menge unterschiedlicher Schätzwerte für die Parameter $\vec{w}$ betrachtet werden kann und die die Variationsbreite der möglichen Parameter überstreicht.

3. Die Güte der Approximation durch das Netz n kann dann durch die Varianz v der Ausgaben $y = F\left(\vec{x}, \left\{\vec{w}^{(b)}\right\}\right)$ (genauer der entsprechenden Prognoseverteilungen) bezüglich der $\vec{w}^{(b)}$ bestimmt werden, d. h. berechne

$$v = \left\langle \left\langle \left(y - \langle y \rangle^{(b)}\right)^2 \right\rangle^{(b)} \right\rangle^{(\Omega)}$$

wobei $\langle ... \rangle^{(b)}$ die Mittelung über die Ausgaben bezüglich der verschiedenen Gewichtsvektoren und $\langle ... \rangle^{(\Omega)}$ die über den Datensatz bedeutet. Die Varianz bestimmt dann wieder die Auswahl des besten Netzes. (Die Idee dahinter ist natürlich, daß für das ideale Netz und nichtverrauschte Daten alle $\vec{w}^{(b)}$ übereinstimmen und damit die Varianz der Ausgabewerte gleich Null wird).

3.5.3 Verfahren zur Verbesserung der Generalisierungsfähigkeit

Obige Verfahren erlauben unter zwei oder mehreren Netzen die geeignetsten auszuwählen. Wegen des Aufwandes beim Training kann man aber nicht beliebig viele und beliebig komplexe Netze betrachten. Für Konstruktion und Training möglichst effektiver Netze stehen eine Vielzahl von Verfahren zur Verfügung, z. B.

Weight decay

Weight decay

Überanpassung ist oft mit dem "Weglaufen" der Parameter (synaptische Verbindungsstärken w_{ij}) nach sehr hohen Werten verbunden. Vermeidung durch Hinzufügen eines Straftermes zur Kostenfunktion, der hohe w Werte bestraft:

$$E\left(\{\vec{w}\},\vec{x}\right) \Rightarrow E\left(\{\vec{w}\},\vec{x}\right) + \lambda \sum_{i,j} w_{i,j}^2 \tag{3.25}$$

Pruning

Pruning zur Verbesserung der Generalisierungsfähigkeit

Beim Prunig werden nach dem Training synaptische Verbindungen zufällig mit einer bestimmten Wahrscheinlichkeitsverteilung gekappt. Vornehmlich solche mit niedrigen Werten für die Verbindungsstärken $w_{i,j}$ bzw. mit starken Fluktuationen beim Lernen.

3.6 Eigenschaften und Varianten des Backprop-Verfahrens

Die Fehlerlandschaft ist im allgemeinen sehr komplex und durch eine Vielzahl lokaler Minima sowie durch stark unterschiedliche Anstiege der Täler gekennzeichnet. Das beeinflußt die Geschwindigkeit und Sicherheit der Konvergenz in erheblichem Maße. Es wurden deshalb eine Vielzahl von Verfahren zur Überwindung dieser Probleme entwickelt, von denen wir im folgenden die wichtigsten besprechen wollen. Zunächst soll aber eine interessante Eigenschaft des Algorithmus dargestellt werden.

3.6.1 Die Austauschbarkeit von Steilheit und Lernrate

In [7] wurde gezeigt, daß die Veränderung des Steilheitsparameters β der Ausgabefunktion durch eine Änderung der Lernrate und der synaptischen Gewichte kompensiert werden kann. Da die Gewichte gelernt werden und die Lernrate ohnehin extern kontrolliert (z. B. nach einer vorgegebenen Strategie eingefroren) wird, kann β auf seinem einmal eingestellten Wert festgehalten werden. Der BP Algorithmus hat damit **einen einzustellenden Parameter weniger**.

Das ist keine Überraschung. Der Steilheitsparameters β multipiziert in der Transferfunktion

$$f\left(\vec{x}\right) = \frac{1}{1+e^{-\beta\vec{w}\vec{x}}}$$

direkt den Vektor der synaptischen Verbindungsstärken, eine Reskalierung $\beta\vec{w} \rightarrow \vec{w}$ eliminiert β aus der Transferfunktion. Allerdings ist die Lerngeschwindigkeit für die reskalierten w_{ij} anders: In der Lernregel 3.7 tritt β (wegen der für die sigmoide Ausgabefunktion $S = 1/\left(1+e^{-\beta z}\right)$ gültigen Regel $S'\left(z\right) = \beta S\left(1-S\right)$)

Steilheit und Lernrate

als multiplizierender Faktor auf. Durch die Reskalierung $\epsilon\beta \to \epsilon$ kann aberder Einfluß von β vollständig aus der Lernregel eliminiert werden, vgl. 3.8, 3.9.

3.6.2 Variable Schrittlänge

Die Lerngeschwindigkeit wird entscheidend von der Lernrate ϵ bestimmt, die die Schrittlänge im Gradientenverfahren

$$\Delta\vec{w} = -\epsilon \frac{\partial}{\partial\vec{w}} E\left(\{\vec{w}\}, \vec{x}\right) \tag{3.26}$$

festlegt. Wegen des Faktors $S'(z)$ in der backprop-Lernregel kann bei gegebenem ϵ die Schrittweite sehr klein ausfallen. Idee: ϵ in flachen Gebieten der Kostenlandschaft gößer wählen. Ausführung: Bestimme Gradient, damit gemäß (3.26) $\Delta\vec{w}$ und $\Delta E = E\left(\{\vec{w}'\}, \vec{x}\right) - E\left(\{\vec{w}\}, \vec{x}\right)$, wobei $\vec{w}' = \vec{w} + \Delta\vec{w}$. Falls $\Delta E < 0$: Bestimme **mit dem gleichen** $\Delta\vec{w}$ ein $\vec{w}'' = \vec{w}' + \Delta\vec{w}$ und damit $\Delta E = E\left(\{\vec{w}''\}, \vec{x}\right) - E\left(\{\vec{w}'\}, \vec{x}\right)$. (*gradient-reuse* Verfahren), usw. solange $\Delta E < 0$. Vergrößerung von ϵ falls das mehrfach erfolgreich ist, Verkleinerung bei Vorzeichenumkehr. Gute Ergebnisse bei kleinen Lernproblemen (Faktor 5). *gradient-reuse*

3.6.3 Backpropagation mit Massenterm.

Die Gradientenregel kann für kleine ϵ in eine Differentialgleichung umgeschrieben werden

$$\frac{d}{dt}\vec{w} = -\epsilon \frac{\partial}{\partial\vec{w}} E\left(\{\vec{w}\}, \vec{x}\right) = -\epsilon \nabla E\left(\{w\}, \vec{x}\right)$$

die die mechanische Bewegung eines masselosen Teilchens mit Ortsvektor $\vec{w}$ in einem viskosen Medium unter Einfluß der Kraft $\frac{\partial}{\partial\vec{w}} E\left(\{\vec{w}\}, \vec{x}\right)$ beschreibt. Falls das Teilchen eine Masse haben soll ist ein Term in $\frac{d^2}{dt^2}\vec{w}$ hinzuzufügen, das entspricht der Ersetzung von (3.26) durch

$$\Delta\vec{w}(t) = -\epsilon \frac{\partial}{\partial\vec{w}} E\left(\{\vec{w}(t)\}, \vec{x}\right) + \gamma\Delta\vec{w}(t-1) \tag{3.27}$$

Durch den zugefügten Massenträgheits– oder Schwungterm zeigt diese Regel ein besseres Verhalten beim Überwinden lokaler Minima und auch beim Überwinden von Plateaus. Wenn der Gradient nämlich sehr klein ist, dann ist im wesentlichen $\Delta\vec{w}(t)$ nur durch seinen Vorgängerwert $\Delta\vec{w}(t-1)$ bestimmt. Ist γ nahe 1, so wird $\Delta\vec{w}$ auf dem Plateau im wesentlichen durch seinen Eingangswert in das Plateau bestimmt. Massenträgheit

3.6.4 Quickprop

Ist ein Verfahren zweiter Ordnung. Entspricht der Newton Methode zur Nullstellensuche. Idee: Bestimme eine quadratische Approximation von E um den

aktuellen Gewichtsvektor $\vec{w}_t$ und Wahl des neuen Gewichtsvektors $\vec{w}_{t+1}$ als Minimum $\nabla E = 0$ dieser quadratischen Fläche, d. h. Taylorentwicklung

$$E\left(\{w\}\right) \approx E\left(\{w_t\}\right) + \left(\vec{w} - \vec{w}_t\right) \nabla E\left(\{w_t\}\right) + \frac{1}{2}\left(\vec{w} - \vec{w}_t\right) H\left(\vec{w}_t\right) \left(\vec{w} - \vec{w}_t\right)^T$$

mit H Hesse-Matrix

$$H_{i,j}\left(\vec{w}\right) = \frac{\partial^2}{\partial w_i \partial w_j} E\left(\{\vec{w}\}\right)$$

Damit

$$\nabla E\left(\{\vec{w}\}\right) = \nabla E\left(\{\vec{w}_t\}\right) + \left(\vec{w} - \vec{w}_t\right) H$$

und Wahl des neuen Gewichtsvektor (vermutetes Minimum) als

$$\vec{w}_{t+1} = \vec{w}_t - H^{-1} \nabla E\left(\{\vec{w}_t\}\right) \tag{3.28}$$

(Das Verfahren wird meist in batch-Mode (Mittelung von $E\left(\{\vec{w}\left(t\right)\}, \vec{x}\right)$ über mehrere $\vec{x}$ vor Bildung des Gradienten) ausgeführt, deshalb $\vec{x}$ in E unterdrückt). Schwierig: Bestimmung von H^{-1} und Speicherung nichtlokaler Information im Netz.

Quickprop **Eigentliches Quickprop:** Ebenfalls quadratische Approximation jedoch nur in Richtung des Gradienten und damit eindimensional! Für ein $w_{i,j}$ gilt

$$\Delta w_{i,j}\left(t\right) = \frac{G\left(t\right)}{G\left(t-1\right) - G\left(t\right)} \Delta w_{i,j}\left(t-1\right) \tag{3.29}$$

wobei $G\left(t\right)$ partielle Ableitung der Fehlerfunktion nach $w_{i,j}$.

Weitere Angaben in z. B. [4].

Literaturverzeichnis

[1] R. Der and U. Steinmetz. Scaling properties of the 'what' and 'where' conundrum (unpublished). Technical report, 1996.

[2] B. Efron. Bootstrap methods: Another look at the jacknife. *The Annals of Statistics*, 7:1–26, 1979.

[3] K. Hornik, Stinchcombe, and H. M., White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359 – 366, 1989.

[4] R. Rojas. *Theorie der neuronalen Netze.* Springer, 1994.

[5] J. G. Rueckl, K. R. Cave, and S. M. Kosslyn. Why are "what" and "where" processed by separate cortical visual systems? *Journal of Cognitive Neuroscience*, 1:171–186, 1989.

[6] I. G. Sprinkhuisen-Kuyper and E. J. W. Boers. The error surface of the simplest XOR network has only global minima. *Neural Computation*, 8:1301 – 1320, 1996.

[7] G. Thimm, P. Moerland, and E. Fiesler. The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation*, 8:451 – 460, 1996.

[8] J. Wiles. Using bottlenecks in feed—forward networks as a dimension reduction technique: An application to optimization tasks. *Neural Computation*, 8:1179—1183, 1996.

[9] A. Zell. *Simulation neuronaler Netze.* Addison–Wesley, 1994.