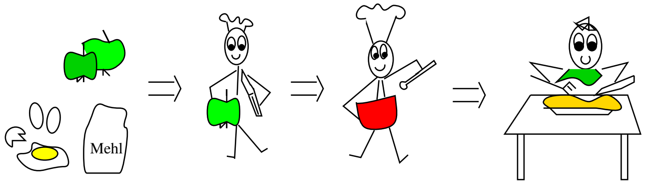
Neuronale Netze
Vorlesung im WS 99/00

Barbara Hammer

23. Juni 2000



"Feed-forward Netz"

Inhaltsverzeichnis

In eigener Sache

Dieses Skript ist am Entstehen, es werden stetig Kapitel hinzugefügt und Fehler verbessert. Für Kommentare und Hinweise auf Fehler bin ich jederzeit dankbar.

Die Vorlesung geht nicht linear mit dem Skript vor, sondern gibt erst einen Grobüberblick über die einzelnen Netzmodelle mit besonderem Wert auf den unmittelbar praktisch relevanten Bereichen, d.h. insbesondere Trainingsalgorithmen, in einem zweitem Durchlauf werden die teilweise eher theoretischen Fragestellungen und die aufwendigeren praktischen Verfahren behandelt. Wer den ersten Teil beherrscht, kann mit Neuro in der Praxis schon viel anfangen. Die weiteren Fragen sind natürlich für das Verständnis interessant, teilweise aber nicht ganz einfach, so daß ein grobes Verständnis nach dem ersten Hören ausreichend ist – zum Teil handelt es sich um sich gerade erst etablierende Gebiete oder auch neue Forschungsergebnisse. Der verzweifle also nicht, der just einige Sätze nicht in anderen Lehrbüchern findet, die kommen dann aus Papern oder stehen noch nirgends :-)

Die Auswahl des Stoffes lehnt sich prinzipiell soweit wie möglich an das im Neurobereich etablierte Repertoire an – allerdings gibt es dieses bisher nur in grundlegenden Teilbereichen. Ein Vergleich mit Lehrbüchern (siehe Semesterapparat) sei hier empfohlen.

Es wird so wenig wie möglich an mathematischen Methoden verwandt. Sollte dennoch der eine oder andere Begriff neu sein, so reicht eine intuitive Vorstellung. Da wir uns in gutartigen Gefilden bewegen, treten Spezialfälle, wo man mit intuitivem Verständnis schief liegt, nicht auf. Zum Trost: Durch seine Aktualität und das Einbeziehen vieler unterschiedlicher mathematischer und anderer Methoden wird Neuro interessant; man kommt sehr schnell an Stellen, wo man selbst forschen kann.

Noch eine Bitte: Es ist manchmal schwierig, einzuschätzen, ob der Stoff beim ersten Hören zu schwer ist. Da es aber niemandem etwas nützt, wenn Stoff zu schnell oder zu schwer präsentiert wird oder umgekehrt uninteressante Sachen zu sehr ausgewälzt werden, bitte ich in dem Fall um möglichst sofortige Rückmeldung, dann kann man's ändern! Danke!

1 Einleitung

1.1 Motivation

Viele Aufgaben sind mit exakten Methoden bisher nur unzureichend gelöst, die Möglichkeit der Automatisierung – obwohl erwünscht – ist nur partiell gegeben: Personenerkennung, Sprachverstehen, Aufräumen, Autofahren, . . . Sie haben gemein, daß eine mathematische Modellierung unmöglich oder aufwendig erscheint. Nichtsdestotrotz können die Aufgaben aber von Menschen zufriedenstellend gelöst werden – nur aufgrund von partieller expliziter Information, Erfahrung und Übung. Letztere beiden Begriffe könnte man auch als ‚Vorhandensein von Beispielen' bezeichnen. Neuronale Netze sind eine Methode, eine Gesetzmäßigkeit (Funktion, Verhaltensweise, . . .) nur mithilfe von Beispielen zu lernen. Sie sind dabei weder das einzig mögliche Verfahren in diesem Bereich, noch ein Allheilmittel – obwohl scheinbar ein universeller Ansatz. Bei jedem neuen Problem wird man den Hauptteil der Arbeit für die Problemrepräsentation, die konkrete Anpassung und das Feintunung verwenden. Jedoch in diesem Rahmen verhelfen neuronale Netze oft zu erstaunlichen Erfolgen, wenn (und nur wenn !) kein ausreichendes explizites Wissen vorhanden ist. Einige Beispiele für Anwendungen:

- Bildverarbeitung
  - Erkennung von handgeschriebenen Ziffern
  - Erkennen von Personen
  - Erkennen von Fehlstellen in Materialien
  - Krankheitsdiagnose anhand von Röntgenbildern
- Klassifikation/Prognose
  - Krankheitsverlaufsprognose anhand von Daten
  - Kreditwürdigkeitsprognose
  - Eigenschaften von Molekülen vorhersagen, z.B. Sekundärstruktur, Aktivitität in Bezug auf Medikamentierung,
  - Klassifikation von Bootstypen anhand von Unterwassergeräuschen
  - Minenerkennung
- Zeitreihenverarbeitung
  - Börsenkursprognose
  - Wettervorhersage
  - Spracherkennung/-erzeugung
- Robotik/Steuerung
  - Robotersteuerung
  - Steuerung von Maschinen
  - Fahren
- Datenaufbereitung

- Clustering
- Dimensionsreduktion
- Data Mining

- Varia
  - Spielstrategien
  - Komponieren

- ...

Eigenschaften sind, daß die zu modellierenden Prozesse/Gesetzmäßigkeiten nicht in einer exakten mathematischen Beschreibung vorliegen. Vorhandenes Wissen ist in der Regel in die Repräsentation der Daten integriert (z.B. Rotationsinvarianz, Inflationsrate, Wichtigkeit oder Unwichtigkeit einzelner Faktoren). Es gibt (viele) Beispiele für die Gesetzmäßigkeit. Neuronale Netze sind dort

- lernfähig,
- hochgradig parallel,
- fehlertolerant, bei wachsendem Fehler der Daten oder Ausfall von Komponenten sinkt die Leistung nur allmählich ab,
- arbeiten auf einer verteilten Darstellung der Information, daher ist die Verknüpfung mit symbolischen Komponenten schwierig, keine Introspektion möglich.

1.2 Das biologische Neuron

Das menschliche Gehirn besteht aus ca. 10^{11} Neuronen mit je ca. 1000 bis 10000 Synapsen (teilweise wesentlich mehr: Purkinje Zellen) ausgehend vom Axon und je 2000 bis 10000 synaptischen Verbindungen von anderen Neuronen zu den Dendriten der Zelle. Information wird mithilfe elektrischer Signale vermittelt. Im Ruhezustand hat die Zelle ein Potential von -70 mV gegenüber der Umgebung. Dieses berechnet sich wie folgt: Die Zellmembran ist permeabel für K^+, aber nicht für Cl^-. Dieses führt zu einer Diffusion von K^+ in die Umgebung, bis sich ein Gleichgewicht mit der durch die in der Zelle verbleibenden Cl^- Ionen erzeugten Spannung einstellt. Gleichzeitig sorgen Ionenpumpen in der Zelle für den Austausch von je drei Na^+ Ionen von innerhalb der Zelle gegen zwei K^+ Ionen von außerhalb der Zelle je Pumpvorgang, so daß im Ruhezustand die Konzentration an Na^+ außerhalb 12 mal höher, die Konzentration an K^+ innerhalb 40 mal höher ist.

Liegt positive Ladung (von anderen Zellen) an, so öffnen sich Natriumkanäle, die durch die einströmenden Ionen ein Erhöhen des Potentials auf 30 mV bewirken. Bei ca. 30 mV öffnen sich Kaliumkanäle, die Kaliumionen nach außen dringen lassen. Das Potential sinkt kurzeitig unter das Ruhepotential ab und gleicht sich dann wieder dem Ruhezustand an. Es ist ein Spike entstanden, ein kurzzeitiger Spannungsanstieg und Abfall.

Der Spike pflanzt sich über das Axon fort, indem die durch das Einströmen der Natriumionen verursachte positive Ladung das Öffnen weiterer spannungsgesteuerter Natriumkanäle bewirkt. Das Quadrat der Geschwindigkeit ist dabei proportional zum Durchmesser des Axons. Einige wichtige Nervenstränge weisen Axone mit Myelinhülle auf. Diese hat eine isolierende Wirkung, was zu einer Übertragungsgeschwindigkeit proportional zum Durchmesser führt: Ohne Myelinhülle nimmt die Dichte der Feldlinien ab, sofern man sich vom depolarisierten Bereich wegbewegt, Übertragung erfolgt durch das Öffnen benachbarter Natriumkanäle. Mit Myelinhülle, die in regelmäßigen Abständen Unterbrechungen aufweist, verlaufen die Feldlinien nahezu parallel vom polarisierten Bereich zur nächsten Unterbrechung der Hülle. Die Übertragung erfolgt durch das Öffnen von Kanälen an der nächsten Unterbrechung, durch das stärkere Feld kann die Distanz überwunden werden.

Über Nervenzellen hinweg pflanzt sich der Spike durch die Synapsen fort: Das Aktionspotential bewirkt an den Synapsen das Öffnen von Kalziumkanälen, die wiederum ein Verschmelzen der in der Synapse enthaltenen Bläschen mit der Membran bewirken, Neurotransmitter werden freigesetzt. Die Transmitter überwinden den synaptischen Spalt und beeinflussen chemisch gesteuerte Ionenkanäle an den anliegenden Neuronen. Neurotransmitter sind etwa Dopamin, Glutaminsäure, Noradrenalin; sie wirken hemmend oder verstärkend je nachdem, welche Art von Ionenkanälen sie beeinflussen.

Den genauen Natrium-/Kalium-/Kalzium-/Chlorfluß kann man lokal durch ein System von Differentialgleichungen, den sog. Hodgin-Huxley-Gleichungen beschreiben, die Fortsetzung auf räumliches Geschehen geschieht etwa durch geeignete Kompartimentmodelle.

Plastizität entsteht durch Verändern der Verbindungsstrukturen (man geht davon aus, daß zwar die Neuronen nach der Geburt komplett vorhanden sind, aber nicht die synaptischen Verbindungen) und durch Verändern der Stärke der Synapsenbindungen.

Es ist nicht klar, welches Element der elektrischen Signale Information trägt, denkbar wäre

- binäre Kodierung durch den Zustand des Neurons (fehleranfällig),
- Frequenzkodierung (langsam),
- Kodierung durch Koinzidenz (wer steuert diese?).

Verifizierbar ist dieses durch die notwendigen Eigenschaften, die so eine Kodierung besitzen muß, und nachgewiesene Effekte (z.B. synchron spikende Neuronen).

Im menschlichen Gehirn sind die Zellen in Regionen mit unterschiedlicher Funktionalität angeordnet. Den größten (und jüngsten) Teil bildet der Neocortex mit unterschiedlichen funktionellen Bereichen, etwa dem motorischen und somato-sensorischen Rindenfeld, welche je Regionen, die für einzelne Organe verantwortlich sind, aufweisen. Etwa das visuelle System ist relativ gut untersucht, man kann hier verschiedene Schichten von Neuronen ausmachen, die untereinander verbunden sind.

1.3 Historische Entwicklung

[43] McCulloch/Pitts: Schaltkreise mit Schwellwertelementen, binäre Gewichte. Sie zeigen Berechnungsuniversalität.

[49] Hebb: Lernparadigma des Hebbschen Lernens, Verstärkung von Verbindungen zwischen Neuronen gleichartiger Aktivierung, Abschwächung der übrigen Verbindungen.

[58] Rosenblatt: Das Rosenblatt-Perzeptron besteht aus einem einzelnen Perzeptron mit vorgelagerten Masken. Es findet Einsatz in der Mustererkennung. Die Gewichte werden mit einer Hebb-artigen Regel trainiert, Beweis der Konvergenz des Algorithmus.

[60] Widrow/Hoff: Adaline, ähnliches Prinzip. Widrow gründet die erste Neurocomputing-Firma, die Memistor-Corporation.

$\Rightarrow$ Neuronale Netze gelten als hinreichendes Modell für selbstlernende intelligente Systeme.

[69] Minsky/Papert: Sie zeigen die theoretische Beschränkung der existenten Modelle, einige einfach erscheinende Aufgaben sind nicht darstellbar.

⇒ Stop der Finanzierung von Forschung im Neuro-Bereich.

[71] Vapnik/Chervonenkis: Möglichkeit, aus empirischen Daten auf die Generalisierungsfähigkeit eines Systems zu schließen; dieses bildet bis heute die mathematische Grundlagen für sog. informationstheoretische Lernbarkeit.

[73] von der Malsburg: Schlägt einen Ansatz für unüberwachtes Lernen vor, der es erlaubt, die Ausbildung spezieller Areale des visuellen Cortex, die auf spezielle Orientierungen von Linien reagieren, zu erklären.

[74] Werbos: Schlägt in seiner Doktorarbeit das noch heute gebräuchliche Lernverfahren Backpropagation für überwachtes Lernen neuronaler Netze vor. Es findet allerdings keine Beachtung, und Werbos wurde lange Zeit nicht als Urheber des Verfahrens anerkannt.

[76] Grossberg: Schlägt (in unleserlichen Artikeln) verschiedene Modelle vor, wobei er sowohl überwachtes als auch unüberwachtes Lernen benutzt. Besonders interessiert ist er am sogenannten Stabilitäts-Plastizitäts-Dilemma, d.h. der Möglichkeit, sich an die Umwelt zu adaptieren und gleichzeitig Gelerntes beizubehalten. Bekannt ist er durch seine Adaptive Resonance Theory.

[77] Kohonen: Effiziente Algorithmen zu selbstorganisierendem Lernen, topologieerhaltende Abbildungen.

[82] Hopfield: Hopfieldnetze als Assoziativspeicher. Er entwickelt sowohl Lernregeln als auch die notwendige Hintergrundtheorie (mit Anleihen an physikalische Modelle).

[83] Kirkpatrick et.al.: Erweiterung des Hopfieldmodells um Simulated Annealing, so daß effizienter Betrieb möglich scheint.

[83] Fukushima: Das Neocognitron als funktionsfähiges Schriftzeichenerkennungssystem.

⇒ Wachsendes Interesse im Neurobereich.

[84] Valiant: Schlägt den Begriff PAC Lernbarkeit vor.

⇒ Formalismus, der die Fähigkeit eines Systems, effizient zu lernen, mathematisch beschreibt.

[85] Hopfield: Anwendung auf klassische Optimierungsaufgaben, z.B. TSP.

[86] Rumelhart et.al.: (Wieder-)Entdeckung von Backpropagation.

[86] Sejnowski/Rosenberg: NetTalk, ein extrem erfolgreiches Projekt zur Spracherzeugung; die Leistung ist vergleichbar mit dem bis dato besten symbolischen Ansatz.

⇒ Aufschwung im Neurobereich.

[89] Blumer et.al.: Verknüpfung von VC-Dimension und PAC Lernbarkeit.

⇒ Mathematische Fundierung von lernenden Systemen.

>> Es gibt eine Fülle von weiteren neuronalen Systemen, Anwendungen in verschiedensten Bereichen, Verknüpfung mit unterschiedlichsten Methoden, internationale Tagungen und Zeitschriften und eine etablierte Neuro-Society.

1.4 Definition

Definition 1.1 *Ein* **neuronales Netz** *ist ein Tupel* $\mathcal{N} = (N, \to, \mathbf{w}, \boldsymbol{\theta}, \mathbf{f}, I, O)$ *mit*

- $N = \{1, \ldots, n\}$ *den* **Neuronen**,
- $\to \subset N \times N$ *der Vernetzungsstruktur,*
- $\mathbf{w} = (w_{ij})_{i \to j}$ *(*$w_{ij} \in \mathbb{R}$*) den* **Gewichten**,
- $\boldsymbol{\theta} = (\theta_i)_{i \in N}$ *(*$\theta_i \in \mathbb{R}$*) den* **Schwellwerten** *oder* **Biases**.
- $\mathbf{f} = (f_i : \mathbb{R} \to \mathbb{R})_{i \in N}$ *den* **Aktivierungsfunktionen**,
- $I \subset N$ *den* **Eingabeneuronen**,
- $O \subset N$ *den* **Ausgabeneuronen**,

Eine **neuronale Architektur** *ist ein Tupel* $\mathcal{N}' = (N, \to, \mathbf{w}', \boldsymbol{\theta}', \mathbf{f}, I, O)$ *wie oben, wobei* $\mathbf{w}'$ *nur für einige (in der Regel keine) Verbindung* $i \to j$ *definiert ist und* $\boldsymbol{\theta}'$ *nur für einige (in der Regel kein) Neuron definiert ist.*

Die Idee ist, daß ein einzelnes Neuron i die entsprechend den Verbindungsgewichten gewichteten Signale durch die Vorgängerneuronen aufsummiert und je nach Größe der berechneten Zahl ein Signal weiterleitet, welches mithilfe der Aktivierungsfunktion berechnet wird. Im Falle, daß die Neuronen nur einen von zwei Zuständen besitzen (Spike oder nicht-Spike), und die Funktion f die Perzeptronaktivierung ist, die je nach Größe den Wert 0 oder 1 ausgibt, findet man direkt das biologische Verhalten unter Vernachlässigung von zeitlichen Aspekten wieder: Die an ein Neuron anliegenden Spikes werden je nach Stärke der Synapsenverbindung gewichtet aufsummiert und bei Überschreiten einer Schwelle ein Spike weitergeleitet. Die Verknüpfungsstruktur regelt das Gesamtverhalten und I und O die Verbindung zur Außenwelt.

Achtung: Diese Definition faßt zunächst nur die meisten gebräuchlichen neuronalen Netze für überwachtes Lernen, es werden weitere Objekte ein neuronales Netz genannt.

Definition 1.2 *Folgende* **Vernetzungsstrukturen** *treten auf:*

Bei einem **feedforward Netz** *ist* $(N, \to)$ *ein azyklischer Graph,* I *sind die Neuronen ohne Vorgänger,* O *sind die Neuronen ohne Nachfolger. Wir nehmen im Folgenden* $I \cap O = \emptyset$ *an. Man findet immer eine Zerlegung der Form* $N = N_0 \cup \ldots \cup N_h$ *mit* $N_i \cap N_j = \emptyset$ *für* $i \neq j$*,* $I = N_0$*,* $O = N_h$*,*

$$\to \subset \bigcup_{i=0}^{h-1} \bigcup_{j=i+1}^{h} N_i \times N_j.$$

Bezogen auf so eine Darstellung, heißt N_i *die ite* **hidden Schicht** *für* $0 < i < h$*,* N_0 *heißt* **Eingabeschicht**, N_h *heißt* **Ausgabeschicht**, h *heißt die* **Tiefe** *des Netzes. Verbindungen in* $N_i \times N_j$ *mit* $i < j - 1$ *heißen* **shortcut Verbindungen**.

Ein (voll vernetztes) **multilayer feedforward Netz** *liegt vor, falls zusätzlich*

$$\to = \bigcup_{i=1}^{h} N_{i-1} \times N_i$$

gilt. Die Vernetzungsstruktur wird durch den Ausdruck $(n_0, \ldots, n_h)$ *mit* $n_i = |N_i|$ *angegeben.*

Ein (voll vernetztes) multilayer feedforward Netz mit shortcut Verbindungen ist ein feedforward Netz mit

$$\to = \bigcup_{i=0}^{h-1} \bigcup_{j=i+1}^{h} N_i \times N_j.$$

Die Vernetzungsstruktur wird durch den Ausdruck $(n_0, \ldots, n_h)$*s mit* $n_i = |N_i|$ *angegeben.*

Diese Netztypen werden hauptsächlich zur Klassifikation oder Funktionsapproximation verwendet.

Ein (voll) **rekurrentes Netz** *liegt für*

$$\to = N \times N$$

vor. Häufig ist dann $I = O = N$ *oder* $I = O$*. Dieser Netztyp wird hauptsächlich als Assoziativspeicher oder für Optimierungsaufgaben verwendet.*

Bei einem **partiell rekurrenten Netz** *ist*

$$\to \subset N \times N$$

(aber i.A. weder azyklisch noch =). I *sind die Neuronen ohne Vorgänger. Gebraucht wird dieses zur Verarbeitung von Sequenzen oder zur Simulation dynamischer Systeme.*

Definition 1.3 *Einige* **Aktivierungsfunktionen***:*

Perzeptronaktivierung $\mathrm{H}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$

Bipolare Aktivierung $\mathrm{sgn}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

Identität $\mathrm{id}(x) = x$

Semilineare Funktion $\mathrm{lin}(x) = \begin{cases} 1 & x \geq 1 \\ x & 0 < x < 1 \\ 0 & x \leq 0 \end{cases}$

Sigmoide $\mathrm{sgd}(x) = 1/(1 + \mathrm{e}^{-x})$

Tangenshyperbolicus $\tanh(x) = (\mathrm{e}^{x} - \mathrm{e}^{-x})/(\mathrm{e}^{x} + \mathrm{e}^{-x})$

Eine **Squashing Funktion** *ist eine monoton wachsende Funktion* f *mit* $\lim_{x \to -\infty} f(x) = a$ *und* $\lim_{x \to \infty} f(x) = b$ *für Werte* $a < b$*.*

Häufig ist $f_i = f$ für alle Neuronen eines Netzes. In diesem Fall lassen wir die Indizes weg.

Definition 1.4 *Ein* **Zustand** *eines Netzes* $\mathcal{N}$ *ist ein Tupel* $\mathbf{o} = (o_i)_{i \in N}$ *mit* $o_i \in \mathbb{R}$*. Eine* **Aktivierung** *von* $\mathcal{N}$ *ist eine Folge von Zuständen* $\mathbf{o}(t)_{t \in \mathbb{N}}$*. Die* **Aktivierung** *des Neurons* i *zum Zeitpunkt* t *ist die Größe*

$$\mathrm{net}_i(t) = \sum_{j \to i} w_{ji} o_j(t) - \theta_i.$$

Es werden jetzt verschiedene Dynamiken definiert, die es erlauben, ausgehend von einem Startzustand $\mathbf{o}(0)$ *und weiteren Eingaben eine Aktivierung zu berechnen. Dieses beschreibt die Funktionsweise der Netze.*

Topologische Schaltdynamik bei feedforward Netzen:
Sei $N = N_0 \cup \ldots \cup N_h$ *eine Zerlegung. Für* $\mathbf{o}(0) \in \mathbb{R}^{|N|}$ *definiere*

$$o_i(t+1) = \begin{cases} o_i(t) & \text{falls } i \in N_j \text{ mit } j \leq t, \\ f_i(\mathrm{net}_i(t)) & \text{sonst.} \end{cases}$$

Das Netz $\mathcal{N}$ *berechnet die Funktion* $f : \mathbb{R}^{|I|} \to \mathbb{R}^{|O|}$, $f(\mathbf{x}) = (o_{i_j}(h))_{i_j \in O}$*, wobei* $o_{i_j}(0) = x_j$ *für* $i_j \in I$ *und* $o_{i_j}(0) = 0$ *sonst. Die Funktion* f *hängt nicht von der Zerlegung ab.*
Synchrone Schaltdynamik bei rekurrenten Netzen:

$$o_i(t+1) = f_i(\mathrm{net}_i(t))$$

für alle $i \in N$, $t \in \mathbb{N}$*.*
Asynchrone Schaltdynamik bei rekurrenten Netzen:

$$o_i(t+1) = \begin{cases} f_i(\mathrm{net}_i(t)) & i = i(t), \\ o_i(t) & \text{sonst} \end{cases}$$

wobei $i(t)$ *ein (zufällig ausgesuchtes) Neuron ist. In beiden Fällen startet man mit einem vorgegebenen Vektor* $\mathbf{o}(0) \in \mathbb{R}^{|N|}$*.*
Rekurrente Schaltdynamik bei partiell rekurrenten Netzen:
Sei $(\mathbb{R}^{|I|})^*$ *die Menge der endlichen Sequenzen mit Elementen in* $\mathbb{R}^{|I|}$*, notiert als* $[\mathbf{x}^0, \ldots, \mathbf{x}^T]$ *(*$T+1$ *ist die Länge der Sequenz). Für vorgegebenes* $\mathbf{y} \in \mathbb{R}^{|N|-|I|}$*, den sogenannten* **Startkontext***, und eine Sequenz der Länge* T *definiere*

$$o_i(t) = \begin{cases} x_j^t & i = i_j \in I, t \leq T-1, \\ y_j & i = i_j \notin I, t = 0, \\ f_i(\mathrm{net}_i(t-1)) & i \notin I, 0 < t \leq T, \\ o_i(t-1) & \text{sonst.} \end{cases}$$

Man kann eine Funktion $f_{\mathbf{y}} : (\mathbb{R}^{|I|})^* \to \mathbb{R}^{|O|}$ *definieren als* $f_{\mathbf{y}}([\mathbf{x}^0, \ldots, \mathbf{x}^T]) = (o_{i_j}(T+1))_{i_j \in O}$*.*

2 Das Perzeptron

2.1 Perzeptronalgorithmus

Wir fangen mal mit einem Neuron an, einem sogenannten **Perzeptron**, formal ein multilayer feedforward Netz $(n, 1)$ mit der Aktivierungsfunktion H. Die Gewichte der Ausgabe seien $w_1, \ldots, w_n$, der Bias θ. Es berechnet also

$$\mathbf{x} \mapsto \mathrm{H}(\sum_{i=1}^{n} w_i x_i - \theta) .$$

Eine **Lernaufgabe** für ein Perzeptron ist folgende Aufgabe: Gegeben sei eine **Trainingsmenge**

$$P = \{(\mathbf{x}^j; y^j) \in \mathbb{R}^n \times \{0, 1\} \mid j = 1, \ldots, m\} .$$

Die Aufgabe ist dann, Gewichte $\mathbf{w}$ und einen Bias θ zu finden, so daß $\mathrm{H}(\sum w_i x_i^j - \theta) = y^j$ gilt für alle j, d.h.

$$\sum w_i x_i^j \geq \theta \text{ falls } y^j = 1, \quad \sum w_i x_i^j < \theta \text{ falls } y^j = 0. \quad (*)$$

Die Punkte $\mathbf{x}^j$ mit $y^j = 1$ heißen **positive Punkte**, die anderen **negative Punkte**.
Elementare Reduktionen:

- Statt $\mathbf{x}^j$ kann man $(\mathbf{x}^j, 1) \in \mathbb{R}^{n+1}$ betrachten und so θ durch ein zusätzliches Gewicht $-w_{n+1}$ simulieren. Wir lassen also θ im Folgenden weg.
- Falls es Gewichte gibt, die $(*)$ erfüllen, so gibt es auch Gewichte, die $(*)$ mit $>$ statt $\geq$ erfüllen. Skalieren von $\mathbf{w}$ führt dazu, daß man sogar $|\sum w_i x_i^j| \geq 1$ verlangen kann. Wir betrachten also im Folgenden $(*)$ mit $>$ statt $\geq$ und ggfs. auch mit letzterer Skalierung.

Die Idee des Perzeptronalgorithmus ist Hebbsches Lernen. Die Gewichte werden sukzessive für falsch klassifizierte Beispiele so angepaßt, daß sie tendentiell korrekt sind. D.h. im Falle der Gewünschten Ausgabe 1 werden Verbindungen, wo positive Aktivierung anliegt, verstärkt, andere abgeschwächt. Entspechend bei gewünschter Aktivierung 0.

Definition 2.1 *Sei*

$$\delta(\mathbf{w}, \mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ positiv und } \sum w_i x_i \leq 0, \\ -1 & \mathbf{x} \text{ negativ und } \sum w_i x_i \geq 0, \\ 0 & \text{sonst}. \end{cases}$$

Dann ist der Algorithmus folgendermaßen:

$\mathbf{w} := \mathbf{w}_0$
Solange ein $\mathbf{x}$ *mit* $\delta(\mathbf{w}, \mathbf{x}) \neq 0$ *existiert*
    $\mathbf{w} := \mathbf{w} + \delta(\mathbf{w}, \mathbf{x}) \cdot \mathbf{x}$

Satz 2.2 *Sofern die Lernaufgabe lösbar ist, konvergiert der Perzeptronalgorithmus in endlich vielen Schritten.*

Beweis: Wir schreiben $\mathbf{w}^t\mathbf{x}$ für $\sum w_i x_i$, $|\mathbf{x}|$ für $\sqrt{\sum x_i^2}$. Sei $\mathbf{w}$ ein Lösungsvektor mit $|\mathbf{w}^t\mathbf{x}| \geq 1$. Sei $\mathbf{w}_k$ der Vektor im Perzeptronalgorithmus nach dem $k$ten Schritt. Sei $x_m = \max\{|\mathbf{x}| \mid \mathbf{x} \text{ ist Beispiel}\}$. Dann gilt

1. $\mathbf{w}^t\mathbf{w}_k \geq \mathbf{w}^t\mathbf{w}_0 + k$,
2. $|\mathbf{w}_k|^2 \leq |\mathbf{w}_0|^2 + kx_m^2$.

Induktion nach k: $k = 0$ ist ok. Die Rechnung

$$\begin{aligned} \mathbf{w}^t\mathbf{w}_{k+1} &= \mathbf{w}^t(\mathbf{w}_k + \delta(\mathbf{w}_k, \mathbf{x})\mathbf{x}) \\ &\geq \mathbf{w}^t\mathbf{w}_0 + k + \underbrace{\delta(\mathbf{w}_k, \mathbf{x})\mathbf{w}^t\mathbf{x}}_{\geq 1} \\ &\geq \mathbf{w}^t\mathbf{w}_0 + k + 1 \end{aligned}$$

zeigt 1. 2 folgt aus

$$\begin{aligned} |\mathbf{w}_{k+1}|^2 &= |\mathbf{w}_k + \delta(\mathbf{w}_k, \mathbf{x})\mathbf{x}|^2 \\ &= |\mathbf{w}_k|^2 + 2\,\underbrace{\delta(\mathbf{w}_k, \mathbf{x})\mathbf{w}_k^t\mathbf{x}}_{\leq 0} + |\mathbf{x}|^2 \\ &\leq |\mathbf{w}_0|^2 + kx_m^2 + x_m^2. \end{aligned}$$

Es gilt $|\mathbf{w}^t\mathbf{w}_k| \leq |\mathbf{w}| \cdot |\mathbf{w}_k|$ (Cauchy-Schwarzsche Ungleichung). Man erhält also

$$\mathbf{w}^t\mathbf{w}_0 + k \leq \mathbf{w}^t\mathbf{w}_k \leq |\mathbf{w}||\mathbf{w}_k| \leq |\mathbf{w}|\sqrt{|\mathbf{w}_0|^2 + kx_m^2}. \quad (**)$$

Das kann für große k nicht gelten. □

Wie sieht so eine Lösung aus? Ein Neuron definiert nichts anderes als eine Hyperebene (in $\mathbb{R}^2$ eine Gerade), wobei die Punkte auf der einen Seite nach 1 abgebildet werden, die Punkte auf der anderen Seite nach 0. Ist eine Lernaufgabe lösbar, dann heißen deswegen die Punkte auch **linear trennbar**. Wir haben gesehen, daß der Algorithmus für linear trennbare Punkte konvergiert. Allerdings gibt es auch nicht linear trennbare Mengen, so z.B. das XOR:

$$(0, 0; 0), (1, 0; 1), (0, 1; 1), (1, 1; 1)\,.$$

Kann man erkennen, wenn eine Trainingsmenge nicht linear trennbar ist?

Satz 2.3 *Gegeben eine Trainingsmenge, dann kann man eine Zahl K berechnen, so daß nach spätestens K Schritten der Perzeptronalgorithmus konvergieren muß, sofern die Menge linear trennbar ist.*

Beweis: Sofern in $(**)$ Gleichheit gilt, dann ist kein weiterer Schritt möglich. Dieses liefert einen Ausdruck für K, der die Größen $\mathbf{w}_0$, x_m und $\mathbf{w}$ benötigt. Für $\mathbf{w}_0 = 0$ etwa

$$K = |\mathbf{w}|^2|x_m|^2\,.$$

Könnten wir $|\mathbf{w}|$ beschränken, dann könnten wir auch K abschätzen. $\mathbf{w}$ ist irgendein Vektor mit $\mathbf{w}^t\mathbf{x}^i \geq 1$ für positive $\mathbf{x}^i$ und $\mathbf{w}^t\mathbf{x}^i \leq -1$ für negative $\mathbf{x}^i$. O.E. fällt letzteres weg. D.h.

$$X\mathbf{w} \geq \mathbf{1}$$

für eine Matrix X. Ist $X\mathbf{w} \geq \mathbf{1}$ lösbar, dann findet man maximal n Zeilen in der Matrix, so daß sich eine Lösung durch das Gleichungssystem $X'\mathbf{w} = \mathbf{1}$ (X' sind die n Zeilen von X) ergibt. Anschaulich bedeutet das, daß man im Lösungspolygon in die Ecken gehen kann.

[Idee: Starte mit einer Lösung. Ändere w_1, bis die erste Ungleichung zur Gleichung wird. Falls das nicht geht, kann w_1 beliebig gewählt werden. Sei $\mathbf{x}^{i_1}$ die zugehörige Zeile. Suche im Orthogonalraum von $\mathbf{x}^{i_1}$ eine neue Richtung, entlang derer die Gewichte geändert werden, bis die nächste Ungleichung zur Gleichung wird. Falls das nicht geht, ist diese Richtung frei wählbar. Sei $\mathbf{x}^{i_2}$ die zugehörige Zeile. Analog kann man im Schnitt der Orthogonalräume von $\mathbf{x}^{i_1}$, $\mathbf{x}^{i_2}$, … fortfahren. Sukzessive erhält man so $n-$dim (Lösungsraum) Gleichungen mit vollem Rang.]

D.h. aber, daß man die Koeffizienten von $|\mathbf{w}|$ durch eines von endlich vielen Gleichungssystemen, die durch die Punkte gegeben sind, erhält, daher kann man $\mathbf{w}$ abschätzen. □

Alternativ kann man das sog. **Perzeptron-Zyklus-Theorem** von Minsky und Papert benutzen, welches besagt, daß man die Länge der durch den Perzeptronalgorithmus erreichbaren Gewichtsvektoren in Abhängigkeit von der Trainingsmenge und dem Startvektor abschätzen kann. Lernt man nur mit ganzzahligen oder rationalen Mustern, so erhält man also im Falle einer nicht linear trennbaren Trainingsmenge einen Zyklus und kann dann stoppen.

Wie schnell konvergiert der Algorithmus bei einer lösbaren Trainingsaufgabe? Wir beschränken uns hier auf binäre Muster, d.h. Eingaben $\mathbf{x} \in \{0, 1\}^n$. Dann konvergiert der Algorithmus bei Start in $\mathbf{0}$ nach spätestens

$$(n + 1)^2(n + 1)^{n+1}$$

Schritten.

[Wir haben $K = |\mathbf{w}|^2|x_m|^2$ berechnet. Für Einträge mit 0 und 1 kann man aber $|\mathbf{w}|$ beschränken, indem man das zugehörige Gleichungssystem wie oben beschrieben löst, das führt über einige Rechnerei zu einer Gewichtsschranke $2^{n \log(n+1)}$.]

Umgekehrt benötigt aber die Funktion

$$(\mathbf{x},\mathbf{y}) \mapsto \begin{cases} 1 & \sum x_i 2^{i-1} \geq \sum y_i 2^{i-1} \\ 0 & \text{sonst} \end{cases}$$

Gewichte $\mathbf{w}$ mit $\sum |w_i| \geq 2^{n-1}$, folglich ein Gewicht der Größe $2^{n-1}/(n+1)$. (Der Beweis benutzt das sog. Diskriminatorlemma aus der Theorie Boolescher Schaltkreise.) Der Perzeptronalgorithmus braucht also mindestens $c \cdot 2^n$ Schritte (c ist eine positive Konstante) für diese Funktion, hat also exponentiellen Aufwand.

Nichtsdestotrotz ist er aufgrund seiner Einfachheit, Plausibilität und häufigen Schnelligkeit beachtenswert.

2.2 Alternativen

Ungleichungen der Form

$$A\mathbf{x} \leq \mathbf{b}$$

für eine feste Matrix A, einen festen Vektor $\mathbf{b}$ und Unbekannte $\mathbf{x}$ können mithilfe Methoden linearer Optimierung gelöst werden. 79 bewies Khachiyan, daß dieses Problem polynomiell lösbar ist. (Eine aus theoretischer Sichtweise bemerkenswerte Tatsache, da dasselbe Problem mit der Restriktion, daß die Lösungen $\mathbf{x}$ ganzzahlig sein müssen, NP-vollständig ist.) Allerdings ist sein Algorithmus in der Regel wesentlich langsamer als der gebräuchliche, aber evtl. exponentielle Simplex-Algorithmus. 84 schlug Karmakar einen Algorithmus mit der Laufzeit $\mathrm{O}(n^{3.5})$ vor, der insbesondere bei großen Instanzen besser als der Simplexalgorithmus ist.

[Idee: Überführe $A\mathbf{x} \leq \mathbf{b}$ zu einem Problem der Form:

$$\begin{aligned} \text{minimiere } & \mathbf{b}^t\mathbf{y} \\ \text{mit Bedingung } & A\mathbf{y} = \mathbf{c}, \\ & \mathbf{y} \geq \mathbf{0}, \end{aligned}$$

wobei man einen die Nebenbedingungen erfüllenden Vektor $\mathbf{y}$ kennt. Genauer: Man erhält die Form $A'\mathbf{x}' = \mathbf{b}'$, $\mathbf{x}' \geq \mathbf{0}$ durch $A\mathbf{x} - A\tilde{\mathbf{x}} + \mathbf{y} = \mathbf{b}$, $\mathbf{x} - \tilde{\mathbf{x}} = 0$, $\mathbf{x} \geq \mathbf{0}$, $\tilde{\mathbf{x}} \geq \mathbf{0}$, $\mathbf{y} \geq \mathbf{0}$. O.E. ist $\mathbf{b}' \geq \mathbf{0}$, sonst ersetze durch $A'\mathbf{x}' + \mathbf{z} = \tilde{\mathbf{b}}$, $\mathbf{z} = \tilde{\mathbf{b}} - \mathbf{b}'$, $\mathbf{z} \geq \mathbf{0}$ für genügend großes $\tilde{\mathbf{b}}$. Dieses wird überführt zum Problem: minimiere $\sum y_i$ mit $A'\mathbf{x}' + \mathbf{y}' = \mathbf{b}'$, $\mathbf{y}' \geq \mathbf{0}$, $\mathbf{x}'' \geq \mathbf{0}$. Das Minimum ist 0 genau dann, wenn es für das vorherige Problem eine Lösung gab. Für dieses Problem kennt man einen Punkt im durch die Nebenbedingungen definierten Polygon: $\mathbf{x}' = \mathbf{0}$, $\mathbf{y}' = \mathbf{b}'$.

Das Problem wird approximiert durch

$$\begin{aligned} \text{minimiere } & \mathbf{b}^t\mathbf{y} - \mu \sum \log y_j \\ \text{mit Bedingung } & A\mathbf{y} = \mathbf{c}, \end{aligned}$$

für $\mu > 0$. Für $\mathbf{y} \to \mathbf{0}$ wird der Wert groß, so daß man die Ungleichung für $\mathbf{y}$ quasi integriert hat. Obiges System kann man mit üblichen Methoden der Analysis angehen. Lagrangemultiplikatoren führen zu einem näherungsweise lösbaren Gleichungssystem und einem neuen (besseren) Wert für $\mathbf{y}$. Dieses Vorgehen wird jetzt geeignet für $\mu \to 0$ iteriert und liefert einen Pfad von Punkten $\mathbf{y}$, die gegen das Optimum konvergieren.]

2.3 Exkurs in die Komplexitätstheorie

Gegeben ist ein Entscheidungsproblem; d.h., gegeben eine geeignet repräsentierte Instanz eines Problems x, soll entschieden werden werden, ob für x ein Sachverhalt zutrifft. Für eine Instanz x sei $|x|$ die Länge der Repräsentation.

Definition 2.4 *Ein Problem heißt* **polynomiell lösbar**, *falls es ein Programm und ein Polynom p gibt, das, gegeben eine Instanz x, nach spätestens $p(|x|)$ Schritten entscheidet, ob der nachzuprüfende Sachverhalt zutrifft. Die Klasse der polynomiell lösbaren Probleme bezeichnen wir mit* **P**.

Beispiele sind:

- Test, ob eine Liste sortiert ist,
- Test, ob das Produkt von x und y die Zahl z ergibt,
- Test, ob eine Trainingsmenge linear separierbar ist.

Nur polynomiell lösbare Probleme sind auch für große Instanzen effizient lösbar. [Stimmt nicht so ganz, die heute als praktikabel anerkannte Klasse ist nicht P, sondern RP, die Klasse der mit einem nichtdeterministischen Algorithmus in polynomieller Zeit und **mit großer Wahrscheinlichkeit** korrekt lösbaren Probleme. Es wird aber ebenso NP≠RP vermutet.]

Definition 2.5 *Ein Problem ist* **nichtdeterministisch polynomiell lösbar**, *falls es ein Programm und Polynome p_1, p_2 gibt mit folgender Eigenschaft: Das Programm läuft mit durch p_1 beschränkter Laufzeit. Auf eine Instanz x trifft die zu testende Eigenschaft zu genau dann, wenn es eine Hilfe y mit $p_2(|x|) \geq |y|$ gibt, so daß das Programm die Eingabe (y, x) mit ‚ja' bescheidet. Die Klasse der nichtdeterministisch polynomiell lösbaren Probleme bezeichnen wir mit* **NP**.

Ein Problem heißt **NP-vollständig**, *falls es in NP liegt, aber die polynomielle Lösbarkeit des Problems die polynomielle Lösbarkeit jedes anderen Problems in NP implizieren würde.*

Die Idee bei NP ist, daß man zwar mit dem Problem selbst nicht viel anfangen kann. Ist allerdings ein guter Freund zur Hand, der uns einen Tipp gibt, dann können wir damit etwas anfangen und die nachzuweisene Eigenschaft testen.

Das Bemerkenswerte ist, daß es tatsächlich sogar eine ganze Latte von NP-vollständigen Problemen gibt! Falls ein Problem NP-vollständig ist, wird vermutet, daß es nicht effizient lösbar ist. Genauer: Für keines der NP-vollständigen Probleme wurde bis dato ein polynomieller Lösungsalgorithmus gefunden [auch kein Algorithmus aus RP].
Einige Beispiele für NP-vollständige Probleme:

- SAT: Gegeben eine Boolesche Formel in konjunktiver Normalform

  $$\varphi = \bigwedge_i \bigvee_j L_{ij},$$

  wobei L_{ij} eine Boolesche Variable oder eine negierte Boolesche Variable ist, gibt es eine erfüllende Belegung? Cook wies das als NP-vollständig nach, sogar im Fall $j \in \{1, 2, 3\}$.

  Das ist in NP: Wir können zwar bei einer Formel (x) die Erfüllbarkeit nur testen, indem wir alle (exponentiell vielen) Belegungen durchprobieren. Verrät uns aber jemand eine erfüllende Belegung (y), dann können wir sehr schnell sehen, ob sie stimmt. Pech haben wir, wenn uns jemand falsch geraten hat.

- TSP: Gegeben Städte, positive Verbindungsdistanzen zwischen den Städten und eine Zahl K, gibt es eine Rundreise mit der Länge maximal K?

- Hitting set: Gegeben Punkte S, Teilmengen C und eine Zahl k, gibt es eine Menge c von k Punkten, so daß $c \cap c_i \neq \emptyset$ für alle $c_i \in C$ gilt? Anschaulich: Kann man verschiedene Interessengruppen mit maximal k Vertretern abdecken?

  [Reduktion von SAT: Eine Reduktion ist ein polynomieller Algorithmus f von Instanzen φ von SAT zu Instanzen $f(\varphi)$ des hitting set Problems, so daß φ lösbar ist, wenn und nur wenn $f(\varphi)$ lösbar ist. Könnte man jetzt das hitting set Problem effizient lösen, so auch SAT und damit alle anderen Probleme in NP.

  Wir reduzieren:
  $$f : \varphi = \bigwedge \varphi_i \mapsto (S, C, k)$$
  mit $k =$ Anzahl der Variablen in φ,
  $$S = \{s_1, \dots, s_k, \bar{s}_1, \dots, \bar{s}_k\},$$
  $$C = \{\{s_i \mid X_i \text{ kommt in } \varphi_j \text{ vor }\} \cup \{\bar{s}_i \mid \neg X_i \text{ kommt in } \varphi_j \text{ vor }\} \mid j\} \cup \{\{s_i, \bar{s}_i\} \mid i\}\ .$$

  Sei φ erfüllbar. Für eine erfüllende Belegung definiere die Menge $c = \{s_i \mid X_i \text{ ist wahr}\} \cup \{\bar{s}_i \mid \neg X_i \text{ ist wahr}\}$. Dadurch wird jedes $c_i \in C$ getroffen.

  Sei umgekehrt eine Menge c von k Punkten gegeben, so daß jedes $c_i \in C$ getroffen wird. Definiere eine Belegung durch X_i wahr $\iff$ $s_i \in c$. Wegen der Mengen $\{s_i, \bar{s}_i\}$ ist für X_i falsch der Punkt $\bar{s}_i \in c$. D.h. aber, daß die Punkte in c genau den erfüllten Literalen entsprechen und also jede Formel φ erfüllt ist, da jede der Formel entsprechende Menge von Punkten durch mindestens einen Vertreter in c abgedeckt ist.]

- 2-SSP: Gegeben Punkte S, Teilmengen C, existiert eine disjunkte Zerlegung $S = S_1 \cup S_2$, so daß $c \not\subseteq S_i$ für alle $c \in C$, $i \in \{1, 2\}$ gilt? Anschaulich: Teile eine Schulklasse in zwei Gruppen ein, so daß die Leute, die zusammen Quatsch machen, getrennt werden.

  [Reduktion von SAT, überführe $\varphi = \wedge\varphi_i$ in $S = \{s_1, \dots, s_n, \bar{s}_1, \dots, \bar{s}_n, p\}$, $n =$ Anzahl Variablen in φ, $C = \{\{s_i, \bar{s}_i \mid i\} \cup \{\{s_i, \bar{s}_j, p \mid X_i \in \varphi_k, \neg X_j \in \varphi_k\} \mid k\}$. Erfüllende Belegungen entsprechen Zerlegungen $S_1 = \{s_i \mid X_i \text{ ist wahr}\} \cup \{\bar{s}_i \mid X_i \text{ ist falsch}\}$ und $S_2 = \{p\} \cup \{s_1, \dots, \bar{s}_n\} \backslash S_1$.]

- k-SSP: Gegeben Punkte S, Teilmengen C, existiert eine disjunkte Zerlegung $S = S_1 \cup \ldots \cup S_k$ mit $c \not\subseteq S_i$ für alle $c \in C$, $i \in \{1, \dots, k\}$? Das gilt sogar für $|c| \leq 3$ für alle $c \in C$.

  [Reduktion von 2-SSP, auch bei letzterem kann $|c| \leq 3$ angenommen werden.]

- Separability in the plane: Gegeben Punkte $S \subset \mathbb{R}^2$, Zerlegung von S in P und Q, Zahl k, gibt es k Linien, so daß für alle $p \in P$ und $q \in Q$ eine Linie zwischen p und q verläuft?

  [Bewiesen durch Megiddo.]

2.4 Das Perzeptron im nicht linear trennbaren Fall

Man kann immer noch versuchen, eine möglichst gute Lösung zu finden, sofern man mit einer nicht linear trennbaren Menge konfrontiert ist. Um möglichst gute Ergebnisse zu erzielen, wird der Perzeptronalgorithmus wie folgt modifiziert:

Definition 2.6 *Gegeben eine Patternmenge P, so hat der* **Pocket-Algorithmus** *folgende Form:*


```
w := 0; w* := w; l := 0; l* := 0;, P_0 := P;
WHILE (P_0 ≠ ∅ und ich habe noch Geduld ) DO
    Wähle x ∈ P_0;   (*)
    IF δ(w, x) = 0
        l := l + 1; P_0 := P_0\{x};
        IF l > l* THEN
            w* := w; l* := l;
        END;
    ELSE
        w := w + δ(w, x)x; l := 0; P_0 := P;
    END;
END;
```

Satz 2.7 *Es gilt das* **Pocket-Konvergenz-Theorem***: Seien die Punkte rational. Für alle $\epsilon > 0$ gibt es ein T_0, so daß für alle $T \geq T_0$ gilt: Wählt man in $(*)$ $\mathbf{x}$ so aus, daß jeder Punkt mit positiver Wahrscheinlichkeit ausgesucht wird, dann ist die Wahrscheinlichkeit, nach T Schleifendurchläufen einen maximal möglichen Wert l^* erreicht zu haben, $\geq 1 - \epsilon$.*

Beweis: Betrachte eine optimale Hyperebene und die Punkte P', die dadurch richtig klassifiziert werden. Egal von welchem Vektor man startet, es gibt eine Folge von je falsch klassifizierten Punkten in P', die zu einem optimalen Vektor führt. Die Wahrscheinlichkeit, genau diese Folge aus den Punkten P zu ziehen, ist n.V. > 0. Nehme jetzt an, l' sei die maximal erreichte Menge an korrekt klassifizierten Punkten und diese sei nicht optimal. Nach dem Zyklustheorem ist die Länge der Gewichtsvektoren, die erreicht werden können, beschränkt. Bei rationalen Mustern gibt es also nur endlich verschiedene erreichbare Gesichtsvektoren. Für jeden von diesen gibt es eine Folge, die mit positiver Wahrscheinlichkeit gezogen wird und zu einem optimalen Vektor führt, d.h. zu jedem Zeitpunkt kann man mit positiver Wahrscheinlichkeit eine optimale Patternfolge folgen. Bei beliebiger Zeitdauer wird also mit Wahrscheinlichkeit 1 eine zum Erfolg führende Folge gezogen. Man kann daher einen Zeitpunkt bestimmen, nach dem mit Wahrscheinlichkeit $\geq 1 - \epsilon$ ein Optimum erreicht ist. □

Allerdings ist es so, daß bei anderer, z.B. zyklischer Reihenfolge nicht notwendig ein Optimum erreicht wird. Wann erhält man so ein Optimum mit hoher Wahrscheinlichkeit? Die aus dem Beweis resultierenden Schranken für die Zeitdauer sind astronomisch. Ist das nötig? Wahrscheinlich ja, oder genauer:

Satz 2.8 *Betrachte folgendes Problem: $k \in \mathbb{N}$, $n \in \mathbb{N}$, eine Patternmenge P in $\{0,1\}^n \times \{0,1\}$ seien gegeben. Gibt es ein Perzeptron, das auf P höchstens k Fehler macht? (k, n und P sind variabel.) Dieses Problem ist NP-vollständig.*

Beweis: In NP ist klar, denn man kann Gewichte raten (deren Darstellung ist durch $O(n \log n)$ beschränkt) und testen, ob sie zu maximal k Fehlern führen.

Das Problem ist auch NP-vollständig: Dieses wird durch eine Reduktion vom hitting set Problem gezeigt. Sei
$$(S = \{s_1, \dots, s_n\}, C = \{c_1, \dots, c_m\}, k)$$
eine Instanz vom hitting set Problem. O.E. $|c_1| = \ldots = |c_m| = l$. (Ansonsten vergrößere c_i um neue Elemente.) Definiere die Eingabedimension $n' = nl$ und die Trainingsmenge mit den Punkten
$$\begin{array}{lcllllll} \mathbf{p}_i & = & (\mathrm{e}_i, & \mathrm{e}_i, & \ldots, & \mathrm{e}_i, & \mathrm{e}_i; & 1), \quad i = 1, \dots, n, \\ \mathbf{p}^1_{c_j} & = & (\mathrm{e}_{i_1,\dots,i_l}, & 0^l, & \ldots, & 0^l, & 0^l; & 0), \\ \mathbf{p}^l_{c_j} & = & (0^l, & 0^l, & \ldots, & 0^l, & \mathrm{e}_{i_1,\dots,i_l}; & 0), \quad j = 1, \dots, m, \end{array}$$

wobei $e_i \in \mathbb{R}^n$ der $i$te Einheitsvektor ist und $e_{i_1,\ldots,i_l}$ der Vektor mit 1 an den Positionen $i_1, \ldots, i_l$ und 0 sonst für $c_j = \{s_{i_1}, \ldots, s_{i_l}\}$. Diese Menge kann mit maximal k Fehlern getrennt werden dann und nur dann, wenn es ein hitting set der Größe k gibt:

Es gebe ein hitting set c der Größe k. Definiere für die Gewichte $(w_{ij})_{i=1,\ldots,l,j=1,\ldots,n}$

$$w_{ij} = \begin{cases} 0 & s_j \notin c \\ -1 & s_j \in c \end{cases}$$

und $\theta = 0$. Das bildet genau die Punkte $\mathbf{p}_i$ mit $s_i \in c$ falsch ab.

Sei umgekehrt eine Lösung mit maximal k Fehlern gegeben. Definiere ein hitting set c wie folgt: Falls $\mathbf{p}_i$ falsch ist, ist $s_i \in c$. Falls ein $\mathbf{p}^h_{c_j}$ falsch, aber alle $\mathbf{p}_i$ für Punkte s_i in c_j richtig sind, ist ein beliebiger Punkt aus c_j in c. Nehme an, ein c_j sei nicht von c getroffen. Dann bekäme man für alle $s_i \in c_j$:

$$\sum_{h=1}^{l} w_{hi} \geq \theta \Rightarrow \sum_{h=1}^{l} \sum_{s_i \in c_j} w_{hi} \geq l\theta$$

und für c_j selber für $h \in \{1, \ldots, l\}$:

$$\sum_{s_i \in c_j} w_{hj} < \theta \Rightarrow \sum_{h=1}^{l} \sum_{s_i \in c_j} w_{hi} < l\theta\,.$$

Widerspruch. □

Obige Reduktion ist sogar kostenerhaltend, d.h. die Größe eines hitting set und die Anzahl der Fehlklassifikationen entsprechen sich genau. Es ist für hitting set sogar schwierig, eine Lösung zu finden, die nur maximal c mal so schlecht wie das Optimum ist (c eine positive Konstante). Dieses transferiert sich auf das Trainingsproblem: Auch nur Lösungen zu finden, die maximal c mal mehr Fehler als optimal machen, ist schwierig.

2.5 Das Rosenblatt-Perzeptron

Um die Mächtigkeit eines Perzeptrons zu erhöhen, sind verschiedene Variationen denkbar. Das Rosenblatt Perzeptron versucht, lineare Trennbarkeit zu erreichen, indem die Daten durch geeignete, aber feste Funktionen vorverarbeitet werden. Konstruiert ist es für Bilddaten, d.h. wir nehmen eine Eingabe aus $\mathbb{R}^{n\times m}$ an. Die Funktionen sollen – dem visuellen System des Menschen ähnlich – lokale Operationen vornehmen, etwa Rauschen unterdrücken, Kanten extrahieren, spezielle lokale Muster erkennen, ... Hinter die so vorverarbeitetn Daten wird ein wie üblich trainierbares Perzeptron geschaltet.

Definition 2.9 *Ein* **Rosenblatt Perzeptron** *berechnet eine Funktion $f : \mathbb{R}^{n\times m} \to \{0,1\}$, die sich durch Verknüpfung eines Perzeptrons p mit festen Funktionen $f_i : \mathbb{R}^{n\times m} \to \{0,1\}$ ergibt, d.h.*

$$f = p(f_1(\mathbf{x}), \ldots, f_M(\mathbf{x}))\,.$$

Die Funktionen f_i heißen **Masken***. Die Maske f_i hat die* **Ordnung** *k, falls f_i nur von k Koeffizienten der Eingabe abhängt. Die Maske f_i hat den* **Durchmesser** *k, falls f_i nur von Koeffizienten in einem Quadrat der Kantenlänge k abhängt.*

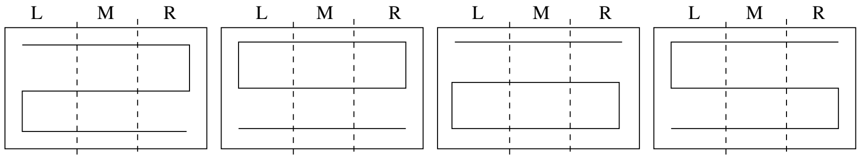
Offensichtlich könnten bei unbeschränkten Masken alle wesentlichen Operationen einfach durch die Masken vorgenommen werden, so daß alles verarbeitet werden kann. Sinnvolle Operationen

sind jedoch begrenzte, lokale Operationen, die durchaus vom Bildmaterial abhängig sein können. Auf einer Auktion von Gemälden sind etwa Masken sinnvoll, die den Schriftzug des Malers erkennen können – dieser ist für den Preis entscheidend. Um einen Text erkennen zu können, sind Masken (für Prototypen) für die einzelnen Buchstaben sinnvoll. Tatsächlich ist es eine gebräuchliche Methode in der Bildverarbeitung, im original Bildmaterial zunächst lokale Merkmale zu extrahieren, etwa Kantendetektoren, mittlerer Grauwert, ... und auf dieses vorverarbeitete Material zu trainieren.

Jedoch das Rosenblatt Perzeptron hat mit beschränkten Masken nur eingeschränkte Trennfähigkeit, so daß es nicht als universeller Mechanismus eingesetzt werden kann. Konkret betrachten wir das Problem, zusammenhängende Muster zu erkennen: Sei Z die Aufgabe, zusammenhängende Muster in $\{0,1\}^{n\times m}$ nach 1 und unzusammenhängende Muster nach 0 abzubilden. Dabei heißt ein Muster $\mathbf{x}$ zusammenhängend, sofern je zwei Punkte x_{ij} und x_{kl} mit Wert 1 über einen Pfad von Punkten mit Wert 1 verbunden werden können.

Satz 2.10 *Ein Rosenblatt Perzeptron mit Masken vom Durchmesser m ($m \geq 5$) kann auf Eingaben aus $\{0,1\}^{n\times m}$ für $n \geq 3m$ das Problem Z nicht lösen.*

Beweis: Betrachte die folgenden Muster.



Es gibt Masken, die auch auf den Bereich L bzw. auch auf den Bereich R, aber nicht auf beide zugreifen, und Masken, die nur auf den mittleren Bereich zugreifen. Von diesen drei Gruppen von Masken erhalten wir also je einen Beitrag für die Aktivierung des Perzeptrons. Dieser wird mit (l_i, m_i, r_i) bezeichnet, wobei i das Muster und der Buchstabe die jeweilige Region bezeichnet. Da die Muster in den entsprechenden Bereichen teilweise gleich sind, erhält man

$$\begin{array}{l} m_1 = m_2 = m_3 = m_4, \\ l_1 = l_3, r_1 = r_2, l_2 = l_4, r_3 = r_4\,. \end{array}$$

Wären die Muster korrekt, erhielte man also folgende Ungleichungen, wobei θ den Bias des Neurons darstellt:

$$\begin{array}{l} l_1 + m_1 + r_1 \geq \theta \\ l_2 + m_1 + r_1 < \theta \\ l_1 + m_1 + r_3 < \theta \\ l_2 + m_1 + r_3 \geq \theta \end{array}$$

$$\Rightarrow l_1 + l_2 + 2m_1 + r_1 + r_3 < 2\theta \leq l_1 + l_2 + 2m_1 + r_1 + r_3$$

Widerspruch. □

Minsky und Papert betrachten spezielle Masken folgender Form:

$$f_i(\mathbf{x}) = \begin{cases} 1 & x_{jk} = 1 \quad \forall jk \in A \\ 0 & \text{sonst} \end{cases}$$

wobei A eine ausgezeichnete Menge von Koeffizienten ist. D.h. Masken dieser Form testen, ob an mindestens den durch A spezifizierten Stellen eine 1 steht. Minsky und Papert zeigen, daß folgendes Problem, sofern es mit einem Perzeptron mit Masken obiger Bauart implementiert werden soll, mindestens eine Maske benötigt, die auf den ganzen Eingaberaum zugreift:

$$f(\mathbf{x}) = \begin{cases} 1 & |\{(j,k) \mid x_{jk} = 1\}| \text{ ist gerade} \\ 0 & \text{sonst} \end{cases}$$

Es kann also nicht mit lokalen Vorverarbeitungen zu einem linear trennbaren Problem transformiert werden.

[Dazu nutzen sie das sog. **Gruppeninvarianztheorem**, welches besagt, daß ein Problem, das gegenüber einer Gruppe G von Transformationen (etwa Rotation, Translation, $\ldots$) invariant ist und mithilfe von Masken dargestellt werden kann, so daß die Menge der zulässigen Masken gegenüber G abgeschlossen ist, dann auch eine Darstellung besitzt, so daß alle durch G ineinander überführbaren Masken dieselbe Gewichtung besitzen.

Daher sind in obigem Problem o.E. die Gewichte aller Masken zu einer Menge A mit demselben $|A|$ gleich, denn wir können als G die Gruppe aller Permutationen der Indizes betrachten. Es sei ein Pattern mit M Stellen 1 gegeben. Für dieses liefern genau die Masken mit festem $|A| = j$ eine 1, bei denen A in den M Stellen enthalten ist, d.h. man erhält von $\binom{M}{j}$ solchen Masken eine Rückgabe. Daher findet man die Aktivierung

$$\sum \alpha_{j=1}^{k} \binom{M}{j}$$

für alle Muster mit M Koeffizienten 1. k ist die maximale Maskengröße. Das ist ein Polynom vom Grad j in M. Betrachte die Funktion f. Die Pattern mit M Werten 1 werden abwechselnd für wachsendes M nach $0, 1, 0, \ldots$ abgebildet. D.h. die Aktivierung muß für wachsendes M das Vorzeichen $n \times m$ Mal wechseln. Daher muß es mindestens eine Maske geben, die auf alle Punkte zugreift.]

Vom Rosenblatt Perzeptron übriggeblieben ist in der modernen Bildverarbeitung immer noch das Verfahren, zunächst lokale Merkmale zu extrahieren, die dann mit – wie wir gesehen haben notwendig mächtigeren – Klassifikatoren weiterverarbeitet werden können.

2.6 Konstruktive Verfahren

Alternativ kann man für diese komplexeren Probleme Perzeptronnetze einsetzen – nur, wie soll man diese trainieren? Basierend auf dem Perzeptronalgorithmus gibt es verschiedene Verfahren, die jeweils nur ein Perzeptron trainieren und geeignet mit einem bestehenden Netz zusammensetzen, so daß sukzessive ein mächtigerer Klassifikator entsteht. Sei $P_0 \subset \mathbb{R}^n \times \{0,1\}$ eine Trainingsmenge. P_0 sei nicht widersprüchlich, d.h. enthalte keine Werte $(\mathbf{x}; y_1)$ und $(\mathbf{x}; y_2)$ mit $y_1 \neq y_2$.

Definition 2.11 Tower-Algorithmus:

> $P := P_0$; $f := (\mathbf{x} \mapsto \mathbf{0})$;
> *Wiederhole, solange f die Menge P noch falsch klassifiziert:*
>   *Trainiere ein Perzeptron p auf P.*
> $f := (\mathbf{x} \mapsto p(\mathbf{x}, f(\mathbf{x})))$;
> $P := \{(\mathbf{x}, f(\mathbf{x})); y) \mid (\mathbf{x}; y) \in P_0\}$;

Nach I Schleifendurchläufen besteht das fertige Netz aus I Perzeptronen neben den Eingabeneuronen, die in einem Turm angeordnet sind, d.h. es gibt Verbindungen von Neuron i zu Neuron $i+1$ für alle i, die berechnete Funktion hat die Form

$$p_I(\mathbf{x}, p_{I-1}(\mathbf{x}, \ldots, (\mathbf{x}, p_1(\mathbf{x})) \ldots)) .$$

Satz 2.12 *Es gibt einen Trainingsverlauf, so daß der Tower-Algorithmus nach Zufügen von maximal $|P_0|$ Neuronen hält.*

Beweis: Ordne die Muster $\mathbf{x}_1, \ldots, \mathbf{x}_m$, so daß $|\mathbf{x}_1| \leq |\mathbf{x}_2| \leq \ldots$ gilt. Sind $\mathbf{x}_1, \ldots, \mathbf{x}_i$ durch die bisher berechnete Funktion f korrekt klassifiziert, aber $\mathbf{x}_{i+1}$ noch falsch, dann können durch Hinzufügen eines weiteren Neurons $\mathbf{x}_1, \ldots, \mathbf{x}_{i+1}$ korrekt klassifiziert werden. Daher gibt es auch eine geeignete Auswahl der Muster, die genau diesen Trainingsverlauf bewirkt.

1.Fall: $\mathbf{x}_{i+1}$ ist positiv. Definiere für das Neuron die Gewichte $(w_1, \ldots, w_n) = \mathbf{x}_{i+1}$, $w_{n+1} = 2|\mathbf{x}_{i+1}|^2$ und den Bias $|\mathbf{x}_{i+1}|^2$. Für $\mathbf{x}_{i+1}$ berechnet sich die Aktivierung $2|\mathbf{x}_{i+1}|^2 \cdot f(\mathbf{x}_{i+1}) = 0$. Für andere Punkte $\mathbf{x}_j$, $j \leq i$ ergibt sich der Wert

$$2|\mathbf{x}_{i+1}|^2 \cdot f(\mathbf{x}_j) + \mathbf{x}_{i+1}^t \mathbf{x}_j - |\mathbf{x}_{i+1}|^2 .$$

Da $|\mathbf{x}_{i+1}^t \mathbf{x}_j| < |\mathbf{x}_{i+1}|^2$ ist, ist dieses genau für $f(\mathbf{x}_j) = 1$ nicht negativ.

2.Fall: $\mathbf{x}_{i+1}$ ist negativ. Definiere für das Neuron die Gewichte $(w_1, \ldots, w_n) = -\mathbf{x}_{i+1}$, das Gewicht $w_{n+1} = 2|\mathbf{x}_{i+1}|^2 - \epsilon$ für $0 < \epsilon < \min_j\{|\mathbf{x}_{i+1}|^2 - |\mathbf{x}_{i+1}^t \mathbf{x}_j|\}$ und den Bias $-|\mathbf{x}_{i+1}|^2$. Das ergibt die Aktivierung

$$(2|\mathbf{x}_{i+1}|^2 - \epsilon) f(\mathbf{x}_j) - \mathbf{x}_{i+1}^t \mathbf{x}_j - |\mathbf{x}_{i+1}|^2 .$$

Dieses ergibt die gewünschten Ausgaben. □

Offensichtlich kann damit der Algorithmus mit jedem Neuron auch bei beliebigem bisherigen Trainingsverlauf mindestens ein Muster mehr korrekt klassifizieren, sofern die Eingabemuster alle denselben Betrag haben. Das gilt also für Muster aus $\{-1,1\}^n$ und, da sie nur durch eine affine Abbildung der einzelnen Komponenten aus diesen hervorgehen, auch für Muster aus $\{0,1\}^n$. D.h. auf binären oder bipolaren Mustern ist man nach spätestens $|P_0|$ Neuronen fertig, sofern jedes einzelne Neuron optimal trainiert wird.

Trainiert man etwa auf das XOR Problem

$$(0,1;1), (1,0;1), (0,0;0), (1,1;0) ,$$

so kann man z.B. nach dem ersten Durchlauf die Trainingsmenge

$$(0,1,1;1), (1,0,1;1), (0,0,0;0), (1,1,1;0)$$

erhalten, sofern das erste Neuron ein OR berechnet. Diese Trainingsmenge ist etwa mit den Gewichten $(-1,-1,2;1)$ linear trennbar, so daß man in diesem Fall nach zwei Durchläufen fertig ist.

Definition 2.13 Upstart-Algorithmus: *Die Prozedur*

> *Training(P; f)*
> {
>   *Trainiere ein Perzeptron p auf P, die Gewichte seien $(\mathbf{w}, \theta)$.*
>   *Falls P nicht korrekt ist:*
> $P_1 := \{(\mathbf{x}; 1) \mid (\mathbf{x}; 1) \in P, p(\mathbf{x}) = 0\} \cup \{(\mathbf{x}; 0) \mid (\mathbf{x}; 0) \in P\}$
> $P_2 := \{(\mathbf{x}; 1) \mid (\mathbf{x}; 0) \in P, p(\mathbf{x}) = 1\} \cup \{(\mathbf{x}; 0) \mid (\mathbf{x}; 1) \in P\}$
>   *Training(P_1, f_1)*
>   *Training(P_2, f_2)*
> $f(\mathbf{x}) := \mathrm{H}(\mathbf{w}^t\mathbf{x} + \lambda f_1(\mathbf{x}) - \lambda f_2(\mathbf{x}) - \theta)$
>   *sonst:* $f := p$
> }

wird mit Training(P_0; f) gestartet. $\lambda > 0$ ist genügend groß gewählt.

D.h. in jedem rekursiven Schritt werden zwei Neuronen eingefügt, die sich quasi auf das Trennen der noch falschen positiven bzw. der noch falschen negativen Muster vom Rest spezialisieren. Bei geeigneter Kopplung dieses Spezialwissens kommt man zum Ziel, da ja in jedem Schritt die Mengen um mindestens einen Punkt kleiner werden. (Ein positiver und ein negativer Punkt können mindestens richtig gemacht werden.) So erhält man ein Netz mit einer Neuronenanzahl von maximal n, 2^I, 2^{I-1}, $\ldots$, 2, 1 Neuronen je Schicht, wobei $I+1$ die maximale Rekursionstiefe ist.

Satz 2.14 *Wenn man λ genügend groß wählt, dann klassifiziert f die Menge P richtig, wenn f_1 und f_2 die Mengen P_1 bzw. P_2 korrekt klassifizieren.*

Beweis: Für ein Pattern gibt es vier Möglichkeiten für die Aktivierung:

- $(\mathbf{x};1)$ mit $\mathbf{w}^t\mathbf{x} \geq \theta$: $\underbrace{\mathbf{w}^t x - \theta}_{\geq 0} + \underbrace{\lambda f_1(\mathbf{x})}_{\geq 0} - \underbrace{\lambda f_2(\mathbf{x})}_{=0} \geq 0$
- $(\mathbf{x};0)$ mit $\mathbf{w}^t\mathbf{x} < \theta$: $\underbrace{\mathbf{w}^t x - \theta}_{<0} + \underbrace{\lambda f_1(\mathbf{x})}_{=0} - \underbrace{\lambda f_2(\mathbf{x})}_{\geq 0} < 0$
- $(\mathbf{x};1)$ mit $\mathbf{w}^t\mathbf{x} < \theta$: $\underbrace{\mathbf{w}^t x - \theta}_{<0} + \underbrace{\lambda f_1(\mathbf{x})}_{=\lambda} - \underbrace{\lambda f_2(\mathbf{x})}_{=0} \geq 0$
- $(\mathbf{x};0)$ mit $\mathbf{w}^t\mathbf{x} \geq \theta$: $\underbrace{\mathbf{w}^t x - \theta}_{\geq 0} + \underbrace{\lambda f_1(\mathbf{x})}_{=0} - \underbrace{\lambda f_2(\mathbf{x})}_{=\lambda} < 0$

Dieses gilt für genügend großes λ. $\square$

2.7 Ensembles

Ein **Ensemble** kombiniert mehrere Klassifikatoren $f_1, \ldots, f_m$, die etwa von Perzeptronen gebildet werden, zu einem einzelnen Klassifikator durch eine einfache Funktion $f : \mathbb{R}^n \to \mathbb{R}$, so daß eine komplexere Funktion $f \circ (f_1, \ldots, f_m)$ entsteht. Üblich sind etwa eine einfache Mittelung

$$x \mapsto \mathrm{H}(\sum_{i=1}^{m} f_i(\mathbf{x}) - \frac{m}{2})$$

oder eine gewichtete Mittelung

$$x \mapsto \mathrm{H}(\sum_{i=1}^{m} \alpha_i f_i(\mathbf{x}) - \theta)$$

mit geeigneten Gewichten α_i, θ, die man z.B so wählen kann, daß Funktionen f_i mit höherer Güte stärker gewichtet werden, oder die man einfach trainieren kann. Dieser Ansatz wird uns später noch einmal begegnen, wobei wir statt einfachen Perzeptronen f_i komplexere Funktionen kombinieren werden.

Die Darstellungsmächtigkeit so eines Ensembles ist gegenüber einem einfachen Perzeptron gesteigert, denn jede Boolesche Funktion kann mit einem Perzeptronnetz der Tiefe zwei dargestellt werden, sogar falls die Gewichte der Ausgabeneuronen alle 1 sind.

Die Frage stellt sich jetzt, wie man die einzelnen Perzeptronen f_i am besten trainiert. Sie sollten jeweils möglichst wenig Fehler machen. Haben sie allerdings Fehler, dann nützt es offensichtlich nichts, wenn diese für alle Neuronen gleich sind. Daher wendet man Heuristiken an, die möglichst unterschiedliche einzelnen f_i produzieren. Möglichkeiten sind etwa:

- Den Perzeptronalgorithmus für die f_i bei unterschiedlichen Werten starten und unterschiedlich lange trainieren lassen.
- Jedes Neuron auf unterschiedliche Trainingsmengen trainieren.

  Boosting: Ziehe aus der Trainingsmenge P mit Zurücklegen eine Menge derselben Größe pro Perzeptron.

  Arcing: Ziehe je neuem Perzeptron aus P eine Trainingsmenge derselben Größe, wobei die bisher noch nicht korrekt klassifizierten Punkte eine höhere Wahrscheinlichkeit haben. Sind $f_1, \ldots, f_k$ schon trainiert, wählt man für $\mathbf{x}$ etwa die Wahrscheinlichkeit

  $$p(\mathbf{x}) = \frac{1 + |\{i \mid f_i(\mathbf{x}) \text{ ist falsch}\}|^4}{\sum_{j=1}^{|P|}(1 + |\{i \mid f_i(\mathbf{x}_j) \text{ ist falsch}\}|^4)}$$

  Etwa für das XOR Problem kann dieser Mechanismus zu den Trainingsmengen

  $$\{(0,0;0),(0,1;1),(1,0;1)\} \text{ und } \{(0,1;0),(1,0;1),(1,1;0)\}$$

  führen, die beide etwa mit Perzeptronen f_1 und f_2 linear trennbar sind. Die Kombination

  $$H(f_1(\mathbf{x}) + f_2(\mathbf{x}) - 1.5)$$

  löst dann XOR.

Ensembles werden häufig eingesetzt, um die Generalisierungsfähigkeit des Klassifikators zu verbessern. Wir kommen später zu diesem Effekt.

2.8 Perzeptronnetze

Man kann natürlich mit beliebigen Perzeptronnetzen starten. Es reicht zur Darstellung jeder Booleschen Funktion eine verborgene Schicht aus. Nichtsdestotrotz können mehr Schichten die Anzahl der Neuronen reduzieren helfen.

[Etwa die Funktion, die n binäre Eingaben der Größe nach sortiert, kann nicht mit einem Netz der Tiefe 2 und nur polynomiell vielen Neuronen dargestellt werden, hingegen doch mit einem Netz der Tiefe 3 und polynomiell vielen Neuronen.]

Sei also ein festes Netz mit Schichten mit n_0, n_1, $\ldots$, n_h Neuronen und der Perzeptronaktivierung gegeben. Wie kann man dieses trainieren? Ein Algorithmus ergibt sich aus folgender Überlegung: In einem fertig trainierten Netz bildet jedes einzelne Neuron i die Punkte $\mathbf{x}_1^i, \ldots, \mathbf{x}_m^i$, die sich durch die Aktivierung auf den Trainingspattern ergeben, auf Werte y_1^i, $\ldots$, y_m^i ab. Dieses geschieht, indem die Aktivierung des Neurons $\mathbf{w}^t\mathbf{x}_j^i - \theta$ mit 0 verglichen wird. Wir hatten schon gesehen, daß es möglich ist, $\mathbf{w}$ und θ so zu ändern, daß sie sich als Lösung eines Gleichungssystems mit durch die Punkte $\mathbf{x}_j^i$ bestimmten Koeffizienten ergeben, ohne die Funktion des Netzes auf den gegebenen Daten zu ändern. Das heißt aber, mit Auswahl von maximal $n_i + 1$ Punkten aus $\mathbf{x}_j^i$ (n_i sei die Eingabedimension des Neurons i) und deren Klassifikation nach 0 oder 1 ist $\mathbf{w}$ und θ bestimmt. Jetzt testen wir einfach für jedes Neuron Schicht für Schicht rekursive alle Möglichkeiten durch. Es gibt zwar exponentiell viele Möglichkeiten in Bezug auf die jeweilige Eingabedimension n_i des betrachteten Neurons i (man sucht bis zu $\binom{m}{n_i+1}$ Punkte und für diese eine aller möglichen Klassifikationen aus), aber nur polynomiell viele Möglichkeiten in Bezug auf die Anzahl der Trainingspunkte m. D.h. die Prozedur ist polynomiell für eine feste Architektur. Die Architekturparameter treten aber exponentiell auf.

Wahrscheinlich geht das prinzipiell nicht besser. Genauer hat man folgende NP-Ergebnisse:

Satz 2.15 *Betrachte folgendes Problem: $n \in \mathbb{N}$ und eine Patternmenge P in $\{0,1\}^n \times \{0,1\}$ seien gegeben. Gibt es ein Perzeptronnetz $(n, 2, 1)$, das auf P korrekt klassifiziert? (n und P sind variabel.) Dieses Problem ist NP-vollständig.*

Beweis: Das Problem ist in NP, denn man kann (polynomielle) Gewichte raten und testen, ob sie stimmen. Es ist auch NP-vollständig, da man das NP-vollständige 2-SSP darauf reduzieren kann:

Sei ein SSP (S, C) gegeben. Sei o.E. $|c| \leq 3$ für alle $c \in C$. Sei $|S| = n$. Folgende Punkte in $\{0,1\}^{n+3} \times \{0,1\}$ sind zu lernen:

- $(0, \ldots, 0, 0, 0, 0; 1)$,
- $\mathbf{p}_i = (0, \ldots, 1, \ldots 0, 0, 0, 0; 0)$ für alle $i \in \{1, \ldots, n\}$, die 1 steht an der Stelle i,
- $\mathbf{p}_{c_j} = (0, \ldots, 1 \ldots, 0, 1 \ldots, 0, 0, 0, 0; 0)$ für alle $c_j = \{s_{i_1}, s_{i_2}, s_{i_3}\} \in C$; die 1 stehen an den Stellen s_{i_1}, s_{i_2}, s_{i_3},
- $(0, \ldots, 0, 1, 0, 1; 1)$,
- $(0, \ldots, 0, 0, 1, 1; 1)$,
- $(0, \ldots, 0, 1, 0, 0; 0)$,
- $(0, \ldots, 0, 0, 1, 0; 0)$,
- $(0, \ldots, 0, 0, 0, 1; 0)$,
- $(0, \ldots, 0, 1, 1, 1; 0)$

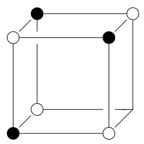
Es ist jetzt zu zeigen, daß dieses Problem mit einem Netz genau dann lösbar ist, wenn das SSP lösbar ist.

Sei eine Lösung S_1, S_2 für das SSP gegeben. Definiere als Gewichte für die beiden verborgenen Neuronen $(\mathbf{w}_1, 1, 1, -1)$ bzw. $(\mathbf{w}_2, -1, -1, 1)$ mit

$$w_{ij} = \begin{cases} -1 & \text{falls } s_j \in S_i \\ 2 & \text{sonst,} \end{cases}$$

und Biases -0.5, die Ausgabe berechne ein und. Das bildet alle Punkte richtig ab. Für die Punkte $\mathbf{p}_{c_j}$ folgt das, da alle Mengen c_j gesplittet werden.

Sei umgekehrt ein neuronales Netz gegeben, das die positiven Punkte korrekt abbildet. Betrachte zunächst folgendes Teilproblem in den letzten drei Koordinaten:



Die unterschiedlich klassifizierten Punkte werden durch zwei Ebenen H_1 und H_2, die den Neuronen in der hidden Schicht entsprechen, voneinander getrennt. O.E. liegt kein Punkt direkt auf einer dieser Ebenen. Sei h_1 die Abbildung, die die Punkte auf der Seite von H_1, auf der $\mathbf{0}$ liegt, nach 1 abbildet, h_2 analog. Dann berechnet das Augabeneuron $h_1 \wedge h_2$: Angenommen, das sei nicht der Fall. Dann würde es nicht nur einen, sondern drei der vier möglichen Werte in der verborgenen Schicht $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$ nach 1 abbilden, da nur eine Ebene offensichtlich die Punkte

nicht trennt und das XOR nicht mit einem Perzeptron lösbar ist. Das hieße aber, daß eine Ebene mindestens zwei schwarze Punkte von allen weißen abtrennt. Das geht offensichtlich nicht.

Betrachte jetzt wieder alle Dimensionen. Die beiden hidden Neuronen definieren je eine Ebene, die wir wieder mit H_1 und H_2 bezeichnen. Analog zu eben seien die Abbildungen h_1 und h_2 definiert. Sei $S_1 = \{s_i \mid h_1(\mathbf{p}_i) = 0\}$ und $S_2 = \{s_i \mid h_2(\mathbf{p}_i) = 0\} \backslash S_1$. Da das Netz $h_1 \wedge h_2$ berechnet und die Punkte $\mathbf{p}_i$ alle nach 0 gehen, ist das eine disjunkte Zerlegung von S. Wären für eine Menge c alle Punkte in S_1, dann würde auch $\mathbf{p}_c$ von h_1 auf 0 abgebildet werden, denn der Punkt liegt dann als Linearkombination der einzelnen $\mathbf{p}_i$ auf derselben Seite der Ebenen H_1, wie die einzelnen $\mathbf{p}_i$. Analog mit S_2. $\square$

Obwohl man das vermutet, folgt nicht automatisch, daß das Training größerer Netze genauso schwierig ist. Dazu bedarf es eines neuen Beweises:

Satz 2.16 *Betrachte folgendes Problem: $n \in \mathbb{N}$ und eine Patternmenge P in $\mathbb{R}^n \times \{0,1\}$ seien gegeben. Gibt es ein Perzeptronnetz $(n, n_1, \ldots, n_h, 1)$, das auf P korrekt klassifiziert? (n und P sind variabel, $n_1 \geq 2, \ldots, n_h$ sind fest.) Dieses Problem ist NP-vollständig.*

Beweis: Auch hier kann man Gewichte raten und testen, ob sie funktionieren.

Der Beweis der NP-Vollständigkeit geht durch eine ähnliche Reduktion vom n_1-SSP: Wir deuten das hier nur an.

Sei ein n_1-SSP (S, C) mit Mengen $c \in C$ der Kardinalität maximal 3 gegeben. Sei $|S| = n$. Sei $n' = n + n_1 + 1$. Folgende Punkte in $\mathbb{R}^{n'} \times \{0,1\}$ sind zu lernen:

- $(0, \ldots, 0, 0, \ldots, 0; 1)$,
- $\mathbf{p}_i = (0, \ldots, 1, \ldots 0, 0, 0, \ldots; 0)$ für alle $i \in \{1, \ldots, n\}$, die 1 steht an der Stelle i,
- $\mathbf{p}_{c_j} = (0, \ldots, 1 \ldots, 0, 1 \ldots, 0, 0, \ldots, 0; 0)$ für alle $c_j = \{s_{i_1}, s_{i_2}, s_{i_3}\} \in C$; die 1 stehen an den Stellen s_{i_1}, s_{i_2}, s_{i_3},
- $(0^n, \mathbf{q}_i, 1; 0)$ für alle $\mathbf{q}_i \in \{-1, 1\}^{n_1} \backslash (1, \ldots, 1)$
- $(0^n, \mathbf{q}_0, 1; 1)$ mit $\mathbf{q}_0 = \{1\}^{n_1}$
- $(0^n, \tilde{\mathbf{z}}_i, 1; 0)$ für $i = 1, \ldots, n_1(n_1 + 1)$ und $(0^n, \bar{\mathbf{z}}_i, 1, 1)$ für $i = 1, \ldots, n_1(n_1 + 1)$, wobei $\tilde{\mathbf{z}}_i$ und $\bar{\mathbf{z}}_i$ wie folgt konstruiert werden:

  Wähle $n_1 + 1$ Punkte in jeder Menge $H_i = \{\mathbf{x} \in \mathbb{R}^{n_1} \mid x_i = 0, \forall j \neq i \; x_j > 0\}$, und nenne sie $\mathbf{z}_1, \mathbf{z}_2, \ldots$, die gesamte Menge Z. Die folgende Eigenschaft soll dabei gelten: Gegeben $n_1 + 1$ verschiedene Punkte in Z, dann liegen diese auf einer Hyperebene dann und nur dann, wenn sie in einem einzigen H_i enthalten sind. (Das kann man durch die Bedingung $\det \begin{pmatrix} 1 & \ldots & 1 \\ \mathbf{z}_{i_1} & \ldots & \mathbf{z}_{i_{n_1+1}} \end{pmatrix} \neq 0$ testen, daher ist das möglich.) Für $z_j \in H_i$ definiere $\tilde{\mathbf{z}}_j \in \mathbb{R}^{n_1}$ als $\tilde{\mathbf{z}}_j = (z_{j1}, \ldots, z_{ji-1}, z_{ji} + \epsilon, z_{ji+1}, \ldots, z_{jn_1})$, und $\bar{\mathbf{z}}_j \in \mathbb{R}^{n_1}$ als $\bar{\mathbf{z}}_j = (z_{j1}, \ldots, z_{ji-1}, z_{ji} - \epsilon, z_{ji+1}, \ldots, z_{jn_1})$, für kleines ϵ mit folgender Eigenschaft: Falls eine Hyperebene in $\mathbb{R}^{n_1}$ mindestens $n_1 + 1$ Paare $(\tilde{\mathbf{z}}_i, \bar{\mathbf{z}}_i)$ separiert, dann sind das die $n_1 + 1$ Paare, die zu denjenigen $n_1 + 1$ Punkten in einer Hyperebenen H_i korrespondieren und die separierende Hyperebene ist nahezu gleich mit derjenigen durch H_i. (Das kann man wieder durch eine Determinante testen, daher geht das.)

  Diese Punkte erzwingen, daß die Neuronen in der ersten verborgenen Schicht nahezu mit den H_i übereinstimmen müssen. Daher werden die $\mathbf{p}_i$ auf ganz $\{0,1\}^{n_1}$ abgebildet in der ersten verborgenen Schicht, das heißt, das restliche Netz berechnet wegen $\mathbf{q}_i$ notwendig ein und!

Sei eine Lösung $S_1, \ldots, S_{n_1}$ des n_1-SSP gegeben. Definiere den Bias des $i$ten Neuron im ersten hidden Layer als -0.5 und die Gewichte als $(a_1, \ldots, a_{|V|}, \mathrm{e}_i, -0.5)$ mit

$$a_j = \begin{cases} -1 & s_j \text{ ist in } S_i \text{ enthalten} \\ 2 & \text{sonst} \end{cases}$$

und dem $i$ten Einheitsvektor e_i. Alle anderen Neuronen berechnen ein und. Das klassifiziert alles richtig.

Sei umgekehrt ein Netzwerk, das alles richtig klassifiziert, gegeben. Wegen der Punkte $\mathbf{q}_i$, $\bar{\mathbf{z}}_i$ und $\tilde{\mathbf{z}}_i$ berechnet dann das Netz ab der zweiten Schicht einfach nur die Funktion und. Definiere $S_i = \{s_j \mid$ das $i$te Neuron in der ersten verborgenen Schicht bildet $\mathbf{p}_j$ nach 0 ab. $\} \backslash (S_1 \cup \ldots \cup S_{i-1})$. Das bildet ein Splitting, wie analog zum obigen Beweis gesehen werden kann. □

Obiges Ergebnis gilt sogar, wenn man nicht eine perfekte, sondern nur eine approximative Lösung sucht, d.h. eine Lösung, die einen großen Bruchteil, aber nicht alle Punkte korrekt klassifiziert.

Man erhält sogar NP-Ergebnisse, wenn man die Eingabedimension fest läßt, aber die Neuronenanzahl variiert. Dieses Problem könnte z.B. auftreten, wenn man nach einer möglichst kleinen Architektur für ein festes Problem sucht. Man erhält:

Satz 2.17 *Für eine Architektur der Form $(n, n_1, \ldots, n_h, 1)$ mit festem $h \geq 2$, $n \geq 2$ und variierendem n_1 und n_2 und eine variierende Patternmenge P in $\mathbb{R}^n \times \{0, 1\}$ ist es NP-hart zu entscheiden, ob P mit geeigneten Gewichten korrekt klassifiziert werden kann.*

Beweis: Reduktion von 'separability in the plane with lines': Seien Punkte Q und R in $\mathbb{R}^2$ gegeben. Betrachte

$$P = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^2 \times \{0,1\} \mid (\mathbf{x}_i \in R \wedge y_i = 1) \vee (\mathbf{x}_i \in Q \wedge y_i = 0)\}.$$

P kann mit einem Netz der Architektur $(2, k, |R|, 1)$ klassifiziert werden dann und nur dann, wenn Q und R sich durch maximal k Geraden trennen lassen:

Falls P korrekt klassifiziert werden kann, dann definieren die durch die k Neuronen in der ersten verborgenen Schicht definierten Geraden k Geraden, die Q und R trennen.

Falls Q und R durch Geraden getrennt werden, dann definiere das Netz wie folgt: Die Neuronen in der ersten verborgenen Schicht entsprechen den k Geraden. Sei $R = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. Das $j$te Neuron in der zweiten verborgenen Schicht berechnet $(x_1, \ldots, x_k) \mapsto (\neg)x_1 \wedge \ldots \wedge (\neg)x_k$, wobei $\neg$ bei x_i auftaucht, falls $\mathbf{p}_j$ auf der negativen Seite der $i$ten Gerade liegt. Insbesondere bildet dieses Neuron $\mathbf{p}_j$ nach 1 und $\mathbf{q}$ nach 0 ab für alle $\mathbf{q} \in Q$. Also tut's ein ‚oder' als Ausgabe. □

Tja, die Situation sieht also ziemlich schnell übel aus, falls die Architektur zu groß wird. Man hofft aber, daß obige schwierigen Fälle in der Praxis nicht auftreten und trainiert trotzdem. Da niemand den oben genannten polynomiellen Algorithmus für kleine Architekturen in der Praxis ernsthaft benutzt, betrachten wir aber jetzt die in der Praxis gebräuchlichen Trainingsmethoden für feedforward Netze.

3 Feedforward Netze

Um ein effizientes Training von feedforward Netzen zu ermöglichen, bedient man sich eines entscheidenden Tricks: Man ersetzt die Aktivierungsfunktion durch die sigmoide Funktion $\mathrm{sgd}(x) = 1/(1 + \mathrm{e}^{-x})$ die ja die Perzeptronaktivierung für Werte gegen $\pm\infty$ gut annähert. Vorteil: Die Netzfunktionen werden differenzierbar, und alles löst sich in Wohlgefallen auf. Alles, was mit

Perzeptronnetzen darstellbar war, läßt sich auch durch Netze mit der sigmoiden Funktion gut approximieren, da für $x \neq 0$

$$\lim_{c \to \infty} \mathrm{sgd}(cx) = \mathrm{H}(x)$$

gilt. Außerdem können jetzt nicht nur binärwertige Abbildungen, sondern Ausgaben aus $[0, 1]$ betrachtet werden.

3.1 Trainingsverfahren

Sei also ein Netz und eine Trainingsmenge $\{(\mathbf{x}_p, \mathbf{y}_p) \in \mathbb{R}^n \times [0,1]^l \mid p = 1, \ldots, m\}$ gegeben. Die Eingabeneuronen seien die Neuronen $1, \ldots, n$. Die Ausgabe des Neurons i bei Eingabe des $p$ten Musters bezeichnen wir mit o_{pi}, die Aktivierung des Neurons i bei Eingabe des $p$ten Musters mit net_{pi}; für die Neuronen, die nicht Eingabeneuronen sind, gilt

$$\mathrm{net}_{pi} = \sum_{j \to i} w_{ji} o_{pj} - \theta_i, \quad o_{pi} = \mathrm{sgd}(\mathrm{net}_{pi}).$$

Der **quadratische Fehler** des Netzes ist die Größe

$$E = \frac{1}{2} \sum_{p=1}^{m} \underbrace{\sum_{j \text{ ist Ausgabeneuron}} (o_{pj} - y_{pj})^2}_{E_p} = \frac{1}{2} \sum_{p=1}^{m} E_p.$$

Falls das Netz alle Beispiele richtig abbildet, dann ist $E = 0$. Training bedeutet, Gewichte zu finden, so daß E möglichst klein ist. Man beachte, daß E differenzierbar ist. Daher ist ein sogenannter **Gradientenabstieg** möglich: Gehe auf der Fehlerfläche schrittweise in die Richtung des steilsten Abstiegs, bis es nicht mehr weiter geht. Die Richtung des steilsten Abstiegs ist aber gerade der sogenannte Gradient

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_{ij}} \right)_{i \to j},$$

wobei wir wieder angenommen haben, daß der Bias jedes Neurons durch ein zusätzliches Gewicht, d.h. eine Verbindung zu einem Eingabeneuron mit konstanter Eingabe 1 realisiert ist. Mathematisch ist Gradientenabstieg daher folgendes Verfahren:

setze $\mathbf{w} := \mathbf{w}_o$ (i.A. kleine Zufallszahlen)
wiederhole
$$\mathbf{w} := \mathbf{w} - \eta \left(\frac{\partial E(\mathbf{w})}{\partial w_{ij}} \right)_{ij}$$

Dabei ist $0 < \eta$ die sogenannte **Schrittweite**, die die Größe der Gewichtsänderungen bestimmt. **Backpropagation** bezeichnet lediglich obiges Verfahren, wobei die Gradienten auf eine spezielle, besonders effiziente Weise berechnet werden; wir betrachten der Einfachheit halber den Fall $p = 1$, und lassen den Index p weg. Für mehrere Pattern muß man die Rechnung für jedes Pattern durchführen und anschließend aufsummieren. Es ist

$$\frac{\partial E(\mathbf{w})}{\partial w_{ij}} = \frac{\partial E}{\partial \mathrm{net}_j} \cdot \frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \delta_j \cdot o_i\,.$$

(Kettenregel: $(f(g(x)))' = f'(g(x)) \cdot g'(x)$) mit dem **Fehlerterm**

$$\delta_j := \frac{\partial E}{\partial \mathrm{net}_j}$$

und

$$\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial \sum_{k\to j} w_{kj} o_k}{\partial w_{ij}} = o_i.$$

Man kann die δ_j sehr einfach durch folgende rekursive Formel berechnen, die es gestattet, ausgehend von den Ausgabeneuronen die Fehlerterme für die Neuronen der einzelnen Schichten zu berechnen. Sie werden quasi Schicht für Schicht zurückpropagiert, daher der Name ‚Backpropagation':

$$\delta_j = \begin{cases} (o_j - y_j) \cdot \mathrm{sgd}'(\mathrm{net}_j), & j \text{ ist Ausgabeneuron} \\ \sum_{j\to k} w_{jk}\delta_k \cdot \mathrm{sgd}'(\mathrm{net}_j), & \text{sonst.} \end{cases}$$

Letzteres ergibt sich wie folgt:

$$\begin{aligned} \frac{\partial E}{\partial \mathrm{net}_j} &= \sum_{j\to k} \frac{\partial E}{\partial \mathrm{net}_k} \cdot \frac{\partial \mathrm{net}_k}{\partial \mathrm{net}_j} \\ &= \sum_{j\to k} \delta_k \sum_{i\to k} w_{ik} \frac{\partial \mathrm{sgd}(\mathrm{net}_i)}{\partial \mathrm{net}_j} \\ &= \sum_{j\to k} \delta_k w_{jk} \cdot \mathrm{sgd}'(\mathrm{net}_j) \end{aligned}$$

(Kettenregel im Mehrdimensionalen: $f(g_1(x),\ldots,g_n(x))' = \frac{\partial f}{\partial g_1} \cdot {g_1}'(x) + \ldots + \frac{\partial f}{\partial g_n} {g_n}'(x)$)

Definition 3.1 Offline/Batch Backpropagation *ist folgender Algorithmus:*

> $\eta := \eta_o$;
> *Initialisiere* $\mathbf{w}$, $\boldsymbol{\theta}$ *mit kleinen Zufallszahlen;*
> *Wiederhole*
> $\triangle\mathbf{w} := \mathbf{0}$; $\triangle\boldsymbol{\theta} := \mathbf{0}$;
>   *Für jedes Pattern* $(\mathbf{x}_p, \mathbf{y}_p)$
> $o_i := \begin{cases} x_{pi} & i \textit{ ist Eingabeneuron,} \\ \mathrm{sgd}(\sum_{j\to i} w_{ji} o_j - \theta_i) & \textit{sonst.} \end{cases}$
> $\delta_j := \begin{cases} (o_j - y_{pj}) o_j (1 - o_j) & j \textit{ ist Ausgabeneuron,} \\ \sum_{j\to k} w_{jk}\delta_k o_j (1 - o_j) & \textit{sonst} \end{cases}$
> $\triangle w_{ij} := \triangle w_{ij} - o_i \delta_j$;
> $\triangle\theta_i := \triangle\theta_i + \delta_i$;
> $\mathbf{w} := \mathbf{w} + \eta\triangle\mathbf{w}$; $\boldsymbol{\theta} := \boldsymbol{\theta} + \eta\triangle\boldsymbol{\theta}$;   *(*)*

Bei **online Backpropagation** *wird die Änderung (*) ersetzt durch die Änderung*

$$w_{ij} := w_{ij} - \eta o_i \delta_j; \theta_i := \theta_i + \eta\delta_i;$$

innerhalb der Schleife.

Dabei ist $\eta_0 > 0$; wir haben $\mathrm{sgd}'(x) = \mathrm{sgd}(x)(1 - \mathrm{sgd}(x))$ benutzt. Online Backpropagation nimmt die Gewichtsänderungen jeweils schon nach jedem Pattern vor, ist also kein tatsächlicher Gradientenabstieg mehr. Es hat sich aber gezeigt, daß die dadurch entstehenden Zufälligkeiten förderlich für die Ergebnisse sind. Der Aufwand von Backpropagation hängt von der Anzahl der nötigen Schleifendurchläufe ab, die sich je nach gewählter Lernrate und Situation ändern kann. Der Aufwand eines einzelnen Schleifendurchlaufs ergibt sich als Anzahl der Pattern multipliziert mit der Anzahl der Gewichte mal eine Konstante, da in jeder ‚Vorwärtswelle' zum Berechnen der o_i bzw. ‚Rückwärtswelle' zum Berechnen der δ_i jedes Gewicht genau einmal angeschaut wird.
Dieses Verfahren ist mit einigen Problemen konfrontiert, die man durch zahlreiche Variationen versucht hat, zu meistern:

- Lokales statt globales Minimum gefunden,
- Minimum wird aufgrund zu großer Schrittweite übersprungen,
- Oszillation in schmalen Tälern,
- Stagnatation im Hochplateaus,
- ...

Folgende teilweise heuristisch motivierte Modifikationen sind etwa möglich. (Bias durch On-Neuron realisiert!):

- **Flat-spot-elimination:**
  In den Fehlertermen δ_j taucht die Größe $\mathrm{sgd}'(x)$ auf, die im besten Fall 0.25, im schlimmsten Fall nahezu 0 ist. Dieses führt zu extrem kleinen Fehlersignalen, insbesondere, sofern sie durch mehrere Schichten propagiert werden. Bei flat-spot-elimination verwendet man statt $\mathrm{sgd}'(x)$ den Wert $\mathrm{sgd}'(x) + \epsilon$ für ein $\epsilon > O$.

- **Momentum Term:** (Nur für die Offline-Version)
  Nahe bei lokalen Minima kann es geschehen, daß der Gradient zu groß ist und deswegen das Verfahren oszilliert. Umgekehrt kann auf einer langen Gefällstrecke ein kleines lokales Minimum den Suchprozeß aufhalten. Die Idee ist, gegen solche Effekte ein Trägheitsmoment einzuführen, so daß tendentiell die letzte Richtung beibehalten wird. Es bezeichne $\triangle\mathbf{w}$ die in $(*)$ berechnete Änderung und $\triangle\mathbf{w}(t)$ die im Folgenden tatsächlich vorgenommene Änderung nach dem $t$ten Schleifendurchlauf, d.h. $w_{ij} := w_{ij} + \triangle w_{ij}(t)$. Es ist $\triangle\mathbf{w}(0) = \mathbf{0}$. Bei Backpropagation mit Momentum ist

  $$\triangle w_{ij}(t+1) = \eta\triangle w_{ij} + \alpha\triangle w_{ij}(t)$$

  mit $\alpha \in [0.2, 0.9]$ dem **Momentum Term**. Allerdings ist der Effekt begrenzt und ändert nichts daran, daß die auch durch den Gradienten bestimmte Schrittweite für die jeweiligen Situationen unpassend ist.

- **Manhattan Training:** Man ersetzt innerhalb der Schleife

  $$\triangle w_{ij} := \triangle w_{ij} - o_i\,\mathrm{sgn}(\delta_j);$$

  Die Fehlersignale bestimmen also nur die Richtung der Änderung und nicht mehr die Größe. Tatsächlich entspricht dieses einem Gradientenabstieg auf der durch $\sum_p \sum_j |o_{pj} - y_{pj}|$ gegebenen Fehlerfläche. Da in 0 keine Differenzierbarkeit gegeben ist, kommt es hier evtl. zu Oszillationen.

- **SuperSAB:**
Die Idee ist, für jedes Gewicht eine eigene Schrittweite zu verwenden, um Verzerrungen für die einzelnen Richtungen zu vermeiden. Die Schrittweiten werden adaptiert.

$$\eta_{ij}(0) = \eta_0 \quad \text{(z. B. 1)}$$
$$\triangle w_{ij}(t) = \eta_{ij}(t)\triangle w_{ij}$$
$$\eta_{ij}(t) = \begin{cases} \eta_{ij}(t-1)\cdot\eta^- & \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) < 0, \\ \eta_{ij}(t-1)\cdot\eta^+ & \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) > 0, \\ \eta_{ij}(t-1) & \text{sonst}, \end{cases}$$

mit $\partial E(t)/\partial w_{ij} = -\triangle w_{ij}$. $\eta^- \in]0,1[$ z.B. 0.5 sorgt für Verkleinerung der Schrittweite, falls sich die Richtung geändert hat, $\eta^+ > 1$ z.B. 1.2 sorgt für Vergrößerung, falls die Richtung beibehalten bleibt. Allerdings ist der Gradient ein immer noch stark bestimmender Faktor. Im zweiten Fall kann es zu einer Explosion der Schrittweite kommen.

- **DeltaBarDelta:**
Dasselbe Verfahren, wobei man den zweiten Fall durch

$$\eta_{ij}(t) = \eta_{ij}(t-1) + \eta^+, \text{ falls } \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) > 0$$

ersetzt. Dadurch soll eine Explosion verhindert werden.

- **RProp:**
Resilient Propagation benutzt auch eine eigene adaptive Schrittweite für jedes Gewicht, verzichtet aber gänzlich darauf, die i.A. irreführende Größe des Gradienten zu verwenden. Zudem werden verschlechternde Schritte, d.h. man ist über das Minimum hinausgegangen, zurückgenommen und mit verbesserter Schrittweite neu probiert.

$$\triangle w_{ij}(t) = \begin{cases} -\triangle w_{ij}(t-1) & \text{falls } \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) < 0, \\ & \text{setze zudem } \frac{\partial E}{\partial w_{ij}}(t) := \frac{\partial E}{\partial w_{ij}}(t-1), \\ -\eta_{ij}(t)\cdot \operatorname{sgn}\left(\frac{\partial E}{\partial w_{ij}}(t)\right) & \text{sonst} \end{cases}$$

$\eta_{ij}(t)$ wird wie bei SuperSAB verändert. Der erste Fall entspricht dem Zurücknehmen eines verschlechternden Schrittes. Die Größe der maximalen Gewichtsänderung wird zusätzlich beschränkt. RProp ist ein extrem robustes und schnelles Verfahren, so daß es häufig die Methode der Wahl ist. Allerdings ist schnelles Training häufig kontraproduktiv für die Generalisierungsleistung, so daß diese durch weight decay oder early stopping erzwungen werden sollte (kommt später).

- **Steepest descent:**
Die Idee ist, in Richtung des Gradienten soweit zu gehen, daß man in dieser Richtung ein Minimum erreicht, und dann erst eine neue Suchrichtung einzuschlagen. In Richtung des Gradienten wird nur die Schrittweite adaptiert, aber kein neuer Gradient berechnet. Dieses hilft etwa bei langen, geraden (!) Tälern oder Hochebenen, schlägt aber schon bei einer einfachen Fehlerfläche mit elliptischen Höhenlinien fehl, da es stark oszilliert.

- **Konjugierte Gradienten:**
Alternativ kann man in einer leicht vom Gradienten abweichenden Richtung suchen, so daß diese Oszillation verhindert wird. Wenn man im Schritt t schon in der Richtung d_t gesucht hat, dann möchte man nicht wieder in die Richtung d_t suchen müssen. D.h. in der neuen Suchrichtung d_{t+1} sollte der Gradient möglichst senkrecht zu d_t sein. Für folgende Überlegungen benutzt man, daß sich jede hinreichend glatte Funktion durch eine Taylorentwicklung gut approximieren läßt. Es gilt

$$f(x) \approx f(x_0) + \nabla f(x_0)(x-x_0) + 1/2\cdot(x-x_0)^t H(x_0)(x-x_0)$$

mit $H =$ Hessematrix von f. Die Approximation ist für Polynome maximal zweiten Grades sogar exakt, für mindestens zweimal stetig differenzierbare Funktionen kann sie durch einen Term abgeschätzt werden, der mit dem Abstand der Punkte x und x_0 skaliert. Obige Idee, den Suchrichtung so zu wählen, daß der Gradient möglichst senkrecht zur alten Suchrichtung bleibt, bedeutet:

$$\begin{aligned} 0 &\approx \nabla E(\underbrace{\mathbf{w}_{t+1} + \lambda\cdot d_{t+1}}_{\text{neue Suchgerade}})^t \cdot d_t \\ &\approx \underbrace{\nabla E(\mathbf{w}_{t+1})^t\cdot d_t}_{\approx 0,\ \text{da entlang } d_t \text{ minimiert}} + \lambda d_{t+1}^t H(\mathbf{w}_{t+1}) d_t, \end{aligned}$$

Dabei sei $\mathbf{w}_t$ der Gewichtsvektor vor Minimieren in Richtung d_t und H die Hessematrix von E. Man minimiert z.B. in sogenannten **konjugierten** Richtungen, d.h. Richtungen mit

$$d_{t+1}^t H(\mathbf{w}_{t+1}) d_t \approx 0.$$

d_{t+1} wird jetzt als um einen Anteil in Richtung d_t korrigierter Gradient gewählt, so daß d_{t+1} und d_t konjugiert sind: $d_{t+1} = -\nabla E(\mathbf{w}_{t+1}) + \beta\cdot d_t$, wobei β so gewählt wird, daß $d_{t+1}^t H d_t = 0$ gilt, d.h. $(-\nabla E(\mathbf{w}_{t+1}) + \beta\cdot d_t)^t H d_t = 0$

$$\Rightarrow \quad \beta = \frac{\nabla E(\mathbf{w}_{t+1})^t H d_t}{d_t^t H d_t}.$$

Dieses ist nur definiert, falls $d_t^t H d_t > 0$ ist. Anderenfalls muß man sich mit ad hoc Richtungen, etwa dem einfachen Gradienten, behelfen. Dieser Ausdruck ist noch sehr ineffizient zu berechnen, daher formt man weiter um: Sei $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\cdot d_t$. Dann ist

$$H\cdot d_t \approx \frac{\nabla E(\mathbf{w}_{t+1}) - \nabla E(\mathbf{w}_t)}{\alpha_t},$$

also

$$\begin{aligned} \beta &\approx \frac{\nabla E(\mathbf{w}_{t+1})^t(\nabla E(\mathbf{w}_{t+1}) - \nabla E(\mathbf{w}_t))}{\underbrace{d_t^t(\nabla E(\mathbf{w}_{t+1})}_{0} - \nabla E(\mathbf{w}_t))} \\ & \qquad\qquad \text{(Hestenes-Stiefel)} \\ &\approx \frac{\nabla E(\mathbf{w}_{t+1})^t(\nabla E(\mathbf{w}_{t+1}) - \nabla E(\mathbf{w}_t))}{\underbrace{-d_t^t}_{\nabla E(\mathbf{w}_t) - \beta d_{t-1}} \nabla E(\mathbf{w}_t)} \\ &\approx \frac{\nabla E(\mathbf{w}_{t+1})^t(\nabla E(\mathbf{w}_{t+1}) - \nabla E(\mathbf{w}_t))}{\nabla E(\mathbf{w}_t)^t \nabla E(\mathbf{w}_t)} \end{aligned}$$

(Pollack-Ribiere)

$$\approx \quad \frac{\nabla E(\mathbf{w}_{t+1})^t \nabla E(\mathbf{w}_{t+1})}{\nabla E(\mathbf{w}_t)^t \nabla E(\mathbf{w}_t)}$$

(Fletcher-Reeves)

da $\nabla E(\mathbf{w}_t) \perp \nabla E(\mathbf{w}_{t+1})$ annähernd gilt, wie wir gleich sehen werden. Der gesamte Algorithmus ist also:

$\mathbf{w} := \mathbf{w}_0$;
$d := -\nabla E(\mathbf{w}_0)$;
Wiederhole
{
  Finde t, so daß $E(\mathbf{w} + t \cdot d)$ minimal ist (line search).
  $\mathbf{w}' := \mathbf{w} + td$;
  $\beta := \dfrac{(\nabla E(\mathbf{w}') - \nabla E(\mathbf{w}))^t \nabla E(\mathbf{w}')}{\nabla E(\mathbf{w})^t \nabla E(\mathbf{w})}$;
  $\mathbf{w} := \mathbf{w}'$;
  $d := -\nabla E(\mathbf{w}) + \beta d$;
}

Für eine quadratische Funktion E konvergiert dieses Verfahren nach spätestens $n =$ Anzahl der Parameter Schritten, denn für quadratisches E sind alle bisherigen Näherungen exakt, außer für Richtungen, wo E konstant ist $\mathbf{x}^t H(\mathbf{w})\mathbf{x} \neq 0$, so daß alle Terme definiert sind, und die berechneten Gradienten sind paarweise orthogonal, $\alpha_t \neq 0$ außer für $\nabla E(\mathbf{w}_t) = 0$.

[Beweis für letzteres: Sei $g_t := \nabla E(\mathbf{w}_t)$, $H := H(\mathbf{w})$ ist konstant. Durch Induktion nach t wird folgendes bewiesen:

1. $g_t \perp d_i \quad \forall i < t$,
2. $\alpha_i \neq 0$ außer für $g_i = 0$,
3. $g_i^t g_j = 0 \quad \forall i \neq j \leq t$,
4. $d_i \neq 0$ außer für $g_i = 0$,
5. $d_i^t H d_j = 0 \quad \forall j < i \leq t$.

In obigen Fällen ist für $t = 0$ nichts zu zeigen. Der Induktionsschritt ist wie folgt:

1. $g_{t+1}^t d_t = 0$ nach Konstruktion. Für $j < t$ ist $g_{t+1}^t d_j \stackrel{IV}{=} (g_{t+1} - g_t)^t d_j = \alpha_t d_t^t H d_j \stackrel{IV}{=} 0$, da $H d_t = (g_{t+1} - g_t)/\alpha_t$.
2. Für eine quadratische Funktion kann man α_i ausrechnen:
$$\begin{aligned} & 0 = \nabla E(\mathbf{w}_i + \alpha_i d_i) = g_i + \alpha_i H d_i \\ \Rightarrow\ & 0 = d_i^t g_i + \alpha_i d_i^t H d_i = -g_i^t g_i + \alpha_i d_i^t H d_i \\ \Rightarrow\ & \alpha_i = (g_i^t g_i)/(d_i^t H d_i). \end{aligned}$$
Die zweite Zeile benutzt dabei, daß in die Suchrichtung minimiert wurde, g_i also senkrecht zur alten Suchrichtung d_{i-1} steht. Also $\alpha_i = 0 \Rightarrow g_i^t g_i = 0 \Rightarrow g_i = 0$
3. $g_{t+1}^t g_0 = -g_{t+1}^t d_0 \stackrel{(1)}{=} 0$
$g_{t+1}^t g_i = g_{t+1}^t(-d_i + \beta_{i-1} d_{i-1}) \stackrel{(1)}{=} 0$
4. $d_{t+1} = 0 \Rightarrow g_{t+1} = \beta_t d_t \stackrel{(1)}{\Rightarrow} g_{t+1} = 0$
5. $d_{t+1}^t H d_t = 0$, da konjugiert.
$d_{t+1}^t H d_j \stackrel{IV}{=} -g_t^t H d_j = -g_t^t(g_{j+1} - g_j)1/\alpha_j \stackrel{(3)}{=} 0$ für $t > j + 1$.]

- **Newton-Verfahren:** Mit dem Ansatz $\nabla E(\mathbf{w}_{t+1}) \approx \nabla E(\mathbf{w}_t) + H(\mathbf{w}_t)(\mathbf{w}_{t+1} - \mathbf{w}_t)$ erhält man die Iterationsvorschrift
$$\mathbf{w}_{t+1} = \mathbf{w}_t - H(\mathbf{w}_t)^{-1} \nabla E(\mathbf{w}_t) .$$
Durch die Matrixinversion ist das Verfahren allerdings aufwendig. Es kann sehr instabil sein, sofern die Approximation schlecht ist.

- **Quickprop:** Die Matrixinversion beim Newtonverfahren wird umgangen. Man nimmt an, H habe Diagonalgestalt, und ersetzt die Einträge durch den Differenzenquotienten
$$\frac{\partial^2 E(\mathbf{w}_t)}{\partial w_{ij}^2} \approx \frac{\dfrac{\partial E(\mathbf{w}_t)}{\partial w_{ij}} - \dfrac{\partial E(\mathbf{w}_{t-1})}{\partial w_{ij}}}{w_{ij}(t) - w_{ij}(t-1)} .$$
Man erhält
$$w_{ij}(t+1) = w_{ij}(t) - \frac{\dfrac{\partial E(\mathbf{w}_t)}{\partial w_{ij}} \triangle w_{ij}(t)}{\dfrac{\partial E(\mathbf{w}_t)}{\partial w_{ij}} - \dfrac{\partial E(\mathbf{w}_{t-1})}{\partial w_{ij}}} .$$
Ebenso wie das Newton Verfahren kann Quickprop sehr instabil sein.

- **Monte Carlo:** In jedem Schritt wird zufällig eine Gewichtsänderung aus einem vorgegebenen Intervall gezogen und bei Verkleinerung des Fehlers auch vorgenommen.

- **Simulated Annealing:** Die obige Änderung wird auch bei Verschlechterung mit der Wahrscheinlichkeit $e^{-\triangle E/T}$ akzeptiert für ein $T > 0$, welches im Laufe des Verfahrens gegen 0 konvergiert, und $\triangle E = E(\mathbf{w}_t) - E(\mathbf{w}_{t-1})$.
Beide Verfahren sind sehr langsam, da sie die Struktur der Fehlerfläche in keiner Weise ausnutzen. Allerdings können sie auch für nicht differenzierbare Fehlerfunktionen E verwandt werden. Gegenüber Monte-Carlo kann Simulated Annealing lokale Minima wieder verlassen, so daß bei geeigneter Verkleinerung von T die Konvergenz gegen ein Optimum sichergestellt ist.

3.2 Präsentation der Daten

In der Praxis sind Beispiele für eine zu lernende Gesetzmäßigkeit gegeben, und die Lernaufgabe soll mit einem feedforward Netz gelöst werden. Dazu muß die Aufgabe so formuliert werden, daß sie als Lernen einer Funktion $f : x \in \mathbb{R}^n \mapsto y \in \mathbb{R}^m$ aufgefaßt werden kann. Die Daten sollten so repräsentiert sein, daß die zu lernende Funktion eine möglichst einfache Form hat und möglichst viel exaktes Vorwissen in die Repräsentation integriert ist. Zugleich sollte die Eingabedimension möglichst niedrig sein, um zu gewährleisten, daß die Daten die unbestimmten Parameter der Architektur festlegen, d.h. überflüssige Information sollte vermieden werden. (Dieser Punkt wird später noch exakt gemacht.) Die Beispiele, die die Gesetzmäßigkeit bezeugen, sollten repräsentativ für den Bereich sein, für den die Funktion gelernt werden soll.

In der Praxis gibt es keine Standardverfahren für eine geeignete Repräsentation. Einige der oben genannten Kriterien sind offensichtlich widersprüchlich, und es gilt, eine geeignete Balance zwischen den Anforderungen zu finden. Eine geeignete Repräsentation zu finden, ist in der Regel eine sehr zeitaufwendiges Unterfangen, von dem allerdings der Erfolg des Lernens wesentlich abhängt. Es folgen einige Kochrezepte für mögliche Repräsentationen:

- Symbolische Daten: Viele Daten sind durch Attribute symbolischer Natur beschrieben. diese müssen als reelle Zahlen kodiert werden. Treten nur zwei Ausprägungen auf oder besitzen die Attribute eine natürliche Anordnung, so kann man sie durch 0 und 1 bzw. durch verschiedene aufsteigende Werte im Intervall $[0, 1]$ repräsentieren. Besitzen die Attribute keine natürliche Anordnung, so ist eine unäre Kodierung angebracht: Attribut a_i wird durch den $i$ten Einheitsvektor in $\mathbb{R}^a$, $a =$ Anzahl der Attribute, repräsentiert. Das ist gegenüber einer theoretisch ebenfalls denkbaren binären Kodierung vorzuziehen, da eine binäre Kodierung nicht begründete Ähnlichkeiten zwischen unterschiedlichen Attributen definieren würde. Unäre Kodierung ist insbesondere angebracht, wenn man eine Klassifikation der Daten in mehr als zwei Klassen lernen möchte, d.h. zur Kodierung der Ausgabe.

- Reelle Daten können direkt eingegeben werden. Nichtsdestotrotz ist im Allgemeinen eine Skalierung der Daten angebracht, um nicht – bei anfänglich gleicher Lernrate für jedes Gewicht – eine Eingabe beim Lernalgorithmus zu bevorzugen. Skalieren erfolgt so, daß tendentiell das Intervall $[0, 1]$ oder $[-1, 1]$ durch die einzelnen Eingaben ausgeschöpft ist. Die Daten sollten dazu linear transformiert werden, z.B.:

  $$x \mapsto \frac{x - x_m}{x_M - x_m}$$

  mit $x_m =$ minimaler angenommener Wert und $x_M =$ maximaler angenommener Wert.

  Vermutet man, daß die Eingaben zwar beschränkt, aber die Extrema nicht in der Datenmenge tatsächlich vorhanden sind, kann man x_m bzw. x_M um z.B. 5% der Distanz $x_M - x_m$ verkleinern bzw. vergrößern.

  Sind die Daten nicht beschränkt bzw. zwar beschränkt, aber mit wenigen Ausreißern, dann kann man sie auch durch den Ausdruck

  $$x \mapsto \frac{x - \sum_{i=1}^{p} x_i/p}{\sum_{i=1}^{p}(x_i - \sum_{j=1}^{p} x_j/p)^2/(p-1)}$$

  normieren, p bezeichne die Anzahl der Muster, x_i die betrachtete Koordinate des Musters i. Die einzelnen Koeffizienten der Patternmenge bilden eine Verteilung. Dieser Ausdruck sorgt dafür, daß diese Verteilung den Erwartungswert 0 und die Varianz 1 hat, d.h. tendentiell liegen die Daten im Intervall $[-1, 1]$.

  Möchte man bei Ausreißern keine numerischen Probleme bekommen, kann man die Daten auch einfach quantifizieren in geeignet viele symbolische Beschreibungen: klein, mittel, hoch, sehr hoch.

- Häufig hat man das Problem, daß einige Daten fehlende Attribute aufweisen. Sind genügend Daten vorhanden, kann man sie einfach wegstreichen. Sind nicht genügend Daten vorhanden, dann müssen die Attribute geeignet ersetzt werden. Möglich sind etwa:

  - geeigneter Default-Wert (etwa bei symbolischen Attributen die häufigste Ausprägung),

  - häufigster Wert bzw. Mittelung der Ausprägung bei den k ansonsten ähnlichsten Mustern,
  - Wert, der angibt, daß der Wert fehlt, (zusätzliches Neuron auf 1),

- Bei der Zeitreihenprognose/-verarbeitung möchte man häufig aus einer prinzipiell unbegrenzten Zeitreihe einen Wert vorhersagen. Da ein feedforward Netz nur mit begrenzt vielen Eingaben umgehen kann, legt man über die Zeitreihe einen Zeitfenster einer festen Größe k, anhand dessen die Daten vorhergesagt werden sollen. k muß durch Ausprobieren bestimmt werden. Die Aufgabe ist dann, aus x_{t-k}, x_{t-k+1}, …, x_t die Ausgabe vorherzusagen, statt aus x_1, …, x_t. Globale Information, d.h. von allen bisherigen Werten abhängige Information, kann man zusätzlich in beschränktem Maße integrieren: man kann in einem zusätzlichen Wert etwa eine exponentiell abfallend gewichtete Summe über alle bisherigen Eingaben speichern oder die Zahl der unmittelbaren Vorgänger, die insgesamt eine aufsteigende/absteigende Folge bilden, aufsummieren. Beide Größen sind von einer im Prinzip unbeschränkten Vergangenheit abhängig.

  Bei Zeitreihenprognose besteht sehr schnell die Gefahr, nicht für repräsentative Daten zu lernen, sofern die Reihen nicht stationär sind. Stationär heißt, daß sich die Reihe prinzipiell bei jedem Zeitpunkt so verhält, wie ganz zu Anfang, sofern man die Vorgänger nicht weiß. Liegt ein Trend (z.B. Inflationsrate) vor, kann man diesen aber leicht beseitigen, indem man etwa zu den Differenzen $x_{i+1} - x_i$ übergeht oder von der Zeitreihen einen linearen Prozeß $l(x_i, \ldots, x_{i-k})$ abzieht, der durch einfache Gaußsche Regression gewonnen wurde.

- In der Bildverarbeitung bestehen die Daten aus Bildern, die z.B. Elemente aus $\mathbb{R}^{n \times m}$ sind. Im allgemeinen ist die Dimension sehr hoch und das Material durch die Aufnahmegegebenheiten (Licht, Verwackeln, …) nicht optimal. Die Dimension kann reduziert werden, indem man z.B. statt jedes Pixels je über mehrere Pixel mittelt oder charakteristische (problemabhängige) Features statt des Bildes verwendet. Oft ist nicht der gesamte Ausschnitt wichtig; relevante Bereiche können ausgeschnitten werden. Etwa bei Ziffernerkennung sollte man alle Ziffern gleich skaliert und zentriert präsentieren. Binarisierung (d.h. alle Werte über einer gewissen Schwelle werden 1, alle anderen 0, auch feiner in Graustufen möglich) verkleinert den benötigten Spericherplatz und Berechnungsaufwand.

  Um das Material zu verbessern, kann es geeignet vorverarbeitet werden. Oft wird dabei über jedes Pixel eine Maske gelegt, die angibt, wie das Pixel mit seiner Umgebung verrechnet werden soll. Diese Filter bewirken z.B., daß Details deutlicher oder umgekehrt Rauschen unterdrückt wird. Lineare Filter können einfach durch die Koeffizientenmatrix angegeben werden, z.B. bewirkt der Filter

  $$\frac{1}{16} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

  ein Glätten, der Filter

  $$\frac{1}{4} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

  extrahiert Kanten. Es gibt auch globale Operatoren wie Fouriertransfomation, Segmentierung, Texturanalyse, ….

  In der pixelbasierten Bildverarbeitung soll jedes einzelne Pixel eines Bildes separat auf eine Ausgabe abgebildet werden, etwa um Luftaufnahmen automatisch zu kartographieren und die verschiedenen Objekte Feld, Wald, bebautes Land, ... herauszufinden. Dazu wendet man einige der obigen Operationen an, so daß man mehrere Repräsentationen desselben Bildes erhält, und verwendet die je an nur einem Pixel stehende Information als Eingabe für ein Netz. Dadurch, daß geeignet vorverarbeitet wurde, erhält das Netz (hoffentlich) auch die zur Klassifikation des Pixels nötige Information, denn nur mit dem Grauwert des Pixels selber kann ein Netz in der Regel nichts anfangen.

- In der Sprachverarbeitung trifft man auch auf verschiedene Filter, Fouriertransformation, ... Hinzu kommt hier der schon erwähnte Aspekt der Zeitreihenverarbeitung.

- Häufig müssen nicht nur die einzelnen Pattern, sondern auch die ganze Patternmenge an das Problem angepasst werden. Es kann z.B. sein, daß gewisse Daten nicht genügend häufig repräsentiert sind, obwohl Ausgaben in diesem Bereich für die spätere Nutzung relevant sind. Etwa bei einem medizinischen Problem kann es wesentlich weniger Krankheitsfälle als andere Fälle geben. Sollen dennoch alle Fälle gleich gut gelernt werden, müssen die weniger vertretenen stärker gewichtet, d.h. die entsprechenden Trainingsdaten z.B. mehrmals identisch kopiert werden. Häufig kennt man zudem Eigenschaften der Abbildung, etwa eine Invarianz gegen gewisse Transformationen, die nicht komplett in die Repräsentation der Pattern eingearbeitet werden kann. Man kann dann dem Netz zusätzliche mithilfe der Transformationen erzeugte Pattern präsentieren, um eine Invarianz durch die Lernaufgabe zu erzwingen. Eine gewisse Robustheit gegen Rauschen wird etwa dadurch erzwungen, daß man die Eingabedaten mehrfach leicht verrauscht repräsentiert.

3.3 Interpretation der Trainingsergebnisse

Auch die trainierten Ausgabedaten sind nicht bzgl. ihres Formates notwendig identisch zu den tatsächlich gewünschten Ausgaben. Möglich ist auch hier, daß bei reellen Daten eine Skalierung auf das Intervall $[0,1]$ oder $[-1,1]$ vorgenommen wurde, so daß eine Rückskalierung erforderlich ist. Die Ausgabewerte sollten im (Abschluß des) Wertebereichs der verwendeten Aktivierungsfunktion liegen! Bei Klassifikationsaufgaben benutzt man häufig eine unäre Kodierung, wie schon beschrieben. Es ist dabei nicht klar, zu welcher Klasse die Eingabe gehören soll, sofern die Ausgabe nicht eine strikt unäre Darstellung besitzt, was in der Regel bei sigmoiden Ausgaben nicht der Fall ist. Folgende Möglichkeiten sind denkbar:

band Es wird um jede gewünschte Ausgabe ein Intervall einer angegebenen Bandbreite gelegt. Sind alle Ausgabewerte innerhalb dieses Intervalls, dann ist die Ausgabe korrekt klassifiziert, sind alle außerhalb, ist sie falsch klassifiziert, ansonsten ist die Ausgabe unbekannt.

402040 Die Sollausgabe ist unär. Die tatsächliche Ausgabe heißt ebenfalls unär, falls genau ein Koeffizient größer als ein vorgegebener Wert ist und alle anderen Koeffizienten kleiner einem anderen vorgegebenen Wert sind. Falls der höchste Wert mit der zu prognostizierenden Ausgabe übereinstimmt, ist die Klassifikation korrekt, sonst falsch. Falls die Ausgabe nicht unär ist, ist das Ergebnis unbekannt.

WTA Die Sollausgabe ist unär. Die tatsächliche Ausgabe heißt ebenfalls unär, falls genau ein Koeffizient größer als ein vorgegebener Wert ist und alle anderen Koeffizienten um mindestens eine vorgegebene Spanne kleiner sind. Falls der höchste Wert mit der zu prognostizierenden Ausgabe übereinstimmt, ist die Klassifikation korrekt, sonst falsch. Falls die Ausgabe nicht unär ist, ist das Ergebnis unbekannt.

Die genaue Wahl obiger Parameter hängt in der Regel vom spezifischen Ergebnis ab. Es ist klar daß die so interpretierten Ergebnisse nicht genau mit dem berechneten Trainingsfehler übereinstimmen müssen, d.h. ein Netz mit schlechterem Trainingsfehler kann durchaus bessere Klassifikationsergebnisse liefern.

Das legt nahe, auch die Fehlerfunktion in Frage zu stellen. In speziellen Situationen können vom quadratischen Fehler verschiedene Funktionen sinnvoll erscheinen. Der Fehler für ein Pattern kann etwa als

$$\sum_j |o_j - y_j|^p$$

für $p \geq 1$ definiert werden. Im Fall $p = 1$ ist das in Null nicht differenzierbar und sorgt evtl. für Oszillation. Dagegen werden aber große Abweichungen nicht so stark gewichtet wie beim quadratischen Fehler. Die Wahl $p > 2$ bestraft dagegen große Abweichungen mehr. Allgemeiner kann man den Fehler als

$$\sum_j f(o_j, y_j)$$

mit einer nichtnegativen differenzierbaren Funktion f mit der Eigenschaft $f(x,x) = 0$ wählen. Je nachdem, ob diese oberhalb oder unterhalb dem quadratischen Fehler liegt, bestraft sie Abweichungen stärker oder schwächer. Dieses kann sinnvoll auch asymmetrisch in o und y geschehen, sofern Abweichungen in der einen Richtung eher zu vermeiden sind als in der anderen.

Bei einer unären Kodierung der Ausgabe trifft man gelegentlich die sogenannte **Kreuzentropie** an, d.h. den Fehler

$$-\sum_j y_j \ln(o_j/y_j)$$

mit der Vereinbarung $0 \cdot \ln\infty := 0$. Dieses ist für $y_j = o_j$ Null und ansonsten positiv. Man kann denselben Ansatz bei einer Wahrscheinlichkeitszuordnung statt einer nur unären Ausgabe verwenden, d.h. im Fall $\sum y_j = 1$, $\sum o_j = 1$. Die Kreuzentropie bildet dann ein Fehlermaß zwischen zwei Wahrscheinlichkeitsverteilungen. Im Vergleich zum Fehler $|y-o|^p$ hat obiges Fehlermaß Singularitäten an den Stellen mit $y_j = 1$ und $o_j \neq 1$, vermeidet also tendentiell definitive Fehlklassifikationen.

Die Änderung des Fehlers hat natürlich veränderte Berechnungsformeln für Backpropagation zur Folge. Falls $E_p = \sum_j d(o_j, y_j)$ gilt, berechnen sich die δ_j als

$$\delta_j = \begin{cases} \dfrac{\partial d(o_j,y_j)}{\partial o_j} o_j(1-o_j) & j \text{ ist Ausgabeneuron,} \\ \sum_{j\to k} w_{jk}\delta_k o_j(1-o_j) & \text{sonst} \end{cases}$$

Die Kreuzentropie verlangt, daß die Ausgaben tendentiell unär sind bzw. eine Wahrscheinlichkeitsverteilung darstellen. Um dieses zu bewirken, bietet sich eine Änderung der Aktivierungsfunktion an. Wird einfach die sigmoide Funktion sgd durch eine andere differenzierbare Funktion f ausgetauscht, dann ändert sich in den Formeln lediglich der Term $o_i(1-o_i)$, der durch $f'(o_i)$ ersetzt wird. Möglich ist etwa eine lineare Funktion oder $\tanh$, sofern die Ausgabe nicht nach $[0,1]$ skaliert werden soll. Eine Ausgabe, die eine Verteilung darstellt, wird durch die sogenannte **Softmax** Funktion erreicht:

$$\text{soft-sgd}_i(x_1,\ldots,x_n) = 1/(1+\mathrm{e}^{x_i - \ln\sum_{j\neq i}\mathrm{e}^{x_j}}) = e^{x_i}/\sum_j \mathrm{e}^{x_j},$$

die keine lokale Funktion mehr ist, sondern in $x_1, \ldots, x_n$ die Aktivierungen aller Ausgaben benötigt, um sie in der Summe auf 1 zu normieren. Ableiten führt zu der geänderten Formel

$$\delta_i = \sum_k \left(\frac{e^{\text{net}_k}}{\sum_j e^{\text{net}_j}} - y_k \right) \cdot \left(\frac{-e^{\text{net}_k} e^{\text{net}_i} + \delta_i^k e^{\text{net}_i} \sum_j e^{\text{net}_j}}{(\sum_j e^{\text{net}_j})^2} \right)$$

für Ausgabeneuronen i mit dem Kroneckersymbol δ_i^k.

Weiter ist natürlich eine Gewichtung der Fehlerterme für die einzelnen Pattern möglich, etwa um relevante Bereiche der Eingabe hervorzuheben oder ein gleichmäßiges Lernen auch auf schwach vertretenen Bereichen der Eingabe zu bewirken. Man sollte auf eine (fast überall gegebene) Differenzierbarkeit der Fehlerfunktion und der Aktivierungsfunktionen, sowie auf die Tatsache, daß die Fehlerfläche nicht in relevanten Gebieten konstant, d.h. der Gradient 0 ist, achten. Letzteres verhindert etwa bei der Perzeptronaktivierungsfunktion, daß ein Gradientenabstieg verwandt werden kann, denn man bleibt in der Regel einfach am Startpunkt auf einem Hochplateau stehen.

Falls der Trainingsfehler klein ist, sagt dieses aber noch nichts über den Erfolg des Trainings aus. In der Regel ist man am Trainingsfehler nicht interessiert, sondern am Verhalten des Netzes auf unbekannten Daten. Dazu sei X der Eingaberaum, P eine Verteilung auf X, Y der Ausgaberaum, $f : X \to Y$ eine zu lernende Funktion und $f_w : X \to Y$ die durch das Netz gelernte Funktion. Dann ist nicht der quadratische Fehler auf den Trainingsdaten die relevante Größe, sondern der Fehler

$$\int_X (f(x) - f_w(x))^2 dP.$$

Dabei bedeutet das Integral, daß man für eine diskrete Verteilung P auf den Werten $x_1, x_2, \ldots$ einfach summiert

$$\sum_{i=1}^{\infty} (f(x_i) - f_w(x_i))^2 P(x_i)$$

und bei einer Dichte p, die P beschreibt, integriert

$$\int_X (f(x) - f_w(x))^2 p(x) dx\ .$$

(Wir nehmen an, daß die obigen Ausdrücke definiert sind, was bei unseren Anwendungen immer der Fall sein wird.) In der Regel ist der Trainingsfehler keine gute Schätzung für obigen **Generalisierungsfehler**, denn es wurde auf die Daten trainiert, so daß sie tendentiell richtig sind. Üblicherweise behält man daher einen Teil der Daten zurück, die sogenannte **Testmenge**, und schätzt den Generalisierungsfehler durch den sogenannten **Testfehler** ab, d.h. durch den quadratischen (oder je betrachteten) Fehler auf der Testmenge, auf die ja nicht trainiert wurde.

Wie gut ist das Training jetzt, wenn der Testfehler bestimmt ist? In der Regel besagt weder ein kleiner Testfehler, daß erfolgreich trainiert wurde, noch ein großer Testfehler, daß das Training fehlgeschlagen ist. Dieses hängt von den Daten ab. In der Regel sind die Daten fehlerbehaftet, so daß man statt Pattern $(x, f(x))$ die Muster $(x, f(x) + \eta)$ mit einem Rauschen η erhält. Bei der Eingabe x kann man für den Fehler berechnen

$$\begin{aligned} E((f(x) + \eta - f_w(x))^2) &= E((f(x) + \eta)^2) + f_w(x)^2 - 2 f_w(x) E(f(x) + \eta) \\ &= \underbrace{(f_w(x) - E(f(x) + \eta))^2}_{\text{systematischer Fehler}} + \underbrace{E((f(x) + \eta - E(f(x) + \eta))^2)}_{\text{unsystematischer Fehler}} \end{aligned}$$

(Hierbei ist der Erwartungswert bzgl. dem Rauschen η gebildet worden. Der **Erwartungswert** $E(X)$ einer Zufallsgröße X ist, sofern definiert, für Variablen X mit diskreten Werten die Größe

$$E(X) = \sum_i a_i P(X = a_i)$$

und für Variablen mit Dichte p die Größe

$$E(X) = \int_x x p(x) dx.$$

Der Erwartungswert ist linear und für stochastisch unabhängige Variablen auch multiplikativ. Die **Varianz** $E((X - EX)^2)$ einer Zufallsgröße X gibt die zu erwartende quadratische Abweichung vom Erwartungswert an.)

Der systematische Fehler rührt daher, daß das Netz die Gesetzmäßigkeit nicht korrekt interpoliert hat. Der unsystematische Fehler liegt an Meßungenauigkeiten oder anderen Zufälligkeiten und kann prinzipiell nicht vermieden werden, er bildet eine untere Schranke für den Generalisierungsfehler. Tatsächlich ist ein Netz optimal, welches nur den unsystematischen Fehler besitzt.

Wie kann man den unsystematischen Fehler abschätzen, um die Güte des Testfehlers zu bestimmen? Die Anzahl der Eingaben in der Trainingsmenge, die widersprüchliche Ausgaben besitzen, können sicher für eine untere Schranke verwandt werden. Problematisch hierbei ist allerdings, daß in der Regel nicht identische, sondern nur fast identische Eingaben mit widersprüchlichen Ausgaben vorliegen werden. Um deren Anteil abzuschätzen, muß man also den Anteil an ähnlichen Daten mit verschiedenen Ausgaben abschätzen. Bei Klassifikationen kann man etwa eine bestimmte kleine Nachbarschaft untersuchen, in der keine widersprüchlichen Daten liegen dürfen. In der Regel wird man hier mehr oder weniger aufwendige statistische Verfahren verwenden, die eine Abschätzung ermöglichen, oder einfach den besten erreichten Testfehler zum Bias deklarieren.

Eine obere Schranke für den Testfehler erhält man etwa durch den Vergleich mit anderen einfachen Verfahren, bei Regression etwa einer einfachen linearen Regression, bei Klassifikation einer Zuordnung aller Daten zur Klasse mit der größten Wahrscheinlichkeit. Dieses gibt eine grobe Richtung für die Komplexität des Trainingsproblems an. In der Regel ist es immer sinnvoll, mehr als nur ein Verfahren zu verwenden, um die Ergebnisse zu bestätigen.

3.4 Architekturauswahl

Wir haben gesehen, wie man eine feste Architektur trainieren, die Daten aufbereiten und die Ergebnisse interpretieren kann, als nächstes stellt sich die Frage, wie man eine geeignete Architektur auswählt. Wir werden später sehen, daß Netze universelle Approximatoren sind, d.h. bei geeigneter Architektur kann prinzipiell alles dargestellt werden. Man kann also eine Architektur finden, wo der Trainingsfehler fast zu Null wird. Dieses Netz zu verwenden, ist in der Regel ein schlechter Ratschlag. Auf den Daten sind in der Regel unsystematische Fehler präsent, die zu einem Bias führen, der auch bei optimaler Wahl des Netzes nicht unterboten werden kann. Die unsystematischen Fehler sind allerdings in der Trainingsmenge nur implizit vorhanden, tatsächliche Widersprüche treten selten auf, so daß ein Netz prinzipiell diese unsystematischen Abweichungen auch lernen kann. Das führt allerdings dazu, daß die Netzfunktion den konkreten Daten und den vorliegenden Fehlern folgt, so daß der Generalisierungsfehler und der Testfehler auf einer Menge, wo die Fehler nicht dieselben sind, hoch ist.

Klar wird dieses durch eine kleine Rechnung. Sei D eine konkrete Datenmenge. Von deren spezieller Ausprägung hängt das Training ab, d.h. man berechnet für das zu erwartende Ergebnisnetz f_w, welches die Funktion f approximieren soll,

$$E_D((f_w(x) - f(x))^2) = \underbrace{(E_D(f_w(x)) - f(x))^2}_{\text{Bias}^2} + \underbrace{E_D((f_w(x) - E_D(f_w(x)))^2)}_{\text{Varianz}}$$

Die Varianz ist hoch, sofern das Netz der jeweiligen Trainingsmenge gut folgt, denn dann werden jedesmal die jeweiligen Zufälligkeiten abgebildet, dafür ist dann allerdings der Bias klein, denn die Daten werden ja perfekt interpoliert. Diese Situation tritt ein, wenn ein Netz viele Freiheitsgrade, d.h. Gewichte hat, der Effekt heißt **Overfitting**. Hat das Netz umgekehrt wenige Gewichte, dann ist die Varianz gering, denn das Netz kann den jeweiligen Zufälligkeiten der Trainingsmenge nicht folgen. Andererseits ist aber der Bias evtl. hoch, da das Netz nicht genügend Freiheitsgrade zur Darstellung der zu lernenden Funktion hat.

De Facto muß man also eine geeignete Balance zwischen den beiden widersprüchlichen Aufgaben, den Bias und die Varianz klein zu halten, finden. D.h. man benötigt ein Netz, das mächtig genug ist, die Funktion darzustellen, aber nicht mächtig genug, den Zufälligkeiten der Trainingsmenge zu folgen. Dieser Zwiespalt wird auch mit **Bias-Varianz-Dilemma** bezeichnet.

Ein naheliegendes Verfahren ist, mehrere Architekturen zu trainieren, den Generalisierungsfehler je auf einer Testmenge abzuschätzen, und anschließend die beste Architektur zum endgültigen Training und Feintuning zu verwenden. Allerdings ist das Ergebnis stark von der jeweiligen Testmenge abhängig, die Daten gehen außerdem für das Training verloren. **k-fache Kreuzvalidierung** zerlegt die Trainingmenge T in k gleich große Teilmengen $T_1, \ldots, T_k$, trainiert eine Architektur N auf $T \backslash T_i$ und schätzt je den Testfehler F_i ab, der sich auf T_i ergibt. Als Schätzer für die Generalisierungsfähigkeit der Architektur gilt dann die Größe

$$\sum_i F_i/k.$$

Man kann zeigen, daß die Varianz dieses Schätzers um eine Ordnung kleiner ist als die Varianz von F_i. Zudem gehen keine Daten für das Training verloren. Die gemittelten Fehler definieren eine Ordnung auf den Architekturen, gemäß der man eine optimale Architektur auswählen kann. Kreuzvalidierung ist zur Zeit eines der häufig benutzten Mittel zur Architekturauswahl.

Ist die Architektur so gewählt, daß die gegebenen Daten gerade die freien Parameter festlegen, dann besteht allerdings im Allgemeinen das Problem, daß die Fehlerminimierung kompliziert ist und die Fehlerfläche viele lokale Minima besitzt. Daher wählt man im Allgemeinen die Anzahl der freien Parameter eher etwas zu groß und verhindert Overfitting in jedem Trainingslauf durch eine Veränderung des Trainings.

Weight Decay erzwingt, daß die Gewichte tendentiell klein sind, indem zum zu minimierenden Fehler der Term

$$\sum_{i,j} w_{ij}^2$$

addiert wird. D.h. von jedem Gewicht wird in jedem Schritt ein kleines Vielfaches abgezogen. Durch tendentiell kleine Gewichte ist die Netzfunktion eher glatt, kann also Zufälligkeiten auf der Trainingsmenge nicht folgen. Tatsächlich entspricht Weight Decay einer Regularisierung, sofern man Gaußsches Rauschen auf den Daten annimmt.

Early Stopping stoppt das Training, bevor das Netz anfängt, sich auf spezielle Ausprägungen in der Trainingsmenge einzustellen. Um diesen Zeitpunkt bestimmen zu können, wird während des Trainings der Fehler auf einer (kleinen) sogenannten **Validierungsmenge** mitprotokolliert, auf die

nicht selbst trainiert wird. Im allgemeinen sinkt der Fehler zu Beginn auf der Validierungsmenge, solange allgemeine Gesetzmäßigkeiten gelernt werden, und steigt wieder, sobald die Spezialitäten der Trainingsmenge gelernt werden. Bevor er steigt, wird gestoppt.

3.5 Pruning

Eine andere Möglichkeit der Regularisierung ist, die Verknüpfungsstruktur der Architektur während oder nach dem Training zu ändern. Verbindungen und Neuronen, die für die Ausgabe nicht relevant sind, werden gelöscht. Dieses verhindert aufgrund der reduzierten Parameterzahl ein Overfitting, ermöglicht aber dennoch effizientes Taining, da für die Anfangsphase des Trainierens genügend Variablitität vorhanden ist.

Methoden, die aufgrund der Netzfunktion irrelevante Netzbereiche löschen, nennt man **Pruningmethoden**. Neben dem Effekt, daß die Generalisierungsleistung verbessert werden kann, erlaubt Pruning eine kompaktere und einfachere Darstellung (fast) derselben Funktion. Ist das Pruning von Eingabeneuronen erlaubt, kann man zudem irrelevante Eingabefaktoren bestimmen. Verschiedene Pruningmethoden sind gebräuchlich:

- **MagPruning** löscht in jedem Schritt die betragsmäßig kleinste Verbindung. Dieses Verfahren ist schnell und erstaunlicherweise auch sehr leistungsfähig.

- **Non-contributing Units** löscht Neuronen, die auf einer gegebenen Trainingsmenge entweder ihre Aktivierung nicht stark ändern, mit der Aktivierung eines anderen festen Neurons übereinstimmen oder mit dem negativen der Aktivierung eines anderen festen Neurons übereinstimmen.

- **Skelettierung** löscht Neuronen, die irrelevant für die Güte des Trainingsfehlers sind. Dazu ersetzt man die Gewichte ausgehend vom Neuron i w_{ij} durch $\alpha_i w_{ij}$ mit einen Faktor α_i. Ist $\alpha_i = 0$ bedeutet das, daß das Neuron wegfällt, ist $\alpha_i = 1$ besteht keine Änderung zum ursprünglichen Netz. Es sei $E(\alpha_i)$ der quadratische Fehler auf den Daten. Man kann Neuron i löschen, falls
$$E(\alpha_i = 0) - E(\alpha_i = 1)$$
klein ist. Man approximiert den Ausdruck
$$\frac{E(\alpha_i = 0) - E(\alpha_i = 1)}{0 - 1} \approx -\frac{\partial E}{\partial \alpha_i}(\alpha_i = 1).$$
Die Ableitung kann mit Backpropagation berechnet werden:
$$\frac{\partial E(\alpha_i)}{\partial \alpha_i} = \sum_{i \to j} \frac{\partial E(\alpha_i)}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial \alpha_i} = \sum_{i \to j} \delta_j(\alpha_i) o_i(\alpha_i) w_{ij},$$
wobei die Ausdrücke $\delta_j(\alpha_i)$ die Fehlersignale aus Backpropagation darstellen, $o_j(\alpha_i)$ die Ausgabe wie bei Backpropagation darstellt. Die Ausgaben können durch eine Vorwärtswelle, die Fehlersignale durch eine Rückwärtswelle, die nur bis zu den Nachfolgern von i berechnet werden muß, erhalten werden.

  In der Praxis verwendet man häufig den durch den Betrag gegebenen Fehler statt des quadratischen Fehlers, da dafür obige Approximation genauer ist. Oft mittelt man die Terme, die die Relevanz der Neuronen angeben, über mehrere Trainingszyklen.

- **Optimum Brain Damage** prunt einzelne Gewichte, die irrelevant erscheinen. Ändert man die Gewichte gegenüber den trainierten Gewichten um einen Vektor $\triangle\mathbf{w}$, dann ergibt sich für die Änderung des quadratischen Fehlers der Term

$$\triangle E \approx \underbrace{\nabla E(\mathbf{w})\triangle\mathbf{w}}_{\approx 0} + \frac{1}{2}(\triangle\mathbf{w})^t H(\mathbf{w})\triangle\mathbf{w},$$

wobei H die Hessematrix der Fehlerfunktion darstellt. Man nimmt an, diese habe Diagonalgestalt und erhält also

$$\triangle E \approx \frac{1}{2}\sum_{ij}\frac{\partial^2 E(\mathbf{w})}{\partial w_{ij}^2}\delta w_{ij}^2.$$

Streicht man nur ein Gewicht, dann ist diese Änderung offensichtlich minimal, falls w_{ij} das Gewicht mit minimalem $\partial^2 E/\partial w_{ij}^2(\mathbf{w})w_{ij}^2$ ist. Die zweite Ableitung erhält man als

$$\frac{\partial(o_i\delta_j)}{\partial w_{ij}} = o_i\frac{\partial\delta_j}{\partial w_{ij}} = o_i^2\frac{\partial\delta_j}{\partial \mathrm{net}_j}.$$

Die Ableitung der δ_j kann man entweder durch den Differenzenquotienten über zwei Zeitschritte approximieren oder in einer erneuten Vorwärts- und Rückwärtswelle (mit dem Aufwand $O(W^2)$) bestimmen.

- **Optimum Brain Surgeon** verwendet denselben Ansatz, nähert aber nicht die Hessematrix durch eine Diagonalform. Zusätzlich werden je alle Gewichte in Bezug auf das zu löschende Gewicht optimal geändert. Mathematisch löst man die Aufgabe, $1/2\triangle\mathbf{w}^t H(\mathbf{w})\triangle\mathbf{w}$ unter der Nebenbedingung $\triangle w_{ij} + w_{ij} = 0$ (d.h. Gewicht w_{ij} wird gelöscht) für ein w_{ij} zu minimieren. Dasjenige w_{ij} mit dem kleinsten Minimum wird entfernt und alle anderen Gewichte entsprechend geändert. Die Minimierung kann Standardmethoden aus der Analysis verwenden, wobei auch hier wieder Heuristiken für eine größere Effizienz sorgen. Das Verfahren ist insgesamt relativ aufwendig.

- **Sensitivitätsanalyse** erlaubt, die Eingabeneuronen gemäß ihrer Bedeutung für die Ausgabe zu ordnen. Die Neuronen werden als tendentiell unwichtig angesehen, wo ein Fehler der Eingaben nicht viel bewirkt. D.h. die Ableitung der Ausgaben nach der betreffenden Eingabe ist für die Muster der Trainingsmenge klein. Dazu benötigen wir die Ableitungen $\partial o_i/\partial x_j = \partial o_i/\partial \mathrm{net}_j$ der Ausgabe o_i nach der Eingabekomponente x_j, die wir hier auch als Aktivierung des entsprechenden Eingabeneurons definieren. Analog zu Backpropagation erhält man die Rekursionsgleichung

$$\frac{\partial o_i}{\partial \mathrm{net}_j} = \begin{cases} \mathrm{sgd}'(\mathrm{net}_i)\delta_i^j & j \text{ ist Ausgabeneuron} \\ \sum_{j\to k}\frac{\partial o_i}{\partial \mathrm{net}_k}\cdot \mathrm{sgd}'(\mathrm{net}_j)w_{jk} & \text{sonst.}\end{cases}$$

Diejenige Eingabe k mit kleinstem Wert $\sum_{p \text{ ist pattern}}\sum_{i \text{ ist Ausgabe}}|\partial o_i/\partial x_k(\mathbf{x}_p)|$ kann als die unwichtigste Eingabekoeffiziente angesehen werden, wenn – und nur wenn – die Eingaben gleich skaliert sind.

3.6 Konstruktive Methoden

Umgekehrt gibt es Verfahren, die die Architektur konstruktiv verändern; einige solche hatten wir bei einfachen Perzeptronen schon kennengelernt.

Cascade Correlation spiegelt Ideen vom Tower-Algorithmus wieder. Es wird in jedem Schritt ein zusätzliches hidden Neuron eingefügt, das von allen anderen hidden Neuronen und den Eingaben Werte liest und seine Ausgabe zur Gesamtausgabe weitergibt. Training geschieht in zwei Stufen: Es wird je das hidden Neuron trainiert, danach die Ausgabe neu trainiert. Da je nur einzelne Neuronen trainiert werden, geschieht dieses sehr effizient. Die Ausgabe kann dabei je auf die gewünschten Ausgaben trainiert werden. Die gewünschte Ausgabe für das hidden Neuron ist nicht offensichtlich. In Cascade Correlation wird es so trainiert, daß die Ausgabe des hidden neuron mit dem bis dato noch gegebenen Fehlers möglichst stark korreliert ist. Falls dieses gelingt, dann würde eine einfache Subtraktion der Ausgabe des hidden Neurons von der Gesamtaktivierung den Fehler verringern.

Um exakt zu werden, nehmen wir an, daß nur eine Ausgabe vorliege, d.h. $f:\mathbb{R}^n\to\mathbb{R}$ ist zu lernen. Wir benutzen Neuronen mit der Aktivierungsfunktion $\tanh$. Es gilt $\tanh'(x) = 1-\tanh(x)^2$. Biase sind durch On-Neuronen realisiert. Sukzessive werden die hidden Neuronen $n_i:\mathbb{R}^{n+i-1}\to\mathbb{R}$ mit Eingaben $\mathbf{x}$, n_1, …, n_{i-1} und je das Ausgabeneuron $f:\mathbb{R}^{n+i}\to\mathbb{R}$ mit Eingaben $\mathbf{x}$, n_1, …, n_i trainiert, so daß die entstehende Funktion

$$f(\mathbf{x}, n_1(\mathbf{x}), n_2(\mathbf{x}, n_1(\mathbf{x})), n_3(\mathbf{x}, n_1(\mathbf{x}), n_2(\mathbf{x}, n_1(\mathbf{x}))), \ldots)$$

die Ausgaben approximiert. Der Algorithmus ist wie folgt:

$I := 1$;
wiederhole
{
  trainiere die Gewichte von f_I mit Eingaben $\mathbf{x}$, n_1, …, n_{I-1}
  trainiere $n_I := \tanh(\sum w_i x_i + \sum_{j=1}^{I-1} w_{n+j}n_j)$ auf $S(n_I, |f_I - f|)$
  $I := I+1$;
}

Dabei bezeichnet $S(h,g)$ die Korrelation zwischen zwei Funktionen h ung g auf den Daten $\mathbf{x}$:

$$\left|\sum_{p=1}^{P}\left(h(\mathbf{x}_p) - \sum_p h(\mathbf{x}_p)/P\right)\left(g(\mathbf{x}_p) - \sum_p g(\mathbf{x}_p)/P\right)\right|.$$

Der Term $\sum h(\mathbf{x}_p)/P$ sei mit $\bar{h}$ abgekürzt. Ist der Term $S(n_i, |f_i - f|)$ groß, so bedeutet das, daß tendentiell entweder die Abweichungen des Fehlers vom Mittel und die Abweichung der Aktivierung vom Mittel gleiche Größenordnung besitzen, oder tendentiell sie immer nur um ein Vorzeichen verschieden sind. Einfaches Addieren bzw. Subtrahieren der Aktivierung zur Ausgabe verringert also den Fehler beträchtlich. Wird die Korrelation durch Gradientenaufstieg maximiert, so benötigt man die Ableitung von $S(n_i, |f_i - f|)$, die sich wie folgt berechnet:

$$\begin{aligned}\frac{\partial S}{\partial w_j} &= \pm\sum\left(\frac{\partial n_i}{\partial w_j}(\mathbf{x}_p) - \frac{\partial\bar{n}_i}{\partial w_j}\right)\left(|f_i(\mathbf{x}_p) - f(\mathbf{x}_p)| - \overline{|f_i - f|}\right)\\ &= \pm\sum(1-n_i(\mathbf{x}_p))^2 x_{pj}\left(|f_i(\mathbf{x}_p) - f(\mathbf{x}_p)| - \overline{|f_i - f|}\right)\end{aligned}$$

denn

$$\begin{aligned}&\sum\partial\bar{n}_i/\partial w_j\left(|f_i(\mathbf{x}_p) - f(\mathbf{x}_p)| - \overline{|f_i - f|}\right) = \\ &\partial\bar{n}_i/\partial w_j\sum\left(|f_i(\mathbf{x}_p) - f(\mathbf{x}_p)| - \overline{|f_i - f|}\right) = 0.\end{aligned}$$

Ensembles hatten wir schon einmal betrachtet. Es ist natürlich auch möglich, feedforward Netze $f_1, \ldots, f_n$ zu einem einzigen Netz

$$\sum \alpha_i f_i$$

zu kombinieren, zumal ja bei Kreuzvalidierung sowieso mehrere verschiedene Netze trainiert werden. Gilt $\sum \alpha_i = 1$, so erweist sich dieser Ansatz als günstig für die Generalisierungsfähigkeit der sich ergebenden Funktion, zudem erhält unsere vormalige Heuristik, die Einzelnetze so verschieden wie möglich zu wählen, eine theoretische Begründung. Die f_i sind Ergebnisse eines von den zufallsbehafteten Daten abhängigen Trainingsprozesses. Für den zu erwartenden Fehler bzgl. der zu lernenden Funktion f kann man berechnen:

$$\begin{aligned}
E\left((\sum \alpha_i f_i - f)^2\right) &= E\left((\sum \alpha_i f_i)^2\right) - 2\sum \alpha_i E(f_i f) + E(f^2) \\
&= \sum \alpha_i E(f_i^2) - 2\sum \alpha_i E(f_i f) + E(f^2) \\
&\quad - \sum \alpha_i E(f_i^2) + 2\sum \alpha_i \sum \alpha_j E(f_i f_j) - E\left((\sum \alpha_i f_i)^2\right) \\
&= \sum \alpha_i E\left((f_i - f)^2\right) - \sum \alpha_i E\left((f_i - \sum \alpha_i f_i)^2\right)
\end{aligned}$$

D.h. der zu erwartende Fehler des Ensembles ergibt sich aus der Mittelung der zu erwartenden Fehler der Einzelnetze verringert um die zu erwartende Varianz der Netze! Die Generalisierung ist also mindestens genauso gut, wie die der Einzelnetze, und sie wird umso besser, ja mehr Variabilität die einzelnen Netze aufweisen. Anschaulich ist das klar, da ja die Netze dort, wo sie durch die Daten bestimmt werden, identisch sind. Die Variabilität betrifft Parameter, die nicht durch die Daten festgelegt sind, so daß sie unterschiedlichem Rauschen folgen können. Im Mittel wird dann das Rauschen unterdrückt.

Die Frage ist, wie man die Gewichtungen α_i wählen kann. Eine Möglichkeit ist eine einfache Mittelung, d.h. $\alpha_i = 1/$ Anzahl der Netze. Man kann aber das Problem auch als Optimierungsproblem betrachten: Suche α_i mit $\sum \alpha_i = 1$, so daß für diese der quadratische Fehler minimal wird. Sei $\epsilon_i = f - f_i$ die Abweichung des $i$ten Netzes. Dann soll

$$E\left((\sum \alpha_i \epsilon_i)^2\right) = \sum \alpha_i \alpha_j \underbrace{E(\epsilon_i \epsilon_j)}_{E_{ij}}$$

unter der Bedingung

$$\sum \alpha_i = 1$$

minimiert werden. [Minimieren einer Funktion $h(\mathbf{x})$ unter der Bedingung $g(\mathbf{x}) = 0$ kann man mithilfe sog. Lagrangemultiplikatoren, d.h. $\nabla(h(\mathbf{x}) + \lambda g(\mathbf{x})) = 0$.] Man erhält

$$\begin{aligned}
\mathbf{0} &= \nabla\left(\sum \alpha_i \alpha_j E_{ij} + \lambda(\sum \alpha_i - 1)\right) \\
&= (2\sum \alpha_i E_{ij} + \lambda)_j \\
\Rightarrow \quad \alpha_k &= -\lambda/2 \sum_l (E_{ij})^{-1}_{kl}
\end{aligned}$$

mit der Matrix $(E_{ij})_{ij}$. Wegen $\sum \alpha_i = 1$ kann man λ ersetzen und erhält

$$\alpha_k = \frac{\sum_l (E_{ij})^{-1}_{kl}}{\sum_{k,l} (E_{ij})^{-1}_{kl}}\,.$$

Die Einträge $E_{ij} = E(\epsilon_i \epsilon_j)$ der Matrix können durch die Daten $\mathbf{x}_p$ abgeschätzt werden:

$$E_{ij} \approx \frac{1}{P} \sum_{p=1}^{P} (f(\mathbf{x}_p) - f_i(\mathbf{x}_p))(f(\mathbf{x}_p) - f_j(\mathbf{x}_p))\,.$$

3.7 Approximationseigenschaften

Es stellt sich aber noch die Frage, ob, gegeben eine Patternmenge, überhaupt ein Netz existiert, das diese Menge approximiert und möglichst auch die zugrundeliegende Gesetzmäßigkeit widerspiegelt. Dieses ist die Frage nach der Darstellungsmächtigkeit von feedforward Netzen. Deren positive Beantwortung erscheint dringend, da ja z.B. einfache Perzeptronen sehr beschränkt sind. Schranken für die Ressourcen würden darüberhinausgehend den Suchraum für geeignete Acrchitekturen bei der Architekturauswahl beschränken.

Wir werden einen Satz aus der Analysis benutzen.

Satz 3.2 *(Stone-Weierstraß) Sei I ein kompaktes Intervall und F eine Algebra von auf I stetigen reellwertigen Funktionen. Falls F separierend ist, d.h. für alle $a \neq b \in I$ eine Funktion mit unterschiedlichem Wert auf a und b existiert, und F Konstanten enthält, dann kann jede auf I stetige Funktion beliebig gut durch eine Funktion aus F auf I approximiert werden.*

Die Eigenschaft, Algebra zu sein, bedeutet hier, daß mit zwei Funktionen in F auch deren Summe, Produkt, und skalare Vielfache in F sind. Approximation bedeutet hier Approximation in der Maximum Norm, d.h. für ein zu approximierenden f und alle $\epsilon > 0$ gibt es ein $g \in F$ mit $|f(x) - g(x)| < \epsilon$ für alle $x \in I$. Unmittelbare Folgerung hiervon ist, daß geeignete Netze mit nur einer verborgenen Schicht approximationsuniversell sind.

Satz 3.3 *Es sei $\sigma : \mathbb{R} \to \mathbb{R}$ eine squashing Funktion. f sei stetig auf I. Dann gibt es für jedes $\epsilon > 0$ ein feedforward Netz f_w mit nur einer verborgenen Schicht mit Aktivierungsfunktion σ und linearer Ausgabe, das f auf I bis auf ϵ approximiert.*

Beweis: Wir benutzen den Satz von Stone-Weierstraß. Zunächst betrachten wir Netze mit linearer Ausgabe, der Aktivierungsfunktion $\cos$ in der hidden Schicht und sog. Σ-Π Neuronen. Ein Σ-Π Neuron kann man sich als Verknüpfung von mehreren einfachen Neuronen vorstellen, deren einzelne Aktivierungen zu einer Gesamtaktivierung multipliziert werden. So ein Netz berechnet also die Ausgabe

$$\sum_{i=1}^{k} w_i \Pi_{k_i} \cos(\mathbf{w}_{k_i}^t \mathbf{x} - \theta_{k_i})$$

mit geeigneten Gewichten $\mathbf{w}$. Offensichtlich kann man die Summe, das Produkt und skalare Vielfache von solchen Ausdrücken wieder als solch einen Ausdruck darstellen, d.h die Menge der durch diese Netze berechneten Funktionen bildet eine Algebra. Diese enthält offensichtlich alle Konstanten, denn $c = c \cdot \cos(\mathbf{0}^t\mathbf{x} - 0)$, und ist separierend, denn für $\mathbf{x} \neq \mathbf{y}$, etwa $x_1 \neq y_1$ ist $\cos(wx_1 + 0.5) \neq \cos(wy_1 + 0.5)$ für $w = 1/(2(\max\{|x_1|, |y_1|\} + 1))$.

Es gilt $\cos(a + b) + \cos(a - b) = 2\cos a \cos b$, also

$$\begin{aligned}
&\Pi_{k=1}^{K} \cos(\textstyle\sum_j w_{jk}x_j - \theta_k) \\
= \;&\Big(\Pi_{k=1}^{K-2} \cos(\textstyle\sum_j w_{jk}x_j - \theta_k \cdot \cos\Big(\sum_j (w_{(K-1)j} + w_{Kj})x_j - \theta_(K-1) - \theta_K\Big) \\
&+ \Pi_{k=1}^{K-2} \cos(\textstyle\sum_j w_{jk}x_j - \theta_k \cdot \cos\Big(\sum_j (w_{(K-1)j} - w_{Kj})x_j - \theta_(K-1) + \theta_K\Big)\Big)/2
\end{aligned}$$

Man kann also induktiv die Produkte gegen Summen tauschen und erhält, daß Netze mit einer hidden Schicht, der Aktivierungsfunktion $\cos$ und linearer Ausgabe die Funktion f auf I bis auf $\epsilon/2$ approximieren können.

Als nächstes wird die Funktion $\cos$ durch die Funktion $\widetilde{\cos}$ ersetzt, wobei

$$\widetilde{\cos}(x) = \begin{cases} 1 & x > \pi \\ (\cos(x - \pi) + 1)/2 & 0 \leq x \leq \pi \\ 0 & x < 0 \end{cases}$$

Auf jedem endlichen Intervall kann die Funktion $\cos$ durch Aneinandersetzen von einigen $\widetilde{\cos}$ dargestellt werden, genauer gilt für $x \in [-(n-1)2\pi, (n-1)2\pi]$

$$\cos(x) = \left(2\left(\sum_{i=-n}^{n}(\widetilde{\cos}(x+\pi+i2\pi)+\widetilde{\cos}(-x+\pi+i2\pi))\right)-2(2n+1)-1\right)(x).$$

Man ist nur an Eingaben aus I interessiert, verändert also die Approximation von f im relevanten Bereich nicht, wenn man $\cos$ durch obigen Term ersetzt für genügend großes n. Das ergibt ein Netz mit der Aktivierungsfunktion $\widetilde{\cos}$ in der hiddenSchicht, da alle linearen Terme durch zusätzliche Neuronen und Gewichtsänderungen dargestellt werden können.

Als nächstes ersetzt man $\widetilde{\cos}$ durch eine squashing Funktion σ, ohne die Approximation um mehr als $\epsilon/2$ zu ändern. Dazu wählt man ϵ', so daß eine Änderung jeder Aktivierungsfunktion um ϵ' die Ausgabe nicht mehr als $\epsilon/2$ beeinflußt. Dieses hängt von der Anzahl der Neuronen und der Größe der Gewichte ab. Wähle Q mit $1/Q < \epsilon'/2$, M mit $\sigma(-M) < \epsilon'/(2Q)$, $\sigma(M) > 1-\epsilon/(2Q)$. Letzteres ist für eine squashing Funktion mit Limes 0 bzw. 1 möglich. Sei $r_0 = -\infty$, $r_j = \sup\{x \mid \widetilde{\cos}(x) = j/Q\}$. Sei A_j eine lineare Transformation von $[r_j, r_{j+1}]$ nach $[-M, M]$, im Fall $r_i = \pm\infty$ ist auch letzteres Intervall entsprechend halboffen. Dann ist

$$\widetilde{\cos}(x) \approx \sum_{j=0}^{Q-1}\frac{1}{Q}\sigma(A_j(x)) .$$

Man kann also alle Terme $\widetilde{\cos}$ durch obigen Ausdruck ersetzen, so daß lediglich weitere hidden Neuronen und Gewichtsänderungen dazukommen. □

Interessant ist, daß man also auch mit einer nicht stetigen Aktivierungsfunktion wie der Perzeptronaktivierung stetige Funktionen approximieren kann. Möchte man in der Ausgabe die lineare Aktivierung durch σ ersetzen, dann ist das ebenfalls möglich, sofern nur der Wertebereich der zu approximierenden Funktion geeignet eingeschränkt ist. Für $\sigma = \mathrm{sgd}$ kann dieser $[0,1]$ sein, man approximiert dann statt f zunächst $\mathrm{sgd}^{-1}(\lambda f)$ mit einem Faktor $\lambda \approx 1$.

Wieviel Neuronen benötigt man in dieser Schicht? In der Regel hängt das von der Glattheit der Approximation ab. Mit Methoden der Funktionalanalysis konnte Barron z.B. zeigen, daß der Approximationsfehler mit $1/\sqrt{N}$, ($N =$ Anzahl Neuronen) skaliert, sofern die zu approximierende Funktion glatt ist (genauer: Die Norm von g sollte im relevanten Bereich durch einen in die Abschätzungsgüte einfließenden Faktor beschränkt sein.) Interessant ist folgendes Resultat, daß die Anzahl der Neuronen für eine konkrete Trainingsmenge beschränkt: Man benötigt für k Punkte maximal k hidden Neuronen bei eindimensionaler Ausgabe.

Satz 3.4 *Sei $F : \mathbb{R}^m \to \mathbb{R}$ eine Funktionenklasse. Gelte: Für alle Punkte $(x_1, y_1), \ldots, (x_k, y_k)$ in $\mathbb{R}^m \times \mathbb{R}$ und $\epsilon > 0$ existieren Funktionen f_j aus F und Gewichte w_j mit $|\sum_j w_j f_{i_j}(x_i) - y_i| < \epsilon$ für alle i. Dann findet man k Funktionen f_{i_j} und Gewichte w_j mit*

$$\sum_{j=1}^{k} w_j f_{i_j}(x_i) = y_i \quad \forall i .$$

Beweis: Wähle $\epsilon > 0$, so daß für jede Matrix C in $\mathbb{R}^{k\times k}$ gilt:

$$|C - I| < \epsilon \Rightarrow C \text{ ist invertierbar.}$$

Dabei bezeichnet $|C|$ die Norm $\max|c_{ij}|$. I sei die Identitätsmatrix. Definiere

$$\Phi(i_1, \ldots, i_l) := \begin{pmatrix} f_{i_1}(x_1) & \ldots & f_{i_l}(x_1) \\ \vdots & & \vdots \\ f_{i_1}(x_k) & \ldots & f_{i_l}(x_k) \end{pmatrix} \in \mathbb{R}^{k\times l}$$

Betrachte das Problem

$$f(x_1) = 1, \quad f(x_j) = 0 \quad \forall j \neq 1 .$$

Dieses sei mit $f = \sum_{j=1}^{l} w_j f_{i_j}$ und Toleranz ϵ approximiert. Es ist also $\Phi(i_1, \ldots, i_l)W_1$ mit der durch die Gewichte w_i gegebenen Spalte W_1 weniger als ϵ entfernt von der ersten Spalte der Identitätsmatrix. Diese Prozedur wird mit

$$f(x_i) = 1, \quad f(x_j) = 0 \quad \forall j \neq i$$

wiederholt. Man erhält so, wenn man die zusätzlichen Koeffizienten in den schon existierenden W_i mit 0 auffüllt, eine Matrix mit Spalten $W_1, W_2, \ldots$, so daß

$$\Phi(i_1, \ldots, i_l)(W_1, \ldots, W_k)$$

weniger als ϵ von der Identität abweicht. $\Phi(i_1, \ldots, i_l)$ hat also den Rang k. k linear unabghängige Spalten führen zur invertierbaren Matrix

$$V = \Phi(i_{j_1}, \ldots, i_{j_k}).$$

Man kann also für die Lösung des Ausgangsproblems einfach das Gleichungssystem

$$VW = (y_1, \ldots, y_k)^t$$

lösen. □

Dieses Ergebnis beschränkt den Suchraum für die Architektur in einem konkreten Trainingsproblem: Es reichen ein bis zwei verborgene Schichten mit maximal derselben Anzahl an Neuronen, wie Pattern vorliegen (multipliziert mit der Ausgabedimension).

3.8 Komplexität

Betrachtet man die Komplexität der einzelnen Verfahren, dann unterteilt diese sich in zweierlei: die Komplexität für einen einzelnen Schleifendurchlauf – dank Backpropagation oder heuristischen Näherungen ist diese zumeist linear oder quadratisch in der Anzahl der Gewichte – und die Anzahl der nötigen Schleifendurchläufe. Für letzteres gibt es nur heuristische Betrachtungen. Zumindest kann man zeigen, daß bei gewissen (allerdings den im Allgemeinen in der Praxis langsamen) Lernverfahren und günstigen Startbedingungen unter gewissen Bedingungen Konvergenz in geeignetem Sinne eintritt. Diese Formulierung deutet schon an, daß die Realität damit nicht unbedingt erfaßt ist – so schöne Aussagen wie beim Perzeptrontraining existieren hier (noch) nicht.

Für Differenzenverfahren, welches einfaches Batch-Backpropagation ja ist, gibt es etwa folgende Beobachtung:

Satz 3.5 *Sei w Fixpunkt des Differenzenverfahrens der Form*

$$w(n+1) = w(n) - \eta \cdot f(w(t)),$$

so daß f zweimal stetig differenzierbar und die Jakobimatrix $Jf(w)$ positive definit ist, dann konvergiert das Verfahren für hinreichend kleine Schrittweite η und Startpunkte, die hinreichend nahe bei w liegen.

Beweis: Die Jakobimatrix sammelt einfach nur alle partiellen Ableitungen des Funktionenvektors f auf, positiv definit bedeutet, daß $x^tJf(w)x > 0$ für alle Vektoren $x \neq \mathbf{0}$ gilt.

Jeden Punkt $w(n)$ kann man als $w + tx$ darstellen mit einem Vektor x der Länge 1. Wir möchten sehen, daß $w(n)$ tendentiell zu w driftet, dazu reicht es, zu zeigen, daß obiges t immer kleiner wird. Wir verwenden eine Taylorentwicklung um w. Dabei sei Hf_i die Hessematrix der $i$ten Komponentenfuntkion von f.

$$\begin{aligned}
|w(n+1) - w(n)|^2 &= |w(n) - \eta f(w(n)) - w|^2 \\
&= |tx - \eta f(w + tx)|^2 \\
&\approx |tx - \eta f(w) - \eta t Jf(w)x - \eta t^2/2(x^t Hf_i(\tilde{w})x)_i|^2 \\
&= t^2 + t^2\eta^2 \underbrace{|Jf(w)x|^2}_{\leq C_1} - 2\eta t^2 \underbrace{x^t Jf(w)x}_{\geq C_2} + 2t^3\eta \underbrace{\text{Rest}}_{\leq C_3} \\
&= (*)
\end{aligned}$$

mit einem Faktor Rest und Konstanten C_1, C_2, C_3. C_2 gibt es nach Voraussetzung, die anderen Konstanten existieren aufgrund der Stetigkeit von f, sofern man sich mit $w(n)$ in Kompakta bewegt. Wähle η mit $2C_2 - \eta C_1 \geq \epsilon > 0$, wähle t mit $2C_3 t < \epsilon/2$. Dann gilt

$$\begin{aligned}
(*) &\leq t^2(1 + \eta^2 C_1 - 2\eta C_2 + 2t\eta C_3) \\
&= t^2(1 - \eta(2C_2 - \eta C_1 - 2tC_3)) \\
&\leq t^2(1 - \eta(\epsilon - \epsilon/2)) \\
&= |w(n) - w|^2 \underbrace{(1 - \eta\epsilon/2)}_{=K\leq 1}
\end{aligned}$$

Es folgt

$$|w(n) - w|^2 \leq |w(n-1) - w|^2 K \leq \ldots \leq K^n \to 0.$$

Für genügend kleine Schrittweite η, die durch die Jakobimatrix bestimmt werden kann, und genügend kleine Umgebung, deren Größe von der Güte der Taylorapproximation abhängt, streben also Anfangswerte gegen w. □

Für lokale Optima w einer Fehlerfunktion gilt häufig, daß die Hessematrix an der Stelle w positiv definit ist, daher konvergiert ein Gradientenabstieg in obigem Sinne. Die Funktion f aus obigem Satz ist bei einem Gradientenabstieg die Funktion ∇E. Die Schnelligkeit der Konvergenz läßt sich aus den Gegebenheiten in der konkreten Funktion abschätzen, der Faktor K^n in obigem Beweis ist da sehr vielversprechend – allerdings gilt das nur für einen eventuell sehr kleinen Bereich um das Optimum. Hornik et.al. haben gezeigt, daß ähnliche Aussagen auch unter geeigneten Bedingungen für Online Backpropagation gelten, das ja kein wirklicher, sondern ein sogenannter statistischer Gradientenabstieg ist.

Desweiteren kann man durch eine analoge Rechnung sehen, daß ein Punkt w mit $x^tJf(w)x < 0$ gilt für gewisse Richtungen x instabil ist, sofern man sich aus diesen Richtungen gegen das Extremum w nähert. Kleine Auslenkungen von w in diese Richtungen führen im nächsten Schritt zunächst von w weg.

Es stellt sich aber die Frage, was bei einem beliebigem Startpunkt passiert und ob und wie man lokale Minima vermeiden kann. Globale negative Aussagen erhält man, sofern Situationen als NP-schwierig nachgewiesen werden können. Hier stellt sich, genau wie beim einfachen Perzeptrontraining auch, das Training eines einfachen sigmoiden Neurons als schwierig heraus. Allerdings benötigt man schon hier etwas diffizilere Argumentationen als einfache Standardreduktionen.

Satz 3.6 *Es ist NP-schwierig zu entscheiden, ob eine vorgegebene Trainingsmenge aus* $\{0,1\}^n \times \{0,1\}$ *mit einem Netz* $(n,1)$ *mit sigmoider Aktivierung und quadratischem Fehler* $\leq k$ *trainierbar ist.* k *und* n *sind dabei variabel.*

Beweis: Das Hitting Set Problem hat folgende Eigenschaft: Gegeben eine Boolesche Formel φ in konjunktiver Normalform und eine Konstante $c > 1$. Dann findet man in polynomieller Zeit ein Hitting Set Problem und eine Konstante K, so daß, falls φ erfüllbar ist, es ein Hitting Set der Größe K gibt, falls φ nicht erfüllbar ist, jedes Hitting Set mindestens die Größe cK hat.

Sei jetzt eine Formel φ gegeben. Reduziere diese zu einem Hitting Set Problem mit obiger Eigenschaft und Konstante K zum Faktor $c = 5$. Dieses Hitting Set Problem kann zu einem Loading Problem für ein einfaches Perzeptron reduziert werden, das mit K Fehlern lösbar ist, falls φ erfüllbar ist, und sonst mindestens $5K$ Fehler macht, wie wir schon gesehen haben.

Betrachte jetzt dieselbe Trainingsmenge, die mit einem sigmoiden Netz $(n,1)$ und quadratischem Fehler maximal $K+1$ trainiert werden soll. Falls φ erfüllbar ist, kann man ein Perzeptron mit nur K Fehlern finden. Dasselbe Netz mit sigmoider Aktivierung hat für große Gewichte asymptotisch den quadratischen Fehler K, für geeignet skalierte Gewichte maximal den Fehler $K+1$. Ist umgekehrt eine Lösung mit quadratischem Fehler $K+1$ gegeben, dann klassifiziert dasselbe Netz mit Perzeptronaktivierung maximal $4(K+1)$ Punkte falsch, da jeder falsch klassifizierte Punkt mindestens 0.5^2 zum quadratischen Fehler beiträgt. Das ist aber kleiner als $5K$, d.h. das Perzeptron macht weniger als $5K$ Fehler, also ist φ erfüllbar. □

Man würde jetzt erwarten, daß sich auch alle anderen Ergebnisse vom Perzeptronfall auf die – scheinbar – kompliziertere Situation der sigmoiden Aktivierungsfunktion übertragen. Mathematisch ist das allerdings nicht ganz so offensichtlich, da sigmoide Netze eine größere Mächtigkeit haben, die die Suche nach Lösungen evtl. einfacher machen könnte. [Es gibt Beispiele, wo das der Fall ist: Etwa Lernen einer konjunktiven Normalform mit n Variablen und k Literalen pro Disjunktion ist NP-schwierig, Lernen einer disjunktiven Normalform mit n Variablen und k Literalen pro Konjunktion aber nicht, obwohl letztere Formeln alle ersteren darstellen können.] Tatsächlich erweisen sich die Beweise in diesem Gebiet als äußerst schwierig – zufriedenstellende Aussagen für die Komplexität des Trainings von sigmoiden Netzen zu finden, ist aktuelles Forschungsgebiet. Es gibt bisher keine Aussagen für Netze mit mehr als nur einer verborgenen Schicht. Für Netze mit einer verborgenen Schicht gibt es etwa die Aussage [Hammer]:

Satz 3.7 *Seien* $B > 2$, $\epsilon < 0.5$ *feste positive Zahlen. Es ist NP-schwierig zu entscheiden, ob eine Patternmenge* P *mit einem Netz* $(n,2,1)$ *mit sigmoiden Knoten in der verborgenen Schicht, Perzeptronaktivierung in der Ausgabe, betragsmäßig durch* B *nach oben beschränkten Ausgabegewichten und betragsmäßig durch* ϵ *nach unten beschränkter Mindestaktivierung der Ausgabeeinheit auf allen Trainingspattern trainiert werden kann.* n *ist dabei variabel.*

Schwierig macht es die Situation, daß hier Netze zur Klassifikation eingesetzt werden. Betrachtet man stattdessen Netze, die Interpolieren sollen, d.h. die Aufgabe, den quadratischen Fehler zu minimieren auf einer Patternmenge mit reellwertigen statt nur binärwertigen Zielvorgaben, dann kann man die Funktionsweise des Netzes durch die vorgegebenen Werte festlegen und quasi das Problem des zwei Knoten Perzeptronfalls als Teilaufgabe erzwingen. Ebenso schwierige Beweise, wie es der Beweis zu obigem Satz wäre, belegen die Komplexität des Trainings von sigmoiden Netzen mit einer verborgenen Schicht, so daß der quadratische Fehler kleiner als eine vorgegebene von der Anzahl der hidden Neuronen abhängigen Größe wird [Jones (für den zwei Knoten Fall), Vu (für den allgemeinen Fall, allerdings unleserlich)].

In der Praxis versucht man, diesen Problemen durch eine geeignete (z.B. unäre) Repräsentation der Daten, an die Problemgröße adaptierte Architekturen und die Möglichkeit der Architekturänderung auch während des Trainings zu begegnen. Allerdings stoßen auch Methoden wie Eingabepruning schnell an ihre Grenzen. Unabhängig davon, ob man mit neuronalen Netzen oder einem anderen Mechanismus lernt, erhält man nämlich die Aussage: ‚Eingabepruning ist NP-schwierig' oder genauer:

Satz 3.8 *Gegeben sei n, eine endliche Menge von Punkten $(\mathbf{x}_i, y_i) \in \{0,1\}^n \times \{0,1\}$ und eine Zahl k. Dann ist es NP-hart zu entscheiden, ob es k Indizes $i_1, \ldots, i_k$ gibt, so daß die auf diese Koeffizienten reduzierten Punkte nicht widersprüchlich werden, d.h. keine $\mathbf{x}_i$ und $\mathbf{x}_j$ existieren mit $(x_{ii_1}, \ldots, x_{ii_k}) = (x_{ji_1}, \ldots, x_{ji_k})$ und $y_i \neq y_j$.*

Beweis: Dieses folgt sofort durch eine Reduktion vom hitting set Problem. Eine Instanz von Punkten $S = \{s_, \ldots, s_n\}$ und Teilmengen $C = \{c_1, \ldots, c_m\}$ besitzt ein hitting set der Größe k dann und nur dann wenn folgende Punkte mit Eingaben im $\mathbb{R}^n$ sich auf k Eingabekoeffizienten reduzieren lassen, ohne widersprüchlich zu werden:

$$\begin{aligned}&(0, \ldots, 0) \mapsto 0,\\ &\mathbf{e}_{c_j} = (0, \ldots, 1, \ldots, 1, \ldots, 1, \ldots, 0) \mapsto 1 \quad \text{für alle } c_j\end{aligned}$$

wobei der i te Koeffzient von $\mathbf{e}_{c_j}$ 1 ist dann und nur dann, wenn $s_i \in c_j$ gilt. □

4 Exkurs in die COLT-Theorie

Das allgemeine Vorgehen ist also so, daß man eine geeignete Architektur auswählt und die Parameter anhand der Daten optimiert. Es ist offensichtlich, daß der empirische Fehler in der Regel kleiner ist als der Generalisierungsfehler. Daher wählt man als Gütekriterium eine Schätzung des Generalisierungsfehlers auf Daten, auf denen nicht gelernt wurde. Hier stellt sich jetzt die Frage, welchen Grund man zur Annahme hat, ein kleiner empirischer Fehler führe auch zu einem kleinen Generalisierungsfehler. Schärfer formuliert: Warum ist sichergestellt, daß die Daten genügend über das zugrundeliegende Modell aussagen, so daß man – möglichst nach einer vorher abschätzbaren Menge an Beispielen – aus Daten lernen kann. Diese Frage ist nicht auf neuronale Netze beschränkt, sondern stellt sich im Zusammenhang jedes Algorithmus, der aus Beispielen lernt. Eine mathematische Präzisierung von ‚Lernbarkeit' bietet die statistische Lernbarkeitstheorie, von der wir jetzt einige grundlegende Ideen erläutern. Die Abkürzung COLT steht dabei für COmputational Learning Theorie.

4.1 PAC Lernbarkeit

Die mathematischen Gegebenheiten seien charakterisiert durch

- einen Datenraum X mit einer Verteilung P, gemäß der Beispiele x_i unabhängig und identisch verteilt (kurz i.i.d.) gezogen werden,
- einen Bildraum Y, der hier immer in $[0, 1]$ enthalten sei,
- eine approximierende Funktionenklasse $\mathcal{F}$ von Funktionen $g : X \to Y$, die etwa durch eine Netzarchitektur gegeben sein kann,
- eine unbekannte zu lernende Funktion f, von der wir annehmen, daß sie auch in $\mathcal{F}$ enthalten ist.

Die letzte Forderung ist nicht unbedingt realistisch, denn häufig ist die Funktion f fehlerbehaftet, so daß man es mit $f + \eta$ statt f zu tun hat; häufig ist f zudem komplexer als die Funktionen in $\mathcal{F}$ und nur ganz gut durch diese approximierbar. Diese allgemeineren Fragestellungen fallen unter den Terminus des **agnostischen Lernens**, der von Haussler eingeführt wurde. Er ist etwas schwieriger zu handhaben, aber viele der Ideen sind dieselben. Insbesondere sind hinreichende Bedingungen für agnostische Lernbarkeit unter realistischen Bedingungen auch durch eine endliche Überdeckungszahl bzw. VC-Dimension von $\mathcal{F}$ gegeben, das heißt durch die Charakteristika, die wir auch in dieser einfacheren Situation erhalten werden. In allem Folgendem muß von mathematischer Warte aus darauf geachtet werden, daß die betrachteten Funktionen und Mengen meßbar sind. Das ist in konkreten Fällen gegeben, daher gehen wir darauf nicht weiter ein.

Ein **Lernalgorithmus** lernt anhand von Daten eine Funktion, das heißt ein Lernalgorithmus ist eine Abbildung

$$h : \bigcup_{i=1}^{\infty} (X \times Y)^i \to \mathcal{F}.$$

Wir schreiben $h_m(\mathbf{x}, \mathbf{y})$ für $h(x_1, y_1, \ldots, x_m, y_m)$ und $h_m(\mathbf{x}, f)$ für $h(x_1, f(x_1), \ldots, x_m, f(x_m))$. Der **empirische Fehler** zwischen zwei Funktionen auf $\mathbf{x}$ ist

$$\hat{d}_m(f, g, \mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} |f(x_i) - g(x_i)|.$$

Der **tatsächliche Fehler** zwischen zwei Funktionen ist

$$d_P(f, g) = \int |f(x) - g(x)| d_P(x),$$

wobei die Notation $d_P(x)$ bedeutet, daß man entsprechend der Verteilung P über die Unterschiede auf allen $x \in X$ mittelt. Es wird jetzt in der Regel gelten daß

$$\hat{d}_m(h_m(\mathbf{x}, f), f) \approx 0$$

gilt. Die Hoffnung ist allerdings, daß auch

$$d(h_m(\mathbf{x}, f), f) \approx 0$$

zumindest für genügend großes m gilt. Ferner würde man gerne eine Schranke für die Anzahl der Beispiele m wissen, so daß der tatsächliche Fehler höchstwahrscheinlich etwa dem empirischen Fehler entspricht. Diese Schranke sollte dabei insbesondere unabhängig von der unbekannten zu lernenden Funktion f sein!

Es wird sich herausstellen, daß Charakteristika der Funktionenklasse $\mathcal{F}$ dieses garantieren können. Für beliebiges $\mathcal{F}$ ist diese Eigenschaft aber nicht gegeben.

Beispiel: Sei $\mathcal{F}$ die Klasse

$$\{f : [0, 1] \to \{0, 1\} \mid \exists t \in \mathbb{R} \quad \mathrm{H}(\cos(tx)) = f(x)\}$$

der Funktionen, deren Verlauf durch Nullstellen einer Cosinusfunkiton bestimmt wird. Diese Klasse kann durch einen reellen Parameter beschrieben werden. Nichtsdestotrotz ist sie schwierig zu lernen. Anschaulich kann man sich das wie folgt klar machen: Für jede Folge von Punkten $x_1, \ldots, x_m$ im Intervall $[0, 1]$, die nicht rational abhängig sind (d.h. die nicht durch eine Linearkombination mit ganzen Zahlen zu 0 kombiniert werden können), ist die Menge

$$\{(tx_1 \mathrm{mod} 2\pi, \ldots, tx_n \mathrm{mod} 2\pi) \mid t \in \mathbb{R}\}$$

dicht in $[0, 2\pi]^m$, d.h. allein durch Skalieren des Vektors kann man die Punkte in nahezu jede Position auf dem Definitionsbereich von $\cos$ setzen, wenn man die Periodizität berücksichtigt (vgl. z.B. [Mane, Ergodic Theory and Differentiable Dynamics]). Die Punkte können also durch geeignete Wahl des Parameters t beliebig nach $\{0, 1\}$ abgebildet werden. Möchte man also eine dieser verschiedenen Funktionen aufgrund der Punkte identifizieren, dann muß man den Wert auf allen m Punkte betrachten. Das heißt aber, da man dieses für beliebiges m erreichen kann, daß man zur Identifikation gewisser Funtkionen beliebig viele Punkte benötigt. Das ist natürlich kein formaler Beweis dafür, daß Lernen schwer ist, wir werden diesen später bekommen.

Zunächst wollen wir formal fassen, wann eine Funktionenklasse lernbar ist.

Definition 4.1 *Eine Funktionenklasse heißt* **PAC lernbar** *(probably approximately correct learnable), falls es mindestens einen Algorithmus h gibt, so daß für alle ϵ*

$$\sup_{f\in\mathcal{F}} P^m(\mathbf{x} \mid d_P(h_m(\mathbf{x}, f), f) > \epsilon) \to 0 \quad (m \to \infty)$$

gilt. Ist ein konkretes ϵ angesprochen, so heißt dieses Genauigkeit. Ist obiger Term explizit durch δ beschränkt, so heißt dieses Konfidenz.

Eine Funktionenklasse heißt **verteilungsunabhängig PAC lernbar***, falls es mindestens einen Algorithmus h gibt, so daß für alle ϵ*

$$\sup_P \sup_{f\in\mathcal{F}} P^m(\mathbf{x} \mid d_P(h_m(\mathbf{x}, f), f) > \epsilon) \to 0 \quad (m \to \infty)$$

gilt.

Eine Funktionenklasse ist **UCED** *(uniform convergence of empirical distances), falls für alle $\epsilon > 0$*

$$P^m(\mathbf{x} \mid \sup_{f,g\in\mathcal{F}} |\hat{d}_m(f, g, x) - d_P(f, g)| > \epsilon) \to 0 \quad (m \to \infty).$$

Eine Funktionenklasse ist **verteilungsunabhängig UCED***, falls*

$$\sup_P P^m(\mathbf{x} \mid \sup_{f,g\in\mathcal{F}} |\hat{d}_m(f, g, x) - d_P(f, g)| > \epsilon) \to 0 \quad (m \to \infty).$$

Der Begriff der PAC Lernbarkeit wurde von Valiant eingefürt. Original faßte er auch Effizienzaussagen, die wir hier entkoppelt haben. PAC Lernbarkeit ist eine Mindestanforderung, so daß man von der unbekannten Funktion unabhängige Schranken für die Anzahl der Beispiele finden kann, die Generalisierung gewährleistet. Es ist klar, daß man nicht notwendig PAC Lernbarkeit für alle $\epsilon > 0$, sondern je nach Gegebenheit etwa nur für ein bestimmtes ϵ verlangen muß. Verteilungsunabhängige PAC Lernbarkeit fordert darüberhinaus eine Unabhängigkeit der Schranken von der den Daten zugrundeliegenden, evtl. unbekannten Verteilung. Beide Formulierungen sichern lediglich die Existenz mindestens eines guten Algorithmus.

Die UCED Eigenschaft hingegen stellt sicher, daß der empirische Fehler jedes Algorithmus eine aussagekräftige Größe darstellt, denn der empirische Fehler des Algorithmus weicht nicht sehr vom tatsächlichen ab für genügend großes m. In der Definition taucht der Algorithmus h nicht explizit auf, sondern ist implizit, da ja h nur eine spezielle Funktion g darstellt, umgekehrt aber auch jede spezielle Funktion g Ausgabe des Algorithmus sein könnte. Insbesondere ist bei einer Funktionenklasse, die UCED ist, jeder Lernalgorithmus mit kleinem empirischem Fehler gut.

Die Frage stellt sich jetzt natürlich, wie man diese einzelnen Bedingungen testen kann. Wir fangen mit einer ganz einfachen Situation an:

Satz 4.2 *Die Funktionenklasse $\mathcal{F}$ sei endlich. Dann ist $\mathcal{F}$ verteilungsunabhängig PAC.*

Beweis: Sei $\mathcal{F} = \{f_1, \ldots, f_n\}$ und $X_{ij} = \{x \mid f_i(x) \neq f_j(x)\}$. Auf der Menge X_{ij} unterscheiden sich also f_i und f_j. Möchte man nur mit Genauigkeit ϵ lernen, dann kann man $P(X_{ij}) \geq \epsilon$ annehmen. Ansonsten sind nÄmlich beide Funktionen bei der hier geforderten Genauigkeit gleich. Definiere einen Lernalgorithmus h daurch, daß h auf den Daten irgendeine Funktion aus $\mathcal{F}$ wählt, die keinen empirischen Fehler macht. Da die zu lernende Funktion in $\mathcal{F}$ ist, gibt es so eine Funktion. Dann ist

$$\begin{array}{ll} & \sup_{f_i} P^m(\mathbf{x} \mid d_P(f_i, h_m(\mathbf{x}, f_i)) > \epsilon) \\ \leq & \sum_i P^m(\mathbf{x} \mid d_P(f_i, h_m(\mathbf{x}, f_i)) > \epsilon) \\ \leq & \sum_i P^m(\mathbf{x} \mid \exists j \text{ es wurde kein Beispiel aus } X_{ij} \text{ gezogen}) \\ \leq & \sum_i \sum_j P^m(\mathbf{x} \mid \text{es wurde kein Beispiel aus } X_{ij} \text{ gezogen}) \\ \leq & \sum_i \sum_j (1 - P(X_{ij}))^m \\ \leq & n^2(1-\epsilon)^m \to 0 \quad (m \to \infty) \end{array}$$

Da diese Abschätzung für jede Verteilung P gilt, ist sie auch für das Supremum korrekt. Setzt man in der letzten Zeile das Minimum über $P(X_{ij})$ statt ϵ ein, erhält man eine Abschätzung für beliebige Genauigkeit. □

Endliche Funktionenklassen sind also PAC lernbar. Insbesondere gilt dieses etwa für Funktionenklassen auf binären Daten. In der Regel hat man es jedoch mit unendlichen Funktionenklassen zu tun. Was ist dann? Im Hinblick darauf, daß man die Funktionen nur bis auf den Faktor ϵ lernen muß, charakterisiert folgender Begriff ein Äquivalent zur Endlichkeit:

Definition 4.3 *Sei F eine Menge mit Pseudometrik d. Dann ist die* **Überdeckungszahl** $N(\epsilon, F, d)$ *die kleinste Kardinalität von Punkten $x_1, \ldots, x_l$, so daß die Menge $\bigcup_i \{x \mid d(x, x_i) \leq \epsilon\}$ ganz F überdeckt. Die* **Packzahl** $M(\epsilon, F, d)$ *ist die größte Kardinalität von Punkten, so daß $d(x_i, x_j) > \epsilon$ für $i \neq j$ gilt.*

Die Packzahl wird aus technischen Gründen benötigt. Wegen der Abschätzung

$$M(2\epsilon, F, d) \leq N(\epsilon, F, d) \leq M(\epsilon, F, d)$$

sind die beiden Begriffe im Wesentlichen austauschbar.

In unserem Fall ist F die Funktionenklasse $\mathcal{F}$, ϵ wird sich aus der Genauigkeit des Lernalgorithmus ergeben und d ist die Pseudometrik d_P, die den Abstand zwischen zwei Funktionen mißt. Dieses ist keine wirkliche Metrik wie etwa der euklidische Abstand, da verschiedene Funktionen durchaus den Abstand 0 bzgl. d_P haben können, wenn sie sich nur auf einer Menge mit Wahrscheinlichkeit 0 unterscheiden.

Satz 4.4 *Falls $N(\epsilon, \mathcal{F}, d_P)$ endlich ist, ist $\mathcal{F}$ mit Genauigkeit 2ϵ PAC lernbar. Falls $N(\epsilon, \mathcal{F}, d_P)$ für alle $\epsilon > 0$ endlich ist, ist $\mathcal{F}$ PAC lernbar.*

Beweis: Es wird der sogenannte Minimum Risk Algorithmus konstruiert: Wähle eine ϵ-Überdeckung $f_1, \ldots, f_k$ von $\mathcal{F}$. Für gegebene Daten $(x_i, f(x_i))_{i=1}^m$ wähle dasjenige f_i als Ausgabe, das den minimalen empirischen Fehler hat.

Dieser Algorithmus ist PAC mit Genauigkeit 2ϵ: Sei dazu f zu lernen, evtl. nach Umnummerieren $d_P(f, f_i) > 2\epsilon$ für $i = 1, \ldots, l$, $d_P(f, f_i) \leq 2\epsilon$ für $i = l+1, \ldots, k$, $d_P(f, f_k) \leq \epsilon$. Es ist

$$\begin{array}{ll} & P^m(\mathbf{x} \mid d_P(f, h_m(\mathbf{x}, f)) > 2\epsilon) \\ \leq & P^m(\mathbf{x} \mid \hat{d}_m(f, f_k, \mathbf{x}) > 3\epsilon/2 \vee \exists i \in \{1, \ldots, l\}\ \hat{d}_m(f, f_i, \mathbf{x}) \leq 3\epsilon/2) \\ \leq & P^m(\mathbf{x} \mid \hat{d}_m(f, f_k, \mathbf{x}) > 3\epsilon/2) + \sum_i P^m(\mathbf{x} \mid \hat{d}_m(f, f_i, \mathbf{x}) \leq 3\epsilon/2) \\ \leq & \mathrm{e}^{-m\epsilon^2/8}(l+1) \leq k\mathrm{e}^{-m\epsilon^2/8} \to 0 \end{array}$$

Dabei wurde in der letzten Zeile die sogenannte **Hoeffding Ungleichung** benutzt:

$$P(\sum_{i=1}^{m} X_i \geq \alpha) \leq \mathrm{e}^{-\alpha^2/(2m)}$$

falls X_i i.i.d. aus $[-1,1]$ mit Erwartungswert 0 stammen. Wählt man jetzt für X_i die Zufallsvariable $|f(x_i) - f_j(x_i)| - d_P(f,f_j)$, dann erhält man

$$\begin{array}{ll} & P^m(\mathbf{x} \mid \hat{d}_P(f,f_k) > 3\epsilon/2) \\ \leq & P^m(\mathbf{x} \mid \sum (|f(x_i) - f_k(x_i)| - d_P(f,f_k)) > \epsilon m/2) \\ \leq & \mathrm{e}^{-\epsilon^2 m/8} \end{array}$$

da ja $d_P(f,f_k) \leq \epsilon$ gilt. Ferner ist für $j \leq l$

$$\begin{array}{ll} & P^m(\mathbf{x} \mid \hat{d}_P(f,f_j) \leq 3\epsilon/2) \\ \leq & P^m(\mathbf{x} \mid -\sum (|f(x_i) - f_j(x_i)| + d_P(f,f_j)) > \epsilon m/2) \\ \leq & \mathrm{e}^{-\epsilon^2 m/8} \end{array}$$

da $d_P(f,f_j) > 2\epsilon$ gilt. Insbesondere ist die Konfidenz maximal δ für

$$m \geq \frac{8}{\epsilon^2} \ln \frac{k}{\delta}\,.$$

Falls die Überdeckungszahl für jedes ϵ endlich ist, erhält man für jede Genauigkeit einen gesonderten Algorithmus, die man jetzt noch zu einem einzigen Algorithmus zusammenfassen muß: Für l besitze der Algorithmus zur Genauigkeit $1/l$ die Konfidenz $1/l$ auf Eingaben von mindestens m_l Eingabedaten. Dabei ist o.B.d.A. $m_{l+1} > m_l$. Man ruft jetzt einfach für m Eingabedaten den Algorithmus zur Genauigkeit $1/l$ für dasjenige l mit dem größten $m_l \leq m$ auf. □

Zur Anmerkung: Die Hoeffding-Ungleichung besagt allgemeiner für unabhängige Zufallsvariablen $X_i \in [a_i, b_i]$:

$$P(\sum X_i \geq \alpha) \leq \mathrm{e}^{-2\alpha^2/\sum(b_i-a_i)^2}\,.$$

Wir bemerken noch einmal, der Minimum Risk Algorithmus benötigt also maximal

$$m \geq \frac{8}{\epsilon^2} \ln \frac{N(\epsilon,\mathcal{F},d_P)}{\delta}$$

Muster zur Genauigkeit 2ϵ und Konfidenz δ. Dieses liefert jetzt eine hinreichende Bedingung für PAC Lernbarkeit, ist sie auch notwendig? Wir unterscheiden zwischen Funktionenklassen und **Konzepten** welche Funktionenklassen mit Werten in $\{0,1\}$ sind.

Satz 4.5 *Für Konzepte $\mathcal{F}$ benötigt jeder Algorithmus mit der Genauigkeit ϵ und Konfidenz δ wenigstens $(1-\delta)\cdot \lg M(2\epsilon,\mathcal{F},d_P)$ Beispiele.*

Beweis: Seien $f_1,\ldots,f_l$ 2ϵ separierte Funktionen. Für jeden Algorithmus h berechnet man

$$\begin{array}{ll} & \sum_i \sum_{\mathbf{y}\in\{0,1\}^m} P^m(\mathbf{x} \mid d_P(f_i, h_m(\mathbf{x},\mathbf{y})) \leq \epsilon) \\ = & \sum_{\mathbf{y}} \int \sum_i 1_{d_P(f_i,h_m(\mathbf{x},\mathbf{y}))\leq\epsilon)}(\mathbf{x},\mathbf{y}) d_{P^m}(\mathbf{x}) \\ \leq & \sum_{\mathbf{y}} 1 = 2^m, \end{array}$$

da die Funktionen den Abstand 2ϵ voneinander haben. Andererseits ist aber für einen PAC Algorithmus

$$\begin{array}{ll} & \sum_{\mathbf{y}\in\{0,1\}^m} P^m(\mathbf{x} \mid d_P(f, h_m(\mathbf{x},\mathbf{y})) \leq \epsilon) \\ \geq & P^m(\mathbf{x} \mid d_P(f, h_m(\mathbf{x}, f)) \leq \epsilon) \\ \geq & 1-\delta, \end{array}$$

also

$$2^m \geq (1-\delta) M(2\epsilon, \mathcal{F}, d_P)\,.$$

□

Jeder noch so komplizierte Algorithmus benötigt also eine durch den Logarithmus der Überdeckungszahl gegebene Anzahl an Beispielen, sofern man mit Konzepten umgeht. Für Funktionenklassen gilt eine analoge Aussage, sofern die Funktionswerte endlich, oder die Daten fehlerbehaftet sind [Vidyasagar, Bartlett et.al.]. Für Daten mit beliebiger Genauigkeit kann dagegen Lernen auf Funktionenklassen mit unendlicher Überdeckundszahl aufgrund von Kodierungstricks möglich sein.

Beispiel: Die Funktionenklasse

$$\left\{ f_i : [0,1] \to [0,1] \mid f_i(0) = 2^{-i}, f_i(x) = \left\{ \begin{array}{ll} 0 & x \in \bigcup_j = 0^{2^i-1}]j/2^i, (j+1)/2^i] \\ 1 & \text{sonst} \end{array} \right. \right\}$$

ist bzgl. der Verteilung mit $P(0) = 1/2$, $P|]0,1] =$ Gleichverteilung PAC lernbar, hat aber unendliche Überdeckungszahl, denn: Der Wert in 0 bestimmt die Funktion eindeutig. Da $P(0) > 0$ ist, ist die Eingabe 0 bei unendlich häufigem Ziehen mit Warscheinlichkeit 1 anzutreffen. Auf dem Rest unterscheiden sich zwei Funktionen je um die Distanz $0.5 \cdot 0.5$.

In diesem Beispiel ist die gesamte Funktion in den Funktionswert der 0 kodiert worden. Solche Kodierungstricks können allerdings in der Praxis aufgrund begrenzter Rechengenauigkeit nicht auftreten, so daß man unter realistischen Bedingungen PAC Lernbarkeit mit endlicher Überdeckungszahl gleichsetzen kann. In diesem Fall hatten wir einen (den Minimum Risk) Algorithmus erhalten, der die Klasse lernt. Im Falle neuronaler Netze würden wir aber gerne eine Form von Backpropagation anwenden. Keine Rücksicht auf den speziellen Algorithmus (außer, daß er den empirischen Fehler minimiert) muß man nehmen, wenn die UCED Eigenschaft nachgewiesen werden kann. Diese zeichnet sich dadurch aus, daß sie, genau wie die Charakterisierung von PAC durch die Überdeckungszahl, nicht auf einen speziellen Algortithmus referiert und also nur aufgrund der Funktionenklasse getestet werden kann. Dennoch kann man noch adäquatere Charakterisierungen von UCED finden. Zunächst soll auch hier gezeigt werden, daß jede endliche Funktionenklasse UCED ist.

Satz 4.6 $\mathcal{F} = \{f_1,\ldots,f_n\}$ *ist UCED.*

Beweis:

$$\begin{array}{ll} & P^m(\mathbf{x} \mid \exists i,j\, |d_P(f_i,f_j) - \hat{d}_m(f_i,f_j,\mathbf{x})| > \epsilon) \\ \leq & \sum_{i<j} P^m(\mathbf{x} \mid |\sum_l (|f_i(x_l) - f_j(x_l)| - d_P(f_i,f_j)| > \epsilon m) \\ \leq & \sum_{i<j} 2\mathrm{e}^{-\epsilon^2 m/2} = n(n-1)\mathrm{e}^{-\epsilon^2 m/2} \to 0 \quad (m \to \infty) \end{array}$$

□

Für unendliche Funktionenklassen würde man gerne eine analoge Abschätzung machen. Dazu muß man für die Summenbildung die Funktionenklasse durch ein endliches Objekt ersetzen. Dieses ist ein wenig trickreich und verlangt zusätzliche Überlegungen.

Satz 4.7 *Eine Funktionenklasse $\mathcal{F}$ ist UCED dann und nur dann, wenn*

$$\lim_{m\to\infty} \frac{E_{P^m}(\lg N(\epsilon, \mathcal{F}|\mathbf{x}, \hat{d}_m))}{m} = 0$$

gilt. Man erhält die explizite Abschätzung

$$P^m(\mathbf{x} \mid \sup_{f,g\in\mathcal{F}} |d_p(f,g) - \hat{d}_m(f,g,\mathbf{x})| > \epsilon) \leq E_{P^{2m}}(2\,N(\epsilon/16, \mathcal{F}|\mathbf{xy}, \hat{d}_{2m})^2)\mathrm{e}^{-m\epsilon^2/32}.$$

Dabei bezeichnet $\mathcal{F}|\mathbf{x}$ die Einschränkung der Funktionenklasse auf die Eingaben $\mathbf{x}$.

Beweis: Ein vollständiger Beweis kann etwa in [Vidyasagar] nachgelesen werden. Es soll lediglich skizziert werden, wie man die Distanz des empirischen zum tatsächlichen Fehler abschätzen kann, da diese Beweismethodik in der Lernbarkeitstheorie häufig in der einen oder anderen Form angetroffen werden kann.

Es wird nicht die UCED Eigenschaft, sondern die sog. UCEM Eigenschaft betrachtet, die die Größe

$$P^m(\mathbf{x} \mid \sup_{f\in\mathcal{F}} |E_P(f) - \hat{E}_m(f,\mathbf{x})| > \epsilon)$$

untersucht. UCEM bedeutet ‚uniform convergence of empirical means'; man beachte, daß hier ja gerade der Erwartungswert mit der empirischen Erwartung verglichen wird. Zur UCED Eigenschaft kommt man, wenn man die UCEM Eigenschaft der Klasse $\triangle\mathcal{F} = \{|f-g| \mid f,g \in \mathcal{F}\}$ untersucht. Die Überdeckungszahl der Klasse $\triangle\mathcal{F}$ zum Parameter ϵ ist maximal quadratisch im Vergleich zur Überdeckungszahl der Klasse $\mathcal{F}$ zum Parameter $\epsilon/2$, daher übertragen sich die Schranken entsprechend.

Sei $m \geq 2/\epsilon^2$. In einem ersten Schritt schätzt man die Abweichung der empirischen von der tatsächlichen Erwartung gegen die Abweichung zweier empirischer Erwartungen voneinander ab. Es sei $\int f(\mathbf{x})p_P(\mathbf{x}) = E_P(f)$, $\sum_i f(x_i)/m = \hat{E}_m(f,\mathbf{x})$. Sei

$$\begin{aligned} Q &:= \{\mathbf{x} \mid \sup\nolimits_{f\in\mathcal{F}} |E_P(f) - \hat{E}_m(f,\mathbf{x})| > \epsilon\}, \\ R &:= \{\mathbf{xy} \mid \sup\nolimits_{f\in\mathcal{F}} |\hat{E}_m(f,\mathbf{y}) - \hat{E}_m(f,\mathbf{x})| > \epsilon/2\}. \end{aligned}$$

Dann ist $P^m(Q) \leq 2P^{2m}(R)$: Mit der **Tschebychev-Ungleichung**, die besagt, daß $P(|X-EX| > \epsilon) < E((X-EX)^2)/\epsilon^2$ für eine Zufallsvariable X gilt, folgert man

$$P^m(\mathbf{x} \mid |\hat{E}_m(f,\mathbf{x}) - E_P(f)| > \epsilon/2) \leq \frac{4}{m\epsilon^2} \leq \frac{1}{2}$$

für $m \geq 8/\epsilon^2$. Für jede Funktion, deren Erwartung von der empirischen Erwartung um ϵ abweicht, weicht die empirische Erwartung also mit Wahrscheinlichkeit mindestens $1/2$ von einer auf einer neuen Menge gemessenen empirischen Erwartung um mindestens $\epsilon/2$ ab. Daher bekommt man

$$\begin{aligned} & P^{2m}(R) \\ \geq\; & P^{2m}(\mathbf{xy} \mid \exists f\,(|\hat{E}_m(f,\mathbf{x}) - E_P(f)| > \epsilon \wedge |\hat{E}_m(f,\mathbf{y}) - E_P(f)| \leq \epsilon/2)) \\ \geq\; & P^m(Q)/2. \end{aligned}$$

Als nächstes kann man $P^{2m}(R)$ abschätzen, indem man sogenannte swapping Permutationen auf den Daten betrachtet. Swapping Permutationen sind Permutationen, die maximal einige Koeffizienten i mit $m+i$ vertauschen, und also zwischen den beiden Samples swappen. Es ist

$$\begin{aligned} P^{2m}(R) &= \tfrac{1}{2^m}\sum\nolimits_\gamma \int_{X^{2m}} 1_R(\gamma\mathbf{z})d_{P^{2m}}(\gamma\mathbf{z}) \\ &= \int_{X^{2m}} \tfrac{1}{2^m}\sum\nolimits_\gamma 1_R(\gamma\mathbf{z})d_{P^{2m}}(\mathbf{z}) = (*), \end{aligned}$$

wobei γ die swapping Permutationen repräsentiert, 1_R ist die charakterischische Funktion zur Menge R.

Für festes $\mathbf{z}$ kann man den Integranden beschränken: Sei durch f_i ($i = 1,\ldots$) eine $\epsilon/8$ Überdeckung der auf $\mathbf{z}$ eingeschränkten Funktionenklasse gegeben. Dann ist $\gamma\mathbf{z}$ in R, falls

$$|\sum_{i=1}^m f(\mathbf{z}_{\gamma(i)}) - f(\mathbf{z}_{\gamma(i+m)})|/m > \epsilon/2.$$

Aufgrund der Überdeckungseigenschaft gibt es dann ein f_j aus der Überdeckung, so daß

$$|\sum_{i=1}^m f_j(\mathbf{z}_{\gamma(i)}) - f_j(\mathbf{z}_{\gamma(i+m)})|/m > \epsilon/4.$$

Für jeden festen Index j kann man aber die Anzahl der swapping Permutationen, für die obige Ungleichung gilt, beschränken. Dazu benutzt man die Hoeffding-Ungleichung für unabhängige Zufallsvariablen X_i mit Werten in $\pm(f_j(\mathbf{z}_{\gamma(i)}) - f_j(\mathbf{z}_{\gamma(i+m)}))$ und erhält die Schranke $2\mathrm{e}^{-2m\epsilon^2/4\cdot 4^2}$ für die Wahrscheinlichkeit einer solchen swapping Permutation. Also ist für jedes feste $\mathbf{z}$ die Anzahl der swapping Permutationen, so daß es ein f_j mit obiger Ungleichung gibt, maximal

$$2^m 2\mathrm{e}^{-m\epsilon^2/32} N(\epsilon/8, \mathcal{F}|\mathbf{z}, \hat{d}_m)\ .$$

Damit ist aber

$$\begin{aligned} (*) &\leq \int_{X^{2m}} 2\mathrm{e}^{-m\epsilon^2/32} N(\epsilon/8, \mathcal{F}|\mathbf{z}, \hat{d}_{2m}) \\ &= 2\mathrm{e}^{-m\epsilon^2/32} E(N(\epsilon/8, \mathcal{F}|\mathbf{z}, \hat{d}_{2m}))\ . \end{aligned}$$

Daß die Abweichung der empirischen von der tatsächlichen Distanz gegen Null geht, wird also durch die Eigenschaft $\lg E(N(\epsilon/8, \mathcal{F}|\mathbf{z}, \hat{d}_{2m}))/m \to 0$ sichergestellt, wie man aus obiger Abschätzung nachrechnen kann. Es folgt dann aus der Eigenschaft $E(\lg N(\epsilon, \mathcal{F}|\mathbf{z}, \hat{d}_m))/m \to 0$ die UCEM Eigenschaft, dadann die interessierende Größe abgeschätzt werden kann gegen die Wahrscheinlichkeit für Pattern $\mathbf{x}$, wo der Logarithmus der empirischen Überdeckungszahl geteilt durch m größer als ein δ ist (wegen $E \lg N/m \to 0$ ist das klein) und den sich für die übrigen Muster ergebenden Term. Da hier die Überdeckung beschränkt ist, wird auch dieses klein. Es folgt die behauptete Aussage. □

Es stellt sich allerdings weiterhin die Frage, wie man die Überdeckungszahl einer Funktionenklasse abschätzen kann. Ein fundamentaler Begriff in der Lernbarkeitstheorie, der einerseits häufig zur konkreten Abschätzung der Überdeckungszahl und damit einhergehender konkreter Schranken, andererseits für die Charakterisierung verteilungsunabhängiger Lernbarkeit verwandt wird, ist die nach Vapnik und Chervonenkis benannte VC Dimension. Sie mißt in gewisser Weise die Kapazität einer Funktionenklasse, oder, anschaulicher, die maximale Anzahl an Punkten, auf denen man mithilfe der Funktionenklasse auswendig lernen kann.

Definition 4.8 *Sei $\mathcal{F}$ eine Klasse von Konzepten. Ein Menge $\{x_1,\ldots,x_m\}$ von Punkten in X wird durch $\mathcal{F}$* **geshattert**, *falls es für jede 0-1-wertige Abbildung d auf den Punkten eine Abbildung $f \in \mathcal{F}$ gibt mit $d = f|\{x_1,\ldots,x_m\}$. Die* **Vapnik-Chervonenkis Dimension** *(VC Dimension) von $\mathcal{F}$ ist die Kardinalität einer größten Punktmenge, die von $\mathcal{F}$ geshattert wird. Sie kann evtl. ∞ werden.*

Sei $\mathcal{F}$ eine Klasse von Funktionen mit Werten in $[0,1]$. Ein Menge $\{x_1,\ldots,x_m\}$ von Punkten in X wird durch $\mathcal{F}$ **ge-fat-shattert** *mit Parameter $\epsilon \geq 0$, falls Folgendes gilt: Es gibt reelle Werte $r_1,\ldots,r_m$ (nur von den Punkten, nicht von d abhängig) und für jede 0-1-wertige Abbildung d auf* *den Punkten eine Abbildung $f \in \mathcal{F}$ mit $|f(x_i) - r_i| \geq \epsilon$ für alle i (d.h. es wird ein Abstand ϵ von r_i gewahrt) und es gilt $d(X_i) = 1$ genau dann, wenn $f(x_i) - r_i \geq 0$ ist. Die ϵ-**fat shattering Dimension** von $\mathcal{F}$ ist die Kardinalität einer größten Punktmenge, die von $\mathcal{F}$ mit Parameter ϵ ge-fat-shattert wird. Im Fall $\epsilon = 0$ spricht man einfach von shattern und der **Pseudodimension**.*

Für Konzepte reduziert sich die Pseudodimesnion zur VC Dimension. Diese Größen sind einerseits (manchmal) handhabbar, andererseits erlauben sie, die verschiedenen Überdeckunszahlen abzuschätzen. Zunächst soll aber eine Beobachtung stehen, die als Sauers Lemma bekannt ist.

Satz 4.9 *Sei $\mathcal{F}$ eine Konzeptklasse mit der VC Dimension d und $\pi(\mathbf{x}, \mathcal{F})$ sei die Anzahl der verschiedenen Funktionen in $\mathcal{F}|\mathbf{x}$. Sei $|\mathbf{x}| = n \geq d$. Dann ist $\pi(\mathbf{x}, \mathcal{F}) \leq (\mathrm{e}n/d)^d$.*

Beweis: Wieder interessiert uns nur der für die Lernbarkeitstheorie typische Teil, daher werden nachzurechnende Abschätzungen weggelassen. Es wird $\pi(\mathbf{x}, \mathcal{F}) \leq \Phi(n, d) := \sum_{i=0}^{d} \binom{n}{i}$ gezeigt. Jenen letzten Ausdruck kann man (mit doppelter Induktion) gegen $(\mathrm{e}n/d)^d$ abschätzen. Er hat eine natürliche Interpretation als die Anzahl der Teilmengen von $\mathbf{x}$ mit maximal d Elementen.

Ist $d = 0$, dann ist die Funktionenklasse einelementig und also $\pi(\mathbf{x}, \mathcal{F}) = 1$. Ist $|\mathbf{x}| = 1$, $d \geq 0$, dann ist $\pi(\mathbf{x}, \mathcal{F}) \leq 2$. Für $|\mathbf{x}| \leq d$ ist offensichtlich nichts zu zeigen. Wir nehmen jetzt an, die Behauptung gelte für Punkte und Konzeptklassen, wo entweder weniger als n Punkte vorhanden sind oder die VC Dimension kleiner als d ist. Die Aussage wird für n Punkte $\mathbf{x}$ und VC Dimension d von $\mathcal{F}$ gezeigt. Es ist $\pi(\mathbf{x}, \mathcal{F}) = \pi(\mathbf{x}, \mathcal{F}|\mathbf{x})$. Betrachtet man einen Punkt aus $\mathbf{x}$, dann bestehen für jede Abbildung maximal zwei Möglichkeiten. Sei $\mathbf{x}' = (x_1, \ldots, x_{n-1})$. Dann ist also

$$\pi(\mathbf{x}, \mathcal{F}|\mathbf{x}) = \pi(\mathbf{x}', \mathcal{F}|\mathbf{x}') + \pi(\mathbf{x}', \mathcal{F}_x|\mathbf{x}')\,,$$

wobei $\mathcal{F}_x$ nur die Funktionen f darstellt, die ein Pendant g besitzen mit $f|\mathbf{x}' = g|\mathbf{x}'$, $f(x_n) \neq g(x_n)$. Im ersten Term werden also alle verschiedenen Funktionen auf $\mathbf{x}'$ aufgesammelt, der zweite fügt diejenigen hinzu, die sich nur auf x_n unterscheiden. Nach Voraussetzung ist $\pi(\mathbf{x}', \mathcal{F}|\mathbf{x}') \leq \sum_{i=0}^{d} \binom{n-1}{i}$. Die VC Dimension von $\mathcal{F}_x|\mathbf{x}'$ ist maximal $d-1$, denn für jede Abbildung in $\mathcal{F}_x|\mathbf{x}'$ gibt es in $\mathcal{F}_x$ sowohl die entsprechende Abbildung mit Wert 0 bzw. 1 auf x_n. Damit erhält man

$$\pi(\mathbf{x}, \mathcal{F}|\mathbf{x}) \leq \sum_{i=0}^{d} \binom{n-1}{i} + \sum_{i=0}^{d-1} \binom{n-1}{i} = \sum_{i=0}^{d} \binom{n}{i}\,.$$

□

Eine analoge Abschätzung ist für Funktionenklassen möglich. Man ersetzt die VC Dimension dann durch die Pseudodimension.

Satz 4.10 *Sei $\mathcal{F}$ eine Funktionenklasse mit der Pseudodimension d und $\pi_{\mathbf{c}}(\mathbf{x}, \mathcal{F})$ sei die Anzahl der verschiedenen Funktionen in $\{\mathrm{H} \circ (f - \mathbf{c}) : \mathbf{x} \to [0,1], x_i \mapsto \mathrm{H}(f(x_i) - c_i) \mid f \in \mathcal{F}\}$. Sei $|\mathbf{x}| = n \geq d$. Dann ist $\pi_{\mathbf{c}}(\mathbf{x}, \mathcal{F}) \leq (\mathrm{e}n/d)^d$.*

Beweis: Dieses folgt sofort, wenn man die durch die VC Dimension gegebene Schranke für die Überdeckungszahl obiger Konzeptklasse anwendet. □

Man kann also die Anzahl der Konzepte anhand der VC Dimension abshätzen. Bemerkenswert ist, daß diese a priori exponentiell wachsende Zahl nur noch polynomiell wächst, sobald die Anzahl der Punkte die VC Dimension überschritten hat und Auswendiglernen auf den Daten nicht mehr möglich ist. Dieses Resultat soll jetzt zur Abschätzung der Überdeckungszahl dienen.

Satz 4.11 *Es ist*

$$N(\epsilon, \mathcal{F}, d_P) \leq 2\left(\frac{2\,\mathrm{e}}{\epsilon} \ln \frac{2\,\mathrm{e}}{\epsilon}\right)^d\,,$$

wobei d die VC bzw. Pseudodimension von $\mathcal{F}$ darstellt.

Beweis: Wieder steht hier nur eine Idee, einige Rechnungen werden weggelassen. Sei $\mathcal{G}$ eine maximale ϵ-separierte Menge in $\mathcal{F}$. Für festes $\mathbf{x}$ und $\mathbf{c}$ sei genau wie eben $\pi_{\mathbf{c}}(\mathbf{x}, \mathcal{F}) = |\{\mathrm{H}(f(\mathbf{x}) - \mathbf{c}) \mid f \in \mathcal{F}\}|$. Es ist

$$\begin{array}{rl} & E_{\mathbf{x},\mathbf{c}}(\pi_{\mathbf{c}}(\mathbf{x}, \mathcal{F})) \\ \geq & E_{\mathbf{x},\mathbf{c}}(|\{f \in \mathcal{G}| \mid \mathrm{H}(f(\mathbf{x}) - \mathbf{c}) \neq \mathrm{H}(g(\mathbf{x}) - \mathbf{c})\ \forall g \in \mathcal{G}\backslash\{f\}\}|) \\ = & \sum_{f \in \mathcal{G}} P(\mathrm{H}(f(\mathbf{x}) - \mathbf{c}) \neq \mathrm{H}(g(\mathbf{x}) - \mathbf{c})\ \forall g \in \mathcal{G}\backslash\{f\}) \\ = & \sum_{f \in \mathcal{G}} 1 - P(\exists g \in \mathcal{G}\backslash\{f\}\ \mathrm{H}(f(\mathbf{x}) - \mathbf{c}) = \mathrm{H}(g(\mathbf{x}) - \mathbf{c})) \end{array}$$

Da sich alle Funktionen in $\mathcal{G}$ um ϵ unterscheiden, ist die Wahrscheinlichkeit, eine Komponente c_i zwischen dem Wert von f und g auf einem zufälligen x_i zu erwischen, mindestens ϵ. Bei zufälligem Ziehen von $\mathbf{c}$ und $\mathbf{x}$ und festem f ergibt sich also die Wahrscheinlichkeit $P(\exists g \in \mathcal{G}\backslash\{f\}\ \mathrm{H}(f(\mathbf{x}) - \mathbf{c}) = \mathrm{H}(g(\mathbf{x}) - \mathbf{c}))$ als maximal $|\mathcal{G}|(1-\epsilon)^n \leq |\mathcal{G}|\mathrm{e}^{-n\epsilon}$, da ja alle Komponenten von $\mathbf{c}$ nicht zwischen dem Wert von f und einem g liegen dürfen. Insgesamt kann man daher abschätzen

$$E_{\mathbf{x},\mathbf{c}}(\pi_{\mathbf{c}}(\mathbf{x}, \mathcal{F})) \leq |\mathcal{G}|(1 - |\mathcal{G}|\mathrm{e}^{-n\epsilon})\,.$$

In diese Ungleichung setzt man jetzt für die linke Seite die durch die VC bzw. Pseudodimension gegebene Abschätzung für die Funktion π ein. Eine längere Rechnung ergibt dann eine obere Schranke für $|\mathcal{G}|$, d.h. die Größe jeder ϵ-separierten Menge, und damit auch eine obere Schranke für die Überdeckungszahl. □

Da diese Abschätzung für jede Wahrscheinlichkeit P gilt, ist sie insbesondere für die empirische Erwartung $N(\epsilon, \mathcal{F}|\mathbf{x}, \hat{d}_m)$ für jedes $\mathbf{x}$ korrekt. Das heißt also, daß eine endliche VC bzw. Pseudodimension sowohl die UCED Eigenschaft als auch PAC Lernbarkeit unabhängig von der jeweiligen Verteilung garantiert. Aufgrund der fundamentalen Bedeutung dieses Ergebnis rechnen wir die sich aus obigen Sätzen ergebende Schranke explizit aus: Für eine Funktionenklasse mit Pseudodimension d bzw. eine Konzeptklasse mit VC Dimension d gilt mit der Wahrscheinlichkeit $1-\delta$ für beliebige Verteilungen P und Funktionen bzw. Konzepte f und beliebige Lernalgorithmen h

$$|d_P(f, h_m(\mathbf{x}, f)) - \hat{d}_m(f, h_m(\mathbf{x}, f), \mathbf{x})| \leq \sqrt{\frac{32}{m}\left(d \ln\left(\frac{32\mathrm{e}}{\epsilon} \ln \frac{32\mathrm{e}}{\epsilon}\right) + \ln \frac{4}{\delta}\right)}\,.$$

Folgender Satz zeigt, daß eine endliche VC Dimension sogar notwendig für verteilungsunabhängige PAC Lernbarkeit ist.

Satz 4.12 *Sei die VC Dimension der Konzeptklasse $\mathcal{F}$ gleich d. Dann gibt es eine Verteilung P, so daß für alle $\epsilon \leq 0.25$*

$$M(2\epsilon, \mathcal{F}, d_P) \geq \mathrm{e}^{2(0.5-2\epsilon)^2 d}$$

gilt.

Beweis: Auf einer Menge von Punkten $\mathbf{x}$, die geshattert werden, betrachtet man die Gleichverteilung P. Der Abstand zwischen zwei Funktionen ist dann genau durch $1/d$ multipliziert mit der Anzahl der Punkte aus $\mathbf{x}$, auf denen sich die Funktionen unterscheiden, gegeben. Die Funktionen, die sich weniger als 2ϵ unterscheiden, können durch die Zahl $\sum_{k \leq 2\epsilon d} \binom{d}{k}$ abgeschätzt werden. Die

Anzahl der übriggebliebenen Funktionen kann man mithilfe der sogenannten Chernoff-Okamoto Ungleichung aus der Statistik gegen die gewünschte Größe abschätzen. □

Damit erhält man die frappierende Konsequenz, daß die VC Dimesnion verteilungsunabhängige PAC Lernbarkeit äquivalent charakterisiert und man die verteilungsunabhängige UCED Eigenschaft frei Haus bekommt. Für Funktionenklassen ist die Situation aufgrund von Kodierungstricks etwas komplizierter. Auch hier kann es Funktionenklassen mit undendlicher fat shattering Dimension geben, die lernbar sind. Allerdings ist dieses bei verrauschten Daten unterbunden und die UCED Eigenschaft wird äquivalent durch endliche fat shattering Dimension charakterisiert. Genauer konnte in [Alon et.al.] eine Verallgemeinerung von Sauers Lemma für mehrwertige Funktionenklassen und die fat shattering Dimension erzielt werden. Die Ungleichung

$$\sup_{\mathbf{x}} N(\epsilon, \mathcal{F}|\mathbf{x}, \hat{d}_m) \leq 2\left(\frac{4m}{\epsilon^2}\right)^{d\,\ln(2em/(d\epsilon))}$$

mit d als der fat shattering Dimension zum Parameter $\epsilon/4$ erlaubt, die Distanz des empirischen und tatsächlichen Fehlers auch gegen die fat shattering Dimension abzuschätzen. In [Alon et.al.] wird zudem gezeigt, daß einerseits endliche fat shattering Dimesnion eine notwendige Bedingung für die verteilungsunabhängige UCED Eigenschaft ist, es aber andererseits Funktionenklassen mit unendlicher Pseudodimension, aber für jedes $\epsilon > 0$ endlicher fat shattering Dimension zum Parameter ϵ gibt.

Die hier hergeleiteten Begriffe sind ausreichend, die Situation, daß man eine feste Architektur trainiert, zu modellieren. Berücksichtigt man allerdings den Prozeß der Architekturauswahl mit, dann hat man es a priori mit einer Funktionenklasse mit unendlicher Kapazität zu tun: Man wählt ein Netz aus der Klasse aller Netze aus, die, wie wir gesehen haben, approximationsvollständig ist. Um diese allgemeinere Situation zu modellieren, kann man das sogenannte Luckiness Framework anwenden: Man bewertet je Trainingslauf a posteriori, wie gut denn das jeweilige Ergebnis ist, d.h. etwa wie groß die konkrete Netzarchitektur bzw. die Kapazität derselbigen geworden ist. Es wird also gemessen, wie glücklich der konkrete Tainingsverlauf gewesen ist. Dieses Maß darf dabei durchaus auch von den konkreten Trainingsdaten abhängen. Zusätzlich wählt man a priori Wahrscheinlichkeiten p_i, die sich zu 1 summieren und je angeben, wie sehr man mit diesem glücklichen Verlauf gerechnet hat. Schließlich erhält man für die Abweichung des empirischen vom tatsächlichen Fehler dieselben Schranken, die man auch bisher in einer (a priori) entsprechend glücklichen Situation erhalten hätte – einzige Änderung: Sie sind um einen der a priori Wahrscheinlichkeit entsprechenden Term ergänzt. Für eine exakte Ausführung sei auf [Shawe-Taylor et.al., Hammer] verwiesen.

4.2 Anwendung für feedforward Netze

Um die Resultate auf Neuronale Netze anwenden zu können, fehlen als erstes Abschätzungen für die VC bzw. Pseudodimension realistischer Netze. Sofern gute Schranken für die Pseudodimension existieren, können diese auch als obere Schranken für die fat shattering Dimension benutzt werden. Sobald diese aber etabliert sind, kann man den Generalisierungsfehler jeder trainierten Architektur anhand des Trainingsfehlers abschätzen. Insbesondere ist dann gewährleistet, daß überhaupt je der Testfehler, der ja üblicherweise zur Schätzung des Generalisierungsfehlers verwandt wird, gegen 0 geht. Von der Idee her sollen die Schranken sogar das Betrachten eines Testfehlers überflüssig machen, da ja der Generalisierungsfehler durch den empirischen Fehler und das durch die VC Schranken abschätzbare strukturelle Risiko beschränkt ist, formal:

$$E(f, h_m(\mathbf{x}, f)) \leq \hat{E}_m(f, h_m(\mathbf{x}, g), \mathbf{x}) + \epsilon$$

mit Wahrscheinlichlkeit δ und dem sich aus dem letzten Abschnitt ergebendem strukturellen Risiko ϵ. Allerdings sind die Schranken für ϵ im Allgemeinen schlechter, als es eine Abschätzung durch den Testfehler wäre, insbesondere, da hier ja ein worst case Szenario betrachtet wird. Daher ist durchaus ein Betrachten des Testfehlers angemessen.

Wie fangen, wie sollte es auch anders sein, mit einem einfachen Perzeptron an.

Satz 4.13 *Ein Perzeptron mit Eingabedimension n besitzt die VC Dimension $n+1$.*

Beweis: Die VC Dimension ist mindestens $n+1$, denn man kann jede $n+1$ Punkte im $\mathbb{R}^n$ in allgemeiner Lage shattern. Dabei bedeutet der Term ‚in allgemeiner Lage' genau, was man sich darunter vorstellt: Die Punkte liegen in typischer Situation im $\mathbb{R}^n$, d.h. keine n Punkte liegen schon auf einer $n-1$ dimensionalen Hyperebene. Algebraisch bedeutet das bei $n+1$ Punkten genau, daß für die Determinante der erweiterten Matrix

$$\det\begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{n+1} \\ 1 & \cdots & 1 \end{pmatrix} \neq 0$$

gilt (die Punkte sind affin unabhängig). Möchte man die Punkte jetzt beliebig nach $\{0,1\}$ abbilden, dann wählt man $y_i \in \{-1,1\}$ entsprechend der gewünschten Ausgabe und löst das Gleichungssystem

$$\mathbf{w}^t\begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{n+1} \\ 1 & \cdots & 1 \end{pmatrix} = \mathbf{y},$$

um die Gewichte $\mathbf{w}$ (Bias als On-Neuron) zu erhalten.

Seien umgekehrt m Punkte $\mathbf{x}_1, \ldots, \mathbf{x}_m$, die geshattert werden, gegeben. Dann gibt es 2^m Gewichte $\mathbf{w}_i$ (inklusive Bias), so daß der Vektor $(\mathbf{w}_i^t(\mathbf{x}_j, 1))_i$ für geeignetes $\mathbf{w}_i$ beliebige Vorzeichen hat. Die Werte für variierendes i und j seien in eine $2^m \times m$-Matrix B geschrieben, d.h. $B_{ij} = \mathbf{w}_i^t(\mathbf{x}_j, 1)$ oder

$$B = \begin{pmatrix} \mathbf{w}_1^t \\ \vdots \\ \mathbf{w}_{2^m}^t \end{pmatrix}\begin{pmatrix} \mathbf{x}_1 & \ldots & \mathbf{x}_m \\ 1 & \ldots & 1 \end{pmatrix}.$$

Falls diese Matrix keinen vollen Rang hätte, gäbe es einen Vektor $\mathbf{y} \neq \mathbf{0}$ in $\mathbb{R}^m$, so daß $B\mathbf{y} = 0$ gilt. Es gibt aber unter den Zeilen der Matriz B eine Zeile, deren Komponenten dasselbe Vorzeichen wie die Koeffizienten in $\mathbf{y}$ haben, d.h. multipliziert man diese Zeile mit $\mathbf{y}$, erhält man garantiert einen positiven Term. Widerspruch. Daher hat die Matrix also vollen Rang m, d.h. aber $\begin{pmatrix} \mathbf{x}_1 & \ldots & \mathbf{x}_m \\ 1 & \ldots & 1 \end{pmatrix}$ hat ebenfalls mindestens den Rang m. Aufgrund der Dimension folgt $m \leq n+1$. □

Als Konsequenz benötigt man für eine gute Generalisierungsleistung eines Perzeptrons eine Anzahl von Beispielen, die linear mit der Eingabedimension wächst. Die Generalisierungsleistung ist dann aber auch unabhängig von der konkreten Verteilung – es gibt natürlich spezielle Verteilungen, wo die Generalisierungsleistung besser ist, etwa wenn nur ein Teilraum des Eingaberaumes benötigt wird. Als unmittelbare Folgerung erhält man eine Schranke für die Pseudodimension eines sigmoiden Neurons:

Satz 4.14 *Die Pseudodimension eines einzelnen sigmoiden Neurons ist $n+1$, wenn n die Eingabedimension darstellt.*

Beweis: Offensichtlich ist die Pseudodimension wenigstens $n+1$, da man ja Perzeptronen approximieren kann. Um eine obere Schranke zu erhalten, beachte man, daß bei der Pseudodimension statt mit 0 mit Referenzwerten r verglichen wird. Es ist $\mathrm{sgd}(x) \geq r \iff x \geq \mathrm{sgd}^{-1}(r)$, daher kann man statt der Pseudodimension des sigmoiden Neurons auch die Pseudodimension des Raumes der affinen Abbildungen im $\mathbb{R}^n$ betrachten. Diese ist maximal gleich der Pseudodimension der linearen Abbildungen in $\mathbb{R}^{n+1}$. Für feste m Punkte im $\mathbb{R}^{n+1}$ hat $T := \{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m) \mid f$ ist linear$\}$ maximal die Dimension $n+1$, da der lineare Vektorraum der Funktionen auf $\mathbb{R}^{n+1}$ die Dimension $n+1$ hat. Falls die Punkte $\mathbf{x}_i$ geshattert werden, gibt es also einen Vektor $\mathbf{r}$, so daß der um diesen Vektor verschobene Raum T alle Orthanten des $\mathbb{R}^m$ trifft. Es gibt aber keinen weniger als m-dimensionalen Raum, der das tut. Für den Nullvektor $\mathbf{r}$ wurde das oben schon gezeigt. Ist $\mathbf{r}$ nicht $\mathbf{0}$, dann bezeichne $\mathbf{y}$ einen zum Raum $\mathbf{r}+T$ orthogonalen Vektor ungleich $\mathbf{0}$, der auf einer zu T senkrechten Gerade durch den Ursprung mit Richtung zum Ursprung liegt. $\mathbf{r}'$ sei der Vektor auf dieser Gerade durch den Ursprung, der die Spitze in $\mathbf{r}+T$ hat. Es gibt keinen Vektor in $\mathbf{r}+T$, der dieselben Vorzeichen wie $\mathbf{y}$ hat. Denn für so einen Vektor würde das Skalarprodukt mit $\mathbf{y}$ einen Wert ≥ 0 liefern, allerdings kann man so einen Vektor als $\mathbf{r}' + \mathbf{x}$ mit einem zu $\mathbf{y}$ orthogonalen Vektor $\mathbf{x}$ schreiben mit $\mathbf{y}^t(\mathbf{r}' + \mathbf{x}) = \mathbf{y}^t\mathbf{r}' < 0$. □

Allerdings sind nicht alle Aktivierungsfunktionen so zahm, wie es sich die Perzeptronaktivierung oder die sigmoide Funktion und damit auch der hyperbolische Tangens oder andere im Wesentlichen äquivalente Aktivierungen erwiesen haben. Wie wir schon gesehen haben, erlaubt die Funktion $\cos$, beliebige rational unabhängige Punkte auf der Zahlengerade auf ein gewünschtes Vorzeichen abzubilden. Man erhält

Satz 4.15 *Die Pseudodimension eines Neurons mit der Aktivierungsfunktion* $\cos$ *ist unendlich. Dasselbe gilt für die fat shattering Dimension zu Parametern kleiner als* 0.5.

Solche Netze sind also unter realistischen Bedingungen nicht zum Lernen geeignet, da zumindest bei einigen Pattern und unbeschränkten Gewichten Auswendiglernen möglich ist.

Wir wenden uns jetzt größeren Architekturen zu. Für feedforward Netze mit der Perzeptronaktivierung kann man die VC Dimension ebenfalls abschätzen.

Satz 4.16 *Die VC Dimension eines feedforward Netzes mit der Perzeptron Aktivierungsfunktion und* W *Gewichten ist von der Ordnung* $W \log W$.

Beweis: [Maass, Sakurai] haben Netze mit W Gewichten und einer VC Dimension der angegebenen Ordnung konstruiert. Maass betrachtet dabei Netze beliebiger Tiefe mit mindestens zwei verborgenen Schichten, Sakurai betrachtet Netze mit einer verborgenen Schicht.

Eine obere Schranke kann mit den hier schon bewiesenen Mitteln hergeleitet werden: Jedes einzelne Neuron des Netzes berechnet eine Funktion f_i auf einem Raum der Dimension n_i, die durch die Anzahl der Vorgängerneuronen gegeben ist. Die Anzahl der verschiedenen Abbildungen, die so ein einzelnes Neuron auf m verschiedenen Eingaben berechnen kann, ist nach Sauers Lemma durch $(\mathrm{e}m/(n_i+1))^{n_i+1}$ gegeben. Ordnet man die Neuronen so an, daß f_i nur Vorgänger unter den Neuronen $1, \ldots, i-1$ hat, dann findet man bei m Eingaben also bzgl. dem ersten Neuron $(\mathrm{e}m/(n_1+1))^{n_1+1}$ verschiedene Abbildungen, für jede Menge der sich ergebenden m Werte für das Neuron 2 wieder $(\mathrm{e}m/(n_2+1))^{n_2+1}$ verschiedene Abbildungen, ... Das gesamte Netz berechnet also auf m verschiedenen Eingaben maximal

$$\prod_i \left(\frac{\mathrm{e}m}{n_i+1}\right)^{n_i+1}$$

verschiedene Funktionen. Das kann durch $(N\mathrm{e}m/W)^W$ nach oben abgeschätzt werden. Es kann maximal eine Menge der Kardinalität m mit $2^m \leq (N\mathrm{e}m/W)^W$ geshattert werden. Dieses ergibt

die obere Schranke $m \leq 2W \log(\mathrm{e}N)$. □

[Bartlett et.al.] haben gezeigt, daß sich die Ordnung auf die Ordnung W verbessern läßt, falls man die maximale Tiefe der Netze beschränkt. Bei realistischerweise maximal zwei bis drei hidden layern gilt also auch die Daumenregel, daß die Anzahl der Beispiele linear mit der Anzahl der Gewichte wachsen sollte. Verbesserungen sind möglich, falls die Gewichte sehr klein gehalten werden oder die Eingaben nur aus einem endlichen Alphabet stammen. Auch hier gibt es in Spezialfällen mitunter bessere Schranken [Bartlett et.al.].

In der Praxis verwendet man differenzierbare Aktivierungsfunktionen wie die sigmoide Aktivierung oder auch, der Effizienz wegen, stückweise lineare Näherungen derselben. Für feeforward Netze mit stückweise polynomieller Aktivierung erhält man eine Schranke, indem man die Netzfunktion als Boolesche Formel in Ausdrücken der Form $p(x) > 0$ für Polynome p schreibt, die unter anderem die Gewichtsvektoren als Parameter besitzen, und die Anzahl der verschiedenen möglichen Wahrheitswertbelegungen abschätzt, wenn man die Gewichte variiert. Dieses ergibt eine Schranke für die Pseudodimension.

Satz 4.17 *Sei ein feedforward Netz mit* N *Neuronen,* $W \geq N$ *einzustellenden Gewichten (die evtl. a priori festen Gewichte werden nicht gezählt), Tiefe* t*, stückweise polynomiellen Aktivierungsfunktionen mit je maximal* q *Stücken und maximalem Grad* d *gegeben. Dann ist die Pseudodimension maximal* $2W \log_2(8\mathrm{e}(W+1)q^W(d^t + (t-1))) =$

$$2W(\log_2(8\mathrm{e}) + \log_2(d^t + (t-1)) + \log_2(W+1) + W \log_2 q)\,.$$

Beweis: Für jede konkrete Eingabe $\mathbf{x}$ des Netzes berechnet sich die Ausgabe als Polynom vom Grad maximal $d^t + (t-1)$ in den Gewichten(!), denn in jeder Schicht wird der bisherige Ausdruck mit w multipliziert und von diesem Wert die Aktivierungsfunktion mit maximalem Grad d berechnet. Betrachtet man die verschiedenen Bereiche der Aktivierungsfunktionen, dann erhält man bei variierenden Eingaben und Gewichten maximal q^N verschiedene Polynome. Für eine konkrete Eingabe $\mathbf{x}$ und Referenzvektor r kann man das Vorzeichen der Netzausgabe verglichen mit r als Boolesche Formel in $2(N+1)q^N$ verschiedenen Ausdrücken der Form $p(\mathbf{x}, r, \mathbf{w}) > 0$ formulieren, wobei p ein Polynom vom Grad $\leq d^t + (t-1)$ in $\mathbf{w}$ ist: Die Ausgabe berechnet sich, wie oben gesagt, als eines von q^N möglichen Polynomen (r subtrahiert); welches dieser Polynome auf die Eingaben und Gewichte zutrifft, testet man, indem man den Bereich jeder einzelnen Aktivierung jedes Neurons testet. Dessen Aktivierung berechnet sich ebenfalls je nach Vorgängeraktivierungen als eines von maximal q^{N-1} möglichen Polynomen. Man führt also zur Bestimmung der Ausgabe maximal $2q \cdot q^{N-1} \cdot N$ Vergleiche durch. Insgesamt treten also maximal $s := 2(N+1)q^N$ verschiedene Ausdrücke in der Booleschen Formel auf. Wie nummerieren die Booleschen Elementarausdrücke mit φ_i, $i = 1, \ldots, s$, das in der Formel φ_i vertretene Polynom sei $f_i(\mathbf{x}, r, \mathbf{w})$.

Man nimmt jetzt an, die Punkte $\mathbf{x}_1, \ldots, \mathbf{x}_m$ seien mit Referenzwerten r_i geshattert. Dann findet man mindestens 2^m Gewichte $\mathbf{w}$, so daß die sich ergebenden Vektoren in $\{0,1\}^{sm}$ mit den Wahrheitswertbelegungen $(\varphi_i(\mathbf{x}_j))_{ij}$, die das Vorzeichen der Netzausgabe auf $\mathbf{x}_i$ eindeutig charakterisieren, für unterschiedliche $\mathbf{w}$ verschieden sind. Die Anzahl der möglichen Wahrheitswertbelegungen läßt sich abschätzen gegen die Anzahl der möglichen Vorzeichenwechsel der beteiligten Polynome: Eine Wahrheitswertbelegung ist höchstens dann verschieden, wenn wenigstens eines der auftretenden Polynome für verschiedene $\mathbf{w}$ unterschiedliches Vorzeichen besitzt. Es muß also die Anzahl der Zusammenhangskomponenten von $\mathbb{R}^W \backslash \{\mathbf{w} \mid \exists \mathbf{x}_j \exists f_k\, f_k(\mathbf{x}, r, \mathbf{w}) = 0\}$ beschränkt werden, denn innerhalb dieser Komponenten ist der die Wahrheitswertbelegungen bestimmende Vektor konstant. Man braucht hier eigentlich noch ein Argument dafür, daß man im Fall eines Polynoms exakt gleich 0 auch analoge Situationen mit echt positivem bzw. negativem Vorzeichen findet. Dieses folgt aus formalen Gründen. Etwa in [Warren] ist die Zahl der Zusammenhangskomponenten durch die Größe

$$\left(\frac{4\mathrm{e}msd'}{W}\right)^W$$

mit dem Grad der Polynome $d' = d^t + (t-1)$ abgeschätzt. Der Ansatz $(4\mathrm{e}msd'/W)^W \geq 2^m$ führt zur Schranke $m < 2W\log_2(4\mathrm{e}sd')$ oder

$$m < 2W\log_2(8\mathrm{e}(N+1)q^N(d^t+(t-1)))\,.$$

□

Betrachtet man allerdings die sigmoide Aktivierungsfunktion, dann steht man vor zwei Problemen: Eine Komposition der Netzfunktionen führt zu einem nicht mehr handhabbarem Ausdruck, und man benötigt Schranken für Nullstellenmengen von Funktionen, die auch die Exponentialfunktion beinhalten. Dem ersten Problem kann man durch Einführen neuer Variablen, die die Aktivierung der Neuronen repräsentieren, begegnen. Mit einem etwas komplizierterem differentialgeometrischen Ansatz kann man dann auch hier die Anzahl der Zusammenhangskomponenten bestimmen. Ein Beweis für den sigmoiden Fall wurde von [Karpinski, Macintyre] gefunden.

Satz 4.18 *Sei ein feedforward Netz mit N Neuronen, W einstellbaren Gewichten und der standard sigmoiden Aktivierungsfunktion gegeben. Dann ist die Pseudodimension maximal von der Ordnung*

$$O(W^2N^2)\,.$$

Interessant ist dabei, daß durchaus nicht alle Funktionen, die der Sigmoide ähnlich sehen, sich auch so schön verhalten. Der wichtige Punkt ist, daß sich die Nullstellenmengen der Funktionen schön verhalten, d.h. die Anzahl der Zusammenhangskomponenten des Komplements endlich ist. Insbesondere ist genau dieses für den Cosinus nicht der Fall. Es gibt Funktionen, wo dieses Verhalten nicht so offensichtlich ist, wie beim Cosinus. Etwa die Funktion

$$\theta(x) = \arctan(x)/\pi + \cos(x)/(10(1+x^2)) + 1/2$$

sieht der sigmoiden Funktion sehr ähnlich und hat auch schöne Eigenschaften, sie ist etwa analytisch und unendlich häufig differenzierbar, sie ist eine squashing Funktion, d.h. monoton mit Limites 0 bzw. 1. Aber es gilt:

Satz 4.19 $(1,2,1)$ *feedforward Netze mit der Aktivierungsfunktion θ haben eine unendliche Pseudodimension.*

Beweis: Die Linearkombination $\theta(tx)+\theta(-tx)-1$ ergibt $\cos(tx)/(5(1+t^2x^2))$. Der Nenner dieses Ausdrucks ist positiv, daher kann man für beliebige rational unabhängige Eingaben x Werte t finden, die diese Werte auf Zahlen mit beliebig wählbarem Vorzeichen abbilden. Wäre also in der Ausgabe die Identität, dann könnte man dieses Eingaben shattern. Die Identität kann aber durch den Differenzenquotienten $x \approx (\theta(\epsilon x)-\theta(0))/(\epsilon\theta'(0))$ für kleines ϵ beliebig gut angenähert werden. D.h. Skalierung der Ausgabegewichte und Vergleich mit der Referenz $-\theta(0)/(\epsilon\theta'(0))$ führt für kleines ϵ zum selben Ergebnis. □

Das heißt: Eine auch durchaus auf den ersten Blick nicht sichtbare Oszillation in der Aktivierungsfunktion kann Lernen verhindern.

Interessant sind natürlich auch hier untere Schranken für die Kapazität, da man zumindest unter realistischen Bedingungen in einigen Situationen eine durch diese Kapazität nach unten beschränkte Anzahl an Beispielen für die Generalisierungsfähigkeit benötigt. Etwa für die sigmoide

Aktivierung kann man dieselben unteren Schranken wie für die Perzeptronaktivierung erhalten, da ja die Perzeptronaktivierung durch die Sigmoide approximiert werden kann. Eine bessere, aber von der oberen Schranke noch weit entfernte Schranke, findet sich in [Koiran, Sontag]:

Satz 4.20 *Die fat shattering Dimension von sigmoiden Netzen mit W veränderbaren Gewichten ist mindestens von der Ordnung W^2.*

Beweis: Es wird ein Netz mit zwei Eingabeneuronen betrachtet. Die Punkte, die geshattert werden, sind die Punkte $\{1,\ldots,n\}^2$. Für eine Abbildung f der Punkte nach $\{0,1\}$ seien Gewichte $\mathbf{w}^1$, …, $\mathbf{w}^n$ mit $\mathbf{w}^i = 0.w_1^i\ldots w_n^i$, $w_j^i = f(i,j)$ gewählt. W ist der Gesamtvektor der Gewichte. Es geht jetzt also darum, mit einem Netz bei Eingabe von i und j die entsprechende Stelle im i ten Gewicht zu berechnen.

Sei

$$f_W^1(y) = \mathbf{w}^1 + \sum_{i=2}^n(\mathbf{w}^i - \mathbf{w}^{i-1})\mathrm{H}(y-i+0.5).$$

Offensichtlich gilt $f^1(j) = \mathbf{w}^j$ für $1 \leq j \leq n$. f^1 kann mit einem Netz mit $n-1$ Perzeptronneuronen, $3(n-1)+1$ Gewichten und einem linearen Neuron berechnet werden.

$f^2(\mathbf{w})$ berechne die Ziffern der Koeffizienten in $\mathbf{w}$, d.h. $f^2(0.w_1\ldots w_n) = (w_1,\ldots,w_n)$. Eine Netzkonstruktion für f^2 ist induktiv möglich: Aus $(w_1,\ldots,w_i,0.w_{i+1}\ldots w_n)$ erhält man

$$w_{i+1} = H(0.w_{i+1}\ldots w_n - 0.1),\qquad 0.w_{i+2}\ldots w_n = 10\cdot 0.w_{i+1}\ldots w_n - w_{i+1}.$$

Insgesamt ergibt dieses ein Netz mit n Perzeptronneuronen, n linearen Neuronen und $4n$ Gewichten.

$f^3(x,\mathbf{w})$ berechnet die x te Komponente von $\mathbf{w}$. Als Netz ist das etwa

$$f^3(x,\mathbf{w}) = w_1 + \sum_{i=2}^n(w_i\mathrm{H}(x-i-0.5) - w_{i-1}\mathrm{H}(x-i-0.5))\,,$$

wobei man jedes Produkt ab für $a,b\in\{0,1\}$ durch $\mathrm{H}(a+b-1.5)$ ersetzen kann. f^3 besitzt ein lineares Neuron, $4(n-1)$ Perzeptronneuronen und $12(n-1)+n$ Gewichte.

Das Netz

$$(x,y)\mapsto f^3(x,f^2(f_W^1(y)))$$

shattert also die angegebenen Zahlen bei geeigneter Wahl von W und besitzt $n+2$ lineare Neuronen, $6n-5$ Perzeptronneuronen und $19n-13$ Gewichte. Man kann die die Perzeptronaktivierung beliebig gut mit der sigmoiden Aktivierung annähern, die lineare Aktivierung beliebig gut mit dem Differenzenquotienten (der zusätzliche Term kann jeweils zum Bias folgender Neuronen gezählt werden), und erhält so ein sigmoides Netz, das dieselben Eingaben mit jeder Genauigkeit < 0.5 shattert. □

Diese Ergebnisse etablieren zusammengenommen die prinzipielle Lernbarkeit einer festen neuronalen feedforward Architektur, wie sie in der Praxis vorkommt. Allerdings sind die konkreten Schranken in realistischen Fällen häufig sehr konservativ, da ja der – in der Regel nicht vorkomende – schlechteste Fall mit abgeschätzt wird. Nichtsdestotrotz sind die nachgewiesenen Ergebnisse prinzipiell beruhigend, sagen sie doch, daß im Gegensatz etwa zum $\cos$ realistische Netze sich prinzipiell gut verhalten.

Es soll hier noch eine Abschätzung folgen, die zu einem konkreten alternativen Lernverfahren, der Support Vektor Maschine geführt hat. Die VC Dimension eines Perzeptrons wächst bei wachsender Eingabedimension. Das ist allerdings nicht der Fall, wenn man sich auf bestimmte Perzeptronen beschränkt, die die Daten nicht irgendwie, sondern mit einem gewissen Mindestabstand zur Trenngerade trennen.

Satz 4.21 *Sei $\mathcal{F}$ die Menge der linearen Funktionen von $\{\mathbf{x} \in \mathbb{R}^n \mid |\mathbf{x}| \leq R\}$ nach $\mathbb{R}$, so daß der Gewichtsvektor maximal die Länge B hat. Dann ist die fat shattering Dimension zum Parameter ϵ maximal*

$$\frac{R^2 B^2}{\epsilon^2}\,.$$

Beweis: Sei die endliche Menge S geshattert mit Parameter ϵ. Dann gilt für jede Teilmenge S_0 in S:

$$|\sum_{x\in S_0} x - \sum_{x \in S\setminus S_0} x| \geq |S|\epsilon/B\,.$$

Um dieses zu zeigen, sei $S = \{x_1, \ldots, x_m\}$ durch die Parameter $\mathbf{w}_i$ geshattert, i je nach gewünschter Abbildung in $\{0,1\}^m$, Referenzvektoren seien r_i. Falls $\sum_{x_i \in S_0} r_i \geq \sum_{x_i \in S\setminus S_0} r_i$ sei $b_i = 1$, falls $x_i \in S_0$, und $b_i = 0$, falls $x_i \notin S_0$. Es gelten dann für dieses $\mathbf{w}_b$ die Ungleichungen: $\mathbf{w}_b^t \mathbf{x}_i \geq r_i + \epsilon$ falls $\mathbf{x}_i \in S_0$ und $\mathbf{w}_b^t \mathbf{x}_i \leq r_i - \epsilon$ falls $\mathbf{x}_i \notin S_0$, also

$$\mathbf{w}_b^t \sum_{x \in S_0} x \geq \sum_{i\; x_i \in S_0} r_i + |S_0|\epsilon\,.$$

Analog folgt

$$\mathbf{w}_b^t \sum_{x \in S\setminus S_0} x \leq \sum_{i\; x_i \in S\setminus S_0} r_i - |S\setminus S_0|\epsilon\,.$$

Also gilt

$$\mathbf{w}_b^t \left(\sum_{x\in S_0} x - \sum_{x\in S\setminus S_0} x \right) \geq |S|\epsilon\,.$$

Mit $|w| \leq B$ und der Cauchy-Schwartzschen Ungleichung erhält man

$$|\sum_{x\in S_0} x - \sum_{x \in S\setminus S_0} x| \geq |S|\epsilon/B.$$

Falls $\sum_{x_i \in S_0} r_i \leq \sum_{x_i \in S\setminus S_0} r_i$ wählt man $b_i = 1$, falls $x_i \notin S_0$, und $b_i = 0$, falls $x_i \in S_0$, und erhält mit einem analogen Argument dieselbe Abschätzung.

Ferner gibt es für alle Menge $S = \{x_1, \ldots, x_d\}$ von Punkten maximal der Länge R ein $S_0 \subset S$ mit

$$|\sum_{x\in S_0} x - \sum_{x \in S\setminus S_0} x| \leq \sqrt{|S|}R\,.$$

Um dieses zu zeigen, sei $b \in \{-1,1\}^d$ zufällig gezogen. Dann gilt für die ebenfalls zufällige Menge $S_0 = \{x_i \in S \mid b_i = 1\}$ die Rechnung

$$\begin{aligned}
E\left(|\sum_{x\in S_0} x - \sum_{x\in S\setminus S_0} x|^2 \right) &= E\left(|\sum_{i=1}^d b_i x_i|^2 \right) \\
&= \sum_{i=1}^d \left(\sum_{j\neq i} E(b_i b_j x_i^t x_j) + E\left(|b_i x_i|^2\right) \right) \\
&= \sum_{i=1}^d E\left(|b_i x_i|^2\right) \\
&\leq |S|R^2\,.
\end{aligned}$$

Diese vorletzte Gleichung folgt, da die b_i unabhängig mit Erwartungswert 0 gezogen werden. Daher ist $E(b_i b_j x_i^t x_j) = E(b_i)E(b_j)x_i^t x_j = 0$. Mit dem Erwartungswert muß auch mindestens eine Ausprägung kleiner als die angegebene Größe sein. Dieses liefert das gesuchte S_0.

Kombiniert man die beiden Ungleichungen, dann erhält man

$$\frac{|S|\epsilon}{B} \leq \sqrt{|S|R^2} \Rightarrow |S| \leq \frac{R^2 B^2}{\epsilon^2}\,.$$

□

Beschränkt man sich bei gegebenen Daten also auf die Perzeptronen, die Gewichtsbeschränkung B und Minimalgüte ϵ haben, dann ist die VC-Dimension nicht gleich der Eingabedimension, sondern durch obigen Term gegeben, der bei hochdimensionalen Daten mitunter wesentlich kleiner sein kann. Statt der Gewichtsbeschränkung und der Mindestgüte kann man auch einen Abstand der Trennhyperebenen von den Daten von mindestens $\gamma = B/\epsilon$ verlangen. Der Abstand eines Punktes $\mathbf{x}$ von der durch $(\mathbf{w}, \theta)$ gegebenen Hyperebene berechnet sich nämlich wie folgt: Der Punkt auf der Hyperebene, der auf dem Schnittpunkt mit der durch $\mathbf{x}$ laufenden Normalen liegt, ist

$$\mathbf{x} + \frac{\theta - \mathbf{w}^t\mathbf{x}}{\mathbf{w}^t\mathbf{w}}\mathbf{w}\,.$$

Der Abstand von $\mathbf{x}$ zur Geraden berechnet sich also als

$$|\frac{\theta - \mathbf{w}^t\mathbf{x}}{\mathbf{w}^t\mathbf{w}}\mathbf{w}| = \frac{\epsilon}{|\mathbf{w}|} \geq \frac{\epsilon}{B}\,.$$

Fazit: Trennt eine Hyperebene die Daten mit einem gewissen Mindestabstand, dann ist die Generalisierung umso besser, je größer dieser Abstand ist. Die Generalisierung skaliert dann nicht mit der Eingabedimension.

Dieses ist allerdings mathematisch nicht korrekt, da man ja seine betrachtete Funktionenklasse wählen muß, bevor man Daten sieht. Der Abstand wird allerdings im Hinblick auf die Daten gemessen und festgesetzt. Formal muß man zu einer exakten Begründung der verbesserten Lernbarkeit bei großem Abstand zu den Daten das schon erwähnte Luckiness-Framework heranziehen. Dieses erlaubt in diesem Fall, a posteriori den Ausgang des Lernens auch in Bezug auf die konkreten Daten zu beurteilen und die dann resultierenden Schranken (ergänzt um a priori Wahrscheinlichkeiten) anzuwenden. In obigem Fall führt das zu um konstante Faktoren erweiterte Schranken für die Lernbarkeit, die statt der Eingabedimension den Abstand zur Geraden und den Betrag der Eingaben verwendet.

4.3 Support Vektor Maschine

Die Tatsache, daß die VC Dimension eines Perzeptrons mit Abstand zu den Daten wesentlich besser und insbesondere unabhängig von der Eingabedimension sein kann, hat Vapnik zu einer zu feedforward Netzen alternativen Lernmethodik inspiriert: Der **Support Vektor Maschine** (SVM). Wir betrachten hier die SVM lediglich als Klassifikator, d.h. sie soll Funktionen $f : \mathbb{R}^m \to \{0,1\}$ lernen.

Die **lineare SVM** ist im Wesentlichen nur ein einfaches Perzeptron, d.h. berechnet eine Funktion

$$f : \mathbb{R}^m \to \{0,1\}\,,\ \mathbf{x} \mapsto \mathrm{H}(\mathbf{w}^t\mathbf{x} - \theta)\,.$$

Wir nehmen an, daß die Daten immer eine durch R beschränkte Länge haben, d.h. $|\mathbf{x}| \leq R$. Wir nehmen hier der Einfachheit halber an, der Bias sei 0, d.h. $\theta = 0$. Eine analoge Herleitung mit Bias (oder vermöge On-Neuronen) ist möglich. Das Ergebnis mit explizitem Bias ist im Allgemeinen verschieden zum Ergebnis mit On-Neuronen, da sich der Abstand zur Gerade verschieden berechnet. Der Knackpunkt ist jetzt, daß zu gegebenen Daten nicht irgendeine, sondern eine bezüglich der Generalisierungsfähigkeit optimale Trennhyperebene gesucht wird, die Hyperebene mit maximalem Abstand zu den Daten. Gegeben eine Trainingsmenge $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, m\}$ ist also ein Gewichtsvektor gesucht mit

$$\mathbf{w}^t\mathbf{x}_i \geq 0 \iff y_i = 1\,.$$

Man kann annehmen, daß kein Vektor exakt auf der Trennhyperebenen liegt und also durch Skalieren des Vektors erreichen, daß der Ausdruck $\mathbf{w}^t\mathbf{x}_i$ betragsmäßig immer ≥ 1 ist. Wählt man $\mathbf{y}_i$ in $\{-1, 1\}$ statt in $\{0, 1\}$, dann schreibt sich dieses als

$$y_i\mathbf{w}^t\mathbf{x}_i - 1 \geq 0\,.$$

Gleichzeitig soll der Abstand der Gerade zu den Daten minimiert werden. Der Abstand eines Punktes $\mathbf{x}$ zur Geraden mit Gleichung $\mathbf{w}^t\mathbf{x} = 0$ läßt sich als $|t/|\mathbf{w}||$ mit $\mathbf{w}^t(\mathbf{x}+t\mathbf{w}) = 0$ berechnen, d.h. der Abstand ist $|\mathbf{w}^t\mathbf{x}|/|\mathbf{w}|$. Der minimale Abstand zur Trennhyperebenen liegt für Punkte vor, für die in obiger Ungleichung die exakte Gleichheit gilt. Man maximiert also den minimalen Abstand der Punkte zur Trenngeraden, wenn der Ausdruck

$$\frac{1}{|w|}$$

maximiert wird. Man möchte $|w|^2/2$ unter der Bedingung $y_i\mathbf{w}^t\mathbf{x}_i - 1 \geq 0$ für alle i minimieren. Das bedeutet,

$$\min_{\mathbf{w}} \max_{\alpha_i \geq 0} \left(\frac{1}{2}|\mathbf{w}|^2 - \sum \alpha_i (y_i\mathbf{w}^t\mathbf{x}_i - 1) \right)$$

für neue Variablen α_i zu bestimmen, denn

$$\max_{\alpha_i \geq 0} \left(\frac{1}{2}|\mathbf{w}|^2 - \sum \alpha_i (y_i\mathbf{w}^t\mathbf{x}_i - 1) \right) = \begin{cases} \infty & \text{falls } y_i\mathbf{w}^t\mathbf{x}_i - 1 < 0 \\ |\mathbf{w}|^2/2 & \text{sonst.} \end{cases}$$

Diese Aufgabe kann man weiter vereinfachen, dazu brauchen wir aber eine Anleihe an die Optimierung.

Eine Funktion $f : \mathbb{R}^m \to \mathbb{R}$ heißt **konvex**, falls für alle $\mathbf{x}$, $\mathbf{x}'$ die Strecke durch die beiden Funktionswerte oberhalb des Graphen verläuft, d.h. $f(t\mathbf{x}+(1-t)\mathbf{x}') \leq tf(\mathbf{x})+(1-t)f(\mathbf{x}')$ für $0 < t < 1$. Stellt man um und läßt t gegen Null gehen, erhält man die Ungleichung $\nabla f(\mathbf{x}')^t(\mathbf{x}-\mathbf{x}') \leq f(\mathbf{x}) - f(\mathbf{x}')$. Die sogenannten **Kuhn-Tucker-Bedingungen** sagen, daß ein globales Minimum einer konvexen Funktion f mit Nebenbedingungen $g_i(\mathbf{x}) \geq 0$, $-g_i$ konvex, genau dann in $\mathbf{x}'$ vorliegt, wenn $g_i(\mathbf{x}') \geq 0$ für alle i gilt und es $\alpha_i \geq 0$ gibt mit $\alpha_i(\mathbf{x}') = 0$ für alle i, $\nabla f(\mathbf{x}') - \sum \alpha_i \nabla g_i(\mathbf{x}') = 0$.

['$\Leftarrow$' Es ist aufgrund der Konvexität

$$f(\mathbf{x}) - \sum \alpha \nabla g_i(\mathbf{x}) \geq f(\mathbf{x}') + \nabla f(\mathbf{x}')^t(\mathbf{x}-\mathbf{x}') + \sum \alpha(\mathbf{x}') - \sum \alpha \nabla g_i(\mathbf{x}')^t(\mathbf{x}-\mathbf{x}')$$

Die letzten beiden Terme fallen aufgrund der Bedingungen weg, d.h.

$$f(\mathbf{x}) \geq f(\mathbf{x}') + \sum \alpha(\mathbf{x}).$$

Also ist $\mathbf{x}'$ global optimal für alle Werte mit $g_i(\mathbf{x}) \geq 0$.

'$\Rightarrow$' Für ein Optimum $\mathbf{x}'$ gilt entweder $g_i(\mathbf{x}') = 0$ oder $g_i(\mathbf{x}') < 0$. Der Gradient von f ist in Bezug auf alle Richtungen 0, die kein g_i betreffen oder nur solche, wo $g_i(\mathbf{x}') < 0$ gilt. Betrachtet man die Gleichungen $g_i(\mathbf{x}') = 0$, so besitzt der Gradient von f einen positiven Anteil in Richtung des Gradienten von g_i, da eine weitere Minimierung von f nur unter Verletzung von $g_i(\mathbf{x}) \geq 0$ in Richtung g_i möglich wäre. D.h. man findet für $g_i(\mathbf{x}') < 0$ den Wert $\alpha = 0$ und für $g_(\mathbf{x}') = 0$ einen Wert $\alpha > 0$ mit $\nabla f(\mathbf{x}') - \sum \alpha \nabla g_i(\mathbf{x}') = 0$.]

Man berechnet also

$$\nabla \frac{1}{2}|\mathbf{w}|^2 - \sum \alpha \nabla (y_i\mathbf{w}^t\mathbf{x}_i - 1) = 0\,,$$

d.h. $\mathbf{w} = \sum \alpha \mathbf{x}_i$. Einsetzen in obige zu optimierende Funktion liefert

$$\sum \alpha_i - \frac{1}{2} \sum \alpha_i\alpha_j y_i y_j \mathbf{x}_i^t\mathbf{x}_j\,.$$

Dieses ist unter der Nebenbedingung $\alpha_i \geq 0$ zu maximieren. Gibt es überhaupt erfüllende Belegungen, kann man $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$ setzen, gibt es keine Lösung, explodieren die α_i gegen ∞. Die lineare SVM besteht daher lediglich aus etwa einem Gradientenauftstiegsverfahren der Funktion $\sum \alpha_i - \frac{1}{2} \sum \alpha_i\alpha_j y_i y_j \mathbf{x}_i^t\mathbf{x}_j$ unter der Bedingung $\alpha_i \geq 0$. Da es sich um eine zu optimierende quadratische Funktion handelt, sind in der Regel hierfür optimierte Verfahren, etwa ein konjugierte Gradientenverfahren, vorzuziehen. Der Klassifikator selber ergibt sich anschließend durch $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$, sobald adäquate α_i gefunden wurden. Diejenigen $\mathbf{x}_i$, für die $\alpha_i \neq 0$ ist, nennen sich auch **Support Vektoren**. Es sind Punkte mit geringstem Abstand zur Trennhyperebenen. Sie legen durch die Bedingung, daß die Aktivierung hier 1 bzw. -1 sein soll, den Lösungsvektor $\mathbf{w}$ eindeutig fest. Im Prinzip bestimmen die Punkte mit minimalem Abstand zur Trennhyperebenen das Ergebnis des Trainings, löschte man alle übrigen Punkte, erhielte man immer noch dasselbe Ergebnis.

Nur im linear trennbaren Fall ergibt sich so ein adäquater Klassifikator, der je nach Abstand der Punkte zur Trennhyperebenen eine wesentlich bessere Generalisierungsleistung als ein einfaches Perzeptron zeigen kann. Möchte man zulassen, daß nicht notwendig alle Punkte korrekt klassifiziert werden müssen, – etwa weil die Menge nicht linear trennbar ist oder der Abstand zur Trennhyperebenen zu klein würde – so modifiziert man wie folgt: Die zu erfüllenden Ungleichungen werden zu

$$y_i(\mathbf{w}^t\mathbf{x}_i) \geq 1 - \xi_i$$

mit Variablen $\xi_i \geq 0$. Für $\mathbf{x}_i < 1$ bedeutet das, daß $\mathbf{x}_i$ korrekt ist, wobei für $\mathbf{x}_i \in]0, 1[$ der minimale Abstand zur Trennhyperebenen unterschritten wird. Im Fall $\xi_i \geq 1$ liegt der Punkt entweder direkt auf oder auf der falschen Seite der Trennhyperebenen. Die Anzahl der Fehler kann also durch $\sum \mathbf{x}_i$ beschränkt werden. Minimiert wird jetzt

$$\frac{1}{2}|w|^2 + C \sum \xi_i$$

mit einer zu wählenden Konstante $C > 0$, die die Gewichtung der Fehler bestimmt. Inklusive Lagrangemultiplikatoren ist also die Funktion

$$\frac{1}{2}|\mathbf{w}|^2 + C \sum \xi_i - \sum \alpha_i (y_i\mathbf{w}^t\mathbf{x}_i - 1 + \xi_i) - \sum \mu_i \xi_i$$

mit $\alpha_i \geq 0$, $\mu_i \geq 0$ zu betrachten. Die Kuhn-Tucker-Bedingungen führen also zu den Gleichungen $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$, $C - \alpha_i - \mu_i = 0$, $\mu_i \xi_i = 0$, $\alpha_i(y_i\mathbf{w}^t\mathbf{x}_i - 1 + \xi_i) = 0$. Es ist notwendig $\alpha_i \leq C$ für $\mu_i \geq 0$, für $\mu_i \neq 0$, d.h. $\alpha_i < C$ ist $\xi_i = 0$. Man erhält also als zu minimierende Funktion wieder

$$\sum \alpha_i - \frac{1}{2} \sum \alpha_i\alpha_j y_i y_j \mathbf{x}_i^t\mathbf{x}_j$$

mit Nebenbedingung $0 \leq \alpha_i \leq C$. Ein Lösungsvektor berechnet sich als $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$.

Neben der Möglichkeit von Fehlern legt es die SVM nahe, lineare Trennbarkeit einfach durch Abbilden der Daten in einen hochdimensionalen Raum zu erzwingen. Statt der ursprünglichen Daten betrachtet man also künstlich um verschiedene Merkmale erweiterte Daten, die, sofern man die Dimension nur groß genug wählt, linear trennbar werden. Bei einem einfachen Perzeptron würde sich dieses Vorgehen verbieten, da mit wachsender Eingabedimension die Generalisierungsleistung sinkt. Bei der SVM hängt die Generalisierung nicht von der Dimension, sondern nur vom Abstand der Daten zur Trennhyperebene ab – es können sich also wesentlich bessere Schranken ergeben. Das prinzipielle Vorgehen ist also, zunächst die Daten vermöge einer Funktion Φ in einen hochdimensionalen Vektorraum abzubilden und anschließend im Bildraum eine lineare SVM zu trainieren. Klassifikation erfolgt anschließend, indem man Eingaben $\mathbf{x}$ nach $\mathrm{sgn}(\mathbf{w}^t\Phi(\mathbf{x}))$ abbildet.

Hier besteht allerdings ein Problem: Skalarprodukte in einem hochdimensionalen Raum zu berechnen, ist aufwendig. Das Training erfordert aber häufiges Berechnen der Skalarprodukte. Hier bedient man sich eines kleinen Tricks, des sogenannten **kernel-Tricks**: Φ taucht nur im Zusammenhang mit Skalarprodukten auf. Ist Φ so definiert, daß $\Phi(\mathbf{x})^t\Phi(\mathbf{y}) = k(\mathbf{x}, \mathbf{y})$ für eine einfach zu berechnende Funktion k gilt, dann ist Training effizient möglich. Man optimiert

$$\sum \alpha_i - \frac{1}{2}\sum \alpha_i\alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

unter der Bedingung $0 \leq \alpha_i$. Der Gewichtsvektor ist dann $\mathbf{w} = \sum \alpha_i y_i \Phi(\mathbf{x}_i)$. Um die Ausgabe eines beliebigen Vektors $\mathbf{x}$ zu berechnen, ist also lediglich $\sum \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$ zu bestimmen. Dieses ist zumindest bei einer in der Regel geringen Anzahl von Support Vektoren effizient zu bestimmen. Man benötigt nicht die explizite Funktion Φ, sondern nur den Kern k. Was kommt als solches k in Frage?

Ein Beispiel ist die Abbildung

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^t\mathbf{y})^2,$$

diese berechnet das Skalarprodukt zu

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3, \mathbf{x} \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

oder auch

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3, \mathbf{x} \mapsto \frac{1}{\sqrt{2}}(x_1^2 - x_2^2, 2x_1x_2, x_1^2 + x_2^2)$$

oder auch

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^4, \mathbf{x} \mapsto (x_1^2, x_1x_2, x_1x_2, x_2^2)\ .$$

Man sieht, es ist Φ nicht eindeutig vorgegeben, sogar die Dimension des Bildraumes kann variieren.

Allgemeiner gilt die **Mercer-Bedingung**: für eine stetige und symmetrische Abbildung $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, den sogenannten **Kern**, gibt es eine Abbildung Φ von $\mathbb{R}^n$ in einen evtl. unendlich dimensionalen Vektorraum und eine Darstellung $k(\mathbf{x}, \mathbf{y}) = \sum \Phi(\mathbf{x})_i\Phi(\mathbf{y})_i$ (das ist eine Darstellung als Skalarprodukt in einem evtl. unendlich dimensionalen Bildraum) genau dann, wenn für alle Funktionen $g : \mathbb{R}^n \to \mathbb{R}$ mit $\int g(\mathbf{x})^2 d\mathbf{x} < \infty$ die Bedingung

$$\int k(\mathbf{x}, \mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0$$

gilt.

Mögliche und häufig verwendete Kerne sind:

- $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^t\mathbf{y})^p$, denn

$$\begin{aligned}\int (\textstyle\sum x_iy_i)^p g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} &= \textstyle\sum b_r \int x_1^{r_1}y_1^{r_1}\dots x_n^{r_n}y_n^{r_n}g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y}\\ &= \textstyle\sum b_r \left(\int x_1^{r_1}\dots x_n^{r_n}g(\mathbf{x})d\mathbf{x}\right)^2 \geq 0\end{aligned}$$

  mit sich aus geeigneten Binomialkoeffizienten ergebenden positiven Faktoren b_r ergibt. Man bildet hierbei alle Polynome vom Grad p. Alle Polynome vom Grad $\leq p$ erhält man durch die modifizierte Version $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^t\mathbf{y} + 1)^p$. Je nach sich ergebender Anzahl von Support Vektoren hat der endgültige Klassifikator also die Form

$$\mathbf{x} \mapsto \mathrm{H}\left(\sum \alpha_i y_i(\mathbf{x}_i^t\mathbf{x} + 1)^p\right)\ .$$

  Dieses entspricht einem neuronalen Netz mit einer verborgenen Schicht. Die Neuronen besitzen die Aktivierungsfunktion $x \mapsto x^p$. Die Anzahl der verborgenen Neuronen richtet sich nach der Anzahl der Support-Vektoren. Insbesondere ist diese Anzahl nicht a priori bestimmt, sondern wird automatisch mit passenden Gewichten während des Trainings ermittelt.

  Man sieht, daß sich für homogene Polynome die zugrundeliegende Abbildung Φ als

$$\sum_{r_1+\dots+r_n=p} d_r x_1^{r_1}\dots x_n^{r_n}$$

  mit geeigneten Binomialkoeffizienten d_r. Die Dimension des Raumes der zugehörigen linearen SVM wäre dabei $\binom{n+p-1}{p}$, denn die Anzahl der möglichen Faktoren ergibt sich als die Anzahl der Möglichkeiten, n Zahlen, die sich zu p summieren, zu wählen. Das entspricht dem Zuordnen von p Einsen auf n Stellen mit Mehrfachbelegung. Die einzelnen Funktionen $x_1^{r_1}\dots x_n^{r_n}$ sind linear unabhängig, daher erhält man auch genau diese Dimension. Für inhomogene Polynome sieht das ähnlich aus.

- Eine andere Möglichkeit ist die Wahl $k(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{-|\mathbf{x}-\mathbf{y}|^2/(2\sigma^2)}$. Der zugehörige Raum ist sogar unendlich dimensional, denn bei genügend vielen Support-Vektoren mit großem Abstand voneinander bestimmt der jeweilige Vektor das Verhalten in seiner Nähe eindeutig. Die entstehende Architektur entspricht einem Netz, das zunächst ähnlich zu einem Kohonennetz die Ähnlichkeit zu den Support Vektoren bestimmt und anschließend gewichtet über die Abstände aufsummiert.

- Ein sigmoiden Neuronen entsprechender Kern ist die Wahl $\tanh(\kappa\mathbf{x}^t\mathbf{y} - \delta)$, welches zu einem einschichtigen sigmoiden Netz führt. Dieses stellt allerdings nur für bestimmte Wahlen von κ und δ einen Kern, der die Mercer-Bedingung erfüllt, dar.

Man kann die SVM auch für eine Klassifikation für mehr als eine Klasse adaptieren. Dazu trainiert man mehrere SVMs jeweils darauf, eine gegebene Klasse von den restlichen zu unterscheiden. Im Betrieb ergibt dann eine Eingabe $\mathbf{x}$ die Ausgabe derjenigen Klasse, wo die zugehörige SVM den maximalen Abstand berechnet.

Um die SVM zur Regression einzusetzen, minimiert man

$$\frac{1}{2}|\mathbf{w}|^2 + C\sum(\xi_i + \xi_i')$$

mit der Bedingung $\mathbf{w}^t\mathbf{x}_i - y_i \leq \epsilon + \xi_i$, $y_i - \mathbf{w}^t\mathbf{x}_i \leq \epsilon + \xi_i'$, $\xi_i, \xi_i' \geq 0$. Das entspricht einen Schlauch vom Durchmesser ϵ um die Funktionswerte y_i zu legen, in dessen Bereich die zu prognostizierenden Werte sein sollen.

4.4 Alternativ: Bayesianische Statistik

Eine Alternative zum PAC-Ansatz stellt die Bayesianische Statistik dar, die hier nur kurz angedeutet werden soll. Es soll wie üblich eine Gesetzmäßigkeit von Eingaben x und Ausgaben y gelernt werden, die etwa durch eine unbekannte Funktion, aber auch (insbesondere bei Bayesianischer Statistik nahegelegt) durch eine unbekannte gemeinsame Verteilung der Daten $x \times y$ gegeben sein kann. Es wird eine durch W parametrisiertes Modell der Daten gebildet, etwa eine Netzarchitektur, eine SVM mit Parametern W oder eine Verteilung bestimmter Form. Eine Beispieldatenmenge P liege vor. Zu einem Wert x soll eine wahrscheinliche Ausgabe y bestimmt werden. Im Bayesianischen Ansatz wird eine Verteilung für y gegeben x und die Daten P bestimmt wie folgt:

$$p(y|P,x) = \int p(y,W|P,x)dW\,,$$

d.h. die Wahrscheinlichkeit für y kann man durch Mittelung über die gemeinsame Wahrscheinlichkeit mit allen möglichen Parametern ermitteln. Es ist nach Definition

$$p(y,W|P,x) = p(y|P,x,W)\cdot p(W|P,x)\,.$$

Ist aber W bestimmt, hängt die Ausgabe y nicht mehr von den Daten P ab, da ja W das Modell vollständig beschreibt. Bei einem Netz ist dieses etwa die Verteilung, die der bei Einagbe von x mit Parametern W berechneten Ausgabe die Wahrscheinlichkeit 1 zuweist. Die sogenannte **Bayes-Formel** aus der Statistik besagt

$$p(W|P,x) = \frac{p(P,x|W)\cdot p(W)}{p(P,x)}.$$

Wir nehmen an, daß die Eingabe x alleine nicht von den konkreten Parametern W abhängt. Weiterhin besteht die Trainingsmenge P aus Beispielen (x_i,y_i), die wir genau wie beim PAC-Ansatz auch als unabhängig und identisch verteilt voraussetzen. D.h. es gilt

$$p(P|W) = \prod p((x_i,y_i)|W)\,.$$

Der Nenner berechnet sich als

$$p(P,x) = \int p(W')p(P,x|W')dW' = \int p(W')\prod p((x_i,y_i)|W')dW'\,.$$

Im Prinzip kann man jetzt die uns interessierende Verteilung berechnen als

$$p(y|P,x) = \int \frac{p(y|x,W)\prod p((x_i,y_i)|W)p(W)dW}{\int\prod p((x_i,y_i)|W')p(W')dW'}\,,$$

wobei die beiden Wahrscheinlichkeiten $p(y|x,W)$ und $p((x_i,y_i)|W) = p(y_i|x_i,W)\cdot p(x_i|W) = p(y_i|x_i,W)\cdot p(x_i)$ aufgrund des gewählten Modells zugänglich sind. $p(W)$ ist eine zu wählende a priori Verteilung der Parameter, die angibt, in welchem Bereich die Parameter erwartet werden, wenn man keinerlei Daten zur Verfügung hat. Dieses kann etwa eine Gauß-Verteilung sein, wenn wir ohne Vorwissen erwarten, daß die Gewichte in einem Netz sich symmetrisch um Null verteilen und nicht zu groß werden.

Allerdings sind in der Regel obige Integrale nicht analytisch lösbar und müssen genähert werden. Eine Näherung durch Monte-Carlo Methoden, d.h. Auswertung an genügend vielen Einzelwerten, kommt nur für einen niedrigdimensionalen Parametervektor W in Betracht. Es gibt je nach

Annahmen über das Modell verschiedene mehr oder weniger effiziente Näherungen, auf die hier nicht weiter eingegangen werden soll. Es soll nur ein Spezialfall erwähnt werden, der vom konkreten Vorgehen her wieder auf das übliche Procedere bei neuronalen Netzen führt: Falls man wenig a priori Information über die Parameter besitzt, ist die Dichte der a priori Verteilung $p(W)$ voraussichtlich flach. Bei genügend vielen Daten besitzt die Dichte $p((x_i,y_i)|W)p(W)$ eine Spitze im Bereich des tatsächlichen Wertes $\hat{W}$, falls es nur einen solchen gibt. Gibt man nun dasjenige y als das wahrscheinlichste aus, wo die Dichte obigen Ausdrucks zentriert ist, erhält man die Ausgabe y zu dem Parameter $\hat{W}$, der die Fehler für die gegebenen Daten minimiert. D.h. bei vielen Daten und wenig a priori Information entspricht die Ausgabe y der Ausgabe mit einem den empirischen Fehler minimierenden Parametersatz.

Bei wenigen Daten oder genauem a priori Wissen kann das Ergebnis allerdings anders ausfallen, als beim üblichen Vorgehen. Der Term $p(W)$ sorgt für eine automatische Regularisierung des Modells. Wendet man die Bayesianische Vorgehensweise etwa auf die SVM an, dann können bei geeigneter Wahl von $p(W)$ Vektoren mit kleinem $|W|^2$ präferiert werden und so die Generalisierungsleistung verbessert werden. Bemerkenswert ist, daß man durchaus nicht auf ein Modell beschränkt ist; es können mehrere Netzarchitekturen simultan betrachtet werden, die alle zum Ergebnis beitragen – der Term $p(W)$ sorgt dabei für eine Gewichtung der einzelnen Arcghitekturen, d.h. eine automatische Regularisierung.

Der Bayes Ansatz stellt zunächst eine exakte Berechnung der vorliegenden Wahrscheinlichkeiten dar und ist also per se konsistent. Problematisch ist dabei die Wahl der a priori Verteilung $p(W)$, die eigentlich immer über alle möglichen Modelle erfolgen müßte – eine schlechte Wahl der a priori Verteilung hat eine schlechte Generalisierungsleistung zur Folge, andererseits wird das kanonische Einbinden von Vorwissen leicht ermöglicht. Eine weitere Schwierigkeit stellen die in der Regel notwendigen Näherungen der Integrale dar – im Einzelfall muß die Gültigkeit der jeweiligen Näherungen verifiziert werden. Da dieses der Knackpunkt ist, noch einmal: Die a priori Verteilung stellt eine implizite Regularisierung der Modelle dar, so daß eine gute Generalisierungsleistung gewährleistet ist. Insbesondere sind daher Bayesianische Methoden für den Fall von relativ wenigen Daten gut geeignet.

5 Partiell Rekurrente Netze

Partiell rekurrente Netze dienen zum überwachten Lernen von Vorgängen, wo Zeit eine Rolle spielt. Sie können im Gegensatz zu einfachen feedforward Netzen Sequenzen beliebiger Länge einlesen. Damit können sie sowohl für Zeitreihenverarbeitung und -prognose, als auch zur Simulation von sich zeitlich entwickelnden Systemen verwandt werden. Durch ihre Möglichkeit, Sequenzen beliebiger Länge verarbeiten zu können, stehen sie an der Schnittstelle zu symbolischen Systemen der KI. Die Daten der symbolischen KI zeichnen sich nämlich im Gegensatz zu den für Netze gebräuchlichen Vektoren durch häufig rekursive Strukturen, die a priori unbeschränkten Darstellungsplatz benötigen können, aus. Tatsächlich kann man die Dynamik von rekurrenten Netzen so erweitern, daß sie in gewisser Weise symbolische Terme als Eingabe verarbeiten können.

5.1 Jordan und Elman Netze

Jordan und Elmannetze sind spezielle sehr einfache rekurrente Netzstrukturen. Sie wurden von gleichnamigen Herren zusammen mit einem einfachen Trainingsalgorithmus vorgeschlagen, um Probleme in der Sprachverarbeitung zu lösen. Hauptsächlicher Unterschied zu allgemeinen partiell rekurrenten Netzen ist die Betrachtungsweise und die Art der Darstellung.

Die Dynamik eines einfachen rekurrenten Netzes kann wie folgt beschrieben werden: Eingaben sind durch eine Sequenz $[\mathbf{x}(0), \ldots, \mathbf{x}(T)]$ vorgegeben. Die Aktivierungen zum Zeitpunkt t sind

$$\text{net}_i(t) = \sum_j w_{ji} o_j(t) - \theta_i$$

für alle bis auf die Eingabeneuronen. Die Eingabeneuronen kopieren einfach die Elemente der Sequenz und setzen dieses als ihre Aktivierung und Ausgabe. Die Ausgabe der anderen Neuronen ist

$$o_i(t+1) = f(\text{net}_i(t)) \,.$$

Man startet dabei bei einem Initialkontext $\mathbf{y}$ und betrachtet als Ausgabe des gesamten Netzes die Ausgabe geeigneter Neuronen, die sich nach Einlesen der kompletten Sequenz berechnet hat. Dieses kann man als dynamisches System sehen, dessen Zustand durch den Zustand der Neuronen beschrieben ist und das seinen Zustand je nach Eingabe ändert. Die Übergangsfunktion g von einem Zustand zum nächsten ist durch ein einschichtiges Netz beschrieben. Die Ausgabe ergibt sich durch Projektion auf die Koeffizienten, die Ausgabeneuronen darstellen. Formal berechnet so ein Netz also eine Funktion

$$h \circ \tilde{g}_{\mathbf{y}} : (\mathbb{R}^m)^* \to \mathbb{R}$$

mit einer Funktion h, welches die Projektion auf die Ausgaben ist, und g, welches durch ein einschichtiges Netz gegeben ist, und $\tilde{g}_{\mathbf{y}}$, welches die durch die Übergangsfunktion g induzierte rekursive Abbildung auf Sequenzen ist, die hier durch $(\mathbb{R}^m)^*$ bezeichnet werden, d.h.

$$\begin{aligned} \tilde{g}_{\mathbf{y}}([\,]) &= \mathbf{y}, \\ \tilde{g}_{\mathbf{y}}([x_0, \ldots, x_T]) &= g(x_T, \tilde{g}_{\mathbf{y}}([x_0, \ldots, x_{T-1}]) \,. \end{aligned}$$

Der zweite Teil der Eingaben an g, der sich durch die rekursiven Verbindungen ergibt, wird häufig durch zusätzliche Neuronen, die je die Ausgaben des vorherigen Schritts kopieren, deutlich gemacht. Die so dargestellten Zellen heißen **Kontextzellen**. In dieser Notation bietet es sich an, komplexere Funktionen als g und h zuzulassen, die sich z.B. durch mehrschichtige feedforward Netze berechnen lassen. Tatsächlich ist das keine echte Erweiterung, da komplexere Funktionen in obiger Notation dadurch simuliert werden können, daß man ein Time-Delay in der Eingabe einführt, d.h. bevor ein neuer Wert der Sequenz eingelesen werden darf, wartet man einige rekursive Schaltschritte, währenddessen das Netz eine komplexere Übergangsfunktion ausrechnen kann. Formal folgen also auf jede echte Eingabe einige Dummy-Eingaben.

Nichtsdestotrotz wird die Notation einfacher, wenn komplexere Funktionen g und h zugelassen werden. In dieser Notation stellen **Jordannetze** den Spezialfall dar, daß h die Projektion auf die ersten n Koeffizienten ist und g die Form

$$g : \mathbb{R}^{m+2n} \to \mathbb{R}^{2n} : (\mathbf{x}, \mathbf{y}_1, \mathbf{y}_2) \mapsto (l(\mathbf{x}, \mathbf{y}_1), \lambda_1 \mathbf{y}_1 + \lambda_2 \mathbf{y}_2)$$

mit einem Netz l mit einer hidden Schicht hat. Häufig ist $\lambda_1 = 0$ und $\lambda_2 \in [0, 1[$. In diesem Fall bedeutet das, daß in $\mathbf{y}_1$ zusätzlich zur Eingabe $\mathbf{x}(t)$ die exponentiell abfallend gewichtete Historie der nach einem Schritt berechneten Ausgaben $\mathbf{o}(t) + \lambda \mathbf{o}(t-1) + \lambda^2 \mathbf{o}(t-2) \ldots$ zur Verfügung steht. Diese Sichtweise zeigt die mögliche Beschränkung des Ansatzes: Etwa bei Zeitreihenprognose sind die gewünschten Ausgaben nach jedem Zeitschritt festgelegt. Ein Jordannetz ist dann nur eine etwas umständliche Art, das zusätzliche Merkmal des zeitlich gewichteten Kontexts zur Verfügung zu stellen. Das Netz hat wenig Möglichkeiten, sich stattdessen die relevanten Merkmale der Zeitreihe in den Kontextzellen zu merken.

Demgegenüber propagieren **Elmannetze** die Aktivierungen einer verborgenen Schicht zurück, so daß das Netz selbsttätig die relevante Information extrahieren kann. Formal ist ein Elmannetz ein rekurrentes Netz in obiger Schreibweise, wo g und h je ein feedforward Netz ohne verborgene Schicht darstellen. Die Ausgaben von g und Eingaben von h, quasi die verborgene Schicht des Elmannetzes, beschreibt damit die vom Netz herauszufindenen Zustände des entstehenden dynamischen Systems.

5.2 Trainingsverfahren

Eine Trainingsmenge besteht aus einer Anzahl von Pattern $(\mathbf{x}^i, \mathbf{y}^i)$, wobei jetzt die Eingaben $\mathbf{x}^i$ Sequenzen beliebiger Länge sein können. Bei Zeitreihenprognose oder der Simulation eines dynamischen Systems ist zusätzlich gegeben, daß die Eingabesequenzen Anfangsstücke voneinander darstellen, welche die Eingaben an das System bis zum Zeitpunkt 1, bis zum Zeitpunkt 2, ... darstellen. Der quadratische Fehler eines Netzes $f_{\mathbf{w}}$ kann genau wie bei feedforward Netzen als

$$E = \sum (\mathbf{y}^i - f_{\mathbf{w}}(\mathbf{x}^i))^2$$

definiert werden. In $\mathbf{w}$ sind die Gewichte des Netzes aufgesammelt; Biase werden durch On-Neuronen simuliert. Der Initialkontext $\mathbf{y}$ wird meistens nicht trainiert, sondern z.B. als $\mathbf{0}$ festgelegt. Je nach Fragestellung sind unterschiedliche Trainingsmethoden adäquat:

- **Backpropagation Through Time (BPTT):** Wir formulieren das Verfahren für die ursprünglich vorgeschlagene Dynamik. Analog zu Backpropagation kann man den Fehler E durch Gradientenabstieg minimieren. Dazu benötigt man den Gradienten ∇E. Dieser kann analog zu Backpropagation durch eine Vorwärts- und eine Rückwärtswelle berechnet werden, bedenkt man folgendes: Für jede einzelne Sequenz $\mathbf{x}^i$ berechnet sich die Ausgabe durch T-fache Komposition der Netzfunktion $f_{\mathbf{w}}$, wenn T die Länge der Sequenz darstellt. Analog kann man ein feedforward Netz N betrachten, das sich durch T-faches Hintereinanderhängen des Ausgangsnetzes $N(0)$ ergibt. In diesem feedforward Netz N, das aus T identischen Netzen $N(t)$ besteht, kann man die Ausgaben o_i und Fehlerterme δ_j im $t$ten Netz durch normales Backpropagation in N berechnen. Die Ableitung $\partial E / \partial w_{ij}$ ergibt sich dann, da ja das Gewicht w_{ij} in jeder Kopie $N(t)$ vorkommt, als Summe über die Produkte dieser Terme.

  Natürlich muß bei einer konkreten Implementation nicht das Netz T mal kopiert werden, es reicht, die jeweiligen Aktivierungen $o_i(t)$ und Fehlersignale $\delta_i(t)$ des $i$ten Neurons in je einer zum Neuron gehörigen Liste zu speichern. Eine besondere Situation tritt weiterhin auf, falls die Sequenzen Anfangsstücke voneinander sind, da dann für die unterschiedlichen Sequenzen die Aktivierungen und Fehlerterme großteils übereinstimmen, also nicht komplett neu berechnet werden müssen. Für eine Sequenz $[\mathbf{x}^1, \ldots, \mathbf{x}^T]$ ist dann der quadratische Fehler $E = \sum_{t=1}^{T} E(t)$, mit

  $$E(t) = \begin{cases} 0 & \text{falls im } t\text{ten Schritt keine Ausgabe verlangt ist,} \\ 0.5 \cdot |\mathbf{o}(t) - \mathbf{y}(t)|^2 & \text{falls im } t\text{ten Schritt die Augabe } \mathbf{y}(t) \text{ verlangt ist.} \end{cases}$$

  $E_j(t)$ sei die lineare Differenz $o_j(t) - y_j(t)$ für das $j$te Neuron, falls dieses existiert, und sonst 0. Faltet man das Netz entsprechend der Eingabesequenz aus, erhält man einige identische Kopien, wo jedes Gewicht w_{ij} an verschiedenen Stellen identisch auftritt. Wir schreiben $w_{ij}(t)$ für das Gewicht an der t-ten Stelle des Netzes. Möchte man eine Größe nach w_{ij} ableiten, so ergibt sich das durch die Summe über die Ableitungen nach allen $w_{ij}(t)$ (Kettenregel). Man erhält also

  $$\frac{\partial E}{\partial w_{ij}} = \sum_t \frac{\partial E}{\partial w_{ij}(t)} = \sum_t \frac{\partial E}{\partial \text{net}_j(t)} \frac{\partial \text{net}_j(t)}{\partial w_{ij}(t)} = \sum \delta_j(t) o_i(t-1)$$

mit den Fehlersignalen

$$\delta_j(t) = \begin{cases} \text{sgd}'(\text{net}_j(t))(E_j(t)) & t = T \\ \text{sgd}'(\text{net}_j(t))\,(E_j(t) + \sum_k w_{jk}\delta_k(t+1)) & \text{sonst} \end{cases}$$

Der Aufwand des Verfahrens ist für Sequenzen der Länge T von der Ordnung TW. Um die Signale $o_i(t)$ zu speichern, benötigt man für jedes Neuron Speicherplatz für eine Liste der Länge T.

- **Jordan/Elmantraining:** Bei Jordannetzen steht für hinreichend gute Netze die Aktivierung der Kontextzellen fest; ebenso ist bei Elmannetzen zu erwarten, daß sich die Aktivierung der Kontextzellen nach genügend langem Training nicht mehr stark ändert. Dieses kann man als Anlaß nehmen, so zu tun, als ob die Aktivierung der Kontexzellen konstant sei und nicht von den Gewichten abhängig. Gradientenabstieg stoppt also, nachdem die Fehlersignale einmal durch das Netz g propagiert worden sind, ohne eine explizite oder implizite zeitliche Entfaltung zu betrachten. Dieses Verfahren ist zwar effizient, man benötigt pro Muster nur $\text{O}(W)$ Rechenschritte, die Konvergenz ist aber nicht gewährleistet, wenn man mit schlechten Netzen startet. Daher sollte man bei Jordannetzen evtl. sogenanntes **Teacher-forcing** verwenden, d.h. in die Kontextzellen nicht den tatsächlich prognostizierten, sondern den zu prognostizierenden Wert einsetzen. Bei Elmantraining sollte man evtl. mit BPTT vortrainieren, um die Kontextzellen zu stabilisieren.

- **Real Time recurrent Learning (RTRL):** Für sehr lange Sequenzen oder Situationen, wo man online trainieren muß und die maximale Länge der Sequenz a priori nicht weiß, ist RTRL geeignet. Wir formulieren dieses für die ursprünglich definierte Netzdynamik rekurrenter Netze. Der Fehler für Eingabesequenzen, die je Anfangsstücke voneinander darstellen, setzt sich zusammen aus

$$E = \sum E(t),$$

wobei $E(t)$ den durch den nach dem $t$ten Zeitschritt verlangten Wert $\mathbf{y}(t)$ gegebenen Fehler darstellt. Die Ableitung $\partial E/\partial w_{ij}$ ergibt sich als $\sum_{kt}(o_k(t) - y_k(t))\partial o_k(t)/\partial w_{ij}$. Es gilt

$$\frac{\partial o_k(0)}{\partial w_{ij}} = 0$$

und

$$\frac{\partial o_k(t)}{\partial w_{ij}} = 0$$

für alle Eingabeneuronen k. Wegen $o_k(t+1) = \text{sgd}(\sum_l w_{lk} o_l(t))$ ergibt sich für den Rest die Rekursionsgleichung

$$\frac{\partial o_k(t+1)}{\partial w_{ij}} = \text{sgd}'(\text{net}_k(t)) \left(\delta_k^j o_i(t) + \sum_l w_{lk} \frac{\partial o_l(t)}{\partial w_{ij}} \right),$$

die es ermöglicht, mit dem Aufwand $\text{O}(W^2)$ in den Gewichten die Änderung zu berechnen. Obwohl langsamer als BPTT ist diese Variante interessant, da sie eine online Version nahelegt: Die im Zeitschritt t induzierten Änderungen hängen nur von den Ausgaben und Änderungen des vorigen Zeitschritts ab. Damit kann man sie auch sofort vornehmen, ohne die Fehlersignale stark zu ändern, und also Sequenzen, deren Länge a priori nicht bekannt ist, trainieren. Der benötigte Speicherplatz hängt nicht von T ab.

- **Kombination von RTRL und BPTT:** Betrachtet man die Dynamik in der Form $h \circ \tilde{g}_{\mathbf{y}}$, dann liegt es nahe, den Gradienten innerhalb von g mit effizientem Backpropagation zu berechnen, bei Propagierung durch die Zeitschritte aber mithilfe von RTRL. Damit wäre der Speicherplatzaufwand durch W beschränkt, und der aufwendige RTRL Schritt wäre nur nach jedem Durchlauf durch g durchzuführen. Allgemeiner kann man bei einem rekurrenten Netz, das für eine Eingabe der Länge T trainiert werden soll, den RTRL Schritt nur alle h Schritte durchführen und dazwischen die Fehlersignale durch Backpropagation Through Time propagieren. Wir verwenden zur Herleitung wieder obige Notation.

Es sei $E_i(t)$ der lineare Fehler des Neurons i zum Zeitpunkt t, falls er existiert, und $E(t_1, t_2)$ der quadratische Fehler vom Zeitpunkt t_1 bis zum Zeitpunkt t_2. Der Gesamtfehler ist also $E(0,T)$. Wie eben bezeichne $\partial/\partial w_{ij}(t)$ die Ableitung nach der Kopie des Gewichts w_{ij} im t-ten ausgefalteten Netz. Es ist

$$\frac{\partial E(0, t_0 + h)}{\partial w_{ij}} = \frac{\partial E(0, t_0)}{\partial w_{ij}} + \frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}},$$

folglich kann man die Berechnung der Ableitungen in Summanden über je h Schritte zerlegen und, im online-Modus, wenn etwa die Maximallänge der Eingabe nicht bekannt ist, die Änderungen sofort nach jeweils h Schritten durchführen. Der zweite Summand ergibt sich als

$$\frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}} = \sum_{\tau=1}^{t_0} \frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h} \frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}(\tau)},$$

d.h. er zerlegt sich in Summanden, die sich auf Gewichte vor dem aktuellen Zeitschritt beziehen und daher RTRL-gemäß vorwärtspropagiert werden müssen, und Größen, die sich auf den gerade betrachteten Bereich beziehen und analog zu Backpropagation berechnet werden können. Genauer erhält man für die Summe über $\tau \le t_0$

$$\sum_{\tau=1}^{t_0} \frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}(\tau)} = \sum_k \frac{\partial E(t_0, t_0 + h)}{\partial \text{net}_k(t_0)} \sum_{\tau=1}^{t_0} \frac{\partial \text{net}_k(t_0)}{\partial w_{ij}(\tau)} = \sum_k \delta_k(t_0) q_{ij}^k(t_0)$$

und für $\tau > t_0$

$$\frac{\partial E(t_0, t_0 + h)}{\partial w_{ij}(\tau)} = \delta_j(\tau) o_i(\tau - 1)$$

mit der Backpropagation ähnlich berechneten Große $\delta_k(t) =$

$$\frac{\partial E(t_0, t_0 + h)}{\partial \text{net}_k(t)} = \begin{cases} \text{sgd}'(\text{net}_k(t)) E_k(t) & \text{falls } t = t_0 + h \\ \text{sgd}'(\text{net}_k(t))(E_k(t) + \sum \delta_j(t+1) w_{kj}) & \text{falls } t_0 \le t < t_0 + h \end{cases}$$

und

$$q_{ij}^k(t) = \sum_{\tau=1}^{t} \frac{\partial \text{net}_k(t)}{\partial w_{ij}(\tau)}.$$

Es ist

$$q_{ij}^k(0) = 0$$

und

$$q_{ij}^k(t_0 + h) = \sum_{\tau=1}^{t_0} \frac{\partial \text{net}_k(t_0 + h)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h} \frac{\partial \text{net}_k(t_0 + h)}{\partial w_{ij}(\tau)}$$

$$= \sum_{\tau=1}^{t_0}\sum_l \frac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_l(t_0)}\frac{\partial \text{net}_l(t_0)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h}\frac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_j(\tau)}\frac{\partial \text{net}_j(\tau)}{\partial w_{ij}(\tau)}$$

$$= \sum_l \frac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_l(t_0)} q^l_{ij}(t_0) + \sum_{\tau=t_0+1}^{t_0+h}\frac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_j(\tau)} o_i(\tau-1)\,.$$

Backpropagation ähnlich kann man berechnen

$$\frac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_l(\tau)} = \begin{cases} \delta^l_k & \text{falls } \tau = t_0+h \\ \text{sgd}'(\text{net}_l(\tau))\sum_m w_{lm}\dfrac{\partial \text{net}_k(t_0+h)}{\partial \text{net}_m(\tau+1)} & \text{falls } h \le \tau < t_0+h\,. \end{cases}$$

Insgesamt berechnet man also zunächst in einer Vorwärtswelle o_i für die neuen h Schritte (Aufwand hn^2), in zwei Rückwärtswellen über h Schritte die Größen $\delta_k(t)$ und $\partial \text{net}_k(t_0+h)/\partial \text{net}_l(\tau)$ (Aufwand hn^2 bzw. hn^3), die von einem Block zum nächsten propagierten Größen q^k_{ij} (Aufwand n^4) und letztendlich die Änderung für die Fehler vom Zeitpunkt t_0 bis t_0+h durch die angegebenen Formeln (Aufwand n^3+hn^2). Dieses ist aber insgesamt über h Schritte verteilt, so daß man einen Aufwand n^3 erhält, wenn h mindestens dieselbe Größenordnung wie n hat. Der Speicheraufwand ist durch die Ordnung n^3+hn beschränkt, insbesondere also von der Maximallänge einer Sequenz unabhängig, da man sich über die Schritte hinaus lediglich die Größen q^k_{ij} und die letzten Ausgaben merken muß.

Alle beschriebenen Verfahren stoßen auf numerische Schwierigkeiten, sobald das Problem der sog. **long-term-dependencies** auftaucht. D.h. Bereiche der Eingabesequenz beeinflussen stark Bereiche, die erst nach einem langen Zeitraum folgen. Um adäquate Änderungen zu bewirken, müßten die Ausgaben die weit zurückliegenden Stellen vermöge der Fehlersignale erreichen. Betrachtet man in obigen Formeln die Fehlersignale, dann fällt der Faktor sgd$'$ auf, der in jeder Schicht hinzukommt. Dieser Faktor ist im besten Fall 0.25, im schlimmsten Fall sehr klein, so daß die Fehlersignale mit zunehmender Distanz zwischen Ausgabe und sie verursachender Eingabe exponentiell abnehmen.

5.3 Approximationseigenschaften

Es stellt sich auch bei partiell rekurrenten Netzen die Frage, ob sie in geeignetem Sinne approximationsuniversell sind, d.h. jede Gesetzmäßigkeit, die sie darstellen sollen, mit einer geeigneten Architektur auch darstellen können. Einfach ist die Situation, sofern eine Funktion des Formates $h\circ\tilde{g}_y$ mit stetigem h ung g approximiert werden soll, d.h. eine Funktion mit a priori rekursiver Gestalt. Dann kann man die Funktionen h und g einzeln durch je ein feedforward Netz H und G auf Kompakta beliebig gut annähern, so daß die Komposition $H\circ\tilde{G}_y$ aufgrund der gleichmäßigen Stetigkeit auf Kompakta die zu approximierende Funktion $h\circ\tilde{g}_y$ beliebig gut für Sequenzen einer Maximallänge T mit Einträgen in Kompakta annähert. Es reicht eine verborgene Schicht in G und in H.

Schwieriger wird die Situation, falls beliebig lange Sequenzen oder nicht a priori rekursive Funktionen approximiert werden sollen oder man, etwa im Fall von endlich vielen Daten, obere Schranken für die Anzahl der benötigten Neuronen herleiten will. Eine Funktion besitzt eine **lokale Linearität**, falls die Funktion in der Umgebung von mindestens einem Punkt, etwa x_0, stetig differenzierbar ist und die Ableitung in x_0 nicht verschwindet.

Satz 5.1 *Sei $\Sigma\subset\mathbb{N}$ endlich, $f:\Sigma^*\to\mathbb{R}^m$ eine Funktion und $x_1,\dots,x_n\in\Sigma^*$. Dann kann man ein rekurrentes Netz $h\circ\tilde{g}_y$ finden mit $h\circ\tilde{g}_y(x_i)=f(x_i)$ für alle i. h ist ein Netz mit der Identität als Aktivierungsfunktion in der Ausgabe und einer verborgenen Schicht mit mn Neuronen mit einer squashing-Aktivierungsfunktion. g besitzt keine hidden Schicht und nur ein Ausgabeneuron mit einer lokalen Linearität.*

Beweis: b sei die Anzahl der Dezimalstellen, die die Werte in Σ maximal benötigen. Die Abbildung

$$g:\Sigma\times\mathbb{R}\to\mathbb{R}, (x,z)\mapsto(x+z)\cdot(0.1)^b$$

induziert die Abbildung $\tilde{g}_y$, welche einfach nur die Dezimalstellen der Eingaben erweitert um führende Nullen auf b Stellen hintereinanderschreibt. Beginnt man mit einem Kontext y, der keinem Wert aus Σ entspricht, dann ist also $\tilde{g}_y$ injektiv aus Σ^*. g kann mit einem Neuron mit der Identität berechnet oder mit einer Aktivierungsfunktion σ mit lokalen Linearität wegen

$$\frac{\sigma(x_0+\epsilon x)-\sigma(x_0)}{\epsilon\sigma'(x_0)}\approx x$$

approximiert werden. Die zusätzlichen linearen Terme können dabei in die Gewichte integriert werden. Auf den injektiven Bildern der Eingabesequenzen kann die gewünschte Ausgabe mit einem feedforward Netz aufgrund der schon nachgewiesenen Approximationseigenschaften exakt berechnet werden. □

Stammen die Eingaben nicht aus einem endlichen Alphabet, sondern aus einem reellen Vektorraum, muß man einen Diskretisierungsschritt vorschalten, da die Stelligkeit der Eingabe a priori unbeschränkt ist.

Satz 5.2 *Sei $f:(\mathbb{R}^l)^*\to\mathbb{R}^m$ eine Funktion und $x_1,\dots,x_n\in(\mathbb{R}^l)^*$. Dann kann man ein rekurrentes Netz $h\circ\tilde{g}_y$ finden mit $h\circ\tilde{g}_y(x_i)=f(x_i)$ für alle i. h ist ein Netz mit der Identität als Aktivierungsfunktion in der Ausgabe und einer verborgenen Schicht mit mn Neuronen mit einer squashing-Aktivierungsfunktion. g besitzt ein Ausgabeneuron mit einer lokalen Linearität und eine hidden Schicht mit $n(n+1)$ Neuronen mit einer squashing Aktivierungsfunktion.*

Beweis: Die Eingabesequenzen unterscheiden sich jeweils in mindestens einer Stelle voneinander. D.h. man findet $2n$ Zahlen, so daß, ersetzt man alle bis auf diese $2n$ Koeffizienten durch beliebige Zahlen, x_1 von allen anderen x_i verschieden bleibt, x_2 von $x_3,\dots,x_n$ verschieden bleibt, ... Insgesamt kann man also alle bis auf $n(n+1)$ Koeffizienten beliebig ersetzen, ohne daß Sequenzen identisch werden. Je Dimension $i=1,\dots,l$ werden die jeweiligen Koeffizienten, die an dieser Stelle zu einer Unterscheidung dienen, aufgelistet und Zahlen $b^i_1,\dots,b^i_{n_i}$ zwischen den Werten gewählt. Sei $q=\max_i n_i$. Die Abbildung p

$$(x_1,\dots,x_l)\mapsto 1+\sum_{i=1}^{l} q^{i-1}\sum_{j=1}^{n_i}\text{H}(x_i-b^i_j)$$

bildet die Koeffizienten auf Werte in $\{1,\dots,q^l\}$ ab, so daß dieses einer Notation zur Basis q der Intervallindizes $]-\infty,b^i_1]$, $[b^1_i,b^2_i]$, ... der einzelnen Koeffizienten entspricht. Die Abbildung

$$g:(x,z)\mapsto(p(x)+z)\cdot(0,1)^{l\lceil\log q\rceil}$$

induziert eine auf den Eingaben injektive Abbildung $\tilde{g}_0$. H kann durch eine squashing-Funktion, die Linearität vermöge einer Aktivierung mit lokaler Linearität approximiert werden. Ein feedforward Netz h, daß die verschiedenen Bilder auf die gewünschten Ausgaben abbildet, vervollständigt die Interpolation. □

Diese Ergebnisse liefern obere Schranken für jede konkrete Trainingssituation: Beim Training eines sigmoiden rekurrenten Netzes benötigt man lediglich eine verborgene Schicht im Teilnetz g bzw. h mit größenordnungsmäßig maximal n^2 bzw. n Mustern. Für symbolische Daten kann g sogar unabhängig von der konkreten Aufgabe gewählt werden, ein Neuron reicht dann.

Es ist eine interessante Frage, ob auch Funktionen als ganzes approximiert werden können. Dazu sei eine Funktion von Listen in einen reellen Vektorraum dann und nur dann stetig bzw. meßbar, wenn jede Einschränkung auf Sequenzen fester Länge stetig bzw. meßbar ist. Meßbar bedeutet hierbei Borel-meßbar. Diese Eigenschaft besitzt so gut wie jede (nicht künstlich eben zum Zwecke eines Gegenbeispiels erdachte) Funktion. Dann gilt:

Satz 5.3 *Sei P ein Wahrscheinlichkeitsmaß auf den Sequenzen $(\mathbb{R}^i)^*$, $f : (\mathbb{R}^i) \to \mathbb{R}^m$ meßbar und $\epsilon > 0$. Dann gibt es ein rekurrentes Netz $h \circ \tilde{g}_y$ mit*

$$P(x \mid |h \circ \tilde{g}_y(x) - f(x)| > \epsilon) < \epsilon\,.$$

h und g besitzen je maximal eine hidden Schicht einer squashing Aktivierung bzw. einer Aktivierung mit lokaler Linearität, die Kodierungsdimension ist 1. Die Ausgabeneuronen besitzen je eine lokale Linearität. Für symbolische Daten kann man h sogar ohne hidden Schicht wählen.

Beweis: Die technischen Details wurden schon gezeigt: Man kann nämlich durch allgemeine Überlegungen das Problem auf eines von quasi endlich vielen Eingaben reduzieren. Genauer: Lange Listen sind unwahrscheinlich, daher reicht es, f nur auf Listen beschränkter Länge mit Konfidenz $\epsilon/2$ zu approximieren. Meßbare Abbildungen auf dem $\mathbb{R}^m$ kann man beliebig gut durch stetige Abbildungen approximieren, daher kann man f als stetig annehmen. Ferner sind auch Listeneinträge mit groäsen Koeffizienten unwahrscheinlich und müssen also nicht weiter betrachtet werden. Auf den verbleibenden Eingaben ist f sogar gleichmäßig stetig, so daß man die Eingabebereiche für die Koeffizienten in Intervalle einteilen kann mit folgender Eigenschaft: Kennt man von jedem Koeffizienten lediglich die Intervallnummer, dann kann man die Ausgabe von f auf dem uns interessierenden Bereich bis auf $\epsilon/2$ bestimmen.

Damit ist die Aufgabe quasi schon gelöst: Die Intervallgrenzen werden mit $b_1, b_2, \ldots, b_q$ aufsteigend durchnummeriert. Die Abbildung p

$$(x_1, \ldots, x_l) \mapsto 1 + \sum_{i=1}^{l} q^{i-1} \sum_{j=1}^{n_i} \mathrm{H}(x_i - b^i_j)$$

berechnet eine eindeutige Repräsentation der Intervallindizes. Daher berechnet die durch

$$g : (x, z) \mapsto (p(x) + z) \cdot (0, 1)^{l \lceil \log q \rceil}$$

induzierte Abbildung eine Repräsentation der gesamten Sequenz.

Auf den so berechneten (endlich vielen) Werten kann man jede gewünschten Ausgaben, die bis auf $\epsilon/2$ festgelegt sind, erhalten. Die Funktion h berechne diese. Man beachte, daß h als Approximation einer stetigen Funktion in der Maximumnorm gesehen werden kann und also leichte Störung der Eingaben toleriert. Die Identität in g kann man durch eine lokale Linearität gleichmäßig approximieren, die Perzeptronaktivierung kann außerhalb von Null durch eine squashing Funktion gleichmäßig approximiert werden. Für geeignete Wahl der Intervallgrenzen kann man also $\tilde{g}_y$ bis auf eine Menge beliebig kleiner Wahrscheinlichkeit beliebig gut auf den uns interessierenden Bereichen approximieren. □

Bei feedforward Netzen wurde ein in gewisser Weise schärferes Resultat erhalten, da Approximation in der Maximumnorm erreicht wurde. Bei rekurrenten Netzen ist dieses nicht möglich, wenn

- Eingabesequenzen beliebiger Länge aus einem einelementigen Alphabet und eine unbeschränkte Ausgabe zugelassen sind oder

- Eingabesequenzen beliebiger Länge aus einem zweielementigen Alphabet und eine binäre Ausgabe zugelassen sind oder

- Eingabesequenzen beschränkter Länge aus einem reellen Vektorraum und eine reelle Ausgabe, aber nur eine unabhängig von der Maximallänge beschränkte Kodierungsdimension zugelassen sind.

In allen Fällen kann man ein nicht in der Maximumnorm approximierbares Gegenbeispiel konstruieren [Hammer]. Allerdings ist Approximation in der Maximumnorm bei unbeschränkter Kodierungsdimension und beschränkter Länge trivial möglich: Der rekurrente Part g schreibt lediglich die Eingaben sukzessive in einen Vektorraum hoher Dimension, d.h. transformiert die Sequenz in einen Vektor der Dimension = Eingabelänge, der Part h hat dann lediglich noch die Aufgabe, eine stetige Funktion zwischen reellen Vektorräumen zu approximieren. Weiterhin ist eine Approximation in der Maximumnorm auf Sequenzen mit Elementen aus einem einelementigen Alphabet und beschränkter Ausgabe möglich – ein Fall, der für die Praxis unerheblich ist, der allerdings die sogenannte Super-Turing-Universalität von rekurrenten Netzen demonstriert. Wir kommen hierauf nochmal zurück, wenn wir rekurrente Netze mit Turingmaschinen vergleichen werden.

5.4 Lernbarkeit

Auch hier soll zunächst die VC- bzw. Pseudodimension abgeschätzt werden. Sei dazu durch $\mathcal{F}$ eine rekurrente Architektur mit W Parametern, N Neuronen und Aktivierungsfunktion σ gegeben. Zunächst nehmen wir an, die Länge der Eingaben sei durch T beschränkt. Man kann einige Änderungen vornehmen: Biases können durch On-Neuronen simuliert werden, ebenso kann ein Initialkontext durch eine zusätzliche Eingabedimension, die nur zu Beginn 1 und anschließend 0 ist, durch Gewichte ersetzt werden. Eingaben, die kürzer als T sind, kann man zu Eingaben der Länge T mit den gleichen Ausgaben erweitern, indem man die Sequenz durch Nullen ergänzt. Damit erhalten wir aber auch schon obere Schranken: Man kann einfach das rekurrente Netz formal für die Maximallänge T ausfalten und die VC bzw. Pseudodimension des zugehörigen feedforward Netzes abschätzen. Dieses Netz hat NT Neuronen und W verschiedene, WT teilweise gleiche Gewichte. Das ergibt als Schranke für die Dimension

$$\begin{cases} \mathrm{O}(W \ln(TN)) & \text{falls } \sigma \text{ linear ist,} \\ \mathrm{O}(WTN \ln d) & \text{falls } \sigma \text{ ein Polynom vom Grad } d \geq 2 \text{ ist,} \\ \mathrm{O}(WT \ln(WT)) & \text{falls } \sigma = \mathrm{H}, \\ \mathrm{O}(W^2 N^2 T^2) & \text{falls } \sigma = \mathrm{sgd}. \end{cases}$$

Man kann durch ein Abzählargument für $\sigma = \mathrm{H}$ zu der Ordnung $WN + W \ln(WT)$ verbessern [Koiran, Sontag]. Bemerkenswert ist, daß in allen Schranken noch die Größe T vorkommt. Das besagt, daß für beliebige Eingaben die Schranken unendlich werden. Als untere Schranken erhält man jeweils durch konkrete Konstruktionen [Koiran, Sontag; DasGupta, Sontag]

$$\begin{cases} \Omega(W \ln(T/W)) & \text{falls } \sigma \text{ linear ist,} \\ \Omega(WT) & \text{falls } \sigma \text{ ein Polynom vom Grad } d \geq 2 \text{ ist,} \\ \Omega(W \ln(WT)) & \text{falls } \sigma = \mathrm{H}, \\ \Omega(WT) & \text{falls } \sigma = \mathrm{sgd}. \end{cases}$$

Die Schranken sind also notwendig von T abhängig. Man behält sogar eine Abhängigkeit von T, wenn man die fat-shattering Dimension statt der Pseudodimension und beschränkte Gewichte und Eingabesequenzen mit Einträgen aus einem beschränkten Alphabet betrachtet. Als unmittelbares Korollar erhält man also:

Korollar 5.4 *Eine feste rekurrente Architektur ist unter realistischen Bedingungen nicht verteilungsunabhängig PAC-lernbar.*

Man muß also genauer hinsehen. Eine Möglichkeit bietet die Überdeckungszahl, die ja verteilungsunabhängige PAC Lernbarkeit charakterisiert.

Satz 5.5 *X seien Sequenzen mit Einträgen in $\mathbb{R}^l$, X_t seien die Sequenzen der Maximallänge t, $\mathcal{F}$ sei eine rekurrente Architektur mit Ausgaben in $[0,1]$, P sei ein Wahrscheinlichkeitsmaß auf X, $\epsilon, \delta \in [0,1]$ und t so gewählt, daß $p_t = P(X_t) \geq 1 - \epsilon/8$. Dann gilt*

$$P^m(\mathbf{x} \mid \sup_{f,g\in\mathcal{F}} |d_P(f,g) - \hat{d}_m(f,g,\mathbf{x})| > \epsilon) \leq \delta$$

für

$$m = \mathrm{O}\left(\frac{1}{\epsilon^2\delta} + d_t\frac{1}{\epsilon^2}\ln\left(\frac{1}{\epsilon}\ln\frac{1}{\epsilon}\right)\right)$$

falls $d_t = \mathcal{VC}(\mathcal{F}|X_t)$ or $\mathcal{PS}(\mathcal{F}|X_t)$ endlich ist. Ist $d_t = \mathit{fat}_{\epsilon/512}(\mathcal{F}|X_t)$ endlich, gilt das auch für

$$m = \mathrm{O}\left(\frac{1}{\epsilon^2\delta} + d_t\frac{1}{\epsilon^2}\left(\ln\frac{d_t}{\epsilon^2}\right)^3\right)\,.$$

Beweis: Man kann die Abweichung des tatsächlichen und empirischen Fehlers abschätzen durch

$$\begin{array}{rl} & P^m(\mathbf{x}\in X^m \mid \sup_{f,g\in\mathcal{F}} |d_P(f,g) - \hat{d}_m(f,g,\mathbf{x})| > \epsilon) \\ \leq & P^m(\mathbf{x}\in X^m \mid \text{maximal } m(1-\epsilon/4) \text{ Einträge in } \mathbf{x} \text{ sind in } X_t) \\ & + P_t^{m'}(\mathbf{x}\in X_t^{m'} \mid \sup_{f,g\in\mathcal{F}} |d_{P_t}(f|X_t, g|X_t) - \hat{d}_{m'}(f,g,\mathbf{x})| > \epsilon/8) \end{array}$$

wobei P_t die von P auf X_t induzierte Wahrescheinlichkeit ist, $m' = m(1-\epsilon/4)$. Dieses gilt, da d_P von d_{P_t} maximal $3\epsilon/8$ abweicht. Falls ein Bruchteil $\epsilon/4$ in $\mathbf{x}$ gestrichen wird, ändert sich $\hat{d}_m$ um maximal $\epsilon/2$. Die Tschebychev-Ungleichung hilft, den ersten Term durch

$$\frac{64p_t(1-p_t)}{m\epsilon^2}$$

abzuschätzen. Wie wir schon gesehen haben, kann man den zweiten Term durch

$$2\mathrm{E}_{P_t^{2m'}}(2N(\epsilon/128, \mathcal{F}|\mathbf{x}, \hat{d}_{2m'})^2)\mathrm{e}^{-m'\epsilon^2/2048}$$

beschränken. Die empirische Überdeckungszahl ist durch

$$2\left(\frac{512\mathrm{e}}{\epsilon}\ln\frac{512\mathrm{e}}{\epsilon}\right)^{d_t}$$

mit $d_t = \mathcal{PS}(\mathcal{F}|X_t)$ bzw. $\mathcal{VC}(\mathcal{F}|X_t)$ oder

$$2\left(\frac{2\cdot 256^2 m'}{\epsilon^2}\right)^{d_t\ln(512\,\mathrm{e}\,m'/(d_t\epsilon))}$$

mit $d_t = \mathit{fat}_{\epsilon/512}(\mathcal{F}|X_t)$ beschränkt. Der gesamte Ausdruck ist maximal δ für

$$m \geq \max\left\{\frac{2\cdot 64p_t(1-p_t)}{\epsilon^2\delta}, \frac{4\cdot 2048}{3\epsilon^2}\left(\ln\frac{4}{\delta} + d_t\lg\left(\frac{512\mathrm{e}}{3\epsilon}\ln\frac{512\mathrm{e}}{3\epsilon}\right)\right)\right\}$$

bzw.

$$\begin{array}{c} m \geq \max\left\{\frac{2\cdot 64p_t(1-p_t)}{\epsilon^2\delta}, \frac{8\cdot 2048}{3\epsilon^2}\ln\frac{4}{\delta}, \frac{8\cdot 2048}{\epsilon^2}d_t\ln\left(\frac{512\mathrm{e}}{d_t\epsilon}\right)\cdot\ln\left(\frac{2\cdot 256^2}{\epsilon^2}\right),\right. \\ \frac{16\cdot 2048}{\epsilon^2}d_t\left(\ln\frac{512\mathrm{e}}{d_t\epsilon} + \ln\frac{2\cdot 256^2}{\epsilon^2}\right)\cdot\ln\left(\frac{16\cdot 2048}{\epsilon^2}d_t\left(\ln\frac{512\mathrm{e}}{d_t\epsilon} + \ln\frac{2\cdot 256^2}{\epsilon^2}\right)\right), \\ \left.\frac{72\cdot 2048}{\epsilon^2}d_t\left(\ln\left(\frac{72\cdot 2048}{\epsilon^2}d_t\right)\right)^2\right\}. \end{array}$$

□

Folglich kann man Generalisierung garantieren, sofern die Wahrscheinlichkeit langer Sequenzen a priori beschränkt werden kann. Genauer:

Korollar 5.6 *$\mathcal{F}$ sei eine feste rekurrente Architektur. X_t seien die Eingabesequenzen der Maximallänge t. P sei ein Wahrscheinlichkeitsmaß auf den Eingaben. t sei so gewählt, daß $P(X\backslash X_t) \leq \epsilon/8$. Dann ist für jeden Lernalgorithmus h die Ungleichung*

$$P^m(\mathbf{x} \mid \sup_{f\in\mathcal{F}} |d_P(f, h_m(\mathbf{x},f)) - \hat{d}_m(f, h_m(\mathbf{x},f),\mathbf{x})| > \epsilon) \leq \delta$$

gültig, falls die Anzahl der Beispiele gemäß dem obigen Satz gewählt wurde. Die Anzahl ist polynomiell in $1/\epsilon$ und $1/\delta$ falls $P(X_t)$ von der Ordnung $1 - d_t^{-\beta}$ für ein $\beta > 0$ und $d_t = \mathcal{VC}(\mathcal{F}|X_t)$ bzw. $\mathcal{PS}(\mathcal{F}|X_t)$ bzw. $\mathit{fat}_{\epsilon/512}(\mathcal{F}|X_t)$ ist.

Beweis: Die Schranken ergeben sich unmittelbar aus obigem Satz. Sie sind polynomiell in $1/\epsilon$ und $1/\delta$ falls die VC, Pseudo-, oder fat shattering Dimension polynomiell in $1/\epsilon$ und $1/\delta$ ist. Die Bedingung $P(X\backslash X_t) \leq \epsilon/8$ führt dann zu obigen Ungleichungen. □

Also ist zumindest die UCED Eigenschaft sichergestellt, und explizite, von der Verteilung abhängige Schranken für die Generalisierung existieren. Die Anzahl der Beispiele, die für adäquate Generalisierung benötigt werden, kann allerdings auch mehr als exponentiell in $1/\epsilon$ wachsen, sofern lange Sequenzen eine zu große Wahrscheinlichkeit besitzen. Man kann hier explizite Beispiele konstruieren.

5.5 Komplexität

Da das Training von rekurrenten Netzen als Spezialfall das Training von feedforward Netzen enthält, wenn man nur Sequenzen der Länge 1 betrachtet, ist das Training mindestens genauso schwierig. Man erhält also bei der Perzeptronaktivierung in analogen Situationen, d.h. bei variierender Eingabedimension oder variierender Anzahl Neuronen in den ersten zwei verborgenen Schichten NP-schwierige Situationen.

Es stellt sich zusätzlich die Frage, wie sich das Training fester Architekturen verhält. Dieses ist evtl. schwieriger als das Training fester feedforward Architekturen, da ja eine zusätzliche Größe, die Eingabelänge von Sequenzen, auftritt. Die Situation erweist sich auch hier als gutartig:

Satz 5.7 *Sei eine feste rekurrente Architektur mit der Perzeptronaktivierungsfunktion gegeben. Gegeben eine Trainingsmenge, kann man in polynomieller Zeit entscheiden, ob es Gewichte für die Architektur gibt, so daß das entstehende Netz die Daten korrekt abbildet.*

Beweis: Falls eine Lösung existiert, dann gibt es auch eine Lösung, so daß keine Aktivierung exakt Null ist, evtl. müssen die Biases leicht geändert werden, und also auch eine Löasung, so daß die Aktivierungen betragsmäßig mindestens 1 sind, evtl. müssen die Gewichte skaliert werden. Betrachtet man jedes Neuron in der Architektur einzeln, dann ist sein Verhalten durch Gleichungen

$$\mathbf{w}^t\mathbf{x} - \theta \geq 1 \text{ oder } \mathbf{w}^t\mathbf{x} - \theta \leq -1$$

bestimmt, wobei je nach Gewichten $\mathbf{w}$ nur einige der Gleichungen erfüllt sind und sich $\mathbf{x}$ aus den Eingaben ergibt. Genauer kann $\mathbf{x}$ aus den Koeffizienten der Eingabesequenzen und den möglichen Ausgaben der Vorgängerneuronen bestimmt werden. Es steht also zur Bestimmung geeigneter Gewichte eine Anzahl von Ungleichungen zur Debatte, die polynomiell in der Eingabe ist (für den Anteil an $\mathbf{x}$, der durch Koeffizienten der Eingabe bestimmt ist), allerdings exponentiell in den Architekturparametern (für den Anteil an $\mathbf{x}$, der von den Vorgängerneuronen stammt, das kann ein beliebiger binärer Vektor geeigneter Stelligkeit sein). Sofern man bestimmt hat, welche dieser Ungleichungen gelten, kann man in polynomieller Zeit nachrechnen, ob sich für alle Eingaben eine korrekte Ausgabe ergibt.

Für jede mögliche Auswahl von Ungleichungen, die gelten, gibt es maximal $W =$ Anzahl der Parameter Ungleichungen, für die exakte Gleichheit gilt und die die Gewichte eindeutig bestimmen. Damit kann man alle möglichen Auswahlen erhalten, indem man maximal W Ungleichungen auswählt und das zugehörige lineare Gleichungssystem löst, um die Gewichte zu erhalten. Es gibt maximal $\binom{G}{W}$, $G =$ Anzahl der Ungleichungen solche Wahlen. Das ist exponentiell in W, aber polynomiell in G und damit auch polynomiell in der Größe der Trainingsmenge.

Der gesamte Algorithmus ergibt sich als: Aufstellen aller möglichen Ungleichungen, Für jede Auswahl von W der Ungleichungen: Lösen des zugehörigen Gleichungssystems und Testen, ob das sich so ergebende W alle Eingaben korrekt abbildet. Das ist polynomiell in der Darstellungsgröße der Trainingsmenge, allerdings exponentiell in den Architekturparametern. □

5.6 Automaten und Turingmaschinen

Einen Hinweis auf die Mächtigkeit von rekurrenten Netzen erhält man durch den Vergleich mit klassischen Formalismen: Turingmaschinen. Durch das Einfügen von Rekurrenz ist es möglich, Rechnungen beliebiger Länge durchzuführen. Zunächst wollen wir aber einen naheliegenderen Zusammenhang untersuchen, denjenigen zu endlichen Automaten. Er bietet vom praktischen Standpunkt aus gesehen die Möglichkeit, leicht Regelwissen zu integrieren, indem man nicht mit einem beliebigen Netz startet, sondern mit einem Netz, das bekannte Automatenregeln bereits implementiert.

Definition 5.8 *Ein* **endlicher Automat** *ist ein Tupel $A = (\Sigma, Z, s, f, \delta)$ von Eingabesymbolen $\Sigma = \{a_1, \ldots, a_n\}$, Zuständen $Z = \{z_1, \ldots, z_q\}$, einem Startzustand $s \in Z$, einem Endzustand $f \in Z$ und einer Übergangsfunktion $\delta : \Sigma \times Z \to Z$. Rekursives Anwenden von δ auf eine Eingabesequenz aus Σ^* mit Startzustand s definiert eine Funktion $\tilde{\delta}_s : \Sigma^* \to Z$. Die von A* **erzeugte Sprache** *ist die Menge*

$$L(A) = \{x \in \Sigma^* \mid \tilde{\delta}_s = f\}.$$

Σ^* bezeichnet dabei Wörter beliebiger Länge mit Elementen aus Σ. Die Schreibweise deutet schon an, daß die Dynamik eines partiell rekurrenten Netzes die Dynamik von Automaten imitieren kann. Man findet folgendes Resultat:

Satz 5.9 *Sei f eine squashing Funktion. Dann kann man für jeden endlichen Automaten A ein rekurrentes Netz $h \circ \tilde{g}_{\mathbf{y}}$ finden, das A simuliert.*

Beweis: O.B.d.A. nehmen wir $\lim_{x\to\infty} f(x) = 1$, $\lim_{x\to-\infty} f(x) = 0$ an. Vor einem Beweis muß gesagt werden, was ‚simulieren' bedeutet. Wir kodieren die Buchstaben a_i unär in $\mathbb{R}^n$. Zu einem Wort w bezeichne $\mathbf{w}$ den Code in $(\mathbb{R}^n)^*$. Für vorgegebenes ϵ wollen wir dann ein Netz finden, so daß

$$h \circ \tilde{g}_{\mathbf{y}}(\mathbf{w}) \begin{cases} \geq 1 - \epsilon & \text{falls } w \in L(A) \\ \leq \epsilon & \text{falls } w \notin L(A) \end{cases}$$

gilt.

Wir betrachten zunächst die Perzeptronaktivierung. Sei $z = |Z|$. Zustände werden unär in $\mathbb{R}^z$ kodiert. Die Funktion

$$g : \mathbb{R}^n \times \mathbb{R}^z \to \mathbb{R}^z,\ (\mathbf{x}, \mathbf{z}) \mapsto (\vee_{i=1}^{n_j}(\mathbf{x} = \mathbf{e}_{j_{i1}} \wedge \mathbf{z} = \mathbf{e}_{j_{i2}}))_j$$

mit dem $i$ten Einheitsvektor $\mathbf{e}_i$, $n_j =$ Anzahl der Paare $(a_{j_{i1}}, z_{j_{i2}})$ mit $\delta(a_{j_{i1}}, q_{j_{i2}}) = z_j$ und (j_{i1}, j_{i2}) allen Indizes solcher Paare berechnet die kodierte Übergangsfunktion δ. Man kann diese mit einem Netz mit einer hidden Schicht und der Perzeptronaktivierung implementieren. h als die Projektion auf die $i$te Komponente, sofern z_i der Finalzustand ist, vervollständigt die Simulation.

Falls man eine squashing Funktion f betrachtet, dann gibt es Zahlen K_1 und K_2 mit

$$f(x) > 0.9 \quad \forall x > K_1, \quad f(x) < 0.1/z \quad \forall x < K_2.$$

Die Gewichte in obiger Simulation können so gewählt werden, daß die Aktivierung für die hidden Schicht in g mindestens 0.5 bzw. maximal -0.5 beträgt. Tauscht man die binärwertigen, die Zustände repräsentierenden Eingaben durch Eingaben aus, die mindestens die Aktivierung 0.9 bzw. maximal die Aktivierung $0.1/z$ haben, so ergibt sich für die verborgenen Neuronen die Mindestaktivierung 0.4 bzw. Maximalaktivierung -0.4, bei hinreichender Skalierung also die Mindestaktivierung K_1 bzw. Maximalaktivierung K_2, welche zu Ausgaben die > 0.9 bzw. $< 0.1/z$ sind, führen. Analog erhält man für die Ausgabeschicht von g bei unären Eingaben die Werte ≥ 0.5 oder ≤ -0.5, bei bis auf 0.9 bzw. $0.1/z$ approximierten Ausgaben die Aktivierung ≥ 0.4 bzw. -0.4, bei hinreichender Skalierung also $\leq K_2$ bzw. $\geq K_1$. Dieses erlaubt, in g die Aktivierungsfunktion durch f zu ersetzen, ohne die Ausgaben sehr zu ändern. Eine analoge Argumentation gilt für h. □

Insbesondere kann man diese explizite Konstruktion dazu benutzen, Automatenregeln in ein rekurrentes Netz zu kodieren. Zu trainierende Variablitität wird etwa durch das Bereitstellen zusätzlicher Neuronen, deren Gewichte mit kleinen Zufallszahlen initialisiert sind, ermöglicht. Im allgemeinen wird während des Trainings die für den Menschen interpretierbare Struktur als endlicher Automat verloren gehen, die Aktivierungen nicht unäre Form haben. Dieses kann man bedingt erzwingen, indem z.B. die Softmax Aktivierung verwandt wird oder ein Penalty Term $\lambda(\sum x_i - 1)^2$ zur Fehlerfunktion addiert wird. Damit summieren sich die Ausgaben von g tendentiell zu 1, so daß die Dynamik als (evtl. probabilistischer) Automat interpretiert werden kann.

Die Rekurrenz ermöglicht es, so etwas wie Rechnungen mit einem rekurrenten Netz beliebig langer Zeitdauer zu definieren: Das Netz rechnet auf seinen internen Zuständen, indem es g rekursive auf eine (evtl. leere) Eingabe anwendet. Durch die Aktivierung eines Neurons wird angezeigt, ob die Rechnung schon zu Ende geführt wurde. Sobald das der Fall ist, kann die Ausgabe in einem spezifizierten Neuron gelesen werden. Formal (für ein Alphabet Σ sind Σ^+ die Wörter der Länmge ≥ 1):

Definition 5.10 *Ein rekurrentes Netz* **berechnet** *eine (evtl. partielle) Funktion* $f : \{0,1\}^+ \to \{0,1\}$ **auf online Eingaben**, *falls das Netz eine Funktion* $h \circ \tilde{g}_{\mathbf{y}} : (\mathbb{R}^2)^* \to \mathbb{R}^2$ *mit folgenden Eigenschaften berechnet:*

- *Für jedes Wort* $w \in \{0,1\}^+$, *wo* $f(w)$ *definiert ist, gibt es ein* $t \in \mathbb{N}$, *die* **Berechnungszeit**, *mit* $h \circ \tilde{g}_{\mathbf{y}}([(w_1,1),(w_2,1),\ldots,(w_n,1),\underbrace{(0,0),\ldots,(0,0)}_{t\,mal}]) = (f(w),1)$ *und für jedes kürzere Präfix dieser Eingabesequenz der Länge* $\geq |w|$ *gibt das Netz* $(0,0)$ *aus.*
- *Falls* $f(w)$ *nicht definiert ist, ist* $h \circ \tilde{g}_{\mathbf{y}}([(w_1,1),(w_2,1),\ldots,(w_n,1),\underbrace{(0,0),\ldots,(0,0)}_{t\,mal}]) = (0,0)$ *für alle* $t \in \mathbb{N}$.

Das heißt, es gibt jeweils zwei ausgezeichnete Ein- und Ausgabeneuronen. Je eins von diesen beeinhaltet die Ein- bzw. Ausgabedaten, sofern es welche gibt. Das jeweils andere gibt durch den Wert 1 an, daß zur Zeit Daten vorliegen, durch den Wert 0, daß zur Zeit keine Daten vorliegen. So wird es dem Netz ermöglicht, prinzipiell unendlich lange zu rechnen. Eine Alternative ist, die Eingabe nicht einzulesen, sondern in die Aktivierung eines ausgezeichneten Neurons zu kodieren. Dazu wird eine Funktion Kodierungsfunktion $c : \{0,1\}^+ \to \mathbb{R}$ spezifiziert, die injektiv und leicht zu berechnen sein sollte. Die Eingaben können dann vermöge c in die Aktivierung eines Neurons kodiert werden. Eine Rechnung benötigt dann keine zusätzlichen Eingaben mehr, um evtl. beliebig viel Zeit für die Rechnung zur Verfügung zu stellen, erhält das Netz als Eingabe eine evtl. beliebig lange Sequenz mit leeren Eingaben, die durch $\top$ notiert werden.

Definition 5.11 $f : \{0,1\}^+ \to \{0,1\}$ *wird durch ein rekurrentes Netz* **offline berechnet**, *falls ein rekurrentes Netz* $h \circ \tilde{g}_{(\_,\mathbf{y})}$ *existiert, wo der erste Koeffizient des initialen Kontextes frei ist, so daß das Folgende gilt:*

- *Für jedes Wort* $w \in \{0,1\}^+$, *so daß* $f(w)$ *definiert ist, gibt es ein* $t \in \mathbb{N}$ *mit*
$$h \circ \tilde{g}_{(c(w),\mathbf{y})}([\underbrace{\top,\ldots,\top}_{t\,mal}]) = (f(w),1)$$
und für jedes kürzere Präfix dieser Eingabe ist $h \circ \tilde{g}_{(c(w),\mathbf{y})} = (0,0)$.
- *Wenn* $f(w)$ *nicht definiert ist, ist*
$$h \circ \tilde{g}_{(c(w),\mathbf{y})}([\underbrace{\top,\ldots,\top}_{t\,mal}]) = (0,0)$$
für jedes $t \in \mathbb{N}$.

Ein klassischer Formalismus, der spezifiziert, was berechenbar ist, sind Turingmaschinen. Um Rechnungen auf Wörtern aus $\{0,1\}^+$ durchzuführen, werden diese auf ein unendliches Band geschrieben und startend an einer definierten Position in einem definierten Zustand verarbeitet. Dieses geschieht, indem sukzessive das jeweils aktuelle Zeichen gelesen wird und je nach Zustand und gelesenem Zeichen dieses geändert, eine Stelle nach links oder rechts gegangen und ein neuer Zustand angenommen wird. Sobald ein Finalzustand angenommen ist, wird die Rechnung beendet und je nach Finalzustand eine 1 oder 0 ausgegeben. Dieses kann formal definiert werden:

Definition 5.12 *Eine* **Turingmaschine** *ist ein Tupel* $T = (Q,\Sigma,\Gamma,\delta,q_0,F)$ *mit folgenden Bestandteilen:*

- Q *ist eine endliche Menge von Zeichen, das Alphabet der* **Zustände**,
- Σ *ist eine endliche Menge von Zeichen mit* $\Box \notin \Sigma$, *das Alphabet der* **Eingabezeichen**, *in unserem Fall besteht dieses aus* $\{0,1\}$,
- Γ *ist eine endliche Menge von Zeichen mit* $\Sigma \subseteq \Gamma$ *und* $\Box \in \Gamma$, *das Alphabet der* **Bandzeichen**, *in unserem Fall ist dieses immer* $\{0,1,\Box\}$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R,S\}$ *ist eine totale Funktion, die* **Übergangsfunktion**,
- $q_0 \in Q$ *ist der* **Startzustand**,
- $F \subset Q$ *sind die* **Endzustände**, *in unserem Fall immer* $\{y,n\}$.

Eine Rechnung sieht jetzt wie folgt aus: Man startet mit einem Band, auf den das zu verarbeitende Wort $w \in \{0,1\}^+$ geschrieben ist. Man startet im Zustand $q = q_0$ und die aktuelle Leseposition ist das rechteste Zeichen des Eingabewortes, das Zeichen sei z. Je nach der Ausgabe von $\delta(q,z) = (q',z',B)$ schreibt man das Zeichen q' an die aktuelle Stelle, geht in den Zustand z' und bewegt die aktuelle Position um eines nach links, falls $B = L$ ist, um eines nach rechts, falls $B = R$ ist, oder bleibt stehen. Sobald einer der Zustände y oder n erreicht ist, terminiert die Rechnung und es wird bei y die Zahl 1, sonst 0 ausgegeben. Entsprechend berechnet eine Turingmaschine eine partielle Funktion $f : \{0,1\}^+ \to \{0,1\}$, die zu einem Wort obigen Wert 0 oder 1 ausgibt, sofern die Rechnung terminiert.

Satz 5.13 *Man kann Turingmaschinen durch rekurrente Netze mit der semilinearen Aktivierungsfunktion simulieren. Folglich kann jede durch eine Turingmaschine berechenbare Funktion* f *durch ein rekurrentes Netz online und offline berechnet werden.*

Beweis: Eine detailierte Ausführung wird schnell sehr technisch. Daher sollen hier nur die wesentlichen Ideen skizziert werden: Die semilineare Funktion berechnet

$$\mathrm{lin}(x) = \begin{cases} 1 & x \geq 1 \\ x & 0 < x < 1 \\ 0 & x \leq 0 \end{cases}$$

Diese Funktion hat den Vorteil, daß sie im Bereich $[0,1]$ der Identität entspricht, außerhalb eine Perzeptronaktivierung darstellt. Man kann also sowohl lineare Berechnungen als auch beliebige Boolesche Verknüpfungen realisieren. Gegeben eine Eingabe $w \in \{0,1\}^+$, wird diese in einem Netz als Zahl $\sum(2+2\cdot w_i)10^{|w|-i-1}$ kodiert. Die Funktion $(x,y) \mapsto 2(x+1)/10$ induziert diese Kodierung und ist mit einem Netz mit der semilinearen Aktivierung berechenbar. Auf offline Eingaben kann man die Kodierung c entsprechend wählen.

Das Netz simuliert jetzt folgendermaßen eine Turingmaschine: Die Aktivierung zweier Neuronen repräsentiert den aktuellen Bandzustand, das eine Neuron repräsentiert dabei als $\sum w_i 10^{-i}$ die Hälfte links startend von der aktuellen Position mit (von rechts gelesen) $w_1, w_2, \ldots$ entsprechenden Buchstaben, das zweite Neuron repräsentiert analog die Hälfte rechts von der Leseposition (von links gelesen). Der aktuelle Zustand wird unär in der Anzahl der Zustände vielen Neuronen kodiert. Je nach Zustand (kann durch ein Perzeptronnetz getestet werden) und aktuellem Zeichen (kann getestet werden, indem getestet wird, in welchem der Intervalle $[0,0.2[$, $[0.2,0.6[$, $[0.6,1[$ die der linken Bandhälfte entsprechenden Zahl liegt) muß der neue Zustand berechnet, der aktuelle Wert überschrieben und die aktuelle Position angenommen werden. Ersteres wird durch endlich viele Vergleiche in einem Perzeptronnetz ermöglicht. Den aktuellen Zustand überschreiben kann man, indem der Wert des der linken Hälfte entsprechenden Neurons mit 10 multipliziert und je

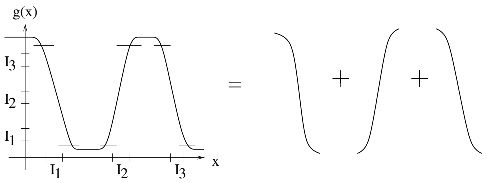


Abbildung 1: Kombination von squashing Funktionen zu g mit $g(I_j) \supset \cup_i I_i$.

nach Aktivierung in $[0,2[$, $[2,6[$ oder $[6,10[$ die Zahl 0, 2 oder 6 subtrahiert wird, anstattdessen entsprechend der neue Wert 0 (für das Symbol $\square$), 2 (für das Symbol 0) oder 6 (für das Symbol 1) addiert wird. Der so erhaltene Wert wird mit $1/10$ multipliziert. Geht man nach jetzt noch nach rechts oder links, dann muß die erste Nachkommastelle des entsprechenden der das Band repräsentierenden Neuronen zum anderen Neuron verschoben werden. Das geht analog, indem man die erste Stelle der einen Aktivierung popt und den berechneten Wert zur Aktivierung des anderen Neurons dazuzählt. $\square$

Allerdings können rekurrente Netze dadurch, daß sie zumindest theoretisch mit unendlicher Genauigkeit auf irrationalen Zahlen arbeiten können, mehr berechnen als Turingmaschinen. Läßt man ihnen beliebig viel Zeit, kann tatsächlich jede Funktion berechnet werden.

Satz 5.14 *Rekurrente sigmoide Netze können jede Funktion $f : \{0,1\}^+ \to \{0,1\}$ offline berechnen.*

Beweis: Es soll wieder nur die Idee skizziert werden. Eine stetige squashing Funktion kann man so linear zu g kombinieren, daß (evtl. nach Verschiebung) das Bild der Intervalle $I_1 = [0.1.0.2]$, $I_2 = [0.45, 0.55]$ und $I_3 = [0.8, 0.9]$ je $I_1 \cap I_2 \cap I_3$ enthält (siehe Skizze). Gibt man eine beliebige Folge von Intervallen $I_{i_1}, \ldots, I_{i_n}$ vor, dann kann man einen geeigneten Startvektor y finden mit

$$\tilde{g}_y(\top^j) \in I_{i_j} \quad \forall j \leq n\,.$$

Da die Menge der möglichen y jeweils kompakt ist, findet man auch einen geeigneten Startwert y wenn man eine Folge von unendlich vielen Intervallen I_i vorgibt.

Startet man mit dem Initialkontext $c(x) = 3^{-\sum_i x_i 2^i + 1}$, dann induziert $g' : x \mapsto 3x$ die Funktion

$$\tilde{g}'_{c(x)}(\perp^j) = 3^{-\sum_i x_i 2^i + 1 + j}\,.$$

Die Identität kann durch die Sigmoide so gut approximiert werden, daß, ersetzt man die lineare Aktivierung in g' durch die Identität, immer noch

$$g'_{c(x)}(\underbrace{[\top, \ldots, \top]}_{n\,\text{mal}}) \begin{cases} < 0.15 & \text{falls } n < \sum_i x_i 2^i\,, \\ \in]0.2, 0.4[& \text{falls } n = \sum_i x_i 2^i\,, \\ > 0.7 & \text{falls } n > \sum_i x_i 2^i \end{cases}$$

gilt, wie man rekursiv nachrechnen kann.

Berechnet man simultan $\tilde{g}_y$ mit Ausgabe o_1 für ein y ensprechend I_i mit

$$i = \begin{cases} 1 & f(i) = 0 \\ 2 & f(i) \uparrow \\ 3 & f(i) = 1 \end{cases}$$

und $\tilde{g}'_{c(x)}$ mit Ausgabe o_2 und schließt die Booleschen Tests

$$(o_1 > 0.9) \wedge o_2 \in]0.2, 0.4[$$

für die Ausgabe und

$$((o_1 > 0.9) \vee (o_1 < 0.1)) \wedge o_2 \in]0.2, 0.4[$$

für das Neuron, das angibt, ob die Rechnung schon beendet ist, an, dann berechnet dieses Netz die Funktion f. $\square$

6 Rekurrente Netze

Voll rekurrente Netze können startend von einem Anfangszustand sich mithilfe der schon definierten synchronen oder asynchronen Schaltdynamik über die Zeit hin entwickeln. Als solches sind sie interessant, sofern Phänomene wie Gedächtnis, spontanes Assoziieren, ... modelliert werden sollen. Um sie allerdings konkret zur Funktionsapproximation oder Assoziation verwenden zu können, müssen wir ihnen eine globale Funktionsweise zuschreiben. Da die Anzahl der Schaltschritte weder wie bei feedforward Netzen durch die Verknüpfungsstruktur, noch wie bei partiell rekurrenten Netzen durch die Länge der Eingabesequenz vorgegeben ist, müssen wir uns hier etwas einfallen lassen.

6.1 Hopfieldnetze

Hopfieldnetze stellen einen 1982 von Hopfield vorgeschlagenen Spezialfall dar, der sich dadurch auszeichnet, daß die Dynamik sich immer irgendwann stabilisiert, als kanonischer Ausgabewert zu einer Eingabe also der irgendwann (asymptotisch oder tatsächlich) erreichte stabile Zustand dienen kann.

Definition 6.1 *Ein* **Hopfieldnetz** *ist ein rekurrentes Netz $(N, \to, \mathbf{w}, \boldsymbol{\theta}, \mathbf{f}, I, O)$ mit $N = I = O$, der Perzeptronaktivierungsfunktion $f = \mathrm{H}$ und $w_{ij} = w_{ji}$ für alle i, j, $w_{ii} \geq 0$ für alle i.*

Bezeichnet W die Gewichtsmatrix, so ist diese also symmetrisch bzgl. der Diagonalen und die Einträge auf der Diagonalen sind positiv. Dieses ist essentiell, damit wir jetzt eine durch das Netz berechnete Funktion definieren können. Wir betrachten zunächst eine asynchrone Schaltdynamik. Bei dieser nehmen wir im Folgenden an, daß jedes Neuron, das aufgrund seiner Aktivierung den Zustand ändern würde, in irgendeinem Schaltschritt auch ausgesucht wird. Das ist etwa der Fall, sofern die Neuronen in einer festen Reihenfolge betrachtet werden oder je aus den Neuronen mit gleicher Wahrscheinlichkeit eines ausgesucht wird.

Ein Zustand $\mathbf{o}$ heißt **stabil**, falls kein Neuron seinen Wert ändern kann, d.h.

$$o_i = H\left(\sum_j w_{ji} o_j - \theta_i\right) \quad \forall i\,.$$

Um auch hier die Rechnungen zu vereinfachen, sind alle Biase durch On-Neuronen realisiert. [Falls man sich daran stört, daß das On-Neuron ja seinen Wert ändern könnte in der gegebenen Schaltdynamik, versieht es mit einem hinreichend großen Gewicht zu sich selbst.] Zu einem Hopfieldnetz mit Zustand $\mathbf{o}$ definieren wir die **Energiefunktion**

$$E(\mathbf{o}) = -\frac{1}{2} \sum_{i,j} o_i w_{ij} o_j = -\frac{1}{2} \mathbf{o}^t W \mathbf{o}.$$

Die Minima der Energiefunktion entsprechen in gewisser Weise stabilen Zuständen des Hopfieldnetzes, wie wir gleich sehen werden.

Satz 6.2 *Bei asynchroner Schaltdynamik wie oben beschrieben schaltet das Hopfieldnetz bei beliebigem Startzustand* $\mathbf{o}$ *in einen stabilen Zustand.*

Beweis: Da jedes Neuron, das seinen Zustand ändern könnte, auch irgendwann gewählt wird, können wir die Schaltdynamik derart verkürzen, daß in jedem Schritt ein Neuron seinen Wert ändert, es sei denn, ein stabiler Zustand ist erreicht. Im Schritt t habe das Neuron i geschaltet. $\triangle\mathbf{o} = \mathbf{o}(t+1) - \mathbf{o}(t)$ sei der Vektor, der die Änderung angibt und also nur an der Stelle i eine 1 oder -1 besitzt, sonst Null ist. Man kann nachrechnen:

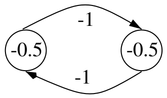
$$\begin{aligned} E(\mathbf{o}(t+1)) - E(\mathbf{o}(t)) &= -\tfrac{1}{2}\mathbf{o}(t+1)^t W\mathbf{o}(t+1) + \tfrac{1}{2}\mathbf{o}(t)^t W\mathbf{o}(t) \\ &= -\tfrac{1}{2}(\mathbf{o}(t)+\triangle\mathbf{o})^t W(\mathbf{o}(t)+\triangle\mathbf{o}) + \tfrac{1}{2}\mathbf{o}(t)^t W\mathbf{o}(t) \\ &= -\tfrac{1}{2}\mathbf{o}(t)^t W\mathbf{o}(t) - \tfrac{1}{2}\triangle\mathbf{o}^t W\mathbf{o}(t) - \tfrac{1}{2}\triangle\mathbf{o}^t W\triangle\mathbf{o} + \tfrac{1}{2}\mathbf{o}(t)^t W\mathbf{o}(t) \\ &= -\triangle o_i \underbrace{\sum w_{ji} o_j(t)}_{\mathrm{net}_i(t)} - \tfrac{1}{2} \underbrace{w_{ii}\triangle o_i^2}_{\geq 0} \end{aligned}$$

Es ist $\triangle o_i \mathrm{net}_i(t) \geq 0$, daher ist obiger Term ≤ 0; 0 kann nur dann auftreten, wenn das Neuron i von 0 auf 1 geschaltet hat, denn nur dann kann $\mathrm{net}_i = 0$ gelten.

Das bedeutet aber Folgendes: In jedem Schritt nimmt die Energie ab. Sie wird maximal dann nicht echt kleiner, wenn ein Neuron von 0 nach 1 geschaltet hat. Da es nur endlich viele Zustände gibt, ist die Energie nach unten beschränkt. Es können jeweils nur endlich viele Werte von 0 nach 1 schalten. Daher muß irgendwann ein stabiler Zustand erreicht sein. □

Dieses ermöglicht es uns, einem Hopfieldnetz eine Funktionalität zuzuordnen. Für ein Hopfieldnetz sei ordnen wir einem Wert $\mathbf{o}$ in $\{0,1\}^{|N|}$ den Wert in $\{0,1\}^{|N|}$ zu, der startend von $\mathbf{o}$ als stabiler Zustand erreicht wird. Dieses ist eine Zuordnung, die von der jeweiligen Schaltreihenfolge abhängt, denn von einem Startvektor ausgehend ist es durchaus möglich, zu mehr als nur einem stabilen Zustand zu gelangen.

Alternativ kann man natürlich das Ergebnis unter synchroner Schaltdynamik betrachten, d.h. in jedem Schritt ist $o_i(t+1) = \mathrm{H}(\mathrm{net}_i(t))$ mit der Perzeptronaktivierung H. Das Ergebnis ist deterministisch, allerdings ist nicht gewährleistet, daß ein stabiler Zustand erreicht wird. Man kann Zyklen bekommen, etwa das Netz



schaltet den Zyklus $(1,1) \to (0,0) \to (1,1) \to \ldots$. Auch wenn ein stabiler Zustand erreicht wird, braucht dieser nicht mit einem der mit asynchroner Dynamik erreichbaren übereinzustimmen. Setzt man in obigem Netz die Verbindung $w_{12} = -2$, $w_{ii} = 1$ und die Biase auf 0.5, dann schaltet synchrone Dynamik von $(1,1)$ in den stabilen Zustand $(0,0)$, asynchrone Dynamik aber je nach gewähltem Neuron in $(1,0)$ oder $(0,1)$. Man kann zeigen, daß die Zyklen bei synchronem Schalten nicht beliebig lang werden können.

Satz 6.3 *Bei synchronem Schalten eines Hopfieldnetzes gibt es maximal Schaltzyklen der Länge* 2.

Beweis: Betrachte zu einem Netz mit Neuronen N das doppelt so große Netz mit Neuronen N und N'. Die Verbindungen im neuen Netz sind 0 für Verbindungen innerhalb von N oder innerhalb

von N' und w_{ij} für Verbindungen $n_i \to n'_j$ und $n'_i \to n_j$. Ist in N ein synchroner Schaltschritt $\mathbf{o}(t) \to \mathbf{o}(t+1)$ anzutreffen, dann kann dieser in folgendem Sinne im doppelt so großen Netz durch asynchrones Schalten simuliert werden: Die Aktivierungen werden mit Tupeln (o, o') bezeichnet, die sich auf die Aktivierung der einen Hälfte N bzw. der anderen Hälfte N' beziehen. Startet man mit $(\mathbf{o}(t), \mathbf{0})$, dann ist die Aktivierung $\mathrm{net}'_i(t)$ dieselbe wie $\mathrm{net}_i(t)$ im Originalnetz. Diese Tatsache wird nicht geändert, sofern ein Neuron in N' schaltet, da ja keine Verbindungen innerhalb N' liegen. Schaltet man also der Reihe nach die Neuronen in N', dann erhält man nach $|N|$ Schritten mit asynchronem Schalten den Zustand $(\mathbf{o}(t), \mathbf{o}(t+1))$.

Wir nehmen jetzt an, es gebe einen Zyklus $\mathbf{o}_1 \to \mathbf{o}_2 \to \ldots \to \mathbf{o}_n$ mit $n > 2$ für N. Dieser führt im größerem Netz und asynchronem Schalten mit spezieller Reihenfolge zu den Aktivierungen

$$(\mathbf{o}_1, \mathbf{0}) \to (\mathbf{o}_1, \mathbf{o}_2) \to (\mathbf{o}_3, \mathbf{o}_2) \ldots \to (\mathbf{o}_1, \mathbf{o}_2)$$

im vergrößerten Netz, wobei jedesmal $|N|$ asynchrone Schaltschritte zusammengefaßt wurden. (Ist n ungerade, so muß man zweimal durch alle $\mathbf{o}_i$ schalten, da man zunächst nur die vertauschte Reihenfolge $(\mathbf{o}_2, \mathbf{o}_1)$ erhält.) Für $n > 2$ wäre dieses aber ein Zyklus, der bei asynchronem Schalten ja nicht auftreten kann. □

Allerdings sind die stabilen Zustände, die bei synchronem und asynchronem Schalten existieren, identisch, denn sie sind in beiden Fällen durch die Eigenschaft

$$\forall i \quad o_i = \mathrm{H}(\mathrm{net}_i)$$

charakterisiert. Sind alle Verbindungen $w_{ii} = 0$, dann sind die stabilen Zustände eines Hopfieldnetzes genau die Energieminima mit einer maximalen Anzahl an 1, wie man sich wie folgt klarmacht: In einem Energieminimum mit einer maximalen Anzahl an 1 kann nicht geschaltet werden, da jeder Schaltvorgang entweder die Energie erniedrigt oder die Anzahl der 1 erhöht. Daher sind alle solchen Zustände Minima.

Ist umgekehrt ein Zustand $\mathbf{o}_1$ gegeben, der kein Energieminimum ist oder keine maximale Anzahl an 1 besitzt, dann gibt es einen Zustand $\mathbf{o}_2$, der sich nur in einer Stelle i von $\mathbf{o}_1$ unterscheidet, und der eine niedrigere Energie oder mehr 1 besitzt. Wie eben kann man nachrechnen, daß

$$E(\mathbf{o}_2) - E(\mathbf{o}_1) = -w_{ii}/2\triangle o_i^2 - \mathrm{net}_i(1)(o_{2i} - o_{1i})$$

für die Aktivierung des Neurons i in $\mathbf{o}_1$ gilt. Ist die Energie gleich, dann ist die Aktivierung 0, d.h. das Neuron i schaltet in einem Schritt von 0 auf 1; ist die Energiedifferenz negativ, dann ist das $\mathrm{H}(\mathrm{net}_i(1)) \neq o_{1i}$, das Neuron schaltet also ebenfalls.

Es ist offensichtlich, daß auch bei $w_{ii} > 0$ alle Energieminima mit einer maximalen Anzahl an 1 stabile Zustände sind. Allerdings können zusätzliche stabile Zustände auftreten, da die Verbindung zu sich selbst eine Selbstverstärkung zur Folge hat. Sofern nur w_{ii} groß genug gewählt ist, werden tatsächlich alle Muster stabil.

Möchte man eine direkte Korrespondenz der Energieminima und stabilen Zustände erreichen, dann sollte man also $w_{ii} = 0$ wählen und zusätzlich die Schaltdynamik zu einer synchronen oder asynchronen Dynamik **mit Gedächtnis** modifizieren, bei der für das/die schaltende/n Neuron/en gilt:

$$o_i(t+1) = \begin{cases} o_i(t) & \text{falls } \mathrm{net}_i(t) = 0 \\ \mathrm{H}(\mathrm{net}_i(t)) & \text{sonst} \end{cases}$$

Mit der Hebb-Regel bzw. Perzeptronlernen sind Hopfieldnetze sehr schnell trainierbar. Dafür kann aber sehr viel Zeit vergehen, bevor ein stabiler Punkt erreicht ist. Diese ist umso länger, je verrauschter das Pattern ist. Die Relaxationszeit kann als Indiz dafür gelten, ob überhaupt ein (verrauschtes) bekanntes Muster als Startvektor gegeben ist, oder ein gänzlich neues.

6.2 Trainingsalgorithmen

Hopfieldnetze werden als Assoziativspeicher verwandt. Zu diesem Zweck sollten sie trainiert werden, daß die durch ein Hopfieldnetz gegebene Funktionalität die zu speichernden Muster auf sich selbst abbildet und ein zu den zu speichernden Mustern ähnliches Muster auf das ähnlichste bekannte Muster abzubilden.

Die Lernaufgabe wird nicht in Form von Beispielen einer zu approximierenden Funktion gegeben, sondern die gewünschten stabilen Zustände werden präsentiert. Diese als stabile Zustände zu erreichen, wird sich schon als teilweise schwierig erweisen. Man versucht, die intendierte Funktionalität dadurch zu erreichen, indem so wenig andere stabile Zustände wie möglich entstehen. Dann sollten unbekannte Muster gegen die jeweils ähnlichsten bekannten Muster konvergieren.

Wir setzen im weiteren w_{ii} immer auf 0, um Selbstverstärkung und damit unerwünschte stabile Zustände zu verhindern. Da die Rechnungen einfacher sind, gehen wir im Folgenden von bipolaren Eingaben aus, d.h. die Muster sind aus $\{-1,1\}^{|N|}$ statt $\{0,1\}^{|N|}$. Die Perzeptronaktivierung wird durch die bipolare Aktivierung sgn ersetzt. Biases sind durch On-Neuronen realisiert. Seien Muster $X^1,\ldots,X^P$ gegeben. Die **Hebb-Regel** setzt für $i\neq j$

$$w_{ij}=\frac{1}{N}\sum_{p=1}^{P}X_i^pX_j^p\,.$$

D.h. Verbindungen zwischen Neuronen, die den gleichen Wert haben, werden verstärkt, Verbindungen zwischen Neuronen mit unterschiedlichem Wert werden vermindert. Würde man die Biases explizit machen, so erhielte man die Summe über alle gewünschten Aktivierungen des betreffenden Neurons.

Es ist nicht sichergestellt, daß hiermit alle Muster stabil sind. Genauer kann man für ein Muster p ausrechnen

$$\mathrm{net}_i^p=\sum_j w_{ji}X_j^p=\frac{1}{N}\sum_j\sum_l X_j^lX_i^lX_j^p=X_i^p+\frac{1}{N}\sum_j\sum_{l\neq p}X_j^lX_i^lX_j^p\,.$$

Sofern der letzte Term betragsmäßig kleiner als 1 ist, ist das Muster also stabil. Insbesondere ist im Fall $P=1$ dieses eine Muster stabil. Sofern eine Eingabe um maximal die Hälfte der Pixel verrauscht ist, strebt sie gegen dieses Muster. Ist mehr als die Hälfte der Pixel verrauscht, strebt die Eingabe gegen das komplementäre Muster, bei dem 1 und -1 vertauscht sind. (Bei einer ungeraden Anzahl an Pixeln ist der Fall, daß genau ein Pixel mehr als die Hälfte verändert ist, nicht klar, da Neuronen die Aktivierung 0 haben können.)

Satz 6.4 *Für orthogonale Muster erlaubt die Hebb-Regel die Speicherung aller Muster.*

Beweis: Orthogonal heißt, daß $\sum_k X_k^iX_k^j=0$ für alle $i\neq j$ gilt. Für solche Muster berechnet sich für obigen Störterm

$$\frac{1}{N}\sum_j\sum_{l\neq p}X_j^lX_i^lX_j^p=\frac{1}{N}\sum_{l\neq p}X_i^l\underbrace{\sum_j X_j^lX_j^p}_{=0}\,.$$

□

Sind die Muster noch hinreichend orthogonal, das heißt die Störterme hinreichend klein, dann können sie auch gespeichert werden. Für zufällige Muster (d.h. jedes Bit ist zufällig und unabhängig voneinander) kann man in verschiedener Weise den Anteil an stabilen Mustern abschätzen. Wir zitieren hier lediglich einige Ergebnisse [Hertz et.al.].

Sei P_e die Wahrscheinlichkeit, daß das Bit i des $p$ten Musters instabil ist. Diese ist offensichtlich für alle i und p gleich und man erhält die Abschätzung

$$P_e=\frac{1}{2}\left(1-\frac{2}{\sqrt{\pi}}\int_0^{\sqrt{|N|/(2P)}}\mathrm{e}^{-x^2}dx\right)\,.$$

Je nach der tolerierten Fehlerwahrscheinlichkeit kann man also das Verhältnis $P/|N|$ der Muster im Vergleich zur Neuronenanzahl wählen. Man erhält z.B. die Werte

P_e	$P/\|N\|$
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61

Fraglich ist trotzdem noch, was sich letztendlich ergibt, da ja die Fehler neue Instabilitäten hervorrufen und letztendlich das ganze Muster sich geändert haben kann, wenn man in einem stabilen Zustand angelangt ist. Man kann berechnen, daß dieser Lawineneffekt ab dem Verhältnis $P/|N|=0.138$ eintritt. Dann sind mit einer Wahrscheinlichkeit von 1.6 von initial 0.0036 die Bits im stabilen Zustand falsch. Bei einem schlechteren Verhältnis wird fast das gesamte Muster gelöscht. Möchte man $P_e<0.01/|N|$ erreichen, so daß mit einer Wahrscheinlichkeit von 99% die Pixel initial richtig und sehr wahrscheinlich auch asymptotisch stabil sind, dann benötigt man ein Verhältnis von

$$P=\frac{|N|}{4\log|N|}$$

d.h. $P/|N|\to 0$ für $N\to\infty$. Besser sieht die Situation aus, falls man sich auf dünn besetzte Muster beschränkt, die nur wenige 1 enthalten. Dieses kommt in der Realität etwa bei Schriftzeichen vor. Formal sei $P(X_i^p=1)=1/(2n^\alpha)$ mit $0\leq\alpha<1$. Dann ist ist das Verhältnis, so daß mehr als 99% der Pixel initial stabil sind, von der Ordnung

$$P/|N|\sim\frac{2\sqrt{|N|}}{5\log|N|}\to\infty\,.$$

Neben der Schwierigkeit, daß nicht alle intendierten Muster stabil sind, taucht das zusätzliche Problem auf, daß es weitere stabile Zustände geben kann. Wir haben schon erwähnt, daß mit einem Muster auch das komplementäre Muster stabil ist. Desweiteren sind Überlagerungen einer ungeraden Anzahl an Mustern mit relativ hoher Wahrscheinlichkeit auch stabil, d.h. Terme der Form

$$\pm X^{p_1}\pm X^{p_2}\pm X^{p_3}\,.$$

Desweiteren konnten auch stabile Zustände nachgewiesen werden, die nichts mit den zu lernende Mustern zu tun haben. Allerdings ist dieses Problem nicht ganz so kritisch, da sich die Attraktorbecken um diese ungewollten Minima als relativ klein erweisen und die Energie noch relativ hoch ist. Boltzmann Maschinen, die wir noch erhalten werden, umgehen genau diese Probleme.

Alternativ zur Hebb-Regel kann man den Perzeptronalgorithmus verwenden, denn das Problem, Muster zu speichern, kann man als Problem, die Neuronen auf eine Patternmenge zu trainieren, auffassen. Um Selbstverstärkung zu verhindern, wird $w_{ii}=0$ gesetzt. Die $N(N-1)/2+N$ verschiedenen übrigen Gewichte w_{ij} $(i\leq j)$ und Biases werden durch den Vektor W beschrieben, wir nummerieren die Koeffizienten durch $(w_{12},\ldots,w_{1n},w_{23},\ldots,w_{2n},\ldots,\theta_1,\ldots,\theta_n)$. Für jedes

Neuron i und Pattern p definiert man das Muster (mit gleicher Nummerierung wie W) $X^{pi}_{ij} = X^p_j$ für $j > i$, $X^{pi}_{ji} = X^p_j$ für $j < i$, $X^{pi}_i = 1$ (On-Neuron für den Bias) und den restlichen Koeffizienten 0. $W^t X^{pi}$ liefert genau die Aktivierung des Neurons i bei Anliegen des Musters p. Das heißt aber, daß alle Muster stabil sind, dann und nur dann, wenn

$$\mathrm{sgn}(W^t X^{pi}) = X^p_i \quad \forall p, i$$

gilt. Das ist eine Trainingsaufgabe für ein einfaches Perzeptron. Die Gewichte des Netzes kann man anschließend durch die Gewichte des Neurons wiedererhalten. Alternative zum Umschreiben des Gewichtsvektors ist folgende Modifikation des Perzeptronalgorithmus (für bipolare Muster):

wiederhole
  betrachte ein Muster X^p
  für ein Neuron i mit $\mathrm{sgn}(\mathrm{net}_i) \neq X^p_i$ ändere für $i \neq j$
    $w_{ij} := w_{ij} + X^p_i X^p_j$;
    $w_{ji} := w_{ji} + X^p_i X^p_j$;
    $\theta_i := \theta_i - X^p_i$;

Es ist offensichtlich, daß auf diese Weise nicht alle beliebigen Muster gespeichert werden können, sondern die Darstellungsmächtigkeit eines Hopfieldnetzes genau wie beim einfachen Perzeptron beschränkt ist. Läßt man $w_{ii} > 0$ zu, kann man natürlich durch Selbstverstärkung die Stabilität jedes Musters erreichen. Eine Alternative stellt es dar, wenn man hidden Neuronen zuläßt, wie wir später noch sehen werden.

6.3 Hopfieldnetze als Optimierer

Hopfieldnetze haben die schöne Eigenschaft, die Energiefunktion zu minimieren. Dieses ausnutzend kann man natürlich zu einem gegebenen geeigneten Polynom ein Hopfieldnetz konstruieren, daß dieses Polynom als Energiefunktion besitzt, und Minima des Polynoms per Relaxation des Hopfieldnetzes gewinnen. Tatsächlich kann man auch NP-vollständige Probleme so angehen, wie Hopfield es für das TSP vorschlug. Unter anderem diese Tatsache führte zu einem wiedererweckten Interesse im Neurobereich – obwohl die Ergebnisse, wie nicht anders zu erwarten, insgesamt eher mäßig sind.

Wir betrachten jetzt ein Hopfieldnetz, das mit Gedächtnis schaltet, so daß eine bijektive Beziehung zwischen den Energieminima und den stabilen Zuständen besteht. Zunächst formulieren wir die Energiefunktion nocheinmal mit explizitem Bias, wir nehmen $w_{ii} = 0$ an:

$$-\frac{1}{2}(\mathbf{o}, 1)^t(\mathbf{w}, -\boldsymbol{\theta})(\mathbf{o}, 1) = -\sum_{i<j} w_{ij} o_i o_j + \sum_i \theta_i o_i \,,$$

wobei $(\mathbf{w}, -\boldsymbol{\theta})$ die Matrix mit letzter Spalte und Zeile $-\boldsymbol{\theta}$ und Diagonalelement 0 ist. Soll ein beliebiges quadratisches Polynom

$$\sum a_{ij} X_i X_j + \sum_i b_i X_i + c$$

mit Elementen X_i aus $\{0, 1\}$ minimiert werden, so kann man dieses in der Form

$$-\sum_{i<j}(-a_{ij} - a_{ji}) X_i X_j + \sum_i (b_i + a_{ii}) X_i$$

schreiben, da $X_i^2 = X_i$ gilt und Konstanten zur Minimierung nichts beitragen. D.h. ein Hopfieldnetz mit Gewichten

$$w_{ij} = -a_{ij} - a_{ji}, \quad \theta_i = b_i + a_{ii}$$

minimiert obiges Polynom.

Als erstes Beispiel betrachten wir das *Acht-Türme-Probem*: Auf einem Schachbrett sollen acht Türme so angeordnet werden, daß sie sich nicht gegenseitig bedrohen. Dazu definiert man Neuronen

$$X_{ij} = \begin{cases} 1 & \text{auf Feld } (i,j) \text{ steht ein Turm} \\ 0 & \text{sonst} \end{cases}$$

Es muß dabei in jeder Zeile und Spalte genau eine Turm stehen. Das heißt:

$$\sum_i X_{ij} = 1, \quad \sum_j X_{ij} = 1.$$

Dieses ist für die Minima der Funktion

$$\sum_j \left(\sum_i X_{ij} - 1\right)^2 + \sum_i \left(\sum_j X_{ij} - 1\right)^2$$

erfüllt. Ausmultiplizieren ergibt das Polynom

$$2\sum_{i,j} X_{ij}^2 - 4\sum_{i,j} X_{ij} + 32 + 2\sum_i \sum_{j<k} X_{ij} X_{ik} + 2\sum_j \sum_{i<k} X_{ij} X_{kj} \,.$$

Konstante Terme weglassend und X_{ij} mit X_{ij}^2 identifizierend erhält man

$$2\sum_i \sum_{j<k} X_{ij} X_{ik} + 2\sum_j \sum_{i<k} X_{ij} X_{kj} - 2\sum_{i,j} X_{ij} \,.$$

Ein zugehöriges Hopfieldnetz hat also Biases -2 und laterale Verbindungen mit Wert -2 zwischen je allen Neuronen einer Zeile bzw. je allen Neuronen einer Spalte. Leider besitzt dieses Netz noch unerwünschte lokale Minima, es können zwei Neuronen in einer Zeile bzw. Spalte den Wert 1 besitzen. Abhilfe schafft, wenn wir den Bias auf -1 statt -2 setzen. Das würde einer Änderung der zu minimierenden Funktion auf

$$\sum_j \left(\sum_i X_{ij} - 1\right)^2 + \sum_i \left(\sum_j X_{ij} - 1\right)^2 + \sum_{i,j} X_{ij}^2$$

entsprechen. Die absoluten Minima dieser Funktion sind dieselben, denn sind weniger als 8 Neuronen 1, dann erhält man vom ursprünglichen Polynom für jedes Neuron, das nicht 1 ist, den Fehler 2, da ja mindestens eine Spalte und Zeile unbesetzt sind. Absolute Minima haben also den Wert 8, der nur durch den zusätzlichen Term belegt wird. Der zusätzliche Term sorgt aber für eine Streckung der ursprünglichen Fehlerfunktion, so daß unerwünschte lokale Minima verschwinden. Man sieht, daß die Wahl einer guten Fehlerfunktion durchaus nicht offensichtlich ist.

Eine andere Anwendung ist das *TSP*: Städte $1, \ldots, n$ seien gegeben mit nichtnegativen Distanzen d_{ij} von der Stadt i zur Stadt j. Es werden die Variablen X_{ij} für $i, j \in \{1, \ldots, n\}$ definiert mit der Bedeutung

$$X_{ij} = 1 \iff \text{Stadt } i \text{ ist die } j\text{te Stadt der Rundreise.}$$

Es gilt, die absolute Distanz zu minimieren, d.h. der Term

$$D=\sum_{j=1}^{n-1}\sum_{kl} d_{kl}X_{kj}X_{lj+1}+\sum_{kl} d_{kl}X_{kn}X_{l1}$$

ist zu minimieren. Gleichzeitig muß dafür gesorgt werden, daß die Belegung tatsächlich eine Rundreise ist, d.h. jede Stadt soll genau einmal besucht sein:

$$\sum_i X_{ki}=1,$$

und an jeder Stelle kann nur eine Stadt stehen:

$$\sum_k X_{ki}=1\,.$$

Absolute Minima der Funktion

$$D+\lambda\left(\sum_k\left(\sum_i X_{ki}-1\right)^2+\sum_i\left(\sum_k X_{ki}-1\right)^2\right)$$

mit $\lambda >$ Länge einer optimalen Lösung des TSP entsprechen eindeutig optimalen Lösungen des TSP, da für diese aufgrund der Wahl von λ die beiden Nebenbedingungen erfüllt sind. Ausrechnen der Produkte führt zu einem Polynom zweiten Grades. Dieses kann man wie oben angegeben mit der Energiefunktion eines Hopfieldnetzes abgleichen. Man erhält ein Netz, in der benachbarte Spalten durch die negativen Abstände d_{ij} verbunden sind, innerhalb einer Zeile bzw. Spalte rufen die mit λ gewichteten Bedingungen inhibitorische Verbindungen hervor. Die Wahl der Parameter λ ist in der Praxis diffizil, da zu große Werte tendentiell zwar korrekte, aber nicht sehr gute Lösungen bewirken; zu kleine Werte hingegen führen zu ungültigen, aber meistens kurzen Rundtouren. Da die Nebenbedingung das n Türme Problem darstellt, sollten wir auch hier wieder den Bias leicht erhöhen, um lokale Minima zu vermeiden. Hopfield und Tank berichten bei einem 10 Städte Problem von etwa in jedem vierten Durchlauf optimalen Lösungen; dieses skaliert allerdings nicht auf größere Probleme. Zudem haben Hopfield und Tank nicht diskret schaltende Netze, sondern eine kontinuierlich schaltende Version benutzt.

Andere Optimierungsprobleme können auf Polynome höheren Grades führen. Kann man auch hier ein Hopfieldnetz zur Minimierung einsetzen? Dazu muß das Konzept zunächst etwas erweitert werden.

Definition 6.5 *Ein* **Hopfieldnetz mit hidden Neuronen** *ist ein Netz* $(N,\to,\mathbf{w},\boldsymbol{\theta},\mathbf{f},I,O)$*, so daß* $(N,\to,\mathbf{w},\boldsymbol{\theta},\mathbf{f},N,N)$ *ein Hopfieldnetz darstellt.*

Das heißt nichts anderes, daß Ein- und Ausgabeneuronen nicht notwendig mit allen Neuronen übereinstimmen. Insbesondere kann es, was wir gleich benutzen werden, nicht nach außen sichtbare Neuronen geben. So einem Netz kann man natürlich auch eine Energiefunktion zuordnen, die neben den sichtbaren Variablen auch die den hidden Neuronen entsprechenden Variablen benutzt. Es wir sich herausstellen, daß man in gewisser Weise Terme höherer Ordnung durch zusätzliche nicht sichtbare Variablen ersetzen kann. Als Konsequenz kann man beliebige Polynome mit einem Hopfieldnetz mit hidden Neuronen minimieren.

Es sei also ein Polynom $P(\mathbf{x})$ gegeben, in dem o.B.d.A. keine Terme x_i^2 vorkommen, da man diese ja einfach durch x_i ersetzen kann. Wir konstruieren ein Polynom $\tilde{P}(\mathbf{x},t)$ mit einer neuen Variablen t, so daß

$$P(\mathbf{x})=\min_t P(\mathbf{x},t)$$

gilt. Ist k der höchste Grad in P, so hat $\tilde{P}$ einen Term mit Grad k weniger als P. Sei $w\cdot\prod_{i=i_1}^{i_k} x_i$ ein Term höchsten Grades.

- Fall $w<0$: Ersetze den Term durch
$$\sum_{i=i_1}^{i_k} 2wx_it-(2k-1)wt\,.$$
- Fall $w>0$: Ersetze den Term durch
$$w\prod_{i=i_1}^{i_{k-1}} x_i-\sum_{i=i_1}^{i_{k-1}} 2wx_it+2wx_{i_k}t+(2k-3)wt\,.$$

Eine etwas längere Fallunterscheidung zeigt obige Eigenschaft. Jetzt kann man sukzessive die Terme höheren Gerades durch neue Variablen ersetzen, ohne die Dynamik der sichtbaren Neuronen und die Lage der globalen Minima zu ändern, und erhält ein die globalen Minima realisierendes Hopfieldnetz mit verborgenen Neuronen entsprechend den neu eingefürten Variablen t.

Eine schöne Konsequenz ist, daß das SAT Problem ebenfalls mit Hopfieldnetzen bearbeitet werden kann. Allgemeiner kann man für jede Boolesche Formel φ ein Hopfielnetz mit verborgenen Neuronen finden, so daß die globalen Energieminima der Energiefunktion beschränkt auf die sichtbaren Variablen genau den erfüllenden Belegungen der Formel entsprechen. Diese Aussage ist auch historisch interessant, da sie zeigt, daß neuronale Netze zumindest begrenzt mit symbolischen Daten, Booleschen Formeln, umgehen können.

Ein Polynom

$$p_\varphi(\mathbf{x})=\begin{cases}0 & \mathbf{x}\text{ erfüllt }\varphi\\ 1 & \text{sonst}\end{cases}$$

ist schnell indukiv über den Aufbau von φ konstruiert:

- $p_x=1-x$
- $p_{\neg\varphi}=1-p_\varphi$
- $p_{\varphi\vee\psi}=p_\varphi\cdot p_\psi$
- $p_{\varphi\wedge\psi}=1-(1-p_\varphi)(1-p_\psi)$
- $p_{\varphi\to\psi}=(1-p_\varphi)p_\psi$
- $p_{\varphi\leftrightarrow\psi}=(1-(1-p_\varphi)(1-p_\psi))(1-p_\varphi p_\psi)$

Etwa die Formel $x_1\vee x_2\vee x_3$ benötigt auch tatsächlich hidden Variablen zu ihrer Realisierung.

Die praktische Relevanz dieser Ansätze ist notwendig beschränkt – schließlich handelt es sich um NP-vollständige Probleme, so daß man auch von neuronalen Netzen keine effizienten Lösungsalgorithmen erwarten kann. Allerdings ist die Verbindung von Netzen und Formeln ein philosophisch und historisch höchst interessantes Thema, da es in den Streit zwischen symbolischer KI und Konnektionismus eingreift. Eine der Hauptkritikpunkte der symbolischen KI an Netzen ist, daß Verarbeiten von Formeln prinzipiell nicht adäquat möglich ist. Das sogenannte **Binding Problem**, welches grob die Frage stellt, wie in einer konnektionistischen Methode die (evtl. zeitlich veränderliche) Verknüpfung zweier verteilt dargestellter Entitäten in einem einzigen Objekt dargestellt werden kann. Konkret etwa: Wie kann man Variablen mit Termen, die diese ausfüllen, in Beziehung bringen, sofern beides verteilt repräsentiert ist.

Eine andere Konsequenz ist, daß Hopfieldnetze mit hidden Neuronen feedforward Netze mit der Perzeptronaktivierungsfunktion in folgendem Sinne simulieren können: Berechnet ein feedforward Netz die Funktion $x \mapsto f(x)$, so gibt es ein Hopfieldnetz mit hidden Neuronen und globalen Minima $(x, f(x))$. Möchte man also zu einer Eingabe x den Wert $f(x)$ berechnen, so kann man das durch Eingabe der Aktivierungen x in einen Teil der sichtbaren Neuronen und Relaxation zu einem (globalen) Minimum erreichen.

6.4 Alternative Schaltdynamiken

Eine Alternative zum asynchronen Schalten ist, die Werte der Neuronen simultan, aber stetig entsprechend ihrer Aktivierung zu ändern. Dazu ersetzt man die diskrete Aktivierungsfunktion wie auch bei vorwärtsgerichteten Netzen durch eine sigmoide Aktivierungsfunktion. Die Neuronenaktivierung besteht also aus reellen Werte zwischen 0 und 1. Anstatt zufällig die Neuronen einzeln zu ändern oder sie simultan gemäß ihrer Aktivierung neu zu berechnen, ändern die Neuronen ihre Werte stetig; d.h. die Dynamik wird durch Differentialgleichungen beschrieben. Ausgehend von einer Startbelegung, etwa dem Eingabemuster, ändern dann die Neuronen ihre Aktivierung $\text{net}_i(t)$ zum Zeitpunkt t gemäß der Vorschrift:

$$\eta \cdot \frac{d\text{net}_i(t)}{dt} = -\text{net}_i(t) + \sum_j w_{ji} \cdot \text{sgd}(\text{net}_j(t))\,,$$

wobei η eine feste positive Zahl ist. Man schaltet also nicht mehr diskret zu den Zeitpunkten 1, 2, $\ldots$, sondern kontinuierlich für $t > 0$. Die Differentialgleichung gibt die Änderung zu jedem Zeitpunkt an. In der Praxis muß die Dynamik in der Regel trotzdem durch Diskretisierungen genähert werden, d.h. man erhält eine synchrone Schaltweise wie schon zuvor. Der Vorteil ist, daß die Diskretisierung im Prinzip beliebig fein vorgenommen werden kann und die einzelnen Schritte nachgebessert werden können. D.h. statt zum Zeitpunkt $0, 1, \ldots$ zu schalten, kann man etwa die Zeitpunkte $0, 0.1, 0.2, \ldots$ berechnen.

Obige Differentialgleichung hat eine eindeutige Lösung, da die beteiligten Funktionen gutartig sind (lokal Lipschitz-stetig). Die Tatsache, daß die Dynamik immer gegen einen stabilen Zustand konvergiert, währenddessen eine Energiefunktion minimiert wird, entspricht hier folgendem Umstand:

Satz 6.6 *Man findet eine Funktion $E(t)$, die monoton fallend und nach unten beschränkt ist. Sie hat die Ableitung exakt 0 dann und nur dann, wenn $d\text{net}_i(t)/dt = 0$ gilt. Es ist $dE(t)/dt \to 0 \iff d\text{net}_i(t)/dt \to 0 \forall i$.*

Beweis: Betrachte die Energiefunktion

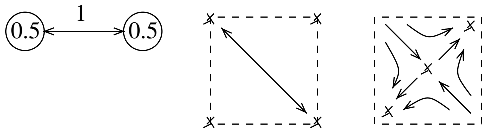
$$E(t) = -\frac{1}{2}\sum_{i,j} w_{ij} o_i(t) o_j(t) + \sum_i \int_0^{o_i(t)} \text{sgd}^{-1}(o)do\,.$$

E ist nach unten beschränkt, da $o_i(t) \in [0,1]$ gilt. Die Ableitung berechnet sich als

$$\begin{aligned}
\frac{dE(t)}{dt} &= -\frac{1}{2}\sum_{i,j} w_{ij}\frac{do_i(t)}{dt}\cdot o_j(t) - \frac{1}{2}\sum_{i,j} w_{ij} o_i(t)\cdot\frac{do_j(t)}{dt} + \sum_i \text{sgd}^{-1}(o_i(t))\cdot\frac{do_i(t)}{dt}\\
&= -\sum_i \underbrace{\frac{do_i(t)}{dt}}_{\text{sgd}'(\text{net}_i(t))d\text{net}_i(t)/dt} \underbrace{\left(\sum_j w_{ij}o_j(t) - \text{net}_i(t)\right)}_{\eta d\text{net}_i(t)/dt}\\
&= -\sum_i \eta\cdot\text{sgd}'(\text{net}_i(t))\cdot\left(\frac{d\text{net}_i(t)}{dt}\right)^2\\
&\leq 0,
\end{aligned}$$

wobei die Regel $d\int_{f(x)}^{g(x)} h(t)dt/dx = h(g(x))g'(x) - h(f(x))f'(x)$ benutzt wurde. Obiger Ausdruck ist wegen $\text{sgd}'(x) > 0$ nur dann exakt 0, wenn $d\text{net}_i(t)/dt = 0$ gilt. $\square$

Da E monoton fallend und beschränkt ist und die Steigung durch die Ableitung von net_i bestimmt wird, ergibt sich für $t \to \infty$ ein Zustand, so daß $d\text{net}_i(t)/dt \to 0$ gilt. D.h. das Netz strebt gegen einen stabilen Zustand, wo sich die Aktivierungen der Neuronen nur noch marginal ändern. Insbesondere sind Zyklen nicht mehr möglich: Sie sind ein Effekt der bei synchronem Schalten zu groben Diskretisierung der Dynamik, wobei gewisse Anfangszustände nicht gegen die aufgrund der Diskretisierung nicht mehr vorkommenden nicht stabilen Fixpunkte der Differentialgleichung streben können. Ein Beispiel für die jetzt wesentlich einsichtigere Dynamik ist das Netz:



Bei synchronem Schalten gibt es zwei stabile Zustände und zwei Zustände, die in einem Zweierzyklus abwechselnd geschaltet werden. Bei der zugehörigen Differentialgleichung (bzw. den zu den net_i gehörenden o_i) findet man in der Nähe der beiden ursprünglichen stabilen Punkte wieder zwei stabile Fixpunkte der Differentialgleichung, die jeweils Attraktoren darstellen. Auf der Grenze zwischen den beiden Attraktorbecken befinden sich auch die beiden Zustände, die sich bei synchronem Schalten zyklisch abwechseln. Jetzt streben sie auf dem Rand zwischen den beiden Attraktorbecken gegen einen instabilen Fixpunkt.

Man kann die Dynamik eines synchron schaltenden Hopfieldnetzes in folgendem Sinne durch eine Differentialgleichung ersetzen:

Satz 6.7 *Für ein Hopfieldnetz, dessen stabile Zustande für kein Neuron zu einer Aktivierung exakt 0 führen, und vorgegebenes $\epsilon > 0$ kann man einen positiven Faktor W finden, so daß für das durch eine Differentialgleichung modellierte Netz mit den Gewichten multipliziert mit W folgendes gilt: Für jeden stabilen Zustand o des Hopfieldnetzes findet man einen stabilen Attraktor des durch die Differentialgleichung beschriebenen Netzes, so daß die zugehörigen Ausgaben o' in keiner Komponente mehr als ϵ von o abweichen.*

Beweis: Sei für alle stabilen Zustände die Aktivierung der Neuronen betragsmäßig größer als δ. ϵ sei so klein, daß für um ϵ in den Komponenten von einem Fixpunkt abweichende Punkte die Aktivierungen betragsmäßig immer noch größer als δ sind. W sei so gewählt, daß

$$x < -\delta \Rightarrow \text{sgd}(Wx) < \epsilon/2, \quad x > \delta \Rightarrow \text{sgd}(Wx) > 1 - \epsilon/2$$

gilt. Insbesondere berechnet man für einen von einem stabilen Zustand o^s in jeder Komponente weniger als ϵ abweichenden Zustand o, daß der durch $\text{sgd}(W\sum w_{ij}o_j)$ gegebene Zustand um weniger als $\epsilon/2$ in jeder Komponenten vom stabilen Zustand abweicht, denn die Aktivierungen sind größer als δ, mit W multipliziert ergibt sich also maximal der Abstand $\epsilon/2$. Das bedeutet aber, daß für Koeffizienten i von o, im Intervall $]\epsilon/2, \epsilon[$

$$-o_i + \text{sgd}(W\sum w_{ji}o_j) < 0$$

und für Koeffizienten i von o im Intervall im Intervall $]1-\epsilon, 1-\epsilon/2[$

$$-o_i + \mathrm{sgd}(W \sum w_{ji} o_j) > 0$$

mit der Dreiecksungleichung. Das heißt aber, daß bei Start in den Intervallen $]\epsilon/2, \epsilon[$ bzw. $]1-\epsilon, 1-\epsilon/2[$ um den stabilen Zustand man in einer ϵ-Umgebung des stabilen Zustands bleibt. Für die zugehörigen Nettoinputs bedeutet das, daß sie über die zeitliche Entwicklung hin entsprechend dem stabilen Zustand $> \delta$ bzw. $< -\delta$ bleiben müssen. Da sie konvergieren, erhält man also einen stabilen Zustand mit maximal ϵ vom ursprünglichen stabilen Zustand entfernten Ausgaben o_i. □

Somit werden also Zyklen vermieden und die Konvergenz gegen einen stabilen Zustand ist deterministisch. Nichtsdestotrotz ist das Problem von lokalen Minima der Energiefunktion gegeben, die man eigentlich verhindern will – und hier nicht umgeht. Eine Alternative stellt eine stochastische Betriebsweise von Netzen dar. Die Zustände sind weiterhin $\{0,1\}$-wertig. Für jeden Schaltschritt wird die Altivierung jedes Neurons berechnet und anschließend synchron alle Neuronen oder auch lediglich ein Neuron mit der Wahrscheinlichkeit

$$\mathrm{sgd}(\mathrm{net}_i/T)$$

auf 1 geschaltet. T heißt dabei die **Temperatur**. Je kleiner diese ist, desto steiler ist die sich ergebende Kurve in Abhängigkeit von net_i und desto mehr approximiert die Kurve die Perzeptronaktivierung. Aber die Möglichkeit, sich aus lokalen Minima der Energiefunktion zu befreien, ist gegeben: Die Wahrscheinlichkeit, zwar lokal verschlechternde Schritte (Energie wird kurzfristig höher), aber global verbessernde Schritte (man erreicht langfristig ein besseres Optimum) durchzuführen, ist größer als Null. Häufig betreibt man das Netz dabei mit **Simulated Annealing**, d.h. die Temperatur ist zu Beginn sehr hoch, so daß der Zustandsraum stark exploriert wird, und wird allmählich abgekühlt, d.h. T vermindert, bis man bei kleinem T in einem (dann hoffentlich globalen) Optimum landet. Die stochastische Sichtweise ermöglicht eine weitere Trainingsmethode, wobei auch hidden Neuronen mittrainiert werden können. Der Ansatz geht auf Hopfield zurück und nennt sich nach der beteiligten stochastischen Verteilung die Boltzmannmaschine.

6.5 Die Boltzmannmaschine

Die **Boltzmannmaschine** ist ein stochastisch schaltendes Hopfieldnetz. Wir nehmen hier an, daß sie asynchron geschaltet wird. D.h. in jedem Schritt wird zufällig ein Neuron ausgwählt, das mit Wahrscheinlichkeit $\mathrm{sgd}(\mathrm{net}_i/T)$ den Wert 1 annimmt, mit Wahrscheinlichkeit $1-\mathrm{sgd}(\mathrm{net}_i/T)$ den Wert 0. Es sollen zunächst einige Notationen eingeführt werden:

Wie schon vorher sei

$$E(\mathbf{o}) = -\frac{1}{2}\sum w_{ij} o_i o_j + \sum \theta_i o_i$$

die Energie im Zustand $\mathbf{o}$. N sei die Anzahl der Neuronen. Jedes Neuron werde mit Wahrscheinlichkeit $1/N$ ausgewählt. Wenn man sich im Zustand $\mathbf{o}$ befindet und das Neuron i ausgewählt hat, sei

$$P(1|\mathbf{o}, i) = \left(1 + \mathrm{e}^{-\mathrm{net}_i/T}\right)^{-1}$$

die Wahrscheinlichkeit, daß das Neuron i auf 1 schaltet,

$$P(0|\mathbf{o}, i) = \left(1 + \mathrm{e}^{\mathrm{net}_i/T}\right)^{-1}$$

ist die Wahrscheinlichkeit, daß das Neuron auf 0 schaltet. Für Zustände $\mathbf{o}, \mathbf{o}'$ sei

$$P(\mathbf{o} \to \mathbf{o}')$$

die Wahrscheinlichkeit, in einem Schritt von $\mathbf{o}$ nach $\mathbf{o}'$ zu schalten, d.h. es ist

$$P(\mathbf{o} \to \mathbf{o}') = \begin{cases} 1/N \cdot \sum P(o_i|\mathbf{o}, i) & \mathbf{o} = \mathbf{o}' \\ 1/N \cdot P(o_i'|\mathbf{o}, i) & \mathbf{o} \text{ und } \mathbf{o}' \text{ unterscheiden sich nur in } i \\ 0 & \text{sonst} \end{cases}$$

Dieses definiert für jedes feste $\mathbf{o}$ eine Verteilung auf den Zuständen $\mathbf{o}'$. Die Wahrscheinlichkeit, in n Schritten von $\mathbf{o}$ nach $\mathbf{o}'$ zu kommen, sei mit

$$P(\mathbf{o} \overset{n}{\to} \mathbf{o}')$$

bezeichnet. Da man über alle möglichen Zwischenzustände $\mathbf{o}''$ gehen kann, gilt offensichtlich

$$P(\mathbf{o} \overset{n}{\to} \mathbf{o}') = \sum_{\mathbf{o}''} P(\mathbf{o} \overset{n_1}{\to} \mathbf{o}'') P(\mathbf{o}'' \overset{n_2}{\to} \mathbf{o}')$$

für alle $n_1 + n_2 = n$. Auch dieses definiert für jedes feste $\mathbf{o}$ eine Verteilung auf den Zuständen $\mathbf{o}'$. Die Situation, wenn man sehr lange schaltet, beschreibt folgender Satz:

Satz 6.8 *Für alle Zustände $\mathbf{o}$ existiert die Grenzverteilung p^* mit*

$$p^*(\mathbf{o}') = \lim_{n\to\infty} P(\mathbf{o} \overset{n}{\to} \mathbf{o}')\,.$$

Sie ist unabhängig von $\mathbf{o}$. Die Verteilung erfüllt die Fixpunktgleichung

$$p^*(\mathbf{o}') = \sum_{\mathbf{o}} p^*(\mathbf{o}) P(\mathbf{o} \to \mathbf{o}').$$

Die Verteilung ist die sog. **Boltzmannverteilung**

$$p^*(\mathbf{o}') = \frac{\mathrm{e}^{-E(\mathbf{o}')/T}}{\sum_{\mathbf{o}} \mathrm{e}^{-E(\mathbf{o})/T}}\,.$$

Beweis: Es sei $M_o^n = \max_{\mathbf{o}'} P(\mathbf{o}' \overset{n}{\to} \mathbf{o})$. Das ist durch 0 nach unten beschränkt und monoton fallend wegen

$$\begin{aligned} M_o^{n+1} &= \max_{\mathbf{o}'} P(\mathbf{o}' \overset{n+1}{\to} \mathbf{o}) \\ &= \max_{\mathbf{o}'} \sum_{\mathbf{o}''} P(\mathbf{o}' \to \mathbf{o}'') P(\mathbf{o}'' \overset{n}{\to} \mathbf{o}) \\ &\leq \max_{\mathbf{o}'} \sum_{\mathbf{o}''} P(\mathbf{o}' \to \mathbf{o}'') \max_{\mathbf{o}''} P(\mathbf{o}'' \overset{n}{\to} \mathbf{o}) \\ &= M_o^n \cdot \max_{\mathbf{o}'} \sum_{\mathbf{o}''} P(\mathbf{o}' \to \mathbf{o}'') = M_o^n \cdot 1 \end{aligned}$$

M_o^n konvergiert also für $n \to \infty$. Analog sieht man, daß $m_o^n = \min_{\mathbf{o}'} P(\mathbf{o}' \overset{n}{\to} \mathbf{o})$ monoton wachsend ist. Da es durch 1 nach oben beschränkt ist, folgt auch hier die Konveregnz.

Für alle $\mathbf{o}$ und $\mathbf{o}'$ ist $P(\mathbf{o} \overset{N}{\to} \mathbf{o}') \geq \delta > 0$ für ein δ, da man ja nach N Änderungen jeden Zustand in jeden anderen überführen kann. Dann gilt für geeignete Indizes M und m

$$M_o^{N(n+1)} - m_o^{N(n+1)} = P(\mathbf{o}_M \overset{N(n+1)}{\to} \mathbf{o}) - P(\mathbf{o}_m \overset{N(n+1)}{\to} \mathbf{o})$$

$$
\begin{aligned}
&= \sum_{\mathbf{o}'} \underbrace{\left(P(\mathbf{o}_M \overset{N}{\to} \mathbf{o}') - P(\mathbf{o}_m \overset{N}{\to} \mathbf{o}')\right)}_{(*)} \cdot P(\mathbf{o}' \overset{Nn}{\to} \mathbf{o}) \\
&\le \sum_{\mathbf{o}',(*)\ge 0} (*) \cdot M_o^{Nn} + \sum_{\mathbf{o}',(*)<0} (*) \cdot m_o^{Nn} \\
&= \sum_{\mathbf{o}',(*)\ge 0} (*) \cdot \left(M_o^{Nn} - m_o^{Nn}\right) \\
&\le (1-\delta)\left(M_o^{Nn} - m_o^{Nn}\right)
\end{aligned}
$$

Die vorletzte Gleichung gilt, da $\sum_{\mathbf{o}',(*)<0}(*)+\sum_{\mathbf{o}',(*)\ge 0}(*) = 1-1 = 0$ ist. Die letzte Ungleichung folgert man wegen $P(\mathbf{o}_M \overset{N}{\to} \mathbf{o}') \le 1$ und $P(\mathbf{o}_m \overset{N}{\to} \mathbf{o}') \ge \delta$. Es folgt also

$$M_o^{Nn} - m_o^{Nn} \le (1-\delta)\left(M_o^{N(n-1)} - m_o^{N(n-1)}\right) \le \ldots \le (1-\delta)^n \left(M_o^0 - m_o^o\right) = (1-\delta)^n \to 0\,.$$

M_o^n und m_o^n streben also gegen denselben Limes. Es sei $\mathbf{o}'$ ein beliebiger Startvektor. Dann gilt

$$m_o^n \le P(\mathbf{o}' \overset{n}{\to} \mathbf{o}) \le M_o^n\,,$$

also konvergiert auch $P(\mathbf{o}' \overset{n}{\to} \mathbf{o})$ unabhängig vom Startvektor gegen dieselbe Grenze $p^*(\mathbf{o})$. p^* ist eine Verteilung, da mit $P(\mathbf{o}' \overset{n}{\to} \mathbf{o}) \ge 0$ und $\sum_{\mathbf{o}} P(\mathbf{o}' \overset{n}{\to} \mathbf{o}) = 1$ dasselbe auch für den Limes gilt. Ebenso erhält man durch Grenzübergang in der Gleichung

$$P(\mathbf{o}'' \overset{n+1}{\to} \mathbf{o}') = \sum_{\mathbf{o}} P(\mathbf{o}'' \overset{n}{\to} \mathbf{o}) P(\mathbf{o} \to \mathbf{o}')$$

die Fixpunktgleichung für die Grenzverteilung. Eine Verteilung, die diese Fixpunktgleichung erfüllt, ist eindeutig bestimmt, denn sei p_0 eine andere Verteilung, die ebenfalls diese Fixpunktgleichung erfüllt. Für eine beliebige Verteilung p sei p^i die Verteilung nach i Schritten. Dann ist $p^n(\mathbf{o}) = \sum_{\mathbf{o}'} p(\mathbf{o}')P(\mathbf{o}' \overset{n}{\to} \mathbf{o})$ zwischen M_o^n und m_o^n angesiedelt. Egal mit welcher Verteilung man startet, man erhält also die Grenzverteilung nach beliebig langem Schalten. Speziell für p_0 berechnet man

$$p_0^n(\mathbf{o}') = \sum p_0^{n-1}(\mathbf{o}) P(\mathbf{o} \to \mathbf{o}') = \ldots = p_0(\mathbf{o}')\,,$$

das heißt die Grenzverteilung stimmt mit p_0 überein.

Es reicht also zu zeigen, daß die Boltzmannverteilung die Fixpunktgleichung erfüllt. Offensichtlich kann man den Normierungsterm $\sum_{\mathbf{o}} \mathrm{e}^{-E(\mathbf{o})/T}$ bei der Rechnung weglassen. $\mathbf{o}[i]$ bezeichne den nur in der Stelle i von $\mathbf{o}$ verschiedenen Zustand. Es gilt

$$
\begin{aligned}
& \textstyle\sum_{\mathbf{o}'} \mathrm{e}^{-E(\mathbf{o}')/T} P(\mathbf{o}' \to \mathbf{o}) \\
= \;& \mathrm{e}^{-E(\mathbf{o})/T} P(\mathbf{o} \to \mathbf{o}) + \textstyle\sum_i \mathrm{e}^{-E(\mathbf{o}[i])/T} P(\mathbf{o}[i] \to \mathbf{o}) \\
= \;& \mathrm{e}^{-E(\mathbf{o})/T} \left(\tfrac{1}{N} \sum P(o_i|\mathbf{o},i) + \sum_i \tfrac{1}{N} \mathrm{e}^{E(\mathbf{o})-E(\mathbf{o}[i]))/T} P(o_i|\mathbf{o}[i],i)\right) \\
= \;& \mathrm{e}^{-E(\mathbf{o})/T} \tfrac{1}{N} \left(\sum P(o_i|\mathbf{o},i) \left(1 + \mathrm{e}^{-\mathrm{net}_i(o_i - o[i]_i)/T}\right)\right) \\
= \;& \mathrm{e}^{-E(\mathbf{o})/T} \tfrac{1}{N} \textstyle\sum 1 = \mathrm{e}^{-E(\mathbf{o})/T}\,,
\end{aligned}
$$

MIT $p(o_i|\mathbf{o}[i],i) = P(o_i|\mathbf{o},i)$, da $w_{ii} = 0$ gilt, und $E(\mathbf{o}) - E(\mathbf{o}[i]) = -\mathrm{net}_i(o_i - o[i]_i)$, wie wir früher schon nachgerechnet haben. □

Die Boltzmannmaschine strebt also, egal in welchem Zustand sie gestartet wurde, gegen eine

Grenzverteilung, die sich ausrechnen läßt. Diejenigen Zustände haben eine hohe Wahrscheinlichkeit, die Energieminima sind. Entspräche die Grenzverteilung der Gleichverteilung auf einigen Zuständen, so würden diese Zustände den gespeicherten Zuständen entsprechen, sie werden im Limes erreicht. Durch langes Schalten kann man die absoluten Energieminima anhand ihrer Häufigkeit eindeutig rekonstruieren. Möchte man die Boltzmannmaschine als Assoziativspeicher verwenden, so besteht auch hier die Hoffnung, daß derjenige Zustand der Grenzverteilung als erstes erreicht wird, der dem Startzustand am ähnlichsten ist. Simulated Annealing kann den Effekt unterstützen, daß ausgehend vom Anfangszustand der ähnlichste gespeicherte Zustand unter Vermeidung lokaler Minima erreicht wird.

Die Grenzverteilung der Boltzmannmaschine wurde nur für ein festes T ausgerechnet. Bei Simulated Annealing strebt T gegen Null. Der Effekt ist dabei folgender:

Satz 6.9 *Die durch die Wahrscheinlichkeit der Zustände induzierte Anordnung ist unabhängig von T. Für $T \to 0$ strebt die Grenzverteilung gegen die Gleichverteilung auf den Zuständen mit minimaler Energie.*

Beweis: Die Wahrscheinlichkeit eines Zustandes ist durch den Ausdruck

$$p^*(\mathbf{o}') = \frac{\mathrm{e}^{-E(\mathbf{o}')/T}}{\sum \mathrm{e}^{-E(\mathbf{o})/T}}$$

gegeben. Die Ungleichung $p^*(\mathbf{o}') < p^*(\mathbf{o})$ ist unabhängig von T, sondern hängt nur von der Energie der beiden Zustände ab.

Für $T \to 0$ berechnet man

$$\lim_{T\to 0} \frac{\mathrm{e}^{-E(\mathbf{o}')/T}}{\sum \mathrm{e}^{-E(\mathbf{o})/T}} = \lim_{T\to 0} \frac{1}{\sum \mathrm{e}^{(E(\mathbf{o}') - E(\mathbf{o}))/T}}\,.$$

Dieses ist 0, falls ein $\mathbf{o}$ mit geringerer Energie als $\mathbf{o}'$ existiert, da dann der Nenner über alle Grenzen wächst. Falls $\mathbf{o}$ eines von k Energieminima ist, findet man im Nenner k Summanden 1 und weitere Summanden, die gegen Null gehen, es ergibt sich also der Term $1/k$. Dieses entspricht einer Gleichverteilung auf den Energieminima. □

Sollen also Muster gespeichert werden, indem die Grenzverteilung in etwa einer Gleichverteilung auf den Mustern entspricht, dann kann man gut Simulated Annealing betreiben: Alle übrigen lokalen Minima verschwinden für kleines T. Möchte man allerdings die Muster mit gewissen unterschiedlichen Häufigkeiten speichern, darf T nicht zu klein werden.

Zum Training versucht man jetzt, die Gewichte so einzustellen, daß die Grenzverteilung einer gewünschten Verteilung – der Gleichverteilung auf den zu speichernden Mustern – entspricht. Dazu wird als Fehlermaß die sog. **Kreuzentropie** der beiden Verteilungen p^*, der Grenzverteilung, und p^+ der gewünschten Verteilung, gebildet:

$$E(p^*, p^+) = -\sum_{\mathbf{o}} p^+(\mathbf{o}) \cdot \ln \frac{p^*(\mathbf{o})}{p^+(\mathbf{o})}$$

und minimiert. Der Fehler ist positiv und genau dann Null, wenn die beiden Verteilungen übereinstimmen, denn:

Lemma 6.10 *Für Zahlen p_i, $q_i > 0$ mit $\sum p_i = 1 = \sum q_i$ ist $\sum p_i \ln(q_i/p_i) \ge 0$ und gleich Null genau dann, wenn alle p_i und q_i übereinstimmen.*

Beweis: Wie man am Graphen leicht sieht, gilt $x + 1 \leq e^x$ für alle x und Gleichheit nur für $x = 0$. Es folgt

$$\sum p_i \ln \frac{q_i}{p_i} \geq \sum p_i \left(\frac{q_i}{p_i} - 1\right) = \sum -q_i + p_i = 0$$

mit Gleichheit genau dann, wenn $q_i/p_i = 1$ für alle i gilt. □

Zum Training wird also Gradientenabstieg auf dem so definierten Fehler vorgenommen. Da offensichtlich eine Verteilung p^+ mit $p^+(\mathbf{o}) = 0$ für ein $\mathbf{o}$ nicht durch eine Boltzmannverteilung mit endlicher Temperatur T dargestellt werden kann, sollte man eine gegebene Zielverteilung p^+ für das Training so abändern, daß die Wahrscheinlichkeit für alle Zustände evtl. klein, aber positiv ist. Dieses verhindert ein Anwachsen der Gewichte über alle Schranken und kann durch Simulated Annealing ausgeglichen werden. Das Training besteht also aus der Vorschrift

$$\triangle \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(p^*, p^+)$$

mit einer Schrittweite $\eta > 0$. Dabei werden die Biases durch On-Neuronen simuliert. Es soll jetzt die Ableitung von E bestimmt werden. Der in der Verteilung auftretende Normierungsterm sei $n = \sum_{\mathbf{o}'} e^{-E(\mathbf{o}')/T}$.

$$\begin{aligned} &\frac{\partial - \sum_{\mathbf{o}} p^+(\mathbf{o}) \ln \frac{p^*(\mathbf{o})}{p^+(\mathbf{o})}}{\partial w_{ij}} \\ = & -\sum_{\mathbf{o}} p^+(\mathbf{o}) \cdot \frac{1}{p^*(\mathbf{o})} \cdot \frac{\partial \left(e^{-E(\mathbf{o})/T}/n\right)}{\partial w_{ij}} \\ = & -\sum_{\mathbf{o}} \frac{p^+(\mathbf{o})}{p^*(\mathbf{o})} \left(e^{-E(\mathbf{o})/T} \frac{-\partial E(\mathbf{o})/T}{\partial w_{ij}} \cdot n - e^{-E(\mathbf{o})/T} \sum_{\mathbf{o}'} e^{-E(\mathbf{o}')/T} \frac{-\partial E(\mathbf{o}')/T}{\partial w_{ij}} \right) /n^2 \\ = & -\sum_{\mathbf{o}} \frac{p^+(\mathbf{o})}{p^*(\mathbf{o})} \left(\frac{e^{-E(\mathbf{o})/T}}{n} \cdot \frac{1}{2T} \cdot o_i o_j - \frac{e^{-E(\mathbf{o})/T}}{n} \sum_{\mathbf{o}'} \frac{e^{-E(\mathbf{o}')/T}}{n} \cdot \frac{1}{2T} \cdot o'_i o'_j \right) \\ = & -\sum_{\mathbf{o}} p^+(\mathbf{o}) \frac{1}{2T} o_i o_j + \sum_{\mathbf{o}} p^+(\mathbf{o}) \frac{1}{2T} \sum_{\mathbf{o}'} p^*(\mathbf{o}') o'_i o'_j \\ = & \frac{1}{2T} \left(\sum_{\mathbf{o}} p^*(\mathbf{o}) o_i o_j - \sum_{\mathbf{o}} p^+(\mathbf{o}) o_i o_j \right) \end{aligned}$$

D.h. w_{ij} wird in jedem Schritt so geändert, daß die Verbindung von i nach j gemäß der Soll-Verteilung gemittelt verstärkt wird, wobei aber die Ist-Verteilung berücksichtigt wird. Dabei muß man die Größe $\sum_{\mathbf{o}} p^*(\mathbf{o}) o_i o_j$ entweder anhand der Gewichte ausrechnen oder statistisch schätzen, indem man das Netz zur Grenzverteilung relaxieren läßt und über die dann auftretenden Zustände mittelt. Ist die Soll- und Ist-Verteilung je eine Gleichverteilung, dann werden das Gewicht von i nach j lediglich anhand der Anzahl des gemeinsamen Auftretens der Pixel i und j in den Mustern bestimmt. Dieses ist also wieder ein Hebbsches Lernen, wobei der aktuelle Zustand durch einen Anti-HebbTerm mit berücksichtigt wird.

Man kann diesen Formalismus auf Netze mit verborgenen Neuronen erweitern. Dazu seien die Zustände auf dem sichtbaren Anteil mit $\mathbf{v}$ notiert, sie sollen auf die Verteilung $p^*(\mathbf{v})$ trainiert werden. Die verborgenen Neuronen, die lediglich zur Stabilisierung benutzt werden und dessen Aktivierung uninteressant ist, seien mit $\mathbf{h}$ notiert. Die Grenzverteilung $p^*(\mathbf{vh})$ induziert auf dem uns interessierenden Teil eine Randverteilung

$$p^*(\mathbf{v}) = \sum_{\mathbf{h}} p^*(\mathbf{vh}).$$

Die Gewichte sollen so gewählt werden, daß diese Randverteilung mit der gegebenen Verteilung $p^+(\mathbf{v})$ übereinstimmt. Es sei

$$p^*(\mathbf{h}|\mathbf{v}) = \frac{p^*(\mathbf{vh})}{p^*(\mathbf{v})}$$

die Verteilung der hidden Neuronen im Grenzfall, wenn wir $\mathbf{v}$ bereits kennen. n sei der in der Verteilung vorkommende Normierungsterm $\sum_{\mathbf{v}'\mathbf{h}'} e^{-E(\mathbf{v}'\mathbf{h}')/T}$. Dann kann man für die uns interessierende Ableitung des Fehlers $E(p^*(\mathbf{v}), p^+(\mathbf{v}))$ nachrechnen:

$$\begin{aligned} &\frac{\partial - \sum_{\mathbf{v}} p^+(\mathbf{v}) \ln \frac{p^*(\mathbf{v})}{p^+(\mathbf{v})}}{\partial w_{ij}} \\ = & -\sum_{\mathbf{v}} \frac{p^+(\mathbf{v})}{p^*(\mathbf{v})} \sum_{\mathbf{h}} \frac{\partial p^*(\mathbf{vh})}{\partial w_{ij}} \\ = & -\sum_{\mathbf{vh}} \frac{p^+(\mathbf{v})}{p^*(\mathbf{v})} \Bigg(e^{-E(\mathbf{vh})/T} \cdot \frac{1}{2T} (vh)_i (vh)_j \cdot n \\ & \qquad - e^{-E(\mathbf{vh})/T} \sum_{\mathbf{v}'\mathbf{h}'} e^{-E(\mathbf{v}'\mathbf{h}')/T} \cdot \frac{1}{2T} \cdot (v'h')_i (v'h')_j \Bigg) /n^2 \\ = & -\frac{1}{2T} \left(\sum_{\mathbf{vh}} \frac{p^+(\mathbf{v})}{p^*(\mathbf{v})} p^*(\mathbf{vh}) (vh)_i (vh)_j - \sum_{\mathbf{vh}} \frac{p^+(\mathbf{v})}{p^*(\mathbf{v})} p^*(\mathbf{vh}) \sum_{\mathbf{v}'\mathbf{h}'} p^*(\mathbf{v}'\mathbf{h}') (v'h')_i (v'h')_j \right) \\ = & -\frac{1}{2T} \left(\sum_{\mathbf{v}} p^+(\mathbf{v}) \sum_{\mathbf{h}} p^*(\mathbf{h}|\mathbf{v}) (vh)_i (vh)_j - \sum_{\mathbf{vh}} p^*(\mathbf{vh}) (vh)_i (vh)_j \right). \end{aligned}$$

Die Gewichte werden also wieder gemäß der gewünschten Verteilung durch Hebbsches Lernen verstärkt, wobei bei den freien Gewichten zu hidden Neuronen statt der Soll-Verteilung die tatsächliche Verteilung eingesetzt wird. Ein Anti-Hebb-Term gemäß der aktuellen Verteilung kommt hinzu. Die Größen kann man entweder aufgrund der bekannten Form der Grenzverteilung anhand der Gewichte berechnen oder statistisch schätzen. Um die Größe $p^*(\mathbf{h}|\mathbf{v})$ zu bestimmen, hält man dazu die Aktivierung der sichtbaren Neuronen fest und läßt den Rest zur Grenzverteilung relaxieren. Die Verteilung entspricht dann der zu schätzenden, denn:

Lemma 6.11 *Die Boltzmannmaschine mit fixem* $\mathbf{v}$ *relaxiert zur Grenzverteilung* $p(\mathbf{h}|\mathbf{v})$.

Beweis: Falls man die Neuronen $\mathbf{v}$ festhält, ergibt sich eine Boltzmannmaschine für die übrigen Neuronen mit geändertem Bias, dem Bias $\theta_i(\mathbf{v}) = -\sum_k w_{ik} v_k$ für das Neuron h_i. Diese relaxiert also gegen eine Grenzverteilung. Sei $E_{\mathbf{v}}(\mathbf{h})$ die Energie dieser Maschine im Zustand $\mathbf{h}$. $E(\mathbf{v})$ sei die Energie, wenn man nur die Neuronen $\mathbf{v}$ berücksichtigt, $E(\mathbf{vh})$ sei die Energie der Boltzmannmaschine für Summation über alle Neuronen. Es gilt:

$$\begin{aligned} E_{\mathbf{v}}(\mathbf{h}) &= -\sum w_{ij} h_i h_j/2 + \sum h_i \theta_i(\mathbf{v}) \\ &= -\sum w_{ij} (vh)_i (vh)_j/2 + \sum w_{ij} v_i v_j/2 = E(\mathbf{vh}) - E(\mathbf{v}). \end{aligned}$$

Folglich ist mit $n = \sum_{\mathbf{v}''\mathbf{h}''} e^{E(\mathbf{v}''\mathbf{h}'')/T}$:

$$p^*_{\mathbf{v}}(\mathbf{h}) = \frac{e^{-E_{\mathbf{v}}(\mathbf{h})/T}}{\sum_{\mathbf{h}'} e^{-E_{\mathbf{v}}(\mathbf{h}')/T}} = \frac{e^{-E(\mathbf{vh})/T}/n}{\sum_{\mathbf{h}'} e^{-E(\mathbf{vh}')/T}/n} = \frac{p^*(\mathbf{vh})}{\sum_{\mathbf{h}'} p^*(\mathbf{vh}')} = \frac{p^*(\mathbf{vh})}{p^*(\mathbf{v})} = p^*(\mathbf{h}|\mathbf{v}).$$

□

Die Boltzmannmaschine wird daher mithilfe folgenden Algorithmus trainiert:

- Die Gewichte werden zufällig initialisiert. Sie werden vermöge Gradientenabstieg geändert.
- Der Gradient besteht aus einem Hebb-Term: w_{ij} wird um die gewichtete Anzahl von Mustern in der Zielverteilung, die i und j beide auf 1 setzen, erhöht. Sind hidden Neuronen vorhanden, dann muß die Verteilung auf den hidden Neuronen, gegeben ein Vektor der Zielverteilung, statistisch geschätzt werden. Dazu hält man $\mathbf{v}$ fest und läßt die resultierende Maschine zum Gleichgewicht relaxieren (**clamping**).
- Zusätzlich besteht der Gradient aus einem Anti-Hebb Term, der sich aus der mit den vorhanden Gewichten induzierten Grenzverteilung ergibt. Die Grenzverteilung kann man dabei schätzen, indem das Netz zum thermischen Gleichgewicht relaxiert (**free running**).

Anschließend kann die Maschine als Assoziativspeicher verwandt werden: Gegeben ein verrauschtes Muster relaxiert sie evtl. mit Simulated Annealing zum nächstgelegenen wahrscheinlichen Zustand der Grenzverteilung, den sie erst nach einiger Zeit wieder verläßt, um die wahrscheinlichen Zustände der Grenzverteilung aufzusuchen. Das Training dauert aufgrund der notwendigen statistischen Schätzungen allerdings relativ lange, ermöglicht durch das Einbeziehen von hidden Neuronen allerdings eine große Variabilität in der Darstellungsmächtigkeit.

7 Selbstorganisierendes Lernen

Bei selbstorganisierendem Lernen ist keine explizite Funktionalität vorgegeben. Eingabedaten sollen so verarbeitet werden, daß sinnvolle Information extrahiert wird. Die Daten sollen sich selbst organisieren. Kennzeichen von selbstorganisierenden Verfahren ist, daß häufig lokale Information verarbeitet wird; das Training ist inkrementell und schnell. Es sollte in den Daten Redundanz enthalten sein, so daß eine Informationsextraktion überhaupt möglich und notwendig ist – ohne Redundanz, wenn auch nur durch z.B. eine metrische Struktur auf den Daten, sind die Daten selbst ihre beste Reräsentation. Konkrete Aufgaben werden sein:

- Ähnlichkeiten in den Daten lernen, irrelevante und zufällige Faktoren ausfiltern,
- Reduktion der Dimension, indem nur relevante Faktoren gespeichert werden,
- Clustering der Daten, Prototyping,
- Abbilden der Topologie der Daten.

7.1 Hebbsches Lernen

Zunächst betrachten wir wieder ein einfaches Neuron, welches lediglich die gewichtete Summe der Eingaben, d.h. die Korrelation mit der Eingabe berechnet. Formal ist dieses ein $(n, 1)$ feedforward Netz mit linearer Ausgabe und Ausgabebias 0. Die Gewichte bezeichnen wir mit $\mathbf{w}$. Das Neuron soll als Gedächtnis fungieren, das sich relevante Information der Eingaben merkt und irrelevante Information vergißt. Neue Vektoren, die stark korreliert sind mit dem die Information repräsentierenden Vektor $\mathbf{w}$, ergeben dann größere Ausgabewerte als Eingaben, die nur schwach korreliert sind.

Als erstes versuchen wir wieder die Hebb-Regel, d.h. startend mit $\mathbf{w}_0$ (z.B. der Nullvektor oder ein Zufallsvektor) wird in jedem Durchlauf ein Muster $\mathbf{x}$ eingelesen, die Ausgabe $y = \mathbf{w}^t\mathbf{x}$ berechnet, und der Vektor $\mathbf{w}$ je nach dem Vorzeichen von y zu $\mathbf{x}$ ähnlicher oder unähnlicher gemacht:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta y\mathbf{x} = w(t) + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}$$

Die Daten $\mathbf{x}$ sind dabei unabhängig und identisch verteilt gemäß einer zugrundeliegenden Wahrscheinlichkeit P gezogen. Um das asymptotische Verhalten solcher stochastischen Verfahren zu analysieren, nimmt man an, daß sich die Gewichte wesentlich langsamer ändern als die Daten repräsentiert werden. Daher ist es gerechtfertigt, über die Änderungen zu mitteln; statt der tatsächlichen Gewichtsänderung in jedem Schritt betrachtet man die zu erwartende Gewichtsänderung.

[Für kleine konstante Lernraten und geeignete Vorbedingungen kann man zeigen, daß die beim stochastischen Verfahren entstehenden Trajektorien auf kompakten Anfangsstücken in Wahrscheinlichkeit gegen die sich bei der gemittelten Dynamik ergebenden Trajektorie streben [Hornik].]

Es ist also die Gleichung

$$\triangle\mathbf{w} = \eta\sum_i w_{ij}E(x_ix_j) = \eta C\mathbf{w}$$

zu betrachten. Dabei ist $C = (E(x_ix_j))_{ij}$ die **Korrelationsmatrix** der Daten. D.h. ist $\mathbf{x}$ ein n-dimensionaler Zufallsvektor; x_ix_j beschreibt das Produkt der $i$ten und $j$ten Komponente des Zufallsvektors. Je Eintrag in der Matrix wird der Erwartungswert des entsprechenden Produktes berechnet. Wir sind am asymptotischen Verhalten der Regel interessiert. Wir werden im folgenden die Verfahren selbst in der stochastischen Version angeben, asymptotische Analysen aber mit der gemittelten Version durchführen. In beiden Fällen verwenden wir die Notation $\mathbf{w}(t)$ für den sich im konkreten Fall bzw. im Mittel nach dem $t$ten Schritt ergebenden Gewichtsvektor. Punkte mit $\triangle\mathbf{w} = 0$ heißen **Fixpunkte** des Verfahrens. Nur in solchen Punkten ändert sich nichts. Fixpunkte heißen **stabil** falls man eine Umgebung finden kann, so daß für jeden Startpunkt innerhalb der Umgebung die sich ergebende Folge gegen den Fixpunkt konvergiert. Insbesondere darf man also auch den Fixpunkt leicht ändern, was ja bei der stochastischen Version in der Regel der Fall ist, ohne sich je zu weit vom Fixpunkt zu entfernen. Nur stabile Fixpunkte können also bei der stochastischen Version des Verfahrens als asymptotisch erreichbare Punkte angesehen werden.

Falls es stabile Fixpunkte dieses Verfahrens gäbe, gegen die der Vektor $\mathbf{w}$ asymptotisch strebt, so würde

$$\triangle\mathbf{w} = \eta C\mathbf{w} = 0$$

gelten. $\mathbf{w}$ wäre also 0 oder ein sogenannter Eigenvektor der Matrix C zum Eigenwert 0. [Ein **Eigenvektor** $\mathbf{x}$ einer Matrix M ist ein Vektor $\mathbf{x} \neq 0$ mit $M\mathbf{x} = \lambda\mathbf{x}$ für eine skalare Größe λ, die dann **Eigenwert** zum Eigenvektor $\mathbf{x}$ heißt.] Es ist interessant zu sehen, daß man obiges Verfahren auch als Gradientenabstieg der Funktion

$$-\frac{1}{2}\mathbf{w}^tC\mathbf{w}$$

auffassen kann. Der Gradient liefert genau $-C\mathbf{w}$.

Als erstes sieht man daran, daß 0 nicht stabil ist, sondern ein lokales Maximum der zu minimierenden Funktion. Desweiteren sieht man, daß Richtungen $\mathbf{w}$, so daß bzgl. dieser Richtung die Korrelation der Daten maximiert wird, als stabile Fixpunkte in Frage kommen. Ist allerdings die Länge von $\mathbf{w}$ nicht beschränkt, dann strebt $|\mathbf{w}|$ gegen unendlich, denn ist $\mathbf{w}^tC\mathbf{w} > 0$, dann ist $(\lambda\mathbf{w})^tC(\lambda\mathbf{w})$ für $\lambda > 1$ nur noch größer. D.h. obige Lernregel ist instabil und liefert einen explodierenden Gewichtsvektor. Wäre allerdings die Länge beschränkt, was wäre dann das Ergebnis?

Hier soll zunächst ein kleiner mathematischer Exkurs folgen, der die Matrix C ein wenig näher beleuchtet.

- C ist offensichtlich symmetrisch.

- C ist positiv semidefinit, d.h. $\mathbf{w}^t C\mathbf{w} \geq 0$ gilt für alle Vektoren $\mathbf{w}$, denn $\mathbf{w}^t E(\mathbf{x}\mathbf{x}^t)\mathbf{w} = E(\mathbf{w}^t\mathbf{x}(\mathbf{w}^t\mathbf{x})^t) = E((\mathbf{w}^t\mathbf{x})^2) \geq 0$.

- Es gibt eine **Orthonormalbasis** für C bestehend aus Eigenvektoren, da C positiv semidefinit ist. Wir nennen die Eigenvektoren $e_1, \ldots, e_n$, die zugehörigen Eigenwerte $\lambda_1, \ldots, \lambda_n$. Orthonormalbasis bedeutet:

$$e_i^t e_j = 0 \quad \text{für } i \neq j, \quad e_i^t e_i = 1.$$

- Man beachte, daß die Eigenwerte die Nullstellen des sogenannten charakteristischen Polynoms der Matrix sind, welches bei konkret vorgegebenen Daten $\xi^1, \ldots, \xi^p$ mithilfe der Koeffizienten $E(\mathbf{x}_i\mathbf{x}_j) \approx \sum_k \xi_i^k \xi_j^k / p$ abgeschätzt werden kann. Die Situation, daß ein Eigenwert genau 0 ist oder zwei Eigenwerte gleich sind, ist dabei eine Nullmenge, d.h. taucht bei Rauschen so gut wie nie auf. Daher nehmen wir im folgenden

$$\lambda_1 > \ldots > \lambda_n > 0$$

an. Die Eigenwerte müssen dabei notwendig positiv sein, da $0 \leq e_i^t C e_i = \lambda_i e_i^t e_i = \lambda_i$ gilt. Die Vektoren e_i sind bis auf ihr Vorzeichen eindeutig, da die Menge aller Eigenvektoren zum Eigenwert λ_i bei n verschiedenen Eigenvektoren einen eindimensionalen Vektorraum darstellt. Jeder andere Eigenvektor ist ein Vielfaches eines e_i, denn man berechnet für einen Eigenvektor $\mathbf{w} = \sum \alpha_i e_i$ zum Eigenwert λ: $\sum \lambda\alpha_i e_i = \lambda\mathbf{w} = C\mathbf{w} = C(\sum \alpha_i e_i) = \sum \lambda_i \alpha_i e_i$. Die Koeffizienten in obiger Darstellung sind aber eindeutig, da die e_i eine Basis bilden. Also sind alle Koeffizienten α_i bis auf maximal einen 0.

- Für einen beliebigen Vektor $\mathbf{w} = \sum \alpha_i e_i$ kann man den Term $\mathbf{w}^t C\mathbf{w}$ mithilfe der Eigenvektoren berechnen als $\sum \alpha_i^2 \lambda_i$.

Die Bedeutung der Eigenvektoren e_i wird anhand folgenden Satzes klar:

Satz 7.1 *Die Korrelation $\mathbf{w}^t C\mathbf{w}$ wird durch e_1 maximiert, sofern man sich auf Vektoren der Länge 1 beschränkt. Im zu e_1 orthogonalen Raum maximiert e_2 die Korrelation, im zu e_1 und e_2 orthogonalen Raum $e_3, \ldots$, alles unter der Bedingung, daß die Vektoren Betrag 1 haben.*

Beweis: Es ist

$$\begin{aligned} \left(\sum \alpha_i e_i\right)^t C \left(\sum \alpha_i e_i\right) &= \sum \alpha_i^2 \lambda_i \\ &\leq \sum \alpha_i^2 \lambda_1 = \lambda_1 \end{aligned}$$

Andererseits wird aber für e_1 der Wert λ_1 angenommen.

Für Vektoren im jeweiligen zu $e_1, \ldots, e_i$ orthogonalen Raum fallen die Komponenten $\alpha_1, \ldots, \alpha_i$ weg, eine analoge Rechnung wie oben zeigt, daß dann noch das Maximum λ_{i+1} etwa durch e_{i+1} erreicht werden kann. □

Wir nehmen jetzt an, daß die Daten um 0 zentiert sind, d.h. $E(x_i) = 0$ Für alle i. Die Korrelation ist also absteigend in Richtung $e_1, e_2, \ldots$ maximal. Diese Richtungen heißen auch **Hauptkomponentenrichtungen** der Verteilung. Der Vektor e_i bzw. sein negatives heißt $i$te **Hauptkomponente** der Matrix. Sind die Daten um den Koordinatenursprung zentriert, so bedeutet das genau, daß die Streuung der Daten in den ersten Hauptkomponentenrichtungen maximal ist. Möchte man also maximale Information über die Daten behalten, aber die Dimensionen auf lediglich k reduzieren,

so kann man die Daten bzgl. der ersten k Hauptkomponenten statt der ursprünglichen Daten betrachten. Dieses ist natürlich nur ein Plausibilitätsargument, allerdings kann man folgendes zeigen: Möchte man die Daten auf k orthogonale Richtungen linear transformieren, so daß der Informationsverlust, d.h. in diesem Fall der quadratische Fehler zwischen den vollen Daten und den auf k Dimensionen reduzierten Daten, minimiert ist, dann sind die ersten k Hauptkomponenten bei um $\mathbf{0}$ zentrierten Daten optimal. Eine **Hauptkomponentenanalyse**, welche auch mit klassischen Methoden durchführbar ist, ist eine verbreitete Vorverarbeitung zur Dimensionsreduktion.

Zurück zur Hebb-Regel: Es gilt zwar, daß das Verfahren divergiert, aber betrachtet man die Größe $\mathbf{w}(t)/|\mathbf{w}(t)|$, so konvergiert diese (im Mittel) gegen e_1 (oder das negative). Um die Stabilität des Verfahrens zu erzwingen, könnten wir die Regel also zu

$$\mathbf{w}(t+1) = \frac{\mathbf{w}(t) + \eta y \mathbf{x}}{|\mathbf{w}(t) + \eta y \mathbf{x}|}$$

modifizieren.

Satz 7.2 *Obiges Verfahren konvergiert im Mittel gegen e_1 oder $-e_1$.*

Beweis: Die Korrelation $e_i^t\mathbf{w}(t+1)$ für den sich im Mittel ergebenden Vektor $\mathbf{w}(t+1)$ berechnet sich als

$$\frac{e_i^t(\mathbf{w}(t) + \eta C\mathbf{w}(t))}{|\mathbf{w}(t) + \eta C\mathbf{w}(t)|} = e_i^t\mathbf{w}(t)\,\frac{1+\eta\lambda_i}{|\mathbf{w}(t) + \eta C\mathbf{w}(t)|}.$$

Bzgl. der Basis $e_1, \ldots, e_n$ sind die Koeffizienten eines Vektors $\mathbf{w}$ genau durch die Korrelation $e_i^t\mathbf{w}$ gegeben. Die Koeffizienten des neuen Vektors $\mathbf{w}(t+1)$ bzgl. der Koordinaten e_i ergeben sich also bis auf einen positiven multiplikativen Faktor als

$$\left(w(t)_1, w(t)_2 \frac{1+\eta\lambda_2}{1+\eta\lambda_1}, \ldots, w(t)_n \frac{1+\eta\lambda_n}{1+\eta\lambda_1}\right),$$

wobei die Terme $w(t)_i$ die alten Koeffizienten bzgl. e_i darstellen. Das heißt, in jedem Schritt wird der Anteil in Richtung e_1 verglichen zum Rest größer, da ja alle Terme $(1+\eta\lambda_i)/(1+\eta\lambda_1)$. für $i \neq 1$ kleiner als 1 sind.

Ein Spezialfall ist gegeben, falls der Anteil in Richtung e_1 zu Beginn 0 ist. Formal würde obiger Ausdruck dann gegen den Nullvektor konvergieren. Sofern man mit einem zufälligen $\mathbf{w}(0)$ startet, tritt dieser Fall aber nur mit Wahrscheinlichkeit Null auf. □

Der Nachteil ist, daß diese Regel globale Information, d.h. den Wert aller Gewichte benötigt, um ein einzelnes Gewicht ändern zu können. Sie kann also nicht verteilt auf die Gewichte implementiert werden. Eine Alternative bietet es, wenn man die Taylorentwicklung um $\eta = 0$ obiger Regel betrachtet; wir betrachten jetzt wieder die stochastische Version:

$$\begin{aligned} &\frac{\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}}{|\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}|} \\ \approx\ & \frac{\mathbf{w}}{|\mathbf{w}|} + \eta\left(\frac{\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}}{|\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}|}\right)'(\eta = 0) \\ =\ & \frac{\mathbf{w}}{|\mathbf{w}|} + \eta\left(\frac{\mathbf{w}^t\mathbf{x}\mathbf{x}|\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}| - (\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x})\dfrac{(\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x})^t(\mathbf{w}^t\mathbf{x}\mathbf{x})}{|\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}|}}{|\mathbf{w} + \eta\mathbf{w}^t\mathbf{x}\mathbf{x}|^2}\right)(\eta = 0) \\ =\ & \frac{\mathbf{w}}{|\mathbf{w}|} + \eta\left(\frac{\mathbf{w}^t\mathbf{x}\mathbf{x}}{|\mathbf{w}|} - \frac{\mathbf{w}\mathbf{w}^t\mathbf{w}^t\mathbf{x}\mathbf{x}}{|\mathbf{w}|^3}\right) \end{aligned}$$

Beachtet man $|\mathbf{w}| \approx 1$, so kann man obigen Term durch die Regel

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(\mathbf{w}(t)^t\mathbf{x}\mathbf{x} - (\mathbf{w}(t)^t\mathbf{x})^2\mathbf{w}(t)) = \mathbf{w}(t) + \eta(y(t)\mathbf{x} - y(t)^2\mathbf{w}(t)),$$

die sogenannte **Oja-Regel**, ersetzen. Mittelt man über die Eingaben, erhält man aus der stochastischen Version die gemittelte Version

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(C\mathbf{w}(t) - (\mathbf{w}(t)^tC\mathbf{w}(t))\mathbf{w}(t)) .$$

Satz 7.3 *Fixpunkte der (gemittelten) Oja-Regel sind die Hauptkomponenten und der Nullvektor. Einzige stabile Fixpunkte sind die Vektoren e_1 und $-e_1$, d.h. die Hauptkomponenten zum größten Eigenwert.*

Beweis: Für Fixpunkte $\mathbf{w}$ gilt

$$C\mathbf{w} - (\mathbf{w}^tC\mathbf{w})\mathbf{w} = 0 \Rightarrow C\mathbf{w} = (\mathbf{w}^tC\mathbf{w})\mathbf{w},$$

also ist $\mathbf{w}$ Eigenvektor von C zum Eigenwert $\mathbf{w}^tC\mathbf{w} = \lambda$ oder der Nullvektor. Im ersten Fall sei etwa $\mathbf{w} = \alpha e_i$ mit $\alpha \neq 0$. Man rechnet

$$\alpha\lambda_i e_i = C\alpha e_i = C\mathbf{w} = (\mathbf{w}^tC\mathbf{w})\mathbf{w} = (\alpha e_i)^tC(\alpha e_i)\alpha e_i = \alpha^3\lambda_i e_i \Rightarrow \alpha = \pm 1.$$

Folglich sind nur die Vektoren e_i (oder $-e_i$) und der Nullvektor Fixpunkte.

Um Stabilität zu testen, berechnen wir, ob die Jakobimatrix J der Funktion $\mathbf{w} \mapsto -(C\mathbf{w} - (\mathbf{w}^tC\mathbf{w})\mathbf{w})$ am Fixpunkt positiv definit ist. In diesem Fall ist der Punkt stabil. Gibt es Richtungen $\mathbf{x}$, so daß $\mathbf{x}^tJ\mathbf{x} < 0$ ist, dann ist das Verfahren instabil, wenn man sich aus diesen Richtungen nähert. Es ist

$$J(\mathbf{w}) = -C + (\mathbf{w}^tC\mathbf{w})I + 2\mathbf{w}\mathbf{w}^tC$$

mit der Identitätsmatrix I.

[An dieser Stelle sind evtl. ein paar Regeln angebracht, die mehrdimensionales Ableiten einfacher machen. Man kann aber alle Rechnungen auch komponentenweise durchführen und benötigt dann nicht mehr als eindimensionale Analysis. Für eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ bezeichne $J(f)$ die Jakobimatrix, d.h. $J(f) = (\partial f_i/\partial x_j)_{ij}$. Dann gilt

- $J(\lambda(f+g)) = \lambda(J(f) + J(g))$,
- $J(x \mapsto Ax) = A$ für eine Matrix A,
- $J(x \mapsto x^tAx) = 2Ax$ für eine symmetrische Matrix A,
- $J(f \circ g) = Jf \cdot Jg$,
- $J(f \cdot g) = f \cdot J(g) + g \cdot (J(f))^t$ für $f : \mathbb{R}^n \to \mathbb{R}$.]

Also

$$\begin{aligned} e_j^tJ(e_i)e_k &= e_j^t(-C + (e_i^tCe_i)I + 2e_ie_i^tC)e_k \\ &= -\lambda_k e_j^te_k + \lambda_i e_j^te_k + 2e_j^te_i\lambda_k e_i^te_k \\ &= \begin{cases} 0 & j \neq k \\ 2\lambda_i & i = j = k \\ \lambda_i - \lambda_j & i \neq j = k \end{cases} \end{aligned}$$

Wegen $(\sum \alpha_k e_k)^tJ(e_i)(\sum \alpha_k e_k) = \sum \alpha_j\alpha_k e_j^tJ(e_i)e_k = \sum_k e_k^tJ(e_i)e_k$ ist also der einzige stabile Fixpunkt e_1 oder das negative, da nur dann alle Werte $\lambda_1 - \lambda_i$ für $i \neq 1$ größer als 0 sind. Für 0 ist die Jakobimatrix $-C$ und also negativ definit (d.h. $-J$ ist positiv definit), für e_i mit $i \neq 1$ sind Richtungen instabil, die von Eigenvektoren zu größeren Eigenwerten kommen. $\square$

Es wurden alternative Regeln zur Oja-Regel vorgeschlagen:

- **Yuille-Regel:**
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(\mathbf{w}(t)^t\mathbf{x}\mathbf{x} - |\mathbf{w}(t)|^2\mathbf{w}(t))$$
mit der gemittelten Version
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(C\mathbf{w}(t) - |\mathbf{w}(t)|^2\mathbf{w}(t)).$$
Fixpunkte sind 0 und die Vektoren $\sqrt{\lambda_i}e_i$. Die Jakobimatrix der das Differenzenverfahren beschreibenden Funktion berechnet sich als
$$J(\mathbf{w}) = -C + \mathbf{w}^t\mathbf{w}I + 2\mathbf{w}\mathbf{w}^t.$$
Diese ist an der Stelle 0 negativ definit und genau in $\sqrt{\lambda_1}e_1$ positiv definit, denn
$$e_j^tJ(\sqrt{\lambda_i}e_i)e_k = \begin{cases} 0 & j \neq k \\ 2\lambda_i & j = k = i \\ \lambda_i - \lambda_j & j = k \neq i \end{cases}$$
Die gemittelte Version der Yuille-Regel ist ein Gradientenabstieg zur Funktion
$$-\frac{1}{2}\mathbf{w}^tC\mathbf{w} + \frac{1}{4}|\mathbf{w}|^4.$$

- **Hassoun-Regel:**
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\left(\mathbf{w}(t)^t\mathbf{x}\mathbf{x} - \lambda\left(1 - \frac{1}{|\mathbf{w}(t)|}\right)\mathbf{w}(t)\right)$$
mit $\lambda > 0$ und hinreichend von $\mathbf{0}$ entferntem $\mathbf{w}$. Die gemittelte Version liefert
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\left(C\mathbf{w}(t) - \lambda\left(1 - \frac{1}{|\mathbf{w}(t)|}\right)\mathbf{w}(t)\right)$$
mit den Fixpunkten $\lambda/(\lambda - \lambda_i)e_i$. Die Jakobimatrix berechnet sich als
$$J(\mathbf{w}) = -C + \lambda\left(1 - \frac{1}{|\mathbf{w}|}\right)I + \lambda\frac{\mathbf{w}\mathbf{w}^t}{|\mathbf{w}|^3}.$$
Diese ist genau in $\lambda/(\lambda - \lambda_1)e_1$ positiv definit, falls $\lambda > \lambda_1$, denn
$$e_j^tJ\left(\frac{\lambda}{\lambda - \lambda_1}e_i\right)e_k = \begin{cases} 0 & j \neq k \\ \lambda - \lambda_i & j = k = i \\ \lambda_i - \lambda_j & j = k \neq i \end{cases}$$
Die gemittelte Version der Hassoun-Regel ist ein Gradientenabstieg zur Funktion
$$-\frac{1}{2}\mathbf{w}^tC\mathbf{w} + \frac{\lambda}{2}(|\mathbf{w}| - 1)^2.$$

Netze und Lernregeln, so daß sämtliche Hauptkomponenten extrahiert werden können, wurden unter anderen von Sanger vorgeschlagen. Statt einem Neuron betrachtet man n Neuronen mit Gewichten $\mathbf{w}_i$. Die Gewichtsänderungen sind nicht mehr lokal, sondern enthalten einen Term, der tendentiell die einzelnen Gewichte zu verschiedenen Ausprägungen zwingt, so daß alle Hauptkomponenten erhalten werden. Genauer ist die Gewichtsänderung bei Sanger von der Form

$$\triangle \mathbf{w}_i = \eta(\mathbf{w}_i^t \mathbf{x}) \left(\mathbf{x} - \sum_{j=1}^{i} (\mathbf{w}_j^t \mathbf{x}) \mathbf{w}_j \right).$$

Das reduziert sich offensichtlich für $i = 1$ zur Oja-Regel. Diese Regel sorgt für eine Konvergenz der Gewichte $\mathbf{w}_i$ im Mittel zu e_i.

Satz 7.4 *Einziger zu erwartender stabiler Fixpunkt der Regel*

$$\triangle \mathbf{w}_i = \eta(\mathbf{w}_i^t \mathbf{x}) \left(\mathbf{x} - \sum_{j=1}^{i} (\mathbf{w}_j^t \mathbf{x}) \mathbf{w}_j \right)$$

ist der Vektor $(e_1, \ldots, e_n)$.

Beweis: Die gemittelte Regel hat die Form

$$\triangle \mathbf{w}_i = \eta \left(C\mathbf{w}_i - \sum_{j=1}^{i} (\mathbf{w}_j^t C \mathbf{w}_i) \mathbf{w}_j \right).$$

Für Fixpunkte berechnet man sukzessive, daß w_i entweder 0 oder ein Eigenvektor e_{i_j} zum Eigenwert λ_{i_j} sein muß: Für $\mathbf{w}_{i+1} = \sum \alpha_l e_l$ gilt

$$\begin{aligned} \sum \lambda_l \alpha_l e_l &= C\mathbf{w}_{i+1} = \\ (\mathbf{w}_{i+1}^t C \mathbf{w}_{i+1}) \mathbf{w}_{i+1} + \sum_{j=1}^{i} (e_{i_j}^t C \mathbf{w}_{i+1}) e_{i_j} &= \sum \lambda_l \alpha_l^3 e_l + \sum_{j=1}^{i} \lambda_{i_j} \alpha_{i_j} e_{i_j}. \end{aligned}$$

Da Darstellungen bzgl. der Basisvektoren eindeutig sind, gilt für alle i_j

$$\lambda_{i_j} \alpha_{i_j} = \lambda_{i_j} \alpha_{i_j}^3 + \lambda_{i_j} \alpha_{i_j},$$

also $\alpha_{i_j} = 0$. Da damit aber $\mathbf{w}_{i+1}$ Eigenvektor ist, ist er gleich einem Vielfachen eines Vektors e_j. Für alle anderen Koeffizienten gilt

$$\lambda_l \alpha_l = \lambda_l^3 \alpha_l^3$$

und damit $\mathbf{w}_{i+1} = \pm e_{j_{i+1}}$. Ein Fixpunkt $\mathbf{w}$ setzt sich aus einer Permutation der Eigenvektoren e_i oder 0 zusammen.

Die das Differenzenverfahren beschreibenden Funktion f setzt sich aus verschiedenen Blöcken zusammen. Betrachtet man nur den ersten Block, dann reduziert sich alles zur einfachen Oja-Regel, deren einziger stabiler Fixpunkt e_1 darstellt. Hat man induktiv gezeigt, daß die ersten Komponenten sich aus $\mathrm{e}_1, \ldots, \mathrm{e}_{i-1}$ zusammensetzen, dann erhält man für den i ten Block notwendig e_i: Die Änderung des i-ten Blocks berechnet sich als

$$\begin{aligned} \triangle \mathbf{w}_i &= \eta \left(C\mathbf{w}_i - \sum_{j=1}^{i} (\mathbf{w}_j^t C \mathbf{w}_i) \mathbf{w}_j \right) \\ &\approx \eta \left(C\mathbf{w}_i - \mathbf{w}_i^t C \mathbf{w}_i \mathbf{w}_i - \sum_{j=1}^{i-1} (\mathrm{e}_j^t C \mathbf{w}_i) \mathrm{e}_j \right) \\ &= \eta (C\mathbf{w}_i^{\perp} - \mathbf{w}_i^t C \mathbf{w}_i \mathbf{w}_i), \end{aligned}$$

mit $\mathbf{w}_i^{\perp}$ als dem Anteil von $\mathbf{w}_i$ im zu $\mathrm{e}_1, \ldots, \mathrm{e}_{i-1}$ orthogonalen Raum. Dieser Anteil wird nämlich gerade durch $\sum_{j=1}^{i-1} (\mathrm{e}_j^t C \mathbf{w}_i) \mathrm{e}_j$ abgezogen. Man beachte, daß C diesen Raum in sich selbst abbildet. Wegen des Terms $\mathbf{w}_i^t C \mathbf{w}_i \mathbf{w}_i$ veringert sich der Koeffizient μ_j von $\mathbf{w}_i$ in Bezug auf e_j mit $j < i$ in jedem Schritt denn für hinreichend kleines η. Dann reduziert sich aber obige Regel asymptotisch zur Oja-Regel im beschränkten Raum der zu $\mathrm{e}_1, \ldots, \mathrm{e}_{i-1}$ orthogonalen Vektoren, konvergiert also gegen e_i. □

Eine Alternative, um die ersten k Hauptkomponenten oder ähnlich relevante Richtungen zu extrahieren, ist etwa durch ein ganz normales feedforward Netz der Architektur (n, k, n) – ein Encoder Netz – mit der Identität als Aktivierungsfunktion gegeben. Trainiert man mit Backpropagation die Identität auf den gegebenen Daten, so muß das Netz diese über das Nadelöhr der k Neuronen realisieren, also auf k relevante Richtungen projizieren.

Als weiter Alternative wurde vorgeschlagen, k einfache lineare Neuronen mit der ursprünglichen Hebb-Regel plus einen Normierungsterm zu trainieren, aber zusätzlich trainierbare laterale Hemmungen zwischen den Neuronen einzuführen. D.h. es gibt Gewichte w_{ij} vom Neuron i zum Neuron j für $j > i$, die in jedem Trainingsschritt mit einer Anti-Hebb-Regel trainiert werden, d.h.

$$\triangle w_{ij} = -\eta y_i y_j.$$

Das führt dazu, daß das erste Neuron wie vorher die erste Hauptkomponente lernt, das zweite Neuron erhält aber vom ersten starke negative Verbindung, sofern es eine zum ersten Neuron ähnliche Ausgabe besitzt. Seine Ausgabe würde also gedämpft, falls es auch die erste Hauptkomponente extrahieren würde. Daher nimmt es für sich die zweite Hauptkomponente in Anspruch u.s.w.

7.2 Learning Vector Quantization

Bei Vektorquantisierung wird ebenfalls ein (n, k) Netz mit k konkurrierenden Neuronen betrachtet, die für repräsentative Bereiche der Daten zuständig sein sollen. Die Gewichte des $i$ten Ausgabeneurons seien mit $\mathbf{w}^i$ bezeichnet. Anders als bei einer Hauptkomponentenanalyse konkurrieren hier die Neuronen miteinander um die Eingabe, so daß jeweils nur das Neuron mit der größten Ausgabe gemäß Hebbschem Lernen seine Gewichte ändern darf. Das führt dazu, daß nicht Hauptkomponentenrichtungen extrahiert, sondern die Daten durch die Neuronen quantisiert, d.h. in Klassen eingeteilt werden. Jeder Vektor reagiert auf Neuronen in gewissen Bereichen des Eingaberaumes relativ stark, bei anderen wird er eine kleine Ausgabe liefern.

Wir müssen jedoch zunächst dafür sorgen, daß die Gewichte nicht explodieren. Ferner ist bei diesem Wettbewerbsansatz erkenntlich, daß beliebige Daten oder beliebige Gewichte zuzulassen nicht ratsam ist: Vektoren mit großer Norm haben in der Regel ein größeres Skalarprodukt allein aufgrund ihrer Länge, auch wenn ihre Richtung nicht übereinstimmt. Daher ist es ratsam, entweder nur normierte Eingaben und Gewichtsvektoren zuzulassen, oder vom Messen der Korrelation zum tatsächlichen euklidischen Abstand überzugehen. Das Neuron mit größtem Skalarprodukt entspricht unter Annahme normierter Daten genau dem Neuron mit kleinstem Abstand zur Eingabe, denn

$$\begin{aligned} \mathbf{x}^t \mathbf{w}^i \text{ ist für } i \text{ maximal} &\iff \mathbf{x}^t \mathbf{x} - 2\mathbf{x}^t \mathbf{w}^i + \underbrace{\mathbf{w}^{i^t} \mathbf{w}^i}_{=1} \text{ ist für } i \text{ minimal} \\ &\iff (\mathbf{x} - \mathbf{w}^i)^t (\mathbf{x} - \mathbf{w}^i) \text{ ist für } i \text{ minimal} \\ &\iff |\mathbf{x} - \mathbf{w}^i| \text{ ist für } i \text{ minimal} \end{aligned}$$

Statt $\mathbf{x}$ zu addieren, kann man eine Explosion der Gewichte verhindern, indem man einen Anteil des Vektors $\mathbf{x} - \mathbf{w}^i$ zu $\mathbf{w}^i$ addiert, der den Gewichtsvektor in Richtung $\mathbf{w}^i$ zieht. Insgesamt wird Vektorquantisierung also zu folgendem Verfahren:

wiederhole
berechne $y^i = |\mathbf{x} - \mathbf{w}^i|$
für ein i mit minimalem y^i
$\mathbf{w}^i := \mathbf{w}^i + \eta(\mathbf{x} - \mathbf{w}^i)$

mit einer Lernrate $\eta < 1$. η muß beschränkt sein, da sonst die Anpassung den Gewichtsvektor über den Vektor $\mathbf{x}$ hinausbewegt. Sofern die Lernrate so klein ist, daß man über die Änderungsschritte für die einzelnen Eingaben mitteln kann, und sofern die Funktion

$$M_i^p = \begin{cases} 1 & i \text{ ist Gewinner für das Pattern } \mathbf{x}^p \\ 0 & \text{sonst} \end{cases}$$

über die Zeit konstant ist, etwa weil sich die Gewichte schon nahezu eingestellt haben, dann ist obiges Verfahren ein Gradientenabstieg zur Funktion

$$\frac{1}{2}\sum_p (\mathbf{x}^p - \mathbf{w}^i)^2 M_i^p$$

für das $i$te Neuron. Über alle i aufsummiert, ist das insgesamt minimal, wenn die Abstände der Neuronen zu den jeweils nächsten Eingabepattern, für die sie zuständig sind, möglichst klein ist, das heißt die Neuronen sich tendentiell auf die Häufungspunkte in den Daten verteilt haben. Allerdings besitzt die Funktion viele lokale Minima und ist bei variierendem M_i^p unstetig. Die Mittelung ist nicht unbedingt gerechtfertigt, man kann den Fall beobachten, daß sich etwa die durch zyklisch aufeinanderfolgende Pattern ergebenden Änderungen genau aufheben.

Das trainierte Netz kann zur Klassifikation der Daten eingesetzt werden, denn die Neuronen bilden sog. **Codebookvektoren** die den Eingabebereich, für den sie minimale euklidische Norm im Vergleich zu den anderen Neuronen besitzen, repräsentieren. Eine Eingabe wird jeweils der durch den nächsten Codebookvektor repräsentierten Klasse zugeordnet. Das Netz kann nicht nur zum Clustering und Prototyping, sondern auch zu einer Funktionsapproximation eingesetzt werden, indem ein Vektor je zum Funktionswert des nächsten Codebookvektors abgebildet wird. Zu beachten ist, daß dieses Netz kein Neuronales Netz im Sinne unserer ursprünglichen Definition ist, da es die Aktivierung $|\mathbf{x}-\mathbf{w}|$ statt $\mathbf{w}^t\mathbf{x}$ berechnet und die Ausgabefunktion – die Winner Takes All Funktion – keine lokale Weiterverarbeitung der Aktivierung ist. Die WTA Funktion könnte man allerdings in einem feedforward Perzeptronnetz der Tiefe zwei realisieren, und wir haben gesehen, daß bei normierten Daten der euklidische Abstand durch ein Skalarprodukt ausgetauscht werden könnte.

Eine weitere Anmerkung ist hier angebracht: Bei diesen Verfahren ist eine Normierung der Eingabedaten unumgänglich notwendig! Hat eine Eingabekomponente wesentlich größere Eingaben als eine zweite, dann ist diese Eingabe bei Ähnlichkeitsberechnungen wesentlich stärker gewichtet als die zweite.

Kohonen hat vorgeschlagen, Vektorquantisierung in einem überwachten Szenario, das heißt als **Learning Vector Quantization** anzuwenden. Die Aufgabe ist, eine Funktion $f : \mathbb{R}^n \to \{1, \ldots, k\}$ zu lernen. Die Idee ist, jede Klasse $f^{-1}(i)$ durch einen oder mehrere Codebookvektoren zu repräsentieren und eine Eingabe $\mathbf{x}$ der Klasse des nächsten Codebookvektors zuzuordnen.

- **LVQ1**: Es werden für die Klassen 1 bis k ein oder mehrere Neuronen erzeugt und deren Gewichtsvektoren zufällig, mit zufälligen Pattern der jeweiligen Klasse, durch die Schwerpunkte der jeweiligen Klasse oder anders sinnvoll initialisiert. Dann werden genau wie bei Vektorquantisierung die Muster präsentiert und je ein Gewinnerneuron mit kleinstem euklidischem Abstand zur Eingabe berechnet. Für dieses Neuron $\mathbf{w}^i$ ändert man

$$\triangle\mathbf{w}^i = \begin{cases} \eta(\mathbf{x} - \mathbf{w}^i) & \text{falls } f(\mathbf{x}) = \text{durch } \mathbf{w}^i \text{ repräsentierte Klasse} \\ -\eta(\mathbf{x} - \mathbf{w}^i) & \text{sonst.} \end{cases}$$

Dabei ist die Lernrate $\eta \in [0, 1[$ entweder konstant oder im Laufe des Verfahrens fallend, um Konvergenz zu erzwingen.

- **OLVQ**: Optimized LVQ verwendet eine eigene Lernrate η_i für jedes Neuron $\mathbf{w}^i$, d.h. für den Gewinner berechnet man

$$\triangle\mathbf{w}^i = \begin{cases} \eta_i(\mathbf{x} - \mathbf{w}^i) & \text{falls } f(\mathbf{x}) = \text{durch } \mathbf{w}^i \text{ repräsentierte Klasse} \\ -\eta_i(\mathbf{x} - \mathbf{w}^i) & \text{sonst.} \end{cases}$$

Die Lernrate η_i wird so angepaßt, daß die Änderung, die jedes Muster hervorruft, möglichst den gleichen Effekt hat. Formal ist $\eta_i(t+1) = \eta_i(t)$, falls i kein Gewinner zum Zeitpunkt t ist, und

$$\eta_i(t+1) = \frac{\eta_i(t)}{1 + s(t)\eta_i(t)}$$

sonst, wobei $s(t) = 1$ gilt, falls die Klasse des Gewinners $\mathbf{w}^i$ korrekt ist, $s(t) = -1$ ist, falls die Klasse des Gewinners falsch ist. Das heißt, Neuronen im Zentrum einer Klasse haben schnell abfallende Lernraten, Neuronen am Rand, die bei Eingaben häufig fälschlicherweise als Gewinner vervorgehen, haben große Lernraten, so daß sie schnell vom Rand weggedrängt werden. Dabei sollte man darauf achten, daß die Lernrate nicht größer als 1 wird.

Formal kann obige Lernrate wir folgt motiviert werden: Zwei Änderungen von $\mathbf{w}^i$, o.E. in aufeinanderfolgenden Schritten, führen zur Änderung

$$\begin{aligned} \mathbf{w}^i(t+2) &= \mathbf{w}^i(t+1) + \eta(t+1)s(t+1)(\mathbf{x}(t+1) - \mathbf{w}^i(t+1)) \\ &= (1 - \eta(t+1)s(t+1))\mathbf{w}^i(t+1) + \eta(t+1)s(t+1)\mathbf{x}(t+1) \\ &= (1 - \eta(t+1)s(t+1))(\mathbf{w}^i(t) + \eta(t)s(t)(\mathbf{x}(t) - \mathbf{w}^i(t))) \\ &\quad + \eta(t+1)s(t+1)\mathbf{x}(t+1) \\ &= (1 - \eta(t+1)s(t+1))\eta(t)s(t)\mathbf{x}(t) + \eta(t+1)s(t+1)\mathbf{x}(t+1) \\ &\quad + (1 - \eta(t+1)s(t+1))\mathbf{w}^i(t) - (1 - \eta(t+1)s(t+1))\eta(t)s(t)\mathbf{w}^i(t) \end{aligned}$$

Damit die durch $\mathbf{x}(t)$ und $\mathbf{x}(t+1)$ bewirkte Änderung gleich gewichtet ist, muß

$$|(1 - \eta(t+1)s(t+1))\eta(t)s(t)| = |\eta(t+1)s(t+1)|$$

gelten, d.h.

$$\eta(t+1) = \frac{\eta(t)}{1 + \eta(t)s(t)}$$

OLVQ sorgt für eine schnelle Konvergenz des Verfahrens.

- **LVQ2.1**: Dieses Verfahren kann zum Feintuning der Klassengrenzen verwendet werden. Es betrachtet jeweils die ersten beiden Gewinner $\mathbf{w}^i$ und $\mathbf{w}^j$ und ändert diese, sofern sie verschiedenen Klassen angehören und das Eingabepattern $\mathbf{x}$ nicht sehr dicht an einem der beiden Gewinner liegt, d.h.

$$A = \min\left(\frac{|\mathbf{x} - \mathbf{w}^i|}{|\mathbf{x} - \mathbf{w}^j|}, \frac{|\mathbf{x} - \mathbf{w}^j|}{|\mathbf{x} - \mathbf{w}^i|}\right) > \frac{1-v}{1+v}, \quad v \approx 0.25$$

sollte gelten. Dann wird das Neuron, das zur selben Klasse wie $\mathbf{x}$ gehört, etwa $\mathbf{w}^i$, zu $\mathbf{x}$ ähnlicher, das andere Neuron $\mathbf{w}^j$ zu $\mathbf{x}$ unähnlicher gemacht, d.h.

$$\begin{aligned} \triangle\mathbf{w}^i &= \eta(\mathbf{x} - \mathbf{w}^i) \\ \triangle\mathbf{w}^j &= -\eta(\mathbf{x} - \mathbf{w}^j) \end{aligned}$$

Wie vorher ist $\eta < 1$. Dieses Verfahren stellt lediglich Neuronen an den Klassengrenzen ein und optimiert nicht die Neuronen in der Mitte von Klasse. Daher wendet man es meist erst an, nachdem eine Grobklassifikation etwa mithilfe von OLVQ gelernt wurde. Die Bedingung, daß $\mathbf{x}$ in einem Fenster liegen muß, bedeutet, daß Punkte nahe bei einem Klassenvektor nicht zur Änderung beitragen, denn der Klassenvektor ist entweder korrekt und bedarf keiner weiteren Einstellung, oder er ist falsch, allerdings eine Verbesserung wahrscheinlich nicht möglich, da der Punkt so ähnlich zu einem typischen Klassenvertreter einer falschen Klasse ist. Wenn man die Verbindungsstrecke von $\mathbf{w}^i$ nach $\mathbf{w}^j$ betrachtet, dann beeinhaltet das Fenster alle Punkte, die mehr als den Anteil $(1-v)/2$ der Strecke von $\mathbf{w}^i$ und $\mathbf{w}^j$ entfernt sind. Geht man von der Strecke weg, dann findet man einen sich allmählich verbreiternden Bereich, in den das $\mathbf{x}$ fallen darf.

- **LVQ3**: Dieses Verfahren kombiniert das Feintuning der Klassengrenzen von LVQ2.1 mit einem korrekten Adaptieren der Neuronen innerhalb einer Klasse. Sofern die beiden Gewinner in dieselbe Klasse fallen und diese korrekt ist, werden beide durch

$$\triangle \mathbf{w}^i = \epsilon\alpha(\mathbf{x} - \mathbf{w})$$

mit $\epsilon \in [0.1, 0.5]$ der Eingabe $\mathbf{x}$ ähnlicher gemacht. Es ist allerdings immer noch so, daß ein falsch klassifiziertes Pattern einen sehr nahen Codebookvektor nicht abstößt.

Kohonen selber hat die beiden letzteren Verfahren in Kombination mit dem sehr schnell konvergierenden OLVQ erfolgreich in verschiedenen Projekten eingesetzt.

7.3 Self Organizing Maps

Selbstorganisierende Karten (SOMs) sind eine von Kohonen vorgeschlagene Erweiterung von Wettbewerbslernen, bei der zusätzlich eine Nachbarschaftsbeziehung auf den Neuronen gegeben ist. Die Neuronen sollen einerseits die gegebenen Daten gut repräsentieren, andererseits aber auch die gegebene Nachbarschaftsstruktur der Neuronen erhalten. Dieses führt letztendlich zu einer Kartierung des Eingaberaumes, wo man nicht nur die Codebookvektoren finden kann, sondern auch durch die zwischen den Codebookvektoren gegebenen Nachbarschaften auch die Möglichkeit zur Naviagtion besitzt.

Seien also Neuronen mit Gewichten $\mathbf{w}^1, \ldots, \mathbf{w}^N$ gegeben, sowie eine Nachbarschaftsfunktion $D(i,j)$. Möglichkeiten sind etwa

- eine Listenanordnung, d.h. $D(i,j) = |i-j|$,
- eine Anordnung als Ring, d.h. $D(i,j) = \min(|i-j|, N-1-|i-j|)$,
- eine Anordnung als Rechteckgitter, d.h. $i = (j,k)$, $D((j_1,k_1),(j_2,k_2)) = |j_1-j_2|+|k_1-k_2|$,
- eine Anordnung als Gitter in einem höherdimensionalen Raum, als unregelmäßiges Gitter, ...

Bei auf Gittern liegenden Neuronen kann als Abstandsmessung zwischen zwei Neuronen die Länge eines Pfades zwischen den Neuronen genommen werden. Alternativ ist eine Anordnung der Neuronen im Raum und D als der euklidische Abstand denkbar.

Die Gewichte der Neuronen werden jetzt zufällig (oder als je ein zufälliges Pattern) initialisiert und anschließend wie bei Vektorquantisierung an die Daten adapatiert, wobei jedoch die Nachbarschaften berücksichtigt werden: Mit dem Gewinner werden auch die Neuronen in einer Nachbarschaft in die Richtung der jeweiligen Eingabe gezogen. Genauer:

Wiederhole
  für eine Eingabe $\mathbf{x}$ berechne $|\mathbf{x} - \mathbf{w}^i|$,
  ermittle den Gewinner i_0,
  $\triangle \mathbf{w}^i = \eta \cdot f(\sigma, D(i,i_0))(\mathbf{x} - \mathbf{w}^i)$,
$\eta := \eta \cdot \alpha_1; \quad \sigma := \sigma \cdot \alpha_2$

mit der Lernrate η, die z.B. mit einer Zahl aus $[1,3]$ initialisiert wird und nach jedem Schritt um den Faktor $\alpha_1 \approx 0.999$ verkleinert wird, einem Term σ, der die Bewertung der Nachbarschaft beeinflußt und etwa mit $\sqrt{N}$ initialisiert und in jedem Schritt um einen Faktor $\alpha \approx 0.999$ verkleinert wird, und einer mit $\sigma \to 0$ und $d = D(i,i_0) \to \infty$ abfallenden Funktion f, die die Änderung der Gewichte entsprechend der Nähe zum Gewinner skaliert, z.B.

$$f(\sigma, d) = \mathrm{e}^{-d^2/(2\sigma^2)}.$$

Die Skalierung von σ und η sorgt dafür, daß das Verfahren konvergiert. Am Anfang werden relativ große Nachbarschaften mit berücksichtigt und die Neuronen sehr geändert, gegen Ende des Verfahrens werden hauptsächlich nur noch die einzelnen Neuronen mit relativ kleinen Änderungen adaptiert.

Die Topologie sorgt dafür, daß sich die Neuronen mit ihrer vorgegebenen Verknüpfungsstruktur auf den Daten quasi ausbreiten. Wozu ist diese zusätzliche Nachbarschaftsbeziehung im Vergleich zu normaler Vektorquantisierung gut? Es folgen einige Punkte möglicher Anwendungen:

- SOMs können aufgrund ihrere einfachen und plausiblen Lernregel zur Erklärung biologischer Phänomene herangezogen werden. Man kann z.B. nachweisen, daß der sensorische Kortex eine topologische Abbildung der entsprechenden Bereiche der Sinneswahrnehmungen ist.

- Genau wie LVQ können auch SOMs zur Klassifikation oder Funktionsapproximation verwandt werden. Dazu wird auf die Daten trainiert und anschließend die Abbildung **kalibriert**. Kalibrieren bedeutet, den Neuronen des Netzes je einen Wert zuzuordnen, so daß eine Abbildung oder Klassifikation realisiert werden kann; als Wert bietet sich etwa der Wert des zum Gewichtsvektor des Neurons nächsten Trainingsmuster oder eine geeignete Mittelung über die durch die Daten, für die diese Neuron Gewinner ist, gegebenen Werte an. Anschließend wird ein Eingabedatum wird auf den Wert, der dem für die Eingabe zuständigen Gewinnerneuron zugeordnet ist, abgebildet. Man erhält also eine Funktion.

  Dieses Verfahren ist natürlich auch möglich, ohne eine explizite Topologieerhaltung zu implementieren. Eine Topologieerhaltung kann jedoch Vorteile haben, wenn man stetige Funktionen auf diese Weise approximieren will. Nahe beieinanderliegende Neuronen haben ähnliche Funktionswerte, so daß eine höhere Fehlertoleranz sichergestellt ist. Ohne Topologieerhaltung wäre es möglich, daß die Einzugsgebiete zweier Vektoren mit völlig unterschiedlichem Funktionswert aneinanderstoßen.

- Man kann ein euklidisches TSP mit einem eindimensionalen rinförmigen Kohonennetz angehen. Die Neuronen werden z.B. mit 0 initialisiert und anschließend mit den Koordinaten der Städte trainiert. Wählt man etwa gleich viele Neuronen wie Städte, dann wird tendentiell am Ende jedes Neuron für genau eine Stadt Gewinner sein. Eine Reihenfolge der Städte erhält man aus der Anordnung dieser Neuronen. Die Topologieerhaltung wirkt dabei dahingehend, daß kurze Touren bevorzugt werden, denn in kurzen Touren sind die Koordinaten benachbarter Städte ähnlicher, entsprechen also der Topologie des Netzes.

- SOMs können Teilaspekte hochdimensionaler Daten visualisieren, etwa eine zweidimensionale Kohonenabbildung läßt Nachbarschaften zwischen den Datenpunkten erkennen. Angewandt wird dieses z.B. in semantischen Netzen zur Erkennung von Synonymen und Klassifikation der Wörter: Die in einigen Texten auftretenden Wörter werden je durch Zufallsvektoren, sog. **Fingerprints**, codiert und anschließend ein SOM mit Tripelkontexten, d.h. den Vektoren, die drei aufeinanderfolgenden Wörtern in den Texten entsprechen, trainiert. Da etwa Synonyme häufig im gleichen Kontext auftauchen, besitzen sie höchstwahrscheinlich dasselbe Gewinnerneuron. in der Bedeutung verwandte Wörter befinden sich in der unmittelbaren Nachbarschaft. Ein einzelnes Wort kann auf dieser semantischen Abbildung wiedergefunden werden, indem man den Fingerprint des Wortes mit zwei Zufallsvektoren als Kontext vervollständigt.

  In der Arbeitsgruppe von Kohonen wurde diese Methode eingesetzt, um das ein Dokument beschreibende Worthistogramm in der Dimension zu reduzieren: Anstelle aller (wesentlichen) auftretenden Wörter wird ein Dokument durch das sich in der Semantischen Abbildung ergebende Neuronenhistogramm repräsentiert. Diese die Dokumente beschreibenden Vektoren dienen im sog. WEBSOM als Eingabe eines weiteren zweidimensionalen Kohonennetzes, das somit Dokumente gemäß ihrer Ähnlichkeit anordnet. Das dabei entstehende Netz kann dazu dienen, auch auf natürlichsprachige Fragen nach Dokumenten zu antworten: Die Fragen werden einfach der semantischen Abbildung gemäß in Histogramme umgewandelt, und letztere dienen als Eingabe an das zweite SOM. Die Topologie des Netzes erlaubt, nicht nur den optimalen Knoten, sondern auch die Umgebung gezielt zu präsentieren, so daß der Fragesteller gezielte Information erhalten kann. Dieses ist insbesondere dann interessant, wenn er nach inhaltlich verschiedenen Ausprägungen des Gebietes sucht, die in der Regel in unterschiedlichen benachbarten Knoten gespeichert sind.

7.4 Hybride Architekturen

In hybiden Architekturen werden selbstorganisierende Karten oder Vektorquantisierungen zusammen mit ebenfalls trainierbaren vorwärtsgerichteten Netzen eingesetzt. Das hat den Vorteil, daß durch die lokale, selbstorganisierende Schicht zunächst die Daten auf für den Menschen einsichtige Weise vorverarbeitet werden: Sie werden etwa mit einem ähnlichen Vektor identifiziert; das Einfügen neuer Daten ist durch Einfügen neuer Referenzvektoren inkrementell und schnell möglich. Anschließend ermöglicht eine flexible Verarbeitung die Approximation auch komplexer Abbildungen.

Counterpropagation kombiniert eine selbstorganisierende Winner-Takes-All Schicht mit einer linearen vorwärtsgerichteten Schicht. Ein Counterpropagation-Netz berechnet also eine Abbildung

$$\mathbf{x} \mapsto (W_{ji})_j \text{ mit } i \,:\, |\mathbf{x} - \mathbf{w}_i| \text{ minimal}\,.$$

Zunächst wird also der zur Eingabe ähnlichste Vektor $\mathbf{w}_i$ bestimmt. Das entsprechende verborgene Neuron berechnet die Ausgabe 1, alle anderen die Ausgabe 0. Anschließend wird die Ausgabe $(0, \ldots, 0, 1, 0, \ldots, 0)$ der verborgenen Schicht durch lineare Neuronen mit Gewichten $\mathbf{W}$ weiterverarbeitet. Ausgfrund der unären Aktivierung entspricht die Ausgabe dem Vektor der i ten Komponenten der Gewichte $\mathbf{W}_j$. Die erste Schicht nach der Eingabeschicht heißt auch **Kohonenschicht**, die zweite Schicht **Grossbergschicht**.

Training der Kohonenschicht erfolgt etwa mit Vektorquantisierung, d.h.

$$\triangle w_{ij} = \eta(x_j - w_{ij}) \text{ falls } i \text{ der Gewinner ist.}$$

Dabei wird die Lernrate $\eta > 0$ häufig über die Zeit hinweg verkleinert, um Konvergenz zu erzwingen. Alternativ kann eine Nachbarschaft zwischen den Neuronen eingeführt und gemäß Kohonens SOM trainiert werden, so daß eine glattere, topologieerhaltende Kartierung des Raumes durch die Kohonenschicht gegeben ist. Dabei werden die Gewichte initial zufällig, zufällig als ein Trainingsmuster oder durch $1/\sqrt{n}$, $n =$ Eingabedimension, bestimmt. Um zu verhindern, daß die Neuronen der Kohonenschicht sich nicht gleichmäßig auf die Trainingspunkte einstellen, kann man in letzterem Fall die Eingabedaten initial auf

$$\alpha x_i + (1-\alpha)/\sqrt{n}$$

mit $\alpha > 0$ verändern. Läßt man α gegen 1 streben, erhält man die ursprünglichen Trainingsdaten zurück. Da sie sich aber anfänglich für kleines α nahe der Vektoren der Kohonenschicht befinden, stellen diese sich optimal auf die Trainingsdaten ein.

Die Großbergschicht wird etwa mit der sich durch Gradientenabstieg auf dem quadratischen Fehler ergebenden Regel

$$\triangle W_{ij} = -\eta(W_{ij} - y_j)o_i$$

trainiert, wobei die Lernrate η über die Zeit gegen Null geht, um Konvergenz zu erzwingen. $\mathbf{y}$ stellt die gewünschte Ausgabe dar und o_i ist 1 für das Gewinnerneuron bei Eingabe des zugehörigen $\mathbf{x}$, für andere Neuronen ist es 0. Stattdessen kann man die Gewichte auch in einem Schritt als

$$W_{ij} = \sum_{i \text{ ist Gewinner für } \mathbf{x}} y_j/N$$

mit $N =$ Anzahl der Eingaben $\mathbf{x}$, für die i Gewinner ist, bestimmen. Dieses minimiert den linearen Ausgabefehler.

Üblicherweise wird ein Counterpropagationnetz auf die Identität auf den Daten $(\mathbf{x}, f(\mathbf{x}))$ mit der zu lernenden Funktion f trainiert. Das hat den Vorteil, daß gleichzeitig die Funktion f als auch eine mögliche Invertierung der Funktion gelernt werden: Eine Eingabe von $(\mathbf{x}, 0)$ führt zu der bei Eingabe von $(\mathbf{x}, f(\mathbf{x}))$ gelernten Ausgabe, da die Eingabe der Eingabe $(\mathbf{x}, f(\mathbf{x}))$ am ähnlichsten ist, wenn man die Trainingsmenge betrachtet. Analog führt eine Eingabe von $(0, f(\mathbf{x}))$ zur selben Ausgabe wie $(\mathbf{x}, f(\mathbf{x}))$.

Zufügen von neuen Mustern ist inkrementell möglich: Man fügt, sofern das Muster $\mathbf{x}$ noch nicht zufriedenstellend abgedeckt ist, ein Neuron mit Gewicht $\mathbf{x}$ zur Kohonenschicht zu; die von diesem Neuron ausgehenden Gewichte werden identisch zur gewünschten Ausgabe gewählt. Dieses ermöglicht, das neue Muster (im schlimmsten Fall: auswendig) zu lernen, ohne die bisherige Funktionalität zu zerstören.

Eine andere Möglichkeit, eine lokale, leicht interpretierbare Schicht zusammen mit einer interpolierenden vorwärtsgerichteten Schicht zu verschalten, stellen **Radiale Basisfunktionen Netze**, kurz RBF-Netze dar. Ein RBF-Netz berechnet eine Funktion

$$\mathbf{x} \mapsto \sum w_i h_i(|\mathbf{x} - \mathbf{x}_i|)$$

mit Funktionen h_i, die häufig als Gaußfunktionen $h_i(x) = \mathrm{e}^{-x^2/\sigma_i^2}$ gewählt werden. Andere Funktionen mit $h(x) \to 0$ für $x \to \infty$ sind aber denkbar, etwa $h_i(x) = \sqrt{x^2 + \sigma_i}$. Das heißt, in der ersten Schicht wird der Abstand der Eingabe zu Referenzvektoren gemessen. Je nach Abstand ergibt sich eine Aktivierung der Neuronen der verborgenen Schicht, die je nach Abstand zu den Referenzvektoren niedrige oder hohe Werte aufweist. Die gewichtet aufsummierte Aktivierung liefert die Ausgabe des Netzes. Auch hier ist die erste Schicht lokal, da die Referenzvektoren je die Ausgabe für Eingaben in ihrere Nähe festlegen. RBF-Netze werden im Vergleich zu feedfoward Netzen häufig eingesetzt, wenn relativ wenige Daten vorliegen. Anhand der Aktivierung der verborgenen Schicht kann man hier leicht feststellen, ob eine Eingabe im relevanten Bereich liegt.

Zum Training werden die Zentren $\mathbf{x}_i$ entweder identisch zu einigen (allen) Trainingsmuster gewählt, oder durch einen selbstorganisierenden Mechanismus, etwa VQ oder SOM, trainiert. Der Glättungsparameter σ_i wird häufig für alle Neuronen identisch im Fall von Gaußfunktionen auf 1 oder $1/$ Anzahl der hidden Neuronen gesetzt. Die Ausgabegewichte können prinzipiell über einen Gradientenabstieg bestimmt werden, so daß man dieselben Formeln (mit unterschiedlicher Ausgabe der verborgenen Schicht o_i) wie bei Counterpropagation erhält. Möchte man die Augabegewichte in einem Schritt bestimmen, so daß der quadratische Fehler möglichst klein ist, dann kann man die Gewichtsmatrix unter gewissen Bedingungen durch

$$\mathbf{W} = (H^t H)^{-1} H Y$$

mit der die Ausgaben aufsammelnden Matrix Y und der Matrix

$$H = \begin{vmatrix} h_1(|\mathbf{x}_1 - \mathbf{w}_1|) & \cdots & h_k(|\mathbf{x}_1 - \mathbf{w}_k|) \\ \vdots & & \vdots \\ h_1(|\mathbf{x}_m - \mathbf{w}_1|) & \cdots & h_k(|\mathbf{x}_m - \mathbf{w}_k|) \end{vmatrix}$$

mit m = Anzahl der Beispiele und k = Anzahl der hidden Neuronen wählen. Die Matrix $(H^t H)^{-1} H$ stellt dabei die sogenannte Pseudoinverse der Matrix H dar und ersetzt die Matrixinversion im Gleichungssystem $H \cdot \mathbf{W} = Y$, das bei einer exakten Interpolation der Ausgaben durch die Gewichte $\mathbf{W}$ gelöst würde. Anstelle der Matrix $(H^t H)^{-1} H$ kann man auch die Matrix $((H^t H + \lambda \tilde{H})^{-1} H$ mit

$$\tilde{H} = \begin{vmatrix} h_1(|\mathbf{w}_1 - \mathbf{w}_1|) & \cdots & h_k(|\mathbf{w}_1 - \mathbf{w}_k|) \\ \vdots & & \vdots \\ h_1(|\mathbf{w}_m - \mathbf{w}_1|) & \cdots & h_k(|\mathbf{w}_m - \mathbf{w}_k|) \end{vmatrix}$$

verwenden. Je nach Wahl von $\lambda \geq 0$ entspricht das einer Regularisierung der Lösung, d.h. die Ausgabefunktion wird evtl. auf Kosten des empirischen Fehlers glatter, so daß eine bessere Generalisierung gewährleistet ist.

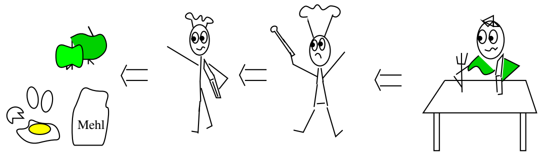
Nachdem die Gewichte in dieser Weise direkt eingestellt sind, kann Nachtraining sinnvoll erscheinen. Dieses kann sich auf alle vorliegenden Parameter, d.h. die Zentren, die Parameter σ_i und die Ausgabegewichte beziehen. In allen Fällen geschieht Nachtraining etwa durch einen Gadientenabstieg auf dem quadratischen Fehler. Zufügen von Daten ist durch Einfügen eines neuen verborgenen Neurons mit entsprechendem Zentrum möglich. Die Ausgabegewichte werden identisch zur gewünschten Ausgabe gewählt und evtl. durch Nachtraining kurz adaptiert.

Tja

Das wär's für's erste. Bleiben noch die Aspekte zu erwähnen, die nicht behandelt wurden:

- Für die Zeitreihenverarbeitung stellen **Time Delay Netze** (TDNN) eine Alternative dar. Sie ermöglichen durch geeignetes weight sharing und eine automatische Sequenzenbehandlung, lokale Merkmale unabhängig von der Lage bzgl. der Zeit und der Länge effizient zu extrahieren und zu verarbeiten.

- Speziell für die Schriftzeichenerkennung entworfen wurde das **Neocognitron**, das sukzessive lokale Merkmale der Schrift translationsinvariant verarbeitet.
- **Probabilistische Neuronale Netze** sind eine spezielle Implementierung, die vermöge des Bayes Ansatz und einer Schätzung der beteiligten Dichten vermöge Fensterfunktionen verarbeiten.
- Eine Kombination von feedforward Netzen und logischen Regeln ist möglich, insbesondere gibt es effiziente neuronale Implementierungen von Fuzzy-Reglern zu verschiedenen **Neuro-Fuzzy-Modellen**.
- Im Vergleich zu Hopfieldnetzen wird Assoziation, indem man nur einmal schaltet, im **bidirektionalen Assoziativspeicher** (BAM) implementiert.
- Verglichen zur PCA extrahiert die **Independent Component Analysis** (ICA) stochastisch unabhängige Komponenten und dient so der Separierung von Signalen aus einem gemischten Signal. Die **kernel PCA** extrahiert nichtlineare Signale, indem sie sich den Kerneltrick der SVM zunutze macht und Skalarprodukte durch Skalarprodukte der nichtlinear in einen hochdimensionalen Raum abgebildeten Daten ersetzt.
- Biologienahe Modelle, die Selbstorganisation, Gedächtnisleistungen und das Stabilitäts-/Plastizitätslemma angehen sind die im Rahmen der **Adaptive Resonanztheorie** (ART) vorgeschlagenen Varianten.
- Biologienahe Modelle, die Neuronen zusammen mit deren zeitlicher Entwicklung modellieren, sind **spiking Netze**.
- ... und noch vieles mehr.



"Back-Propagation"