Clusteranalyse mit Hilfe neuronaler Netze

– eine Anwendung selbstorganisierender Netze auf Klimadaten –

Winfried Beer

Diplomarbeit

Fachhochschule München
Fachbereich Informatik

Prüfer	:	Prof. Dr. K. Köhler
Zweitprüfer	:	Prof. Dr. P. Haberäcker
Betreuer und Aufgabensteller	:	Priv.–Doz. Dr. R. Sausen DLR, Institut für Physik der Atmosphäre, Weßling
Abgabetermin	:	11. August 1995

Die Maschinen werden dann zum Leben erwachen,
wenn sie über sich selbst nachdenken,
und der Mensch nicht weiß, warum.

Winfried Beer

Inhaltsverzeichnis

Kapitel 1

Einführung

Ziel dieser Diplomarbeit ist die Bereitstellung und Anwendung eines Verfahrens für die Clusteranalyse, das auf der Theorie der *neuronalen Netze* basiert.

Für diesen Zweck entstand das Programmpaket *Sonnet* zur Simulation von „selbstorganisierenden" neuronalen Netzen, insbesondere von *Kohonen-Netzen* und einer speziellen Variante, dem *neuronalen Gas*. Aufgrund der Eigenschaften dieser neuronalen Netze können sie zur Aufteilung von Datenvektoren in verschiedene Klassen verwendet werden.

Die konkrete Anwendung erfolgt im Bereich Klimaforschung, nämlich zur Generierung von *Wetterklassen*. Dies ist nur ein Zwischenschritt in einem größeren Projekt, der *Regionalisierung globaler Klimamodelle*. Die neuronalen Netze erweisen sich für diese Problemstellung als sehr gut geeignet.

Diese Diplomarbeit ist in zwei Teile aufgeteilt. Im ersten Teil erörtere ich die Grundlagen neuronaler Netze und statistischer Clusteranalyseverfahren. Für die Clusteranalyse mit neuronalen Netzen stelle ich anschließend das Modell des Kohonen-Netzes und seine Erweiterungen vor.

Im zweiten Teil erläutere ich die praktische Anwendung dieser selbstorganisierenden neuronalen Netze. Nach der Beschreibung des *Regionalisierungsproblems* und des Simulators *Sonnet* werden an einfachen, niederdimensionalen Fällen die Probleme und Auswirkungen der einstellbaren Parameter ausführlich demonstriert. Auf die Schwierigkeiten, die sich bei höher- und hochdimensionalen Daten ergeben, wird nachfolgend hingewiesen.

Mit diesen Informationen sollte es möglich sein, das Programmpaket *Sonnet* als Werkzeug zur Clusteranalyse auch für andere Aufgabenstellungen einzusetzen. Man sollte jedoch generell prüfen, ob klassische Clusteranalyseverfahren geeignetere Ergebnisse liefern (insbesondere für Auswertungen in der Statistik). Das Anwendungsziel, das Erzeugen von guten Wetterklassen, konnte nur in Teilbereichen konkret durchgeführt und analysiert werden.

Teil I

Die Theorie

Kapitel 2

Was ist ein neuronales Netz?

In den nachfolgenden Abschnitten stelle ich kurz die prinzipiellen Komponenten eines künstlichen neuronalen Netzes vor. Einen tieferen Einblick in die Theorie der neuronalen Netze bietet weiterführende Literatur wie [Zell 1994] oder [Brause 1991].

2.1 Allgemeines

Ein (künstliches) neuronales Netz ist ein mathematisches Modell, welches gewisse Ähnlichkeiten mit der Struktur und der Funktionsweise eines menschliches Gehirns aufweist. Erste Arbeiten dazu gab es schon 1943 [McClelland/Pitts 1943]. Nach einer Ruhepause haben die neuronalen Netze nun in den letzten Jahren einen starken Aufschwung erlebt, insbesondere durch den drastischen Anstieg der Rechenleistung der Computer.

Zwischen einem neuronalem Netz und dem menschlichem Gehirn bestehen Gemeinsamkeiten:

- große Anzahl einfacher Einheiten (Zellen, Neuronen),
- Kommunikation zwischen Neuronen über einfache Signale.

Baut man nun ein künstliches neuronales Netz oder simuliert es auf einem Computer, kann man damit verschiedene Probleme lösen, z. B. Optimierungsaufgaben, Prognosen von Börsenkursen, Mustererkennung, Steuerung von Robotern oder die Simulation intelligenten Verhaltens. All dies könnte auch mit Hilfe normaler Computer und traditioneller Rechenalgorithmen im Prinzip gelöst werden. Mit neuronalen Netzen lassen sich diese Probleme aber manchmal einfacher beschreiben und lösen.

Folgenden Eigenschaften kommen den neuronalen Netzen zugute:

- Wissenserwerb durch Lernen an Beispielen,
- Möglichkeit einer massiven Parallelverarbeitung,
- Robustheit gegenüber (kleineren) Ausfällen von Neuronen.

2.2 Die Neuronen

Alle neuronalen Netze sind aus einfachen Bausteinen aufgebaut, den Neuronen. Diese sind meist vom gleichen Grundtyp. Die Funktionsweise eines Neurons ist in Abbildung 2.1 dargestellt. Das Neuron wird aktiv und sendet ein Ausgabesignal, wenn die gewichtete Summe der Eingangssignale einen bestimmten Schwellwert überschreitet.

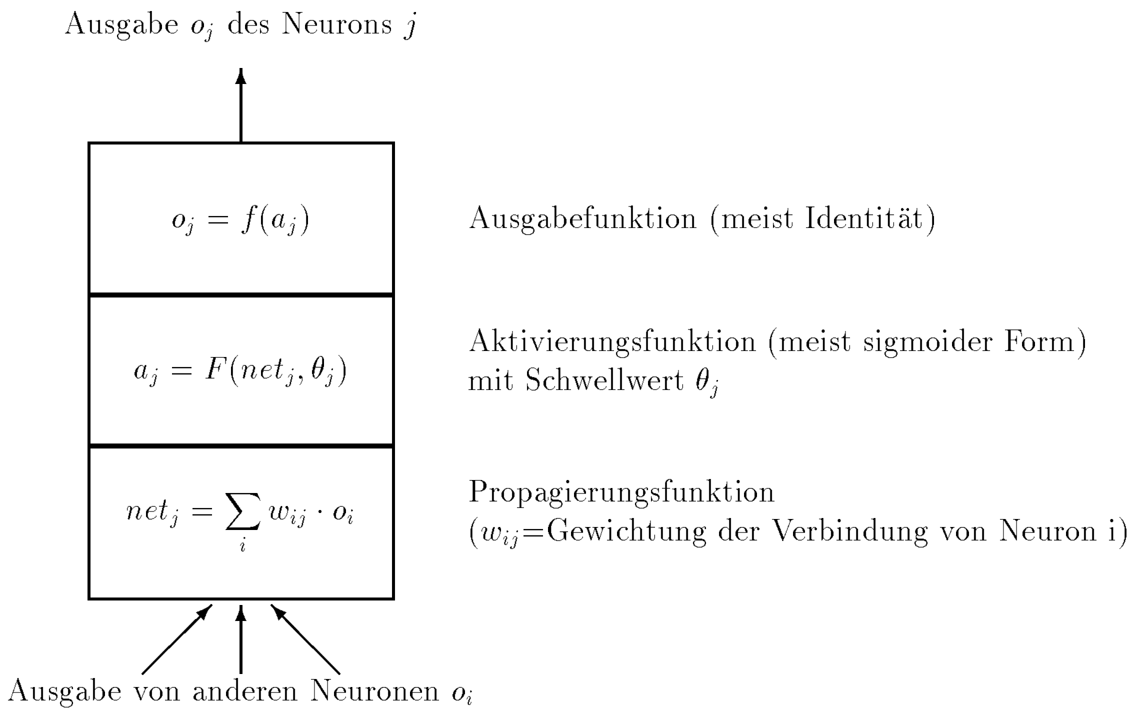


Abbildung 2.1: Aufbau eines künstlichen Neurons.

2.3 Die Netztopologie

Die Neuronen werden miteinander verknüpft, d. h. die Ausgabe eines Neurons wird zur Eingabe anderer Neuronen.[1] Diese Verknüpfung nennt man Netztopologie. Dadurch lassen sich die neuronalen Netze in verschiedene Kategorien einteilen, z. B. schichtweise Vernetzung, rekursive Strukturen oder Totalvernetzung (Prinzipzeichnungen siehe Abbildung 2.2). Durch die Vernetzung werden auch die Schnittstellen des Netzes definiert, d. h. die Ein- und Ausgabe des gesamten Netzes.

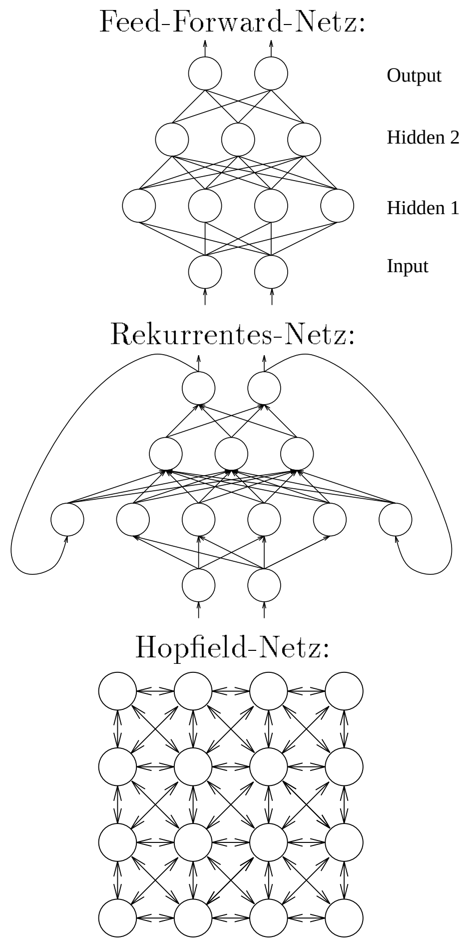


Abbildung 2.2: Beispiele neuronaler Netztopologien.

[1] Im Normalfall gibt es nur ein einziges Ausgabesignal je Neuron, das jedoch zu mehreren anderen Neuronen als Eingabe geführt werden kann.

2.4 Die Handhabung

Meistens verlangt man von einem neuronalen Netz, daß es zu einem bestimmten Eingabemuster ein bestimmtes Ausgabemuster liefert, also mathematisch betrachtet, eine mehrdimensionale Abbildung emuliert. Diese Abbildung ist aber meist nur durch Beispielfälle beschrieben, die zudem widersprüchlich oder ungenau sein können.

Um dieses Verhalten zu erreichen, wird das Netz mit den Beispielfällen trainiert. Dies bedeutet, daß die variablen Parameter der Neuronen (meist nur die Gewichtungsfaktoren) und evtl. die Netztopologie so lange schrittweise geändert werden, bis die Ausgabemuster vom Netz und die gewünschten Ausgabemuster genügend gut übereinstimmen. Dieses Training wird durch „Lernregeln“ beschrieben.

Nachdem man das Netz trainiert hat, hofft man, sowohl für trainierte wie auch für nicht trainierte Eingabemuster „sinnvolle“ Ausgabemuster zu erhalten.[2]

Etwas anders verwendet man neuronale Netze, bei denen man keine expliziten Ausgabemuster vorgibt. Bei diesen wird in der Trainingsphase das Netz nur nach einem Fehlerkriterium geändert, welches von den Aktivierungszuständen der Neuronen abhängt.[3] Diese Netze lassen sich gut für Optimierungsaufgaben einsetzen.

[2] Diese Netze werden „überwacht lernende“ Netze genannt, da das Netz eine vorgegebene Ausgabe liefern soll. Ein typischer Vertreter für diesen Netztyp ist das Backpropagation-Netz.

[3] „Unüberwacht lernende“ Netze oder selbstorganisierende Netze, z. B. Kohonen-Netze.

Kapitel 3

Was ist Clusteranalyse?

Die Theorie der „Clusteranalyse“ ist ein Teilgebiet der Statistik. In diesem Kapitel wird dieses Themengebiet umrissen und an Hand zweier Algorithmen versucht, die Lösungsstrategien der klassischen Statistik zu verdeutlichen.[1] Die eingeführten mathematischen Formalismen und Variablen sind auch die Basis für die weiteren Kapitel.

3.1 Definition Clusteranalyse

Die Clusteranalyse ist ein algorithmisches Verfahren zur *objektiven Klassifikation* von N Objekten, die durch λ Merkmale beschrieben werden, in K *a priori nicht definierte* Klassen, wobei *ähnliche* Objekte in gleiche, unähnliche Objekte in verschiedene Klassen eingeteilt werden sollen [Fuentes 1994].

Mit dem Begriff „Cluster“ wird normalerweise eine Punktwolke bezeichnet, die von anderen Punktwolken räumlich gut getrennt ist. Eine „Klasse“ ist eine Menge von Objekten, welche ein bestimmtes Kriterium erfüllen.

In der Literatur wird manchmal zwischen den Begriffen „Clusteranalyse“ und „Clusterformation“ unterschieden [Späth 1983]. Ersteres ist eine Einteilung von Objekten in *natürliche* Klassen, d. h. in gut abgegrenzte *Punktwolken* (ein gutes Beispiel dafür ist die Abbildung 3.1, links). Clusterformation ist hingegen die willkürliche Einteilung einer Objektmenge in *künstliche* Klassen (Abbildung 3.1, rechts). Diese Unterscheidung wird hier nicht getroffen, sondern nur der Begriff Clusteranalyse verwendet, wenn auch meistens „Clusterformation“ treffender wäre.

[1] Für Details siehe z. B. [Späth 1983] und [Bacher 1994].

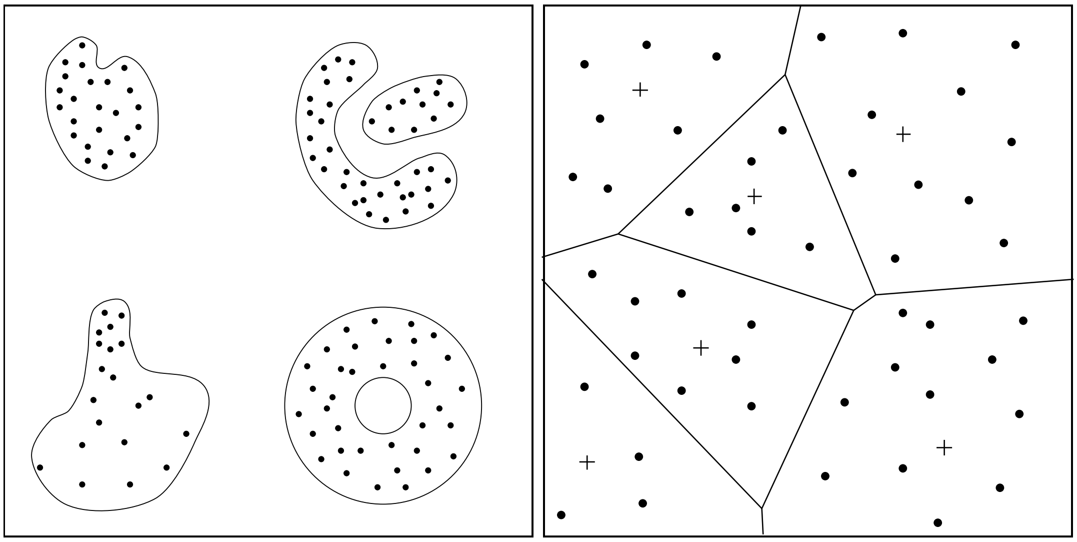


Abbildung 3.1: Beispiele einer Klasseneinteilung.
Im linken Bild ist eine Einteilung der Objekte in *natürliche* Klassen, im rechten Bild in *künstliche* Klassen dargestellt.

3.2 Klassenbildung

Folgende Schritte bilden das Grundprinzip jeder Klassenbildung:

- Gegeben sind N Objekte, die in einem Zustandsraum liegen:
  Jedes Objekt wird mathematisch durch einen Vektor der Dimensionalität λ realisiert — der Zustandsraum ist also λ-dimensional. Jedes Objekt bildet einen Punkt in diesem Raum. Jede Komponente dieses Vektors beschreibt eine Eigenschaft des Objekts als Zahlenwert. Dieser Wert kann kontinuierlich (z. B. Temperatur; Gewicht) oder diskret (ganze Zahl; 0 oder 1) sein.

  Man kann diese Objekte als endliche Stichprobe einer statistischen Verteilung betrachten.

- Definiere Maß für Ähnlichkeit:
  Es wird zwischen zwei Objekten oder zwischen zwei Objektmengen ein Maß für die Ähnlichkeit festgelegt. Meist benutzt man die euklidische Distanz zwischen den zwei Objekten. Bei Objektmengen kann man den kleinsten oder größten Abstand aller Objektpaare der einen und der anderen Menge verwenden. Zudem sollte man eine Skalierung der einzelnen Dimensionen vornehmen, um eine Vergleichbarkeit der Zahlen zu erreichen. Diese Skalierung hat einen entscheidenden Einfluß auf die Einteilung der Objekte in Klassen.

Aus diesem Ähnlichkeitsmaß läßt sich ein Fehlermaß ableiten, welches minimiert werden soll.

- Wähle ein Verfahren:
  Das Verfahren gibt einen Weg zur Minimierung des Fehlers vor. Es kann jedoch i. a. nur ein lokales Minimum gefunden werden. Zu berücksichtigen sind auch Nebenbedingungen wie z. B. „gleiche Anzahl von Objekten je Klasse“.

Ergebnis: K-Partition[2] der Objektmenge
Die N Objekte sind in K Klassen eingeteilt. Die Größe und Form einer Klasse ist durch ihre zugeordneten Objekte festgelegt. Bei vielen Algorithmen reicht aber der Schwerpunkt der Mitglieder oder ein „Zentrumsvektor“ der Klasse als Beschreibung der Klasse aus.

3.3 Klassische Clusteranalyseverfahren

Die klassischen Clusteranalyseverfahren lassen sich im Prinzip in zwei verschiedene Kategorien einteilen, den *hierarchischen* und den *partitionierenden* Verfahren.

1. Hierarchische Verfahren:

   (a) Initialisierung:
   Bilde zu jedem Objekt eine eigene Klasse, d. h. es gibt N Klassen.

   (b) Vereinigung von zwei ähnlichen Klassen:
   Zwischen allen Klassen werden die Ähnlichkeiten bestimmt. Die beiden Klassen mit der größten Ähnlichkeit werden zu einer einzigen Klasse fusioniert.

   Wiederhole (b) bis nur noch K Klassen übrig sind. (K-Partition)

   Diese Verfahren haben den Nachteil, daß sie für große N viel Rechenzeit oder Speicher benötigen.

2. Partitionierende Verfahren:

   (a) Initialisierung:
   Man teilt die Objekte in K Klassen ein. Dabei kann man schnelle heuristische Verfahren verwenden oder eine eher zufällige Einteilung vornehmen.

[2] Eine K-Partition ist die Einteilung aller Objekte in genau K Klassen.

(b) Austausch von Objekten zwischen den Klassen:
Nun werden zwischen den Klassen ein oder mehrere Objekte gewechselt, so daß die neue Klasseneinteilung besser als die alte ist. „Besser“ bezieht sich hier auf eine Fehlerfunktion.

Wiederhole (b) bis keine Verbesserung erfolgt.

Diese Verfahren sind nicht so rechen- oder speicherintensiv wie die hierarchischen Verfahren. Sie reagieren jedoch sehr sensitiv auf die Gestalt der Ausgangspartition.

3.4 Gütemaß: Fehlerfunktion

Ein gebräuchliches Maß für die Güte einer Partition ist die sog. „Streuungsquadratsumme“. Sie ist die quadratische Summe der Abstände der Objekte zum jeweiligen Zentrum. Das Zentrum ist im Normalfall der Mittelwert der Objekte in der Klasse. Deswegen spricht man auch vom „Varianzkriterium“.

Anmerkung:
Nicht jedes Clusteranalyseverfahren zielt direkt auf eine Minimierung dieser oder einer anderen Fehlerfunktion ab. Eine Klasseneinteilung kann auch nach heuristischen Kriterien durchgeführt werden.

3.5 Mathematische Beschreibung

Ich möchte hier kurz anhand des „K-Means“-Verfahren die mathematischen Variablen und Formeln erklären. Das K-Means-Verfahren ist ein partitionierendes Verfahren. Es verwendet eine euklidische Metrik, basiert auf Zentrumsvektoren und minimiert schrittweise die Streuungsquadratsumme. Wie sich später zeigt, ist es einem Spezialfall der Clusteranalyse mit neuronalen Netzen sehr ähnlich.

Gegeben sind:
N Objekte $\mathbf{x}_i \in \Re^\lambda, i = 1, 2, \ldots, N$
K Klassen, beschrieben durch K Zentren $\mathbf{w}_i \in \Re^\lambda, i = 1, 2, \ldots, K$

Ein Objekt $\mathbf{x}_n$ gehört zur Klasse k ($\mathbf{x}_n \in \mathbf{C}_k$) wenn

$$\|\mathbf{x}_n - \mathbf{w}_k\| = \min_{i=1}^{K}(\|\mathbf{x}_n - \mathbf{w}_i\|). \tag{3.1}$$

Durch diese Zuordnung werden im Zustandsraum die Klassen durch Hyperebenen voneinander getrennt. $\mathbf{C}_k$ ist das Gebiet der Klasse k. Für $\lambda = 2$ spricht man von sog. Voronoj-Polygonen. Die Kanten dieser Polygone liegen auf den Mittelsenkrechten über den Strecken zwischen den Zentren und ähneln einer Bienenwabenstruktur.

Mit diesem Formalismus läßt sich die Streuungsquadratsumme Q_k einer Klasse k angeben als:

$$Q_k = \sum_{\forall \mathbf{x}_i \in \mathbf{C}_k} \|\mathbf{x}_i - \mathbf{w}_k\|^2 \tag{3.2}$$

Für die Streuungsquadratsumme Q der gesamten Partition erhält man:

$$Q = \sum_{k=1}^{K} Q_k = \sum_{k=1}^{K} \sum_{\forall \mathbf{x}_i \in \mathbf{C}_k} \|\mathbf{x}_i - \mathbf{w}_k\|^2 \tag{3.3}$$

Eine notwendige Bedingung für ein Minimum von Q ist, daß die Zentren $\mathbf{w}_k$ gleich den Schwerpunkten der Klassen k sind,[3] also

$$\mathbf{w}_k = \frac{1}{n_k} \sum_{\forall \mathbf{x}_i \in \mathbf{C}_k} \mathbf{x}_i \quad \text{mit } n_k \text{ gleich der Anzahl der Objekte in Klasse } k. \tag{3.4}$$

Daraus ergibt sich das K-Means-Verfahren:

1. Initialisieren der $\mathbf{w}_k$.
2. Bestimme die Klassen $\mathbf{C}_k$ nach Formel (3.1).
3. Setze die Zentren $\mathbf{w}_k$ gleich den Schwerpunkten nach Formel (3.4).
4. Wenn sich dadurch die Klassen $\mathbf{C}_k$ ändern, fahre mit Schritt 2 fort.

Dieser Algorithmus führt zu einer monotonen Abnahme der Streuungsquadratsumme. Er liefert wegen der euklidischen Metrik meist kugel- oder würfelförmige Klassen; längliche Punktwolken werden in mehrere Klassen aufgeteilt.

3.6 Beurteilung des Fehlers

Für die praktische Anwendung ist eine Beurteilung des Fehlers sehr wichtig. Vorteilhaft wäre es, wenn man den Fehler einer optimalen Partition berechnen könnte. Dies ist aber im Normalfall nicht ohne großen Rechenaufwand möglich. Die

[3] Dann sind die Fehler Q_k der einzelnen Klassen absolut minimal und Q hat ein lokales Minimum.

Anzahl aller möglichen Klasseneinteilungen beträgt etwa K^N. Sie steigt also exponentiell mit der Anzahl der Objekte.[4]

Eine gute Abschätzung des Fehlers ist aber zumindest bei folgendem Spezialfall leicht möglich:[5]

> Alle Objekte $\mathbf{x}_i$ können durch die Neuronengewichte $\mathbf{w}_j$ in K gleich große Hyperwürfel eingeteilt werden. Innerhalb dieser Hyperwürfel sind die Objekte gleichverteilt.[6] Dann gilt für den Fehler $Q = Q^{(\mathrm{eq})}$:
>
> $$Q^{(\mathrm{eq})} = \frac{1}{12}\lambda N a^2 \tag{3.5}$$
>
> mit λ = Dimensionalität der Datenvektoren, N = Anzahl Datenvektoren, $a = \sqrt[\lambda]{\frac{\text{Gesamtvolumen}}{K}}$ = Kantenlänge der Hyperwürfel, K = Anzahl Neuronen.

Der Fehler $Q^{(\mathrm{eq})}$ ist normalerweise eine gute obere Schranke für den Fehler einer *optimalen* Partition. Im allgemeinen Fall gilt Gleichung (3.5) eigentlich nicht, da die Bedingung der Gleichverteilung nicht erfüllt ist. Schätzt man jedoch eine entsprechende Kantenlänge a ab, kann man sie trotzdem zur Berechnung eines *Referenzfehlers* verwenden. Insbesondere gilt die Proportionalität des Fehlers Q zu $N \cdot K^{-\frac{2}{\lambda}}$ für große N und K, wenn trotz Änderung von N und K der „Charakter“ der Verteilung von Objekten und Gewichten gleich bleibt.[7]

Ein besonderes Problem ist bei der praktischen Berechnung von $Q^{(\mathrm{eq})}$ die Bestimmung der Kantenlänge a. Einfacher ist es, ein „Gesamtvolumen“ zu ermitteln, in dem sich alle Objekte befinden. Als einfacher Ansatz bietet sich an, das Volumen eines minimalen, alle Objekte umhüllenden Hyperquaders zu bestimmen.

Eine andere Möglichkeit besteht darin, die Streuungsquadratsumme für den Fall $K = 1$ zu berechnen (das einzige Zentrum ist der Mittelwert aller Objekte!) und mittels diesem Wert und der Gleichung (3.5) ein Referenzvolumen zu ermitteln.

---

[4] Betrachtet man jedoch die Eingabevektoren und die Klassenzentren als kontinuierliche Verteilungen ($N \to \infty, K \to \infty$), dann ist die Dichte der Klassenzentren proportional zu $p(\mathbf{x})^{\frac{\lambda}{\lambda+2}}$, wenn $p(\mathbf{x})$ die Dichte der Eingabevektoren ist [Zador 1982].

[5] Herleitung im Anhang A. Sie erweitert sogar die Gültigkeit von Formel (3.5) auf die Fälle, in denen die Abstände aller Objekte zum jeweiligen Klassenzentrum die gleiche Verteilung besitzen, wie wenn alle Objekte in einem λ-dimensionalen Hyperwürfel der Kantenlänge a gleichverteilt wären.

[6] Nur für $N \to \infty$ kann man von einer exakten Gleichverteilung sprechen. Praktisch reicht es aus, wenn die Objekte statistisch gleichverteilt sind.

[7] Wenn man davon absieht, daß für $K \to N$ der Fehler gegen Null geht.

3.7 Generelle Probleme

Bei der Clusteranalyse treten folgende Probleme auf:

- Vergleichbarkeit der Variablen:
  In der Praxis repräsentieren die Komponenten der Objekte $\mathbf{x}_i$ häufig unterschiedliche physikalische Größen. Durch das Ähnlichkeitsmaß werden diese miteinander verknüpft. Wie nun die einzelnen Dimensionen zueinander gewichtet werden, ist von den Daten und der weiteren Verwendung der Klasseneinteilung abhängig. Besitzen die Objekte in einer Dimension eine größere Varianz als in den anderen, wird diese Dimension auch in mehr Klassen eingeteilt die anderen Dimensionen.

  Deswegen ist es oft vorteilhaft, die Objekte in den einzelnen Dimensionen so zu skalieren, daß jede Dimension die Varianz Eins besitzt (Standardnormierung). Dann ist jede Dimension gleich relevant.

  Es gibt aber auch Clusteranalyseverfahren die unabhängig von der Skalierung arbeiten. Man denke nur an gut getrennte Punktwolken. Diese sind bei (fast) jeder Skalierung immer noch gut getrennt. Ein solches skalenunabhängiges Verfahren teilt die Objekte immer in die gleichen Klassen ein.

  Um die Rechenzeit oder den Speicherbedarf zu verkleinern, kann es sinnvoll sein, die Dimensionalität der Objekte mittels Hauptkomponententransformation zu reduzieren. Damit können irrelevante Informationen eliminiert werden. Prinzipiell wird dadurch das Ergebnis nicht beeinflußt, wenn man eine rotations- und translationsinvariante Metrik wählt (wie die euklidische Metrik). Es können sich aber Vorteile durch bessere Skalierungsmöglichkeiten ergeben.

- Klassenanzahl:
  Die Anzahl der Klassen ist nicht immer vorgegeben. Manchmal sucht man nach *natürlichen* Clustern, deren Anzahl sich erst durch die Clusteranalyse ergibt. Bei den hierarchischen Verfahren ergeben sich automatisch Partitionen mit verschiedener Klassenanzahl. Bei den partitionierenden Verfahren muß man verschiedene Durchläufe mit Variationen der Klassenanzahl durchführen. Durch Definition eines Fehlermaßes läßt sich dann zu jeder Partition das Verhältnis des Fehlers zur Anzahl der Klassen bewerten.

- Veranschaulichung hochdimensionaler Vektoren:
  Die Dimensionalität λ kann sehr groß werden. Eine Kontrolle der Klasseneinteilung durch einfaches Zeichnen der Objekte und Klassen ist nur bis drei Dimensionen sinnvoll. Für höhere Dimensionen kann man eine Abbildung durchführen, welche die Verhältnisse des hochdimensionalen

Raumes möglichst gut in zwei Dimensionen nachbildet. Solche Abbildungen werden als „Unvollständige Clusteranalyseverfahren“ (siehe [Bacher 1994], [Sammon 1969]) bezeichnet.

- Wahl des Verfahrens:
  Die Anzahl und Art der Daten sowie das Ziel der Clusteranalyse bestimmen das Verfahren. Manche Verfahren sind wegen des Rechenzeit- und Speicherbedarfs auf kleine Datenmengen beschränkt. Andere Verfahren sind auf bestimmte Metriken festgelegt und erzeugen nicht die für die Aufgabe gewünschte Klasseneinteilung.

- lokales Fehlerminimum:
  Die Clusteranalyseverfahren führen i. a. nicht zu der optimalen Klasseneinteilung, sondern nur zu einer anderen, meist nur wenig schlechteren Klasseneinteilung. Durch Variieren der Parameter und Verfahren lassen sich verschiedene Lösungen erzeugen. An ihnen läßt sich auch die Stabilität einer Lösung erkennen. Eine Überprüfung der Klasseneinteilung ist auch nötig, denn manche Verfahren liefern auch leere Klassen oder für die Praxis unbrauchbare Einteilungen.

Kapitel 4

Clusteranalyse mit neuronalen Netzen

Nachdem im letzten Kapitel klassische Methoden zur Clusteranalyse erläutert wurden, stelle ich nun eine alternative Möglichkeit auf Basis von neuronalen Netzen vor. Damit werden manche Probleme der klassischen Clusteranalyseverfahren vermieden, es treten aber neue auf. Die Untersuchung dieser Alternative in Hinblick auf einen speziellen Anwendungsfall ist Kernstück dieser Diplomarbeit.

4.1 Welche neuronalen Netze eignen sich für die Clusteranalyse?

Für die Clusteranalyse eignen sich im Prinzip die meisten unüberwacht lernenden neuronalen Netze. Diese haben per Definition keine Vorgabe von Ausgabemustern. Sie können also nur die Strukturen in den Eingabemustern auswerten. Dies ist auch bei der Clusteranalyse der Fall. Als Eingabemuster werden die zu klassifizierenden Objekte verwendet, das Erregungsmuster der Neuronen spezifiziert die Klasse.

Es können aber auch überwacht lernende Netze für die Clusteranalyse eingesetzt werden, wenn man passende Ausgabemuster findet. Es ist möglich, als Vorgabe für die Ausgabe ebenfalls die Eingabemuster zu verwenden. Die Anzahl der Neuronen sollte in der Größenordnung der gewünschten Klassenanzahl liegen. Dann muß das Netz intern Klassen bilden, damit es mit der geringeren Anzahl der Freiheitsgrade trotzdem die Abbildung von Ein- auf Ausgabe gut approximiert. Aus den Gewichten im Netz lassen sich dann Rückschlüsse auf die Form und Position der Klassen ziehen.[1]

[1] Diese Auswertung ist i. a. nicht einfach.

Es wäre auch denkbar, daß man zu Beginn des Trainings jedes Eingabemuster zufällig in eine Klasse einordnet. Für jede Klasse ist ein Ausgabemuster definiert. (Z. B. das Ausgabemuster besteht aus $K-1$ Nullen und einer Eins. Die Position der Eins gibt die Klassennummer an.) Im Laufe des Lernprozesses verändert man an Hand der Netzausgabe diese Zuordnung von Eingabemuster und Ausgabemuster so, daß ähnliche Muster in die gleiche Klasse fallen.[2]

Unter den unüberwacht lernenden Netzen ist das sog. „Kohonen"-Netz (Self-Organizing-Map, kurz SOM) am bekanntesten. Es wurde Anfang der 80er Jahre von dem finnischen Professor Teuvo Kohonen entwickelt [Kohonen 1984, und später], [Ritter et al. 1991]. Seitdem gab es einige Erweiterungen und Varianten. Die Besonderheiten beim SOM sind die einfachen Lernregeln und Netzstrukturen. Damit lassen sich schnelle und einfache Clusteranalysen vornehmen. Deswegen beschränke ich mich in dieser Diplomarbeit auf diesen Typ von neuronalen Netzen.

Ein anderes unüberwacht lernendes Netzmodell ist das „ART" (*Adaptive Resonance Theory*) [Grossberg 1976, und später]. Es wurde ursprünglich entwickelt um das *Stabilitäts-Plastizitäts-Dilemma* neuronaler Netze zu lösen: *Wie können neue Muster gelernt werden, ohne daß alte Muster vergessen werden?* Das ART besteht aus zwei Neuronenschichten, deren Ein- und Ausgabe wechselseitig verknüpft sind. Zudem gibt es ein paar „Bausteine" zur Steuerung des Resonanzverhaltens. Beim Training werden neue Muster entweder schon bestehenden Klassen zugeordnet (große Ähnlichkeit zwischen Muster und einer Klasse) oder es wird eine neue Klasse gegründet (so lange dies möglich ist).

4.2 Beschreibung des Kohonen-Netzes

Ein Kohonen-Netz besteht aus K Neuronen mit je einem Gewichtsvektor $\mathbf{w}_i$, $i = 1, 2, \ldots, K$ der Dimensionalität λ. Es gibt N Eingabevektoren $\mathbf{x}_j$, $j = 1, 2, \ldots, N$ (=Objekte) derselben Dimensionalität. Zudem gibt es K Topologievektoren $\mathbf{n}_i$, $i = 1, 2, \ldots, K$, deren Dimensionalität unabhängig von λ ist und die Netzstruktur definieren.

Der Recall:
An jedem Neuron wird derselbe Eingabevektor angelegt. Die Aktivität eines Neurons ist proportional der Ähnlichkeit seiner Gewichte zum Eingabevektor. Die Neuronen beeinflussen sich gegenseitig in ihrer Aktivität nicht. Die Ausgabe des gesamten Netzes ist die „Nummer" (bzw. Position) des *aktivsten* Neurons.[3]

[2]Es sollte aber auch darauf geachtet werden, daß unähnliche Muster in verschiedene Klassen eingeteilt werden.

[3]Dies ist die praktisch eingesetzte Form des Kohonen-Netzes. Der ursprüngliche Ansatz von Teuvo Kohonen war ein dynamisches Netz, in denen sich die Aktivitäten gegenseitig hemmen

Das Training:
Die Besonderheit von selbstorganisierenden neuronalen Netzen ist die Verwendung einer *Nachbarschaftstopologie*, welche in der Lernphase einen wichtigen Einfluß auf die Änderung der Gewichte hat. Die Nachbarschaft wird durch die Position der Neuronen in einem *Topologieraum* und der Angabe einer *Nachbarschaftsfunktion* festgelegt.

Die Position der Neuronen im Topologieraum wird durch die *Topologievektoren* $\mathbf{n}_i$, $i = 1, 2, \ldots, K$ spezifiziert. Dieser Topologieraum ist unabhängig vom *Eingaberaum*, in dem die Eingabevektoren liegen.[4]

Die Nachbarschaftsfunktion $h(\mathbf{n}_r, \mathbf{n}_i)$ gibt die Stärke der Nachbarschaft zwischen Neuron r und Neuron i an. Meistens ist $h(\mathbf{n}_r, \mathbf{n}_i)$ eine monoton fallende Funktion des Abstands $\|\mathbf{n}_r - \mathbf{n}_i\|$. Beispiele für die Nachbarschaftsfunktion findet man im Abschnitt 4.2.2.

In der Trainingsphase wird in jedem Lernschritt ein Eingabevektor an alle Neuronen angelegt. Die Gewichte des aktivsten Neurons *und seiner Nachbarn* werden so angepaßt, daß sie noch aktiver werden.

Damit ergibt sich am Ende der Lernphase folgende Eigenschaft:
Jedem Neuron wird ein Teil des Eingaberaumes zugeordnet. In den Bereichen des Eingaberaumes mit höherer Dichte der Eingabevektoren ist die Dichte der Neuronen auch größer. Zudem werden im Topologieraum benachbarte Neuronen ähnliche Gewichte haben, d. h. sie sind auch im Eingaberaum benachbart (*Topologieerhaltung*).

Abbildung 4.1 ist ein ein Beispiel für ein Kohonen-Netz. Es gibt neun Neuronen die im Topologieraum eine Kette bilden. Der Topologieraum ist damit eindimensional. Dort beträgt der Abstand zwischen zwei benachbarten Neuronen genau eine Einheit. Der Eingaberaum ist zweidimensional. Man zeichnet dort die Neuronen gemäß ihrer Gewichte ein und verbindet diejenigen Neuronen, die im Topologieraum direkt benachbart sind.

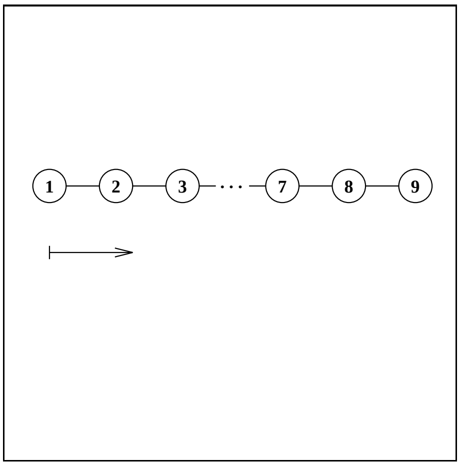
Für das Kohonen-Netz gibt es eine gute biologische Analogie (Abbildung 4.2). In diesem Fall sind der Topologieraum (die Neuronen an der Oberfläche des menschlichen Gehirns) und der Eingaberaum (die Nervenzellen in der menschlichen Haut) zweidimensional.

---

oder verstärken. Dies führt letztendlich zu einem kleinen, hochaktiven Gebiet an Neuronen. Diese Dynamik wird in dem vereinfachten Kohonen-Netzmodell zu einer *winner-takes-it-all*-Strategie reduziert.

[4] Oft geben die Topologievektoren $\mathbf{n}_i$ die Position des Neurons in einem regulären zweidimensionalen Gitter oder einer eindimensionalen Kette an. In den grafischen Darstellungen werden dann unmittelbare Nachbarn mit einer Linie verbunden.

Topologieraum



Eingaberaum

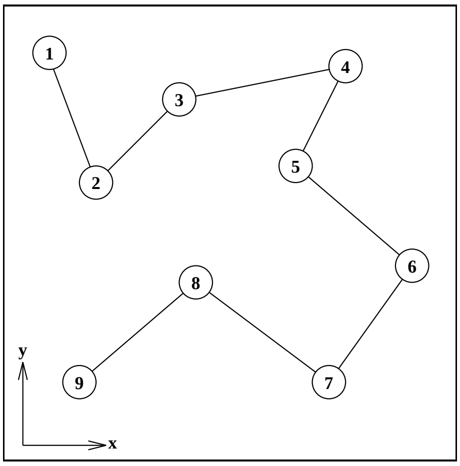


Abbildung 4.1: Beispiel eines Kohonen-Netzes.

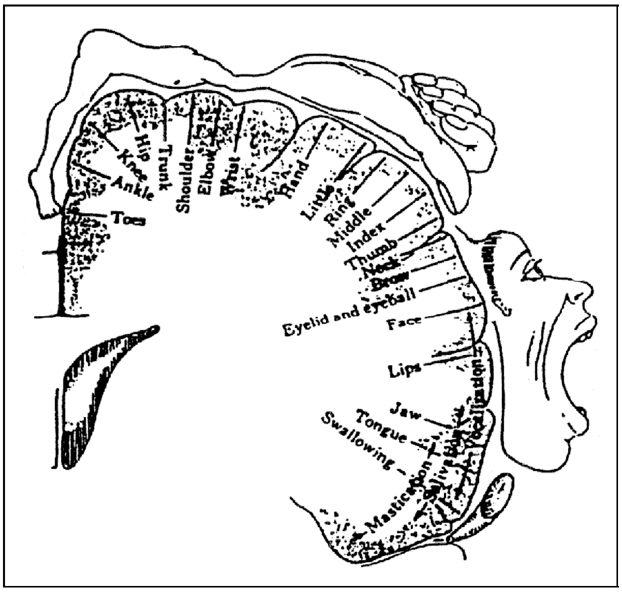
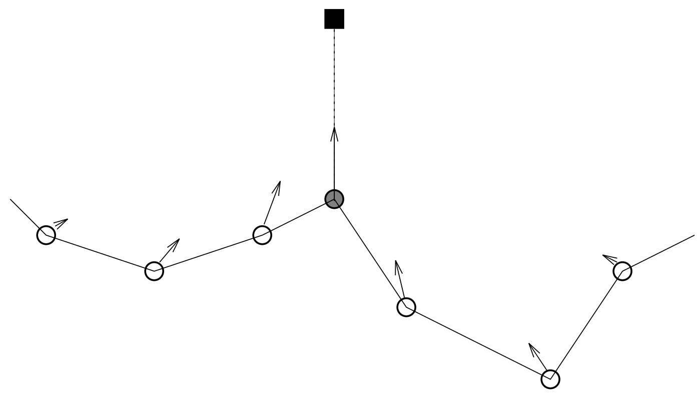


Abbildung 4.2: Biologische Analogie (aus [Köhle 1990]).

4.2.1 Algorithmus eines Kohonen-Netzes



Algorithmus (nach [Kohonen 1984]):

1. *„Initialisierung“:* Starte mit geeigneten Anfangswerten für die Gewichtsvektoren $\mathbf{w}_i$. In Abwesenheit irgendwelcher a-priori-Informationen können die $\mathbf{w}_i$ z. B. zufällig gewählt werden.

2. *„Stimuluswahl“:* Wähle zufällig einen Eingabevektor $\mathbf{x} = \mathbf{x}_j$.

3. *„Response“:* Bestimme das aktivste Neuron r aus der Bedingung

$$\|\mathbf{x} - \mathbf{w}_r\| = \min_{i=1}^{K} \|\mathbf{x} - \mathbf{w}_i\|. \tag{4.1}$$

4. *„Adaptationsschritt“:* Führe einen Lernschritt durch Veränderung aller Gewichte $\mathbf{w}_i$ gemäß

$$\mathbf{w}_i^{\text{neu}} = \mathbf{w}_i^{\text{alt}} + \alpha\, h(\mathbf{n}_r, \mathbf{n}_i)\,(\mathbf{x} - \mathbf{w}_i^{\text{alt}}) \tag{4.2}$$

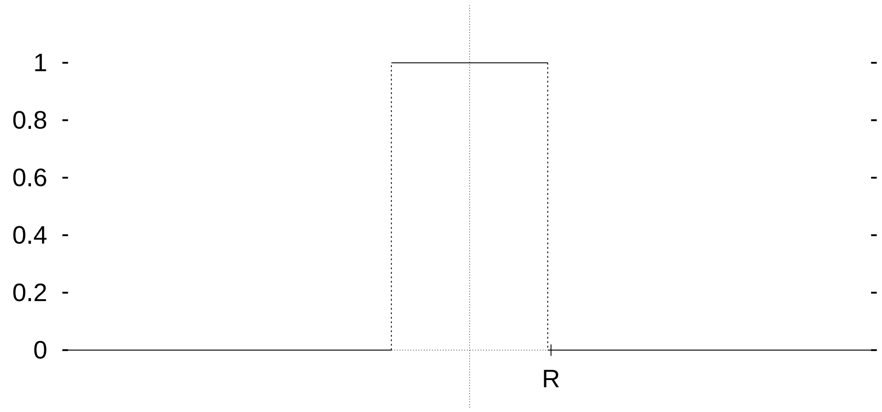
   aus, wobei α die Lernrate und $h(\mathbf{n}_r, \mathbf{n}_i)$ die Nachbarschaftsfunktion ist.

   Fahre mit Schritt 2 fort.

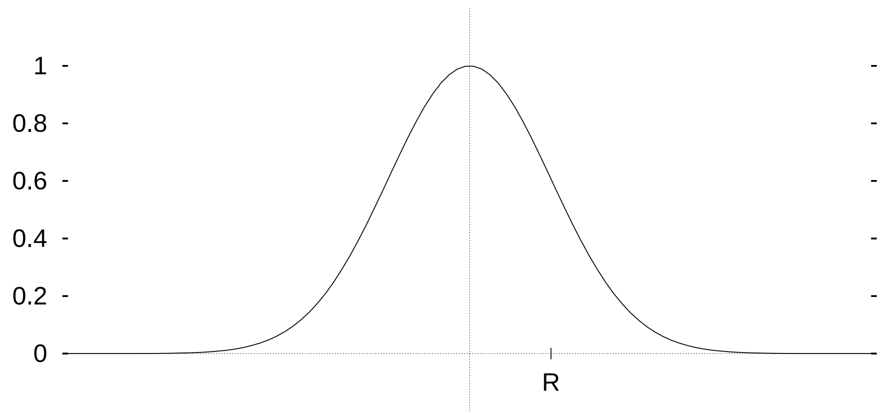
Damit der Lernvorgang konvergiert, sollte die Lernrate α abnehmen. Ebenso sollte der Wirkungsbereich der Nachbarschaftsfunktion $h(\mathbf{n}_r, \mathbf{n}_i)$ (realisiert durch den Nachbarschaftsradius) sinken.

4.2.2 Beispielfunktionen für die Nachbarschaft

Bubble-Nachbarschaft: $h(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 : \|\mathbf{x} - \mathbf{y}\| \leq R \\ 0 : \text{sonst} \end{cases}$



Gauß-Nachbarschaft: $h(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2R^2}\right)$



4.3 Clusteranalyse mit dem Kohonen-Netz

Es ist eine Menge von Datenvektoren $\mathbf{x}_i$ gegeben. Für diese Daten muß entschieden werden, ob und wie sie in einem Vorverarbeitungsschritt skaliert oder auf andere mathematische Räume abgebildet werden. Wenn in den Daten *natürliche* Cluster gesucht sind, ist es vorteilhaft, die einzelnen Dimensionen so zu skalieren, daß die Cluster kugelförmig sind. Dies ist aber ohne Inspektion der Daten oder Vorwissen nicht trivial. Auch haben die Cluster i. a. nicht die gleiche Form, so daß eine lineare Skalierung nur ein Kompromiß ist.

Das andere Ziel der Clusteranalyse kann die Suche nach Repräsentanten[5] für die Klassen sein (*Kompressionseffekt*). Diese Repräsentanten können anstatt der Objekte weiterverarbeitet werden. Der Vorteil liegt in der reduzierten Datenmenge. Dadurch, daß nur die Repräsentanten und nicht die Objekte weiterverarbeitet werden, ergibt sich allerdings ein Fehler im Endergebnis, der möglichst klein sein sollte. Dieser Fehler hängt von der Güte der Klasseneinteilung und der nachfolgenden Weiterverarbeitung ab. Die Klasseneinteilung versucht eine bestimmte Fehlerfunktion zu minimieren. Wählt man eine geeignete Metrik und eine passende Skalierung der Daten, führt die Minimierung des Fehlers der Klasseneinteilung zugleich zu einer Minimierung des Fehlers im Endergebnis der Weiterverarbeitung.

Aus der Fehlerfortpflanzung (genauer aus dem Gradienten der Fehlerfortpflanzung) durch die Weiterverarbeitung läßt sich die Metrik für das Kohonen-Netzes bestimmen. Näherungsweise können im Vorverarbeitungsschritt die Dimensionen nichtlinear skaliert und als Metrik die euklidische Metrik gewählt werden. Diese Näherung kann man auch dadurch erreichen, daß nicht der *Eingaberaum* skaliert, sondern die *Häufigkeit* der Datenvektoren variiert wird, beispielsweise durch öfteres Training mit demselben Datenvektor oder einer höheren Lernrate.

Oft wird man aber die optimale Skalierung nicht angeben können, sondern sie, vielleicht erst nach mehreren Testläufen, willkürlich festlegen.

Nach der Vorverarbeitung müssen folgende Parameter des Kohonen-Netzes festgelegt werden:

- Anzahl der Neuronen K,
- Netztopologie $\mathbf{n}_i$,
- Startwerte für die Gewichte der Neuronen $\mathbf{w}_i$,
- Nachbarschaftsfunktion $h(\mathbf{n}_r, \mathbf{n}_i)$ (Gauß-, Bubble-Nachbarschaft),

[5] Repräsentanten sind normalerweise die Zentren der Klassen.

- Anzahl der Iterationen T,
- Verlauf der Lernrate $\alpha = \alpha(t)$,
- Verlauf des Nachbarschaftsradius $R = R(t)$,
- Reihenfolge der Datenvektoren.

Damit kann ein Training des Kohonen-Netzes erfolgen. Im Normalfall wird sich das Netz zu Beginn zu einem kleinen Gebiet um den Schwerpunkt der Daten zusammenziehen. Nach langsamer Verringerung des Nachbarschaftsradius beginnt das Netz, sich auf die Gebiete auszudehnen, in denen Datenvektoren vorhanden sind. Zum Schluß sollte die Verteilung der Neuronen in etwa der Verteilung der Datenvektoren entsprechen.[6] Dann können die Gewichte der Neuronen als Repräsentanten der Klasse weiterverwendet werden. Die Zugehörigkeit eines Datenvektors zu einer Klasse ist durch die gleiche Formel wie beim K-Means-Verfahren definiert (Formel 3.1).

Es kann auch gezeigt werden, daß für $h(\mathbf{n}_r, \mathbf{n}_i) = \delta(\|\mathbf{n}_r - \mathbf{n}_i\|)$ (δ=Impulsfunktion), d. h. nur der Gewichtsvektor des aktivsten Neurons im Lernschritt verändert wird, so daß kein Einfluß der Topologie besteht, der Kohonen-Algorithmus dem K-Means-Verfahren entspricht. Der Unterschied liegt darin, daß bei K-Means der Fehler monoton gegen ein Minimum konvergiert und es nach endlich vielen Schritte auch exakt erreicht. Das Kohonen-Netz konvergiert hingegen nur bei hinreichend kleiner Lernrate. Dies liegt daran, daß K-Means „epochenweise“ arbeitet, also daß alle Eingabedaten auf einmal verwendet werden. Dagegen entwickelt sich das Kohonen-Netz schrittweise.

Nun fragt man sich, welche Rolle die Topologie beim Training des Kohonen-Netzes spielt. Das K-Means-Verfahren hat den Nachteil, bei schlechten Ausgangsklassen auch schlechte Endklassen zu liefern, also in einem hohen lokalen Minimum stecken zu bleiben. Dies liegt an der starken Lokalität des Verfahrens. Beim Kohonen-Netz sorgt die Topologie hingegen für einen größeren globalen Zusammenhang. Der *Prozeß* des Lernens bestimmt, welches (lokale) Minimum erreicht wird. Welche Topologie nun für die Suche eines guten Minimums die richtige ist, wird vom Einzelfall abhängen.

Die Topologie hat aber noch andere Vorteile. Durch sie werden Ähnlichkeiten zwischen den Klassen definiert. Für manche praktische Anwendungen ist dieser Nebeneffekt recht nützlich.

[6] Dieser Zusammenhang ist nur qualitativ. Es ist keine Proportionalität!

4.4 Erweiterungen des Kohonen-Netz-Modells

Das Kohonen-Netzmodell kann für Klassifizierungszwecke geeignet modifiziert werden. Ein erster Ansatz ist die dynamische Bestimmung der Topologie. Die einfachste Möglichkeit ist, aus den Gewichten die Topologie abzuleiten.

Eine Variante wird in [Martinetz et al. 1993] beschrieben und nennt sich „Neuronales Gas“. Zu Beginn des Simulationslaufes werden keine Topologievektoren $\mathbf{n}_i$ festgelegt. Für die Berechnung der Nachbarschaft $h(\mathbf{n}_r, \mathbf{n}_i)$ werden dann bei jedem Adaptationsschritt die Topologievektoren $\mathbf{n}_i$ neu bestimmt:
Die Neuronen werden gemäß der Abstände ihrer Gewichte $\mathbf{w}_i$ zu $\mathbf{w}_r$ (dem Gewicht des aktivsten Neurons) sortiert. Sie bilden dann eine Kette von Neuronen mit dem Abstand Eins zwischen zwei Nachbarn. Dadurch sind die $\mathbf{n}_i$ bestimmt. Man beachte, daß diese Topologie nur für den Fall gilt, in dem r das aktivste Neuron ist. Wäre das Neuron $\tilde{r}$ am aktivsten, gibt es i. a. ein andere Topologie.

Eine weitere Idee ist, die Topologie nur durch nächste Nachbarn zu beschreiben (nächste Nachbarn werden in Grafiken durch Linien verbunden). Das Netz beginnt mit einem einzigen Neuron. Während des Lernprozesses werden Neuronen und Verbindungen hinzugefügt oder gelöscht, bis ein stabiler Zustand erreicht ist. Dieses Modell nennt sich „Growing-Cell-Structures“ [Fritzke 1992].

Teil II

Die Anwendung

Kapitel 5

Das zu lösende Problem

Mittels Clusteranalyse soll ein Teilproblem eines größeren Projekts in der Klimaforschung gelöst werden. Dieses Projekt ist unter dem Begriff „Regionalisierung globaler Klimamodelle“ bekannt. Dieses Projekt bestimmt die Rahmenbedingungen und das Ziel der Clusteranalyse.

5.1 Das Regionalisierungsproblem

Die Vorgänge und Reaktionen in der Erdatmosphäre lassen sich durch physikalische (und chemische) Gleichungen annähernd beschreiben. So kann man ein Klimamodell entwickeln, das ansatzweise die Strömungen und andere physikalischen Daten der gesamten Erdatmosphäre nachbildet. Dieses Modell wird auf einen Computer übertragen und numerisch simuliert. Damit können Prognosen für zukünftige Klimasituationen gestellt werden.[1] Auch für die Simulation der Eiszeiten oder der Atmosphärendynamik auf anderen Planeten leisten diese Modelle gute Dienste. Ein anderes Beispiel ist die Wettervorhersage für die nächsten Tage.

Durch die Beschränkungen des Computers in Hinblick auf Rechenzeit und Datenvolumen sowie das noch unvollständige Wissen der Physiker über die Zusammenhänge in der Atmosphäre kann ein Klimamodell nur eine unvollkommene Nachbildung der Wirklichkeit sein. Die Güte einer Simulation hängt im wesentlichen von Faktoren ab, wie

- Länge des simulierten Zeitraums,

[1]Ein aktuelles Beispiel ist die Bestimmung des Einflusses von anthropogenen Spurengasen auf das Erdklima (Treibhauseffekt) für die nächsten 100 Jahre.

- Güte der verwendeten Formeln,
- räumliche und zeitliche Auflösung (Diskretisierung für numerischen Ansatz),
- Güte der Anfangsdaten.

Durch die nichtlineare Dynamik der Erdatmosphäre lassen sich Prognosen in der Qualität der amtlichen Wettervorhersage für einen Zeitrum von ungefähr sieben Tagen erstellen. Wesentlich größere Zeiträume sind nicht vorhersagbar und werden auch bei erhöhtem Rechen- und Datenaufwand nicht vorhersagbar sein. Es läßt sich also keine Frage wie „Welche Tagestemperatur gibt es am 1. April 2020 in München?“ beantworten.

Trotzdem simuliert man mit einem Klimamodell die gesamte Erdatmosphäre für Zeiträume von Jahrzehnten. Man hegt die nicht unbegründete Hoffnung, daß man statistische Aussagen über das zukünftige Klima treffen kann. („Welches Jahresmittel besitzt die Temperatur im Jahre 2020 in Süddeutschland?“)

Globale Klimamodelle werden für sehr lange Simulationszeiträume angewendet und haben eine grobe Auflösung (ca. 500 km). Eine höhere Auflösung ist nur für ein begrenztes Gebiet oder für kürzere Zeiträume möglich. Im Projekt „Regionalisierung“ wird eine Auflösung von unter 30 km angestrebt, welche von *mesoskaligen Klimamodellen*[2] erreicht wird.

Ein Problem besonderer Art stellt sich, wenn für ein kleines Gebiet (z. B. Alpen) eine langfristige Prognose erstellt werden soll. Das mesoskalige Klimamodell kann keine langen Zeiträume simulieren, weil erstens die Rechenzeit zu groß wäre und zweitens der Einfluß des nicht berücksichtigten Gebiets mit der Zeit steigt (Randwertproblem).

Ein globales Klimamodell kann die regionalen Phänomene nicht auflösen.[3] Ein erster Ansatz ist die Koppelung des mesoskaligen Modells an ein dazu parallel laufendes globales Klimamodell. Damit ist das Randwertproblem gelöst. Das Problem der Rechenzeit besteht immer noch.

Deshalb wird folgende Lösung versucht:
Da eine dauerhafte Koppelung von globalem und regionalen Klimamodell zu rechenaufwendig ist, wird das regionale Modell nur für ausgewählte Zeiträume an das globale Klimamodell gekoppelt. Dieses Vorgehen kann nur statistische Aussagen liefern. Praktisch umgesetzt bedeutet dies: Es wird ein Simulationslauf mit dem globalen Klimamodell durchgeführt und für jeden Tag ein Datensatz

[2] Sie werden auch als *regionale Klimamodelle* bezeichnet.

[3] Solche regionalen Phänomene sind im Fallbeispiel „Alpenraum“ beispielsweise Berg- und Talwinde oder Veränderungen von Gletschern und Schneegrenzen.

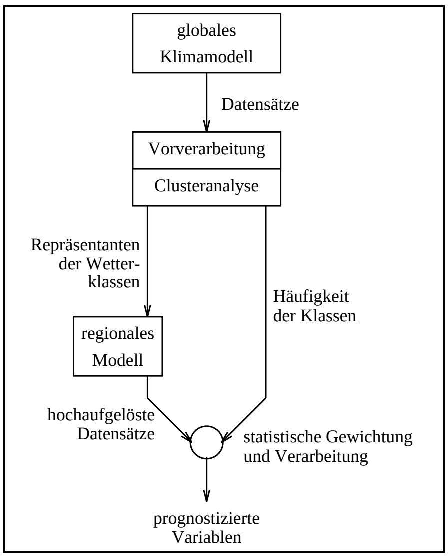


Abbildung 5.1: Einbettung der Clusteranalyse in das Anwendungsproblem.

abgespeichert.[4] Aus diesen Datensätzen werden mittels Clusteranalyse die „typischen“ Wetterlagen ermittelt. Aus jeder dieser „typischen“ Wetterlagen werden die Anfangswerte für das regionale Modell errechnet und ein Simulationslauf mit dem regionalen Modell durchgeführt. Das Modell berechnet aus diesen Anfangsdaten einen besseren Tages-Datensatz, in dem sich die regionalen Phänomene ausprägen. Aus diesen Datensätzen und der Häufigkeit der dazugehörenden Wetterlagen lassen sich statistische Daten berechnen, wie beispielsweise die regionale Verteilung der Temperaturmittelwerte über den mehrjährigen simulierten Zeitraum [Frey-Buness et al. 1995]. Der Ablauf ist in Abbildung 5.1 veranschaulicht.

Die Ermittlung der „typischen“ Wetterlagen (Wetterklassen) ist das Clusteranalyseproblem, bei dem neuronale Netze zum Einsatz kommen.

[4]Der simulierte Zeitraum kann dabei mehrere Jahrzehnte lang sein.

5.2 Der konkrete Anwendungsfall

Im letzten Abschnitt wurde das Projekt „Regionalisierung“ allgemein beschrieben. Um die Brauchbarkeit dieses Ansatzes zu verifizieren, wurde an einem konkreten Fall dieser Ablauf durchgeführt. Dabei kam eine einfache Klasseneinteilung zum Einsatz. Um die Klasseneinteilung zu verbessern, sollen in Zukunft teilweise neuronale Netze verwendet werden.

5.2.1 Beschreibung

Zur Verifikation verwendet man statt der von einem Klimamodell erzeugten Datenreihe über ein zukünftiges Klimaszenario reale Beobachtungsdaten[5] als Ausgangsdatenbasis. Diese Daten beziehen sich auf den Zeitraum vom 1. Januar 1981 bis 31. Dezember 1990. Je Tag gibt es einen Datensatz, der die Erdatmosphäre in Rahmen der Auflösungsgenauigkeit[6] beschreibt. Insgesamt gibt es also $N = 3652$ Datensätze. Das Zielgebiet, für welches das regionale Modell eingesetzt werden soll, ist der Alpenraum. Deswegen beschränkt man sich auf ein Gitter von 4x3 mal 9 Schichten. Für jeden Gitterpunkt gibt es sechs wichtige physikalische Größen (dreidimensionaler Windvektor, Temperatur, Luftfeuchte und Geopotential).[7] Damit beträgt die Dimension λ eines Datensatzes $\lambda = 4 \cdot 3 \cdot 9 \cdot 6 = 648$.

Ziel ist also die Einteilung der $N = 3652$ Datensätze aufgrund bestimmter Merkmale in ihren $\lambda = 648$ Dimensionen. Diese Daten werden (bisher) durch einfache Schwellwerte in Klassen eingeteilt: *Jahreszeit:* Sommer, Winter; *Gesamtniederschlag:* trocken, konvektiver Niederschlag, stratiformer Niederschlag; *mittlere horizontale Windstärke:* stark, schwach, windstill; *mittlere horizontale Windrichtung:* Aufteilung in 12 Sektoren von je 30°. Damit kommt man auf ungefähr[8] $N = 150$ Klassen. Man sieht, daß einige Variablen nicht mit einbezogen wurden, dafür aber auch eine Aufteilung in der Zeit vorgenommen wird. Was völlig fehlt, ist eine Unterscheidung verschiedener Strukturen im Raum — es werden nur Mittelwerte verwendet.

---

[5] Diese Daten werden weltweit von Meßstationen und Satelliten an das ECMWF (European Centre for Medium Range Weather Forecasts) in Reading/UK gesendet und dort miteinander verknüpft. Sie werden in einem Wettervorhersagemodell so verarbeitet, daß für jeden Tag ein kompletter Datensatz mit der Zustandsbeschreibung der Erdatmosphäre erhältlich ist.

[6] Für die Anwendung wurde die Auflösung auf ein dreidimensionales Gitter reduziert, welches 128 Komponenten in West-Ost-Richtung, 64 Komponenten in Nord-Süd-Richtung und 13 Schichten in der Höhe besitzt. Dies ist die typische Auflösung eines globalen Klimamodells.

[7] Es gibt noch die Größe „Wolkenwasser“, welche aber in die Klasseneinteilung nicht mit einbezogen wird.

[8] Manche Klassen sind fast leer und fallen weg.

Für jede dieser Klassen wird ein Lauf des regionalen Modells durchgeführt (ca. 24 simulierte Stunden je Klasse). Der Simulationslauf erzeugt aus dem groben 4x3x9-Gitter ein viel höher aufgelöstes Gitter, in dem die regionalen Phänomene in Erscheinung treten. Mit diesen Datensätzen lassen sich Monats- oder Jahresmittel von bestimmten Größen (Temperatur, Niederschlag, etc.) für jeden Punkt des Gitters errechnen. Der Vergleich mit beobachteten Werten von 30 Meßstationen, verteilt über den Alpenraum, bringt bei manchen Variablen eine gute Übereinstimmung, bei anderen zeigt sich ein systematischer Fehler [Fuentes et al. 1995].

Zur Verbesserung des Verfahrens soll die einfache Klasseneinteilung durch eine komplexere ersetzt werden. Der erste Ansatz ist eine bessere Einteilung der Windrichtung und -stärke (siehe Abschnitt 7.5).[9] Der nächste Schritt ist die Einteilung kompletter Windfelder in Klassen. Dabei stellt sich schon das Problem der Skalierung (siehe nächster Abschnitt), denn der Wind wird nicht an jedem Gitterpunkt die gleiche Varianz haben. Wahrscheinlich ist die Skalierung des Windes an jeden Gitterpunkte auf die Varianz Eins eine sinnvolle Lösung.

Dann können schrittweise auch die anderen physikalischen Größen hinzugefügt und der Effekt auf die Klasseneinteilung beurteilt werden. Eine Skalierung der Daten kann dann für eine ausgewogene Klassenstruktur sorgen. Den Einfluß auf die Güte des Endergebnisses können letztendlich aber nur die (sehr rechenintensiven) Läufe des regionalen Modells beantworten.

5.2.2 Vorverarbeitung und Skalierung

Aus der Projektbeschreibung ist ersichtlich, nach welchem Ziel die Clusteranalyse die Datenfelder aufteilen soll:
Der Fehler, der durch die verringerte Anzahl der Datenfelder in den prognostizierten Variablen (am Ende der Verfahrenskette) entsteht, soll minimal sein.

Aus diesem Ansatz lassen sich im Prinzip die Skalierung der Datenfelder in der Vorverarbeitung und die Metrik bestimmen. Das regionale Klimamodell ist im Prinzip nur eine mehrdimensionale Funktion, ebenso die statistische Berechnung der zu prognostizierenden Variablen. Daraus könnte man eine neue Fehlerfunktion ableiten, die minimiert werden soll. Dies führt zu einer neuen Metrik, bei welcher i. a. der Abstand zweier Vektoren abhängig von dem Ort der Vektoren ist. Das Problem ist aber, daß die Berechnung der Funktion „regionales Klimamodell“ sehr rechenaufwendig ist: Es ist ein ganzer Simulationslauf.

Auch die partiellen Ableitungen sind nicht bekannt. Diese könnten als Näherung für die Skalierung verwendet werden. Damit besteht nur die Möglichkeit,

[9] Der Wind ist deswegen so bedeutend, da er die Hauptantriebskraft für die Vorgänge in der Atmosphäre ist.

die Skalierung durch physikalische Überlegungen, empirische Erfahrungen und Ausprobieren zu bestimmen.

Zur Verringerung der Rechenzeit können mittels einer *Hauptkomponententransformation*[10] korrelierte Dimensionen entfernt werden. Die selbstorganisierenden neuronalen Netze haben aber im Normalfall keine Probleme mit hochdimensionalen Daten; sie können sogar zur Dimensionsreduzierung eingesetzt werden.

[10]Sie ist auch unter den Begriff *Empirische-Orthogonal-Transformation* bekannt.

Kapitel 6

Simulatoren für neuronale Netze

Um die theoretischen Netzmodelle in die Praxis umzusetzen, verwendet man meist Simulatoren auf einem Computer.[1] Welche Programmpakete verfügbar sind und warum ich für die praktischen Versuche ein neues entwickelte, wird in diesem Kapitel beschrieben.

6.1 Bekannte Programmpakete

Für neuronale Netze gibt es einige Simulatoren, als Share- oder Freeware im Internet erhältlich oder käuflich zu erwerben. Eine gute Übersicht findet man in [Zell 1994]. Eine unkomplizierte Lösung ist „SOM-PAK", entwickelt an der University of Technology in Helsinki. Es ist per anonymous FTP von *cochlea.hut.fi* im Verzeichnis */pub/som_pak* unter dem Namen *som_pak-3.1.tar.Z* erhältlich. Es bietet im Gegensatz zu den anderen Paketen keine grafische Darstellungsmöglichkeiten, kann jedoch auf fast jeden Computer mit C-Compiler verwendet werden, von einem Großrechner bis zum PC. Die einzige größere Schwäche liegt in der Beschränkung der Netztopologie auf ein- oder zwei Dimensionen. Eine Erweiterung von SOM-PAK auf mehrere Dimensionen oder andere Topologien ist programmtechnisch nur schwer realisierbar.

Zwei andere Programmpakete, „Xerion" von der Universität von Toronto und „SNNS" von der Universität Stuttgart, bieten eine bessere grafische Präsentation. Beide unterstützen aber in nur geringem Umfang die Simulation von Kohonen-Netzen (Stand Ende 1994). Die Stärke dieser Programme liegt mehr bei überwacht lernenden Netzen. Zudem sind sie für Forschungszwecke frei verwendbar und über das Internet leicht zu besorgen.

[1] Es gibt auch Projekte, in denen neuronale Netze direkt aus elektronischen Bausteinen aufgebaut werden.

Für das kommerzielle Grafik- und Mathematikpaket „MATLAB“ von der Firma „The Mathworks“ gibt es eine sog. *Toolbox* für neuronale Netze. Laut Beschreibung dieser Toolbox sind die Funktionen in Bezug auf neuronale Netze sehr starr. Die Vorgabe von frei definierten Topologien oder Verwendung von leicht geänderte Versionen des Kohonen-Netzmodells sind nicht möglich. Da „MATLAB“ ein mächtiges Programmierwerkzeug ist, könnte man trotzdem schnell einen Prototypen eines Simulators für neuronale Netze entwickeln.

6.2 Das eingesetzte Programmpaket „Sonnet“

Von den bekannten Programmpaketen würde SOM-PAK die gestellten Anforderungen (Portabilität, Bedienung, Geschwindigkeit) sehr gut erfüllen. Nur die Möglichkeiten in Bezug auf die Variation der Netztopologie sind zu eingeschränkt. Besonders da bei der Anwendung die Daten sehr hochdimensional sein können und höherdimensionale Netztopologien vielleicht besser geeignet sind.

Das Resümee: Entwicklung von *Sonnet*, einem Simulators für selbstorganisierende neuronale Netze[2], wobei SOM-PAK als Vorbild diente.

Die Vorteile von *Sonnet*:

- **Portabilität**
  Durch Verwendung der Programmiersprache „ANSI C“ ist man auf keine besonderen Softwareprodukte angewiesen. Der Kern des Simulators läßt sich im Prinzip auf jedem Rechner mit C-Compiler übersetzen und verwenden. Die grafische Darstellung ist Aufgabe anderer Programme. Für den praktischen Einsatz ist jedoch Unix ideal, da die Datenvorverarbeitung, Auswertung und grafische Präsentation mit entsprechenden Unix-Befehlen sehr flexibel zu realisieren sind. Diese Trennung von Rechenleistung und Visualisierung ist ideal für den geplanten Batchbetrieb des Simulators auf einem Backend (Cray YMP, NEC-Großrechner) und der Auswertung auf einem Frontend (Sun Workstation, PC).

- **Erweiterungsmöglichkeiten**
  In diesem Programmpaket wurden von Beginn an Erweiterungen des Kohonen-Algorithmus und der Datenstrukturen eingeplant. Damit läßt sich nicht nur das originale Kohonen-Netz simulieren, sondern auch andere Varianten. Zur Zeit ist neben dem Kohonen-Netz auch das „Neuronale Gas“ implementiert.

[2]Den Begriff „selbstorganisierende neuronale Netze“ will ich als Oberbegriff für die Kohonen-Netze („SOM = Self-Organizing-Maps/selbstorganisierende Karten“) und deren Varianten verwenden.

- **Geschwindigkeit**
  Da sich der Simulator auf die selbstorganisierenden Netze beschränkt, gibt es keinen Verwaltungsoverhead, der entsteht, wenn alle möglichen Netze mit demselben Programm laufen sollen. Außerdem wird das Programm durch einen Compiler in Maschinensprache übersetzt und nicht von einem Interpreter Schritt für Schritt abgearbeitet, wie es bei MATLAB (vielleicht) der Fall ist. Ein anderer Gesichtspunkt für die Effizienz der Implementierung ist der Umgang mit großen Datenmengen.

Ich möchte natürlich auch nicht die Nachteile des entwickelten Simulators verschweigen:

- Da (fast) keine eigenen grafischen Tools oder Werkzeuge für die Auswertung der Daten erstellt wurden, ist man dabei auf andere Programme angewiesen. Die Umformatierung der Daten ist Sache des Benutzers.
- Es gibt auch keine grafische Oberfläche, sondern mehrere Programme, die durch Parameter beim Aufruf gesteuert werden.
- Die Softwarewartung und Weiterentwicklung des Paketes liegt in den Händen der Anwender. Das Qualitätsniveau der Software ist (noch) nicht so hoch wie der eines ausgereiften Produkts.

6.3 Funktionsbeschreibung von „Sonnet“

Funktionen des entwickelten Simulatorpakets:

- **Der Simulator**
  Der Simulator ist das Kernstück des Pakets. Er generiert aus den Eingabevektoren und einem initialisierten Netz ein trainiertes Netz.
  - Implementierte Netztopologien:
    - Kohonen-Netz: Gitter beliebiger Dimension und Größe. Für jede Dimension kann eine ringförmige Nachbarschaft festgelegt werden. Damit wird z. B. aus einer Kette ein Ring oder aus einem zweidimensionalen Netz ein Zylinder oder ein Torus.
    - Neuronales Gas:
      Außer einem Gitter kann der Benutzer auch ein „Neuronales Gas“ als Topologie definieren. Die Nachbarschaft wird dann dynamisch berechnet.

    - * Growing-Cell-Structures:
      Ist leider (noch) nicht implementiert. Dafür müßten weitere Datenstrukturen zur Verwaltung der Neuronen und Nachbarschaften angelegt werden.

  - Nachbarschaftsfunktionen:
    Es sind die Gauß- und Bubble-Nachbarschaftsfunktion implementiert.
  - Eingabevektoren/Gewichte:
    Die Dimensionalität λ der Eingabevektoren und Gewichte dürfen beliebig groß gewählt werden. Für jede Dimension kann angegeben werden, ob es sich dabei um ein Winkelmaß (Bereich 0-360 Grad) handelt, welches periodisch ist.
  - Parameter für das Training:
    Für das Training des selbstorganisierenden Netzes gibt man folgende Parameter an: Anzahl Iterationen, Start- und Endwerte von Lernrate α und Nachbarschaftsradius R. Die Werte für den einzelnen Lernschritt berechnen sich durch lineare Interpolation zwischen Start- und Endwert. (Eine exponentielle Interpolation der Werte fehlt leider. Sie würde vielleicht besser geeignet sein.)

    Zudem kann festgelegt werden, ob der Eingabevektor, für den der einzelne Lernschritte durchgeführt wird, zufällig oder sequentiell aus der Liste aller Eingabevektoren ausgewählt wird.
  - Ausgaben des Simulators:
    Der Simulator liefert ein trainiertes Netz. Optional kann während des Trainings regelmäßig die Streuungsquadratsumme berechnet werden. Auf Unix-Rechnern mit X11-Oberfläche kann der Simulator in einem Grafikfenster die Gewichte und die Eingabevektoren während des Trainings anzeigen.

- **Auswertungsmöglichkeiten**
  Nach dem Training berechnet ein entsprechendes Programm zu jedem Neuron die Anzahl der zugeordneten Eingabevektoren (Klassenmitglieder) und die Streuungsquadratsumme innerhalb der Klasse. Diese Streuungsquadratsummen werden aufsummiert und bilden die Streuungsquadratsumme der gesamten Klasseneinteilung. Zum Vergleich werden zwei Referenzfehler berechnet.

- **Generierung von Testdaten und Netzinitialisierung**
  Ein Programm liefert künstliche Testdaten. Aus diesen Testdaten oder realen Daten können mit einem anderen Programm passende Anfangswerte für die Gewichte der Neuronen ermittelt werden.

Kapitel 7

Niederdimensionale Daten

Um den Einfluß der Parameter des Kohonen-Netzes auf die Clusteranalyse zu bestimmen, wurden mit niederdimensionalen Daten Versuche gemacht. Die aufzuteilenden Objekte wurden entweder zufällig erzeugt oder aus realen Datensätzen extrahiert.

Der ein- und zweidimensionale Fall hat den Vorteil der einfachen grafischen Darstellung auf Papier oder Bildschirm. Dadurch ist auch der Effekt der Skalierung der Objekte leicht zu bewerten.

7.1 Beispiel: Gleichverteilung

Einer der einfachsten und doch interessanten Fälle ist die Gleichverteilung der Eingabevektoren innerhalb eines Quadrates. Die Anzahl der Daten beträgt 1000. Die Anzahl der Klassen wird für den ersten Versuch auf 12 festgesetzt. Als Netz wird ein Kohonen-Netz mit Kettenstruktur verwendet. Die Gewichte der Neuronen werden mit dem Schwerpunkt der Daten initialisiert. Um ein gutes Verhältnis zwischen Nachbarschaftsradius und Lernrate zu erreichen, wird das Netz in zwei Phasen[1] trainiert. In der ersten Phase bleibt die Lernrate konstant und der Nachbarschaftsradius geht linear gegen Null. In der zweiten Phase ist der Nachbarschaftsradius fast Null[2] und die Lernrate geht linear gegen Null. Die genauen Parameter für diesen Fall sind aus Tabelle 7.1 zu entnehmen. (Diese Tabelle gilt auch für alle weiteren Fälle in dieser Diplomarbeit. Auf Abweichungen von diesen Werten wird an entsprechender Stelle hingewiesen.)

[1] Begründung siehe Abschnitt 7.2.1.

[2] Aus programmtechnischen Gründen darf R nicht gleich Null sein, da sonst die Gauß-Nachbarschaftsfunktion eine Division durch Null durchführt. Bei der Bubble-Nachbarschaft könnte es zu Problemen durch Rundungsfehler kommen.

Daten: $N = 1000$ $\lambda = 2$ $\mathbf{x}_j$ = gleichverteilt ($-\sqrt{3}$ bis $\sqrt{3}$)	**Neuronen:** $K = 12$ $\mathbf{w}_i = \frac{1}{N}\sum_{j=1}^{N}\mathbf{x}_j$ (Schwerpunkt) $\mathbf{n}_i = i$ (Kette, $\mathbf{n}_i \in \Re^1$) Nachbarschaft = Gauß
Phase 1: T = 5000 $\alpha_{\text{start}} = 0.1$ $\alpha_{\text{end}} = 0.1$ $R_{\text{start}} = 4.0$ $R_{\text{end}} = 0.0001$ select = random step = 20	**Phase 2:** T = 1000 $\alpha_{\text{start}} = 0.1$ $\alpha_{\text{end}} = 0.0001$ $R_{\text{start}} = 0.0001$ $R_{\text{end}} = 0.0001$ select = sequential step = 20

Tabelle 7.1: Parameter für das Training.

Das Ergebnis nach 6000 Lernschritten ist in Abbildung 7.1 zu sehen. Es sind die Gewichte der Neuronen und die Eingabevektoren eingetragen und benachbarte Neuronen mit einer Linie verbunden. Wie man sieht, sind die Gewichte auch ungefähr gleichverteilt. Diese Gewichte definieren nach Formel (3.1) die Klasseneinteilung. In den beiden rechten Bildern ist die Fläche eines Kreises proportional der Anzahl der Klassenmitglieder (oben) bzw. der Streuungsquadratsumme innerhalb der Klasse (unten).

Berechnet man regelmäßig während des Trainings die Streuungsquadratsumme, so ergibt sich ein Diagramm wie in Abbildung 7.2.[3] Der Fehler fällt von anfangs 2000 auf 160.948.

Zum Vergleich ist der Referenzfehler $Q^{(\text{eq})}$ als konstante Linie bei 166.666 eingezeichnet. Dieser Referenzfehler berechnet sich nach Formel (3.5):

$$Q^{(\text{eq})} = \frac{1}{12}\lambda N a^2 \quad \text{mit} \quad a = \sqrt[\lambda]{\frac{\text{Volumen}}{K}}$$

Das Volumen beträgt $\left(2\sqrt{3}\right)^2 = 12$. Damit ergibt sich

$$Q^{(\text{eq})} = \frac{1}{12} \cdot 2 \cdot 1000 \cdot \left(\sqrt{12.0/12}\right)^2 = 166.\bar{6}.$$

[3]Der Parameter „step“ in Tabelle 7.1 gibt die Anzahl der Lernschritte zwischen zwei Berechnungen an.

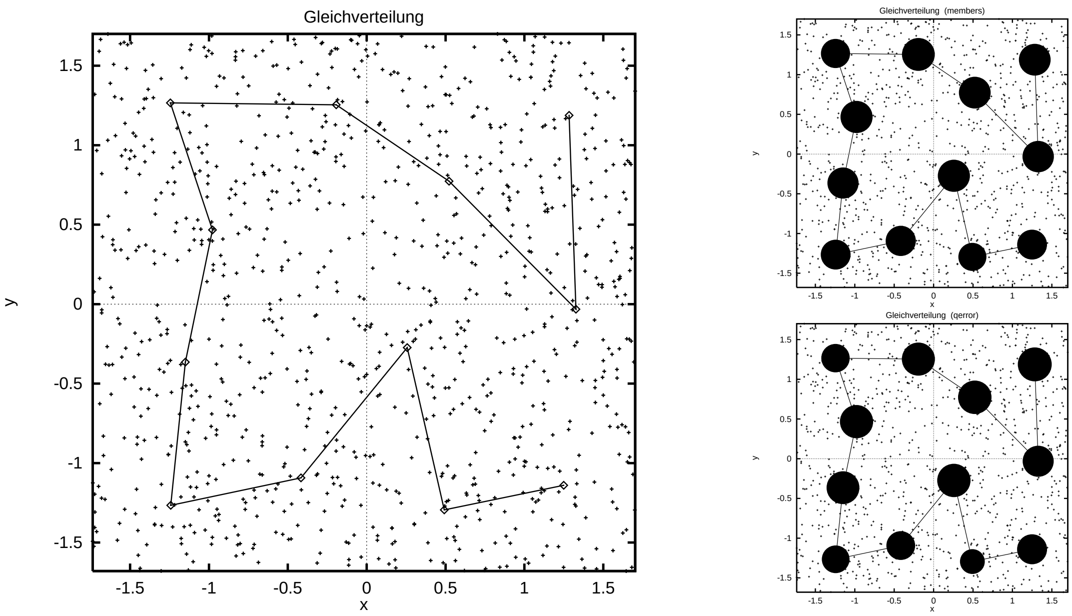


Abbildung 7.1: Eingabevektoren und Neuronen.
(Rechts oben ist die Mitgliederanzahl, rechts unten die Fehlersumme proportional der Kreisfläche.)

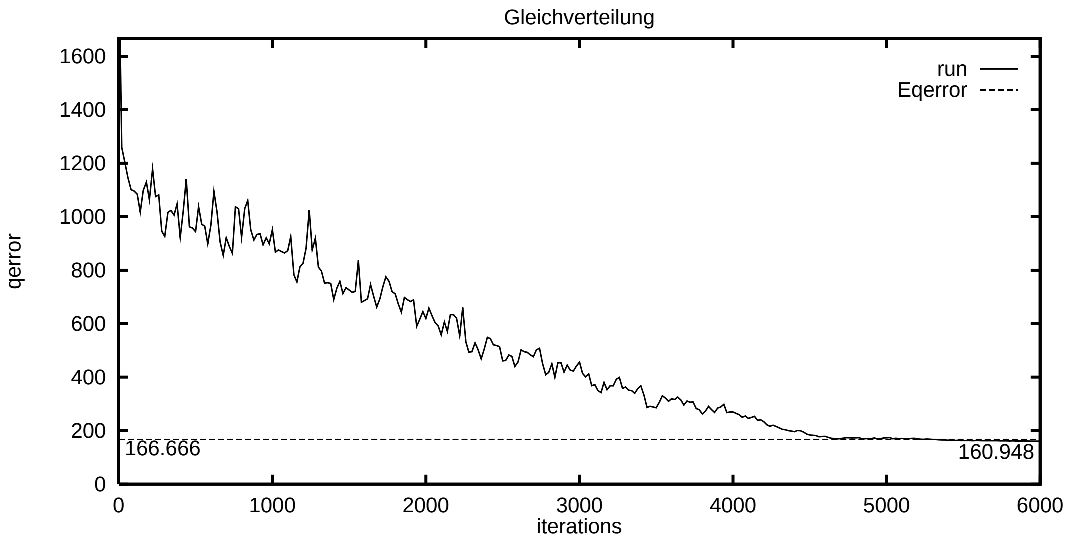


Abbildung 7.2: Trainingsverhalten.

Daß der reale Fehler in diesem Fall kleiner als der theoretische Fehler ist, läßt sich dadurch erklären, daß erstens die Position der Gewichte relativ gut ist, und zweitens, daß die Eingabevektoren nur eine *diskrete* Gleichverteilung realisieren und damit nicht absolut gleichverteilt sind.

Die Entwicklung des gesamten Netzes ist in den Abbildungen 7.3 und 7.4 festgehalten. Die Gewichte des Netzes liegen zu Beginn genau im Schwerpunkt. Im Laufe des Trainings wächst das Netz, bis es zum Schluß über das ganze Gebiet verteilt ist.

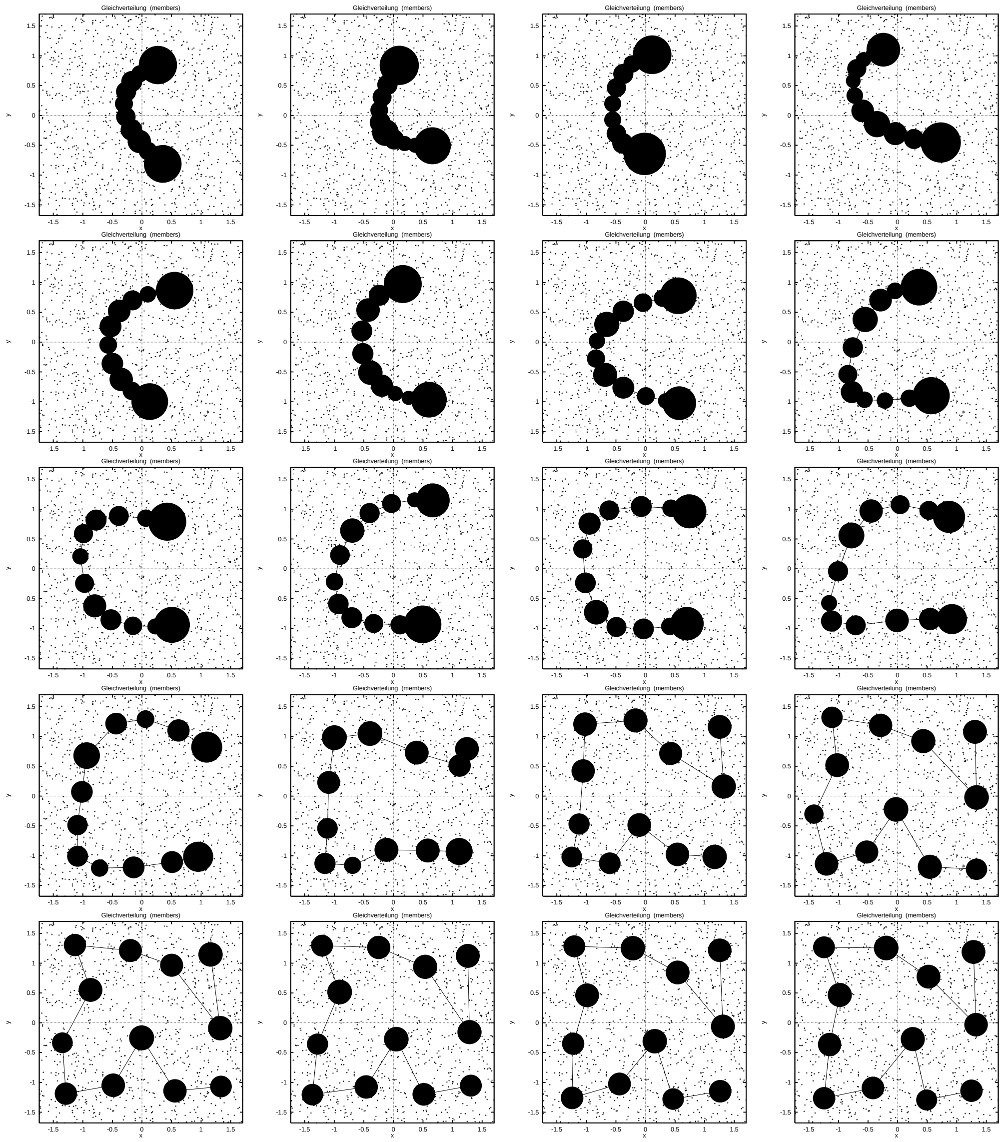


Abbildung 7.3: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt ($300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren (Zentrum gleich Neuronengewicht) sind proportional zur Anzahl der Klassenmitglieder.

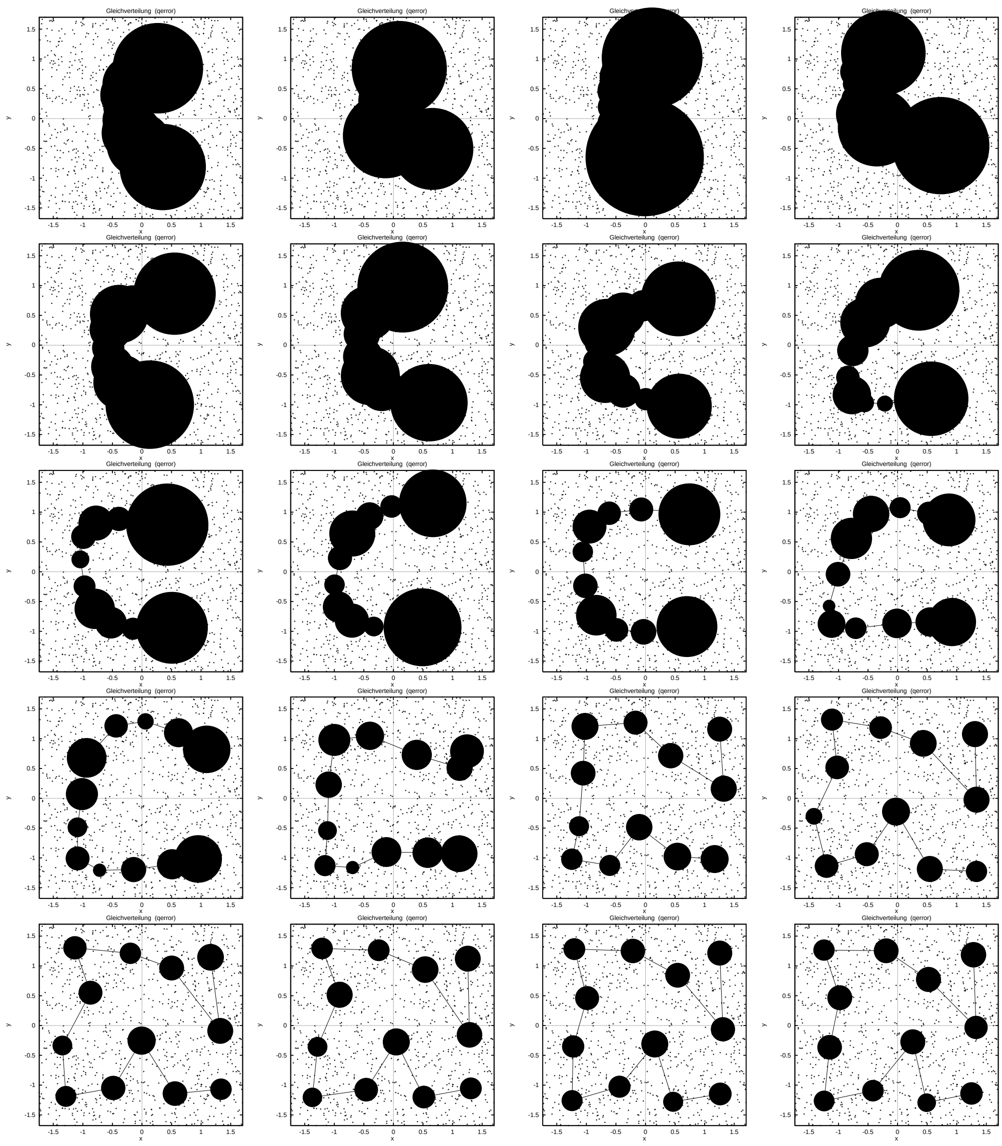


Abbildung 7.4: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt $(300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren sind proportional zur Streuungsquadratsumme innerhalb der Klasse.

7.2 Variation der Parameter

Die Wahl der Parameter ist nicht trivial. Welche Effekte sich bei falsch gewählten Parametern ergeben und wie man sie abschätzt, soll in diesem Abschnitt am Fall der Gleichverteilung veranschaulicht werden. Die Werte beziehen sich auf das Beispiel im letzten Abschnitt.

7.2.1 Trainingsdauer und Lernrate

Die Aufteilung des Trainings in zwei Phasen ist ein Kompromiß, um ein schnelles und gutes Training durchzuführen. Die erste Phase erzeugt die Grobstruktur des Netzes.[4] Die zweite Phase soll für das Erreichen eines lokalen Minimums sorgen; es ist das „Feintuning“ der Gewichte.

Die Anzahl der Trainingsschritte T einer Phase richtet sich nach der Lernrate α, der Anzahl der Daten N und der Anzahl der Neuronen K. Halbiert man die Lernrate, um z. B. einen stabileren Lernvorgang zu erreichen (der Fehler variiert weniger und der Einfluß der Reihenfolge der Eingabevektoren ist nicht so stark, besonders bei kleinen N/K-Verhältnis), sollte die Anzahl der Trainingsschritte verdoppelt werden. Das läßt sich am besten mit Energiepaketen veranschaulichen: Bei jeden Lernschritt wird das Netz durch ein kleines Energiepaket (α) geändert. Die aufgewendete Gesamtenergie sollte konstant sein, wenn der gleiche Gesamteffekt erreicht werden soll.

In Bezug auf die Anzahl von Daten und Neuronen kann man annehmen:

$$\frac{T}{N} \geq 1 \tag{7.1}$$

$$\frac{T}{K} \geq m \quad \text{(Schwellwert)} \tag{7.2}$$

Diese Formeln bedeuten, daß jedes Gewicht mindestens m-mal geändert wird und daß jeder Datenvektor mindestens einmal in den Lernvorgang einfließt. Der Schwellwert ist von Fall zu Fall verschieden und von den anderen Parametern abhängig. Diese Ungleichungen sind nur Faustregeln, die als Anhalt dienen, wenn man verschiedene Trainingsläufe mit ungefähr den gleichen Daten durchführen will.

Anmerkung:
Aus Konvergenzgründen wäre es besser, in Phase 2 die Lernrate exponentiell

[4] Dies bezieht sich hier auf die Verteilung der Gewichte. Das Netz im Topologieraum ist unverändert.

und nicht linear abnehmen zu lassen. Noch besser wäre es, einen K-Means-Algorithmus einzusetzen, da dieser monoton gegen ein Minimum konvergiert. Dem Trainingsverhalten in Abb. 7.2 ist aber zu entnehmen, daß die zweite Phase keine große Verbesserung des Fehlers bringt. Sie ist eben das „Feintuning“. Dies ist um so erstaunlicher, da in Phase 1 die Lernrate mit 0.1 relativ groß ist.

Abbildungen 7.5 bis 7.8 zeigen exemplarisch Trainingsläufe mit Variationen von Lernrate und Trainingsdauer. In Hinblick auf die Klasseneinteilung ergibt jedes Training eine ähnliche Einteilung wie Bild 7.1. Sie sind jedoch nicht gleich, da es im Falle der Gleichverteilung viele verschiedene Klasseneinteilungen gibt, die einen fast absolut minimalen Fehler besitzen.

Wie sich in den Beispielen zeigt, ist die Lernrate nicht allzu kritisch und kann in einem großen Bereich festgelegt werden. Für die Trainingsdauer sollten mehrere Werte versucht werden. Eine längere Trainingsdauer führt im Prinzip nur zu besseren oder gleichguten Ergebnissen.

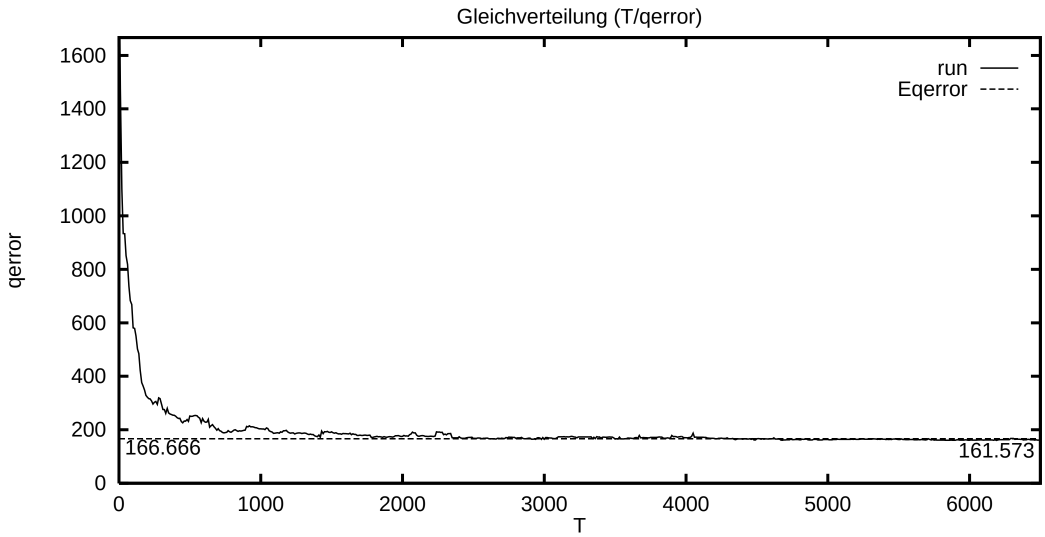


Abbildung 7.5: Abhängigkeit des Endfehlers von der Gesamttrainingsdauer. Durch verschiedene Trainingsläufe mit steigender Gesamttrainingsdauer T entstand dieses Diagramm. (Verhältnis Phase 1 zu Phase 2 ist 5:1; T ist die Summe aus beiden.) Man sieht, daß der Fehler schon ab $T = 1000$ dem absoluten Minimum nahe ist, wenn man beachtet, daß es nur 1000 Eingabevektoren gibt. Man erkennt, daß mit $T = 6000$ im Eingangsbeispiel die Trainingdauer reichlich groß gewählt ist.
Anmerkung: Bitte diese Art von Diagramm (T als Abszisse) nicht mit den meisten anderen Diagrammen („iterations" als Abszisse) verwechseln. Hier wurden hunderte von Trainingsläufen durchgeführt (Parameter wie Tabelle 7.1, nur T wird variiert) und der *Endfehler* aufgetragen; in den anderen Diagrammen nur *ein* Durchlauf mit regelmäßiger Fehlerberechnung.

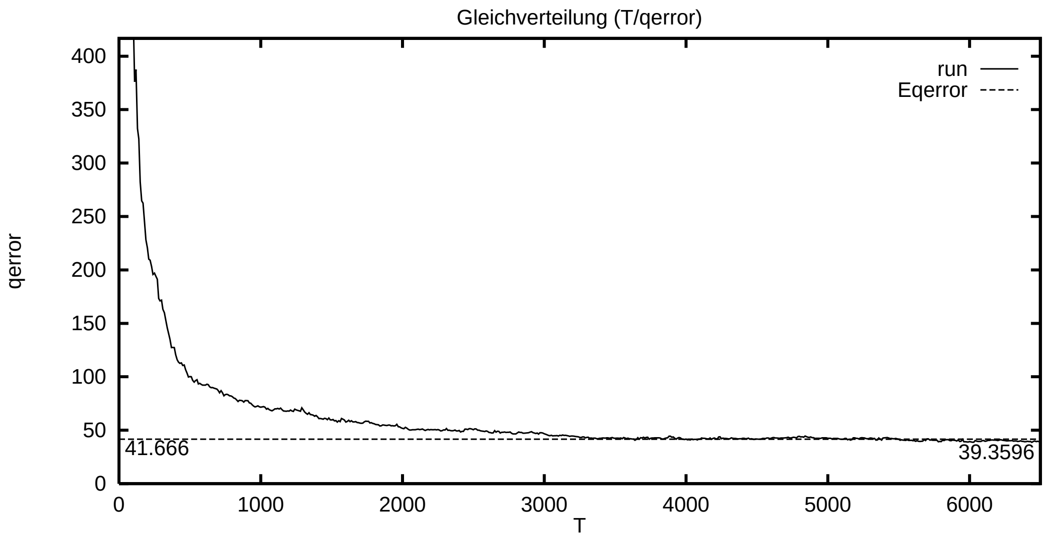


Abbildung 7.6: Abhängigkeit des Endfehlers von der Gesamttrainingsdauer.
Wie Bild 7.5, jedoch werden statt wie bisher 12 Neuronen nun 48 Neuronen verwendet. Der Fehler ist erst ab 4000 Schritten fast minimal. Im Normalfall wird man aber bei einer Erhöhung der Neuronenanzahl auch den Nachbarschaftsradius erhöhen (damit der Gesamteinfluß der Nachbarschaft gleich bleibt), so daß auch eine kürzere Trainingsdauer gute Ergebnisse bringt.

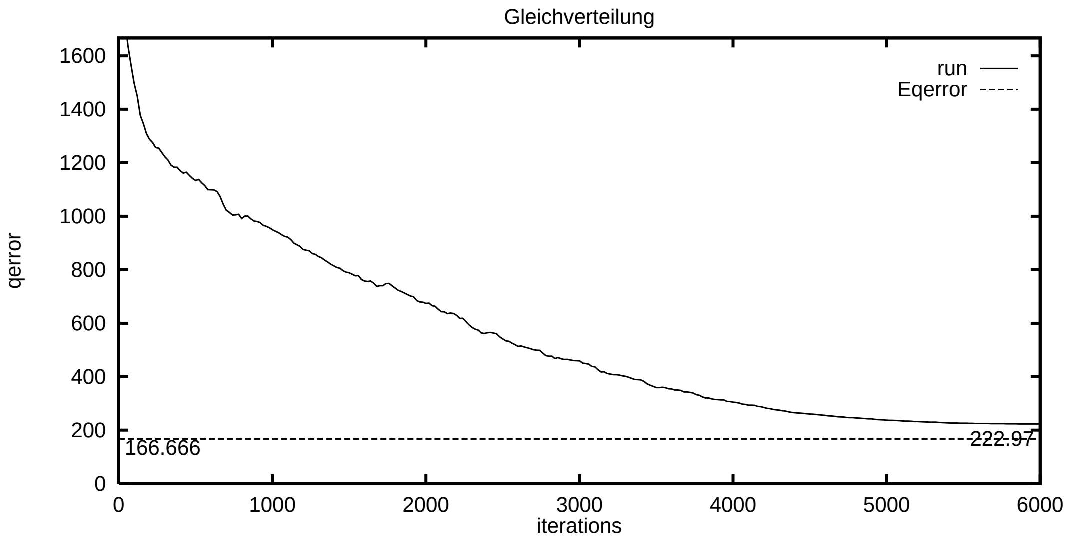


Abbildung 7.7: Trainingslauf mit $\alpha = 0.01$.
Der Fehlerverlauf ist glatter, jedoch wird das Minimum nicht erreicht. Das Netz hat nicht genug „Energie", um sich den geänderten Bedingungen durch die Abnahme des Nachbarschaftsradius anzupassen.

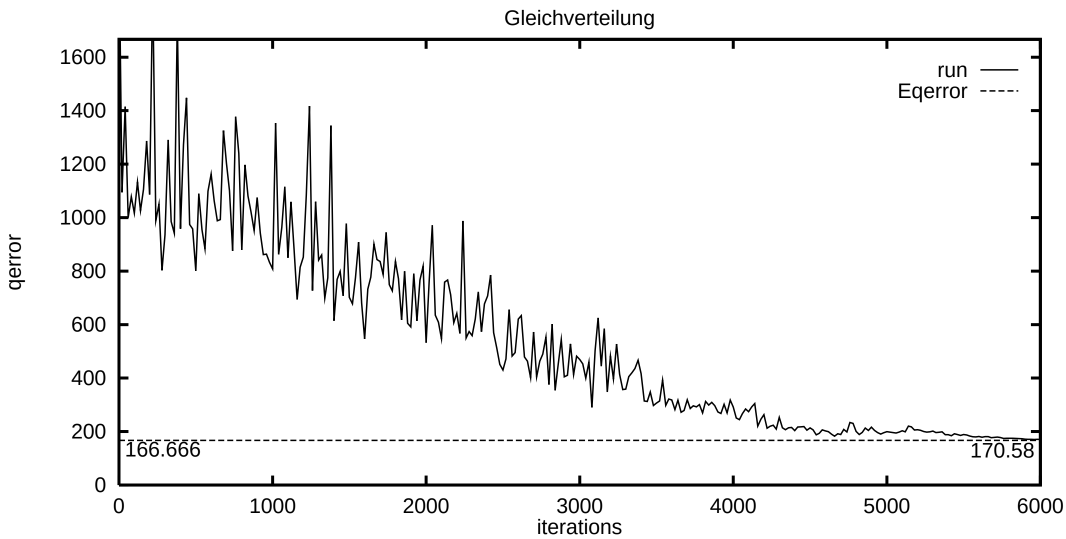


Abbildung 7.8: Trainingslauf mit $\alpha = 0.4$.
Der Fehler schwankt sehr stark. Daß er am Ende trotzdem relativ gut ist, liegt an der langen Trainingsdauer.

7.2.2 Nachbarschaftsradius

Der Nachbarschaftsradius (nur Phase 1) ist eng mit der Anzahl der Neuronen verknüpft. Wählt man ihn zu klein, dann kann sich das Netz zu Beginn der Trainings nicht richtig entfalten, so daß das Netz sich selbst durchdringt (höherdimensionale Netztopologien) oder kreuzt (Kette). Solche deformierten Netze haben im Normalfall einen großen Fehler. Ausnahme ist die Topologie „Neuronales Gas“, da bei dieser Netzart keine feste Nachbarschaftsstruktur definiert ist.

Als Faustregel hat sich ergeben: Setze den Nachbarschaftsradius zu Beginn gleich einem Drittel der längsten Kantenlänge des Topologieraumes.

Beispiele für die Entwicklung des Fehlers bei einem großen und einem kleinen Anfangswert für den Nachbarschaftsradius sieht man in den Abbildungen 7.9 und 7.10. Ein zu großer Wert kann durch eine längere Trainingsdauer ausgeglichen werden. Ein zu kleiner Wert kann sich dagegen auf den gesamten Trainingsverlauf negativ auswirken.[5]

Welche Nachbarschaftsfunktion (Gauß oder Bubble) besser ist, läßt sich nicht beantworten. Teuvo Kohonen verwendet vorzugsweise die Bubble-Nachbarschaftsfunktion, da dann bestimmte mathematische Betrachtungen der Konvergenz möglich sind. Die Gauß-Nachbarschaft hat den Vorteil einer stetigen Änderung des Nachbarschaftseinflusses, bei Bubble sind die Nachbarschaftseinflüsse immer phasenweise konstant. Dies führt zu einer ungefähr stufenförmiger Abnahme des Fehlers (Abbildung 7.11).

[5]Dieser Effekt trat bei dem Trainingslauf für Abbildung 7.10 nicht auf. Erst bei höherdimensionalen Netztopologien, bei schlechteren Startwerten für die Gewichte oder anderen Datenvektoren wird dieser Effekt sichtbar.

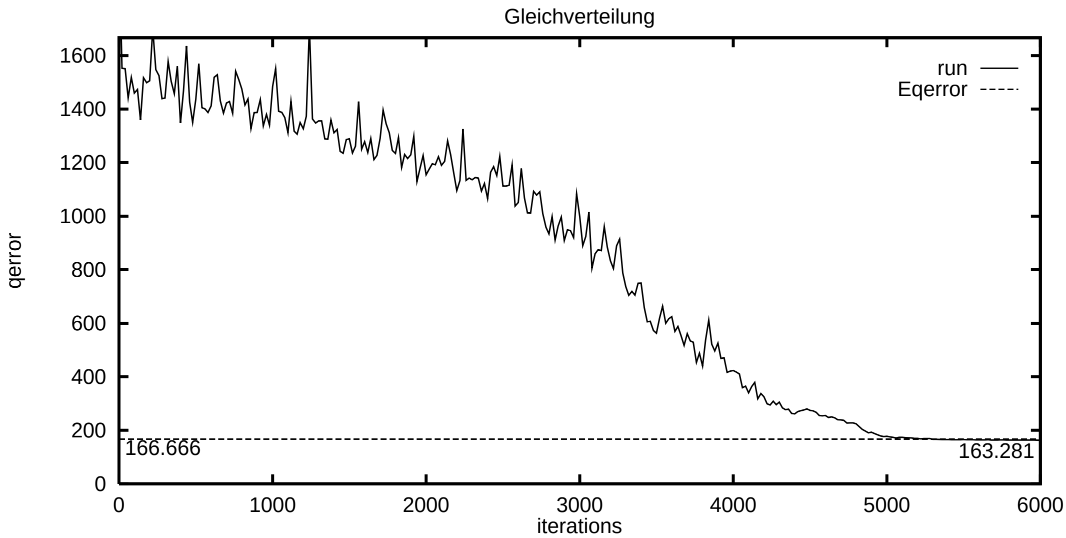


Abbildung 7.9: Trainingslauf mit $R_{\text{start}} = 8$.
Im Vergleich zu Abbildung 7.2 sinkt der Fehler langsamer. Bei einer kürzeren Trainingsdauer kann dies zu einer schlechteren Klasseneinteilung führen.

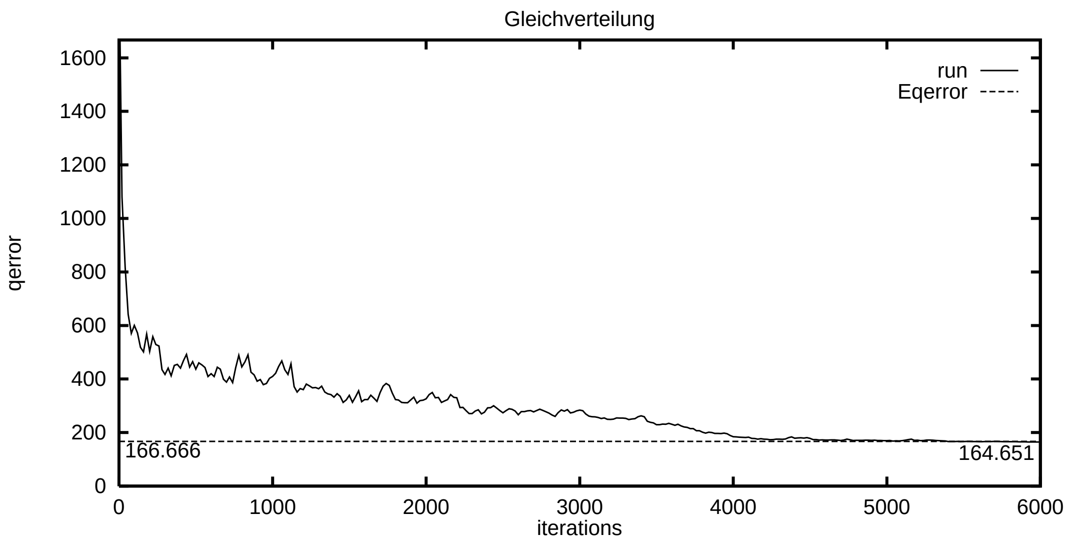


Abbildung 7.10: Trainingslauf mit $R_{\text{start}} = 2$.
Im Vergleich zu Abbildung 7.2 ist zwar eine schnellere Fehlerverringerung zu verzeichnen, jedoch kann der Endfehler bei anderen Verteilungen der Eingabevektoren in einem schlechteren Minimum liegen.

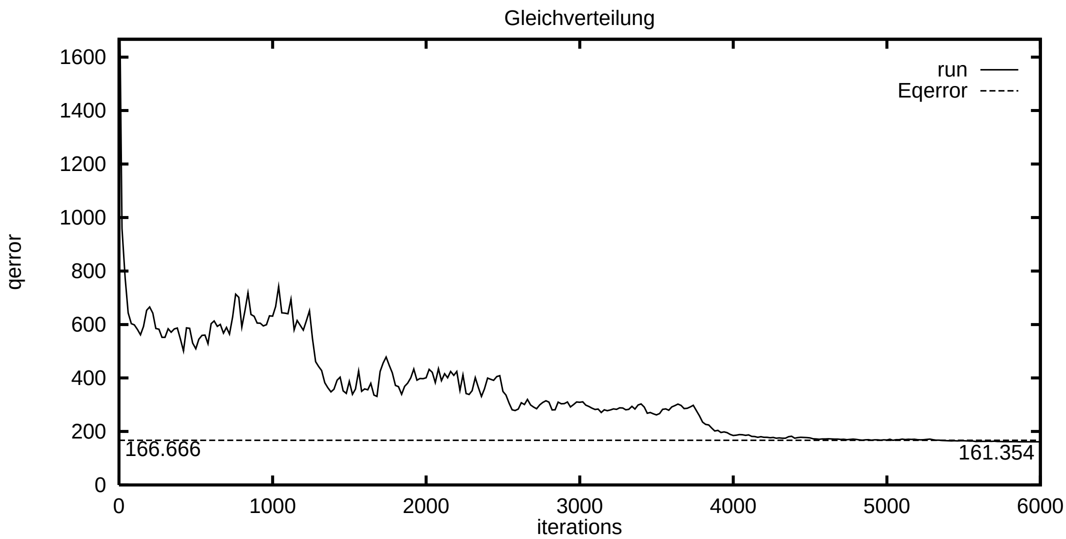


Abbildung 7.11: Trainingslauf mit Bubble-Nachbarschaft $R = 4$.
Der Fehler nimmt stufenweise bei jeder Änderung des Ganzzahlanteils des Nachbarschaftsradius ab. (Bis 1250 ist $R > 3$, bis 2500 ist $R > 2$, bis 3750 ist $R > 1$ und ab 3750 gilt $1 > R > 0$.) Nur an diesen Stellen ändert sich der effektive Einfluß der Nachbarschaft. In den Bereichen dazwischen ist die Nachbarschaft konstant, da im Topologieraum immer die gleichen Neuronen innerhalb der „Nachbarschaftsblase“ liegen. Der Fehlerverlauf ist flacher, da der Nachbarschaftsradius R bei Verwendung der Bubble-Funktion nicht den gleichen Einfluß hat wie bei der Gauß-Funktion. Man müßte für die Bubble-Funktion den Nachbarschaftsradius ungefähr doppelt so groß wählen wie für Gauß, möchte man eine quantitativ ähnliche Fehlerkurve erhalten.

7.2.3 Nachbarschaftstopologie

Die Nachbarschaftstopologie hat wohl den am schwierigsten zu bestimmenden Einfluß auf das Training und damit auf die Güte des Klasseneinteilung. Die Topologie kann durch die Problemstellung gegeben sein, wenn beispielsweise ein hochdimensionaler Raum zur Visualisierung auf zwei Dimensionen abgebildet werden soll. Die Nachbarschaft ist also auch eine erwünschte Eigenschaft. Soll nur der Fehler minimiert werden, liegt die Wahl der Nachbarschaftstopologie beim Anwender.

Bei dem originalen Kohonen-Netzmodell liegt die Topologie als zweidimensionales Gitter vor. Man kann dieses auf beliebig viele Dimensionen erweitern. Welche Dimensionalität optimal ist, hängt von den verwendeten Eingabedaten ab.

Diese Entscheidung muß man beim „Neuronalen Gas“ nicht treffen. Ein Nachteil vom Neuronalen Gas ist die etwas höhere Rechenzeit für die dynamische Bestimmung der Nachbarschaft.

Bei den nachfolgenden Versuchen (Abbildungen 7.12 bis 7.17) mit verschiedenen Topologien (Ring, Netz und Neuronales Gas) zeigt sich, daß sie (zumindest in diesem Beispielfall[6]) keinen größeren Einfluß auf den *Fehler* des Endergebnisses haben.[7] Das etwas andere Aussehen der Fehlerkurven ergibt sich durch den unterschiedlichen Einfluß des Nachbarschaftsradius.

Interessant ist bei der Topologie „Kette“, daß die beiden äußersten Neuronen während des Trainings einen größeren Fehler als die anderen Neuronen besitzen (siehe Abbildung 7.4). Ist im fertig trainierten Netz der Fehler immer noch größer, kann dies ein Hinweis auf eine zu kurze Trainingsdauer sein.

[6]Bilden die Datenvektoren *echte* Cluster, wird das Neuronale Gas bessere Ergebnisse liefern [Martinetz et al. 1993].

[7]Die Parameter für das Training sind aus der Tabelle 7.1 zu entnehmen, wenn nicht anders angegeben.

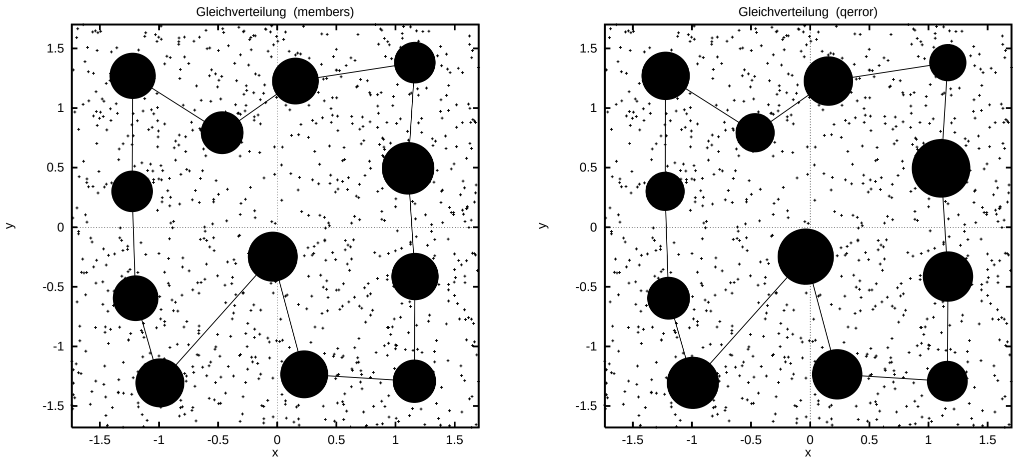


Abbildung 7.12: Eingabevektoren und Neuronen (Ringtopologie).
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.)

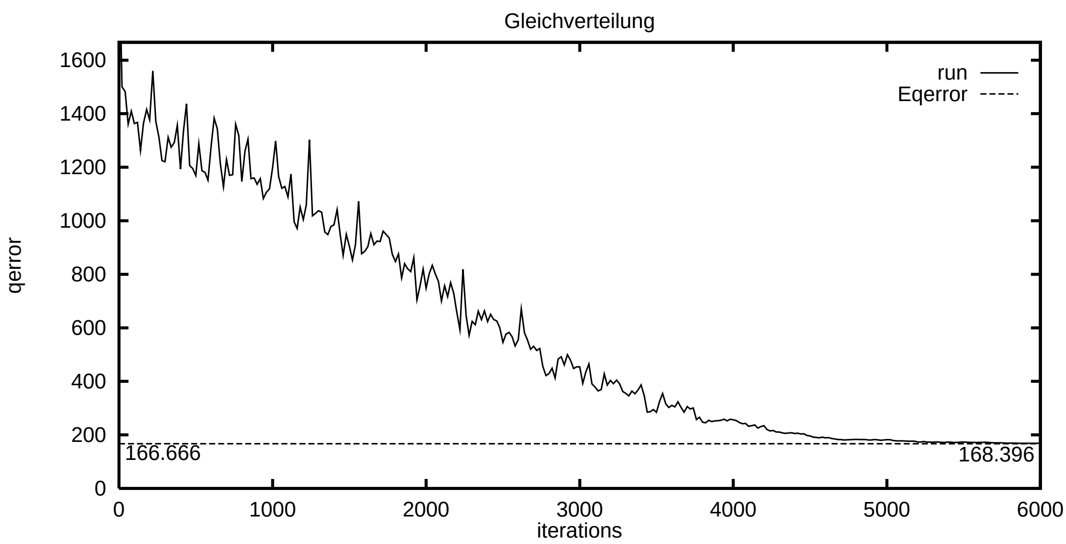


Abbildung 7.13: Trainingsverhalten.

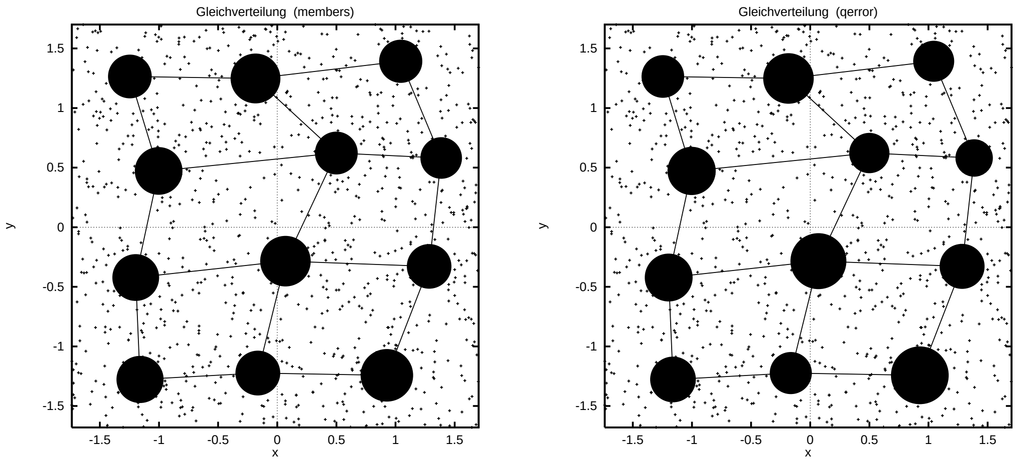


Abbildung 7.14: Eingabevektoren und Neuronen (zweidimensionales Gitter). Das Gitter hat die Maße 4x3, der Nachbarschaftsradius $R = 2$. (Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.)

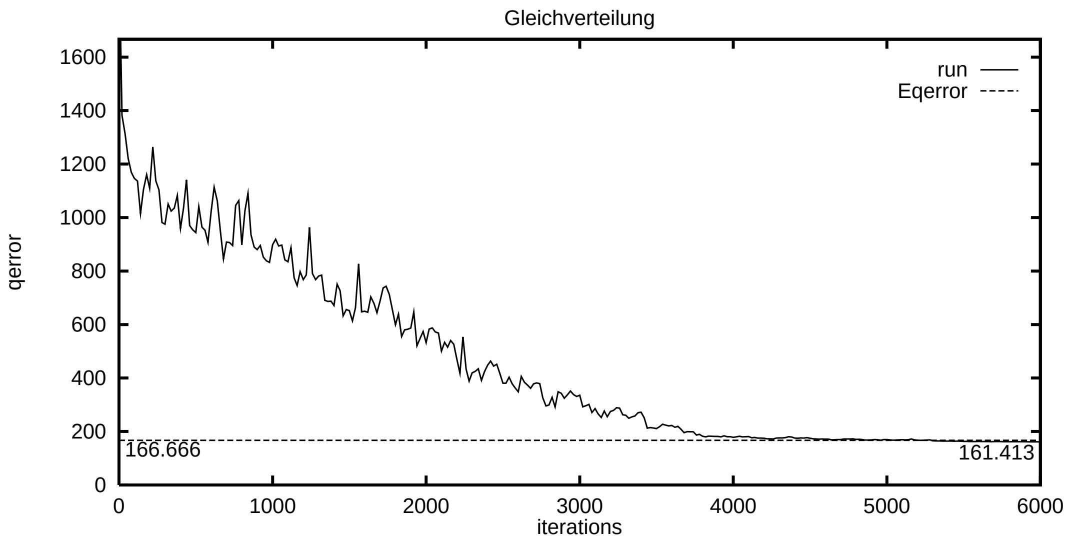


Abbildung 7.15: Trainingsverhalten.

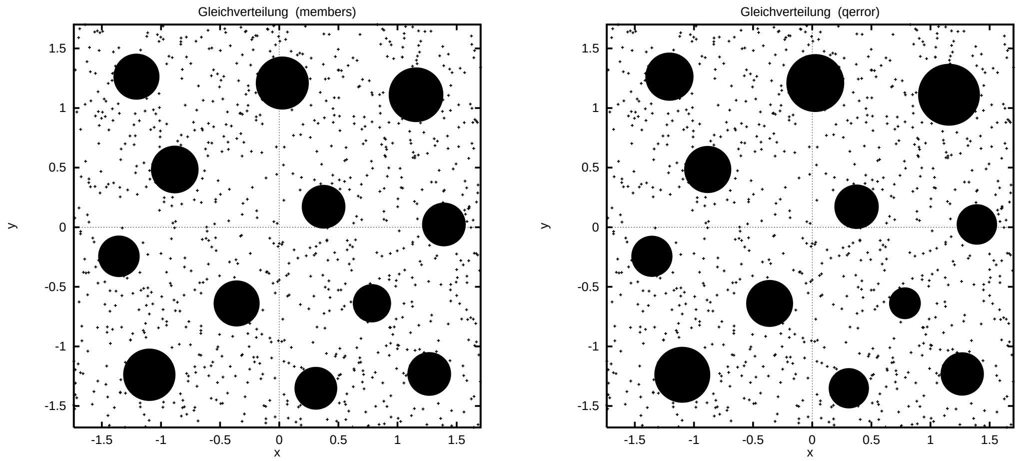


Abbildung 7.16: Eingabevektoren und Neuronen (Neuronales Gas).
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.)

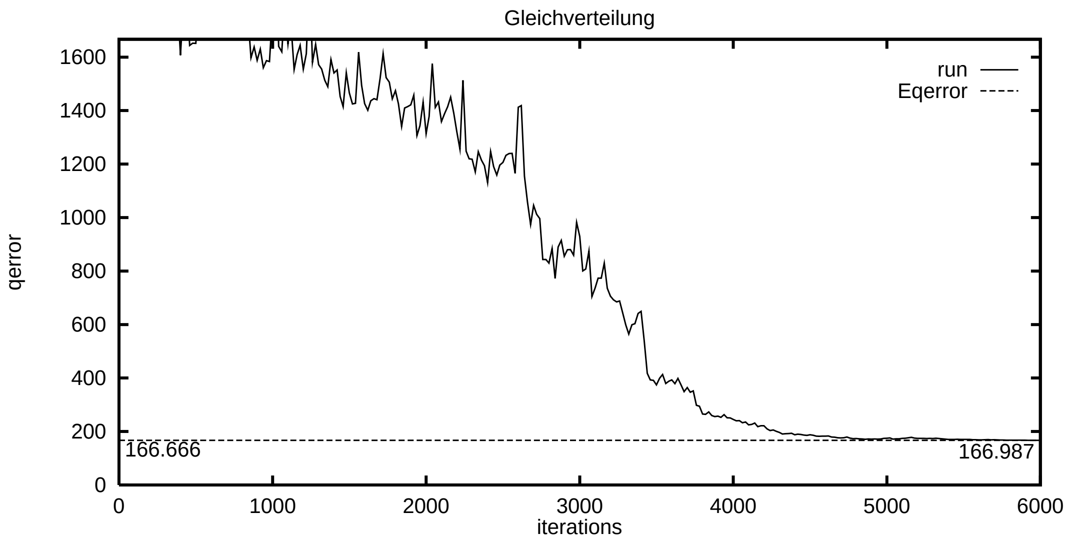


Abbildung 7.17: Trainingsverhalten.

7.2.4 Reihenfolge der Daten

Für jeden Trainingsschritt muß ein Eingabevektor ausgewählt werden. Wählt man diese immer in der gleichen Reihenfolge aus, sind im Fehlerdiagramm sich periodisch wiederholende Schwankungen erkennbar (Periodenlänge gleich N, siehe Abbildung 7.18). Ist die Reihenfolge schlecht gewählt, beispielsweise so, daß die Daten schon ungefähr in Klassen vorsortiert sind,[8] dann erzeugt des Netz eine insgesamt schlechte Klasseneinteilung.

Wählt man jedoch die Vektoren zufällig aus der Gesamtmenge aller Datenvektoren, kann man diesen Effekt vermeiden. Dies hat aber programmtechnische Nachteile für die Verwaltung großer Datenmengen.

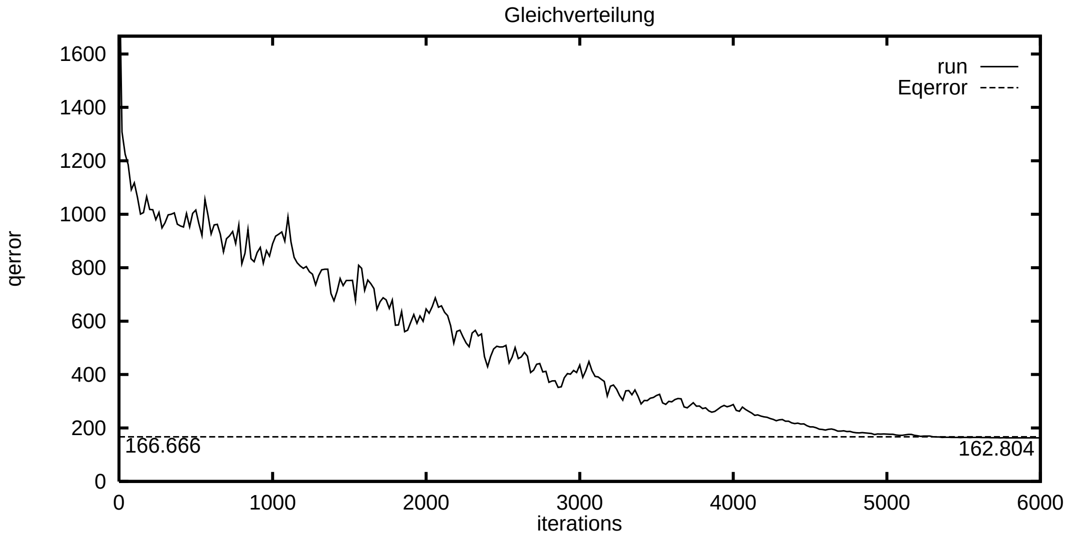


Abbildung 7.18: Trainingsverhalten mit sequentieller Auswahl.
Die 1000 Eingabevektoren wurden in Phase 1 immer in der gleichen Reihenfolge für das Training ausgewählt. Für Phase 2 gilt (unverändert) dasselbe.

7.2.5 Anfangswerte

Die Gewichte müssen mit Anfangswerten vorbelegt werden. Diese können bei ungünstiger Wahl der anderen Parameter einen Einfluß auf das Endergebnis ha-

[8]Beispiel: Messung von Tagestemperaturen über einen Zeitraum von Frühjahr bis Winter. Trainiert man das Netz mit den Daten in der gleichen Reihenfolge wie gemessen wurde, dann werden zwar die letzten Temperaturen (tiefe Temperaturen) gut in Klassen aufgeteilt, die Klassen für die Sommertemperaturen sind jedoch unterrepräsentiert. Dieser Effekt des „Vergessens" oder „Anpassens" kann aber bei anderen Anwendungen auch nützlich sein.

ben. Wählt man zu Beginn einen großen Nachbarschaftsradius und eine ausreichende Lernrate, kann man aber diesen Einfluß fast ausschließen, da das Netz sich dadurch (anfangs) sehr stark ändert: Das Netz ist sehr kompakt und schwankt um den Schwerpunkt der Eingabevektoren, egal mit welchen Werten es initialisiert wurde. Deswegen ist es empfehlenswert, die Gewichte mit dem Schwerpunkt der Daten zu initialisieren.

7.3 Gauß-Verteilung

Nun kann man alle Parametervariationen an verschiedenen Eingabedaten ausprobieren. Das Resümee ist aber in etwa gleich. Darum möchte ich hier nur noch kurz auf die zweidimensionale Gaußverteilung eingehen. Die Gaußverteilung ist deswegen erwähnenswert, da viele physikalische Daten so verteilt sind.

7.3.1 Ein weiteres Trainingsbeispiel

Analog zu Abschnitt 7.1 ergeben sich Abbildungen 7.19 bis 7.22. Der eingezeichnete Referenzfehler entspricht in diesem Falle nicht dem erreichbaren Optimum. Dies liegt daran, daß die Formel für den Referenzfehler eigentlich nur gilt, wenn die Eingabevektoren in gleichgroße Quadrate eingeteilt werden können. Wollte man trotzdem einen besseren Referenzfehler berechnen, müßte man zumindest die Kantenlänge a in Formel (3.5) besser abschätzen.[9]

7.3.2 Skalierung

Der Effekt der Skalierung wurde noch nicht verdeutlicht. Deswegen sind in Abbildungen 7.23 und 7.24 der Fall einer Streckung der Gaußverteilung um Faktor 2 dargestellt. Es ergibt sich im Vergleich zur normalen Gaußverteilung eine andere Verteilung der Gewichte und damit eine andere Aufteilung der Daten in Klassen, auch wenn man die Klassen wieder zurückskaliert. Bei solchen Rückskalierungen von Gewichten und Daten muß man beachten, daß das Kriterium zur Klassenaufteilung (Formel 3.1) *vor* der Rückskalierung andere Klassen definiert als *nach* der Skalierung.

[9] In den Grafiken wird für die automatische Berechnung des Referenzfehlers die Kantenlänge a als Funktion des „Volumens“ berechnet. Dieses Volumen wird durch den Ansatz „$Q^{(\mathrm{eq})}(K = 1) =$ Streuungsquadratsumme zum Schwerpunkt“ ermittelt.

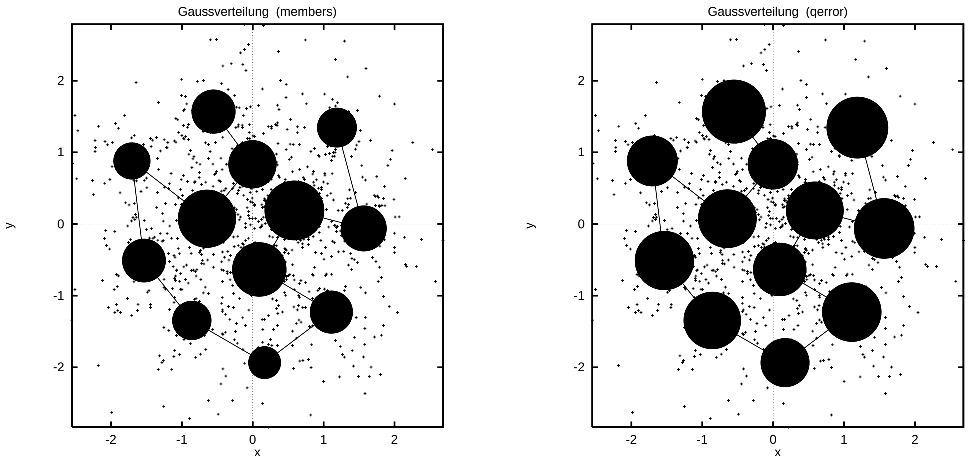


Abbildung 7.19: Eingabevektoren und Neuronen (Gaußverteilung).
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.)

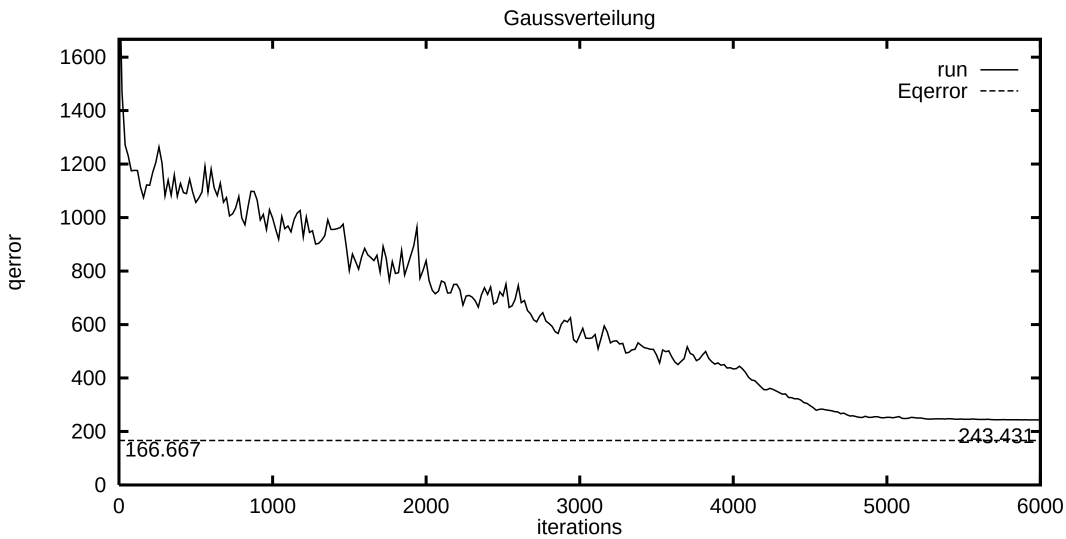


Abbildung 7.20: Trainingsverhalten.

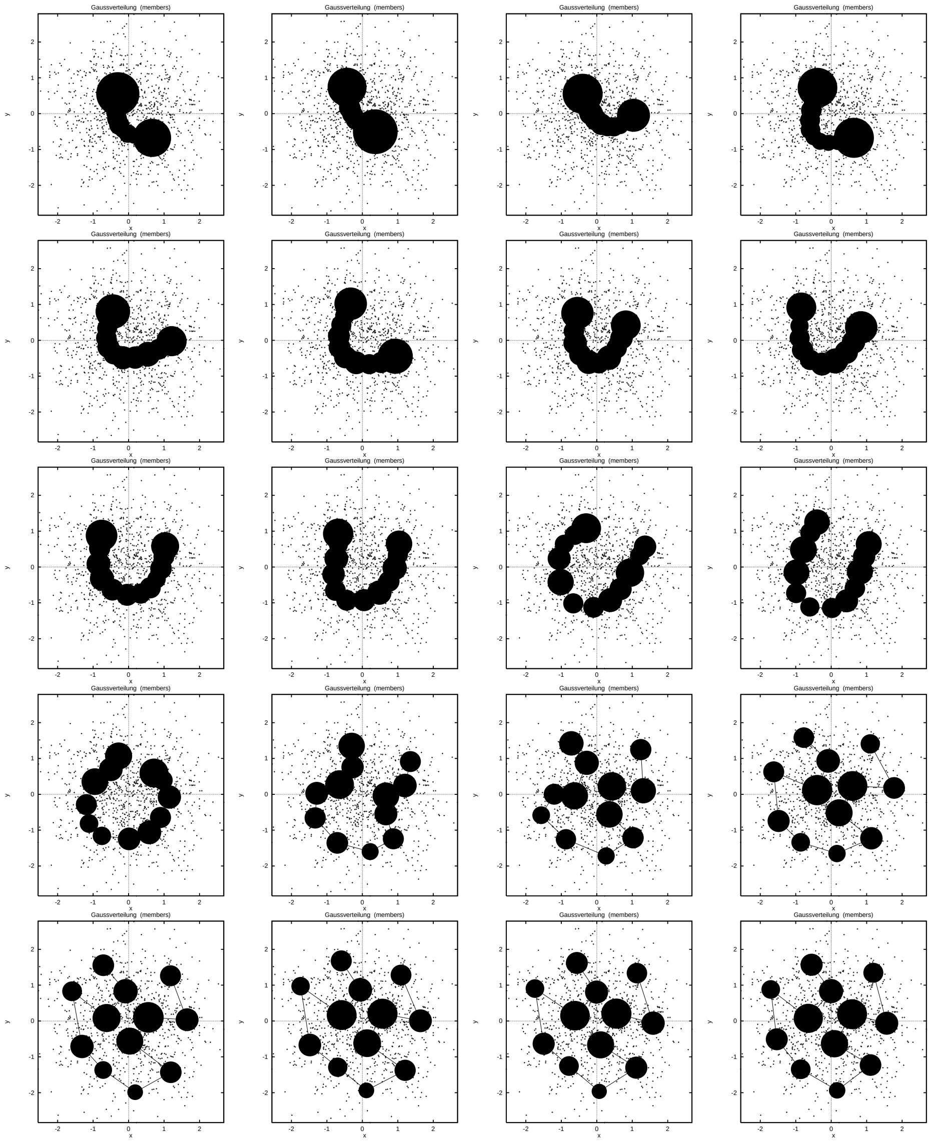


Abbildung 7.21: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt $(300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren sind proportional zur Anzahl der Klassenmitglieder.

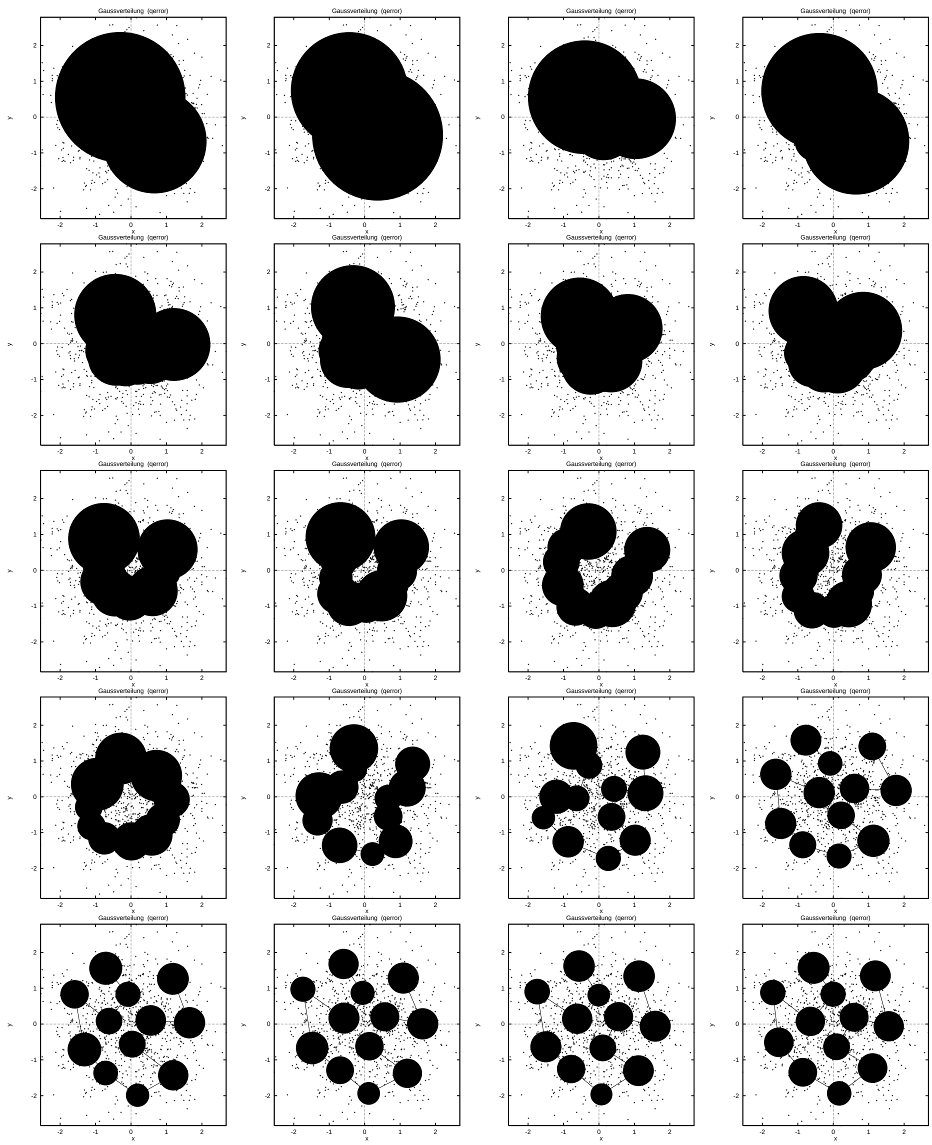


Abbildung 7.22: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt $(300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren sind proportional zur Streuungsquadratsumme innerhalb der Klasse.

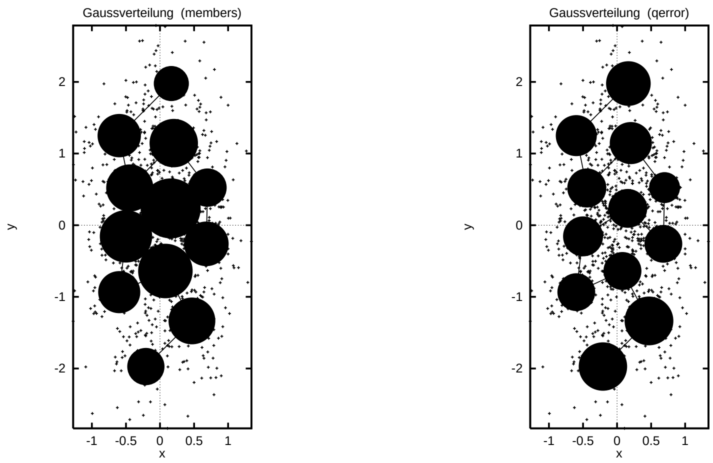


Abbildung 7.23: Eingabevektoren und Neuronen einer verzerrten Gaußverteilung. (Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.) Man sieht, daß sich nun die Neuronen mehr in Richtung der längsten Ausdehnung verteilen. In diese Richtung gibt es dann eine feinere Klasseneinteilung.

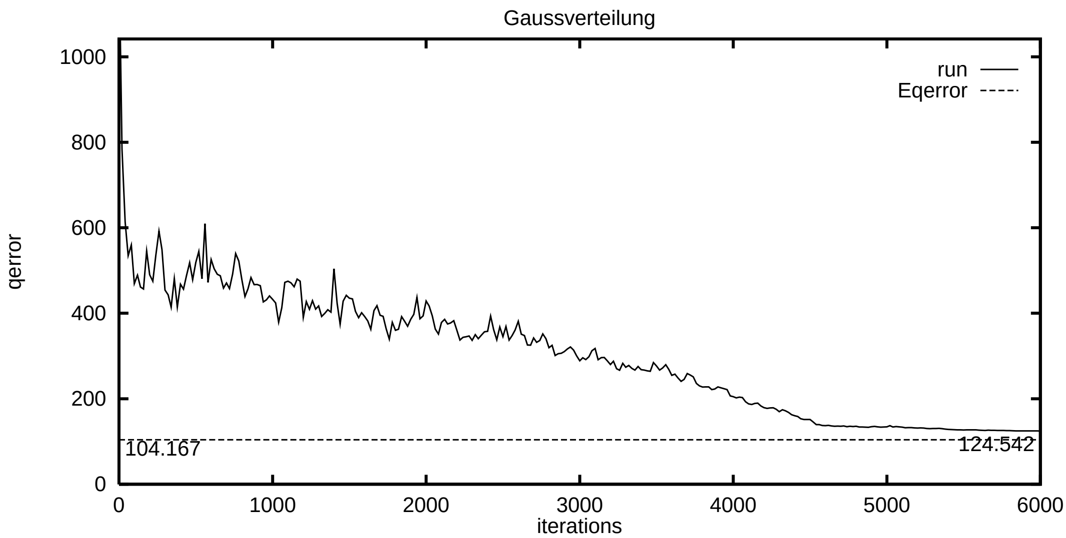


Abbildung 7.24: Trainingsverhalten.

7.4 Eine Verteilung mit vier separaten Clustern

Nachdem in den beiden gezeigten Beispielen „Gleichverteilung“ und „Gaußverteilung“ die Eingabevektoren in einem geschlossenen Gebiet liegen, soll als letzter künstlicher Testfall eine Beispiel mit vier gut getrennten Punktwolken untersucht werden (Abbildungen 7.25 bis 7.28).

Zwei Punkte sind dabei beachtenswert: Beim Training fällt auf, daß zwischen den Punktwolken Neuronen liegen, für die es fast keine Klassenmitglieder gibt. (In Abbildung 7.27 die sehr kleinen und leeren Kreise.) Ich bezeichne diese Neuronen gern als „Lückenbüßer“ oder „Füllneuronen“. Sie sind der Nachteil einer fest vorgegeben Topologie. Diese Neuronen werden von zwei Seiten ungefähr gleich stark beeinflußt. Damit bleiben diese Neuronen (nutzlos) an der gleichen Stelle und werden auch gegen Ende des Trainings nicht oder nur sehr schlecht in die Klasseneinteilung einbezogen.[10] Beim Neuronalen Gas ist dieses Problem nicht so stark ausgeprägt.

Der zweite interessante Aspekt ist in Abbildung 7.29 zu sehen. Probiert man verschiedene Anzahlen von Neuronen aus, gibt es eine Fehlerkurve. In ihr läßt sich die *echte Klassenanzahl* ablesen: Dort, wo der Fehler im Diagramm relativ stark abfällt und dann normal[11] weitersinkt, ist eine charakteristische Klassenanzahl. In diesem Beispiel ist dies bei $K = 4$, also genau der Anzahl der echten Punktwolken. Es kann in einem solchen Diagramm auch mehrere solcher Knicke geben.

[10] Siehe linke untere Punktwolke, drittes Neuron von unten in den Bildern. Solche Füllneuronen bilden sich oft auch bei anderen Verteilungen, wenn gut getrennte Cluster existieren.

[11] Mit „normal“ ist gemeint, daß das Kurvenstück fast proportional zu $F(K) = K^{-\frac{2}{\lambda}}$ ist. Im zweidimensionalen ist dies eine Hyperbel.

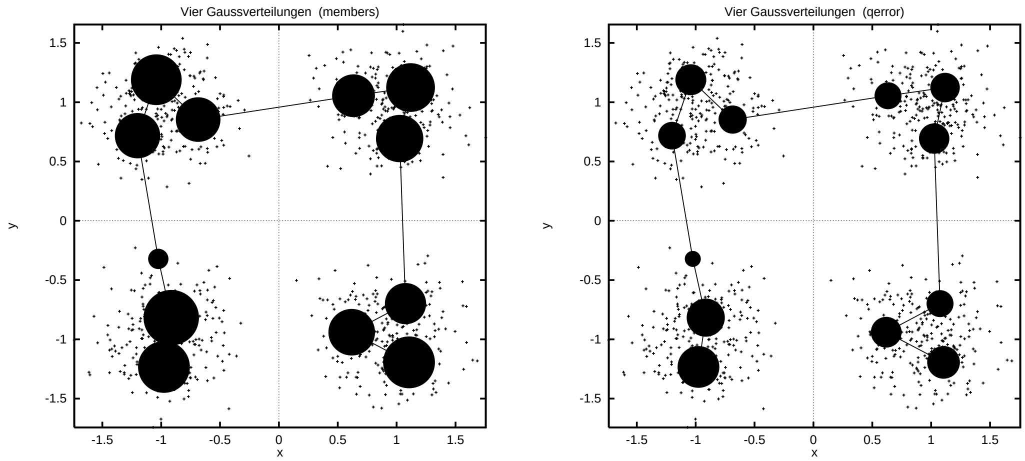


Abbildung 7.25: Eingabevektoren und Neuronen.
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.)

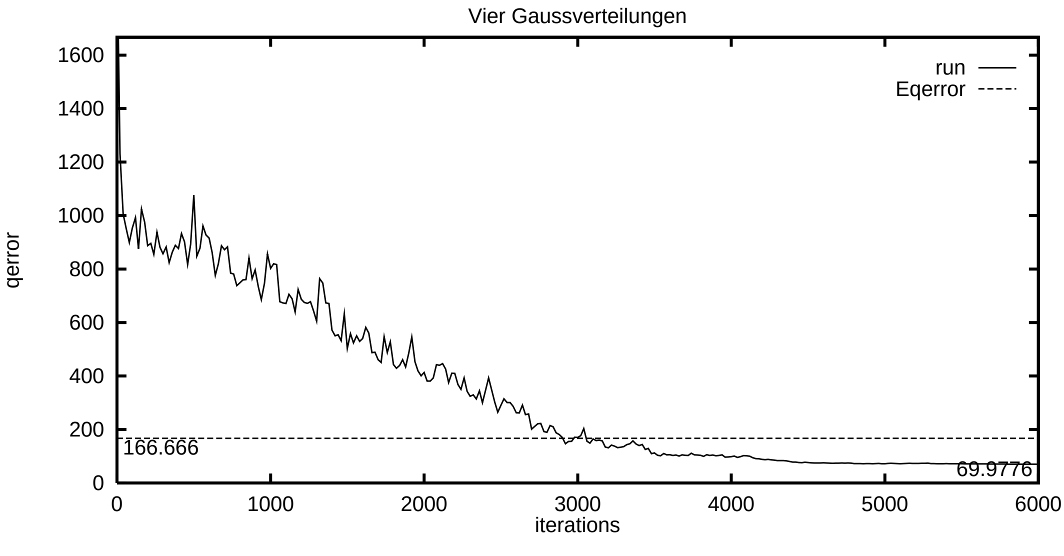


Abbildung 7.26: Trainingsverhalten.

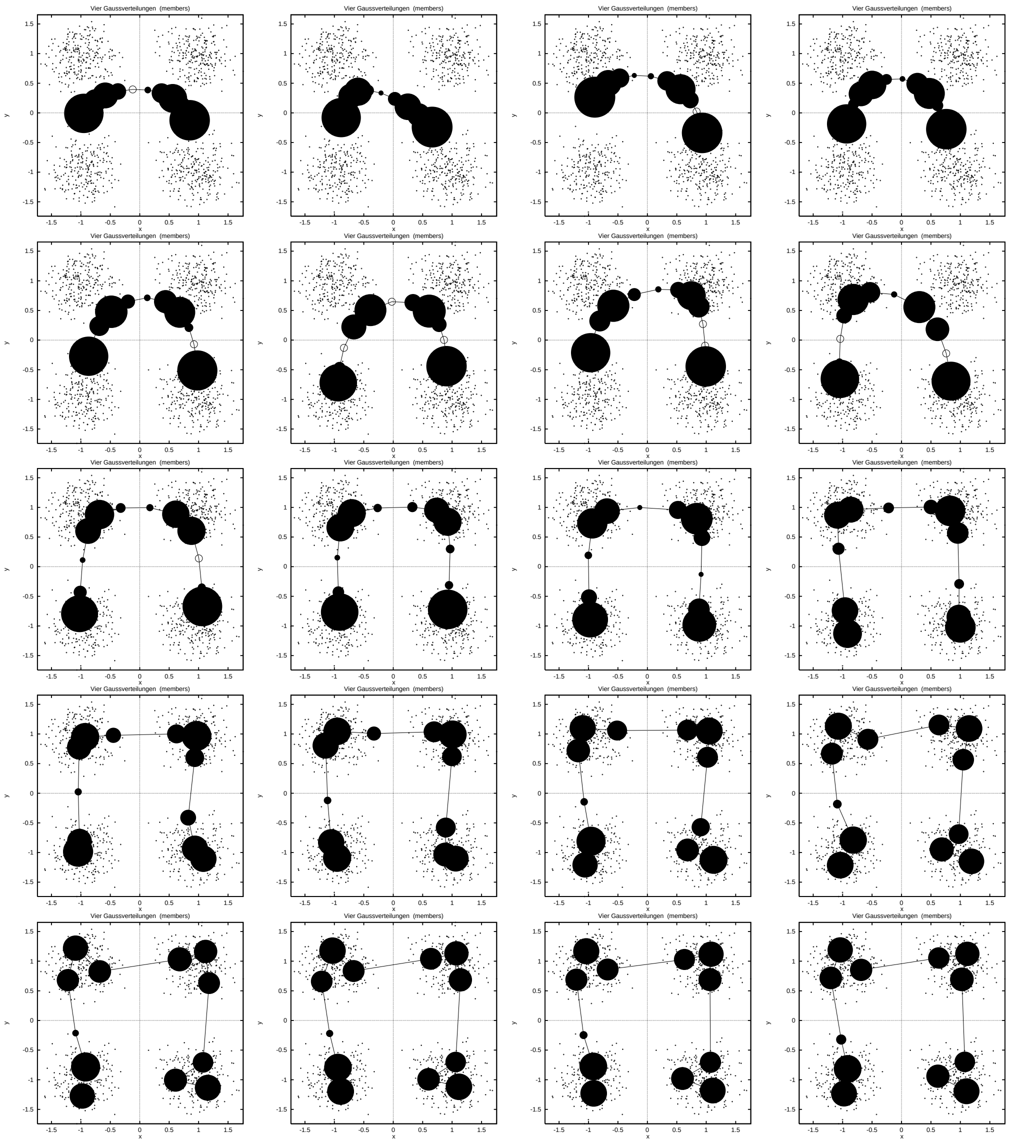


Abbildung 7.27: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt $(300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren sind proportional zur Anzahl der Klassenmitglieder. (Leere Kreise: die wirkliche Kreisfläche ist für die Darstellung zu klein.)

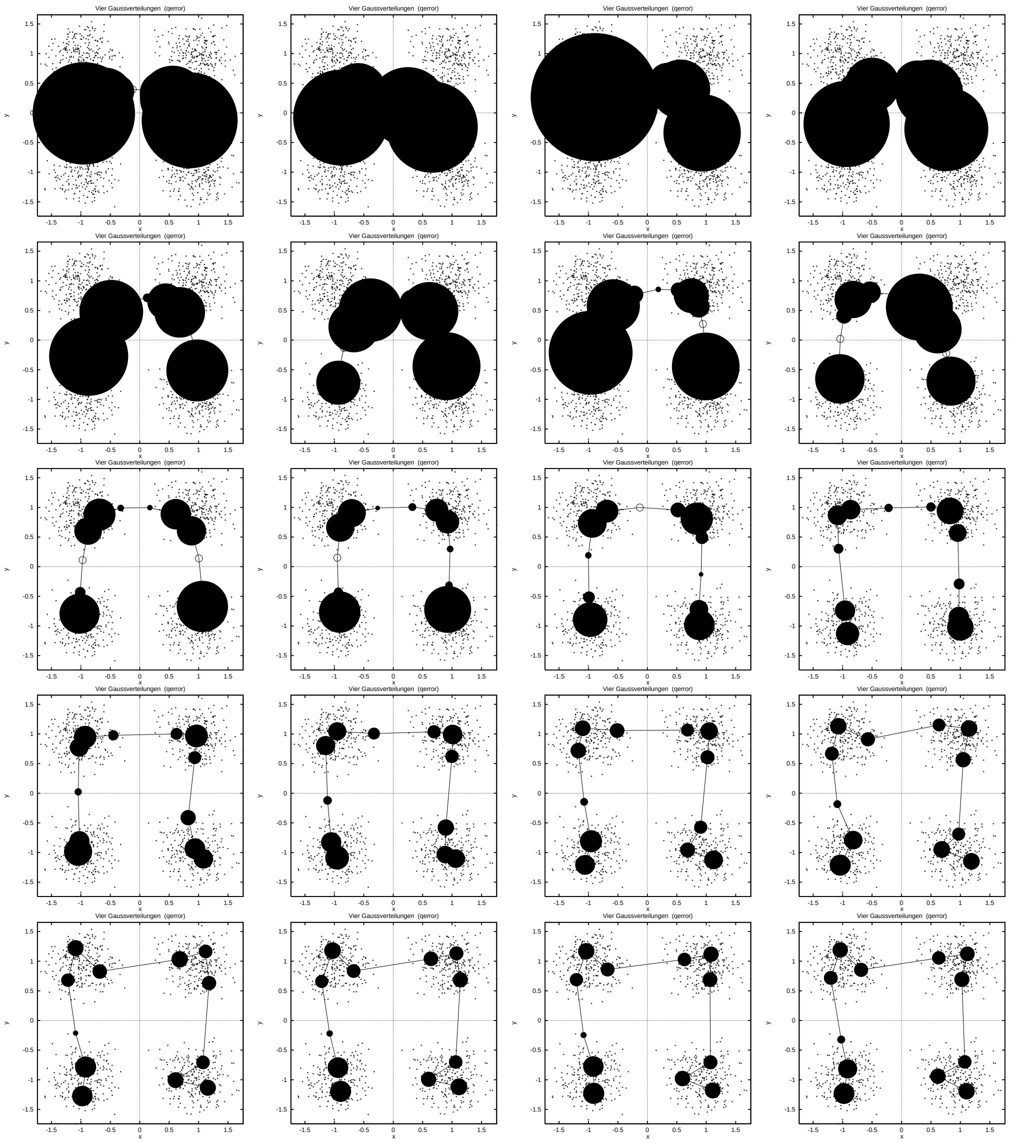


Abbildung 7.28: Entwicklung des Netzes.
Zur Veranschaulichung ist die Klasseneinteilung nach jeweils 300 Lernschritten dargestellt $(300, 600, \ldots, 6000$ Lernschritte). Die Kreisflächen um die jeweiligen Klassenzentren sind proportional zur Streuungsquadratsumme innerhalb der Klasse. (Leere Kreise: die wirkliche Kreisfläche ist für die Darstellung zu klein.)

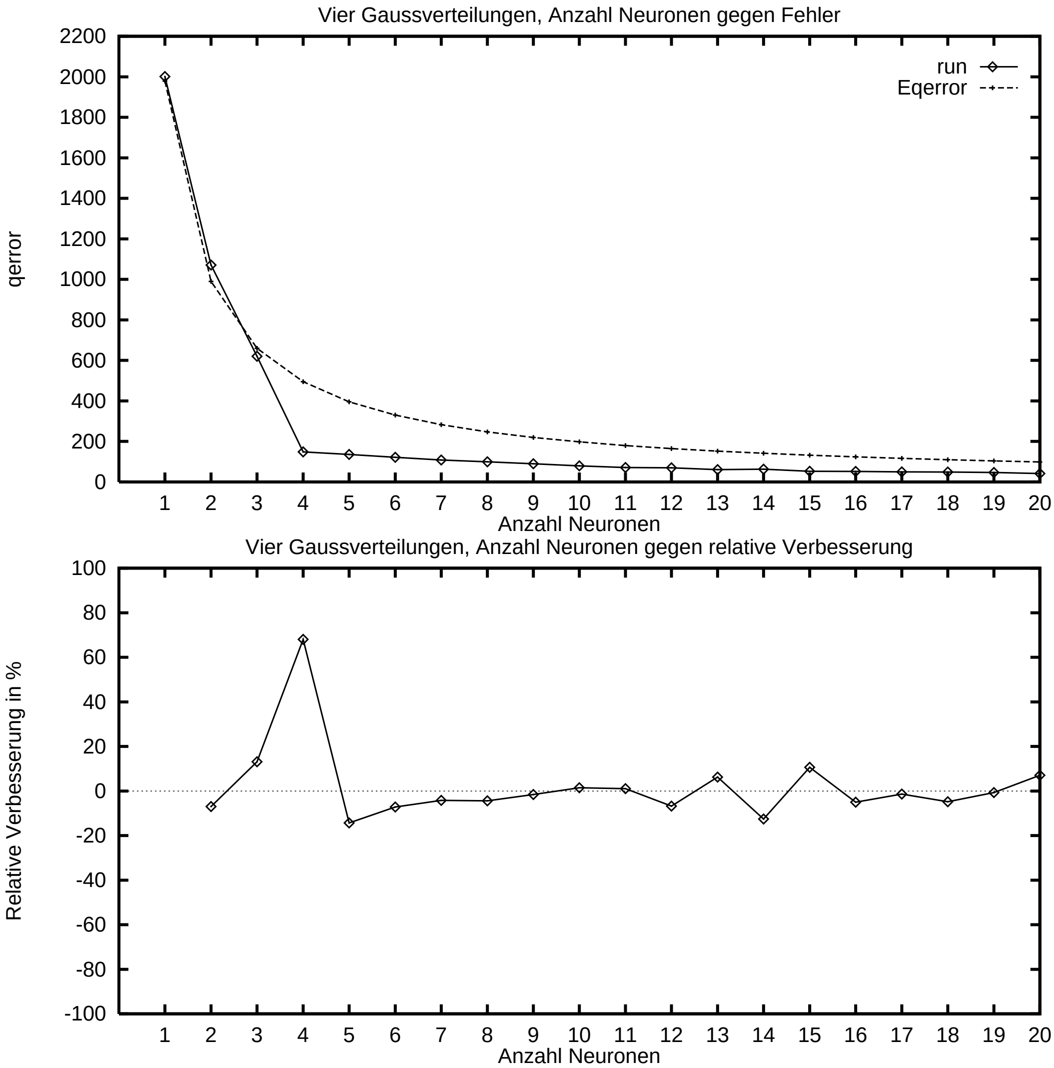


Abbildung 7.29: Abhängigkeit des Endfehlers von der Anzahl der Neuronen.
Im oberen Bild ist der Referenzfehler (Hyperbel) und der reale Fehler eingetragen. Man sieht einen starken Knick bei $K = 4$. Dieser Knick wird besser sichtbar, wenn man die Verbesserung die folgende Funktion $F(K)$ berechnet:
$f(K) = Q(K-1) \cdot \frac{K}{K-1}; F(K) = \frac{f(K)-Q(K)}{f(K)} \cdot 100; K \geq 2.$
Dabei ist $f(K)$ der absolute, aus $Q(K-1)$ geschätzte Fehler und $F(K)$ die relative Fehlerverbesserung in Prozent, welche auch im unteren Bild aufgetragen ist. $Q(K)$ ist der reale Fehler eines Trainingslaufs mit K Neuronen.

7.5 Wind-Daten

Nun folgen Beispiele, basierend auf dem Datenbestand des angezielten Hauptanwendungsproblems. Da der Wind in der einfachen (bisher eingesetzten) Klasseneinteilung schon in mehrere Klassen aufgeteilt wird, liegt es nahe, dies zu verfeinern. Will man die anderen Kriterien zur Klasseneinteilung gleich lassen, gibt es sechs Dateien (drei Niederschlagsklassen, jeweils für das Sommer- und Winterhalbjahr).

7.5.1 Wind

Der (horizontale) Wind ist ein zweidimensionaler Vektor. Eine Einteilung einer Datei (Winter, Tage mit konvektiven Niederschlag, Wind in 1500 m Höhe) in $K = 12$ Klassen sieht man in Abbildung 7.30. Die Daten sind einer Gaußverteilung ähnlich, jedoch ist der Schwerpunkt nicht im Nullpunkt. Es herrschen Westwindlagen vor, die in dem Bild rechts liegen, da der Windgeschwindigkeitsvektor eingetragen ist — Westwind ist der Wind in Richtung Osten —. Man sieht, daß die Klassen in der Mitte im Vergleich zu den außen liegenden Klassen einen kleineren Fehler und eine höhere Anzahl von Mitgliedern besitzen. Aber es lassen sich keine gut getrennten Klassen ausmachen. Diese Erkenntnis gibt es auch bei den anderen Windfeldern, die eine ähnliche Form haben. Jedoch sind die Schwerpunkte näher beim Nullpunkt.

Es stellt sich auch hier die Frage einer Skalierung; beide Dimensionen haben die gleiche Einheit, aber ist West-Ost-Wind genauso bedeutend wie Süd-Nord-Wind?

7.5.2 Windrichtung

Die wichtigste Eigenschaft vom Wind ist die Richtung. Die Windstärke ist weniger bedeutend. Deswegen ist es lohnenswert, die Datensätze nur nach dem Winkel einzuteilen. Dabei stellt sich das Problem, daß der Winkel eine periodische Eigenschaft hat. Dies muß man entsprechend berücksichtigen. Für die Klasseneinteilung in Bild 7.31 werden die gleichen Daten verwendet wie im letzten Abschnitt (Winter; konvektiver Niederschlag). Aus den Windvektoren wird der Winkel (zur Nordachse im negativen mathematischen Sinne) und die Länge berechnet. Aus dieser Sicht gibt es wieder eine große geschlossene Punktwolke, wenn man bedenkt, daß der Winkel von 0 Grad gleich 360 Grad entspricht. Für das Training werden nur die Winkel verwendet. Für die grafische Darstellung sind die Neuronen bei einer willkürlichen Windstärke eingetragen. Man erkennt, daß schwach besetzte Bereiche in gebietsmäßig größeren Klassen eingeteilt werden.

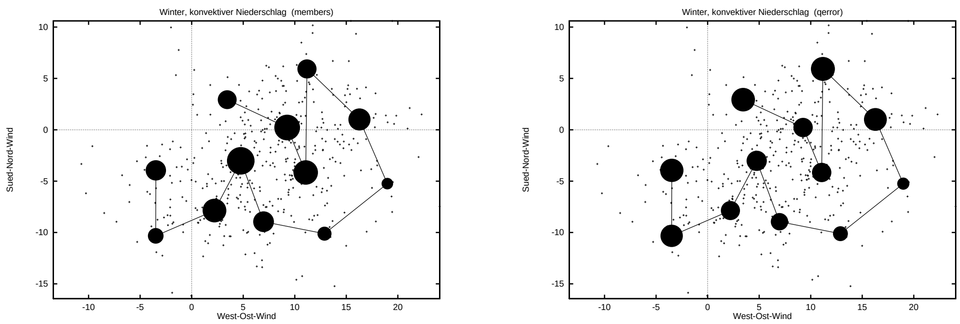


Abbildung 7.30: Eingabevektoren und Neuronen.
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.) Die Parameter für das Training sind die gleichen wie in Tabelle 7.1 (Seite 37), nur die Trainingsdauer der Phase 1 beträgt 6000 Schritte, die Phase 2 2000 Schritte. $N = 426$.

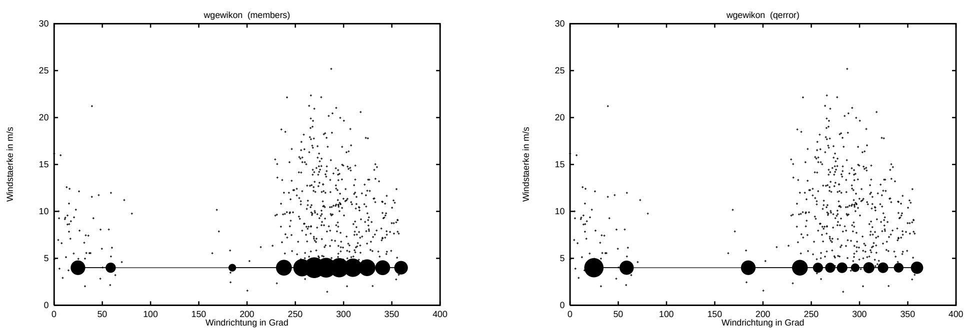


Abbildung 7.31: Eingabevektoren und Neuronen.
(Links ist die Mitgliederanzahl, rechts die Fehlersumme proportional der Kreisfläche.) Die Windstärke wurde für das Training nicht verwendet.

Vergleicht man den Fehler mit einer fixen Einteilung der Winkel in 12 Segmente mit einer Breite von je 30 Grad, so ist der Fehler nur halb so groß.

Eine besondere Eigenschaft ist die Periodizität des Eingaberaumes. Durch sie kann bei einem eindimensionalen Eingaberaum auch eine Ringtopologie sinnvoll eingesetzt werden.[12] Der Neuronen-Ring kann gleichsam wie ein „Gummiring" um einen Zylinder (bzw. Kreis) gespannt werden. Mit den im Simulator implementierten Datenstrukturen ist es leider nicht möglich, „diesen Gummiring von Anfang an über den Zylinder zu spannen",[13] sondern dies geschieht (manchmal) spontan im Laufe des Trainings.

In Abbildung 7.32 ist das Trainingsverhalten einer normalen Neuronenkette zu sehen.[14] Im Vergleich dazu wurde ein Neuronenring mit den gleichen Parametern trainiert. Im dazugehörenden Trainingsverlauf (Bild 7.33) ist eine sprunghafte Verbesserung des Fehlers nach 2600 Lernschritten zu erkennen. Danach bleibt der Fehler klein.

Wollte man schon zu Beginn des Trainings diesen Effekt nutzen (man spart die „Entfaltungsphase" des Netzes ein), muß eine Dimension der Netztopologie fest mit einer Dimension des Eingaberaumes verknüpft werden. Die Lernkurve würde dann zwar größere Schwankungen aufweisen, der Fehler aber im Mittel niedriger liegen. Damit kann die Anzahl der Lernschritte kleiner gewählt werden.

Wegen der besonderen Bedeutung der Windrichtung werden in Abbildung 7.34 die Klasseneinteilungen für alle sechs Dateien als „Windrosenplot" dargestellt. Für jede Zeile dieser Tabelle wurde eine andere Datei verwendet. In den ersten beiden Spalten ist der Winkel in 12 gleichbreite Klassen eingeteilt. Die Länge eines Segments ist in der ersten Spalte proportional zur Anzahl der Klassenmitglieder, in der zweiten Spalte proportional zum Fehler. In Spalte 3 und 4 sind die Daten durch Trainingsläufe mit dem neuronalen Netz in Klassen eingeteilt und analog zu Spalte 1 und 2 grafisch umgesetzt. Man sieht, daß bei der Einteilung in flächenmäßig gleichgroße Klassen der Fehler in einer Klasse proportional zur Anzahl der Klassenmitglieder ist (Spalte 1 und Spalte 2 sehen fast gleich aus). Bei der Clusteranalyse hingegen gibt es eine Tendenz, jeder Klasse gleichviele Mitglieder zuzuordnen. Dies erkennt man in Gebieten, die in den ersten beiden Spalten kleine Fehler und wenig Klassenmitglieder besitzen. In der dritten Spalte ist die Mitgliederanzahl etwas gewachsen und der Fehler in der vierten Spalte überproportional angestiegen.

[12] Ohne periodischen Eingaberaum degeneriert der Ring zu einem Doppelstrang, der eine schlechte Klassenaufteilung liefert.

[13] Das Problem beruht im Prinzip darauf, daß es in einem periodischen Eingaberaum zwei Richtungen gibt, in denen ein Gewicht im Adaptationsschritt (siehe 4.2.1) bewegt werden kann. Der kürzere Weg ist dabei nicht immer der richtige.

[14] Die verwendeten Trainingsdaten sind dem Beispiel in Bild 7.31 sehr ähnlich, nur die Winkel sind gleichmäßiger verteilt. Mit den Daten in Abbildung 7.31 kann kein Sprung von „Doppelstrang" zu „gespannter Ring" erreicht werden.

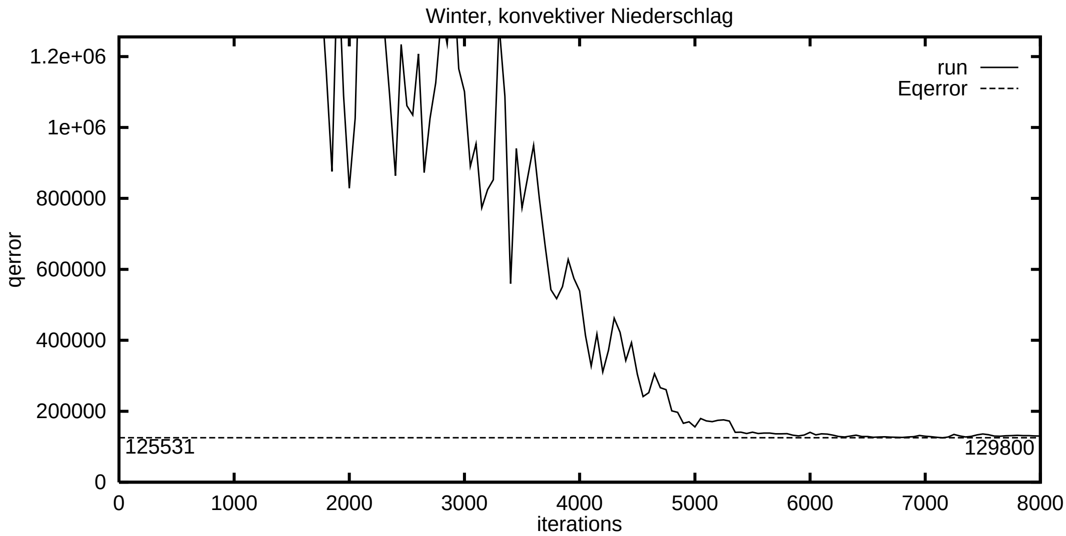


Abbildung 7.32: Trainingsverhalten einer Neuronenkette im periodischen Eingaberaum.
(Phase 1: $T = 6000$ Schritte; Phase 2: $T = 2000$ Schritte; weitere Parameter für das Training siehe Tabelle 7.1.)

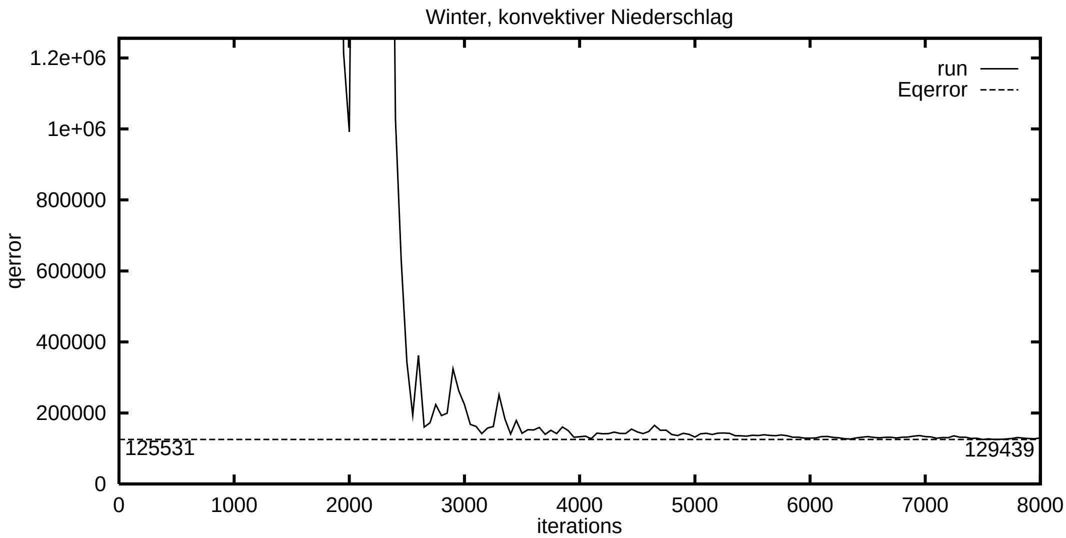


Abbildung 7.33: Trainingsverhalten eines Neuronenrings im periodischen Eingaberaum.
(Phase 1: $T = 6000$ Schritte; Phase 2: $T = 2000$ Schritte; weitere Parameter für das Training siehe Tabelle 7.1.)

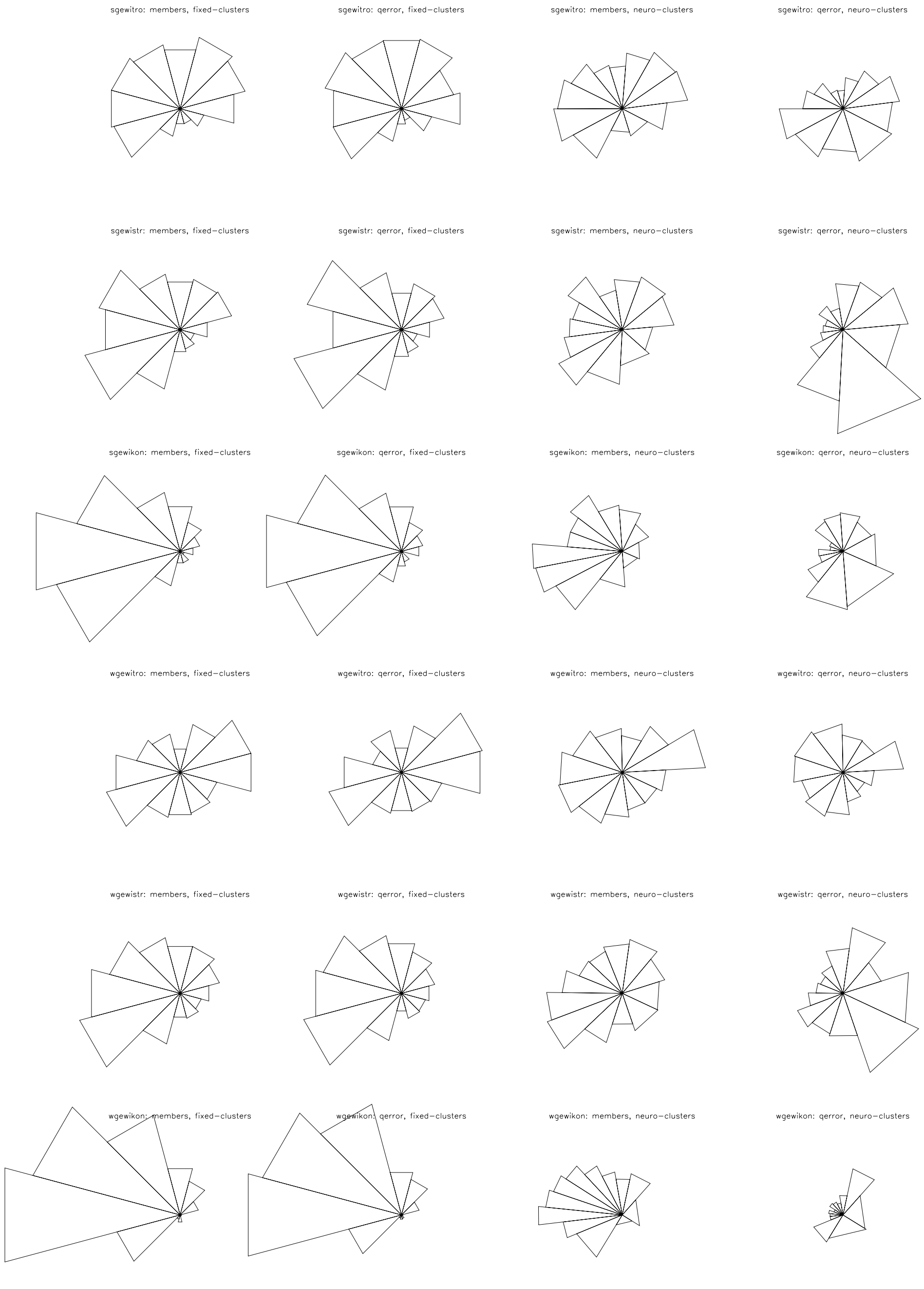


Abbildung 7.34: Verschiedene Windrosen.
Zeile 1-6: Sommer/trocken, Sommer/stratiform, Sommer/konvektiv, Winter/trocken, Winter/stratiform, Winter/konvektiv.
Spalte 1-2: Klassenmitglieder und Fehler bei 30°-Klassen.
Spalte 3-4: Klassenmitglieder und Fehler bei Clusteranalyse-Klassen.

Kapitel 8

Höherdimensionale Daten

Nachdem ich in den letzten Abschnitten ein- und zweidimensionale Daten untersucht habe, gehe ich nun kurz auf die Besonderheiten bei der Verwendung höherdimensionalen Daten ein.

8.1 Allgemeines

Prinzipiell bleibt das Verhalten eines selbstorganisierenden Netzes im Vergleich mit den niederdimensionalen Fällen gleich. Es wird jedoch schwierig, die Daten visuell zu untersuchen oder die Verteilung der Neuronengewichte zu überprüfen. Man kann sich mit mehreren Schnitten oder Projektionen der Daten auf zwei Dimensionen behelfen. Lineare Abhängigkeiten lassen sich anhand einer Eigenwertbestimmung der *Kovarianzmatrix* der Datenvektoren ermitteln. Für die Veranschaulichung der erzeugten Klassen kann man die Anzahl der Mitglieder gegen die Streuungsquadratsumme innerhalb der Klasse grafisch auftragen. Klassen mit extremen Eigenschaften sind so leicht zu identifizieren. Gegebenenfalls ist ein Neutraining angebracht.

Die Berechnung eines (brauchbaren) Referenzfehlers wird bei höheren Dimensionen auch schwieriger, da durch starke Korrelation mancher Dimensionen der Datenvektoren die *echte* Dimension λ in Formel (3.5) kleiner als die Dimension der Datenvektoren sein kann.[1]

Je höher die Dimension des Eingaberaumes ist, desto höher kann man auch die Dimension der Netztopologie wählen.[2] Bei Versuchen mit verschiedenen Netztopologien zeigte sich, daß diese ungefähr gleich große Fehler liefern. Als Datenvektoren wurden aus dem Datenbestand des Anwendungsproblems folgende

[1] Diesen Effekt könnte man zur Bestimmung einer *echten* oder *fraktalen* Dimension nutzen.

[2] Man beachte aber, daß für hochdimensionale Netztopologien sehr viele Neuronen erforderlich sind.

vier Werte verwendet: West-Ost- und Süd-Nord-Wind (horizontaler Mittelwert in 1.5 km Höhe), Luftfeuchte (3 km Höhe) und Temperaturgradient zwischen 5 km und 1.5 km Höhe. Diese Datenvektoren bilden eine einzige große Punktwolke ohne Struktur und ohne größere Korrelationen. Dies kann auch der Grund für die Unabhängigkeit der Güte der Klasseneinteilung von der Netztopologie sein.

Eine Bestimmung einer optimale Skalierung der Daten wird schwieriger, wenn keine konkreten Zusammenhänge der einzelnen Dimensionen der Vektoren bekannt sind. Oft gibt es auch Dimensionen, die für das gewünschte Endergebnis nicht relevant sind. Haben diese Dimensionen im Verhältnis zu den anderen Dimensionen einen großen Wertebereich (durch falsche Skalierung oder Nichtskalierung), kann die resultierende Klasseneinteilung eine schlechte Lösung für das Problem sein. Sind jedoch alle Dimensionen ungefähr in der gleichen Größenordnung, wirken sich wenige irrelevante Dimensionen nicht allzu negativ aus, wenn es vergleichsweise viele relevante Dimensionen gibt.

Eine Korrelation der Daten ist für das Training des Netzes kein Problem. Die Gewichtsvektoren sind nach dem Training auch entsprechend korreliert. Will man die Korrelationen eliminieren oder nur dämpfen, muß entsprechend vorverarbeitet werden. Dies kann man durch Bestimmung der Eigenwerte und Eigenvektoren der Kovarianzmatrix und anschließender Transformation (Hauptkomponententransformation) durchführen. Sind die korrelierten Dimensionen des Datenvektors auch ohne Berechnung der Eigenwerte bekannt, so kann auch eine einfachere Methode verwendet werden.[3] Dazu bildet man den Mittelwert über alle zueinander korrelierten Dimensionen eines Datenvektors und subtrahiert diesen Wert von den Dimensionen.[4] Dieser Mittelwert wird zudem als weitere Dimension in den Vektor eingefügt. Damit erhöht sich die Dimensionalität der Vektoren und es gibt dann eine zusätzliche, exakt lineare Korrelation. Es lassen sich aber so die Einflüsse von *Größe* und *Struktur* der Dimensionen auf die Klasseneinteilung mittels Skalierungsfaktoren sehr einfach gewichten.

8.2 Regionalisierungsproblem

Die im letzten Abschnitt beschriebenen Anmerkungen können auf das Clusteranalyseverfahren innerhalb des Regionalisierungsproblems angewendet werden.

[3] Solche korrelierten Dimensionen sind beispielsweise physikalische Variablen, die im geographischen Raum nur gering um einen Mittelwert schwanken. Dieser Mittelwert schwankt jedoch zeitlich sehr stark. Temperaturfelder oder Windvektorfelder sind ein solches Beispiel. Die Werte bei jeden Gitterpunkt sind pro Tag relativ gleich, wenn man sich auf ein kleinräumiges Gebiet beschränkt. Die Tagesmittelwerte können jedoch nach Jahreszeit oder globaler Windlage sehr stark variieren.

[4] Es gibt *je* Datenvektor einen anderen Mittelwert. Würde man den Mittelwert über alle Datenvektoren verwenden, bringt dies keine Änderung der Klasseneinteilung, da das Lernverfahren *translationsinvariant* ist.

Die Daten sind im Prinzip hochdimensional und stark korreliert. Da die Daten aber geographisch auf einen relativ kleinen Raum beschränkt sind, und wegen der Arbeitsweise des regionalen Klimamodells, wurden bisher bei der festen Einteilung in die 150 Klassen (siehe Abschnitt 5.2.1) nur Mittelwerte verwendet. Geographische Unterschiede der physikalischen Variablen in Breite, Länge oder Höhe haben keinen Einfluß. Damit gibt es keine starken Korrelationen, da die Mittelwerte relativ unabhängig sind. Erst für zukünftige Projekte ist geplant, bei einem größeren Zielgebiet auch die regionale Verteilung der Werte zu berücksichtigen.

Zum Vergleich der festen Klasseneinteilung mit dem Einteilung durch ein selbstorganisierendes neuronales Netz muß man die gleiche Datenbasis verwenden. Dies bedeutet, man führt ein Trainingslauf eines Netzes mit K=150 Neuronen und für alle $N = 3652$ 4-dimensionalen Datenvektoren mit folgenden Komponenten durch: horizontaler Windvektor, Niederschlag und Temperatur. Zum Vergleich können dann die Klassenzentren der festen Klasseneinteilung den Klassen aus dem Trainingslauf zugeordnet werden (entweder über den Abstand der Zentren oder die Anzahl der gemeinsamen Mitglieder).

Eine andere Beurteilungsmöglichkeit ergibt sich durch Betrachtung der zeitlichen Verteilung der Klassen. Da die Datenvektoren zeitlich geordnet sind und das Wetter oft über ein paar Tage gleich bleibt, sollten aufeinanderfolgende Tage oft in die gleiche Klasse fallen.

Die wirkliche Qualität der gefundenen Klassen läßt sich aber nur nach den 150 Simulationsläufen des regionalen Klimamodells endgültig beantworten. Wegen des hohen Rechenaufwands[5] wäre es vielleicht günstig, schon vorhandene Simulationsergebnisse wiederzuverwerten. So kann man manche Gewichte mit den Zentren der festen Klasseneinteilung initialisieren und diese während des Trainings konstant lassen. Nur die anderen Gewichte werden verändert.

[5]Die Rechenzeit für 150 Simulationsläufe beträgt auf einer Cray Y-MP ungefähr 190 CPU-Stunden.

Kapitel 9

Zusammenfassung und Diskussion

Die in dieser Arbeit vorgestellten selbstorganisierenden neuronalen Netze (Kohonen-Netz, neuronales Gas) sind für die Erzeugung von Klassen durch Minimierung eines bestimmten Fehlerkriteriums sehr gut einsetzbar. Speziell für das gegebene Anwendungsproblem, in dem eine einzige, mehr oder weniger dicht gefüllte Punktwolke in Klassen aufgeteilt werden soll, ist die Minimierung der Streuungsquadratsumme ein guter Ansatz. Da die generierten Klassen statistisch weiterverarbeitet werden, führt bei einer geeigneten Skalierung die Minimierung der Streuungsquadratsumme zu einer Minimierung des Fehlers in den Ausgabevariablen.

Im Vergleich mit den (klassischen) statistischen Verfahren basiert ein Kohonen-Netz auf einer anderen Denkweise. Man versucht, in einem *Lernprozeß* von einer Anpassung an die *globale* Struktur der Daten schrittweise zu einer Anpassung an die *lokalen* Strukturen zu gelangen. Durch diesen „Trick“ werden die Klassen über das gesamte Datengebiet verteilt, so daß zwischen Klassengröße und Fehler je Klasse ein guter Kompromiß gefunden wird. Die Probleme bei vielen klassischen Verfahren liegen in der Abhängigkeit von den Startwerten oder in der Unbeweglichkeit, eine einmal gefundene Teillösung zu verwerfen und einen neuen Weg einzuschlagen. Zur Lösung dieser Probleme könnte man auch *genetische Algorithmen* einsetzen.

Die Minimierung der Streuungsquadratsumme ist aber nicht für jedes Problem ein guter Ansatz. Oft werden klassische Verfahren geeignetere Klassen finden, beispielsweise bei der Suche nach *natürlichen* Clustern, die verschieden groß, aber gut getrennt sind.

Wie groß der Einfluß einer verbesserten Klasseneinteilung auf die Ausgabedaten des *Regionalisierungschemas* ist, läßt sich zum jetzigen Zeitpunkt nicht konkret angeben. Jedoch sind die Werkzeuge vorhanden und die nötigen Informationen in dieser Diplomarbeit zusammengestellt, um für diesen konkreten Fall oder für andere Daten eine Clusteranalyse mit selbstorganisierenden neuronalen Netzen durchzuführen.

Ein direkter Vergleich der Klasseneinteilung durch Kohonen-Netze oder neuronale Gase und einem klassischen Verfahren in Bezug auf Rechenzeit und Minimierung des Fehlers wäre sicherlich interessant.

Literaturverzeichnis

[Bacher 1994] **Bacher, Johann:** *Clusteranalyse: Anwendungsorientierte Einführung.* München: Oldenbourg, 1994.

[Brause 1991] **Brause, Rüdiger:** *Neuronale Netze.* Stuttgart: Teubner, 1991.

[Frey-Buness et al. 1995] **Frey-Buness, A.; Heimann, D.; Sausen, R.:** „A statistical-dynamical downscaling procedure for global climate simulations“. *Theor. Appl. Climatol. 50*, 117-131. 1995.

[Fuentes 1994] **Fuentes, Ursula:** *Clusteranalyse.* Interner DLR-Abteilungsvortrag, Mai 1994.

[Fuentes et al. 1995] **Fuentes, U.; Sept, V.; Heimann, D.; Sausen, R.:** „A statistical-dynamical downscaling procedure for global climate simulations“. *6th International Meeting on Statistical Climatology*, 37-40. 1995.

[Fritzke 1992] **Fritzke, Bernd:** „Growing cell structures – a self-organizing network in k dimensions.“ In Aleksander, I., and Taylor, J., (eds): *Artificial Neural Networks II*, 1051-1056. North Holland, 1992.

[Fritzke 1995] **Fritzke, Bernd:** „A growing neural gas network learns topologies.“ In Tesauro, G., Touretzky, D. S., and Leen, T. K., (eds): *Advances in Neural Information Processing Systems 7.* Cambridge MA: MIT, 1995.

[Grossberg 1976] **Grossberg, Stephen:** „Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors“. *Biological Cybernetics 23*: 121-134. 1976.

[Haberäcker 1991] **Haberäcker, Peter:** *Digitale Bildverarbeitung*, 4., durchges. Auflage. München: Hanser, 1981.

[Köhle 1990] **Köhle, Monika:** *Neurale Netze.* Wien: Springer, 1990.

[Köhler 1992] **Köhler, Klaus:** *Vorlesungsskript: Neuronale Netze.* Fachhochschule München, Fachbereich Informatik, WS 1991/92.

[Kohonen 1984] **Kohonen, Teuvo:** *Self-organization and associative memory.* Springer Series in Information Sciences 8. Heidelberg: Springer, 1984.

[Martinetz et al. 1993] **Martinetz, Thomas; Berkovich, Stanislav; Schulten, Klaus:** „"Neural-Gas" Network for vector quantization and its application to time-series prediction". *IEEE Transactions on Neural Networks*, 4(4):558-569. July 1993.

[McClelland/Pitts 1943] **McClelland, W.S.; Pitts, W.:** „A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics*, 5:115-133. 1943.

[Ritter et al. 1989] **Ritter, Helge; Martinetz, Thomas; Schulten, Klaus:** „Ein Gehirn für Roboter". *mc 2/89*, 8-61. Februar 1989.

[Ritter et al. 1991] **Ritter, Helge; Martinetz, Thomas; Schulten, Klaus:** *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke,* 2. erweiterte Auflage. Bonn: Addison-Wesley, 1991.

[Sammon 1969] **Sammon, Jr.; John W.:** „A Nonlinear mapping for Data Structure Analysis". *IEEE Transactions on Computers,* C-18(5):401-409, May 1969.

[Späth 1983] **Späth, Helmut:** *Cluster-Formation und -Analyse.* München: Oldenbourg, 1983.

[Zador 1982] **Zador, Paul L.:** „Asymptotic quantization error of continuous signals and the quantization dimension." *IEEE Transactions on Information Theory,* IT-28(2):129-149, 1982.

[Zell 1994] **Zell, Andreas:** *Simulation Neuronaler Netze.* Bonn: Addison-Wesley, 1994.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben.

Diesen Dank spreche ich der Deutschen Forschungsanstalt für Luft- und Raumfahrt aus, im besonderen Maße Herrn Priv.-Doz. Dr. R. Sausen für die Vergabe des Themas und die Betreuung der Arbeit.

Für die konstruktiven Anregungen bedanke ich mich bei Herrn Prof. Dr. K. Köhler.

Besonders herzlich möchte ich mich bei Frau Dipl. Phys. Ursula Fuentes für die Zusammenarbeit und die Durchsicht des Manuskripts bedanken. Auch den anderen Mitarbeitern des Instituts für Physik der Atmosphäre möchte ich meinen Dank für die offene Arbeitsatmosphäre aussprechen.

Desweiteren danke ich meinem Studienkommilitonen Marco Ballack für manchen wertvollen Hinweis.

Ein spezieller Dank geht an all die vielen Menschen (besonders Linus Torvalds in Helsinki), die Software entwickeln und diese dann kostenlos zur Verfügung stellen.

Anhang A

Herleitung des Referenzfehlers

Gegeben sind N Vektoren $\mathbf{x}_i \in \Re^\lambda$.

Gesucht ist eine Näherung für die Streuungsquadratsumme einer optimalen Einteilung der Vektoren $\mathbf{x}_i$ in K Klassen.

Annahme:
Die K Klassenzentren $\mathbf{w}_j \in \Re^\lambda$ können so gewählt werden, daß die Abstandsvektoren $\mathbf{v}_i$ zum nächstliegenden Klassenzentrum $\mathbf{w}_k$, gegeben durch

$$\mathbf{v}_i = \mathbf{x}_i - \mathbf{w}_k \quad \text{mit} \quad \|\mathbf{x}_i - \mathbf{w}_k\| = \min_{j=1}^{K}(\|\mathbf{x}_i - \mathbf{w}_j\|),$$

eine statistische Gleichverteilung in einem Hyperwürfel mit der Dimension λ und der Kantenlänge a bilden.

Satz:
Für den Fall, daß diese Annahme gilt, läßt sich die Streuungsquadratsumme dieser Klasseneinteilung für große N durch folgende Formel approximieren:

$$Q = Q^{(\mathrm{eq})} = \frac{1}{12}\lambda N a^2.$$

Diese Formel ist nur eine Approximation, da die Abstandsvektoren $\mathbf{v}_i$ nur eine endliche Stichprobe einer Gleichverteilung bilden.

Beweis:
Die Streuungsquadratsumme Q (Definition siehe Formel (3.2)) läßt sich in folgender Weise formulieren:

$$Q = \sum_{i=1}^{N} \|\mathbf{v}_i\|^2.$$

Durch Annahme 1 und Übergang zu einer kontinuierlichen Verteilung ergibt sich ein Mehrfachintegral über den Raum $\mathbf{V}$ des Hyperwürfels.[1]

$$\begin{aligned}
Q^{(\mathrm{eq})} &= \frac{N}{a^\lambda} \int\limits_{\mathbf{V}} \|\mathbf{z}\|^2 d\mathbf{V} \\
&= \frac{N}{a^\lambda} \int\limits_{z_1=-\frac{a}{2}}^{z_1=\frac{a}{2}} \int\limits_{z_2=-\frac{a}{2}}^{z_2=\frac{a}{2}} \cdots \int\limits_{z_\lambda=-\frac{a}{2}}^{z_\lambda=\frac{a}{2}} z_1^2 + z_2^2 + \cdots + z_\lambda^2 \; dz_\lambda \cdots dz_2\, dz_1
\end{aligned}$$

Dieses Mehrfachintegral läßt sich einfach lösen, da nur unabhängige Summenterme zu integrieren sind.

$$\begin{aligned}
Q^{(\mathrm{eq})} &= \frac{N}{a^\lambda}\left(\frac{2}{3}\left(\frac{a}{2}\right)^3\left(\frac{2a}{2}\right)^{\lambda-1} + \frac{2}{3}\left(\frac{a}{2}\right)^3\left(\frac{2a}{2}\right)^{\lambda-1} + \cdots + \frac{2}{3}\left(\frac{a}{2}\right)^3\left(\frac{2a}{2}\right)^{\lambda-1}\right) \\
&= \frac{N}{a^\lambda}\lambda\frac{1}{12}a^{\lambda+2} \\
&= \frac{1}{12}\lambda N a^2
\end{aligned}$$

Anmerkung:
Es kann mehrere Möglichkeiten geben, die Annahme zu erfüllen. Findet man unter allen denkbaren Möglichkeiten diejenige Klasseneinteilung, so daß die Kantenlänge a am kleinsten ist, bildet $Q^{(\mathrm{eq})}$ im Normalfall eine gute obere Schranke für den Fehler einer optimalen Klasseneinteilung.

Oft ist die Erfüllung der Annahme überhaupt nicht möglich. Man kann aber trotzdem versuchen, eine „äquivalente“ Kantenlänge a abzuschätzen. Die Ermittelung von einer guten Kantenlänge a ist aber meistens nicht einfach.

$Q^{(\mathrm{eq})}$ ist nur eine obere Schranke, da die Annahme nicht für den Fall der optimalen Klasseneinteilung gelten muß.[2]

[1] Im zweidimensionalen Fall entspricht die Streuungsquadratsumme dem physikalischen Trägheitsmoment des Hyperwürfels, der dann eine quadratische Fläche ist.

[2] Beispiel: Die optimale Klasseneinteilung einer zweidimensionalen Gleichverteilung ist nicht die Aufteilung in Klassen mit quadratischer Form, sondern wahrscheinlich in sechseckige Klassen, würde man die Effekte am Rand vernachlässigen. Der Fehler kann sich aber in diesem Fall maximal um $1 - \frac{3}{\pi} \approx 4.5\%$ verbessern. (Vergleich des Trägheitsmoments einer dünnen Kreisscheibe mit einer dünnen, quadratischen Platte bei gleichem Volumen. Das Trägheitsmoment einer sechseckigen Platte liegt dazwischen.)

Anhang B

Wegweiser ins Internet

Für die Suche nach Programmen, Texten oder Diskussionsforen im Internet möchte ich einige Tips geben. Die Verweise sind im URL-Format[1] angegeben. Diese können von WWW-Browsern[2] wie *mosaic*, *netscape* oder *lynx* verwendet werden. Man sollte jedoch beachten, daß diese Verweise einer starken Dynamik unterliegen und meist nicht über Jahre hinweg gültig sein werden.

`ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS`
SNNS, ein Simulator für verschiedene neuronale Netzmodelle mit graphischer Benutzeroberfläche (X11).

`ftp://cochlea.hut.fi/pub/som_pak`
SOM-PAK, ein Simulator für das Kohonen-Netz.

`ftp://me.uta.edu/pub/neural/annsim/kohsim24.lzh`
ROXANNE, ein einfacher Kohonen-Netz-Simulator mit graphischer Oberfläche (X11). Nur für Demonstrationszwecke geeignet.

`ftp://ftp.funet.fi/pub/sci/neural/`
`ftp://ftp.funet.fi/pub/sci/neural/neuroprose/`
Hier sind viele Programme und Texte von anderen FTP-Servern zusammengetragen. Im Verzeichnis `neuroprose` gibt es hunderte von veröffentlichten Artikeln zum Thema „neuronale Netze".

`ftp://crl.ucsd.edu/pub/neuralnets/cluster-2.5.tar.Z`
Ein Programm für die hierarchische Clusteranalyse.

`ftp://ftp.funet.fi/pub/sci/graphics/packages/xgobi/xgobi.tar.Z`
Ein Programm zur Visualisierung mehrdimensionaler Datenvektoren.

[1] Uniform Resource Locator
[2] World Wide Web

`ftp://csr.uta.edu:/pub/autoclass-c.tar.Z`
Ein Programm zur Clusteranalyse unter Verwendung des Bayes'schen Klassifikationskriteriums.

`news://comp.ai.neural-nets`
Die Newsgruppe (Diskussionsforum) für neuronale Netze.

`http://www.pitt.edu/~csna/`
Die Homepage der „Classification Society of North America“. Hier findet man Informationen und weitere Verweise auf Programme zum Themengebiet „Clusteranalyse“.