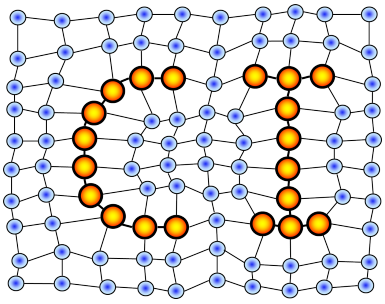
HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

Frau
Lydia Fischer

Modifikation unüberwachter Vektorquantisierer für funktionale Daten und Einbindung einer neuen Optimierungsstrategie



2012

Fakultät **Mathematik/Naturwissenschaften/Informatik**

MASTERARBEIT

Modifikation unüberwachter Vektorquantisierer für funktionale Daten und Einbindung einer neuen Optimierungsstrategie

Autorin:
Lydia Fischer

Studiengang:
Diskrete und Computerorientierte Mathematik

Seminargruppe:
ZD10

Erstprüfer:
Prof. Dr. rer. nat. habil. Thomas Villmann

Zweitprüfer:
Marika Kästner, M.Sc.

Mittweida, 2012

Bibliografische Angaben

Fischer, Lydia: Modifikation unüberwachter Vektorquantisierer für funktionale Daten und Einbindung einer neuen Optimierungsstrategie, 59 Seiten, 29 Abbildungen, Hochschule Mittweida, Fakultät Mathematik/Naturwissenschaften/Informatik

Masterarbeit, 2012

Referat

In dieser Arbeit wird der Neural Gas mit funktionalen Prototypen vorgestellt, der sich insbesondere zur Analyse von funktionalen Daten eignet. Hierbei werden die Prototypen als diskrete Repräsentanten einer Funktion interpretiert bzw. als Linearkombination von Basisfunktionen dargestellt. Außerdem wird an Stelle der euklidischen Abstandsbestimmung eine Sobolev Quasi-Metrik verwendet. Im zweiten Teil der Arbeit werden der Pulsing Neural Gas, der Pulsing Neural Gas Batch und der Pulsing Fuzzy Neural Gas dargelegt. Die ursprünglichen Algorithmen sind in diesen Versionen mit Simulated Annealing kombiniert, um das Konvergenzverhalten der Algorithmen zu verbessern.

Danksagung

Die vorliegende Masterarbeit entstand im Rahmen meines Studiums der Diskreten und Computerorientierten Mathematik an der Hochschule Mittweida. Bereitgestellt wurde die bearbeitete Aufgabenstellung von der Computational Intelligence Group unter der Leitung von Herrn Professor Villmann.

Bedanken möchte ich mich bei Marika Kästner und Tina Geweniger, die mich stets mit Rat und Tat unterstützten. Außerdem möchte ich ihnen für das Korrekturlesen meiner Masterarbeit danken.
Bei Herrn Professor Villmann möchte ich mich für die Möglichkeit zur Durchführung der Masterarbeit in seiner Computational Intelligence Forschungsgruppe bedanken. Die Zusammenarbeit mit ihm und der Forschungsgruppe war für mich sehr bereichernd. Insbesondere waren die Diskussionen über bestehende Aufgaben und neue Ideen in der Forschungsgruppe stets sehr motivierend und abwechslungsreich. Auch für die Ratschläge und Anregungen für die Anfertigung meiner Masterarbeit möchte ich Herrn Professor Villmann danken.
Danken möchte ich allen CI'lern für die herzliche Aufnahme in die Forschungsgruppe.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungs- und Symbolverzeichnis

Im Folgenden sind die wichtigsten Formelzeichen und Symbole der Arbeit aufgeführt. Vektoren (Kleinbuchstaben) und Mengen (Großbuchstaben) sind durch Fettdruck, skalare Größen durch Normaldruck gekennzeichnet.

Abkürzungsverzeichnis

Abs. ..... Abschnitt

Alg. ..... Algorithmus

FNG ..... Fuzzy Neural Gas

HPNG ..... Pulsing Neural Gas mit Heuristik

LK ..... Linearkombination

NG ..... Neural Gas

PFNG ..... Pulsing Fuzzy Neural Gas

PNG ..... Pulsing Neural Gas

SA ..... Simulated Annealing

SV ..... SV-Index

SVF ..... SVF-Index

XB ..... XieBeni-Index

Mathematische Symbole

$\subseteq$ ..... Untermenge

$\in$ ..... Element von ...

$\ll$ ..... sehr viel kleiner als

$\searrow$ ..... geht gegen

$\rightarrow$ ..... geht gegen

$\forall$ ..... Allquantor

∞ ..... Symbol für Unendlichkeit

π ..... Kreiszahl

$\mathbf{V}$ ..... Menge der Daten

$\mathbf{V}_k$ ..... k-te Voronoi Zelle

$\mathbf{V}_m$ ..... Untermenge von $\mathbf{V}$ der Mächtigkeit m

$\mathbf{v}$, $\mathbf{v}_k$ ..... Datenpunkt $\mathbf{v}$ bzw. der k-te Datenpunkt $\mathbf{v}$

$\dot{\mathbf{v}}$ ..... erste Ableitung des Datenpunktes $\mathbf{v}$ nach t

$v(t)$ ..... t-te Komponente des Vektors $\mathbf{v}$

$\mathbb{N}$ ..... die natürlichen Zahlen

$\mathbb{R}$	die reellen Zahlen
$\mathbb{R}^D$	D-dimensionaler Raum der reellen Zahlen
$\mathbf{W}$	Menge der Prototypen
$\mathbf{w}$, $\mathbf{w}_j$	Prototyp bzw. der j-te Prototyp
$w(t)$, $w(t)_j$	Prototyp bzw. der j-te Prototyp an der Stelle t bzw. die t-te Vektorkomponente
$\dot{\mathbf{w}}$	erste Ableitung des Prototypen $\mathbf{w}$ nach t
$\mathbf{w}_{s(\mathbf{v})}$	Sieger-Prototyp bzgl. Datenpunkt $\mathbf{v}$
$\mathbf{w}_{l(\mathbf{v})}$	Verlierer-Prototyp bzgl. Datenpunkt $\mathbf{v}$
$\Delta\mathbf{w}_j$	Änderung eines Prototyps beim Adaptionsschritt.
$\langle\Delta\mathbf{w}\rangle$	mittlere Änderung eines Prototypen
$\mathscr{L}^p$	Lebesgue-Raum, der p-fach integrierbaren Funktionen
D	Dimension der Daten
M	Mächtigkeit der Datenmenge $\mathbf{V}$
N	Mächtigkeit der Menge der Prototypen $\mathbf{W}$
K	Gesamtanzahl der Basisfunktionen für einen Prototypen
α	Wichtung der ersten Ableitung in der Sobolev Quasi-Metrik
γ	konvexer Wichtungsfaktor beim negativen Lernschritt beim Pulsing Neural Gas Batch
λ	Abklingkonstante in der Nachbarschaftsfunktion h_λ
ε	Lernrate
ε_β	Lernrate für die Gewichte der Basisfunktionen
ε_θ	Lernrate für die Stützstellen der Gaußfunktionen
$C(\lambda)$	Normalisierungsfaktor
c_λ	Normalisierungsfaktor
m	*fuzziness* Parameter
κ_{jk}	Nachbarschaftsrang des j-ten Prototyps bzgl. des Datenpunkts $\mathbf{v}_k$
β_{i_j}	Gewicht der i-ten Basisfunktion des j-ten Prototypen
σ_{i_j}	Breite der i-ten Gaußfunktion des j-ten Prototypen
θ_{i_j}	Stützstelle der i-ten Gaußfunktion des j-ten Prototypen
u_{kj}	*fuzzy assignment*: Stärke der Zugehörigkeit des Datenpunktes $\mathbf{v}_k$ zum j-ten Prototyp
p_0	maximaler Wert der Wahrscheinlichkeitsverteilung $p(T)$
σ_p	Halbwertszeit der Wahrscheinlichkeitsverteilung $p(T)$
z	gleichverteilte Zufallszahl
j, k, l, t	Laufindizes

$\text{argmin}(\cdot)$	gibt den Index zurück, für den das Argument minimal ist
$\text{argmax}(\cdot)$	gibt den Index zurück, für den das Argument maximal ist
$\|\cdot\|_E^2$	quadratische, euklidische Norm eine Vektors
$d(\cdot,\cdot)$	Ähnlichkeitsmaß für zwei Vektoren
d_E^2	quadratische, euklidische Quasi-Metrik
$d_S^p(\cdot,\cdot)$	Sobolev-Norm in die p-te Potenz erhoben
$d_S^2(\cdot,\cdot)$	quadratische Sobolev Quasi-Metrik
$rg_j(\mathbf{v},\mathbf{W})$	Nachbarschaftsrang des j-ten Prototyps bzgl. des Datenpunkts $\mathbf{v}$
$h_\lambda(rg_j(\mathbf{v},\mathbf{W}))$, $h_\lambda(\mathbf{v},\mathbf{w}_j)$	Funktionswert der Nachbarschaftsfunktion des j-ten Prototyps bzgl. des Datenpunkts $\mathbf{v}$
$lc_\lambda(\mathbf{v}_k,\mathbf{w}_j)$	lokale Kosten bzgl. des j-ten Prototyps und des Datenpunkts $\mathbf{v}_k$
$P(\mathbf{v})$	Dichteverteilung der Daten
$p(T)$	Wahrscheinlichkeitsverteilung für einen negativen Lernschritt
$\mathcal{K}_{i_j}(t)$	i-te Basisfunktion des j-ten Prototypen, an der Stelle t
E	Energiefunktion
E_{NG}	Energiefunktion des Neural Gas
E_{NG_S}	Energiefunktion des Neural Gas mit Sobolev Quasi-Metrik
E_{FNG}	Energiefunktion des Fuzzy Neural Gas
$\frac{\partial E_{NG}}{\partial \mathbf{w}_j}$	partielle Ableitung der Energiefunktion des Neural Gas nach dem j-ten Prototyp
$\frac{\partial E_{NG}}{\partial \beta_{i_j}}$	partielle Ableitung der Energiefunktion des Neural Gas nach dem Gewicht β_{i_j}
$\frac{\partial E_{NG}}{\partial \theta i_j}$	partielle Ableitung der Energiefunktion des Neural Gas nach der Stützstelle θ_{i_j}
$H(x)$	Heaviside-Funktion
$\binom{M}{m}$	Binomialkoeffizient

1 Ziele und Motivation

Tagtäglich werden Unmengen an Daten aller Art gesammelt und gespeichert. Besonders im Internet durch Google, Facebook, Amazon, ... werden viele Daten der Nutzer für eine Weiterverarbeitung aufbewahrt. Amazon nutzt diese Informationen um Besuchern der Website Kaufvorschläge zu unterbreiten, die speziell auf jeden Nutzer abgestimmt sind. Die Analyse des Kaufverhaltens von Kunden ist nur ein Gebiet, in welchem große Mengen an Daten untersucht werden. Die zunehmende Automatisierung von Transaktionen in der Geschäftswelt sowie das Speichern von Telefongesprächen beispielsweise führen ebenfalls zu großen Datenbeständen. Viele Unternehmen, wie zum Beispiel DYMATRIX, verdienen ihr Geld mit der Analyse und Auswertung von Daten. Auch die Werbeindustrie ist stets bemüht, ihre Werbekampagnen so kundenansprechend wie möglich zu gestalten bzw. ist sie an Strategien interessiert, die den Erfolg der Kampagnen steigern. Diese Verbesserungsvorschläge können aus der Analyse von Kundendaten resultieren. Bei der Qualitätssicherung von Lebensmitteln gibt es ebenfalls Methoden, die auf der Auswertung von Daten beruhen. Beispielsweise kann man die enthaltenen Bestandteile der zu untersuchenden Lebensmittel auf Schadstoffe oder auf ihren prozentualen Anteil überprüfen.

Manuell kann die zur Verfügung stehende Datenmenge nicht mehr ausgewertet werden. Die Auswertung der Daten ist demzufolge automatisiert mit Algorithmen durchzuführen. Möglich ist eine statistische oder numerische Analyse der Daten. Fokussiert man sich auf die numerische Analyse, so stehen u. a. Verfahren aus dem Bereich Data Mining oder aus dem Bereich der Cluster-Algorithmen zur Verfügung. In dieser Arbeit werden Methoden aus dem Bereich der Cluster-Algorithmen vorgestellt und weiterentwickelt, im Speziellen Methoden der unüberwachten Vektorquantisierung. Zahlreiche Algorithmen, wie zum Beispiel der c-means, der Neural Gas und der Fuzzy Neural Gas, stehen dem Anwender aus dem Gebiet der unüberwachten Vektorquantisierung zum Extrahieren von Informationen aus Daten bereits zur Verfügung. Für viele Problemstellungen arbeiten die Algorithmen zufriedenstellend, doch es gibt Aufgabenstellungen, in denen die bekannten Methoden nicht geeignet sind (vgl. No Free Lunch Theorem [WM95]). In der Forschung ist man demzufolge stets bemüht die bestehenden Verfahren und Methoden weiter zu entwickeln, um für verschiedenste Aufgabenstellungen Lösungen anzubieten.

Zwei unterschiedliche Ansätze, wie Wissen über die Daten in den bestehenden Algorithmen berücksichtigt wird, werden in dieser Arbeit vorgestellt.
Für Daten, die einen funktionalen Zusammenhang beschreiben, wird der genannte Neural Gas den Daten entsprechend angepasst bzw. stellt dieser neue Algorithmus eine Spezialisierung des bestehenden Algorithmus dar. Konkret werden die Daten und die zu extrahierenden Merkmalsvektoren als diskrete Repräsentanten der zugrundeliegenden Funktion aufgefasst. Außerdem wird zur Abstandsbestimmung eine funktionale Metrik

verwendet, statt der i. A. verwendeten euklidischen Metrik. Ein Anwendungsgebiet für diesen angepassten Neural Gas besteht in der Analyse von Spektraldaten.
Der zweite Ansatz ist die Verbindung der Idee von Methoden der Vektorquantisierung mit der Grundidee von Simulated Annealing. In der Vektorquantisierung wird häufig eine Zielfunktion mit Hilfe eines stochastischen Gradientenabstiegs minimiert. Dabei kommt es vor, dass das Verfahren in lokalen Minima „stecken bleibt“. Durch die Kombination mit Simulated Annealing besteht die Möglichkeit lokale Minima wieder zu verlassen. Benannt wird diese neue Methodik als Pulsierende Vektorquantisierung.

Im Kapitel 2 werden zunächst die Grundlagen der Vektorquantisierung erläutert. Außerdem werden der Neural Gas und der Fuzzy Neural Gas im Speziellen ausgeführt, da diese die Grundlage der vorliegenden Arbeit darstellen. Anschließend wird im Kapitel 3 die Modifikation des Neural Gas für funktionale Daten vorgestellt. Im Detail wird die Darstellung der Merkmalsvektoren, auch Prototypen genannt, als Linearkombination von Basisfunktionen erläutert. Die Verwendung einer funktionalen Metrik wird im Anschluss daran vorgestellt. Nachfolgend sind die Ausführungen zur Pulsierenden Vektorquantisierung in Kapitel 4 dargelegt. Im Besonderen wird auf Simulated Annealing eingegangen und daraus werden in Kombination mit dem Neural Gas und dem Fuzzy Neural Gas neue Verfahren entwickelt, um eine schnellere Konvergenz zu erreichen. Experimente zu den Verfahren sowie deren Validierung sind im Kapitel 5 dokumentiert. Hier werden verschiedene Validierungsmaße für Clusterlösungen, sowie deren Interpretation vorgestellt. Die zum Testen verwendeten Datensätze sind ebenfalls in Kapitel 5 zu finden. Kernstück dieses Kapitels sind die Testergebnisse für die verschiedenen Algorithmen und Datensätze. Im Anschluss daran werden eine Zusammenfassung und ein Ausblick gegeben.

2 Unüberwachte Vektorquantisierung

2.1 Prinzip der unüberwachten Vektorquantisierung

Die unüberwachte Vektorquantisierung ist eine Methode zur Kompression von Datensätzen bzw. Datenmengen. Merkmalsvektoren, im Folgenden Prototypen genannt, fassen die in der Datenmenge enthaltenen Informationen zusammen. Gelingt die Vektorquantisierung, genügt es die Prototypen eines Datesatzes zu kennen, um diesen zu charakterisieren. Die hier zur Anwendung kommende Art der Vektorquantisierung wird in zwei Schritten durchgeführt. Zuerst werden die Prototypen gemäß eines stochastischen Gradientenabstiegs zur Minimierung einer Energiefunktion bzw. Kostenfunktion für die gegebene Datenmenge ermittelt. Die Energiefunktion berechnet einen Beschreibungsfehler, der bei der Bestimmung der Prototypen anhand der Datenmenge entsteht. Nach dieser Trainingsphase können neue Daten den gelernten Prototypen zugeordnet werden.

Beispielsweise wird die Vektorquantisierung in der Spracherkennung verwendet. Anhand einer Datenmenge mit Sprachmustern werden Prototypen trainiert, die diesen Datensatz repräsentieren. Jetzt kann ein neues Sprachmuster mit den Prototypen abgeglichen und einem Prototypen zugeordnet werden.

Die folgenden Ausführungen zur Vektorquantisierung sind der Dissertation von Thomas Martinetz [Mar92] entnommen. Sei $\mathbf{V} \subseteq \mathbb{R}^D$ eine Datenmenge mit D-dimensionalen Datenpunkten $\mathbf{v} \in \mathbf{V}, |\mathbf{V}| = M$. Des Weiteren sei $\mathbf{W} = \left\{\mathbf{w}_j \in \mathbb{R}^D, j = 1, \ldots, N\right\}$ die Menge der Prototypen, welche die Datenmenge $\mathbf{V}$ mittels Vektorquantisierung repräsentiert. Im Allgemeinen ist $N \ll M$. Ein Datenpunkt $\mathbf{v} \in \mathbf{V}$ wird durch den Sieger-Prototyp $\mathbf{w}_{s(\mathbf{v})}$ dargestellt, für den der Beschreibungsfehler $d(\mathbf{v}, \mathbf{w}_{s(\mathbf{v})})$ am geringsten ist, d. h.

$$s(\mathbf{v}) = \underset{j=1,\ldots,N}{\operatorname{argmin}}\, d(\mathbf{v}, \mathbf{w}_j). \tag{2.1}$$

Oft wird der quadratische, euklidische Fehler $\|\mathbf{v} - \mathbf{w}_{s(v)}\|_E^2$ für d verwendet. Die minimale Anforderung an d ist, dass d ein Ähnlichkeitsmaß [PD05] widerspiegelt. Dadurch ergibt sich für jeden Prototypen $\mathbf{w}_j$ ein Einflussbereich auf der Datenmenge $\mathbf{V}$, die Voronoi Zelle (Abb. 2.1):

$$\mathbf{V}_k = \{\mathbf{v} \in \mathbf{V} |\ d(\mathbf{v}, \mathbf{w}_k) \leq d(\mathbf{v}, \mathbf{w}_j)\ \forall j\}. \tag{2.2}$$

Sei $P(\mathbf{v})$ die Dichteverteilung der Datenpunkte aus $\mathbf{V}$. Der durchschnittliche Beschreibungsfehler aufgrund der Aufteilung in Voronoi Zellen (2.2) ist dann gegeben durch

$$E = \int P(\mathbf{v}) \cdot d(\mathbf{v}, \mathbf{w}_{s(\mathbf{v})})\, d\mathbf{v}. \tag{2.3}$$

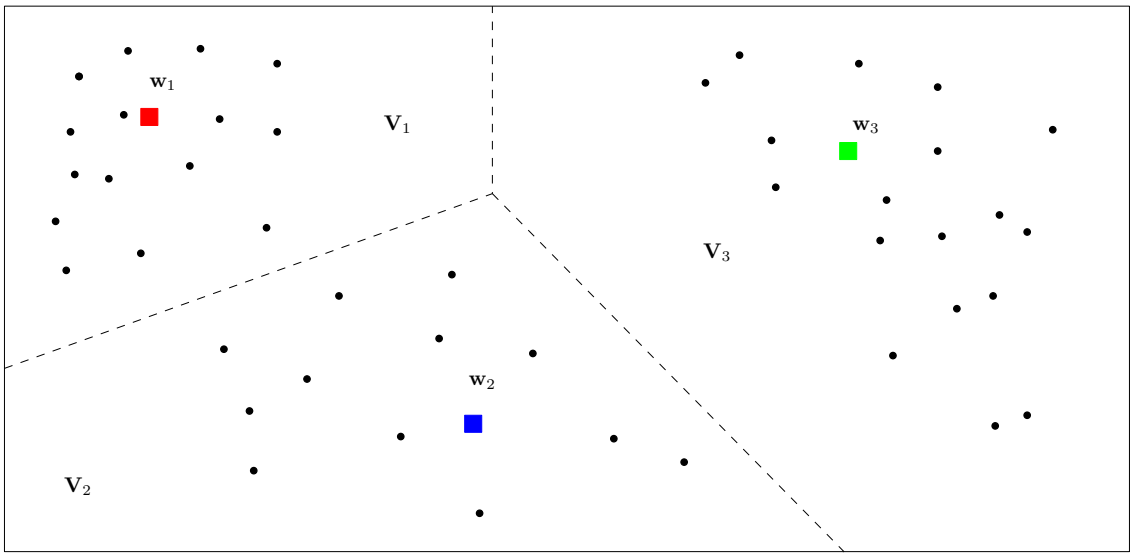


Abbildung 2.1: Eine beispielhafte Einteilung des Datenraumes $\mathbf{V}$ in Voronoi Zellen $\mathbf{V}_k$

Der Beschreibungsfehler E (2.3) ist nun durch eine „optimale“ Verteilung der Prototypen $\mathbf{w}_j$ im Sinne eines quadratischen Beschreibungsfehlers zu minimieren. Zur Lösung dieser Aufgabe gibt es verschiedene Ansätze [Mar92].

Hat man einen Satz Prototypen ermittelt, so können die Daten anhand der Abstandsbestimmung (2.1) dem nächstgelegenen Prototyp zugeordnet werden. Diese Aufteilung der Daten wird Clustern genannt. Man kann zwischen hartem Clustern und fuzzy Clustern unterscheiden. Beim harten Clustern wird ein Datenpunkt genau einem Cluster zugeordnet. Soll ein Datenpunkt hingegen zu bestimmten Teilen zu verschiedenen Clustern gehören, so handelt es sich um eine fuzzy Zuordnung. Im Folgenden wird der Neural Gas als Vertreter für hartes Clustern und der Fuzzy Neural Gas als Vertreter für fuzzy Clustern vorgestellt.

2.2 Neural Gas

Der von Thomas Martinetz entwickelte Neural Gas Algorithmus (NG) [Mar92] ist ein Verfahren der Vektorquantisierung, das einen stochastischen Gradientenabstieg auf einer Energiefunktion realisiert. Er verbindet die Idee einer „soft-max“ Adaptierung der Prototypen aus Annealingverfahren [RGF90] mit Anlehnungen an den Kohonenalgorithmus [Koh82c, Koh82a, Koh82b].

Für jeden präsentierten Datenpunkt $\mathbf{v}$ wird gemäß der quadratisch, euklidischen Quasi-Metrik [PD05]

$$d_E^2 = \|\mathbf{v} - \mathbf{w}_j\|_E^2 \tag{2.4}$$

die Rangfolge der Prototypen $\mathbf{w}_{j_0}, \mathbf{w}_{j_1}, \ldots, \mathbf{w}_{j_{N-1}}$ bestimmt. Hierbei ist $\mathbf{w}_{j_0}$ der nächstgelegene Prototyp zum Datenpunkt $\mathbf{v}$, $\mathbf{w}_{j_1}$ der Zweitnächste und $\mathbf{w}_{j_l}$, $l = 0, \ldots, N-1$ der

Prototyp für den l Vektoren $\mathbf{w}_j$ mit $\|\mathbf{v}-\mathbf{w}_j\|_E^2 \leq \|\mathbf{v}-\mathbf{w}_{j_l}\|_E^2$ existieren (Abb. 2.2).

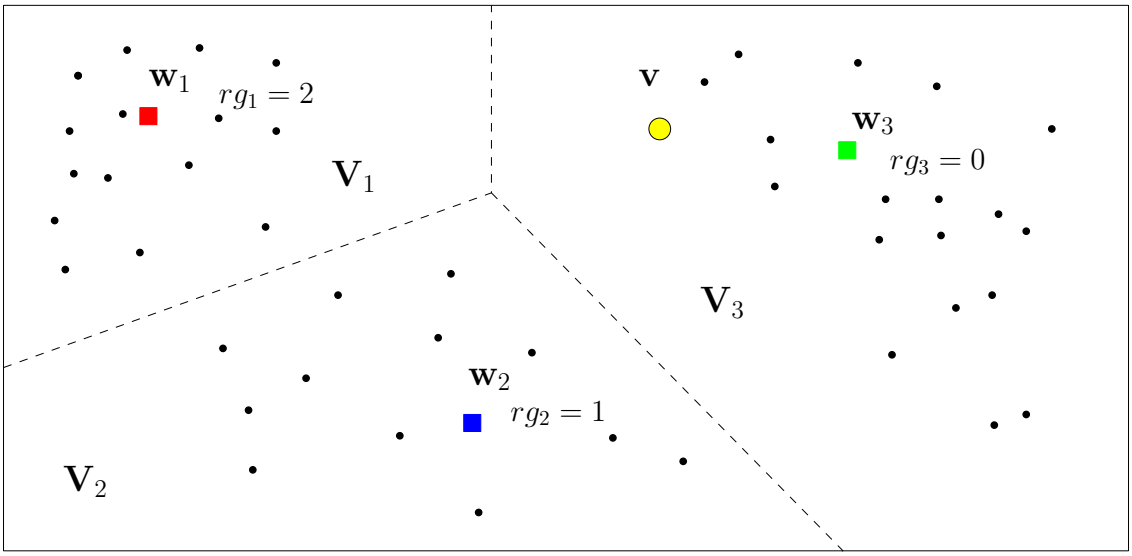


Abbildung 2.2: Rangfolge der Prototypen für den Datenpunkt $\mathbf{v}$: $\mathbf{w}_3, \mathbf{w}_2, \mathbf{w}_1$

Sei $rg_j(\mathbf{v}, \mathbf{W})$ der Nachbarschaftsrang des Prototypen $\mathbf{w}_j$, dann gilt $rg_j = l$ für $\mathbf{w}_{j_l}$. Dann lässt sich der Rang rg_j eines jeden Prototypen $\mathbf{w}_j$ nach folgender Vorschrift berechnen [MBS93]:

$$rg_j(\mathbf{v}, \mathbf{W}) = \sum_{i=1}^{N} H(d(\mathbf{v}, \mathbf{w}_j) - d(\mathbf{v}, \mathbf{w}_i)), \tag{2.5}$$

wobei $H(x)$ die Heaviside-Funktion ist:

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

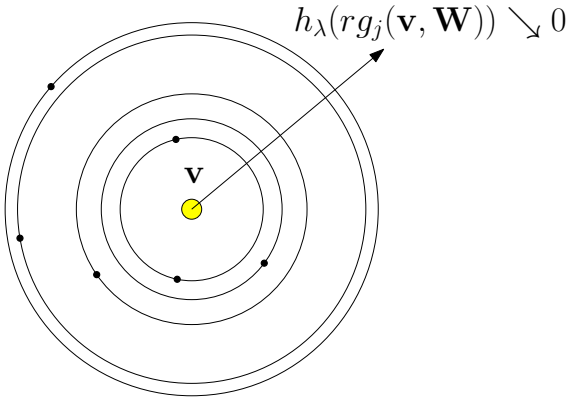


Abbildung 2.3: Einfluss der Nachbarschaftsfunktion (2.9) auf die Adaptierung der Prototypen $\mathbf{w}_j$

Sowohl der Datenpunkt als auch die Prototypen sind maßgebend für die Rangfolge der Prototypen. Eine Funktion $h_\lambda(rg_j(\mathbf{v},\mathbf{W}))$ bestimmt das Ausmaß der Veränderung des j-ten Prototypen. Hierbei gilt, dass $h_\lambda(rg_j(\mathbf{v},\mathbf{W})) = 1$ für $rg_j = 0$ und für steigendes rg_j konvergiert $h_\lambda(rg_j(\mathbf{v},\mathbf{W}))$ monoton fallend gegen Null mit einer charakteristischen Abklingkonstanten λ. Je näher ein Prototyp am präsentierten Datenpunkt liegt, desto stärker wird er verändert bzw. adaptiert (Abb. 2.3). Die Idee, innerhalb einer Iteration nicht nur den Sieger-Prototypen $\mathbf{w}_{s(\mathbf{v})}$ zu verändern, sondern alle Prototypen, wird als „soft-max“ Adaptierung bezeichnet.

Die Energiefunktion des Neural Gas lautet:

$$E_{NG} = \frac{1}{2} \cdot \frac{1}{C(\lambda)} \int \sum_{j=1}^{N} P(\mathbf{v}) \cdot h_\lambda(rg_j(\mathbf{v},\mathbf{W})) \cdot (\mathbf{v} - \mathbf{w}_j)^2 d\mathbf{v}. \tag{2.6}$$

$C(\lambda)$ ist ein Normalisierungsfaktor, der nur von der Abklingkonstanten λ abhängt. Die Energiefunktion E_{NG} (2.6) kann als derjenige Beschreibungsfehler interpretiert werden, der minimiert werden muss, falls ein Datenpunkt $\mathbf{v}$ nicht nur durch den am nächsten liegenden Prototypen $\mathbf{w}_{s(\mathbf{v})}$ beschrieben wird, sondern durch ein gewichtetes Mittel aus allen Prototypen $\mathbf{w}_j$. Die Gewichtsfaktoren sind hierbei mit $h_\lambda(rg_j(\mathbf{v},\mathbf{W}))/C(\lambda)$ durch den jeweiligen Nachbarschaftsrang eines Prototyps $\mathbf{w}_j$ zum Datenpunkt $\mathbf{v}$ gegeben. Mit dem Parameter λ, der die Nachbarschaftsreichweite einer Iteration bestimmt, verändert man die Gestalt der Energiefunktion E_{NG} (2.6). Für $\lambda \to \infty$ wird E_{NG} parabolisch, für $\lambda \to 0$ hingegen approximiert E_{NG} den eigentlichen Beschreibungsfehler E (2.3) mit seinen vielen lokalen Minima (zu einer Diskussion vgl. [VHB09]).

In jeder Iteration des Neural Gas werden alle Prototypen wie folgt adaptiert:

$$\Delta\mathbf{w}_j = -\frac{\partial E_{NG}}{\partial \mathbf{w}_j} = \varepsilon \cdot h_\lambda(rg_j(\mathbf{v},\mathbf{W})) \cdot (\mathbf{v} - \mathbf{w}_j),\ j = 1,\ldots,N. \tag{2.7}$$

Hierbei ist die partielle Ableitung nach $\mathbf{w}_j$ der Energiefunktion E_{NG} (2.6) in Gleichung (2.7) als stochastischer Gradientenabstieg zu verstehen. Thomas Martinetz hat gezeigt, dass die Dynamik der Prototypen (2.7) im Mittel den Gradientenabstieg auf der Energiefunktion E_{NG} (2.6) beschreibt. Betrachtet man eine Menge Prototypen $\mathbf{W} = \{\mathbf{w}_1,\ldots,\mathbf{w}_N\}$, so ist die mittlere Änderung $\langle\Delta\mathbf{w}_j\rangle$ eines Prototypen mit Adaptierung (2.7) durch die Mittelung von (2.7) über die Datendichte $P(\mathbf{v})$ gegeben und man erhält

$$\langle\Delta\mathbf{w}_j\rangle = \varepsilon \int P(\mathbf{v}) \cdot\ h_\lambda(rg_j(\mathbf{v},\mathbf{W})) \cdot (\mathbf{v} - \mathbf{w}_j) d\mathbf{v}. \tag{2.8}$$

Die Gleichung (2.8) entspricht dem Gradienten der Energiefunktion E_{NG} (2.6).

Die Lernrate $0 < \varepsilon \ll 1$ gibt die Intensität der Adaptierung an. Weiterhin muss ε für die Konvergenz des Neural Gas [KC78] folgende Bedingungen erfüllen:

$$1. \sum_{i=1}^{\infty} \varepsilon_i = \infty$$
$$2. \sum_{i=1}^{\infty} \varepsilon_i^2 < \infty$$

Hierbei ist ε_i die Lernrate für die i-te Iteration bzw. der i-te $Trainingsschritt$. Bei der Umsetzung des Neural Gas wurde

$$h_\lambda(rg_j(\mathbf{v}, \mathbf{W})) = e^{\left(-\frac{rg_j(\mathbf{v},\mathbf{W})}{\lambda}\right)} \tag{2.9}$$

gewählt. Setzt man die Adaptierung der Prototypen (2.7) um, so werden die Prototypen gemäß ihres Ranges zum Datenpunkt hingezogen.

Wie bereits erwähnt erfolgt zu Beginn der Adaptation bei hohem λ zunächst ein stochastischer Gradientenabstieg auf einer Parabelfläche und das Verfahren gelangt in das globale Optimum. Im Laufe des Verfahrens wird λ langsam verringert. Dadurch nimmt E_{NG} (2.6) langsam die Gestalt des Beschreibungsfehlers E (2.3) an. Man erwartet, dass sich die lokalen Minima hinreichend langsam bilden, und daher das System während der Übergangsphase in der Lage ist, dem Ort des globalen Minimums zu folgen. Zusammenfassen lässt sich der Neural Gas im Algorithmus 1.

Algorithmus 1 Neural Gas

Eingabe: Datenmenge $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$

```
Initialisierung
for i = 0 to Trainingsschritte do
    Wähle zufällig einen Datenpunkt v ∈ V
    Bestimme die Rangfolge der Prototypen rg_j(v, W) gemäß (2.5)
    Bestimme die Nachbarschaftsfunktion h_λ(rg_j(v, W)) gemäß (2.9)
    Adaptiere alle Prototypen gemäß (2.7), d. h. w_j = w_j + Δw_j
end for
```

Der Neural Gas Algorithmus, der mittels stochastischem Gradientenabstieg die Energiefunktion E_{NG} (2.6) minimiert, kann für eine endliche Datenmenge $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M\}$ als schnellere Batch Variante formuliert werden [CHHV06, Has09], die nun erläutert wird: Für eine endliche Datenmenge geht die Energiefunktion des Neural Gas E_{NG} (2.6) über in

$$E_{NG} = \frac{1}{2} \cdot \sum_{j=1}^{N} \sum_{k=1}^{M} h_\lambda(rg_j(\mathbf{v}_k, \mathbf{W})) \cdot (\mathbf{v}_k - \mathbf{w}_j)^2.$$

In jeder Iteration des Neural Gas Batch werden alle Prototypen gleichzeitig gemäß der Datenmenge adaptiert. Die neuen Prototypen müssen demzufolge effizient berechnet

werden. Analysiert man dieses Optimierungsproblem, so ist offensichtlich, dass

$$\frac{\partial E_{NG}}{\partial \mathbf{w}_j} = \sum_{k=1}^{M} h_\lambda(rg_j(\mathbf{v}_k, \mathbf{W})) \cdot (\mathbf{v}_k - \mathbf{w}_j) \stackrel{!}{=} 0$$

wegen der Rangfunktion nicht explizit gelöst werden kann. Somit kann nicht sofort eine Regel zur Adaptierung der Prototypen hergeleitet werden. Deswegen wird eine Menge mit Variablen κ_{jk} zum Ersetzen der Rangfunktion $rg_j(\mathbf{v}_k, \mathbf{W})$ eingeführt. Hierbei gilt, dass $\{\kappa_{jk} | j \in \{1,\ldots,N\}\}$ eine Permutation von $\{0,1,\ldots,N-1\}$ für alle $j \in \{1,\ldots,N\}$ ist. Das führt zu

$$\hat{E}_{NG} = \frac{1}{2} \cdot \sum_{j=1}^{N} \sum_{k=1}^{M} h_\lambda(\kappa_{jk}) \cdot (\mathbf{v}_k - \mathbf{w}_j)^2. \tag{2.10}$$

Nach einer alternierenden Optimierungstechnik vorgestellt von Hathaway und Bezdek [BH03] kann man abwechselnd die Variablen κ_{jk} mit fixierten Prototypen und die Prototypen mit konstanten κ_{jk} adaptieren. Unter Verwendung der Energiefunktion (2.10) erhält man die folgende Regel zur Adaptierung der Prototypen:

$$\mathbf{w}_j = \frac{\sum_{k=1}^{M} h_\lambda(\kappa_{jk}) \cdot \mathbf{v}_k}{\sum_{k=1}^{M} h_\lambda(\kappa_{jk})} \quad \forall j. \tag{2.11}$$

Regel (2.11) kann so interpretiert werden, dass die Prototypen eine Linearkombination der Datenpunkte sind, bei der sich im Lauf des Verfahrens die skalaren Faktoren der Datenpunkte verändern. Die ausführliche Herleitung kann in der Dissertation von Hasenfuß [Has09] nachvollzogen werden. Zusammenfassen lässt sich der Neural Gas Batch im Algorithmus 2.

Algorithmus 2 Neural Gas Batch

Eingabe: Datenmenge $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$

```
Initialisierung
for i = 0 to Trainingsschritte do
    Bestimme die Rangfolge der Prototypen κ_jk = rg_j(v_k, W) gemäß (2.5)
    Bestimme die Nachbarschaftsfunktion h_λ(κ_jk) gemäß (2.9)
    Adaptiere alle Prototypen gemäß (2.11)
end for
```

2.3 Fuzzy Neural Gas

Der Fuzzy Neural Gas (FNG) [VGKL11] kombiniert die Vorgehensweise des Fuzzy-C-Means [Dun73, Bez81] mit der „soft-max“ Adaptierung aus dem Neural Gas [Mar92]. In diesem Algorithmus ist es möglich, Datenpunkte $\mathbf{v}_k$ mehreren Prototypen $\mathbf{w}_j$ zuzuordnen.

Die Stärke der Zugehörigkeit eines Datenpunktes $\mathbf{v}_k$ zu einem Prototyp $\mathbf{w}_j$ wird durch sogenannte *fuzzy assignments* $u_{kj} \in [0,1], \forall k, j$ ausgedrückt. Für diese gilt:

$$\sum_{j=1}^{N} u_{kj} = 1 \ \forall k. \tag{2.12}$$

Die Energiefunktion des Fuzzy Neural Gas lautet

$$E_{FNG} = \sum_{k=1}^{M} \sum_{j=1}^{N} (u_{kj})^m \cdot lc_\lambda(\mathbf{v}_k, \mathbf{w}_j), \tag{2.13}$$

wobei der Parameter $m > 1$ die *fuzziness* ausdrückt. In Analogie zum Neural Gas werden die lokalen Kosten $lc_\lambda(\mathbf{v}_k, \mathbf{w}_j)$ über eine Nachbarschaftsfunktion h und ein Ähnlichkeitsmaß d definiert:

$$lc_\lambda(\mathbf{v}_k, \mathbf{w}_j) = \sum_{l=1}^{N} h_\lambda(\mathbf{w}_j, \mathbf{w}_l) \cdot d(\mathbf{v}_k, \mathbf{w}_l). \tag{2.14}$$

Hervorzuheben ist, dass die Nachbarschaftsfunktion h des Fuzzy Neural Gas nur die Prototypen untereinander in Beziehung setzt, wobei im Neural Gas die Nachbarschaftsbeziehung von Prototypen bzgl. des Datenpunktes berücksichtigt werden. Die Nachbarschaftsfunktion h ist definiert als

$$h_\lambda(\mathbf{w}_j, \mathbf{w}_l) = c_\lambda \cdot e^{-\frac{1}{2} \cdot \left(\frac{rg_j(\mathbf{w}_l, \mathbf{W})}{\lambda} \right)^2}. \tag{2.15}$$

Der Rang rg_j eines jeden Prototypen $\mathbf{w}_j$ wird berechnet nach folgender Vorschrift:

$$rg_j(\mathbf{w}_l, \mathbf{W}) = \sum_{i=1}^{N} H(d(\mathbf{w}_l, \mathbf{w}_j) - d(\mathbf{w}_l, \mathbf{w}_i)). \tag{2.16}$$

Die Nachbarschaftsreichweite in (2.15) wird über dem Parameter $\lambda > 0$ definiert. Analog zum Neural Gas ist λ eine Abklingkonstante und geht im Laufe des Algorithmus gegen Null. Der Parameter c_λ wird so gewählt, dass die Gleichung

$$\sum_{l=1}^{N} h_\lambda(\mathbf{w}_j, \mathbf{w}_l) = 1, \forall j$$

gilt. Ziel des Fuzzy Neural Gas ist die Minimierung der Energiefunktion E_{FNG} (2.13) unter Berücksichtigung der Gleichungsnebenbedingungen (2.12). Durch Anwendung des Lagrange-Formalismus [ERSD77] lassen sich die Regeln zur Adaptierung der Prototypen und der *fuzzy assignments* herleiten. Die Adaptierung der Prototypen und der *fuzzy assignments* folgt einem Optimierungsschema mit zwei Schritten [BH03], das als Alternating Optimization bezeichnet wird. Zuerst werden die Prototypen adaptiert, wobei die *fuzzy assignments* als konstant angesehen werden. Danach bleiben die Prototypen fixiert, um die *fuzzy assignments* zu berechnen. Sei $d = d_E^2$ (aus 2.4), dann ergeben sich

die folgenden Update-Regeln:

$$\mathbf{w}_j = \frac{\sum_{k=1}^{M}\sum_{l=1}^{N}(u_{kl})^m \cdot h_\lambda(\mathbf{w}_j, \mathbf{w}_l) \cdot \mathbf{v}_k}{\sum_{k=1}^{M}\sum_{l=1}^{N}(u_{kl})^m \cdot h_\lambda(\mathbf{w}_j, \mathbf{w}_l)} \tag{2.17}$$

$$u_{kj} = \left(\sum_{l=1}^{N}\left(\frac{lc_\lambda(\mathbf{v}_k, \mathbf{w}_j)}{lc_\lambda(\mathbf{v}_k, \mathbf{w}_l)}\right)^{\frac{1}{m-1}}\right)^{-1} \tag{2.18}$$

Zusammenfassen lässt sich der FNG wie folgt:

Algorithmus 3 Fuzzy Neural Gas

Eingabe: Datenmenge $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$, *fuzzy assignments* u_{kj}

Initialisierung
for $i = 0$ to $Trainingsschritte$ **do**
  $iter = 0$
  $\Delta = eps$
  while $\Delta > eps$ und $iter \leq iter_{max}$ **do**
    $iter = iter + 1$
    Bestimme die Rangfolge der Prototypen $rg_j(\mathbf{w}_l, \mathbf{W})$ gemäß (2.16)
    Bestimme die Nachbarschaftsfunktion $h_{\lambda_i}(rg_j(\mathbf{w}_j, \mathbf{w}_l))$ gemäß (2.15)
    Adaptiere alle Prototypen gemäß (2.17)
    $\Delta = \max_{j=1,\ldots,N} |\mathbf{w}_j^{neu} - \mathbf{w}_j|$
    Bestimme die Rangfolge der Prototypen $rg_j(\mathbf{w}_l, \mathbf{W})$ gemäß (2.16)
    Bestimme die Nachbarschaftsfunktion $h_{\lambda_i}(rg_j(\mathbf{w}_j, \mathbf{w}_l))$ gemäß (2.15)
    Berechne die lokalen Kosten $lc_{\lambda_i}(\mathbf{v}_k, \mathbf{w}_j)$ gemäß (2.14)
    Adaptiere alle *fuzzy assignments* u_{kj} gemäß (2.18)
  end while
end for

3 Modifikationen des Neural Gas

3.1 Neural Gas mit funktionalen Prototypen

Zur Anwendung des Neural Gas Algorithmus 1 genügt die Kenntnis der Datenpunkte $\mathbf{v}$. Sind jedoch zusätzliche Informationen bzgl. des Datensatzes bekannt, sollten diese nach Möglichkeit im Algorithmus genutzt werden. Existiert zum Beispiel ein funktionaler Zusammenhang in den Daten, wie beim Tecator Datensatz aus Abschnitt 5.1.1, so ist dies eine charakteristische Eigenschaft des Datensatzes. Der Datenpunkt $\mathbf{v}$ wird nun als diskreter Repräsentant der zugrundeliegenden Funktion aufgefasst. Man interpretiert die t-te Komponente des Datenpunktes $\mathbf{v}$ als Funktionswert der zugrundeliegenden Funktion an der Stelle t. Die Idee ist nun, diese charakteristische Eigenschaft im Algorithmus zu verwerten. Umsetzen kann man das, indem die Prototypen den funktionalen Charakter der Daten widerspiegeln. Ein Prototyp wird folglich ebenfalls als diskreter Repräsentant einer Funktion aufgefasst und $w(t)$ sei die t-te Komponente des Prototypen.

Im Neural Gas ist jeder Prototyp $\mathbf{w}_j$ von seiner Gestalt her ein Vektor. Stellt man den Prototyp als Linearkombination (LK) von unabhängigen Basisfunktionen (3.1) dar, so bekommt dieser einen funktionalen Charakter:

$$w_j(t) = \sum_{i=1}^{K} \beta_{i_j} \cdot \mathscr{K}_{i_j}(t) \quad j = 1,\ldots,N, \;\; t = 1,\ldots,D. \tag{3.1}$$

Hierbei sind β_{i_j} die Gewichte mit $\beta_{i_j} \in \mathbb{R}, \forall i, j$ und $K \in \mathbb{N}$ gibt die Anzahl der Basisfunktionen an. Die Basisfunktionen $\mathscr{K}_{i_j}$ können dem Datensatz entsprechend gewählt werden. Mögliche Basisfunktionen sind zum Beispiel die Gauß- (Abb. 3.1) oder Lorentzfunktion. Wird als Basisfunktion $\mathscr{K}_{i_j}$ die Gaußfunktion gewählt, so kann jeder einzelne Prototyp $\mathbf{w}_j$

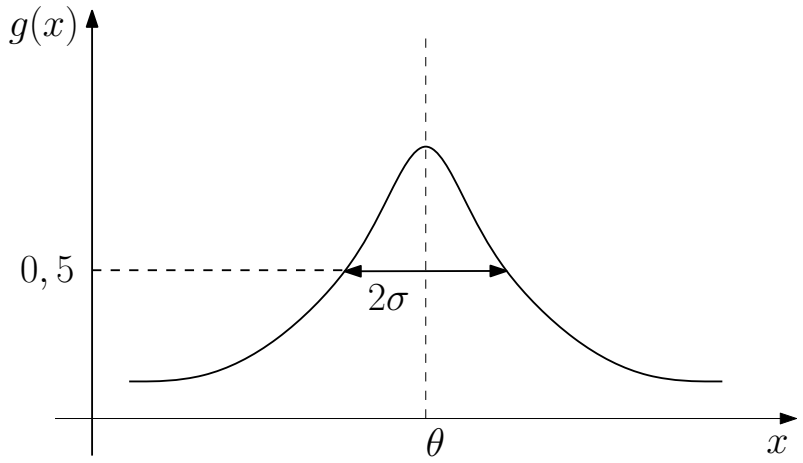


Abbildung 3.1: Die Gaußfunktion $g(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma} e^{-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2}$

als Linearkombination von Gaußfunktionen (Abb. 3.2) dargestellt werden:

$$w_j(t) = \sum_{i=1}^{K} \beta_{i_j} \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_{i_j}} \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{i_j}}{\sigma_{i_j}}\right)^2} \quad j = 1,\ldots,N, \;\; t = 1,\ldots,D. \tag{3.2}$$

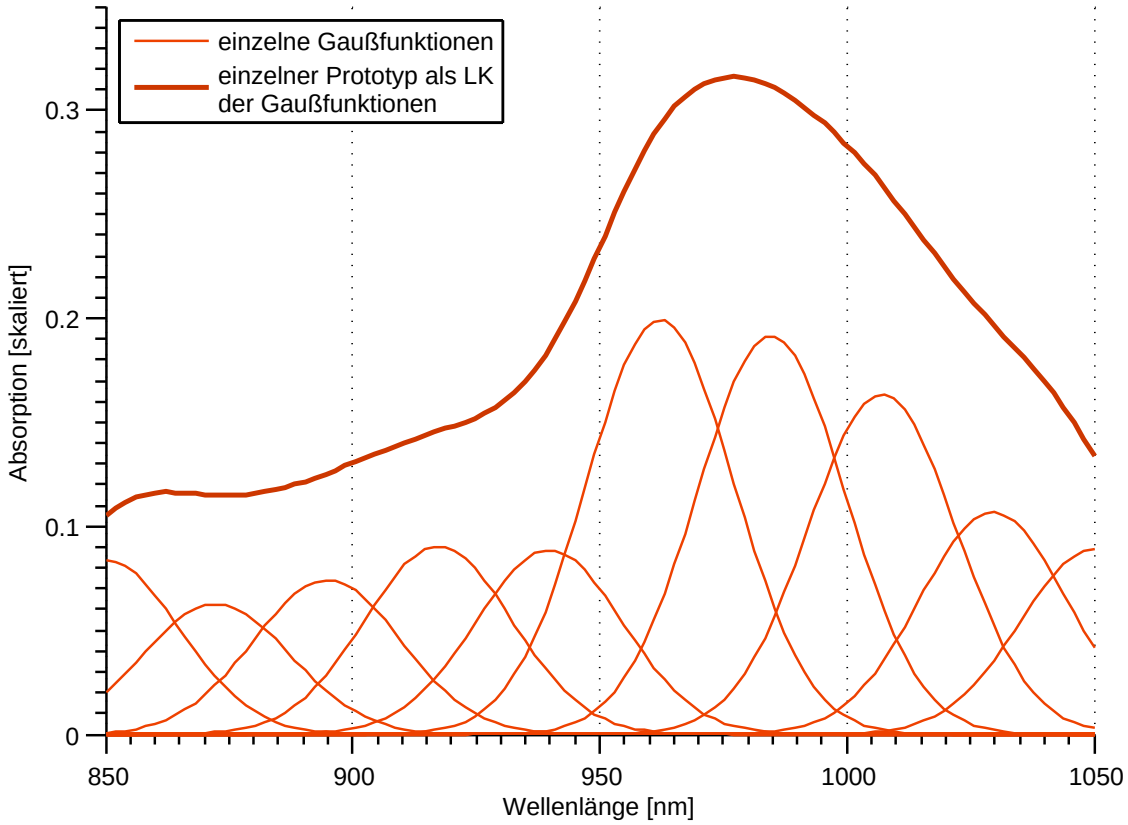


Abbildung 3.2: Ein Prototyp für den Tecator Datensatz als Linearkombination von Gaußfunktionen

Entsprechend den funktionalen Prototypen muss deren Regel zur Adaptierung angepasst werden. In der ersten Umsetzung werden nur die Gewichte β_{i_j} adaptiert und in der zweiten kommt die Adaption der Stützstellen θ_{i_j} hinzu. Nicht adaptiert werden die Breiten σ_{i_j}, da sich die Breiten und Gewichte der Gaußfunktionen zu stark beeinflussen [Hay09]. Die Parameter, die nicht adaptiert werden, bleiben nach der Initialisierung konstant.

Betrachtet man die Variante, in der nur die Gewichte β_{i_j} adaptiert werden, genauer, kann man feststellen, dass die gewählten Basisfunktionen linear unabhängig sind (siehe Anhang A.1). Anhand dieser Umsetzung kann zunächst überprüft werden, ob das Verfahren mit Prototypen als Linearkombination von Gaußfunktionen konvergiert und datenrepräsentierende Prototypen zustande kommen, oder ob sich Instabilitäten zeigen. Lernt man nur die Gewichte β_{i_j}, so sind die Basisfunktionen fest gewählt. Man hat sich auf ein Basissystem festgelegt. Dies kann sich als Nachteil erweisen, wenn das Basissystem schlecht gewählt wurde bzw. den Daten nicht entspricht. Für ein festes

Basissystem kann man auch zuerst die Daten in dieses Basissystem transformieren und erhält die entsprechenden Gewichte bzw. Koeffizienten der Gaußfunktionen. Mit den ermittelten Koeffizienten kann man anschließend in dem entsprechenden Koeffizientenraum mit dem normalen Neural Gas weiterrechen um die Gewichte bzw. Koeffizienten der Gaußfunktionen zu erhalten.
Ein positiver Nebeneffekt ist, dass die Anzahl der zu lernenden Parameter reduziert wird im Vergleich zum originalen Neral Gas (Alg. 1). In der Standardvariante des Neural Gas müssen pro Prototyp die Vektorkomponenten adaptiert werden, die den enzelnen Dimensionen der Daten entsprechen. Üblicherweise ist die Dimension D von funktionalen Daten sehr hoch. Verwendet man hingegen Prototypen als Linearkombination von Gaußfunktionen, so werden pro Prototyp nur K Gewichte gelernt und in der Regel ist $K \ll D$.
Bei der zweiten Umsetzung werden zusätzlich die Stützstellen θ_{i_j} gelernt. Somit lernt der Neural Gas in der zweiten Umsetzung nicht nur den Einfluss der Gaußfunktionen, sondern das Basissystem, das den Daten entspricht. Es sind nur diese zwei Umsetzungen realisiert, da weitere Umsetzungen zur Instabilität neigen, die im Folgenden erklärt wird [Hay09]. Eine Umsetzung, in der zusätzlich noch die Breiten σ_{i_j} gelernt werden, ist nicht realisiert, da sich die Gewichte und die Breiten der Gaußfunktionen zu stark beeinflussen und der Algorithmus dann oft instabil wird [Hay09]. Im Folgenden wird auf die Herleitung der Adaptionsregeln eingegangen, um das neue Vorgehen vorzustellen.

Wie bereits weiter vorn erläutert, lautet die Energiefunktion des Neural Gas (2.6):

$$E_{NG} = \frac{1}{2} \cdot \frac{1}{C(\lambda)} \int \sum_{j=1}^{N} P(\mathbf{v}) \cdot h_\lambda(rg_j(\mathbf{v}, \mathbf{W})) \cdot \underbrace{(\mathbf{v} - \mathbf{w}_j)^2}_{=d_E^2(\text{aus } 2.4)} d\mathbf{v}.$$

Die Adaptionsregel für die Gewichte β_{i_j} entspricht einem stochastischen Gradientenabstieg auf E_{NG} nach β_{i_j}, d. h.

$$\beta_{i_j}^{neu} = \beta_{i_j} - \varepsilon_\beta \cdot \frac{\partial E_{NG}}{\partial \beta_{i_j}}, \tag{3.3}$$

wobei ε_β die Lernrate der Gewichte β_{i_j} beschreibt. Für die Nachbarschaftsfunktion h_λ (2.9) wird die folgende verkürzte Schreibweise verwendet:

$$h_\lambda(\mathbf{v}, \mathbf{w}_j) = h_\lambda(rg_j(\mathbf{v}, \mathbf{W})).$$

Die Gewichte β_{i_j} werden gemäß des stochastischen Gradientenabstiegs (3.3) und den partiellen Ableitungen (A.2, A.3, A.4) wie folgt aktualisiert:

$$\begin{aligned} \beta_{k_j}^{neu} &= \beta_{k_j} + \varepsilon_\beta \cdot h_\lambda(\mathbf{v}, \mathbf{w}_j) \cdot \sum_{t=1}^{D} (v(t) - \sum_{i=1}^{K} \beta_{i_j} \cdot \mathscr{K}_{i_j}(t)) \cdot \mathscr{K}_{k_j}(t) \quad \forall k, j \\ &= \beta_{k_j} + \varepsilon_\beta \cdot h_\lambda(\mathbf{v}, \mathbf{w}_j) \cdot \sum_{t=1}^{D} (v(t) - w_j(t)) \cdot \mathscr{K}_{k_j}(t) \qquad \forall k, j. \end{aligned} \tag{3.4}$$

Zusammenfassend lässt sich der Neural Gas mit funktionalen Prototypen und Adaptierung der Gewichte im Algorithmus 4 beschreiben.

Algorithmus 4 Neural Gas mit funktionalen Prototypen; Adaptierung der Gewichte β_{i_j}

Eingabe: Datensatz $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Funktionale Prototypen $\mathbf{w}_j$

```
Initialisierung
for i = 0 to Trainingsschritte do
  Wähle zufällig einen Datenpunkt v ∈ V
  Bestimme die Rangfolge der Prototypen rg_j(v, W) gemäß (2.5)
  Bestimme die Nachbarschaftsfunktion h_λ(rg_j(v, W)) gemäß (2.9)
  Adaptiere alle Gewichte β_{i_j} gemäß (3.4)
  Berechne die neuen Prototypen gemäß (3.2)
end for
```

Um die Stützstellen θ_{i_j} der Gaußfunktionen in den Lernprozess einzubeziehen, geht man analog vor. Die Adaption der Stützstellen θ_{i_j} mittels stochastischem Gradientenabstiegs und der Lernrate ε_θ lautet demzufolge

$$\theta_{i_j}^{neu} = \theta_{i_j} - \varepsilon_\theta \cdot \frac{\partial E_{NG}}{\partial \theta_{i_j}}. \tag{3.5}$$

Die Aktualisierung der θ_{i_j} erfolgt nach (3.5) unter Berücksichtigung von (A.5), (A.6) und (A.7), d. h.

$$\theta_{k_j}^{neu} = \theta_{k_j} + \varepsilon_\theta \cdot h_\lambda(\mathbf{v}, \mathbf{w}_j) \cdot \sum_{t=1}^{D} (v(t) - w_j(t)) \cdot \beta_{k_j} \cdot \frac{t - \theta_{k_j}}{\sigma_{k_j}^3 \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \quad \forall k, j. \tag{3.6}$$

Zusammenfassen lässt sich der Neural Gas mit funktionalen Prototypen und Adaptierung der Gewichte und der Stützstellen im Algorithmus 5. Einzelheiten zur Implementierung und zu Ergebnissen sind im Anhang C und in Kapitel 5.3 ausführlich erläutert.

Algorithmus 5 Neural Gas mit funktionalen Prototypen; Adaptierung der Gewichte β_{i_j} und der Stützstellen θ_{i_j}

Eingabe: Datensatz $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Funktionale Prototypen $\mathbf{w}_j$

```
Initialisierung
for i = 0 to Trainingsschritte do
  Wähle zufällig einen Datenpunkt v ∈ V
  Bestimme die Rangfolge der Prototypen rg_j(v, W) gemäß (2.5)
  Bestimme die Nachbarschaftsfunktion h_λ(rg_j(v, W)) gemäß (2.9)
  Adaptiere alle Gewichte β_{i_j} gemäß (3.4)
  Adaptiere alle Stützstellen θ_{i_j} gemäß (3.6)
  Berechne die neuen Prototypen gemäß (3.2)
end for
```

3.2 Neural Gas mit funktionalen Prototypen und Sobolev-Norm

Bislang wurde in den vorgestellten Algorithmen stets die quadratische euklidische Norm zur Abstandsbestimmung verwendet. Um Information, die in den Ableitungen von funktionalen Daten verborgen ist, aufzuspüren und zu verwerten, wird die Sobolev Quasi-Metrik [Har12] verwendet. Die Anwendung einer funktionalen Quasi-Metrik bietet sich auch deshalb an, weil die Prototypen diskrete Repräsentanten von Linearkombinationen von Basisfunktionen darstellen und somit Ableitungen analytisch berechenbar sind. Können die Ableitungen nicht analytisch berechnet werden und stehen außerdem nicht zur Verfügung, kommen Differenzenquotienten zur Bildung der Ableitung zum Einsatz. Im Folgenden wird die Sobolev Quasi-Metrik

$$d_S^p(\mathbf{v},\mathbf{w}_j) = \sum_{q=0}^{Q} \left\| \frac{d^q}{dt^q}\left(\mathbf{v}(t),\mathbf{w}_j(t)\right) \right\|_{\mathscr{L}^p}^p \quad t = 1,\ldots,D$$

als neues Abstandsmaß vorgestellt. Sie wird allerdings nicht in dieser allgemeinen Form zur Anwendung kommen, sondern mit $Q = 1, p = 2$ und einer konvexen Wichtung der einzelnen Terme (3.7):

$$d_S^2(\mathbf{v},\mathbf{w}_j) = (1-\alpha)\cdot\|\mathbf{v}-\mathbf{w}_j\|_E^2 + \alpha\cdot\|\dot{\mathbf{v}}-\dot{\mathbf{w}}_j\|_E^2, \tag{3.7}$$

wobei $\alpha \in [0,1]$ und $\dot{\mathbf{v}}$ bzw. $\dot{\mathbf{w}}_j$ als erste Ableitung nach t definiert ist.

Seien die Prototypen $\mathbf{w}_j$ Linearkombinationen von Gaußfunktionen (3.2), dann ist die quadratische Sobolev Quasi-Metrik (3.7) gegeben durch

$$\begin{aligned} d_S^2(\mathbf{v},\mathbf{w}_j) &= (1-\alpha)\cdot\sum_{t=1}^{D}\left(v(t)-w_j(t)\right)^2 + \alpha\cdot\sum_{t=1}^{D}\left(\dot{v}(t)-\dot{w}_j(t)\right)^2 \\ &= (1-\alpha)\cdot\sum_{t=1}^{D}\left(v(t)-\sum_{i=1}^{K}\beta_{i_j}\cdot\frac{1}{\sqrt{2\pi}\cdot\sigma_{i_j}}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{i_j}}{\sigma_{i_j}}\right)^2}\right)^2 \\ &\quad +\alpha\cdot\sum_{t=1}^{D}\left(\dot{v}(t)+\sum_{i=1}^{K}\beta_{i_j}\cdot\frac{t-\theta_{i_j}}{\sqrt{2\pi}\cdot\sigma_{i_j}^3}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{i_j}}{\sigma_{i_j}}\right)^2}\right)^2. \end{aligned} \tag{3.8}$$

Analog zu Abschnitt 3.1 können nun die Gradienten der Energiefunktion des Neural Gas E_{NG} mit der quadratischen Sobolev-Norm (3.8)

$$E_{NG_S} = \frac{1}{2}\cdot\frac{1}{C(\lambda)}\int\sum_{j=1}^{N}P(\mathbf{v})\cdot h_\lambda(rg_j(\mathbf{v},\mathbf{W}))\cdot d_S^2(\mathbf{v},\mathbf{w}_j)\,d\mathbf{v} \tag{3.9}$$

nach den Gewichten β_{i_j} und den Stützstellen θ_{i_j} gebildet werden. Kommen diese Gradienten in den Update-Formeln des Neural Gas Algorithmus 5 zum Einsatz, so hat

man den Neural Gas mit funktionalen Prototypen unter Verwendung der quadratischen Sobolev Quasi-Metrik. Demzufolge ergibt sich die Adaptionsregel für die Gewichte β_{k_j}:

$$\begin{aligned}\beta_{k_j}^{neu} &= \beta_{k_j} + \varepsilon_\beta \cdot h_\lambda(\mathbf{v}, \mathbf{w}_j) \cdot \left(\tfrac{2}{\pi}\right)^{\frac{1}{2}} \cdot \left[\frac{\alpha-1}{\sigma_{k_j}} \cdot \sum_{t=1}^{D} \left(v(t) - w_j(t)\right) \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \right. \\ &\quad \left. + \frac{\alpha \cdot (t-\theta_{k_j})}{\sigma_{k_j}^3} \cdot \sum_{t=1}^{D} \left(\dot{v}(t) - \dot{w}_j(t)\right) \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \right]\end{aligned}$$

und für die Stützstellen θ_{k_j}

$$\begin{aligned}\theta_{k_j}^{neu} &= \theta_{k_j} + \varepsilon_\theta \cdot h_\lambda(\mathbf{v}, \mathbf{w}_j) \cdot \frac{-\beta_{k_j}}{\sqrt{2\pi}\sigma_{k_j}^3} \cdot \left[(1-\alpha) \cdot \sum_{t=1}^{D} \left(v(t) - w_j(t)\right) \cdot (t - \theta_{k_j}) \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \right. \\ &\quad \left. + \alpha \cdot \sum_{t=1}^{D} \left(\dot{v}(t) - \dot{w}_j(t)\right) \cdot \left(\frac{(t-\theta_{k_j})^2}{{\sigma_{k_j}}^2} - 1 \right) \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \right].\end{aligned}$$

Die ausführlichen Schritte zur Herleitung der vorangegangenen Formeln können im Anhang A.4 nachvollzogen werden.

4 Pulsierende Vektorquantisierung

4.1 Pulsing Neural Gas

Der Neural Gas hat sich als Vektorquantisierer in seinem Anwendungsbereich bewährt. Er beruht auf der Minimierung der Energiefunktion E_{NG} unter Verwendung eines stochastischen Gradientenabstiegs. In dem Gebiet der Optimierung hat sich die Heuristik des Simulated Annealing (SA) ebenfalls als gutes Verfahren etabliert. Bei der Minimierung der Energiefunktion des Neural Gas handelt es sich letztlich auch um ein Optimierungsproblem. Der Neural Gas konvergiert in der Theorie zum globalen Minimum, allerdings benötigt er hierfür unendlich lange und die Nachbarschaftsreichweite λ muss hinreichend langsam abgekühlt werden. Kühlt man λ zu schnell ab, kann der Neural Gas dem globalem Optimum der Energiefunktion E_{NG} (2.6) beim Übergang von parabolisch zu gebirgig nicht folgen. In der Praxis kann endliches Lernen daher zu lokalen Minima führen. Um die Wahrscheinlichkeit für dieses Verhalten zu reduzieren, wird die λ-Abkühlstrategie des Neural Gas mit Simulated Annealing für bessere Konvergenz überlagert. Besonders wichtig ist dies für den Neural Gas Batch, da dieser oft in lokalen Minima trotz vorsichtiger λ-Abkühlstrategie stecken bleibt.

Zu diesem Zweck wird an dieser Stelle Simulated Annealing [KGV83] kurz vorgestellt. Die Grundidee dieses Verfahrens stammt aus der Thermodynamik, speziell aus dem physikalischen Abkühlungsvorgang. Beim Herstellen von Stahl kann beispielsweise zweierlei passieren:
Kühlt man den Stahl zu schnell ab, so haben die Stahlatome wenig Zeit sich in Gitterstruktur anzuordnen und der Stahl wird spröde. Lässt man dem Stahl hingegen genügend Zeit zum Abkühlen oder erwärmt ihn zwischenzeitlich wieder, um die Beweglichkeit der Atome zu erhöhen, so gelingt es eher eine gleichmäßige Gitterstruktur zu erhalten und somit qualitativ hochwertigen Stahl herzustellen. Dieser Vorgang des langsamen Abkühlens und zwischenzeitlichen Erwärmens ist der Kern von Simulated Annealing.

Im Folgenden wird nun Simulated Annealing mathematisch formuliert:
Seien eine Zielfunktion $f(\mathbf{x}) \rightarrow min$ und eine Menge zulässiger Lösungen $\mathbf{X}$ gegeben, wobei $f : \mathbf{X} \rightarrow \mathbb{R}$. Ausgehend von einer zulässigen Lösung $\mathbf{x} \in \mathbf{X}$, wird aus der Nachbarschaft $N(\mathbf{x}) \subseteq \mathbf{X}$ eine zulässige Lösung $\mathbf{x}^{neu}$ ermittelt. Gilt $f(\mathbf{x}^{neu}) < f(\mathbf{x})$, so wird das Verfahren mit $\mathbf{x}^{neu}$ fortgesetzt. Andernfalls wird die neue Lösung $\mathbf{x}^{neu}$ nur mit einer gewissen Wahrscheinlichkeit $p(\Delta, T)$ akzeptiert. Hierbei ist $\Delta = f(\mathbf{x}^{neu}) - f(\mathbf{x})$ und T entspricht der Temperatur, die sich im Laufe des Verfahrens verringert. Die Wahrscheinlichkeit $p(\Delta, T)$ berechnet sich dann wie folgt:

$$p(\Delta, T) = e^{-\frac{\Delta}{T}}.$$

Durch die Abkühlung der Temperatur werden zu Beginn des Verfahrens häufiger Verschlechterungen akzeptiert, zum Ende hingegen fast nur noch Verbesserungen. Zusammenfassen lässt sich Simulated Annealing in dem Algorithmus 6:

Algorithmus 6 Simulated Annealing

Eingabe:
Zielfunktion $f(\mathbf{x}) \rightarrow min$
Menge zulässiger Lösungen $\mathbf{X}$
Zulässige Startlösung $\mathbf{x}$
Ausgabe: optimale zulässige Lösung $\mathbf{x}^*$

$\mathbf{x}^* = \mathbf{x}$
$z^* = f(\mathbf{x})$
while Abbruchkriterium nicht erfüllt **do**
  Konstruiere Nachbarschaft $N(\mathbf{x}^*) \subseteq \mathbf{X}$
  Wähle $\mathbf{x}^{neu} \in N(\mathbf{x}^*)$
  if $f(\mathbf{x}^{neu}) < z^*$ **then**
    $\mathbf{x}^* = \mathbf{x}^{neu}$
    $z^* = f(\mathbf{x}^{neu})$
  else
    $\mathbf{x}^{neu}$ wird mit Wahrscheinlichkeit $p(\Delta, T)$ (4.1) akzeptiert, $\mathbf{x}^* = \mathbf{x}^{neu}$, $z^* = f(\mathbf{x}^{neu})$
  end if
end while
Gib $\mathbf{x}^*$ aus.

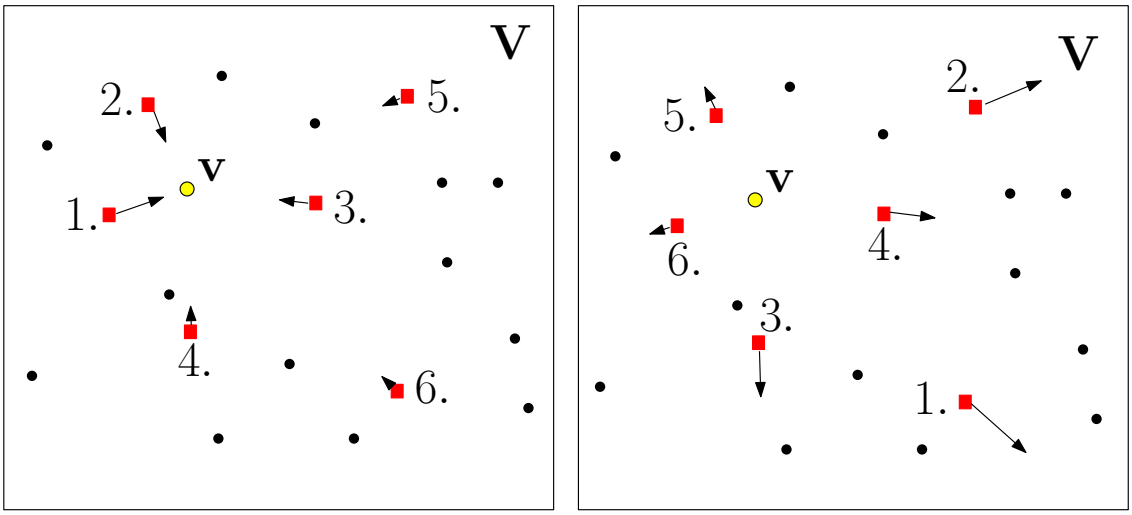
Im Folgenden geht es um die Kombination von Neural Gas (Alg. 1) und Simulated Annealing (Alg. 6). Dieser neue Algorithmus wird mit Pulsing Neural Gas (PNG) bezeichnet. Im Neural Gas wird ein stochastischer Gradientenabstieg realisiert. Da die Auswertung der Energiefunktion des Neural Gas E_{NG} (2.6) im Allgemeinen sehr aufwendig ist, soll bei der Kopplung mit Simulated Annealing eine Entscheidung mittels Vergleich der Zielfunktionswerte zweier Sätze von Prototypen vermieden werden.

Abweichend von Simulated Annealing wird während des Verfahrens in jedem Schritt entschieden, ob eine bewusste Verschlechterung der Prototypen zugelassen wird, um aus lokalen Minima zu gelangen. Diese bewusste Verschlechterung wird als *negativer Lernschritt* bezeichnet. Das bedeutet konkret:
Im Neural Gas wird in einem Lernschritt ein Datenpunkt $\mathbf{v}$ zufällig ausgewählt und die Rangfolge der Prototypen $\mathbf{w}_j$ gemäß deren Abstand zum Datenpunkt bestimmt. Anschließend werden die Prototypen entsprechend ihrem Rang zum Datenpunkt hingezogen (Abb. 4.1.1). In einem negativen Lernschritt wird nun die Rangfolge der Prototypen beginnend mit dem größten Abstand zum Datenpunkt gebildet. Außerdem werden die Prototypen nicht zum Datenpunkt hingezogen, sondern abgestoßen (Abb. 4.1.2) gemäß

$$\mathbf{w}_j^{neu} = \mathbf{w}_j + \frac{\partial E_{NG}}{\partial \mathbf{w}_j} = \mathbf{w}_j - \Delta \mathbf{w}_j,\ j = 1, \ldots, N. \tag{4.1}$$

Da die Abstände der Prototypen zum Datenpunkt sehr groß sein können, kann es pas-

sieren, dass die Prototypen durch das Abstoßen die konvexe Hülle der Daten verlassen bzw. können Instabilitäten auftreten. Um diesem Problem auszuweichen, werden beim negativen Lernschritt die Abstände normiert.



4.1.1: Positiver Lernschritt beim PNG (2.7)

4.1.2: Negativer Lernschritt beim PNG (4.1)

Abbildung 4.1: Die Lernschritte beim Pulsing Neural Gas

Weiterhin wird das Verhalten von Simulated Annealing übernommen, dass zu Beginn des Verfahrens öfter Verschlechterungen zugelassen werden und zum Ende fast nur noch Verbesserungen vorgenommen werden. In jedem Adaptionsschritt wird mit einer Wahrscheinlichkeit $p(T)$ entschieden, ob ein positiver oder negativer Lernschritt erfolgt. Die Wahrscheinlichkeit $p(T)$[1] wird wie folgt berechnet:

$$p(T) = p_0 \cdot e^{-\frac{1}{2}\cdot\left(\frac{T}{\sigma_p}\right)^2}. \tag{4.2}$$

Der Einfluss der Parameter p_0 und σ_p auf $p(T)$ sind exemplarisch in Abbildung 4.2 dargestellt. Mit dem Parameter p_0 aus der Gleichung (4.2) legt man tendenziell im Pulsing Neural Gas fest, wie oft man einen negativen Lernschritt durchführen will. Je höher p_0 desto größer ist die Wahrscheinlichkeit $p(T)$ zu Beginn des Verfahrens einen negativen Lernschritt durchzuführen. Der Parameter σ_p beeinflusst die Wahrscheinlichkeit $p(T)$ eines negativen Lernschritts während der Durchführung des Pulsing Neural Gas. Je kleiner σ_p, desto schneller nimmt die Wahrscheinlichkeit für einen negativen Lernschritt ab. Die Ausführungen zum Pulsing Neural Gas sind im Algorithmus 7 zusammengefasst.

Alternativ zu dem eben vorgestellten negativen Lernschritt kann dieser auch durch die folgende heuristische Adaptierung ersetzt werden. Hierbei werden die Prototypen in einem negativen Lernschritt nicht vom Datenpunkt weggestoßen, sondern zum Verlierer-Prototyp hingezogen (Abb. 4.3.2). Die Adaptierung der Prototypen $\mathbf{w}_j$ erfolgt dann

[1] Gibbs-Boltzmann-Wahrscheinlichkeit, es wird nicht zwischen Dichte und Verteilung unterschieden, dies ergibt sich aus dem Zusammenhang

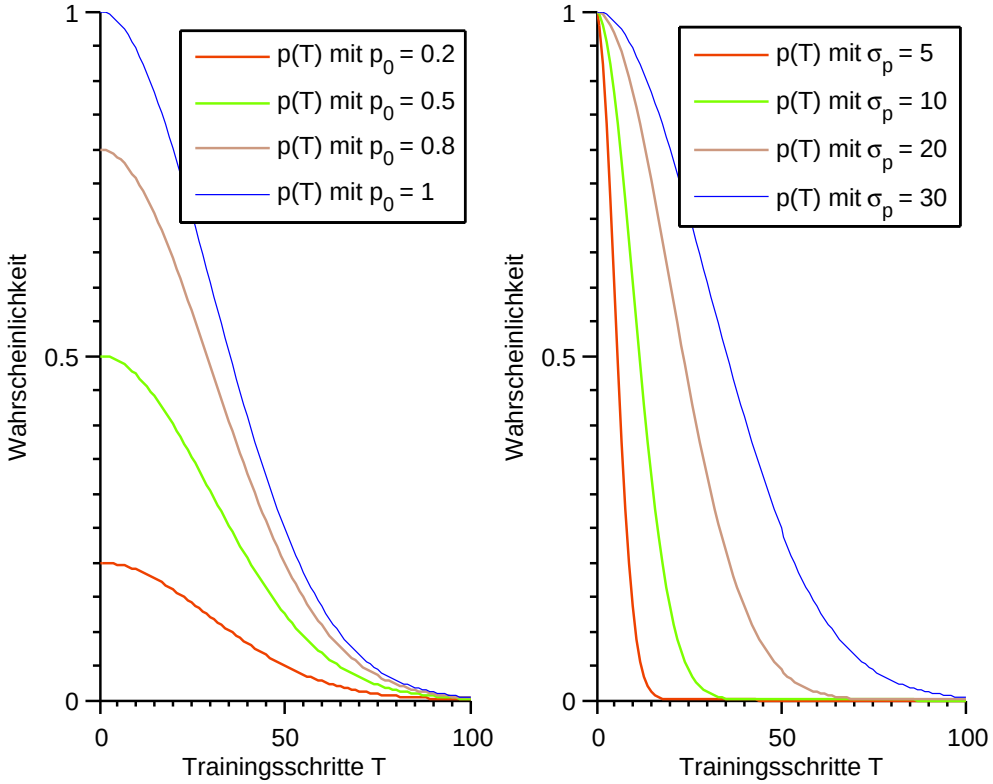


Abbildung 4.2: Der Einfluss der Parameter p_0 und σ_p auf die Wahrscheinlichkeit $p(T)$

Algorithmus 7 Pulsing Neural Gas

Eingabe: Datensatz $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$

```
Initialisierung
for T = 0 to Trainingsschritte do
    Wähle zufällig einen Datenpunkt v ∈ V
    Bestimme p(T), gleichverteilte Zufallszahl z
    if p(T) < z then
        Bestimme die Sieger-Rangfolge der Prototypen rg_j(v, W) gemäß (2.5)
        Adaptiere alle Prototypen gemäß (2.7), d. h. w_j = w_j + Δw_j
    else
        Bestimme die Verlierer-Rangfolge der Prototypen gemäß (4.4)
        Adaptiere alle Prototypen gemäß (4.1), d. h. w_j = w_j − Δw_j
    end if
end for
```

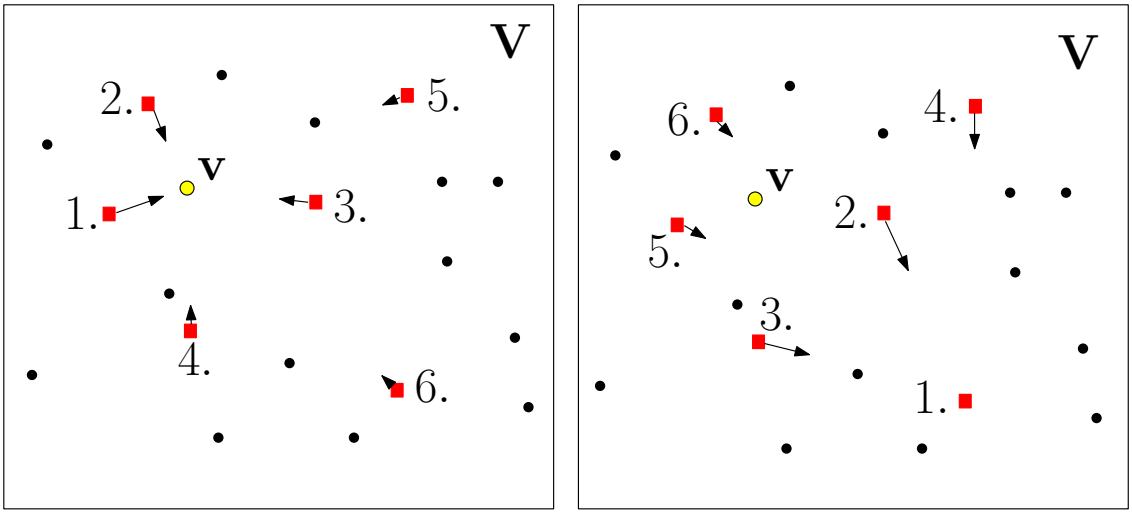
mittels

$$\Delta \mathbf{w}_j = \varepsilon \cdot h_\lambda(rg_j(\mathbf{w}_{l(\mathbf{v})}, \mathbf{w})) \cdot (\mathbf{w}_{l(\mathbf{v})} - \mathbf{w}_j),\ j = 1, \ldots, N, \tag{4.3}$$

wobei $\mathbf{w}_{l(\mathbf{v})}$ der Verlierer-Prototyp mit

$$l(\mathbf{v}) = \underset{j=1,\ldots,N}{\operatorname{argmax}}\, d(\mathbf{v}, \mathbf{w}_j) \tag{4.4}$$

ist.



4.3.1: Positiver Lernschritt beim HPNG (2.7) 4.3.2: Negativer Lernschritt beim HPNG (4.3)

Abbildung 4.3: Die Lernschritte beim Pulsing Neural Gas mit Heuristik

4.2 Pulsing Neural Gas Batch

Der Grundaufbau des Pulsing Neural Gas (Alg. 7) wird bei der Batch Variante beibehalten. In jeder Iteration wird mit einer Wahrscheinlichkeit $p(T)$ (4.2) entschieden, ob eine originale Batch-Adaptierung (2.11) der Prototypen vorgenommen wird, oder ein negativer Lernschritt. Der negative Lernschritt der Batch Variante des Pulsing Neural Gas wird sich von dem des Pulsing Neural Gas unterscheiden. Der Grund ist, dass in jeder Iteration der Batch Variante alle Datenpunkte berücksichtig werden und nicht nur ein einzelner. Auch werden die Prototypen nicht sukzessive in eine bestimmte Richtung bewegt, sondern alle Prototypen werden in einer Iteration mit einem Mal komplett im Datenraum versetzt. Zu klären ist, wie der negative Lernschritt nun konkret realisiert wird. Zur Erinnerung, die Adaptierung der Prototypen ist wie folgt (vgl. 2.11):

$$\mathbf{w}_j = \frac{\sum_{k=1}^{M} h_\lambda(\kappa_{jk}) \cdot \mathbf{v}_k}{\sum_{k=1}^{M} h_\lambda(\kappa_{jk})} \quad \forall j,$$

wobei alle Datenpunkte in die Berechnung einfließen. Um eine bewusste Verschlechterung der Prototypen zu erreichen, ist die Idee, eine zufällig ausgewählte Untermenge $\mathbf{V}_m$, $|\mathbf{V}_m| = m$, der Datenmenge $\mathbf{V}$ zu deren Berechnung zu verwenden. Durch die Verwendung der Untermenge $\mathbf{V}_m$ wird bei der Adaptierung der Prototypen weniger Information genutzt und die Prototypen haben die Möglichkeit in Bereiche des Datenraumes versetzt zu werden, in die sie mit der originalen Adaptierung selten hingelangen. Wie stark

sich der negative Lernschritt auswirkt, hängt von der Mächtigkeit m der Untermenge $\mathbf{V}_m$ ab sowie den zufällig ausgewählten Datenpunkten $\mathbf{v} \in \mathbf{V}_m$ bzw. vom Verlauf $p(T)$ (4.2), der Auswirkung auf die Anzahl der negativen Lernschritte hat. Die Mächtigkeit m der Untermenge $\mathbf{V}_m$ muss ebenfalls in jedem Schritt zufällig gewählt werden, da sonst die Anzahl der in Frage kommenden Untermengen nur $\binom{M}{m}$ beträgt und im Verhältnis zur Iterationsanzahl zu gering sein kann. In Analogie zum Pulsing Neural Gas wird im negativen Lernschritt die Verlierer-Rangfolge beibehalten. Das kann man als Abstoßung für die Sieger-Prototypen gemäß Siegerbestimmung (2.1) verstehen. Da diese beiden Elemente des negativen Lernschritts sich sehr stark auf die Prototypen auswirken und man nur eine geringe Verschlechterung zulassen möchte, wird eine konvexe Wichtung zwischen den alten und den neuen Prototypen in folgender Form eingebaut:

$$\mathbf{w}_j^{neu} = \gamma \cdot \mathbf{w}_j^{neu} + (1-\gamma) \cdot \mathbf{w}_j^{alt}, \tag{4.5}$$

wobei $\gamma \in (0,1)$ die Stärke des negativen Lernschritts beeinflusst. Alternativ kann man auch einen *reduced information learning*-Schritt (*RIL*) durchführen. In diesem wird die Sieger-Rangfolge beibehalten und nur eine zufällige Untermenge $\mathbf{V}_m$ zur Adaptierung verwendet. Hierbei benutzt man bewusst weniger Information der zur Verfügung stehenden Daten um die Prototypen bzw. den Algorithmus zu destabilisieren. Zusammenfassen lässt sich der Pulsing Neural Gas Batch im Algorithmus 8. Die genaue Durchführung des negativen Lernschrittes und des *reduced information learning*-Schrittes sind in den Algorithmen 9 und 10 dargestellt.

Algorithmus 8 Batch PNG

Eingabe: Datensatz $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$

```
Initialisierung
for T = 0 to Trainingsschritte do
  Bestimme p(T), gleichverteilte Zufallszahl z
  if p(T) < z then
    Bestimme die Sieger-Rangfolge der Prototypen κ_jk = rg_j(v_k, w) gemäß (2.5)
    Bestimme die Nachbarschaftsfunktion h_λ(κ_jk) gemäß (2.9)
    Adaptiere alle Prototypen gemäß (2.11)
  else
    Bestimme gleichverteilte Zufallszahl m GV [1, M]
    Wähle zufällig m Datenpunkte V_m aus V aus
    Negativ Lernschritt (Alg. 9) oder reduced information learning (Alg. 10)
  end if
end for
```

Algorithmus 9 Negativer Lernschritt

Bestimme die Verlierer-Rangfolge der Prototypen κ_{jk} unter Verwendung von $\mathbf{V}_m$
Bestimme die Nachbarschaftsfunktion $h_\lambda(\kappa_{jk})$ unter Verwendung von $\mathbf{V}_m$
Adaptiere alle Prototypen gemäß der konvexen Wichtung (4.5) und (2.11) unter Verwendung von $\mathbf{V}_m$

Algorithmus 10 *reduced information learning*

Bestimme die Sieger-Rangfolge der Prototypen κ_{jk} unter Verwendung von $\mathbf{V}_m$
Bestimme die Nachbarschaftsfunktion $h_\lambda(\kappa_{jk})$ unter Verwendung von $\mathbf{V}_m$
Adaptiere alle Prototypen gemäß (2.11) unter Verwendung von $\mathbf{V}_m$

4.3 Pulsing Fuzzy Neural Gas

Der Fuzzy Neural Gas (Alg. 3) ist ebenfalls ein Batch Algorithmus, da in jeder Adaptierung alle Datenpunkte in die Berechnung mit einfließen (2.17). Um das Pulsing in den Algorithmus einzubauen, wird analog zum Neural Gas Batch (Alg. 8) vorgegangen. Für den negativen Lernschritt der Prototypen wird eine zufällige Untermenge des Datensatzes $\mathbf{V}$ von zufälliger Mächtigkeit erzeugt und mit diesen Datenpunkten werden die Prototypen gemäß der Gleichung (2.17) unter Verwendung der Verlierer-Rangfolge adaptiert. Alternativ kann auch wieder der *reduced information learning*-Schritt verwendet werden. Der Algorithmus 11 gibt die Umsetzung des Pulsing Fuzzy Neural Gas (PFNG) wieder.

Algorithmus 11 Pulsing Fuzzy Neural Gas

Eingabe: Datensatz $\mathbf{V}$, Anzahl der Prototypen
Ausgabe: Prototypen $\mathbf{w}_j$, *fuzzy assignments* u_{kj}

Initialisierung
for $T = 0$ to $Trainingsschritte$ **do**
  $iter = 0; \Delta = eps$
  while $\Delta > eps$ und $iter \leq iter_{max}$ **do**
    $iter = iter + 1$
    Bestimme die Rangfolge der Prototypen $rg_j(\mathbf{w}_l, \mathbf{W})$ gemäß (2.5)
    Bestimme die Nachbarschaftsfunktion $h_{\lambda_T}(rg_j(\mathbf{w}_j, \mathbf{w}_l))$ gemäß (2.9)
    Bestimme $p(T)$, gleichverteilte Zufallszahl z
    if $p(T) < z$ **then**
      Adaptiere alle Prototypen gemäß (2.17)
    else
      Bestimme gleichverteilte Zufallszahl m $GV[1, M]$
      Wähle zufällig m Datenpunkte $\mathbf{V}_m$ aus $\mathbf{V}$ aus
      Negativ Lernschritt (Alg. 9) **oder** *reduced information learning* (Alg. 10)
    end if
    $\Delta = \max_{j=1,\ldots,N} |\mathbf{w}_j^{neu} - \mathbf{w}_j|$
    Bestimme die lokalen Kosten $lc_{\lambda_i}(\mathbf{v}_k, \mathbf{w}_j)$ gemäß (2.14)
    Adaptiere alle *fuzzy assignments* u_{kj} gemäß (2.18)
  end while
end for

5 Experimente

5.1 Daten

5.1.1 Der Tecator Datensatz

Zum Testen der in Kapitel 2 und 3 vorgestellten Algorithmen wurde der bekannte Tecator-Datensatz gewählt [Tho]. Dieser besteht aus 215 Messungen mit jeweils 100-dimensionalen Datenpunkten. Ein Datenpunkt entspricht dem Infrarotabsorbtionsspektrum einer Fleischprobe. Jede der 215 Proben bestand aus zerhacktem purem Fleisch mit unterschiedlichem Wasser-, Fett- und Eiweißanteil. Die Infrarotabsorbtionsspektren sind mit einem „Tecator Infratec Food and Feed Analyzer“, der mit einer Wellenlänge von 850 bis 1050 nm im nahen Infrarot arbeitet, aufgenommen. Dieser hat eine spektrale Auflösung von 2 nm. Die Daten sind gemäß ihres Fettgehaltes in zwei Klassen eingeteilt. Ein Überblick über die Daten ist in Abbildung 5.1 gegeben. Man erkennt leicht, dass sich beide Klassen stark überlappen und eine Trennung nur schwer möglich ist.

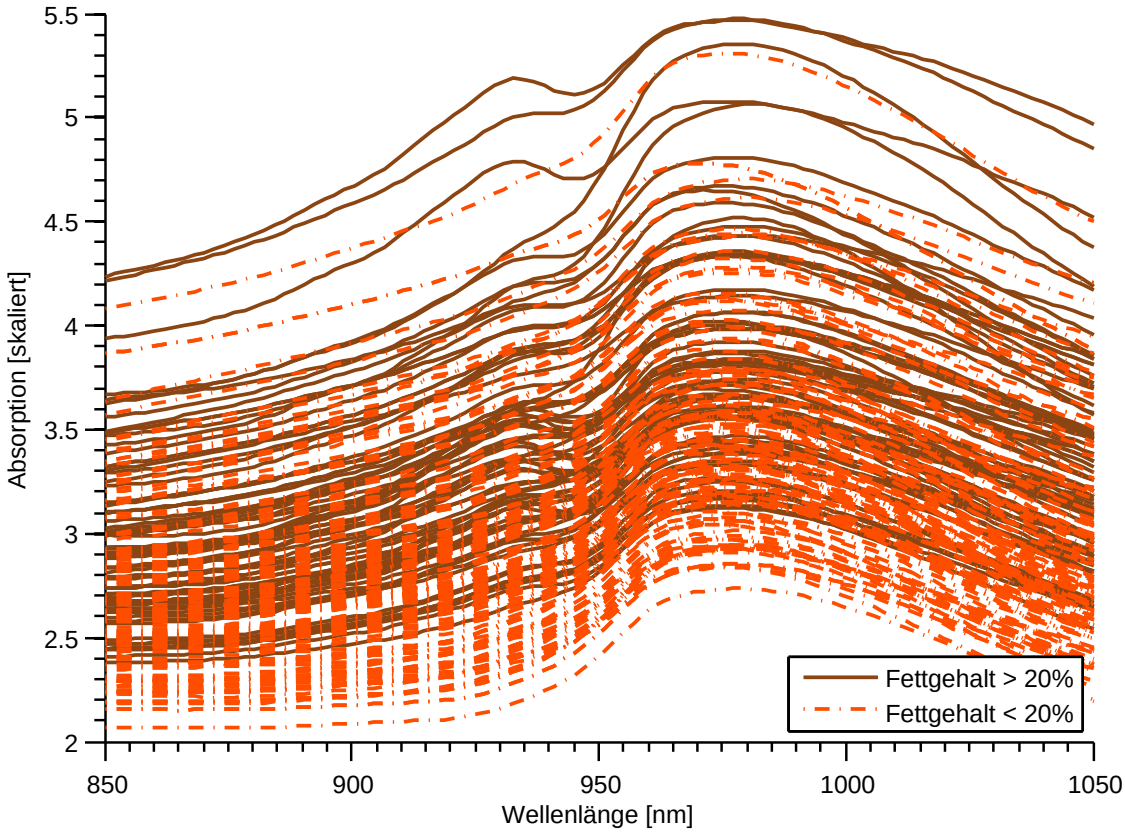


Abbildung 5.1: Der Tecator Datensatz

Für Testzwecke wurde der Tecator Datensatz mit kleinen standard normalverteilten Zufallszahlen verrauscht (Abb. 5.2).

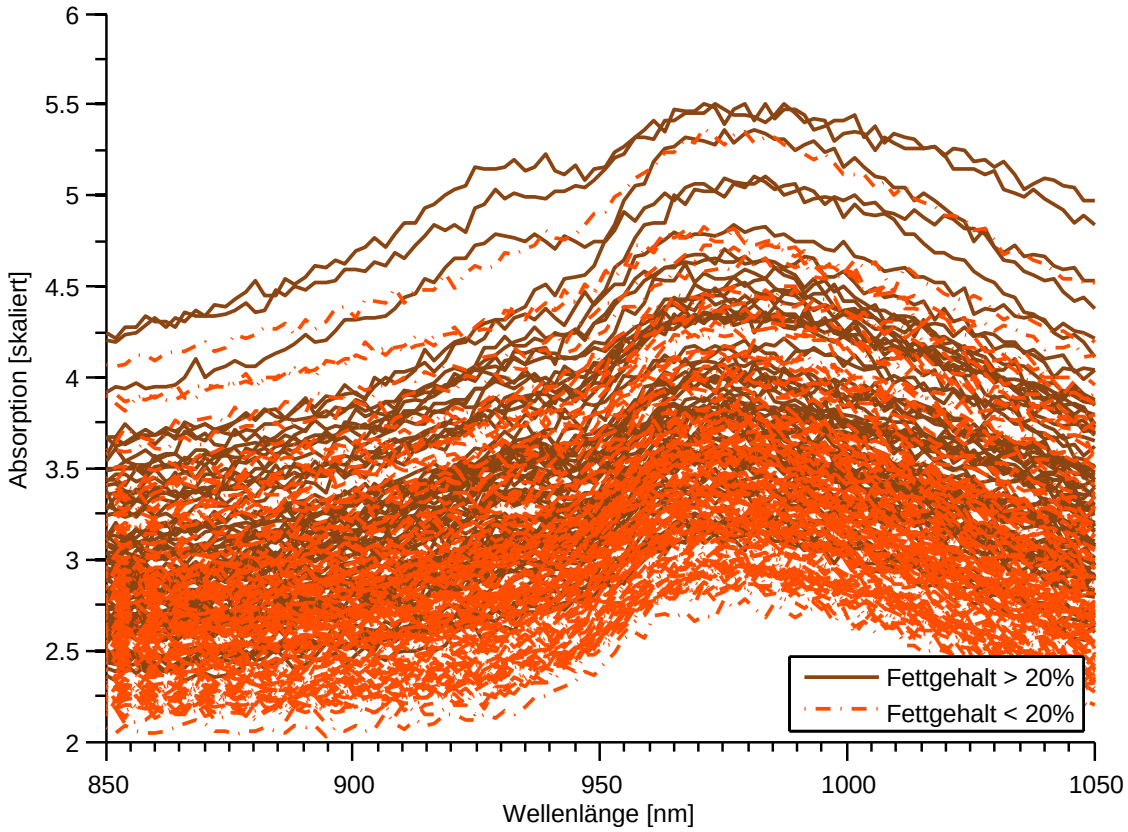


Abbildung 5.2: Der verrauschte Tecator Datensatz

5.1.2 Schachbrett Datensatz

Der Schachbrett Datensatz enthält 11 250 Datenvektoren aus dem $\mathbb{R}^2$, deren Komponenten normalverteilte Zufallszahlen $\mathcal{N}(0,1)$ enthalten. Zwischen den Mittelpunkten der Cluster beträgt der mittlere Abstand 1,5. Auf Grund der geringen Datendimension ($D = 2$) eignet sich dieser Datensatz hervorragend zur Visualisierung (Abb. 5.3). Wie in Abbildung 5.3 zu erkennen ist, beinhaltet der Datensatz 15×15 Felder bzw. Cluster. Diese Cluster sind optisch gesehen sehr kompakt und gut voneinander trennbar und eignen sich gut um Ergebnisse der vorgestellten Pulsing Algorithmen zu validieren und darzustellen.

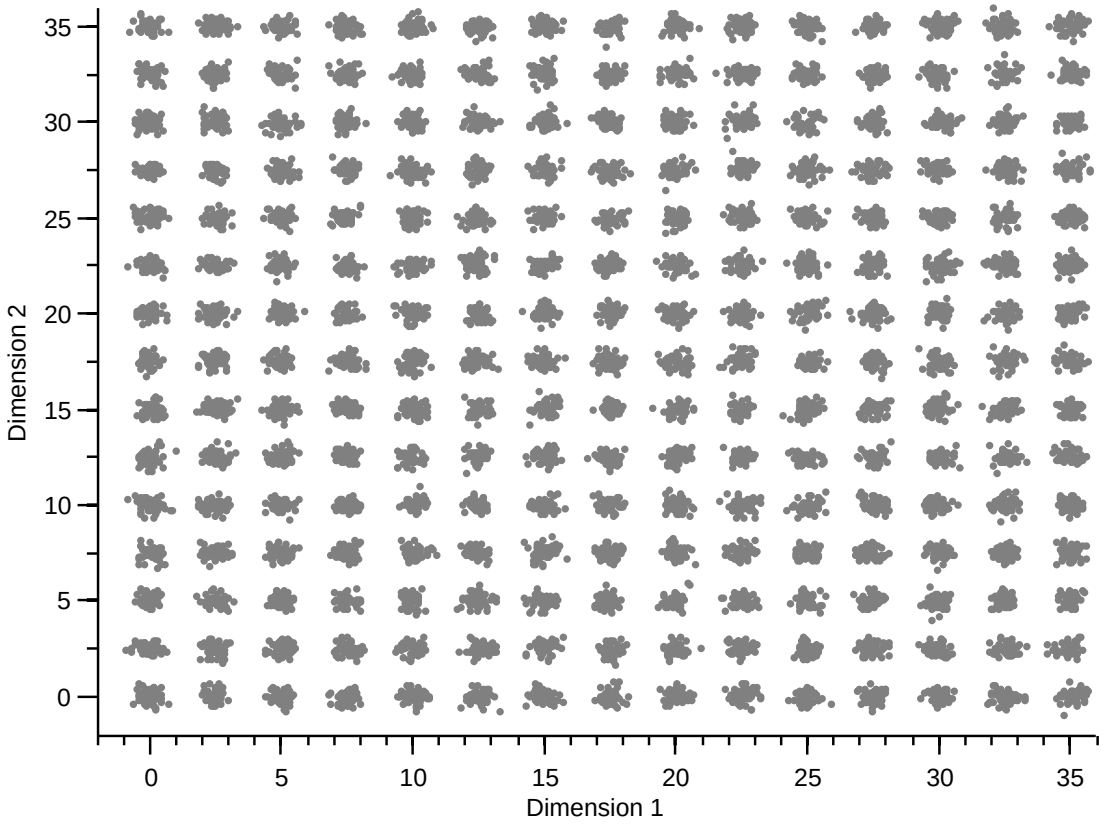


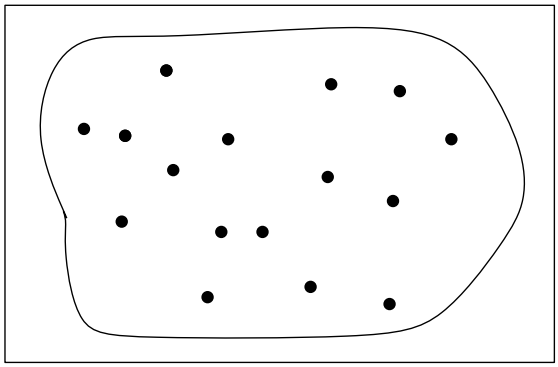
Abbildung 5.3: Der Schachbrett Datensatz

5.2 Validierung von Clusterlösungen

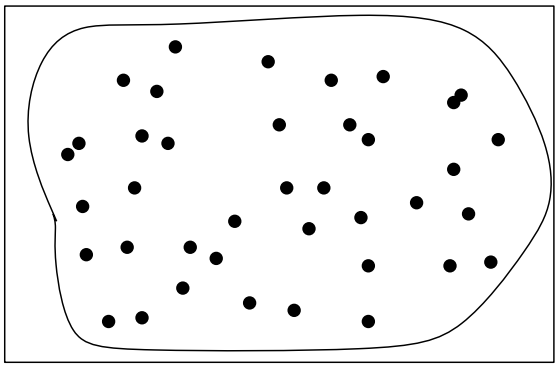
5.2.1 Begriffsklärung

Das Clustern und die Bewertung einer Clusterlösung sind nicht korrekt gestellte Probleme (ill-posed problems) und je nach Sichtweise wird ein und dieselbe Lösung verschieden bewertet. Mögliche Gütekriterien sind Kompaktheit und Separation.

Unter Kompaktheit eines Clusters versteht man die Dichte der Datenpunkte in den einzelnen Clustern. In Abbildung 5.4 sind zwei Cluster mit unterschiedlicher Datendichte dargestellt. In dem Cluster mit geringerer Dichte (Abb. 5.4.1) spricht man von einem nicht kompakten Cluster. Hingegen in dem Cluster mit den vielen Datenpunkten (Abb. 5.4.2) handelt es sich um ein kompaktes Cluster. Ein weiteres Gütekriterium für Clusterlösungen ist die Separation, d. h. wie stark überlappen sich einzelne Cluster (Abb. 5.5). In Abbildung 5.5.1 überlappen sich die drei Cluster und sind demzufolge nicht voneinander separiert. Hingegen in Abbildung 5.5.2 sind die drei Cluster frei von Überschneidungen zu separieren.
Im nächsten Abschnitt werden verwendete Cluster Validierungsmaße vorgestellt, welche die eben vorgestellten Gütekriterien berücksichtigen.

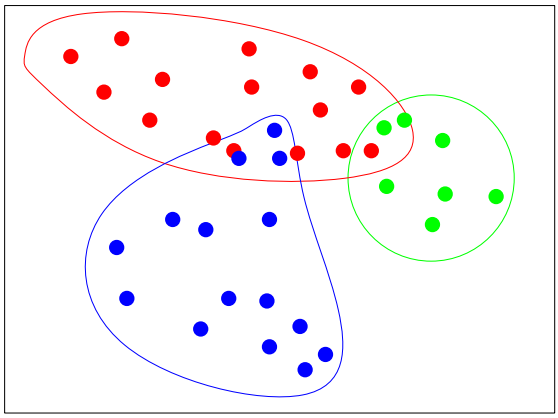


5.4.1: Cluster mit wenig Daten

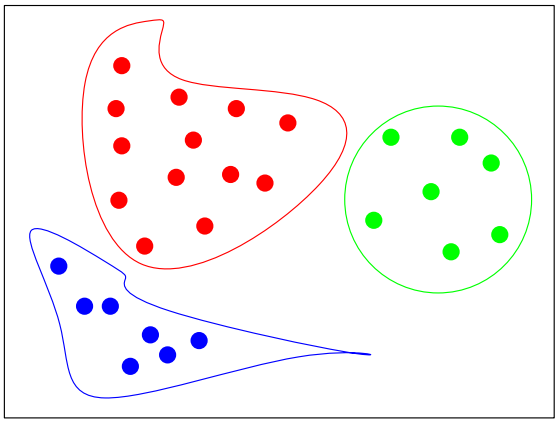


5.4.2: Cluster mit vielen Daten

Abbildung 5.4: Die Kompaktheit von Clustern



5.5.1: Überlappende Cluster



5.5.2: Separierte Cluster

Abbildung 5.5: Die Separierbarkeit von Clustern

5.2.2 Cluster Validierungsmaße

In diesem Abschnitt werden die zur Auswertung und zum Vergleich der erhaltenen Lösungen verwendeten Validierungsmaße kurz zum Verständnis erläutert. Ein mögliches Validierungsmaß ist der XieBeni-Index (XB) [XB91]. Dieser Index sagt aus, dass eine Clusterlösung gut ist, wenn die Kompaktheit minimal und die Separation maximal ist. Je kleiner der Wert des XieBeni-Index ist, desto „besser“ ist die Clusterlösung. Berechnet wird der Index für Ergebnisse des Fuzzy Neural Gas wie folgt:

$$XB = \frac{\sum_{j=1}^{N} \sum_{i=1}^{M} (u_{ij})^m \cdot d(\mathbf{v}_i, \mathbf{w}_j)}{\min_{j \neq l} d(\mathbf{w}_j, \mathbf{w}_l)}. \tag{5.1}$$

Hierbei gibt m die *fuzziness* an und u_{ij} sind die *fuzzy assignments* (vgl. Abschnitt 2.3). Für harte Clusterlösungen kann der XieBeni-Index ebenfalls verwendet werden, indem

man die *fuzzy assignments* u_{ij} wie folgt anpasst

$$u_{ij} = \begin{cases} 1, j = s(\mathbf{v}_i) \\ 0, sonst \end{cases}$$

mit $s(\mathbf{v}_i)$ dem Index des Sieger-Prototyps (2.1).

Außerdem kommt der SV-Index [ZZ11] zum Einsatz:

$$SV = \frac{S}{V} = \frac{\sum_{j=1}^{N} \min_{i=1,\ldots,N,\ i\neq j} d(\mathbf{w}_i, \mathbf{w}_j)}{\sum_{j=1}^{N} \max_{i\in\mathbf{V}_j} d(\mathbf{v}_i, \mathbf{w}_j)}. \tag{5.2}$$

Hierbei repräsentiert V die maximale Varianz und beschreibt somit wie verschieden die Datenpunkte eines Clusters sind. Ein niedriger Wert ist demzufolge ein Indikator für Ähnlichkeit. S misst die minimalen Abstände zwischen zwei Clusterzentren und gibt somit Aufschluss über die Separation. Je größer S ist, desto besser. Der SV-Index misst Separation gegen Varianz und je größer der Wert des Indexes ist, umso „besser“ ist die Clusterlösung. Dieser Index ist erweiterbar zum SVF-Index [ZZ11], so dass er auch für Ergebnisse des Fuzzy Neural Gas anwendbar ist.

$$SVF = \frac{S}{V_F} = \frac{\sum_{j=1}^{N} \min_{i=1,\ldots,N,\ i\neq j} d(\mathbf{w}_i, \mathbf{w}_j)}{\sum_{j=1}^{N} \max_{i\in\mathbf{V}_j} (u_{ij})^m \cdot d(\mathbf{v}_i, \mathbf{w}_j)}.$$

Gemäß den verwendeten Algorithmen wird die Metrik in den Validierungsmaßen entsprechend angepasst. In dieser Arbeit wurde entweder die quadratische Euklidische Distanz oder die Sobolev Quasi-Metrik verwendet. Die Ergebnisse sind mittels der Validierungsmaße unter Verwendung des im Algorithmus angewendeten Abstandsmaßes ausgewertet.

5.3 Ergebnisse

5.3.1 Ergebnisse Neural Gas mit funktionalen Prototypen zum Abschnitt 3.1

Zur Identifikation des Einflusses der einzelnen Parameter des Neural Gas mit funktionalen Prototypen wurden Rechnungen durchgeführt, in denen immer nur ein Parameter verändert wurde.
Als erstes wird der Einfluss von K, der Anzahl der Gaußfunktionen, untersucht. Zum

besseren Verständnis wird dies anhand der Abbildung 5.6 erläutert. Grau eingezeichnet ist der Tecator Datensatz. Er dient als Referenz, ob die farbig dargestellten Prototypen dem Datensatz entsprechen.

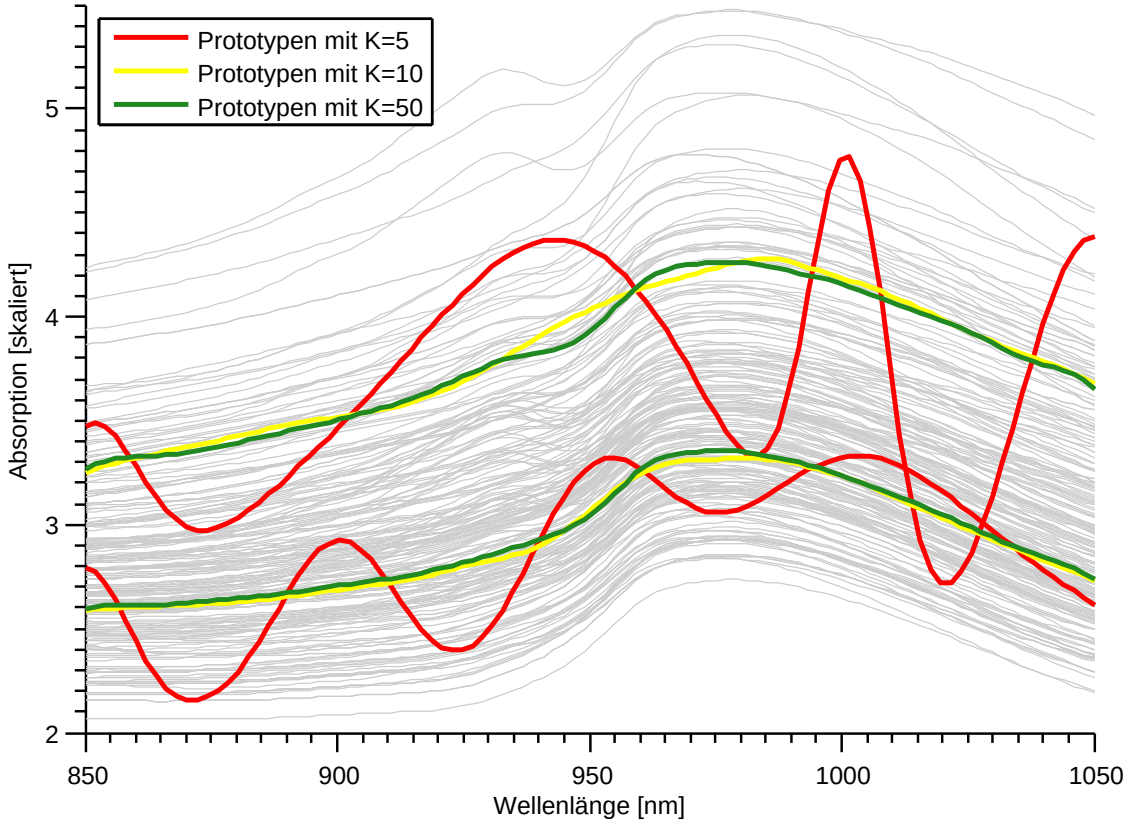


Abbildung 5.6: Der Einfluss der Anzahl K der Gaußfunktionen bei der Version des Neural Gas mit funktionalen Prototypen (Alg. 4)
Die in gleicher Farbe dargestellten Prototypen sind das Ergebnis eines Durchlaufes des Algorithmus 4, wobei K entsprechend gewählt wurde.

In Abbildung 5.6 sind Prototypen für den Tecator Datensatz dargestellt, wobei K wie angegeben variiert und nur die Gewichte β_{i_j} gelernt sind. Man erkennt, dass eine zu geringe Anzahl (hier: $K = 5$) an Gaußfunktionen zu datenuntypischen Prototypen führt. Der Grund dafür ist, dass durch die geringe Anzahl an Basisfunktionen und fixierten Stützstellen sich insbesondere bei den Gaußfunktionen die Maxima ausprägen und demzufolge auch ausgeprägte Minima zwischen den Maxima entstehen. Die Minima bzw. Täler können aufgrund der geringen Anzahl an Basisfunktionen nicht ausgeglichen werden und die Prototypen bekommen eine so kurvige Erscheinung wie in Abbildung 5.6 für $K = 5$ dargestellt. Sobald K hinreichend groß gewählt wird, tritt dieses Phänomen nur noch in abgeschwächter Form oder gar nicht mehr auf. Lernt man beim Neural Gas mit funktionalen Prototypen ausschließlich die Gewichte der Basisfunktionen, so ist darauf zu achten, dass die Anzahl der Basisfunktionen hinreichend groß gewählt ist und dass die Stützstellen intelligent gemäß dem Datensatz gesetzt sind. Für Bereiche in den Daten mit

vielen lokalen Optima ist es beispielsweise sinnvoll mehrere Gaußfunktionen in diesem Bereich zu initialisieren.

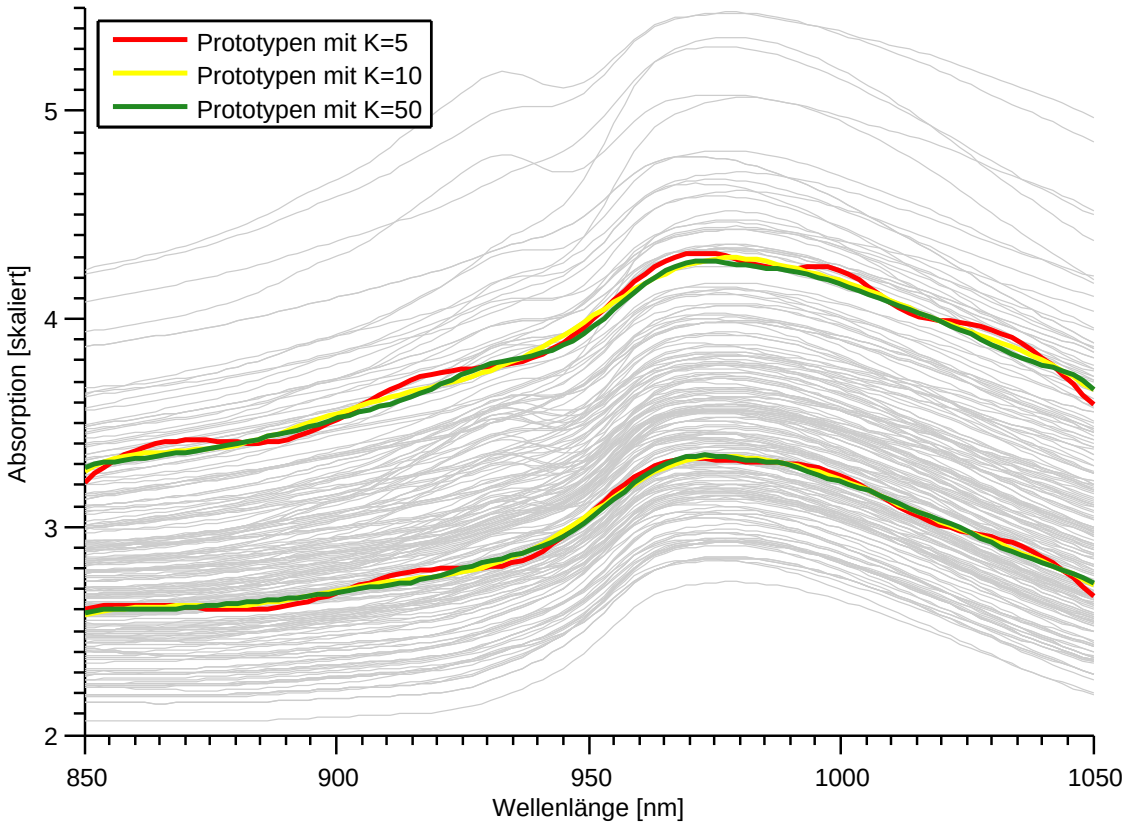


Abbildung 5.7: Der Einfluss der Anzahl K der Gaußfunktionen bei der Version des Neural Gas mit funktionalen Prototypen (Alg. 5)
Die in gleicher Farbe dargestellten Prototypen sind das Ergebnis eines Durchlaufes des Algorithmus 5, wobei K entsprechend gewählt wurde.

In Abbildung 5.7 sind Prototypen dargestellt, bei denen sowohl die Gewichte als auch die Stützstellen gelernt sind. Die Anzahl der Basisfunktionen hat optisch keinen markanten Einfluss auf die Gestalt der Prototypen. Der Grund hierfür ist, dass der Algorithmus nun die Möglichkeit hat, die Stützstellen zu variieren. Somit ist er in der Lage auch mit wenigen Basisfunktionen die Struktur der Daten zu repräsentieren.

Zur Validierung des Einflusses von K sind außerdem der XieBeni- (5.1) und der SV-Index (5.2) zur Anwendung gekommen. Die tabellarische Auflistung der Index-Werte ist im Anhang in Tabelle B.1 und Tabelle B.2 zu finden. Aus diesen Tabellen ergibt sich, dass für den Tecator Datensatz $K = 15$ eine gute Wahl ist.

Wichtige Einflussgrößen im Neural Gas mit funktionalen Prototypen sind weiterhin die Lernraten ε_β und ε_θ. Lernt man beim Tecator Datensatz nur die Gewichte β_{i_j}, so ist die Wahl der Lernrate ε_β nicht von entscheidender Bedeutung. Werte im Intervall $(0, 1]$ liefern annähernd gleiche Index Werte (Tab. B.3). Anders verhält es sich, wenn die Stützstellen

θ_{i_j} ebenfalls gelernt werden. Die Lernrate für die Stützstellen ε_θ sollte kleiner als ε_β sein. Ein gutes Ergebnis wurde mit $10 \cdot \varepsilon_\theta \sim \varepsilon_\beta$ erreicht (Tab. B.5).

Beispielhaft sind für ein Ergebnis die Cluster bzgl. der Prototypen in Abbildung 5.8 dargestellt.

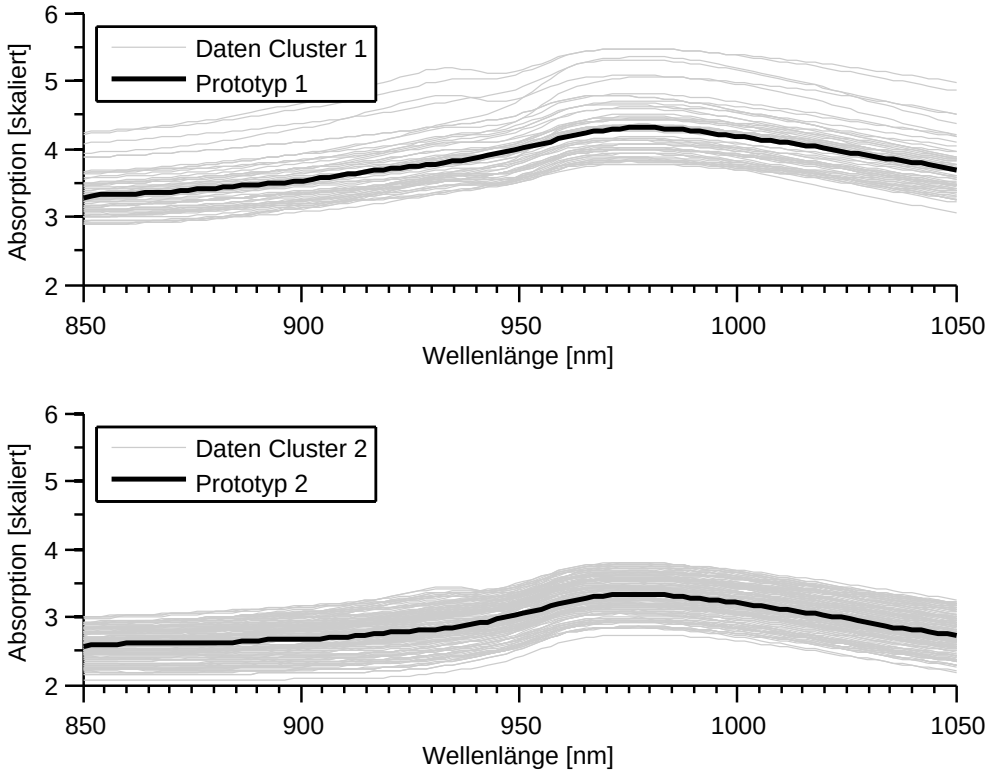


Abbildung 5.8: Eine Clusteraufteilung des Tecator Datensatzes mit euklidischer Abstandsbestimmung (2.4)

Vergleicht man die erhaltenen Ergebnisse dieses Abschnittes mit Ergebnissen des originalen Neural Gas (Alg. 1), so sind die Werte des XieBeni- (5.1) und des SV-Indexes (5.2) unwesentlich besser. Man erzielt beim Tecator Datensatz folglich keine Verbesserung der Ergebnisse, wenn man für die Prototypen eine Linearkombination von Gaußfunktionen verwendet. Unter Verwendung anderer Basisfunktionen oder anderer Gütekriterien könnte eine Verbesserung aber durchaus möglich sein.

5.3.2 Ergebnisse Neural Gas mit funktionalen Prototypen und der Sobolev Quasi-Metrik zum Abschnitt 3.2

Bei der Durchführung der Tests mit dem Tecator Datensatz unter Verwendung der Sobolev Quasi-Metrik (3.7) hat sich gezeigt, dass die funktionalen Prototypen nicht

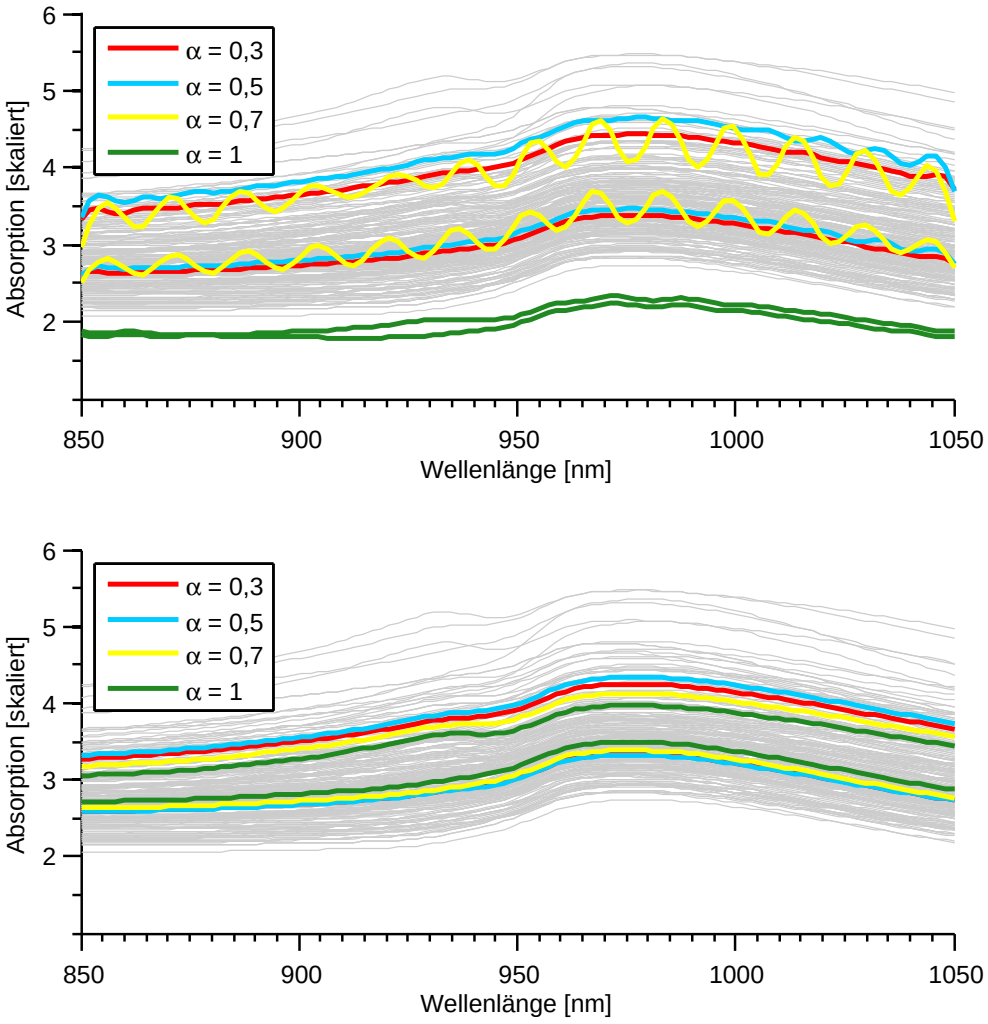


Abbildung 5.9: Der Einfluss des Parameters α der Sobolev Quasi-Metrik (3.7, 3.8) beim Neural Gas mit funktionalen Prototypen (Alg. 5) und dem originalem Neural Gas (siehe Abs. 2.2)

immer den charakteristischen Verlauf der Daten widerspiegeln (Abb. 5.9 oberer Teil). Im oberen Teil der Abbildung 5.9 sind die funktionalen Prototypen abgebildet und im unteren Teil sind die Prototypen des originalen Neural Gas zum Vergleich dargestellt.

Für die Berechnung der ersten Ableitung der Daten mittels eines Differenzenquotienten, wurden diese zunächst geglättet. Je stärker die erste Ableitung der Daten und der Prototypen im Neural Gas mit funktionalen Prototypen in die Berechnung des Abstandes gemäß der Sobolev Quasi-Metrik eingeht, desto untypischer sehen die Prototypen in der euklidischen Metrik aus. Jedoch konnte durch die Benutzung der Sobolev Quasi-Metrik die im Anstieg der Daten verborgene Information genutzt werden, denn die Werte der Validierungsmaße XieBeni- und SV-Index (siehe Tab. 5.1) haben sich stark verbessert. Im Vergleich zu den erzielten Index-Werten unter Verwendung der euklidischen Abstandsbe-

stimmung (2.4) ist eine deutliche Verbesserung zu erkennen. Lediglich für $\alpha = 1$, d. h. nur die Information der ersten Ableitung der Daten und der Prototypen wird genutzt, ist die Bewertung der Clusterlösung sehr schlecht. Die beste Lösung wurde für $\alpha = 0,1$ und $\alpha = 0,5$ erhalten. Für den Tecator Datensatz empfiehlt sich demzufolge, die Sobolev Quasi-Metrik statt der euklidischen Abstandsbestimmung zu verwenden. Beispielhaft

	NG funktionale Prototypen		originaler NG	
α	XieBeni	SV	XieBeni	SV
0.1	0,0097	**0,6539**	0,0091	**0,6887**
0,3	0,0112	0,6269	**0,0065**	0,4945
0,5	**0,0033**	0,557	0,0146	0,4214
0,7	0,0536	0,1201	0,0414	0,1469
0,9	0,3500	0,0137	0,1106	0,0264
1	1520,6	0,000003	66	0,00004

Tabelle 5.1: Einfluss von α bei der Sobolev Quasi-Metrik beim Tecator Datensatz
Parameter: $N = 2$, $Trainingsschritte = 5000$, $K = 25$, $\varepsilon_\beta = \varepsilon = 0,5$, $\varepsilon_\theta = 0,05$

für ein Ergebnis ist in Abbildung 5.10 die Aufteilung der Daten bzgl. der ermittelten Prototypen in Cluster graphisch dargestellt. Als Eingabeparameter ist $\alpha = 0,5$ gewählt.

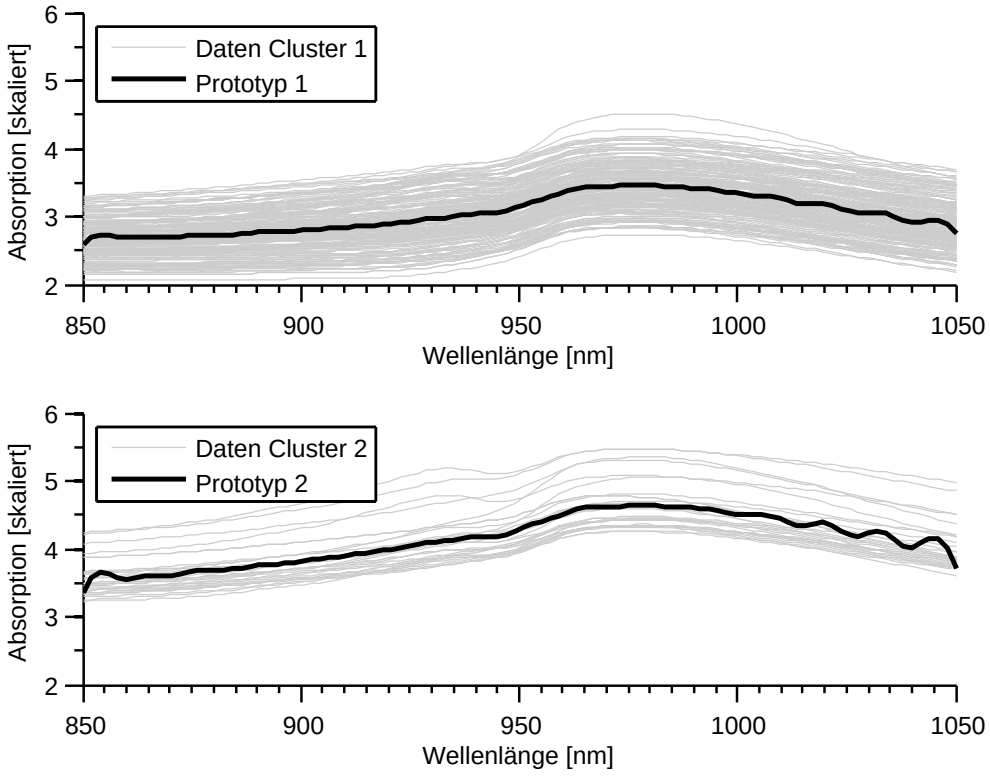


Abbildung 5.10: Eine Clusteraufteilung des Tecator Datensatzes mit $\alpha = 0,5$

Um die erhaltenen Ergebnisse dieses Abschnittes mit denen des originalen Neural Gas (Alg. 1) vergleichen zu können, wird in diesem statt der euklidischen Metrik die Sobolev Quasi-Metrik verwendet. Die berechneten Ergebnisse sind in Tabelle 5.1 aufgelistet.

5.3.3 Ergebnisse des verrauschten Tecator Datensatz

Tests, die mit dem verrauschten Tecator Datensatz (Abb. 5.2) durchgeführt wurden, haben gezeigt, dass die verrauschten Daten sich genauso verhalten wie die nicht Verrauschten. Vergleicht man die Tabellen aus Anhang B.1 mit der Tabelle B.6, so sind die besten Werte des XieBeni- und SV-Index bis auf geringe Unterschiede die gleichen. Dass die Daten verrauscht sind, ist für das Verfahren nicht signifikant. Auch bei dem verrauschten Datensatz werden unter Verwendung der Sobolev Quasi-Metrik deutlich bessere Index Werte erzielt (vgl. Tab. 5.2). Jedoch bei $\alpha = 1$ werden die Index Werte sehr schlecht, analog zu den nicht verrauschten Daten. Dass die Ergebnisse des verrauschten Datensatzes und der nicht verrauschten Daten so nah beieinander liegen, liegt an den folgenden Punkten:

- Zum einen werden die Daten zum Bilden der ersten Ableitung geglättet, d. h. die Zacken innerhalb der Daten werden runder und kommen so den ursprünglichen Daten näher.
- Des Weiteren sind die Prototypen als Linearkombination von Gaußfunktionen nicht in der Lage die vielen Zacken der Daten abzubilden. Sie spiegeln die Daten ebenfalls geglättet wieder.

α	XieBeni	SV
0.1	**0,0094**	**0,6664**
0,3	0,0108	0,6456
0,5	0,0133	0,5494
0,7	0,0153	0,1898
0,9	0,2741	0,0163
1	1200	0,0000038

Tabelle 5.2: Einfluss von α bei der Sobolev Quasi-Metrik beim verrauschten Tecator Datensatz
Parameter: $N = 2$, $Trainingsschritte = 5000$, $K = 25$, $\varepsilon_\beta = 0,5$, $\varepsilon_\theta = 0,05$

5.3.4 Ergebnisse Pulsing Neural Gas

Für den in Abschnitt 4.1 beschriebenen Pulsing Neural Gas werden an dieser Stelle die Ergebnisse vorgestellt. Verglichen werden die Ergebnisse des Neural Gas (Alg. 1) und des Pulsing Neural Gas (Alg. 7). Der Einfluss des Zufalls wurde beim Auswerten der Ergebnisse ausgeschaltet. Um die erhaltenen Ergebnisse zu vergleichen, wurde der Wert der Energiefunktion E_{NG} (2.6) herangezogen.

Bei den Tests mit dem Tecator Datensatz (Abs. 5.1.1) konnte keine merkliche Verbesserung der Energiefunktion E_{NG} (2.6) erreicht werden. Das könnte daran liegen, dass der Tecator Datensatz bereits durch den Neural Gas gut geclustert wird bzw. weil der Tecator Datensatz ein einfacher Datensatz ist.

Für weitere Untersuchungen des Pulsing Neural Gas wird der Schachbrett Datensatz (Abs. 5.1.2) verwendet, wobei die Prototypen in der Mitte des Datensatzes initialisiert sind. Diese Initialisierung ist die Schlechteste, die bei zufälliger Initialisierung auftreten kann und simuliert somit den *worst case*. Bei der Anwendung des Pulsing Neural Gas auf den Schachbrett Datensatz kann eine Verbesserung des Zielfunktionswertes im Vergleich zum Neural Gas erreicht werden. Exemplarisch ist in Abbildung 5.11 der Verlauf des Zielfunktionswertes dargestellt. Man erkennt in Abbildung 5.11, dass die Zielfunktionswerte sich nur geringfügig unterscheiden und der negative Lernschritt sich im Algorithmus nur wenig auswirkt, ganz so wie es beabsichtigt ist. Die Lagen der zugehörigen Prototypen sind in Abbildung 5.12 visualisiert. Hierbei stellt man fest, dass sowohl Prototypen des Neural Gas als auch Prototypen des Pulsing Neural Gas nicht in Clusterzentren liegen. Allerdings ist die Verteilung der Prototypen beim Pulsing Neural Gas besser, denn der Zielfunktionswert E_{NG} des Pulsing Neural Gas nimmt einen bessern Wert als beim Neural Gas an. Durch hinreichend langes Lernen haben die Prototypen die Möglichkeit, in die noch fehlenden Clusterzentren zu gelangen. An dieser Stelle sei noch erwähnt, dass datenuntypische Prototypen ein Indikator für eine zu große Lernrate ε sind.

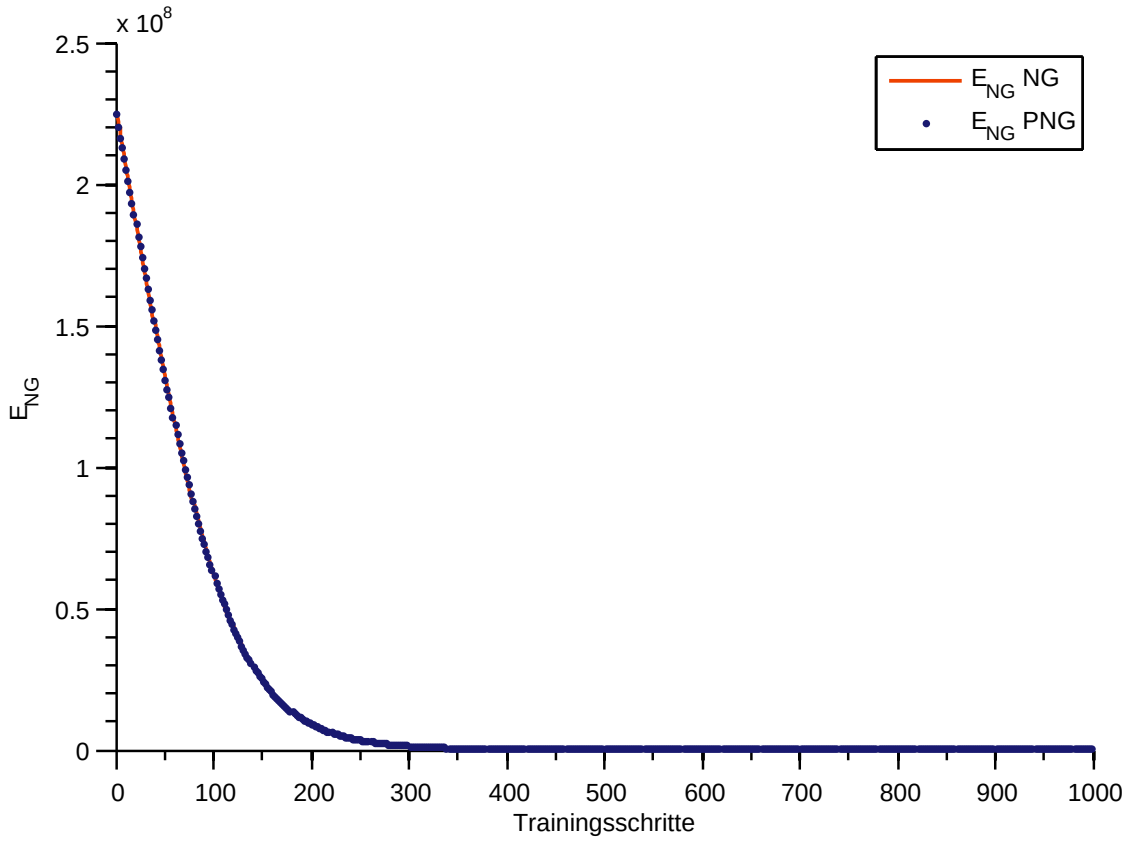


Abbildung 5.11: Exemplarischer Verlauf der Zielfunktionswerte des Neural Gas und des Pulsing Neural Gas für den Schachbrett Datensatz

5.3.5 Ergebnisse Pulsing Neural Gas Batch

Der Pulsing Neural Gas Batch (Alg. 8) wurde anhand des Tecator Datensatz 5.1 getestet und mit dem Neural Gas Batch (Alg. 2) verglichen. Hierfür wurden die Prototypen gleich initialisiert und alle Parameter, die in beiden Algorithmen vorkommen, identisch gewählt. Die Ergebnisse sind in Tabelle B.7 aufgelistet. Für den negativen Lernschritt wurde der *reduced information learning*-Schritt (Alg. 10) mit der Sieger-Rangfolge verwendet, da sich bei der Umsetzung des negativen Lernschrittes (Alg. 9) zeigte, dass dieser sich negativ auf das Ergebnis auswirkt. Der negative Lernschritt (Alg. 9) wirkt sich für die hier getesteten Datensätze zu stark auf die Prototypen aus und führt zu schlechteren Ergebnissen im Vergleich zum Neural Gas Batch und Pulsing Neural Gas Batch mit dem *reduced information learning*-Schritt (Alg. 10). Als Gütekriterium dient der Zielfunktionswert (ZFW) der Energiefunktion E_{NG} (2.10).

Für weitere Untersuchungen wurde der Schachbrett Datensatz (Abs. 5.1.2) verwendet. Auch hier hat sich gezeigt, dass ein besserer Zielfunktionswert als mit dem originalen Neural Gas erreicht werden kann. Dies ist allerdings nicht immer der Fall, da es sich um einen randomisierten Algorithmus handelt und der Parameter p_0 (4.2), der bei der Berechnung der Wahrscheinlichkeit eine maßgebende Rolle spielt, je nach Wahl das

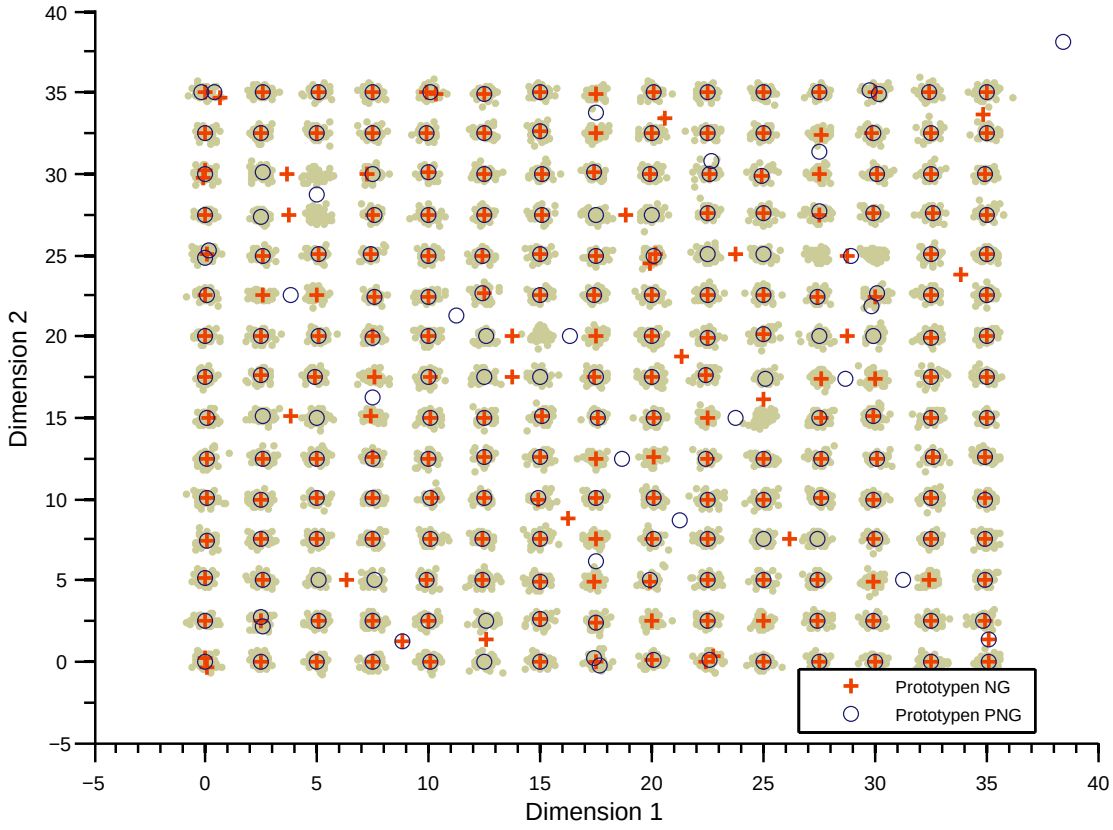


Abbildung 5.12: Beispielhafte Darstellung der Prototypen des Neural Gas und des Pulsing Neural Gas für den Schachbrett Datensatz

Ergebnis beeinflusst. Ersichtlich ist, dass mit dem Pulsing Neural Gas Batch bessere Zielfunktionswerte erreichbar sind. Dies kann man allerdings nicht immer garantieren und im Allgemeinen wird man bei der Anwendung des Pulsing Neural Gas Batch nicht genau wissen, ob das Ergebnis besser oder schlechter als das Ergebnis des Neural Gas Batch ist. Man kann den Neural Gas Batch zunächst anwenden und wenn mit diesem kein zufriedenstellendes Ergebnis erreicht wird, kann auf den Pulsing Neural Gas Batch zurückgegriffen werden. Dadurch hat man Ergebnisse zum Vergleich und kann die Güte der Ergebnisse des Pulsing Neural Gas Batch besser beurteilen. Der Verlauf der Zielfunktionswerte des Pulsing Neural Gas Batch und des Neural Gas Batch sind exemplarisch in Abbildung 5.13 dargestellt. Man erkennt deutlich, dass der Zielfunktionswert des Neural Gas Batch monoton fallend ist. Wie zu erwarten ist, schwankt der Zielfunktionswert des Pulsing Neural Gas Batch, doch sinkt dieser zum Ende hin immer stärker ab und liefert einen kleineren Zielfunktionswert als der Neural Gas Batch. Die zugehörigen Prototyen des Neural Gas Batch und Pulsing Neural Gas Batch zum Verlauf der Energiefunktion E_{NG} aus Abbildung 5.13 sind in Abbildung 5.14 visualisiert. Insgesamt sind 27 Prototypen des Neural Gas Batch nicht in Clusterzentren zu finden. Hingegen beim Pulsing Neural Gas Batch liegen nur 24 Prototypen nicht in Clusterzentren. Dieses Ergebnis lässt sich sicher durch längeres Lernen noch verbessern, doch es zeigt, dass der Pulsing Neural Gas Batch durchaus Potential hat sehr gute Ergebnisse zu ermitteln.

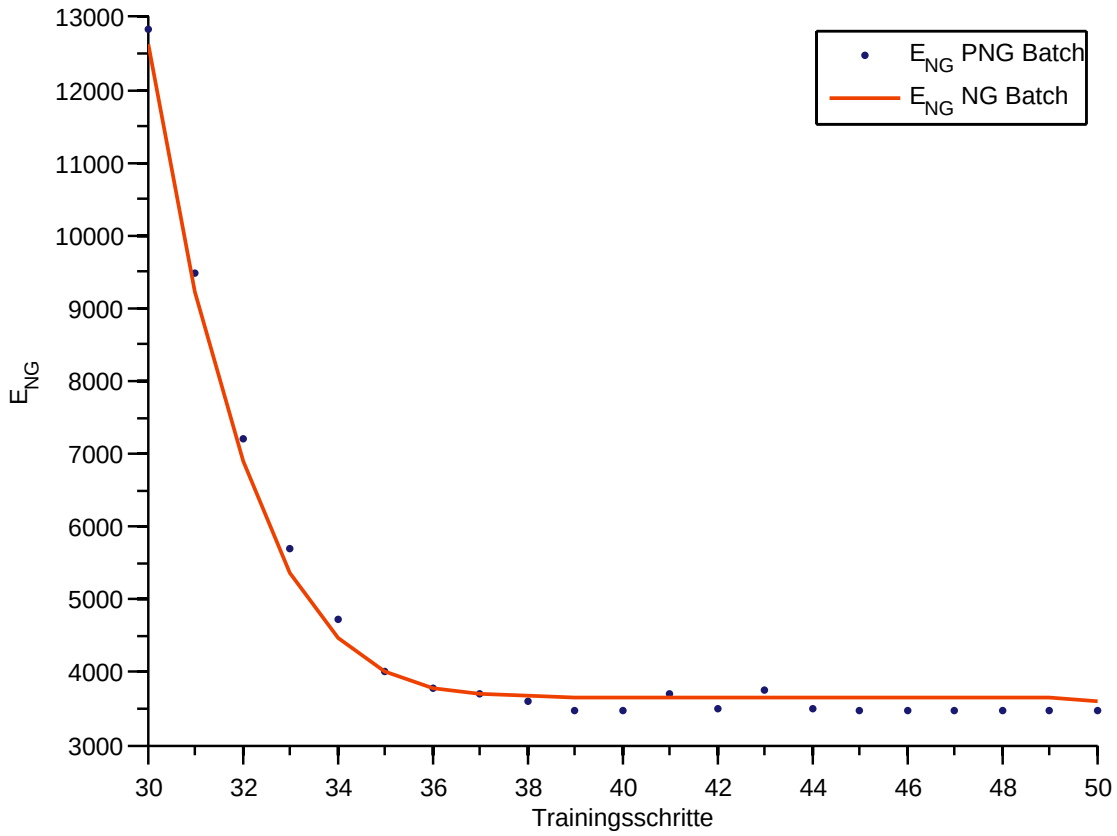


Abbildung 5.13: Exemplarischer Verlauf des Zielfunktionswertes des Pulsing Neural Gas Batch und des Neural Gas Batch für den Schachbrett Datensatz

5.3.6 Ergebnisse Pulsing Fuzzy Neural Gas

Bei Experimenten mit dem Pulsing Fuzzy Neural Gas (PFNG) zeigte sich, dass beim Tecator Datensatz keine signifikante Verbesserung im Ergebnis des Pulsing Fuzzy Neural Gas bzgl. des Resultats aus dem Fuzzy Neural Gas erreichen lässt. Allerdings kann man beim Schachbrett Datensatz mittels Pulsing Fuzzy Neural Gas ein besseres Ergebnis als beim Fuzzy Neural Gas erhalten. Ein Beispiel hierfür ist in den Abbildungen 5.15 und 5.16 gegeben.

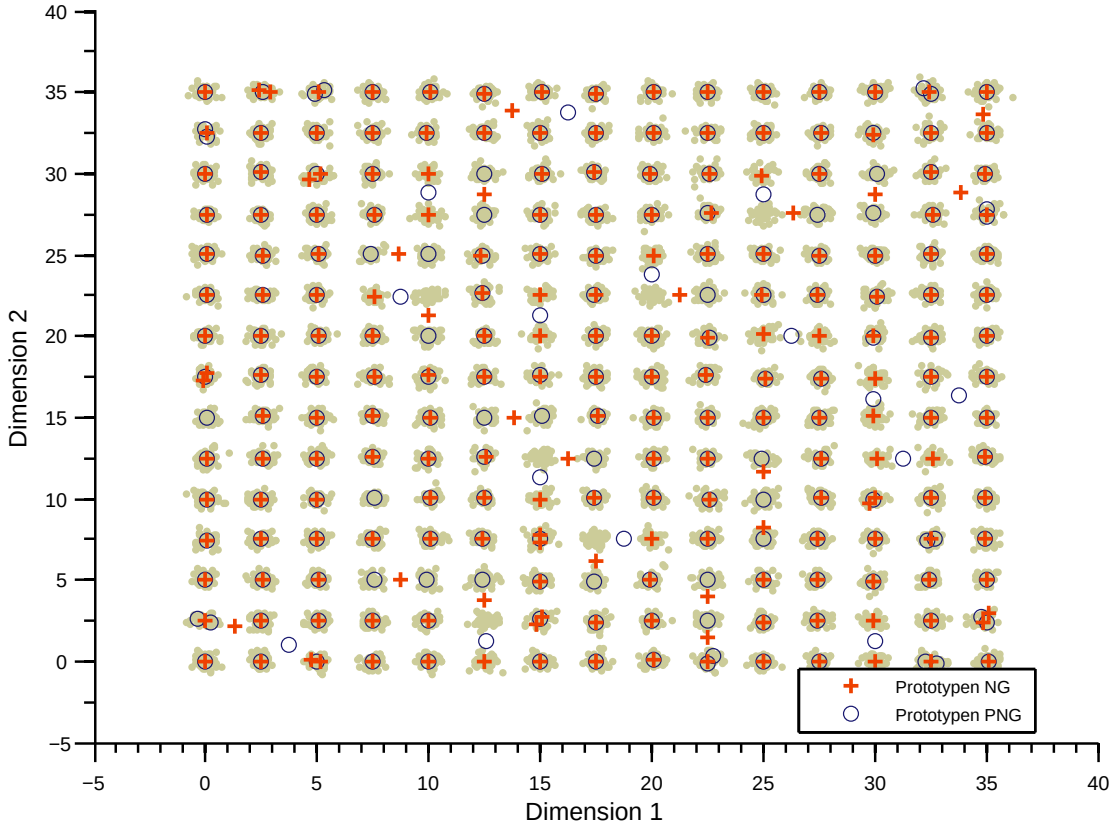


Abbildung 5.14: Beispielhafte Darstellung der Prototypen des Neural Gas Batch und des Pulsing Neural Gas Batch für den Schachbrett Datensatz

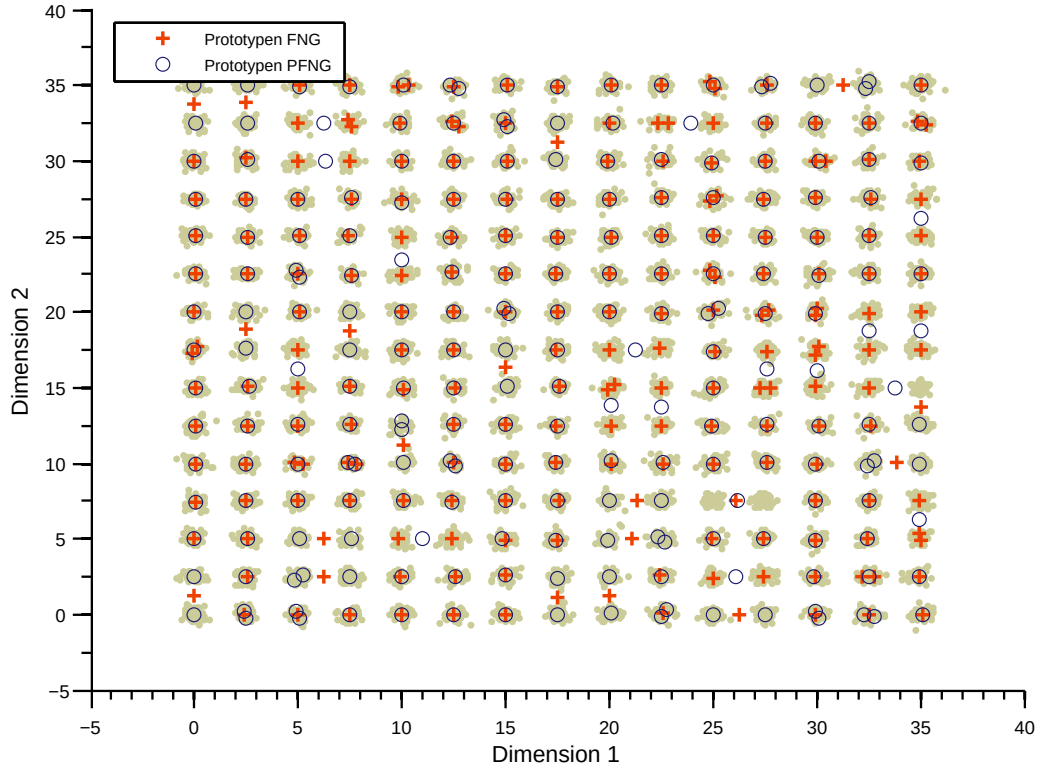


Abbildung 5.15: Beispielhafte Darstellung der Prototypen des Pulsing Fuzzy Neural Gas und des Fuzzy Neural Gas für den Schachbrett Datensatz

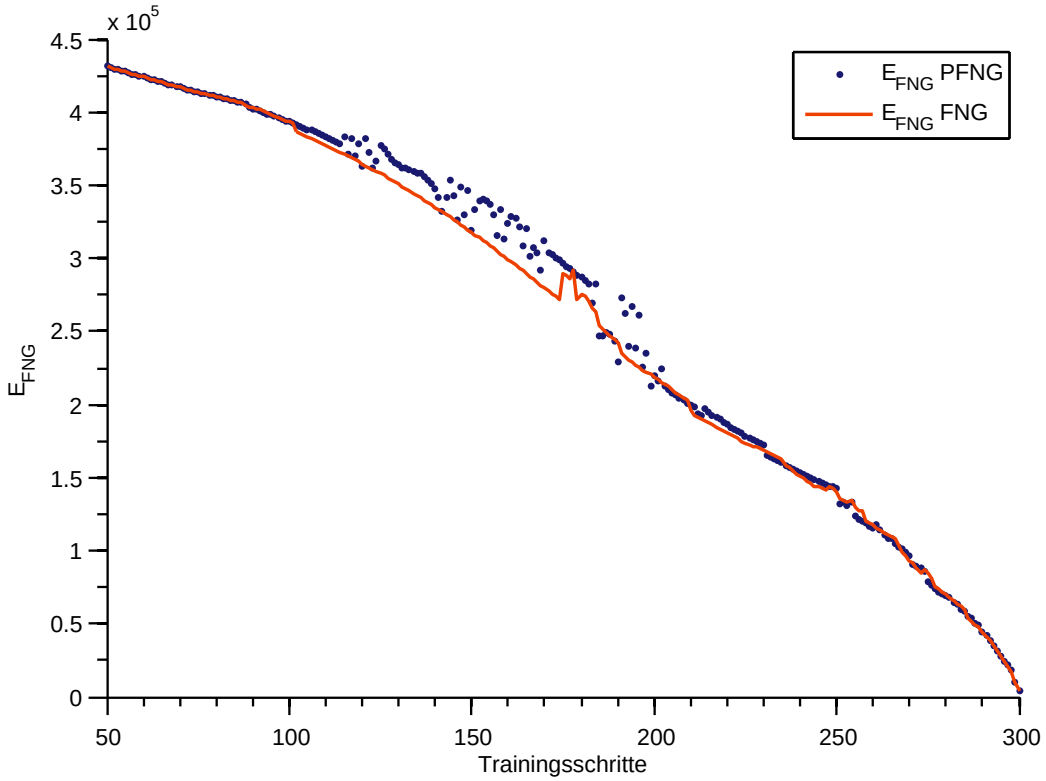


Abbildung 5.16: Verlauf der Zielfunktionswerte beim Pulsing Fuzzy Neural Gas und Fuzzy Neural Gas für den Schachbrett Datensatz

6 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit bestand in der Realisierung neuer Ideen in den bestehenden unüberwachten Vektorquantisierern Neural Gas und Fuzzy Neural Gas. Zwei neue Ansätze sind in diesem Zusammenhang realisiert.

Im Speziellen für funktionale Daten wurde der Neural Gas angepasst. Um die Information, dass die Daten einen funktionalen Zusammenhang beschreiben, nutzbringend in den Neural Gas zu integrieren, wurden Prototypen als Linearkombination von Basisfunktionen eingeführt. Als Basisfunktionen wurden Gaußfunktionen verwendet.
Vergleicht man die erhaltenen Testergebnisse von Neural Gas mit und ohne funktionale Prototypen, so sind diese vergleichbar. Bei dem Algorithmus mit funktionalen Prototypen hat man jedoch den Vorteil, dass man deutlich weniger Parameter (die der Basisfunktionen) im Gegensatz zum originalen Neural Gas (Dimension der Daten) lernen muss. Für den Tecator Datensatz kommt man beispielsweise mit 15 Gaußfunktionen, also 30 zu lernende Parameter, statt 100 zu lernenden Dimensionen für einen Prototypen aus. Durch die Reduktion von Parametern gelingt einem eine bessere Generalisierung. Das Gewicht und die Stützstelle jeder Gaußfunktion werden im Neural Gas mit funktionalen Prototypen gelernt, d. h. man lernt das Basissystem der Daten gemäß der gewählten Basisfunktionen. Bei den Tests hat sich gezeigt, dass es sich empfiehlt die Stützstellen langsamer als die Gewichte der Gaußfunktionen zu lernen.
Bislang wird in den Algorithmen i. A. der Abstand nach Euklid bestimmt. Da der Neural Gas nun auf funktionale Daten angepasst ist, wird nun zur Abstandsbestimmung eine funktionale Metrik verwendet. Zum Einsatz kommt eine Sobolev Quasi-Metrik, die die Information aus der ersten Ableitung der Daten und der Prototypen mit berücksichtigt. Hierbei wird die erste Ableitung der Daten numerisch mittels Differenzenquotient berechnet, sofern sie nicht bekannt ist. Für die Prototypen kann die erste Ableitung analytisch berechnet werden, da diese Linearkombinationen von den gewählten Basisfunktionen sind. Der modifizierte Neural Gas beinhaltet funktionale Prototypen und die Sobolev Quasi-Metrik zur Abstandsbestimmung. Unter Verwendung der Sobolev Quasi-Metrik konnten sowohl im Neural Gas als auch im modifizierten Neural Gas die Resultate für den Tecator Datensatz im Vergleich zu den Resultaten mit euklidischer Abstandsbestimmung verbessert werden. Vergleicht man die Ergebnisse untereinander, so sind diese näherungsweise gleich. Das Fazit für den modifizierten Neural Gas ist, dass er sehr leistungsstark ist. Für den hier zum Testen verwendeten Tecator Datensatz war die Wahl der Basisfunktion allerdings ungeeignet, da die Resultate im Vergleich zum Neural Gas nicht signifikant besser waren. Gezeigt wurde, dass der modifizierte Neural Gas mindestens so leistungsstark wie der ursprüngliche Neural Gas ist und noch ausbaufähiges Potential besitzt.

Im Zusammenhang mit dem modifizierten Neural Gas kann in Zukunft über die Umsetzung anderer Basisfunktionen (Lorentzfunktion, Sigmoidfunktion,...) oder anderer funktionaler Metriken nachgedacht werden. Außerdem bleibt offen, ob die funktionalen Prototypen in der Batch Version des Neural Gas oder im Fuzzy Neural Gas umsetzbar sind.

Der zweite Ansatz zur Verbesserung einiger bestehender Algorithmen der unüberwachten Vektorquantisierung war die Kombination mit der Kernidee von Simulated Annealing. In der Praxis kann man oftmals nur lokal gute Ergebnisse erreichen, insbesondere bei den Batch Algorithmen. Durch die Kombination mit Simulated Annealing, einer Metaheuristik der Optimierung, erhofft man sich bessere Resultate, nicht nur lokal sondern global. In der vorliegenden Arbeit wurde Simulated Annealing im Neural Gas, Neural Gas Batch und im Fuzzy Neural Gas umgesetzt. Diese neue Algorithmen Gattung wurde mit Pulsierender Vektorquantisierung bezeichnet.
Für die hier getesteten Datensätze (Tecator und Schachbrettdatensatz) konnten bessere Resultate erzielt werden. Allerdings spielt der Zufall in den Pulsing Algorithmen eine wesentliche Rolle, denn dieser legt fest, wie oft ein negativer Lernschritt ausgeführt wird. Außerdem ist es möglich, dass wesentlich schlechtere Ergebnisse mit den Pulsing Algorithmen zustande kommen als bei den Standard-Varianten. Bei der Anwendung in der Praxis sollte man zunächst die Standard-Algorithmen anwenden, um bereits hinreichend gute Ergebnisse zu erhalten bzw. um eine Referenzlösung zur Verfügung zu haben.

In weiterführenden Arbeiten zu diesem Thema, kann man prüfen, ob die Idee von Simulated Annealing in anderen Verfahren der Vektorquantisierung eine Verbesserung bewirkt. Untersuchen kann man auch, welchen Effekt das Simulated Annealing im Zusammenhang mit dem Neural Gas mit funktionalen Prototypen bewirkt.

Anhang A: Herleitungen

A.1 Lineare Unabhängigkeit der Gaußfunktionen

Sind die Gaußfunktionen linear unabhängig, ist zu zeigen, dass für

$$\sum_{i=1}^{K} \alpha_i \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_i} \cdot e^{-\frac{1}{2}\left(\frac{t-\theta_i}{\sigma_i}\right)^2} = 0 \tag{A.1}$$

nur die triviale Lösung $\alpha_i = 0, \forall i$ existiert. In Gleichung (A.1) erkennt man, dass die Gaußfunktionen stets verschiedene positive Werte annehmen und nicht Null werden. Demzufolge ist die einzige Lösung der Gleichung (A.1) $\alpha_i = 0, \forall i$ und die Gaußfunktionen sind somit linear unabhängig.

A.2 Herleitung der Adaptionsregel der Gewichte

Für endliche Datensätze ist die Energiefunktion des Neural Gas (2.6) gegeben durch:

$$E_{NG} = \frac{1}{2} \cdot \sum_{j=1}^{N} \sum_{r=1}^{M} h_\lambda(\mathbf{v}_r, \mathbf{w}_j) \cdot d_E^2(\mathbf{v}_r, \mathbf{w}_j).$$

Sei $d_E^2 = d_E^2(\mathbf{v}_r, \mathbf{w}_j)$ (2.4) eine verkürzende Schreibweise und die Prototypen $\mathbf{w}_j$ seien eine Linearkombination von Gaußfunktionen (3.2):

$$w_j(t) = \sum_{i=1}^{K} \beta_{i_j} \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_{i_j}} e^{-\frac{1}{2}\left(\frac{t-\theta_{i_j}}{\sigma_{i_j}}\right)^2} \quad j = 1, \ldots, N, \;\; t = 1, \ldots, D.$$

Bildet man die partielle Ableitung von E_{NG} nach β_{k_j}, so kommt die Kettenregel zum Einsatz

$$\frac{\partial E_{NG}}{\partial \beta_{k_j}} = \frac{\partial E_{NG}}{\partial d_E^2} \cdot \frac{\partial d_E^2}{\partial \beta_{k_j}}. \tag{A.2}$$

Man erhält durch Differentiation

$$\frac{\partial E_{NG}}{\partial d_E^2} = h_\lambda(\mathbf{v}, \mathbf{w}_j) \tag{A.3}$$

und

$$\begin{aligned}\frac{\partial d_E^2}{\partial \beta_{k_j}} &= 2 \cdot \sum_{t=1}^{D} (v(t) - \sum_{i=1}^{K} \beta_{i_j} \cdot \mathscr{K}_{i_j}(t)) \cdot (-\mathscr{K}_{k_j}(t)) \quad \forall k, j \\ &= -2 \cdot \sum_{t=1}^{D} (v(t) - \sum_{i=1}^{K} \beta_{i_j} \cdot \mathscr{K}_{i_j}(t)) \cdot \mathscr{K}_{k_j}(t) \quad \forall k, j \\ &= -2 \cdot \sum_{t=1}^{D} (v(t) - w_j(t)) \cdot \mathscr{K}_{k_j}(t) \quad \forall k, j.\end{aligned} \tag{A.4}$$

A.3 Herleitung der Adaptionsregel der Stützstellen

Bildet man die partielle Ableitung von E_{NG} nach θ_{k_j}, so ergibt sich laut Kettenregel

$$\frac{\partial E_{NG}}{\partial \theta_{k_j}} = \frac{\partial E_{NG}}{\partial d_E^2} \cdot \frac{\partial d_E^2}{\partial w_j} \cdot \frac{\partial w_j}{\theta_{k_j}}. \tag{A.5}$$

$\frac{\partial E_{NG}}{\partial d_E^2}$ ist bereits bekannt (vgl. A.3) und für die beiden anderen Faktoren erhält man

$$\frac{\partial d_E^2}{\partial w_j} = 2 \cdot (v(t) - w_j(t)) \cdot (-1) \tag{A.6}$$

und

$$\begin{aligned}\frac{\partial \mathbf{w}_j}{\partial \theta_{k_j}} &= \beta_{k_j} \cdot \frac{1}{\sigma_{k_j} \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \cdot (-1) \cdot \left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right) \cdot \left(-\frac{1}{\sigma_{k_j}}\right) \quad \forall k, j \\ &= \beta_{k_j} \cdot \frac{t-\theta_{k_j}}{\sigma_{k_j}^3 \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \quad \forall k, j.\end{aligned} \tag{A.7}$$

A.4 Herleitung der Adaptionsregel der Gewichte mit der Sobolev Quasi-Metrik

Für endliche Datensätze ist die Energiefunktion des Neural Gas mit der Sobolev Quasi-Metrik (3.9) gegeben durch:

$$E_{NG_S} = \frac{1}{2} \cdot \sum_{j=1}^{N} \sum_{r=1}^{M} h_\lambda(\mathbf{v}_r, \mathbf{w}_j) \cdot d_S^2(\mathbf{v}_r, \mathbf{w}_j).$$

Sei $d_S^2 = d_S^2(\mathbf{v}_r, \mathbf{w}_j)$ (3.7) eine verkürzende Schreibweise. Dann erhält man für den Gradientenabstieg der β_{k_j} auf der Energiefunktion E_{NG_S} (3.9):

$$\frac{\partial E_{NG_S}}{\partial \beta_{k_j}} = \frac{\partial E_{NG_S}}{\partial d_S^2} \cdot \frac{\partial d_S^2}{\partial \beta_{k_j}} \tag{A.8}$$

Für den ersten Faktor in (A.8) ergibt sich

$$\frac{\partial E_{NG_S}}{\partial d_S^2} = h_\lambda(\mathbf{v}, \mathbf{w}_j) \tag{A.9}$$

und für den zweiten

$$\begin{aligned}
\frac{\partial d_S^2}{\partial \beta_{k_j}} &= (1-\alpha)\cdot 2\cdot \sum_{t=1}^{D}(v(t)-w_j(t))\cdot\left(\frac{-1}{\sqrt{2\pi}\sigma_{k_j}} e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\right) \\
&\quad +\alpha\cdot 2\cdot\sum_{t=1}^{D}(\dot{v}(t)-\dot{w}_j(t))\cdot\frac{t-\theta_{k_j}}{\sqrt{2\pi}\cdot\sigma_{k_j}^3}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \\
&= \left(\tfrac{2}{\pi}\right)^{\frac{1}{2}}\cdot\left[\frac{\alpha-1}{\sigma_{k_j}}\cdot\sum_{t=1}^{D}\left(v(t)-w_j(t)\right)\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} + \frac{\alpha\cdot(t-\theta_{k_j})}{\sigma_{k_j}^3}\cdot\sum_{t=1}^{D}\left(\dot{v}(t)-\dot{w}_j(t)\right)\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\right].
\end{aligned} \tag{A.10}$$

A.5 Herleitung der Adaptionsregel der Stützstellen mit der Sobolev Quasi-Metrik

Analoges Vorgehen führt für die θ_{k_j} auf folgenden Gradientenabstieg:

$$\frac{\partial E_{NG_S}}{\partial \theta_{k_j}} = \frac{\partial E_{NG_S}}{\partial d_S^2}\cdot\frac{\partial d_S^2}{\partial \theta_{k_j}} \tag{A.11}$$

mit

$$\begin{aligned}
\frac{\partial d_S^2}{\partial \theta_{k_j}} &= (1-\alpha)\cdot 2\sum_{t=1}^{D}\left(v(t)-w_j(t)\right)\cdot\frac{-\beta_{k_j}}{\sqrt{2\pi}\sigma_{k_j}}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\cdot\frac{-1}{\sigma_{k_j}^2}\cdot(t-\theta_{k_j})\cdot(-1) \\
&\quad +\alpha\cdot 2\sum_{t=1}^{D}(\dot{v}(t)-\dot{w}_j(t))\cdot\frac{-\beta_{k_j}}{\sqrt{2\pi}\sigma_{k_j}^3}\cdot\left[-e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} + (t-\theta_{k_j})\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\cdot\frac{1}{\sigma_{k_j}^2}\cdot(t-\theta_{k_j})\right] \\
&= (1-\alpha)\cdot\sum_{t=1}^{D}\left(v(t)-w_j(t)\right)\cdot\frac{-\beta_{k_j}(t-\theta_{k_j})}{\sqrt{2\pi}\sigma_{k_j}^3}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2} \\
&\quad +\alpha\cdot\sum_{t=1}^{D}(\dot{v}(t)-\dot{w}_j(t))\cdot\frac{-\beta_{k_j}}{\sqrt{2\pi}\sigma_{k_j}^3}\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\cdot\left[\frac{(t-\theta_{k_j})^2}{\sigma_{k_j}^2}-1\right] \\
&= -\frac{\beta_{k_j}}{\sqrt{2\pi}\sigma_{k_j}^3}\cdot\left[(1-\alpha)\cdot\sum_{t=1}^{D}\left(v(t)-w_j(t)\right)\cdot(t-\theta_{k_j})\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\right. \\
&\quad \left.+\alpha\sum_{t=1}^{D}(\dot{v}(t)-\dot{w}_j(t))\cdot\left(\frac{(t-\theta_{k_j})^2}{{\sigma_{k_j}}^2}-1\right)\cdot e^{-\frac{1}{2}\left(\frac{t-\theta_{k_j}}{\sigma_{k_j}}\right)^2}\right].
\end{aligned} \tag{A.12}$$

Anhang B: Ergebnis Tabellen

B.1 Anwendung des Neural Gas mit funktionalen Prototypen auf den Tecator Datensatz

In diesem Abschnitt sind die Ergebnisse des Tecator Datensatzes 5.1.1 tabellarisch aufgelistet.

- $N = 2$
- $Trainingsschritte = 5000$

Einfluss der Anzahl der Gaußfunktionen

- $\varepsilon_\beta = 0,5$
- $\varepsilon_\theta = 0$
- $\alpha = 0$

Anzahl der Gaußfunktionen K	XieBeni-Index	SV-Index
5	0,4192	**0,0096**
10	0,2805	0,0092
15	**0,2733**	0,0093
20	0,2769	0,0093
25	0,2774	0,0093
50	0,2884	0,0091

Tabelle B.1: Einfluss der K Gaußfunktionen bei Adaption der Gewichte β_{i_j}

- $\varepsilon_\beta = 0,5$
- $\varepsilon_\theta = 0,05$
- $\alpha = 0$

Anzahl der Gaußfunktionen K	XieBeni-Index	SV-Index
5	0,2778	0,0093
10	0,2778	0,0093
15	0,2771	0,0093
20	0,2774	0,0093
25	**0,2723**	**0,0094**
50	0,2802	0,0092

Tabelle B.2: Einfluss der K Gaußfunktionen bei Adaption der Gewichte β_{i_j} und der Stützstellen θ_{i_j}

Einfluss des Lernparameters ε_β

- $K = 15$
- $\varepsilon_\theta = 0$
- $\alpha = 0$

ε_β	XieBeni-Index	SV-Index
1	0,2739	**0,0094**
0,8	**0,2735**	**0,0094**
0,5	0,2773	0,0093
0,2	0,2769	0,0093
0,05	0,2787	0,0093
0,005	0,2786	0,0093

Tabelle B.3: Einfluss des Lernparameters ε_β ohne Lernen der Stützstellen θ_{i_j}

- $K = 15$
- $\varepsilon_\theta = 0,05$
- $\alpha = 0$

ε_β	XieBeni-Index	SV-Index
1	0,2776	**0,0093**
0,8	0,2827	0,0092
0,5	**0,2751**	**0,0093**
0,2	0,2758	**0,0093**
0,05	0,2759	**0,0093**
0,005	0,2781	**0,0093**

Tabelle B.4: Einfluss des Lernparameters ε_β mit Lernen der Stützstellen θ_{i_j}

Einfluss des Lernparameters ε_θ

- $K = 15$
- $\varepsilon_\beta = 0,8$
- $\alpha = 0$

ε_θ	XieBeni-Index	SV-Index
1	0,2777	**0,0093**
0,8	0,2782	**0,0093**
0,5	0,2754	**0,0093**
0,2	0,2766	**0,0093**
0,05	**0,2737**	**0,0093**
0,005	0,2762	**0,0093**

Tabelle B.5: Einfluss des Lernparameters ε_θ

B.2 Verrauschter Tecator Datensatz

In diesem Abschnitt sind die Ergebnisse des verrauschten Tecator Datensatzes (Abb. 5.2) tabellarisch aufgelistet.

- $N = 2$
- $Traininsschritte = 5000$

Einfluss der Anzahl der Gaußfunktionen

- $\varepsilon_\beta = 0,5$
- $\varepsilon_\theta = 0,05$
- $\alpha = 0$

Anzahl der Gaußfunktionen K	XieBeni-Index	SV-Index
5	0,2837	**0,0093**
10	0,2813	0,0092
15	0,2805	**0,0093**
20	**0,2788**	**0,0093**
25	0,2820	**0,0093**
50	0,2801	**0,0093**

Tabelle B.6: Einfluss der K Gaußfunktionen beim verrauschten Tecator Datensatz

B.3 Ergebnisse des Pulsing Neural Gas und Pulsing Neural Gas Batch

- Tecator Datensatz
- $Trainingsschritte = 3000$
- $p_0 = 0,2$

N	ZFW PNG Batch	ZFW NG Batch
2	1974	1974
5	453	460
10	146	147
15	99	99
20	74	78

Tabelle B.7: Gegenüberstellung der Batch Varianten von PNG und NG (ZFW $\hat{=}$ Zielfunktionswert der Energiefunktion E_{NG})

N	ZFW PNG	ZFW HPNG	ZFW NG
2	1975	1975	1975
5	459	459	459
10	152	154	152
15	100	104	100
20	72	76	71

Tabelle B.8: Gegenüberstellung der Batch Varianten von PNG, HPNG und NG (ZFW $\hat{=}$ Zielfunktionswert der Energiefunktion E_{NG})

Anhang C: Initialisierung der funktionalen Prototypen beim Neural Gas

In diesem Abschnitt wird detailliert auf die Initialisierung der funktionalen Prototypen in den Algorithmen 4 und 5 eingegangen. Folgende Möglichkeiten stehen dem Nutzer zu Initialisierung zur Verfügung:

1. äquidistante Stützstellen
2. zufällig gleichverteilte Stützstellen
3. eine Mischung aus den beiden ersten Varianten.

Bei der Variante 1 wird die Achse, auf der die Dimensionen abgetragen werden, in K gleichlange Intervalle eingeteilt. An die Stellen, an denen die Intervalle aneinander treffen, werden die Stützstellen θ_{i_j} für die Prototypen (Abb.C.1) gesetzt. Die Breiten σ_{i_j}

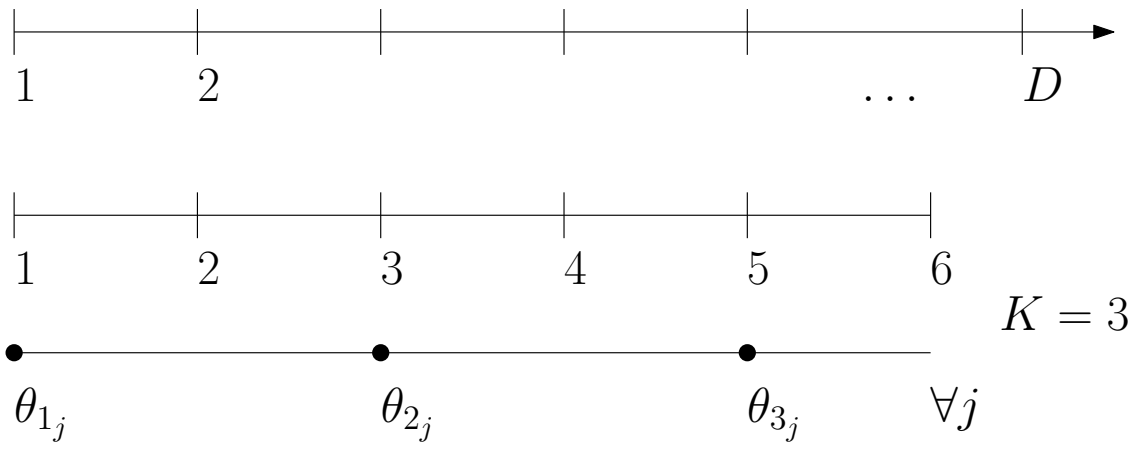


Abbildung C.1: Ein Beispiel für die Initialisierung äquidistanter Stützstellen für funktionale Prototypen (3.1)

bekommen bei dieser Initialisierung alle einen konstanten Wert zugewiesen, der zwei Drittel der Intervalllänge beträgt. Mit $1/K$ sind die Gewichte β_{i_j} initialisiert. Damit die Prototypen nicht alle identisch initialisiert werden, addiert man kleine Zufallszahlen auf die Gewichte. Als Beispiel für diese Initialisierung sind in Abbildung C.2 zwei Prototypen für den Tecator Datensatz abgebildet.
Bei der Umsetzung der Variante 1 tritt folgendes Problem auf, wenn man nur die Gewichte β_{i_j} adaptiert. Wie man in Abbildung C.2 erkennt, ist am rechten Rand die letzte Stützstelle bei ca. 1032 nm. Der Algorithmus versucht die Daten zu lernen und hat am rechten Rand nur die Ausläufe der Gaußfunktionen zum Überlagern zur Verfügung. Dementsprechend wird über diese letzte Gaußfunktion vor dem rechten Rand der meiste Beitrag zum Überlagern beigesteuert und das Gewicht β_{10_j} entsprechend hoch. Im Ergebnis (Abb. C.3) spiegelt sich das in dem lokalen Maximum der Prototypen am rechten Rand wider.

Das Problem mit den lokalen Maxima kann vermieden werden, indem man sicherstellt, dass eine Stützstelle stets am linken und am rechten Rand liegt (Abb. C.4), bzw. indem man die Stützstellen ebenfalls adaptiert.

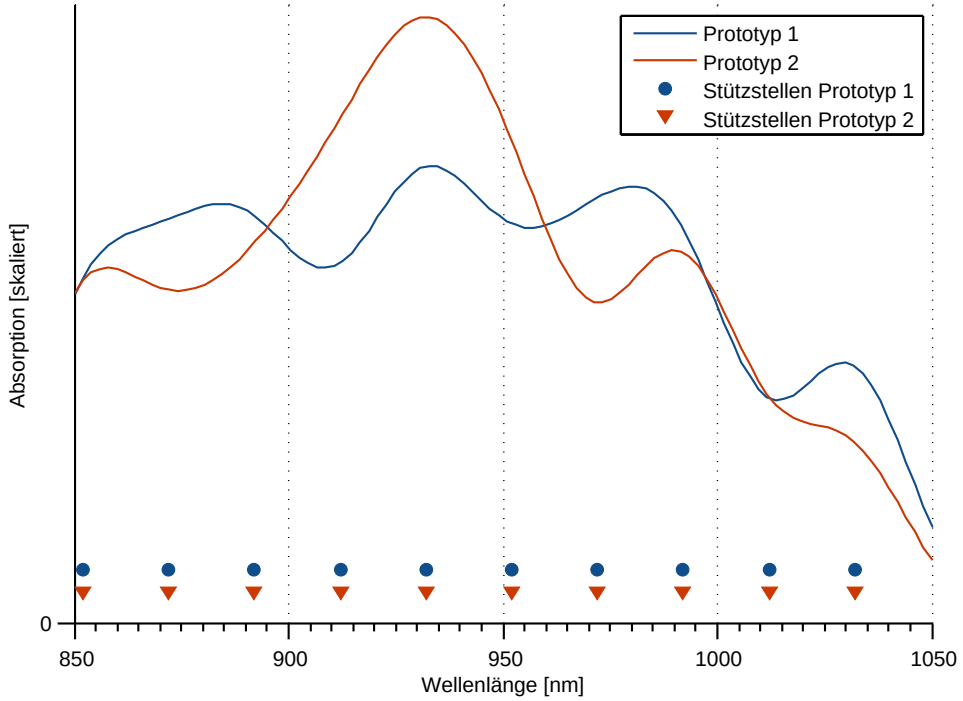


Abbildung C.2: Zwei initialisierte Prototypen für den Tecator Datensatz, zum besseren Verständnis sind die Stützstellen zusätzlich eingezeichnet

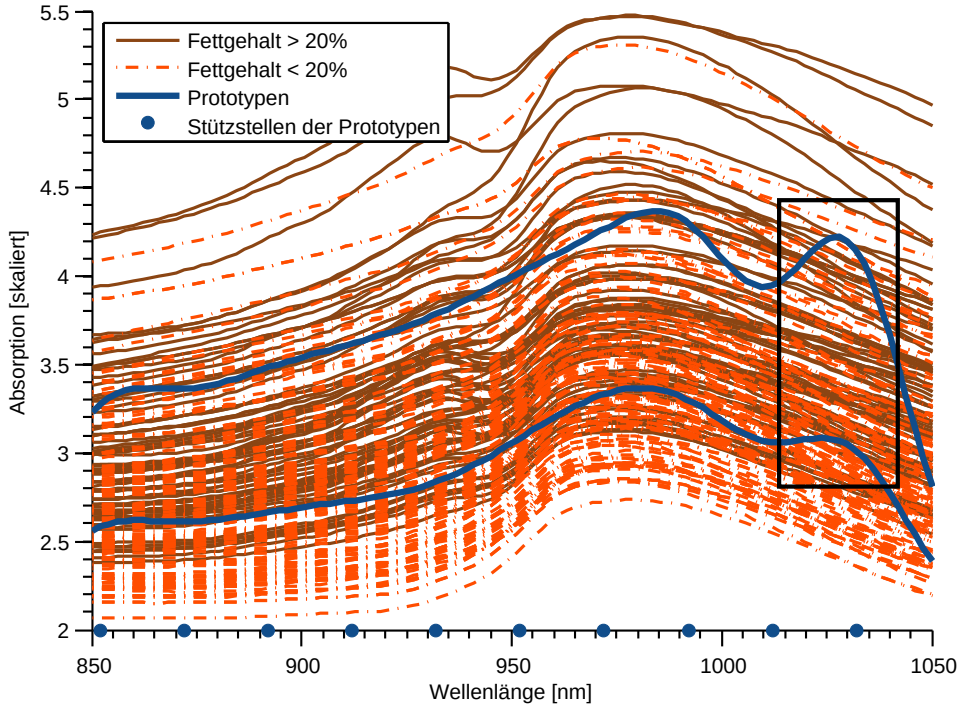


Abbildung C.3: Exemplarisches Ergebnis mit zwei Prototypen für Tecator Datensatz

Die zweite Variante unterscheidet sich zur ersten nur durch die Initialisierung der Stützstellen. Dies geschieht hier zufällig gleichverteilt. Solange man nur die Gewichte lernt, ist diese Art der Initialisierung unbrauchbar. Tests haben gezeigt, dass oft viele Stützstellen nah beieinander liegen bzw. große Intervalle der Dimensionsachse ohne Stützstelle sind. Dadurch können nur in seltenen Fällen datenrepräsentierende Prototypen entstehen. Aufgrund dieser Resultate und Überlegungen ist eine dritte Idee zu Initialisierung entstanden, bei der es sich um eine Mischung aus den beiden Varianten 1und 2 handelt. Für diese Initialisierung werden zwei Drittel der Stützstellen äquidistant verteilt und ein Drittel zufällig gleichverteilt (Abb. C.5). Wahlweise kann diese Aufteilung auch verändert werden.

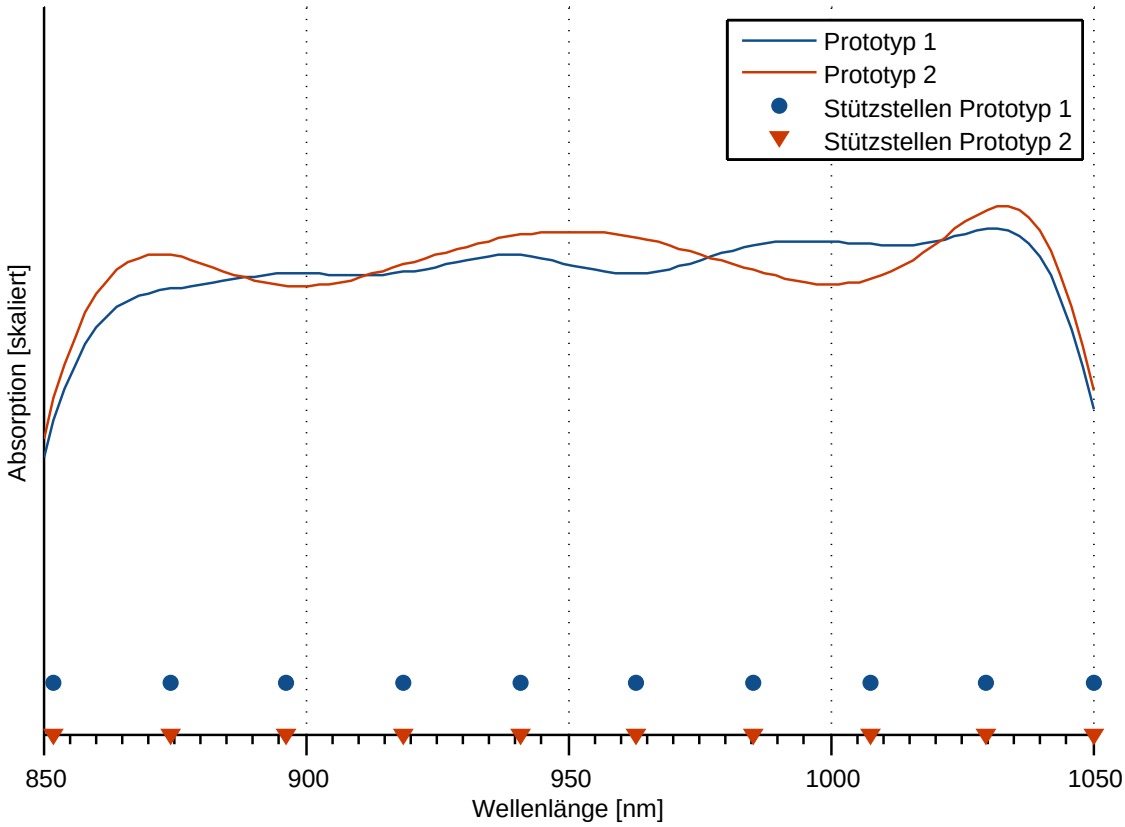


Abbildung C.4: Zwei initialisierte Prototypen für den Tecator Datensatz, eine Stützstelle liegt nun stets am linken und am rechten Rand

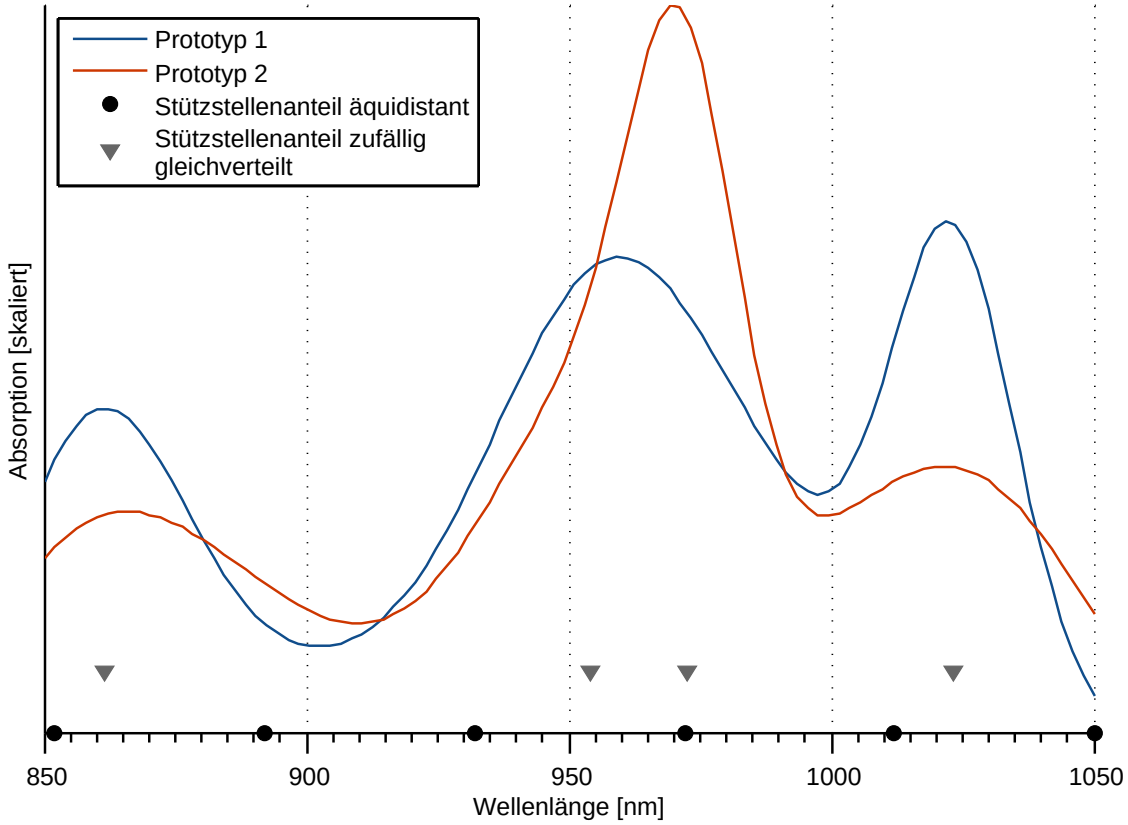


Abbildung C.5: Initialisierung zweier Prototypen nach Variante drei

Literaturverzeichnis

[Bez81] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[BH03] James C. Bezdek and Richard J. Hathaway. Convergence of alternating optimization. *Neural Parallel & Scientific Comp*, 11:351–368, 2003.

[CHHV06] Marie Cottrell, Barbara Hammer, Alexander Hasenfuss, and Thomas Villmann. Batch and median neural gas. *Neural Networks*, 19(6-7):762–771, 2006.

[Dun73] J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[ERSD77] K.-H. Elster, R. Reinhardt, M. Schäuble, and G. Donath. *Einführung in die nichtlineare Optimierung*. BSB B.G. Teubner Verlagsgesellschaft Leipzig, 1977.

[Har12] Christian Harth. Erweiterung von Generalized [Relevance|Matrix] Learning Vector Quantization zur Anwendung auf funktionale Daten, 2012.

[Has09] Alexander Hasenfuß. *Topographic mapping of dissimilarity datasets*. PhD thesis, Clausthal University of Technology, 2009. urn:nbn:de:gbv:104-1094728; http://d-nb.info/997448334.

[Hay09] S. Haykin. *Neural Networks and Learning Machines*. Pearson Prentice Hall, Boston, 3rd edition, 2009.

[KC78] H. Kushner and D. Clark. *Stochastic Appproximation Methods for Constrained and Unconstrained Systems*. Springer Verlag, 1978.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Koh82a] Teuvo Kohonen. Analysis of a simple self-organizing process. *Biol. Cyb.*, 44(2):135–140, 1982.

[Koh82b] Teuvo Kohonen. Clustering, taxonomy, and topological maps of patterns. In M. Lang, editor, *Proceedings of the Sixth International Conference on Pattern Recognition*, Silver Spring, MD, 1982. IEEE Computer Society Press.

[Koh82c] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Mar92] Thomas Martinetz. Selbstorganisierende neuronal Netzwerkmodelle zur Bewegungssteuerung, 1992. Dissertationen zur künstlichen Intelligenz.

[MBS93] Th. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, July 1993.

[PD05] E. Pekalska and R. P. W. Duin. *The Dissimilarity Representation for Pattern Recognition*. World Scientific Publishing Co. Pte. Ltd., 2005.

[RGF90] K. Rose, E. Gurewitz, and G. C. Fox. Statistical mechanics and phase transitions in clustering. Technical Report CFP-895, California Institute of Technology, 1990.

[Tho] Hans Henrik Thodberg. Tecator meat sample dataset, available on: http://lib.stat.cmu.edu/datasets/tecator.

[VGKL11] Thomas Villmann, Tina Geweniger, Marika Kästner, and Mandy Lange. Theory of fuzzy neural gas for unsupervised vector quantization. In Thomas Villmann and Frank-Michael Schleif, editors, *Machine Learning Reports 06/2011, MiWoCI 2011*, 2011.

[VHB09] Thomas Villmann, Barbara Hammer, and Michael Biehl. Some theoretical aspects of the neural gas vector quantizer. In Michael Biehl, Barbara Hammer, Michel Verleysen, and Thomas Villmann, editors, *Similarity-Based Clustering*, volume 5400 of *Lecture Notes in Computer Science*, pages 23–34. Springer, 2009.

[WM95] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[XB91] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(8):841–847, August 1991.

[ZZ11] Krista Rizman Zalik and Borut Zalik. Validity index for clusters of different sizes and densities. *Pattern Recognition Letters*, 32(2):221–234, 2011.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 26. August 2012